A Stochastic Approach for Selective Search Algorithms

Hans Gaiser
Delft University of Technology
Mekelweg 4, Delft, Netherlands
hansg91@gmail.com

Abstract

This paper introduces a new algorithm that generates candidate proposals for an object detection pipeline. We introduce Stochastic Selective Search (SSS), a segmentation based selective search method, which differs from previous work in two ways. First and most importantly, SSS is much faster than current state-of-the-art algorithms while maintaining comparable accuracy. This is a result of our efficient stochastic segment merging process. Other work requires the computation of features to determine the order in which segments are merged. We show that currently used features from other work does not improve the results of SSS significantly and are therefore omitted. This makes our algorithm nearly twice as fast as the fastest prior selective search algorithms. Secondly, due to the stochastic merging process of SSS, it is not critically affected when two wrong segments are merged during the merging process, which leads to object proposals of higher quality. We show that SSS outperforms existing deterministic selective search methods while generating the same amount of proposals in less time. Additionally, we demonstrate the performance of our SSS algorithm in a state-of-the-art object detection pipeline based on convolutional networks.

1. Introduction

Traditional object classifiers try to label an image as belonging to some class. To locate an object that is part of a larger image, object classifiers are often combined with object proposal algorithms. Object proposal algorithms create many subimages as proposals for possible object locations. An object classifier processes these subimages to determine the presence of an object in those subimages. The combination of proposing subimages as objects from an image and classifying these proposals is called object detection.

An example of an object proposal algorithm is the sliding window technique, which is used in many traditional object detection algorithms [6, 20, 21, 9]. The sliding window technique creates object proposals by sliding a window over the image, sampling the image at each step.

This object detection pipeline has been used for many years with great results, however it suffers from an inherently inefficient step: the large amount of object proposals generated by the sliding window technique. Objects are often of different shapes and sizes across images, as a result of which using a single size does not suffice. Commonly, multiple different windows are used to slide over the image using different sizes and aspect ratios. However, this still often leads to a few hundred thousand object proposals. Until recently, object classifiers dealt with the large number of proposals by using computationally efficient features[6, 20, 21, 17] or cascaded classification algorithms[20, 21]. However, the top results from the ILSVRC2014 competition have been from convolutional networks, which require a lot of computation per object proposal. It is therefore essential for the object proposal algorithm to propose as few locations as possible for classifiers such as convolutional networks[14, 4] to work in realtime.

Instead of the exhaustive search approach of a sliding window technique, selective search algorithms propose locations based on the contents of the image. Selective search algorithms[3, 19, 16] generally work much alike. The first step is to create a segmentation of the image. The generated segments naturally follow the contours of objects and are therefore well suited to delineate objects from one another. In the second step, the proposal algorithm proposes multiple subsets of segments as object proposals. This process produces significantly fewer proposals than the exhaustive sliding window technique, which allows the use of classifiers otherwise unfeasible.

Previous selective search algorithms[19, 3, 7] merge segments in a deterministic manner to create what is called a segmentation hierarchy. As a result of this hierarchy, a wrongful merge propagates through to the rest of the hierarchy and thus to the rest of the object proposals, which negatively affects the quality of the proposal algorithm.

In this paper, we introduce a stochastic selective search algorithm (SSS). Instead of creating a segmentation hierarchy, SSS selects a given amount of random subsets of neigh-

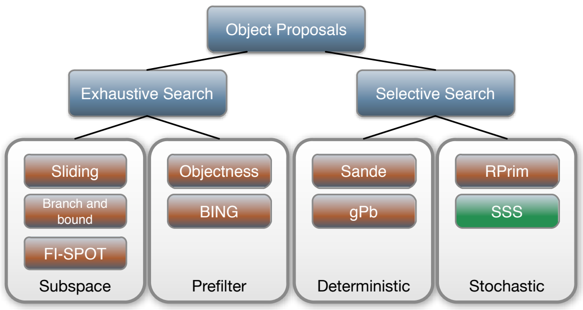


Figure 1: A visualization of the different types of categories for creating object proposals. Subspace[15, 22], prefilter[2, 5], deterministic[19, 3] and stochastic[16].

boring segments to use as proposals. Unlike the deterministic segmentation hierarchy approach of other algorithms[19, 3], our algorithm is not crucially affected when a wrong subset of segments is selected as object proposal. Since SSS does not compute any image features for the segments, the algorithm is much faster than other algorithms, while maintaining comparable accuracy.

The algorithm introduced here has the following advantages: it (i) easily recovers from false merges; (ii) generates any number of proposals, allowing "detection on a budget"; (iii) is very efficient and easily parallelizable.

This paper first discusses the related work in section 2, after which we introduce Stochastic Selective Search in section 3. Section 4 evaluates SSS and compares it to other state-of-the-art selective search algorithms and section 5 contains the final conclusions and future work.

2. Related Work

In addition to the already mentioned *exhaustive search* and *selective search* distinction, other types of object proposal algorithms exist. Some improvements over the sliding window technique exists, such as using the *appearance model* or using a measure of *objectness* for the object proposals. This section discusses the different types of object proposal algorithms. An overview of the categories discussed is shown in figure 1.

2.1. Exhaustive Search Algorithms

Objects can be of any size, aspect ratio and orientation in an image, making the complete search space huge. Exhaustive search algorithms, such as the sliding window technique, try to exhaust a constrained search space in which the object of interest will most likely be found. This is done by limiting the possible values of the parameters used. A face detector for example does not require a wide variety of aspect ratio values, however some variety in the number of sizes checked is likely required. Although constraining the search space helps, a large amount of proposals still need to be processed. The reduction in number of proposals generated by selective search algorithms allows the use of computationally more expensive classifiers such as convolutional networks. Unlike exhaustive search methods, SSS uses the contents of the image to guide the proposal generating process. Since the proposals of SSS are based upon segments rather than multiple sliding windows, they provide a much tighter fit around the objects of interest.

Improved exhaustive search algorithms. Other algorithms [15, 22] use the appearance model to help guide the sliding window technique. Using a *branch and bound* technique, [15] drastically reduces the search space. Although reducing the number of proposals generated, [1] argues that for non-linear classifiers [15] still visits over 100.000 windows per image.

FI-SPOT[22] uses the most important features in the appearance model to quickly prune many of the proposed windows, after which the remaining windows are checked using the full appearance model. Although an improvement over the regular sliding window approach, this algorithm still processes a large amount of windows. The algorithm introduced here generates far fewer proposals. Similarly as with the sliding window technique, the nature of segments used in SSS allow a much tighter fit around objects compared to window based approaches. In addition, the algorithm introduced here does not exclude the use of [15, 22]. Both types of algorithms can be used simultaneously to help speed up the object detection pipeline even more.

Prefilter step. A subset of exhaustive search algorithms [5, 2, 11], such as *BING*[5] and *Objectness*[2], uses a filter step on the set of windows obtained through the sliding window technique. These filters determine whether the window is likely to contain an object of interest by measuring the 'objectness' value of that window. If a window is considered likely to contain an object, it becomes an object proposal and is passed down to the rest of the object detection pipeline. Although an improvement over the traditional sliding window technique, these algorithms still function as exhaustive search methods and thus have a large search space to process. SSS produces significantly fewer proposals and in addition, the proposals which SSS generates provide a much tighter fit around the object of interest due to the nature of segmentation. Since [11, 5, 2] are still window based approaches, they often fail to get a high overlap with the object of interest. Also, as with [15, 22], these type of algorithms do not exclude the use of SSS. SSS can be used to propose the initial objects, after which [5, 2] could measures their 'objectness' value to determine which proposals are likely to contain objects.

2.2. Selective Search Algorithms

In contrast to exhaustive search algorithms, selective search algorithms extract information from the image to determine which parts are possible object locations. By doing so, selective search algorithms generally propose far fewer objects. These segmentation based selective search methods use a bottom-up approach. In the bottom-up approach, algorithms start with an oversegmentation of the image and use combinations of segments to obtain proposals. In a good oversegmentation, segments do not cross the contour of objects of interest, however it is allowed for an object to be composed of multiple segments. It is in this category of selective search algorithms where most recent related work is found.

Deterministic selective search. Algorithms like *gPb*[3] and *Selective Search*[19] greedily keep merging neighboring segments together until only one segment remains. This type of proposal algorithms are deterministic selective search algorithms. Eventually, these methods create what is called a segmentation hierarchy. Since these algorithms [3, 19] build up a hierarchy, each object proposal is dependent on the object proposals down the hierarchy which it is composed of. An error in the early steps of the merging process then propagates throughout the hierarchy, influencing all resulting proposals after the wrongful merge. SSS does not suffer from this effect, since it is not building up a segmentation hierarchy. This makes each proposal independent from each other, limiting the effect of a wrongful merge in the process. In addition, deterministic selective search algorithms need image features to determine the order in which segments are merged. SSS does not compute image features to help merge segments. This makes SSS much faster than deterministic selective search algorithms while maintaining comparable accuracy.

Stochastic selective search. Unlike deterministic selective search algorithms, stochastic selective search algorithms do not merge segments in a greedy fashion. Instead, the algorithm introduced here and *Randomized Prim's*[16] produce a chosen amount of subsets from the set of segments. However, SSS is much faster than Randomized Prim's[16], since it does not require the computation of image features for merging segments. This allows SSS to process multiple scales at the same time as other selective search algorithms, similarly to what is done in [19]. Without computing features for the segments we significantly speed up the algorithm, while maintaining comparable accuracy. Experiments comparing the use of features to guide the process and using no features are discussed in section 4.

3. Stochastic Selective Search

As mentioned in the introduction, all segmentation based selective search algorithms work much alike, including Stochastic Selective Search. The first step of SSS is to perform a segmentation using the Felz-Hutt[10] algorithm to create an oversegmentation of the image. Based on this oversegmentation, a graph $G(V, E)$ is constructed that connects every segment (as vertices in V) with their connecting neighbors (through edges in E). Then a starting segment is selected based on the location of its center in the image and training data from the VOC2012 dataset. This initial segment is the first of a set of segments which compose an object proposal. Next, a random number is uniformly generated in a chosen range to determine the amount of segments that are added. The process of selecting a starting segment and adding a random number of segments to generate an object proposal is repeated n times to generate n proposals, where n is selected by the user. This section discusses each of these steps in detail. An overview of the algorithm is given below.

Algorithm 1: Stochastic Selective Search

```
Input  : An image, segmentation parameters $k$ and $\sigma$,
         maximum proposal size $ms$ and the number
         of object proposals to generate $n$.
Output: A list of object proposals in the form of
         bounding boxes.
1  Segment the image using [10] with parameters $k$ and $\sigma$
2  Create a graph $G = (V, E)$ based on the segmented
   image
3  Proposals ← $\emptyset$
4  for $i$ in range $[0, n]$ do
5      $s \leftarrow$ Location($V$)
6      $l \leftarrow$ Uniform($[0, ms]$)
7      for $i$ in range $[0, l]$ do
8          $s \leftarrow$ merge($s$, Uniform($N(s)$))
9      end
10     Append $s$ to Proposals
11 end
```

Herein, G is a graph with vertices V (segments) and edges E, $N(s)$ returns the neighboring segments of segment s, n is the number of proposals that are generated and ms the maximum of each proposal in number of segments added. The Location function returns a segment of V according to a prior distribution over locations.

Segmentation. Creating the segmentation is done using the algorithm of [10] since it is relatively fast, provides excellent results and is easily configured. The Felz-Hutt algorithm has already proven to work successfully in other selective search algorithms[19, 16]. To segment an image, it treats every pixel as a vertex in a graph which has edges connecting each vertex to its four direct neighbors. The Felz-Hutt algorithms gives each edge a weight based on the difference in pixel value for two neighboring pixels. The edges are sorted based on their weights and processed one at a time, processing the most similar segments first. If the connection of an edge is lower than some threshold, the clusters merge and any connected edges to this new cluster has their similarity updated. The threshold in the Felz-Hutt algorithm that determines whether two clusters should be merged can be changed by setting a parameter k, which in effect is proportional to the size of the segments that are produced. A visualization of the effect of different k values is shown in figure 2b and 2c.

The σ value is used to smooth the image before processing using a Gaussian blur. By smoothing the image, edges become less sharp, which increases similarity between pixels. As with k, the size of generated segments is proportional to the value of σ. A visualization of the effect of different σ values is shown in figure 2d and 2e. We found that a value of $k = 130$ and $\sigma = 1.2$ worked well in our experiments.

Generating the graph. The graph that connects segments to one another is efficiently computed by going through every pixel in the segmented image once, where each pixels' value is the index of the segment it belongs to. By looking at the right and lower neighboring pixel we record the neighbors for each segment. In addition, this process allows us to capture the bounding box of each segment, which is used later on. Eventually, this process creates a graph G with vertices (segments) V and edges E, connecting neighboring segments to each other.

Initial segment selection. Selecting a starting segment is done by selecting one at random with a prior probability which favors segments in certain locations. By using a dataset with annotated segmentations, we compute this prior probability. We resize all annotated segmentations to a standard size (we used 512×512) and average all the masks of the segments in the training set to create the location prior. A visualization of this prior probability is shown in figure 3, where we used the segmentation dataset from the VOC2012 dataset. In our case this prior probability resembles that of a Gaussian distribution, but a different prior can be trained depending on the application. To determine the probability for a segment to be selected as starting segment, we look up the location of its center point in the prior probability map. We then sample one segment uniformly using these prior probabilities as weights.

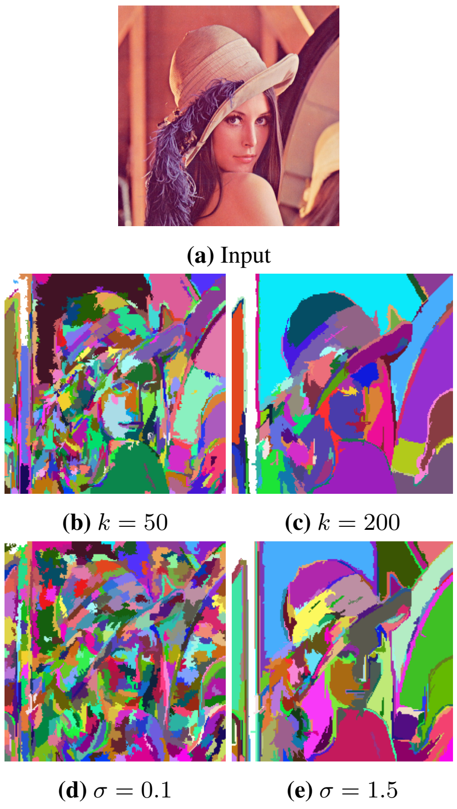


Figure 2: Results of the segmentation algorithm of [10] for varying parameters. (a) Shows the input image. The rest of the images show the segmented result for (b) $k = 50$ and $\sigma = 0.5$, (c) $k = 200$ and $\sigma = 0.5$, (d) $k = 100$ and $\sigma = 0.1$ and (e) $k = 100$ and $\sigma = 1.5$.

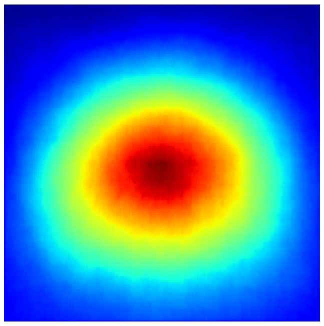


Figure 3: A visualization of the location prior used to select a starting segment using the annotated segmentations from the VOC2012 segmentation dataset.

Merging. The merging process is done by using the information of the graph we created earlier. Each segment has a list of connected neighbors. Using the starting segment we uniformly select a neighboring segment and merge the two segments together. Merging these segments means merging their list of neighbors and updating their neighbors to notify them of the newly merged segment. Each time a new proposal is being generated a number l in the range of $[0, ms]$ is uniformly generated, where ms is the maximum number of segments that can get added. During experiments, we found that a value of 26 for parameter ms appears to give good results. l segments get added to the selected starting segment, resulting in a set of $(l + 1)$ segments. This results in an object proposal, which is one of n proposals generated. Next, a new starting segment is selected and the process is repeated n times.

Multiple strategies. As is done in [19], results can be improved by making use of multiple strategies. A strategy in this sense is a certain configuration of the parameters, where multiple strategies can be used to create segments of different sizes. One strategy may have k and σ set rather small in order to create fine segments, while a second strategy might set these parameters quite high to create coarser segments. This allows SSS to check for objects in multiple scales. Using multiple scales increases the quality, but lowers the efficiency. The processing time is linear in the number of strategies selected, a setup with 10 strategies takes approximately 10 times longer than a single strategy approach. Parallelization can easily be applied here, since the strategies are completely independent. For the experiments later on we did not use multiple strategies, since these only complicate the comparison with other algorithms.

4. Experiments

This section describes the experiments we performed to test the quality and efficiency of SSS and compare it to other state-of-the-art algorithms.

4.1. Dataset

We use the VOC2007 dataset[8] for evaluation since many algorithms[13] are already compared using that dataset, which allows us to easily compare SSS with state-of-the-art algorithms. The VOC2007 dataset contains 4952 images with one or more annotations per image. A few examples are shown in figure 4. In total there are 20 classes in the VOC2007 dataset, such as dogs, trains, birds, bikes, etc.

4.2. Setup

Most selective search algorithm papers[16, 19] compare the quality of the proposals on a dataset, but we argue that an object proposal algorithm should optimize both quality and efficiency to be a viable alternative to the sliding window approach. We measure efficiency in two ways: as the time it takes for the algorithm to generate all proposals and the number of proposals generated. If the object proposal algorithm is very fast but generates far too many proposals, the object detection pipeline as a whole still suffers and becomes inefficient. In addition, we measure the quality of the proposals by computing the intersection-over-union[8] (*IoU*) score. This value is computed as follows:

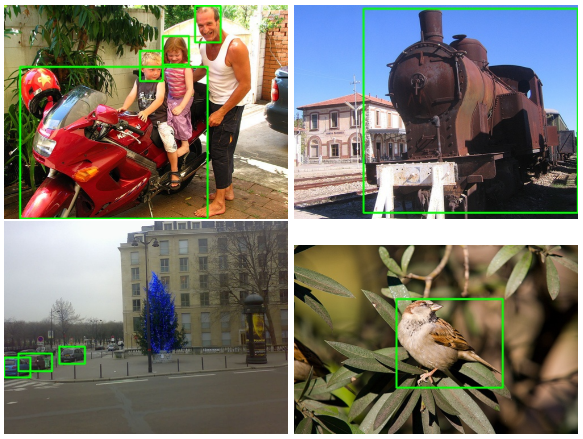


Figure 4: A few example images of the VOC2007 dataset with the supplied annotations.

$$O(a, b) = \frac{R_a \cap R_b}{R_a \cup R_b} \tag{1}$$

Here, O is the intersection over union score for segments a and b, R_s is the area of the region covered by segment s, $R_a \cap R_b$ is the joint area of the region covered by *both* segments a and b in number of pixels and $R_a \cup R_b$ is the area of 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 compute the best IoU score for that object. We do this for every object of each class in a dataset and in such a way compute the average best IoU score for every class. By computing the mean of these class scores we obtain the mean average best overlap (MABO) score, which is one value representing the quality of the algorithm. While optimizing parameters we use this MABO score.

To compare the quality of algorithms with each other, we compute the best IoU score for all objects in the dataset and create IoU detection rate graphs. These graphs show on the horizontal axis the threshold in IoU score and on the vertical axis the percentage of detections that have a higher IoU score. The algorithms in these graphs are compared to one another through the 'area under graph' (AUC) value, as is done in [13].

In this paper we present five experiments, for which we created three slightly modified algorithms based on the code of SSS to ensure the difference in results is caused by a difference in theory, not in implementation quality. Firstly, we perform a sensitivity analysis on the hyperparameters of SSS. We do this by varying the parameters and recording the MABO, time to process and number of bounding boxes generated. Secondly, we compare SSS with DSS, a variant of SSS which constructs a segmentation hierarchy in a deterministic manner. Thirdly, we compare SSS to two other versions of SSS which use features (F-SSS) and weighted features (WF-SSS) to guide the merging process of segments. Fourthly, we compare the quality of SSS with other algorithms using the code provided by their authors and the evaluation framework of [13]. Lastly, we use a state-of-the-art convolutional network[14] to test the effect of SSS in a complete object detection pipeline.

4.3. Results

Here we present the results from the experiments in the order as mentioned above.

Sensitivity analysis of hyperparameters. The SSS algorithm uses a few parameters that influences its quality. In order to optimize these values, we performed a 10-fold cross-validation experiment on the VOC2012 training data. Each fold evaluates the MABO score of the parameters in some range, the best scoring value is then scored using the test data of this fold. These scores determine which parameter value provides the best end result. A visualization of one fold of this experiment is shown in figure 5. This cross-validation experiment resulted in the following optimal parameters: $k = 130$, $\sigma = 1.2$ and $ms = 26$. Since a larger value for k and σ produce larger segments, SSS more often creates duplicate bounding boxes. Since only the number of unique bounding boxes is counted, the number of bounding boxes decreases as these parameters increase.

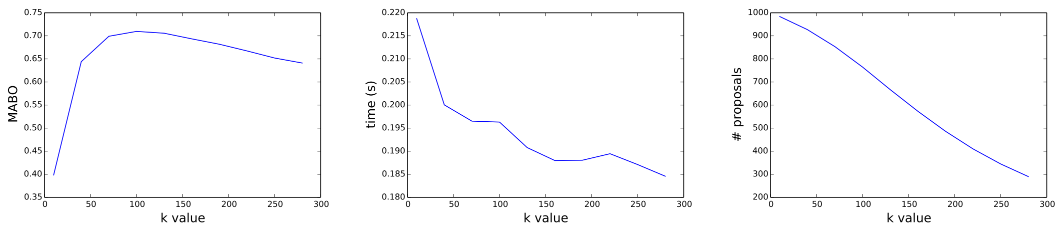
Deterministic vs. Stochastic Selective Search. In order to evaluate the difference between creating a deterministic segmentation hierarchy and selecting subsets of segments in a stochastic manner, we created a deterministic selective search algorithm based on the code of SSS. The deterministic selective search (DSS) works very similar to [19] in that it uses the same features (color, texture, size and fill) to determine the order in which the segments are merged. Since DSS and SSS share similar code, we are well equipped to compare the difference in efficiency and quality between a deterministic segmentation hierarchy and stochastic selective search. A comparison of SSS and DSS is shown in figure 6. It is clear that SSS outperforms DSS by a significant amount at every IoU threshold, illustrating the potential merits of a stochastic approach over a deterministic approach.

Effect of using features to guide the merging of segments. Randomized Prim's[16] uses almost identical features as [19] to guide the merging process. However, it is interesting to see how effective these features really are; do features make a significant difference in guiding the process of merging segments? To evaluate the effect of features as weights instead of uniformly sampling a neighbor, we created a feature based version of SSS (F-SSS). As described in [16], we also use a logistic function to add weights per feature. This leads to a small modification: a weighted features SSS (WF-SSS) algorithm. An IoU graph of SSS, F-SSS and WF-SSS is shown in figure 7 and a summary is shown in table 1. It is shown in both table 1 and figure 7 that using features described by [19] does not significantly improve the result on the final object proposal algorithm. SSS scores just marginally lower than F-SSS, however it is more than twice as fast. This is also shown in figure 8, where we show the MABO on the horizontal axis and the average time to process on the vertical axis. The increase in score does not outweigh the increased time the algorithm needs to process an image. This result suggests that using features does not significantly improve the merging process of segments. This is presumably caused by most objects having widely varying segments in terms of the features that are used. Segments from a persons face and hair have completely different features, however they belong to the same object.

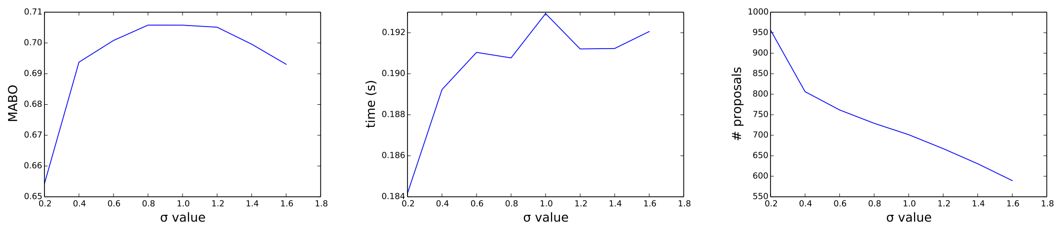
Algorithm	MABO	Time (s)	# proposals
SSS	0.696	**0.174**	**517**
F-SSS	**0.697**	0.440	524
WF-SSS	0.693	0.437	551

Table 1: Results of comparison between SSS and modifications of SSS using features (F-SSS) and weighted features (WF-SSS).

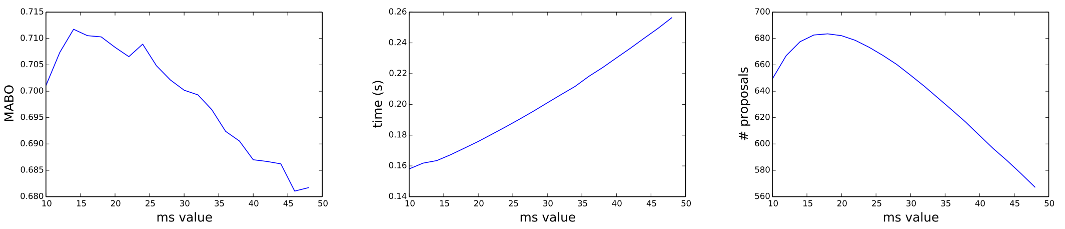
Comparison with state-of-the-art. To compare SSS to state-of-the-art algorithms, we use the framework provided by [13]. The result of this experiment is shown in figure 9. Computing the data for the framework of [13] amounts to providing a set of bounding boxes (object proposals) for each image in the VOC2007 dataset, which we did for all algorithms but BING, Objectness and the sliding window technique. These algorithms use the data provided by [13] to generate the graph. The number behind the names of the algorithms in figure 9 represent the area under the curve (AUC) score, whereas the number between parenthesis represents the average number of object proposals used. As is shown, both SSS and Randomized Prim's give the best results: both obtain an AUC score of 62.9. This is no surprise, as both methods are very similar. The main difference however is that SSS is more than twice as fast as Randomized Prim's, which is the result of SSS not using image features to merge segments. A summary of this experiment is shown in tabel 2. The increase in performance is essential if selective search algorithms are to be a viable alternative to the sliding window approach for realtime applications. In figure 9 the sliding window approach performs very poorly. This is caused by the number of windows the sliding window approach is allowed to generate. The power of the sliding window approach is that it exhausts many of the possible



(a) Results for SSS when varying the parameter k. $\sigma = 1.2$, $ms = 26$, $n = 1000$.



(b) Results for SSS when varying the parameter σ. $k = 130$, $ms = 26$, $n = 1000$.



(c) Results for SSS when varying the maximum number of segments in a proposal. $k = 130$, $\sigma = 1.2$, $n = 1000$.

Figure 5: Results for varying the parameters of SSS: k, σ and the maximum number segments added per proposals. The number of proposals generated is always 500. However, only the number of unique proposals are counted in the third column.

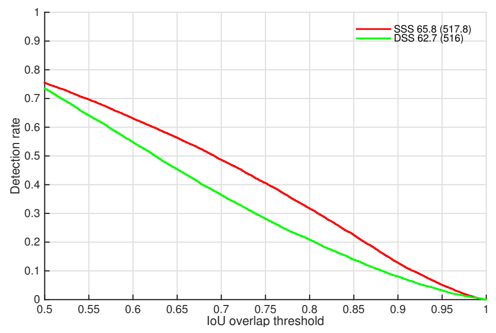


Figure 6: IoU detection rate for deterministic selective search (DSS) and SSS. DSS took on average 0.47s per image, while SSS took on average 0.17s. The number behind the name is the area under the curve value. The number between parenthesis is the average number of proposals generated.

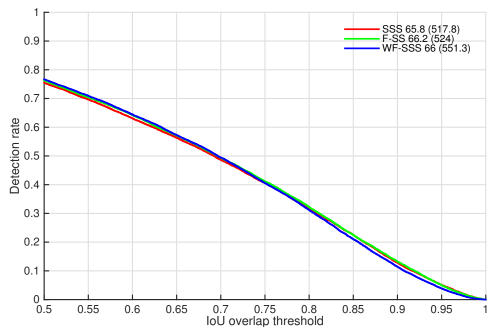


Figure 7: IoU detection rate for SSS, SSS using features (F-SSS) and SSS using weighted features (WF-SSS). The number behind the name is the area under the curve value. The number between parenthesis is the average number of proposals generated.

windows, but limiting the number of windows generated to a few hundred cripples the sliding window approach significantly.

SSS in an object detection pipeline. In order to show that SSS can improve the result of a complete object detection pipeline, we combined SSS with a state-of-the-art convolutional network[14] using the R-CNN model[12]. The R-CNN model is a state of the art model which achieved 7th place in the ILSVRC2014 competition for object detection, with an average precision of 34.52%. In our setup, SSS generates object proposals from an image which the convolutional network processes to determine the content. For this

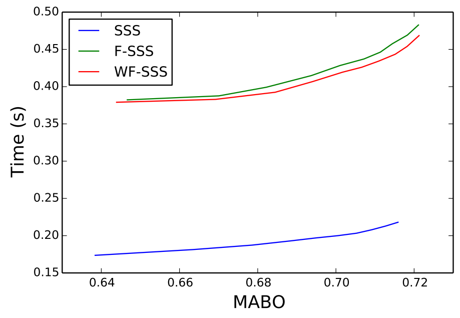


Figure 8: A comparison which shows time versus MABO for SSS, SSS using features (F-SSS) and SSS using weighted features (WF-SSS).

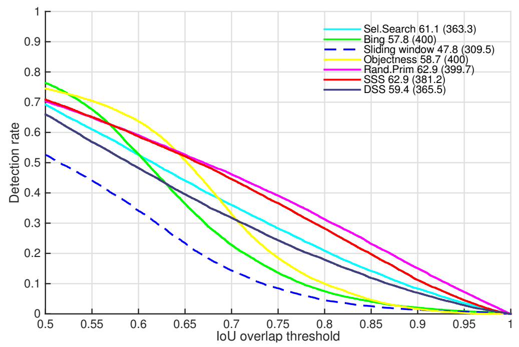


Figure 9: IoU detection rate graph for SSS and state-of-the-art algorithms. The number behind the name is the area under the curve value. The number between parenthesis is the average number of proposals generated.

Algorithm	AUC	Time (s)	# proposals
SSS	**62.9**	**0.174**	381.2
DSS	59.4	0.450	365.5
Sel.Search[19]	61.1	0.416	363.3
Rand.Prim's[16]	**62.9**	0.587	399.7
BING[5]	57.8	-	400
Objectness[5]	58.7	-	400
Sliding Window	47.8	-	309.5

Table 2: Results of comparison between SSS and state-of-the-art algorithms.

experiment SSS generated on average 528 object proposals. We used the ILSVRC2013[18] dataset, because [14] provides a pretrained model for this dataset. The output of the detection pipeline contains 200 confidence scores per proposal, one score for each object class in the dataset. Using non-maxima supression we eliminate many of the object locations that are covered by more confident detections. The remaining object detections are used in this experiment. To compare with other algorithms we also experimented with the deterministic variant of SSS (DSS) and the original Selective Search[19]. All three algorithms used the same segmentation parameters. In order to compare the results with

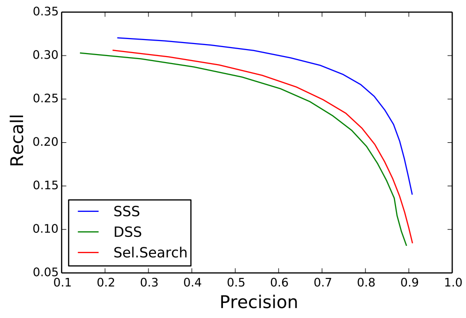


Figure 10: Curve which shows the precision versus recall for SSS, DSS and Selective Search[19]. All three algorithms generated around 500 object proposals.

each other we created a recall-precision graph. Recall is the percentage of annotated objects that have been properly detected, whereas precision is the percentage of detections which were correct. The result for this experiment is shown in figure 10. This result illustrates that the ability of Stochastic Selective Search to recover from merging errors leads it to outperform deterministic selective search methods.

The convolutional network[14, 12] in our experiments processed on average 67 object proposals per second. If we were to use a sliding window technique which generates a few hundred thousand proposals the detection pipeline would be much slower. If for example 400.000 object proposals are generated, the processing time would be nearly 6.000 seconds. Instead, the entire pipeline took on average about 10 seconds when using our setup with a selective search method. The reduction in number of proposals generated by selective search methods becomes a significant advantage in pipelines such as these.

5. Conclusions and Future Work

This paper proposed a new object proposal algorithm, with as goal to work towards a viable alternative for the sliding window technique. Existing state-of-the-art object proposal algorithms, although often of high quality, are not efficient enough with respect to processing time compared to the sliding window technique. SSS aims to find a balance between quality and efficiency. SSS is much faster due to its simplicity, while it produces candidate locations of the same quality as current state-of-the-art algorithms. In addition, SSS is resilient to mistakes made in the merging process, unlike its deterministic counterpart, and SSS allows the use of "detection on a budget". Although not as fast at generating the object proposals as the sliding window technique, SSS produces far fewer proposals. Combined with classifiers such as convolutional networks, which require a lot of computation per object proposal, this results in a much faster algorithm.

Future work. In order to increase efficiency, the segmentation algorithm which is used could be improved. The segmentation algorithm uses more than half of the time required for SSS to process an image. In addition, when creating a deterministic segmentation hierarchy, each object proposal is dependent on the generated object proposals it is composed of. SSS does not have this dependency, which would allow the generation process to be run in parallel.

In order to obtain better quality proposals, SSS can be combined with the appearance model of a classifier. By adding prior knowledge about the object of interest, such as its size, color or location in the image, SSS can be improved to use a more directed search for segments matching those descriptions. In addition it would be interesting to see SSS combined with other work such as [15, 22, 5, 2] to potentially improve the quality further without losing much efficiency.

Acknowledgements. I would like to acknowledge the outstanding help and feedback I have received from my supervisor, Laurens van der Maaten. I am thankful he introduced me to the topic of selective search algorithms, but also for guiding me whenever I needed guidance and giving me freedom to do my work whenever I needed freedom. I am also especially thankful for the very interesting weekly discussions we had on how to improve this field of research further. I have improved a lot in the field of scientific research and writing scientific documents.

I would also like to acknowledge the help of John Beuving, who helped me set up my code in his model-free tracker using convolutional networks[4]. Using this setup, we managed to experiment the effect of SSS in a full tracking system. The positive results from this experiment were very encouraging.

References

[1] B. Alexe, T. Deselaers, and V. Ferrari. What is an object? In *The Twenty-Third IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2010, San Francisco, CA, USA, 13-18 June 2010*, pages 73–80, 2010.

[2] B. Alexe, T. Deselaers, and V. Ferrari. Measuring the objectness of image windows. *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, 2012.

[3] P. Arbelaez, M. Maire, C. Fowlkes, and J. Malik. Contour detection and hierarchical image segmentation. *IEEE Trans. Pattern Anal. Mach. Intell.*, 33(5):898–916, May 2011.

[4] J. Beuving, L. van der Maaten, and H. Gaiser. Model-free tracking using convolutional networks. In *IEEE CVPR*, 2014.

[5] M.-M. Cheng, Z. Zhang, W.-Y. Lin, and P. H. S. Torr. BING: Binarized normed gradients for objectness estimation at 300fps. In *IEEE CVPR*, 2014.

[6] N. Dalal and B. Triggs. Histograms of oriented gradients for human detection. In C. Schmid, S. Soatto, and C. Tomasi, editors, *International Conference on Computer Vision & Pattern Recognition*, volume 2, pages 886–893, INRIA Rhône-Alpes, ZIRST-655, av. de l'Europe, Montbonnot-38334, June 2005.

[7] I. Endres and D. Hoiem. Category independent object proposals. In *Proceedings of the 11th European Conference on Computer Vision: Part V*, ECCV'10, pages 575–588, Berlin, Heidelberg, 2010. Springer-Verlag.

[8] M. Everingham, L. Gool, C. K. Williams, J. Winn, and A. Zisserman. The pascal visual object classes (voc) challenge. *Int. J. Comput. Vision*, 88(2):303–338, June 2010.

[9] P. F. Felzenszwalb, R. B. Girshick, D. McAllester, and D. Ramanan. Object detection with discriminatively trained part-based models. *IEEE Trans. Pattern Anal. Mach. Intell.*, 32(9):1627–1645, Sept. 2010.

[10] P. F. Felzenszwalb and D. P. Huttenlocher. Efficient graph-based image segmentation. *Int. J. Comput. Vision*, 59(2):167–181, Sept. 2004.

[11] J. Feng, Y. Wei, L. Tao, C. Zhang, and J. Sun. Salient object detection by composition. In *Computer Vision (ICCV), 2011 IEEE International Conference on*, pages 1028–1035. IEEE, 2011.

[12] R. Girshick, J. Donahue, T. Darrell, and J. Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2014.

[13] J. Hosang, R. Benenson, and B. Schiele. How good are detection proposals, really? In *BMVC*, 2014.

[14] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. B. Girshick, S. Guadarrama, and T. Darrell. Caffe: Convolutional architecture for fast feature embedding. *CoRR*, abs/1408.5093, 2014.

[15] C. H. Lampert, M. B. Blaschko, and T. Hofmann. Efficient subwindow search: A branch and bound framework for object localization. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 31(12):2129–2142, 2009.

[16] S. Manén, M. Guillaumin, and L. Van Gool. Prime Object Proposals with Randomized Prim's Algorithm. In *International Conference on Computer Vision (ICCV)*, Dec. 2013.

[17] T. Ojala, M. Pietikäinen, and D. Harwood. A comparative study of texture measures with classification based on featured distributions. *Pattern recognition*, 29(1):51–59, 1996.

[18] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei. ImageNet Large Scale Visual Recognition Challenge, 2014.

[19] J. R. R. Uijlings, K. E. A. van de Sande, T. Gevers, and A. W. M. Smeulders. Selective search for object recognition. *International Journal of Computer Vision*, 104(2):154–171, 2013.

[20] P. Viola and M. Jones. Rapid object detection using a boosted cascade of simple features. pages 511–518, 2001.

[21] P. Viola and M. J. Jones. Robust real-time face detection. *Int. J. Comput. Vision*, 57(2):137–154, May 2004.

[22] L. Zhang, H. Dibeklioglu, and L. van der Maaten. Speeding up tracking by ignoring features. June 2014.