Superpixels and Supervoxels in an Energy Optimization Framework

Olga Veksler, Yuri Boykov, and Paria Mehrani

Computer Science Department, University of Western Ontario
London, Canada
{olga,yuri,pmehrani}@uwo.ca
http://www.csd.uwo.ca/faculty/olga/

Abstract. Many methods for object recognition, segmentation, etc., rely on a tessellation of an image into "superpixels". A superpixel is an image patch which is better aligned with intensity edges than a rectangular patch. Superpixels can be extracted with any segmentation algorithm, however, most of them produce highly irregular superpixels, with widely varying sizes and shapes. A more regular space tessellation may be desired. We formulate the superpixel partitioning problem in an energy minimization framework, and optimize with graph cuts. Our energy function explicitly encourages regular superpixels. We explore variations of the basic energy, which allow a trade-off between a less regular tessellation but more accurate boundaries or better efficiency. Our advantage over previous work is computational efficiency, principled optimization, and applicability to 3D "supervoxel" segmentation. We achieve high boundary recall on images and spatial coherence on video. We also show that compact superpixels improve accuracy on a simple application of salient object segmentation.

Keywords: Superpixels, supervoxels, graph cuts.

1 Introduction

Many vision applications benefit from representing an image as a collection of *superpixels*, for example [1,2,3,4,5,6,7,8], to cite just a few. While the exact definition of a superpixel is not feasible, it is regarded as a perceptually meaningful atomic region. A superpixel should contain pixels that are similar in color, texture, etc., and therefore are likely to belong to the same physical world object. The atomic region notion is old, but a popular term *superpixel* has been coined recently [1].

The assumption that all pixels in a superpixel belong to the same object leads to the advantage of superpixel primitives over pixel primitives. The first advantage is computational efficiency. If one needs to compute a property that stays approximately constant for an object, then superpixel representation is more efficient since the total number of primitives is greatly reduced [9]. Computational efficiency also comes from a reduction in the number of hypothesis. Instead of

K. Daniilidis, P. Maragos, N. Paragios (Eds.): ECCV 2010, Part V, LNCS 6315, pp. 211–224, 2010.

exhaustive examining of all rectangular patches [10], an alternative is to examine only superpixels [1,2,4,5,6,7]. In addition to efficiency, superpixels are used for computing features that need spatial support [3].

To obtain superpixels, one often uses image segmentation algorithms such as meanshift [11], graph based [12], normalized cuts [13]. To increase the chance that superpixels do not cross object boundaries, a segmentation algorithm is run in an oversegmentation mode. However, most segmentation algorithms produce regions of highly irregular shape and size, for example the meanshift [11] and the graph-based [12], see Fig. 1, first two images. The boundaries are also highly irregular, since there is no explicit constraints on length. A large superpixel with a highly irregular shape is likely to straddle more than one object.

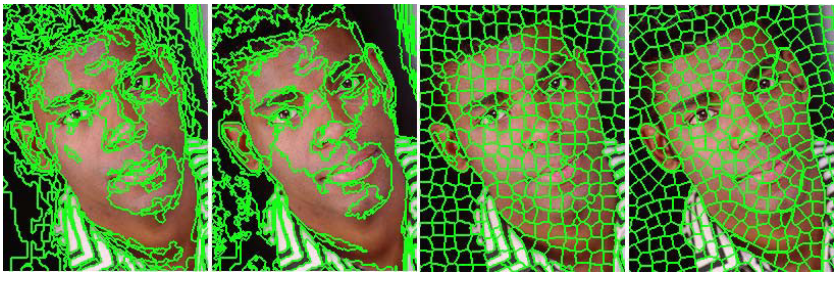


Fig. 1. From left to right: meanshift [11], graph based [12], turbopixels [14], NC superpixels [1]. Implementation was obtained from the authors' web sites.

There are advantages to superpixels with regular shapes and sizes, such as those in Fig. 1, right. A regular shape is less likely to cross object boundaries, since objects rarely have wiggly shapes. If a superpixel does cover more than one object, if its size is not too large, the error rate is likely to be controlled.

The normalized cuts algorithm [13] can be adapted to compute superpixels that are regular in size and shape [1], see Fig. 1. Many methods that need regular superpixels use normalized cuts [9,1,2,4,5]. However, NC superpixels [1] are very expensive, and have the following unappealing property, noticed by [14]. The smaller is the size of target superpixels, the longer the computation takes.

Our work was inspired by the turbopixel algorithm [14], Fig. 1. It is based on curve evolution from seeds placed regularly in the image, which produces a regular "turbopixel" space tessellation. Using various constraints during curve evolution, they encourage a uniform space coverage, compactness of superpixels in the absence of image edges, and boundary alignment when image edges are present. They have to devise a collision detection mechanism to insure no turbopixels overlap. The algorithm runs in seconds on the images in Berkeley dataset [15], as compared to minutes with NC superpixels [1].

We propose a principled approach to compute superpixels in an energy minimization framework. Our method is simple to understand and implement. The basic algorithm, illustrated in Fig. 2, is similar in spirit to texture synthesis [16].

Fig. 2. Overview of our algorithm. Left: the original image overlayed with square patches. For clarity, only some patches are shown. Middle: result of patch stitching. Right: superpixel boundaries.

An image is covered with overlapping square patches of fixed size, Fig. 2, left. Each pixel is covered by several patches, and the task is to assign a pixel to one of them. If two neighboring pixels are assigned to the same patch, there is no penalty. If they belong to different patches, then there is a stitching penalty that is inversely proportional to the intensity difference between the pixels. Intuitively, we are stitching patches so that the seams are encouraged to align with intensity edges. The stitching result is in Fig. 2, middle, and superpixel boundaries are in Fig. 2, right. Boundaries are regularized due to the stitching energy function. A superpixel cannot be too large, not larger than a patch size. Small superpixels are discouraged because they contribute a higher cost to the stitching energy. Thus the sizes of superpixels are also regularized. We extend this basic algorithm to other formulations, which allow a trade-off between a less regular space tessellation but more accurate boundaries or better efficiency.

Our work has several advantages over turbopixels [14]. First, we have an explicit energy, and a principled way to optimize it. In contrast, the method in [14] is described only procedurally. Our approach is simpler to understand and analyze. Unlike [14], we do not need a collision detection mechanism, overlap is not allowed by design. Since we have an explicit energy function, we can change its terms to encourage different superpixel types. One modification we add is a term that encourages intensity homogeneity inside a superpixel, not something that is easy to include explicitly into [14]. Another advantage is optimization. Turbopixels are based on level set evolution [17], which is known to have numerical stability issues. We optimize with graph cuts [18], which is known to perform well [19]. Our running time is better. Last, but not least, our approach naturally transfers to 3D for "supervoxel" segmentation of video.

An interesting work on superpixels is in [20,21]. Their goal is somewhat different from ours. They seek superpixels conforming to a grid, which has storage and efficiency advantages. The work in [20] is based on greedy optimization, and [21] uses a more global approach, which, like our work, is also based on graph cuts. However, the formulation in [20,21] poses restrictions on superpixel shapes: the boundary between superpixels cannot "turn back" on itself.

We evaluate our approach on Berkeley dataset [15] and show that we achieve high boundary recall and low undersegmentation error, similar or better than that of [1,14]. We also show that our supervoxels have a high spatial coherence on 3D volumes constructed from video. To show that compact superpixels are more appropriate for some applications, we compare the performance of our superpixels vs. those of [12] on a simple application of salient object segmentation.

2 Superpixel Segmentation

In this section we give a detailed description of our superpixel segmentation approach. We review graph cut optimization in Sec. 2.1. Then we explain the basic "compact" superpixel algorithm is in Sec. 2.2. In Sec. 2.3 we show how to incorporate variable patch size. The resulting algorithm is called "variable patch" superpixels. Variable patch superpixels are more efficient computationally, and their boundary recall does not suffer a performance loss. However they do have more widely varying sizes. Lastly, in Sec. 2.4 we show how to incorporate intensity constancy constraints, and the resulting algorithm is called "constant intensity" superpixels. Constant intensity superpixels perform better on boundary recall, but, again, have more widely varying sizes.

2.1 Energy Minimization with Graph Cuts

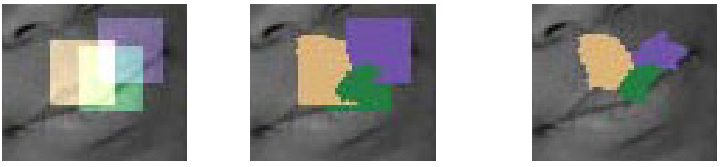
We now briefly review the graph-cut optimization approach [18]. Many problems in vision can be stated as labeling problems. Given a set of pixels $\mathcal{P}$ and a finite set of labels $\mathcal{L}$, the task is to assign a label $l \in \mathcal{L}$ to each $p \in \mathcal{P}$. Let f_p denote the label assigned to pixel p, and let f be the collection of all label assignments. There are two types of constraints. Unary constraints $D_p(l)$ express how likely is a label l for pixel p. Binary constraints $V_{pq}(l_1, l_2)$ express how likely labels l_1 and l_2 are for neighboring pixels p and q. An energy function is:

$$E(f) = \sum_{p \in \mathcal{P}} D_p(f_p) + \lambda \sum_{\{p,q\} \in \mathcal{N}} w_{pq} \cdot V_{pq}(f_p, f_q), \tag{1}$$

In Eq. (1), the first and the second sums are called the data and the smoothness terms, and $\mathcal{N}$ is a collection of neighboring pixel pairs. We use 8-connected grid, and Potts model $V_{pq}(f_p, f_q) = \min(1, |f_p - f_q|)$. The coefficients w_{pq} are inversely proportional to the gradient magnitude between p and q, encouraging discontinuities to coincide with intensity edges. This energy is NP-hard to optimize. We use the expansion algorithm from [18], which guarantees a factor of 2 approximation. For the max-flow/min-cut algorithm, we use [22].

2.2 Compact Superpixels

First recall the intuitive explanation, Fig. 2. We cover an image with overlapping square patches of fixed size, equal to the maximum allowed superpixel size. We



(a) Three patches (b) Their optimal stitching (c) In the final stitching

Fig. 3. A simple illustration of patch stitching. Left: orange, green, and purple patches. Middle: their optimal stitching. Right: their optimal stitching in the final result.

seek a stitching of the patches, or, in other words, an assignment of each pixel to a unique patch. The stitches cost cheaper if they coincide with intensity edges. A simple illustration is in Fig. 3. Suppose only three patches in Fig. 3(a) participate. There is a strong intensity gradient on the lip boundary, and therefore the cut between the patches aligns to the lip boundary, Fig. 3(b). Fig. 3(c) shows the shape of these patches in the final stitching, with all patches participating.

We now formalize the problem in the energy minimization framework. We number allowed patches with consecutive integers $1, ..., k$, where k is the number of allowed patches. We identify ith patch with an integer label i, therefore $\mathcal{L} = \{1, 2, ..., k\}$. Even though $\mathcal{L}$ is ordered, this order has no meaning. Let $S(l)$ denote the set of the pixels contained in patch $l \in \mathcal{L}$. For example, in Fig. 3(a), if l is the "orange" label, then $S(l)$ is the set of pixels covered by the orange square. Label l can be assigned only to pixels in $S(l)$. Therefore the data term is:

$$D_p(l) = \begin{cases} 1 & \text{if } p \in S(l) \\ \infty & \text{otherwise} \end{cases} \tag{2}$$

We have to decide how many patches to use and how to spread them out in the image. We address these issues after the energy function is completely specified.

We now discuss the smoothness term. To better approximate Euclidean metric [23] we use 8-connected $\mathcal{N}$. V_{pq} is Potts model with w_{pq} from [24]: $w_{pq} = exp(-\frac{(I_p - I_q)^2}{dist(p,q) \cdot 2\sigma^2})$. Here I_p is the intensity of pixel p, and $dist(p, q)$ is the Euclidean distance between p and q.

Observe that with D_p as defined in Eq. (2), the data term in Eq. (1) is equal for all finite energy labelings. This implies that parameter λ in Eq. (1) has no effect on optimization, so we set $\lambda = 1$. Usually λ is an important parameter to choose correctly as it controls the relative weight between the data and the smoothness terms, and, therefore, the length of the boundary. Parameter λ is often set by hand through a tedious trial and error process. In our case the parameter that controls the boundary length is the patch size. Larger patches lead to fewer boundaries in the optimal labeling. Patch size is in some sense a more "natural" parameter since it is chosen by the user to control the size of the maximum superpixel, as appropriate for an application.

We now address the question of how many patches to place in the image. Observe that only some labels (patches) are present in the final labeling. It is

clear from our energy function that the more patches we have, the lower is the final energy, since adding patches only helps to discover better stitches. Thus for the best stitching, we should use a dense strategy, i.e. put a patch at every image pixel. The dense strategy is too expensive. In practice we obtain good results by spreading patches at intervals four times less than the length of the square side.

We use the expansion algorithm [18] for optimization. It does not guarantee an optimum but finds an approximation within a factor of two. We initialize by randomly picking a label l, and assigning l to pixels in $S(l)$ until there are no uninitialized pixels. An intuitive optimization visualization is as follows. An expansion for label l improves the boundaries under the patch $S(l)$ and its border.

In addition to the maximum size, the minimum superpixel size is also controlled. Suppose that there is a small superpixel A. Then for any neighboring superpixel B, there is no label l s.t. the patch $S(l)$ completely covers A and B. Otherwise, an expansion on l would obtain a smaller energy by assigning l to pixels in $A \cup B$, since the boundary between B and A disappears and no new boundary is created. The smaller is A, the less likely it is that there is no neighboring superpixel B s.t. A and B are covered completely by some patch.

Despite a large number of labels, for our energy the expansion algorithm is very efficient. An expansion on label l needs to be performed only for pixels in $S(l)$. This is both memory and time efficient. We run the expansion algorithm for two iterations, and it takes about 5 seconds for Berkeley images [15]. Our algorithm would be easy to implement on multiple processors or GPU.

To summarize, the properties of compact superpixels are as follows. In the presence of large image gradient, superpixel boundaries are encouraged to align with image edges. In the absence of large gradient, superpixels tend to divide space into equally sized regular cells. Superpixel sizes tend to be equalized, and their boundaries are encouraged to be compact by the energy function.

2.3 Variable Patch Superpixels

In the previous section we assumed that the patch size is fixed. This helps to ensure that the superpixel sizes are equalized. If one is willing to tolerate a wider variance in superpixel sizes, then it makes sense to allow larger superpixels in the areas with lower image variance.

We develop a simple approach to variable patch superpixels. We allow a variable set of square patches, with the smallest side of size k_{min} and the largest of size k_{max}. Let S be a be a patch centered at pixel p. Let $C(S)$ be the square patch of side twice less than the side of S also centered at p, i.e. $C(S)$ is the "central" part of S. Let $P(S)$ be the set of pixels contained in S but not in $C(S)$. As a measure of quality of S we take $Q(S) = var(C(S)) - var(P(S))$. Here $var(S)$ measures the intensity variance in the patch S. The lower is $Q(S)$, the better is the quality of a patch. That is we want the central part of a patch to be of low variance and the periphery to have a high variance. The expectation is that the inside part of patch S is not going to contain stitches, and therefore should be uniform in intensity. The cuts are encouraged to lie in the periphery of the patch S, therefore this part is encouraged to have a high variance.

We measure the quality of all possible patches of sizes in the range from k_{min} to k_{max}. This can be done efficiently using integral images [10]. After this, we sort all patches in terms of quality and select the m best ones so that each pixel is contained in at least 4 patches. We found experimentally that variable patch superpixels do not worsen the boundary recall compared to compact superpixels, while improving efficiency by about a factor of 2.

2.4 Constant Intensity Superpixels

Since we formulate superpixel segmentation in the energy minimization framework, we can change certain properties of superpixels by simply changing the energy function. We now address one useful change. In the energy for compact superpixels, Sec. 2.2, there is no explicit encouragement that superpixels have constant intensity. Consider a grey and white rectangles adjacent to each other in front of a black background. If there is a patch that covers both rectangles, they will be assigned to the same superpixel, since there is no incentive to split them across two superpixels, regardless of their difference in intensity.

We can explicitly encourage constant intensity inside a superpixel but at the price of obtaining superpixels that are less equalized in terms of size. Let $c(l)$ be the pixel at the center of patch $S(l)$. We change the data term to:

$$D_p(l) = \begin{cases} |I_p - I_{c(l)}| & \text{if } p \in S(l) \\ \infty & \text{otherwise} \end{cases} \tag{3}$$

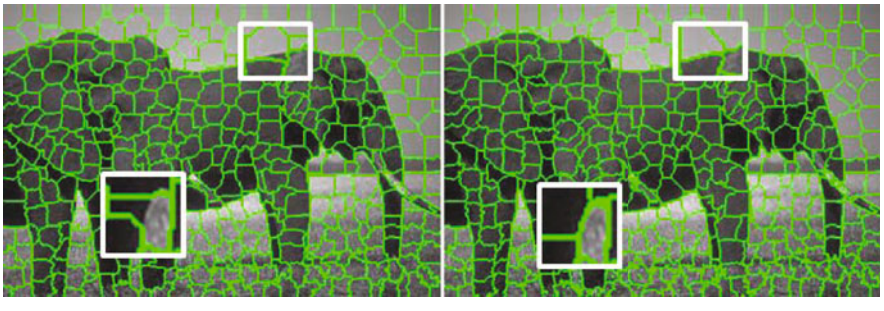


Fig. 4. Two enlarged pieces overlayed over original superpixel images. Left: Compact superpixels, part of the boundaries between elephant legs and on top are missed. Right: constant intensity superpixels, these boundaries are captured.

Now each pixel that is assigned label l is encouraged to be of the same intensity as the center of patch $S(l)$. To ensure this new energy is not increasing during optimization, we have to make sure that that if p is assigned label l, then the center of the patch $c(l)$ is also assigned l. We can easily do this with addition of the following new term $T_{new}(f)$ to the energy in Eq. (1):

$$T_{new}(f) = \sum_{p \in \mathcal{P}} W(f_p, f_{c(f_p)}), \tag{4}$$

where $W(\alpha, \beta) = \infty$ if $\alpha \neq \beta$ and 0 otherwise.

See Fig. 4 for an example where intensity constancy constraint helps to get more accurate boundaries. The cost is that approximately 20% more superpixels are found for the same patch size, some being quite small.

3 Supervoxel Segmentation

Our approach naturally extends to segmenting "supervoxels" in 3D space. A voxel has three coordinates (x, y, t), with t being the third dimension. A supervoxel is a set of spatially contiguous voxels that have similar appearance (intensity, color, texture, etc.). Notice that the slices of a voxel at different values of the coordinate t do not necessarily have the same shape. Segmentation of volumes into supervoxels can be useful, potentially, for medical image and for video processing. In particular, for video processing, there is an interest in coherent 3D segmentation for video abstraction and animation [25,26].

First we create a 3D volume by stacking the frames together, Fig. 5, left. Analogously to the 2D case, we cover the 3D volume by overlapping 3D blocks. For clarity, in Fig. 5 we show only a few non-overlapping blocks. The depth of a block can be different from its width and height. The larger the depth, the more temporal coherency is encouraged. As before, each block corresponds to a label. $\mathcal{N}$ is now 16-connected and contains neighbors between the frames. Just as in the 2D case, we place blocks overlapping in step size equal to a quarter of the

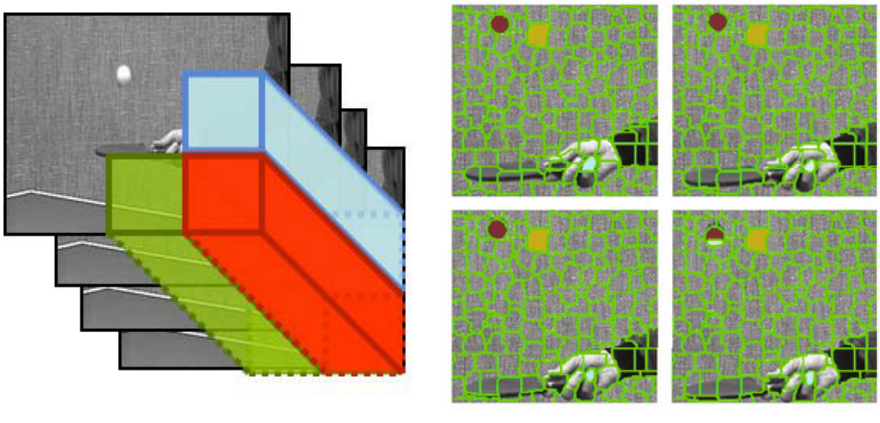


Fig. 5. Supervoxels. Left: video frames are stacked into a 3D volume and covered by a set of 3D blocks. Right: supervoxels, shown separately in each frame. Three supervoxels are highlighted with color (red, yellow, light blue). This figure is better viewed in color.

size of the block (in each dimension). The algorithm is efficient, since we only need to work on a little more than a single block at a time.

Fig. 5, right, shows the results on four consecutive frames of the "tennis" video sequence. We show the section of supervoxels with each frame separately. Notice the high degree of spatial coherency between the frames.

4 Experimental Results

First we evaluate how well superpixel boundaries align to image edges. We use Berkeley database [15] that has ground truth provided by human subjects. We use the same measure of boundary recall as in [1,14]. Given a boundary in the ground truth, we search for a boundary in superpixel segmentation within a distance of t pixels. For experiments we set $t = 2$. Recall is the percentage of ground truth boundary that is also present in superpixel segmentation (within a threshold of t). Fig. 6(a) plots the dependency of boundary recall on the number of superpixels. The smaller is the number of superpixels, the less boundaries there are, and the worse is the recall. These results were obtained by averaging over 300 images in the database. We compare our compact (OursCompact) and constant intensity (OursIntConstant) superpixels with turbopixels (Turbo), method from [12] (FH), and NC superpixels (NC) [1]. Our variable patch superpixels have performance similar to compact superpixels, so we omit them from Fig. 6 for clarity. From the plot, it is clear that our constant intensity superpixels have a comparable performance to FH and NC methods, and are superior to turbopixels, at least for lower superpixel number. For high superpixel number, all methods have similar performance. Our constant intensity superpixels are superior to compact superpixels for any number of superpixels. The running time of our algorithm is better than that of Turbopixel and NC algorithms.

In Fig. 6(b) shows the undersegmentation error from [14]. Given a ground truth segment and a superpixel segmentation of an image, undersegmentation

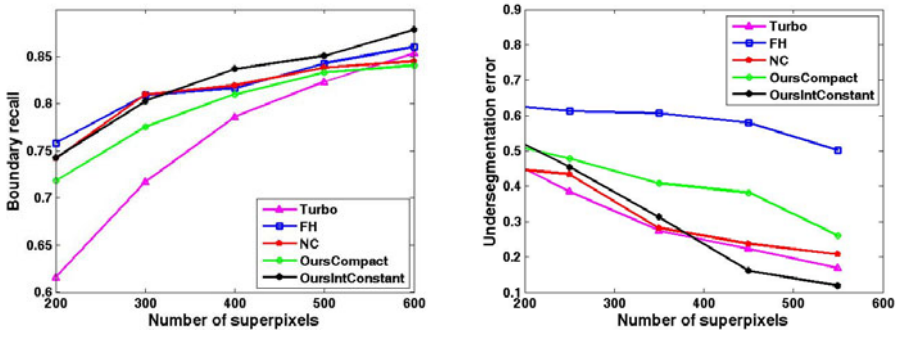


Fig. 6. Performance vs. number of superpixels. Left: boundary recall vs. number of superpixels. Right: undersegmentation error vs. number of superpixels.

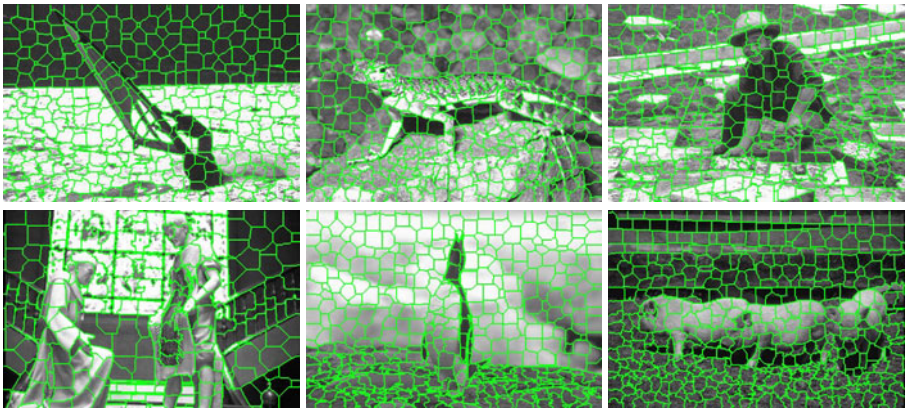


Fig. 7. Top: compact superpixels, bottom: constant intensity superpixels

error measures what fraction of pixels leak across the boundary of a ground truth segment. The FH algorithm [12] is particularly susceptible to this error because it produces segments of highly variable shapes. Our normalization is slightly different from that in [14], so the vertical axis is on a different scale.

The running times of our algorithms for the images in Berkeley dataset are, on average, as follows. The variable patch superpixels take 2.7 seconds to compute. The compact superpixels take from 5.5 to 7.4 seconds to compute, depending on the patch size. A larger patch size corresponds to a slightly longer running time. The constant intensity superpixels take from 9.7 to 12.3 seconds to compute, again depending on the patch size. Patch sizes are from 20 by 20 to 90 by 90. The turbopixel algorithm [14] takes longer to compute, on average 21.3 seconds. The average running time of NC superpixels [1] is 5.7 minutes.

We now compare the dense strategy of using all patches with the sparse patch placement described in Sec. 2.2. In both cases, we run the expansion algorithm for two iterations. Since the dense strategy is expensive, we ran the experiment for 20 images chosen at random from the Berkeley database [15]. To compare energies across different images, we measure the relative energy difference. For an image I, let $E^d(I)$ be the energy with dense patch placement, and $E^s(I)$ be the energy with the sparse patch placement. Then the relative percent difference in energy is $100 \cdot \frac{E^s(I)-E^d(I)}{E^d(I)}$. The mean running time for the dense strategy was 123.5 seconds, whereas for the sparse strategy it was 5.8 seconds. The mean energy difference is 13%, with standard deviation of 0.8%. It makes sense to gain a factor of 21 in computational efficiency while worsening the energy by 12%.

Fig. 1 and Fig. 2 can be used to visually compare our results with turbopixels [14] and NC superpixels [1]. Visually the results are similar, except the NC superpixels appear to have smoother boundaries. This is because in [1] they use a sophisticated boundary detector from [27]. We could incorporate this too in our framework, but it is rather expensive, it takes approximately 30 seconds to

compute boundaries for one image. In Fig. 7, we show some of our segmentations. The top row is compact and the bottom row is intensity constant superpixels.

Fig. 5(b) shows the results on four frames of the "tennis" video sequence. We show the section of supervoxels with each frame separately. Notice the high degree of spatial coherency, even in the areas that are not stationary. Between the first and the last frames shown, the ball moves by about 5 pixels, and the hand by about 8 pixels in the vertical direction. The fingers and the ball are segmented with a high degree of temporal coherency between the frames. We highlight 3 different supervoxels: the one on the ball with red, on the hand with light blue, and on the wall with yellow. The wall is stationary and the supervoxel shape is almost identical between the time slices. The ball and hand are moving, but still the supervoxels slices have a high degree of consistency.

The results of supervoxel segmentation are best to be viewed in a video provided in the supplementary material. We show the original "tennis" sequence and the result of supervoxel segmentation. For visualization, we compute the average intensity of each supervoxel and repaint the video with the average supervoxel intensity. To appreciate the degree of temporal coherence in the supervoxel segmentation, we also perform superpixel segmentation on each frame of the "tennis" sequence separately, using the algorithm in Sec. 2.2. We display the results by painting superpixels with their average intensity. The result of segmentation on each frame separately has much less temporal coherence, as expected. We also provide several other video sequences.

The code for superpixel segmentation will be made available on our web site.

5 Application to Salient Object Segmentation

To show that regular superpixels are useful, we evaluated them for salient object segmentation, similar to [28]. The goal is to learn to segment a salient object(s) in an image. We use Berkeley dataset [15], 200 images for training and 100 for testing. Using human marked boundaries as a guide, we manually select salient object(s). Of course, our ground truth is somewhat subjective.

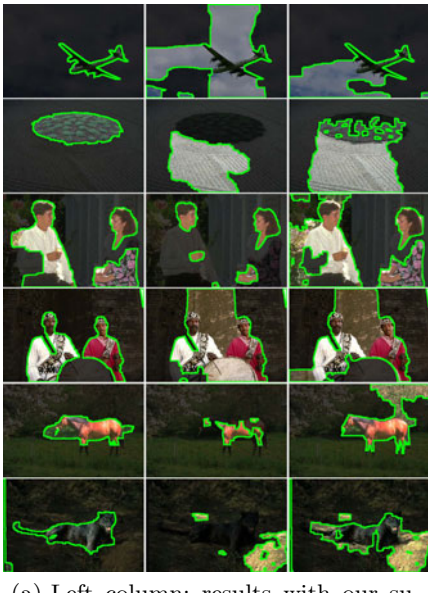
We segment images using rectangular boxes, FH superpixels [12], and our compact superpixels. For boxes, we found that different sizes with overlap give better results. We used 4 different box sizes, from 80 by 80 to 20 by 20. For segmentation, we choose parameters that give the best results on the training data. From each box/superpixel, we extract features similar to those used in [3]. We use features based on color, position (relative to the image size) in the image, and texture. We use gentleboost [29] for training[1].

The testing error is as follows. Our compact superpixels: 20.5%, FH superpixels [12]: 27.4%, rectangular boxes: 24.0%. Thus performance with our superpixels is significantly better than that of boxes and of FH superpixels [12]. With boxes, the size is controlled, but boxes do not align well to object boundaries. With FH superpixels [12], boundaries are reasonable, but segment size is not controlled,

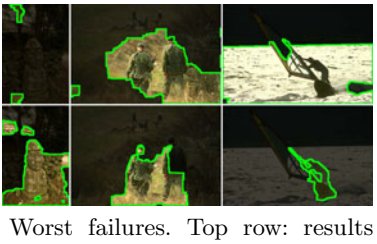
[1] The implementation by A. Vezhnevets downloaded from graphics.cs.msu.ru/ru/science/research/machinelearning/adaboosttoolbox.

some segments are very large. Thus it appears to be important that our compact superpixels have both regularized size and boundary alignment. We expect that we would have gotten performance similar to ours using turbopixels [14] or NC superpixels [1], but our computational time is much better. Our results in this section are consistent with those of [6], who show that having more accurate spatial support (more accurate superpixels) improves object segmentation.

We also investigate whether the results from classification can be further improved by spatial coherence. We apply the binary segmentation algorithm of [24] to separate an image into the salient object and background components. For the data term, we use the confidences provided by boosting. Using confidences only can smooth results, but will not help to rectify large errors. Additional information is gathered from the histogram of pixels with a high confidence either the object or background class. Thus the data term is computed from quantized color histogram weighted by class confidences. After binary graph cut segmentation the errors are as follows. Our compact superpixels: 21.1%, FH superpixels [12] 25.6%, rectangular boxes 28.4%. Interestingly, the results for FH superpixels [12] improve, results for our superpixels slightly worsen, and results for boxes worsen significantly. Fig. 8(a) shows some results after graph cut segmentation. While what exactly constitutes a salient object may be arguable, our results most often



(a) Left column: results with our superpixels, middle column: results with FH superpixels [12], last column: results with boxes.



(b) Worst failures. Top row: results with our superpixels. Bottom row: first result is with boxes, the other two are with FH superpixels [12]

Fig. 8. Some results for salient object segmentation

correspond to recognizable object(s) occupying a significant portion of a scene, with minimal holes. For most images, results with our superpixels are better or comparable than that of boxes and superpixels of [12]. However sometimes there are significant failures, the worst of them are in Fig. 8(b).

6 Future Work

In the future, we plan to investigate more variations on the "basic" energy function to produce superpixels with other interesting properties, such as certain predetermined orientations, etc. We can also use our algorithm to integrate results from different segmentation algorithms, taking advantages or their respective strengths.

Acknowledgements

We would like to thank Kyros Kutulakos for bringing the superpixel problem to our attention and for useful discussions. We are grateful to Lena Gorelick for the video sequences used in the supplementary material. This research was supported, in part, by NSERC-DG, NSERC-DAS, CFI, and ERA grants.

References

1. Ren, X., Malik, J.: Learning a classification model for segmentation. In: ICCV, vol. 1, pp. 10–17 (2003)
2. Mori, G., Ren, X., Efros, A.A., Malik, J.: Recovering human body configurations: combining segmentation and recognition. In: CVPR, vol. 2, pp. 326–333 (2004)
3. Hoiem, D., Efros, A., Hebert, M.: Geometric context from a single image. In: ICCV, pp. 654 – 661 (2005)
4. Mori, G.: Guiding model search using segmentation. In: ICCV, pp. 1417–1423 (2005)
5. He, X., Zemel, R.S., Ray, D.: Learning and incorporating top-down cues in image segmentation. In: Leonardis, A., Bischof, H., Pinz, A. (eds.) ECCV 2006. LNCS, vol. 3951, pp. 338–351. Springer, Heidelberg (2006)
6. Malisiewicz, T., Efros, A.A.: Improving spatial support for objects via multiple segmentations. In: BMVC (2007)
7. Pantofaru, C., Schmid, C., Hebert, M.: Object recognition by integrating multiple image segmentations. In: Forsyth, D., Torr, P., Zisserman, A. (eds.) ECCV 2008, Part III. LNCS, vol. 5304, pp. 481–494. Springer, Heidelberg (2008)
8. Fulkerson, B., Vedaldi, A., Soatto, S.: Class segmentation and object localization with superpixel neighborhoods. In: ICCV (2009)
9. van den Hengel, A., Dick, A., Thormählen, T., Ward, B., Torr, P.H.S.: Videotrace: rapid interactive scene modelling from video. ACM SIGGRAPH 26, 86 (2007)
10. Viola, P., Jones, M.: Rapid object detection using a boosted cascade of simple features. In: CVPR, vol. 1, pp. 511–518 (2001)
11. Comaniciu, D., Meer, P., Member, S.: Mean shift: A robust approach toward feature space analysis. TPAMI 24, 603–619 (2002)
12. Felzenszwalb, P.F., Huttenlocher, D.P.: Efficient graph-based image segmentation. IJCV 59, 167–181 (2004)

13. Shi, J., Malik, J.: Normalized cuts and image segmentation. TPAMI 22, 888–905 (1997)
14. Levinshtein, A., Stere, A., Kutulakos, K., Fleet, D., Dickinson, S., Siddiqi, K.: Fast superpixels using geometric flows. TPAMI 31, 2290–2297 (2009)
15. Martin, D., Fowlkes, C., Tal, D., Malik, J.: A database of human segmented natural images and its application to evaluating segmentation algorithms and measuring ecological statistics. In: ICCV, vol. 2, pp. 416–423 (2001)
16. Kwatra, V., Schödl, A., Essa, I., Turk, G., Bobick, A.: Graphcut textures: Image and video synthesis using graph cuts. ACM SIGGRAPH 22, 277–286 (2003)
17. Osher, S., Sethian, J.A.: Fronts propagating with curvature-dependent speed: Algorithms based on hamilton-Jacobi formulations. Journal of Computational Physics 79, 12–49 (1988)
18. Boykov, Y., Veksler, O., Zabih, R.: Efficient approximate energy minimization via graph cuts. TPAMI 21, 1222–1239 (2001)
19. Szeliski, R., Zabih, R., Scharstein, D., Veksler, O., Kolmogorov, V., Agarwala, A., Tappen, M., Rother, C.: A comparative study of energy minimization methods for markov random fields with smoothness-based priors. TPAMI 30, 1068–1080 (2008)
20. Moore, A., Prince, S., Warrell, J., Mohammed, U., Jones, G.: Superpixel lattices. In: CVPR (2008)
21. Moore, A., Prince, S.J., Warrel, J.: Lattice cut - constructing superpixels using layer constraints. In: CVPR (2010)
22. Boykov, Y., Kolmogorov, V.: An experimental comparison of min-cut/max-flow algorithms for energy minimization in vision. TPAMI 24, 137–148 (2004)
23. Boykov, Y., Kolmogorov, V.: Computing geodesics and minimal surfaces via graph cuts. In: ICCV, pp. 26–33 (2003)
24. Boykov, Y., Funka Lea, G.: Graph cuts and efficient n-d image segmentation. IJCV 70, 109–131 (2006)
25. Truong, B.T., Venkatesh, S.: Video abstraction: A systematic review and classification. ACM SIGGRAPH 3, 3 (2007)
26. Wang, J., Xu, Y., Shum, H., Cohen, M.F.: Video tooning. ACM SIGGRAPH, 574–583 (2004)
27. Martin, D., Fowlkes, C., Malik, J.: Learning to find brightness and texture boundaries in natural images. NIPS (2002)
28. Liu, T., Sun, J., Zheng, N.N., Tang, X., Shum, H.Y.: Learning to detect a salient object (2007)
29. Friedman, J., Hastie, T., Tibshirani, R.: Additive logistic regression: A statistical view of boosting. The Annals of Statistics 38, 337–374 (2000)