SIAM J. IMAGING SCIENCES
Vol. 3, No. 4, pp. 878–903

KDE Paring and a Faster Mean Shift Algorithm*

Daniel Freedman† and Pavel Kisilev‡

Abstract. The kernel density estimate (KDE) is a nonparametric density estimate which has broad application in computer vision and pattern recognition. In particular, the mean shift procedure uses the KDE structure to cluster or segment data, including images and video. The usefulness of these twin techniques—KDE and mean shift—on large data sets is hampered by the large space or description complexity of the KDE, which in turn leads to a large time complexity of the mean shift procedure that is superlinear in the number of points. In this paper, we propose a sampling technique for *KDE paring*, i.e., the construction of a compactly represented KDE with much smaller description complexity. We prove that this technique has good properties in that the pared-down KDE so constructed is close to the original KDE in a precise mathematical sense. We then show how to use this pared-down KDE to devise a considerably faster mean shift algorithm, whose time complexity we analyze formally. Experiments show that image and video segmentation results of the proposed fast mean shift method are similar to those based on the standard mean shift procedure, with the typical speed-up several orders of magnitude for large data sets. Finally, we present an application of the fast mean shift method to the efficient construction of multiscale graph structures for images, which can be used as a preprocessing step for more sophisticated segmentation algorithms.

Key words. kernel density estimate, mean shift, segmentation, sampling

AMS subject classifications. 62H30, 68T45, 68T10, 62H35

DOI. 10.1137/090765158

1. Introduction.

Many of today's real-world applications in computer vision and pattern recognition need to handle and analyze continually increasing amounts of data. Examples of such applications are medical imaging systems, web applications, traffic management, network monitoring, and surveillance systems. A related phenomenon is the continually decreasing cost of collecting data. This is due partially to the fact that various sensors have become cheaper to produce; for example, a 12–15 megapixel point-and-shoot camera under $300 is commonplace. Furthermore, information in general is easier to acquire, due to the large number of public repositories on the web. In this context, the main technological hurdle is that the data can be acquired faster than it can be processed. Data reduction and clustering methods are thus increasingly important, as they allow one to extract the most relevant and important information from large data sets.

A common starting point for data reduction is the construction of a probability density function; as the density describes the distribution of the underlying data, it can be used to extract the data's most salient characteristics. In this paper, we will be most interested in

*Received by the editors July 15, 2009; accepted for publication (in revised form) June 11, 2010; published electronically November 2, 2010. A preliminary version of this paper appeared in [13].
http://www.siam.org/journals/siims/3-4/76515.html
†Corresponding author. Hewlett-Packard Laboratories, Technion City, Haifa 32000, Israel (daniel.freedman@hp.com).
‡Hewlett-Packard Laboratories, Technion City, Haifa 32000, Israel (pavel.kisilev@hp.com).

using the density for the purposes of clustering, but it is also useful in other tasks such as dimension reduction and visualization. There are a variety of methods for estimating a density from data; generally speaking, the most effective techniques are nonparametric [11]. One of the most popular nonparametric density estimates is the kernel density estimate (KDE) [32], and we shall focus on this estimate in this paper. The KDE is a useful structure, as it provides not only an accurate description of the data, but also a framework for clustering via the mean shift procedure [14, 4, 7]. This well-known technique is widely used in many imaging applications such as image and video segmentation [7, 35], denoising [3], object tracking [9], and texture classification [15], inter alia.

1.1. Contributions. A problem arises when trying to use the KDE as a density estimate on large data sets: the KDE, like most nonparametric density estimates, suffers from a high description complexity of roughly $O(n)$, where n is the data set size. Furthermore, this problem is actually worse for the mean shift technique; the time complexity of this technique is superlinear in n (with a naive implementation, it is $O(n^2)$). One of the goals of clustering is to summarize the data so that it is of a manageable size, but the foregoing analysis seems to indicate that it is too time-consuming to actually perform clustering on very large data sets.

This paper presents two main contributions which solve these two problems. The first contribution is a method for *KDE paring*, that is, compression of the KDE. The proposed framework uses a sampling technique to generate a pared-down KDE with much smaller space complexity. We prove a theorem showing that our pared-down density estimate is close to the complete data KDE, to within a given accuracy, in a specific mathematical sense. The second contribution of our work is the fast mean shift clustering procedure which is based on the above pared-down KDE. The time complexity of the proposed clustering method is roughly K times lower compared to the regular mean shift clustering procedure, where K is the subsampling factor.

We show results demonstrating that the proposed fast mean shift clustering algorithm allows one to achieve very accurate results comparable to the results of the standard mean shift procedure using the complete data set, with typical speed-up factors of about 1000. While the fast mean shift technique may be most relevant for many computer vision applications, we also believe that the general KDE reduction technique is quite promising, as KDEs can be used more widely for a variety of data representation tasks in pattern recognition.

Note that an earlier version of this work appeared in [13].

1.2. Relation to prior work. As we have mentioned, one of the main difficulties in applying mean shift–based clustering to big data sets is its computational complexity, which is superlinear in the number of data points. There are several existing techniques which have been developed to increase the speed of mean shift. An early method which sought to speed up mean shift is that of DeMenthon [10], in which the end goal was to segment video. DeMenthon applied ordinary mean shift in a hierarchical fashion; the algorithm consists of stages, where at each stage ordinary (nonaccelerated) mean shift is applied using the cluster centers of the prior stage, with the weight of a cluster given by the fraction of points that contributed to that cluster. Thus, at each stage mean shift applies over a large range. The speed-up comes from the multiscale structure: for large mean shift radii, the technique has access to

an already computed binary tree structure, which makes range searching much more efficient. Unfortunately, it is not entirely clear how the final segmentation one arrives at is related to the original mean shift segmentation.

Yang et al. [38] used the fast Gauss transform to speed up the sum in the mean shift iteration. The fast Gauss transform is a version of the fast multipole method, which accelerates the computation of many series expansions. The method works by expanding each function in the expansion as the sum of several basis functions; the number of such basis functions used is determined by the accuracy with which one represents the problem. If the number of such basis functions is small compared to the number of points in the original expansion, then the fast multipole method speeds up the computation of the series expansions considerably. As mean shift requires the continual evaluation of such expansions (sums of derivatives of Gaussians), the method can be applied. The related method of Guo, Guo, and Lu [18] decomposed the mean shift sum into a number of local subsets.

Paris and Durand [28] introduced an approximate method which works well when the feature vector is low-dimensional. They began by coarsening the data by putting it into a histogram; as the number of dimensions is typically at least three, the dimensions of each bin in the histogram must be sufficiently large, and the number of bins sufficiently small. The regular structure of the resulting histogram can then be exploited, as it can be shown that the mean shift iterations can be expressed as a multidimensional convolution of the histogram with an object related to the kernel. If a Gaussian kernel is used, then this kernel is separable; as a result, the multidimensional convolution becomes d separate one-dimensional convolutions.

Wang et al. [36] used a clever data structure, the dual tree, to speed up mean shift. The idea here is as follows: in computing the mean shift iterations for all points simultaneously, there are many sums to compute, each of which has many terms. These correspond to a series of query points (one for each sum) and reference points (one for each term in the sum). The dual tree method speeds up such computation by computing two separate trees: one for the query points, and one for the reference points. Then the contribution of a node from the reference tree to a node from the query tree is updated by first comparing these two nodes, and then, if needed, comparing the pairs of the children of these nodes; thus, this technique is effectively the simultaneous traversal of two trees. The advantage of the dual tree technique, as compared to the methods of Yang et al. [38] and Paris and Durand [28], is that it maintains a relative error bound of the mean shift iteration at each stage, leading to a more accurate algorithm. The disadvantage is that it can be quite a bit slower.

Finally, we would be remiss not to point out the EDISON software implementation of mean shift [5]. EDISON embodies several heuristics for speeding up mean shift, such as path recycling: several points may have approximately overlapping paths to the mode, so that the overlapping portion need be computed only once. Also somewhat related is the paper by Vedaldi and Soatto on "quick shift" [34].

The proposed method presents several advantages versus the methods described above. First, it gives both a more compact representation of the KDE as well as a fast mean shift technique, as opposed to many of the above techniques which are focused only on accelerating mean shift. If the KDE itself is the end goal, this is an important consideration. Second, the approach is "orthogonal" to the above techniques and can thus be combined with any one of them for even better performance. Third, it is very easy to implement.

It should be noted that in parallel to the limited literature on fast mean shift, several methods have been developed for compact KDE representation, though not for mean shift purposes. This latter literature mainly involves techniques which rely heavily on neural networks and self-organizing maps [33, 37, 17]. Neural networks lead to a high implementation complexity (as well as other issues) which we wish to avoid in this work. There is also the contribution of Goldberger, Greenspan, and Dreyfuss [16], which uses the unscented transform to find a compact mixture model. This technique is formulated in terms of an optimization for which the algorithm finds a local optimum; thus, one may expect all of the traditional problems associated with local optima.

1.3. Paper outline. The remainder of the paper is organized as follows. In section 2 we review facts about KDEs, introduce the problem of KDE paring, i.e., compactly representing a given KDE, and then propose our probabilistic solution to this problem. We state a theorem which shows that the pared-down KDE we construct is close to the true KDE in a prescribed mathematical sense. In section 3 we prove this theorem; the reader may wish to skip this section on first reading the paper and return to it afterwards. In section 4 we show the application of the pared-down KDE to a fast version of mean shift and analyze its computational complexity compared to the standard mean shift. We also propose a soft variant of the fast mean shift algorithm, which is more suitable for smoothing than clustering. In section 5, we demonstrate the performance of the proposed method on the tasks of image and video segmentation and compare results with standard mean shift. In section 6, we present an application of the fast mean shift method for constructing multiscale graph structures for images, a task which is often useful in more sophisticated segmentation schemes. Section 7 is a conclusion.

2. KDE paring.

In this section, we present an algorithm for KDE paring, i.e., for finding a compact representation of a KDE. We begin by reviewing salient facts about KDEs, after which we formalize the problem of compactly representing KDEs in terms of an optimization. We then propose a simple randomized algorithm for tackling this problem, based on sampling from the original KDE. Finally, we state a theorem showing that this pared-down KDE is close in expectation to the original KDE in the L_2 norm.

2.1. A brief review of KDEs. We begin by briefly defining the KDE. Our data is a set of points $\{x_i\}_{i=1}^n$, sometimes referred to as feature vectors, living in a Euclidean space: $x_i \in \mathbb{R}^d$. In computer vision applications, there is generally one such vector per pixel (or voxel in the three-dimensional case); the vector may be color, or color augmented with position, texture, and so on. The KDE of this data is then taken to be

$$f(x) = \frac{1}{n} \sum_{i=1}^{n} K_{\mathbf{H}}(x - x_i).$$

This is an estimate of the probability density underlying the points, where the function $K_{\mathbf{H}}(x - x_i)$ is essentially a bump centered at x_i. More specifically, we take $K_{\mathbf{H}}(z) = |\mathbf{H}|^{-1/2} K(\mathbf{H}^{-1/2} z)$, where the *kernel* K is itself a probability density with zero mean and identity covariance and satisfying $\lim_{\|x\| \to \infty} \|x\|^d K(x) = 0$. Common choices for K include Gaussian, uniform, and (multidimensional) Epanechnikov kernels. In many cases of interest, $\mathbf{H}$ will be diagonal; in general, though, this need not be the case.

2.2. A more compact KDE: The problem. Without explicitly computing the complexity of mean shift clustering (we put off this exercise until section 4.3), we note that the running time will be superlinear in n, the number of data points. If we are concerned with large images coming from today's cameras, n can easily be in the range of 10^7. Worse yet, we may be interested in using mean shift on video or on three-dimensional imagery, such as CT or MRI images; in this case, n of the order of 10^8 or higher is easily conceivable. It is therefore worthwhile to consider a faster mean shift technique. One of the main speed bottlenecks is the description complexity of the KDE, which is $O(n)$; we thus begin by discussing the problem of finding a more compact description for the KDE.

We wish to find a KDE which is close to $f(x)$, but which uses many fewer points to generate it. That is, we wish to solve the following problem:

$$\min_{\{\hat{x}_j\}_{j=1}^m, \hat{\mathbf{H}}} D(\hat{f}(\cdot), f(\cdot)) \quad \text{subject to} \quad \hat{f}(x) = \frac{1}{m}\sum_{j=1}^{m} K_{\hat{\mathbf{H}}}(x - \hat{x}_j),$$

where D is a distance measure between probability densities and m is a fixed number of points, with $m \ll n$. This optimization seeks to find the data points $\hat{x}_j$ underlying a second KDE $\hat{f}$ which well approximates the original KDE f, but uses many fewer points.

There are several natural choices for the distance measure D, ranging from L_p-type distances to more information theoretic measures such as the Kullback–Leibler divergence. In all such cases, the resulting optimization problem is hard to solve globally; this is due to the fact that the $\hat{x}_j$ appear within the kernel K, leading to a nonconvex optimization for which global solutions cannot generally be found efficiently. However, f and $\hat{f}$ are more than functions—they are both densities—and this fact gives us a means of finding an efficient solution to the compact representation problem.

2.3. A more compact KDE via sampling. Our solution is very simple: we choose the samples $\hat{x}_j$ by sampling from the distribution given by $f(\cdot)$. This sampling provides an elegant solution, as it is very simple to implement, and also gives the desired approximation property, albeit in a probabilistic sense. It is important to note that we are *not* suggesting sampling the $\hat{x}_j$ from the discrete set of centers $\{x_i\}_{i=1}^n$ but rather from the entire distribution given by the KDE $f(\cdot)$. The advantage of such a sampling procedure is that it leads to theoretical guarantees of the proximity of the compact KDE and the original KDE, which we shall detail shortly.

Before discussing the logic of this approach, note that such sampling is quite feasible in general, and may be implemented as a three-step procedure. For each $j = 1, \ldots, m$, we construct $\hat{x}_j$ as follows:

1. choose a random integer $r_j \in \{1, \ldots, n\}$;
2. choose a random sample δ_j from $K(\cdot)$;
3. set $\hat{x}_j = x_{r_j} + \mathbf{H}^{1/2}\delta_j$.

It is a simple matter to establish that this procedure samples from the distribution given by $f(\cdot)$. As long as one can sample easily from $K(\cdot)$, as is the case for the Gaussian, uniform, and Epanechnikov kernels, then the procedure is easy to implement.

Now, the main question becomes, If we construct a KDE $\hat{f}$ based on the random samples $\hat{x}_j$, will it be close to the true KDE f? Of course, the KDE $\hat{f}$ itself will be a random variable

(i.e., a random function), and thus any result we prove will be of a probabilistic nature. In fact, we have the following result, which guarantees that f and $\hat{f}$ are close in expectation, in an L_2 sense.

Theorem 2.1. *Let f be a KDE with n points. Then we may construct a KDE $\hat{f}$ with m points in time $O(m)$, with diagonal bandwidth matrix $\hat{\mathbf{H}} = \hat{h}^2\mathbf{I}$, using the above procedure. Let the expected squared L_2 distance between the two densities be given by $J = E[\int (f(x) - \hat{f}(x))^2 dx]$. Then*

$$J \leq 4A\hat{h} + A^2\hat{h}^2V + \frac{B}{m\hat{h}^d} + \frac{ABV}{m\hat{h}^{d-1}}, \tag{2.1}$$

where A, B, V are constants which do not depend on $\hat{h}$ or m.

Proof. See section 3. ∎

The meaning of this theorem is straightforward: the two KDEs f and $\hat{f}$ will be close (in expectation) if m is large enough and if the bandwidth $\hat{h}$ is chosen properly[1] as a function of m. This is precisely what we want in a compact representation: the description complexity of the KDE has been reduced from $O(n)$ to $O(m)$, but we generate close to the same density, in the expected L_2 sense.

One may naturally ask the following question. It may be the case that the densities $f(\cdot)$ and $\hat{f}(\cdot)$ are close in the L_2 sense, but for the purposes of the mean shift algorithm, we are concerned with their modes. Is it the case that the modes of $f(\cdot)$ and $\hat{f}(\cdot)$ are close? Unfortunately, this question is considerably more difficult to answer. The tools which are used to prove the proximity in L_2 are not sufficient to deal with the problem of mode proximity. Indeed, it is the case that two functions could be quite close in *any* norm and could still have vastly different numbers of modes. This will occur, for example, if $\hat{f}(x) = f(x) + \varepsilon \sin(\pi x/\varepsilon)$ for small ε; $\hat{f}$ will have far more modes than f will, even though $\|f - \hat{f}\| \approx \varepsilon$.

Nonetheless, the L_2 result is suggestive: if the two densities f and $\hat{f}$ are close in the L_2 sense, we might reasonably expect—given the way $\hat{f}$ is generated from f through a sampling procedure—that they will have similar numbers and locations of modes. We will establish this notion empirically in section 5, through the experimental results presented there.

Another natural question is, Why sample directly from f rather than from the discrete set of centers $\{x_i\}_{i=1}^n$? The intuition is that we wish to construct a suitable approximation to f, and f contains certain information that cannot be found in the centers themselves. In particular, information about the *bandwidth* h is missing from the set of centers. This issue can be illustrated with the following hypothetical scenario. Imagine two possible bandwidths, h_{low} and h_{high}, with the latter much larger than the former. The KDE f_{low} constructed from h_{low} will be quite different from the KDE f_{high} constructed from h_{high}. As a result, the sets of m samples constructed using our sampling procedure will be quite different for the two cases, as one would wish. By contrast, sampling directly from the discrete set of centers $\{x_i\}_{i=1}^n$ will lead to identical sets of m samples in both cases, as the bandwidth information is not taken into account; this is problematic. In summary, for our fast mean shift algorithm, we wish to approximate f well; sampling directly from f is the most effective way to achieve this.

[1] The question of how precisely to choose the bandwidth is discussed in section 2.4. For the moment, it is sufficient to note that $\hat{h}$ should go to 0 as m goes to ∞, but not too quickly, i.e., in such a way that $m\hat{h}^d \to \infty$ as $m \to \infty$.

A final comment on Theorem 2.1 is in order. Related theorems may be found in the literature on kernel density estimation; see, for example, [11]. Note that although we are comparing two KDEs in Theorem 2.1, our proof relies only on the fact that $\hat{f}$ is a KDE, and f need only be a density. However, the fact that f is a KDE is implicit in the statement of Theorem 2.1 itself, in the following sense. We have claimed to be able to not only construct the reduced KDE $\hat{f}$, but also to do so quickly—in time $O(m)$, where m is the number of samples used to construct $\hat{f}$. This is important practically, as the overall goal is to speed up mean shift, and since the key ingredient in this speeding-up process is the construction of $\hat{f}$, it is key that this construction itself have small time complexity. Now, for a general density f we cannot guarantee fast sampling; our sampling algorithm explicitly relies on the KDE structure of f. Thus, for Theorem 2.1 to hold, and particularly for it to have usefulness within the overall goal of speeding up mean shift, it is important that both f and $\hat{f}$ be KDEs.

2.4. The optimal bandwidth. Let us return to the expression for the L_2 distance between the original KDE f and the reduced KDE $\hat{f}$ as given in (2.1). Suppose we look at the asymptotic case, where m is sufficiently large and $\hat{h}$ is sufficiently small; in this case, the upper bound on J is well approximated by

$$L = 4A\hat{h} + \frac{B}{m\hat{h}^d}.$$

Now we may try to find the bandwidth[2] which minimizes L, taking m fixed; L is convex in $\hat{h}$, and setting its derivative to 0 yields

$$\hat{h}^*(m) = C_1 m^{-\frac{1}{d+1}} \quad \text{and} \quad L^*(m) = C_2 m^{-\frac{1}{d+1}},$$

where C_1 and C_2 are constants.

It is possible to compute C_1 and C_2, and to compute the optimal bandwidth on the basis of the prior equation. Alternatively, one may proceed by imagining the following experiment. Suppose that one creates the reduced KDE $\hat{f}$ by sampling exactly n points from f and constructing the KDE from that sample. Since n is the size of the original KDE f, the reduced KDE in this case is not actually reduced but is the same size as f. Now, suppose that we compute the optimal bandwidth as above, yielding $\hat{h}^*(n)$.

It is straightforward to see that it is *not* necessarily the case[3] that the optimal $\hat{h}^*(n) = h$. Nonetheless, we may proceed as if it were the case that $\hat{h}^*(n) = h$ and thereby avoid computing the constant C_1. For if $\hat{h}^*(n) = h$, then we can solve for C_1 in terms of h and n, yielding

$$\hat{h} = (n/m)^{\frac{1}{d+1}} h. \tag{2.2}$$

Once again, we stress that in general we would not expect $\hat{h}^*(n) = h$ to hold true. In practice h is not actually chosen optimally; it is generally chosen by heuristic or trial and error. Nonetheless, we have used (2.2) to set the bandwidth in our experiments, with good results.

[2]Note that our choice of bandwidth here is a scalar, yielding a diagonal covariance. For work which tries to find an adaptive nondiagonal bandwidth matrix from the data, see [6].

[3]It is interesting to note that if, instead of sampling the $\{\hat{x}_j\}$ from the density f, we had sampled them (without replacement) from the data points $\{x_i\}$ themselves, then we would have indeed had that $\hat{h}^*(n) = h$.

3. Proof of Theorem 2.1.

Lemma 3.1. *Let* $T_1 = \int (E[\hat{f}(x)] - f(x))f(x)dx$ *and* $T_2 = \int (E[\hat{f}^2(x)] - f^2(x))dx$. *Then* $J \le 2|T_1| + |T_2|$.

Proof.

$$\begin{aligned} J &= E\left[\int (f(x) - \hat{f}(x))^2 dx\right] \\ &= \int f^2(x)dx - 2\int E[\hat{f}(x)]f(x)dx + \int \hat{f}^2(x)dx \\ &= \int f^2(x)dx - 2\int E([\hat{f}(x)] - f(x))f(x)dx \\ &\quad - 2\int f^2(x)dx + \int (\hat{f}^2(x) - f^2(x))dx + \int f^2(x)dx \\ &= -2\int (E[\hat{f}(x)] - f(x))f(x)dx + \int (E[\hat{f}^2(x)] - f^2(x))dx \\ &= -2T_1 + T_2, \end{aligned}$$

and the lemma follows. ∎

Lemma 3.2. *Let* $x, y \in \mathbb{R}^d$ *be such that* $\|x - y\| \le \hat{h}$, *and let* $A = \sup_{x\in\mathbb{R}^d} \|\nabla f(x)\|$. *Then* $|f(y) - f(x)| \le A\hat{h}$.

Proof. Define a function $s : [0, 1] \to \mathbb{R}^d$ by $s(t) = (1 - t)x + ty$. Then

$$\begin{aligned} |f(y) - f(x)| &= |f(s(1)) - f(s(0))| \\ &= \left|\int_0^1 \frac{d}{dt} f(s(t))dt\right| \\ &= \left|\int_0^1 \nabla f(s(t)) \cdot s'(t)dt\right| \\ &= \left|\int_0^1 \nabla f(s(t)) \cdot (y - x)dt\right| \\ &\le \int_0^1 |\nabla f(s(t)) \cdot (y - x)|\, dt \\ &\le \int_0^1 \|\nabla f(s(t))\|\|y - x\|dt \\ &\le \int_0^1 A\hat{h}dt \\ &= A\hat{h}, \end{aligned}$$

where we have made use of the Hölder and Cauchy–Schwarz inequalities. ∎

Lemma 3.3. *Let* A *be as in Lemma 3.2. Then* $\left|\int K_{\hat{\mathbf{H}}}(x - y)f(y)dy - f(x)\right| \le A\hat{h}$.

Proof.

$$\left|\int K_{\hat{\mathbf{H}}}(x - y)f(y)dy - f(x)\right| = \left|\int K_{\hat{\mathbf{H}}}(x - y)\,[f(y) - f(x)]\,dy\right|$$

$$\begin{aligned}
&\leq \int K_{\hat{\mathbf{H}}}(x-y)\,|f(y)-f(x)|\,dy \\
&\leq \int K_{\hat{\mathbf{H}}}(x-y)A\hat{h}dy \\
&= A\hat{h},
\end{aligned}$$

where, in the third line, we have made use of the fact that in the support of $K_{\hat{\mathbf{H}}}$ we have that $\|y-x\| \leq \hat{h}$, and therefore Lemma 3.2 applies. In the first and fourth lines, we have used the fact that each kernel $K_{\hat{\mathbf{H}}}$ integrates to 1. ■

Lemma 3.4. *Let T_1 be as in Lemma 3.1. Then $|T_1| \leq A\hat{h}$.*

Proof.

$$\begin{aligned}
T_1 &= \int \left(E[\hat{f}(x)] - f(x)\right) f(x)dx \\
&= \int \left(E\left[\frac{1}{m}\sum_{j=1}^{m} K_{\hat{\mathbf{H}}}(x-\hat{x}_j)\right] - f(x)\right) f(x)dx \\
&= \int \left(\frac{1}{m}\sum_{j=1}^{m}\int K_{\hat{\mathbf{H}}}(x-\hat{x}_j)f(\hat{x}_j)d\hat{x}_j - f(x)\right) f(x)dx \\
&\leq \int \left(\frac{1}{m}\sum_{j=1}^{m}\left|\int K_{\hat{\mathbf{H}}}(x-\hat{x}_j)f(\hat{x}_j)d\hat{x}_j - f(x)\right|\right) f(x)dx \\
&\leq \int \left(\frac{1}{m}\sum_{j=1}^{m} A\hat{h}\right) f(x)dx \\
&= A\hat{h},
\end{aligned}$$

where, in the fifth line, we have used the result of Lemma 3.3 and, in the sixth line, we have used the fact that f integrates to 1. ■

Lemma 3.5. *Let A be as in Lemma 3.2. Then*

$$\left|\iint K_{\hat{\mathbf{H}}}(x-y)K_{\hat{\mathbf{H}}}(x-z)f(y)f(z)dydz - f^2(x)\right| \leq 2A\hat{h}f(x) + A^2\hat{h}^2.$$

Proof.

$$\begin{aligned}
&\left|\iint K_{\hat{\mathbf{H}}}(x-y)K_{\hat{\mathbf{H}}}(x-z)f(y)f(z)dydz - f^2(x)\right| \\
&= \left|\left(\int K_{\hat{\mathbf{H}}}(x-y)f(y)dy\right)\left(\int K_{\hat{\mathbf{H}}}(x-z)f(z)dz\right) - f^2(x)\right| \\
&= \left|\left(\int K_{\hat{\mathbf{H}}}(x-y)f(y)dy\right)^2 - f^2(x)\right| \\
&= \left|\left(\int K_{\hat{\mathbf{H}}}(x-y)f(y)dy - f(x)\right)\left(\int K_{\hat{\mathbf{H}}}(x-y)f(y)dy + f(x)\right)\right|.
\end{aligned}$$

Now, apply Lemma 3.3 twice. The first term in the product is simply upper bounded by $A\hat{h}$; for the second term, note that Lemma 3.3 implies that $\int K_{\hat{\mathbf{H}}}(x-y)f(y)dy \le f(x)+A\hat{h}$, so that the second term is bounded by $2f(x)+A\hat{h}$. The lemma follows. ∎

Lemma 3.6. *Let A be as in Lemma 3.2, and let $B=\int K^2(z)dz$. Then $\int K^2_{\hat{\mathbf{H}}}(x-y)f(y)dy \le B(f(x)+A\hat{h})\hat{h}^{-d}$.*

Proof. From Lemma 3.3, we have that within the integrand's support, $|f(y)-f(x)|\le A\hat{h}$, so that $f(y)\le f(x)+A\hat{h}$. Also, recall that $K_{\hat{\mathbf{H}}}(z)=|\hat{\mathbf{H}}|^{-1/2}K(\hat{\mathbf{H}}^{-1/2}z)$. Then

$$\begin{aligned}
\int K^2_{\hat{\mathbf{H}}}(x-y)f(y)dy &\le \int K^2_{\hat{\mathbf{H}}}(x-y)(f(x)+A\hat{h})dy \\
&= (f(x)+A\hat{h})\int K^2_{\hat{\mathbf{H}}}(z)dz \\
&= (f(x)+A\hat{h})|\hat{\mathbf{H}}|^{-1/2}\int |\hat{\mathbf{H}}|^{-1/2}K^2(\hat{\mathbf{H}}^{-1/2}z)dz \\
&= (f(x)+A\hat{h})|\hat{\mathbf{H}}|^{-1/2}\int K^2(w)dw \\
&= B(f(x)+A\hat{h})\hat{h}^{-d},
\end{aligned}$$

where, in the fifth line, we have used a change of variables, noting that the Jacobian is exactly accounted for by the $|\hat{\mathbf{H}}|^{-1/2}$ term. ∎

Lemma 3.7. *Let T_2 be as in Lemma 3.1, and let V be the volume of the support of f. Then $|T_2|\le 2A\hat{h}+A^2\hat{h}^2V+\frac{B}{m\hat{h}^d}+\frac{ABV}{m\hat{h}^{d-1}}$.*

Proof. Since $\hat{f}(x)=\frac{1}{m}\sum_{i=1}^m K_{\hat{\mathbf{H}}}(x-\hat{x}_i)$, then $\hat{f}^2(x)=\frac{1}{m^2}\sum_{i\ne j}K_{\hat{\mathbf{H}}}(x-\hat{x}_i)K_{\hat{\mathbf{H}}}(x-\hat{x}_j)+\frac{1}{m^2}\sum_i K^2_{\hat{\mathbf{H}}}(x-\hat{x}_i)$. Then, dropping the $\hat{\mathbf{H}}$ subscript for convenience, T_2 can be simplified as follows:

$$\begin{aligned}
&\int \left(E\left[\hat{f}^2(x)\right]-f^2(x)\right)dx \\
&= \int\left(E\left[\frac{1}{m^2}\sum_{i\ne j}K(x-\hat{x}_i)K(x-\hat{x}_j)+\frac{1}{m^2}\sum_i K^2(x-\hat{x}_i)\right]-f^2(x)\right)dx \\
&= \int\left(\frac{1}{m^2}\sum_{i\ne j}\iint K(x-\hat{x}_i)K(x-\hat{x}_j)f(\hat{x}_i)f(\hat{x}_j)d\hat{x}_i d\hat{x}_j\right. \\
&\qquad \left.+\frac{1}{m^2}\sum_i\int K^2(x-\hat{x}_i)f(\hat{x}_i)d\hat{x}_i-\left(\frac{m-1}{m}\right)f^2(x)-\left(\frac{1}{m}\right)f^2(x)\right)dx \\
&= \int\left(\frac{1}{m^2}\sum_{i\ne j}\left[\iint K(x-\hat{x}_i)K(x-\hat{x}_j)f(\hat{x}_i)f(\hat{x}_j)d\hat{x}_i d\hat{x}_j-f^2(x)\right]\right. \\
&\qquad \left.+\frac{1}{m^2}\sum_i\int K^2(x-\hat{x}_i)f(\hat{x}_i)d\hat{x}_i-\left(\frac{1}{m}\right)f^2(x)\right)dx \\
&\le \int\left(\left(\frac{m-1}{m}\right)(2A\hat{h}f(x)+A^2\hat{h}^2)+\left(\frac{1}{m}\right)B(f(x)+A\hat{h})\hat{h}^{-d}\right)dx
\end{aligned}$$

$$= 2A\hat{h} + A^2\hat{h}^2V + \frac{B}{m\hat{h}^d} + \frac{ABV}{m\hat{h}^{d-1}},$$

where, in the fifth line, we have made use of Lemmas 3.5 and 3.6 and have thrown away the $-(1/m)f^2(x)$ term and, in the sixth line, we have upper bounded the $(m-1)/m$ prefactor with 1. ∎

Proof of Theorem 2.1. Combining the results of Lemmas 3.1, 3.4, and 3.7 gives us the result. ∎

4. Fast mean shift. In this section, we present the fast mean shift algorithm. After a brief review of the standard mean shift algorithm, we propose a fast technique for mean shift based on the pared-down KDE of section 2. We then show that the theoretical complexity of the proposed technique is considerably lower than that of the standard mean shift for a variety of different nearest neighbor data structures. We conclude this section with an examination of a soft variant of the fast mean shift algorithm, which may be useful for image smoothing.

4.1. Review of mean shift. In this section, we review the ordinary mean shift procedure. Given a KDE, the mean shift algorithm is essentially a hill-climbing algorithm; that is, starting at any point x, the mean shift algorithm is an efficient iteration scheme which brings x up the hill of the KDE, finally stopping at the local maximum (or mode) of the KDE in whose basin of attraction x lies. To specify the mean shift iteration formally, let us simplify the form of the kernel and take the radially symmetric form $K(x) = ck(\|x\|^2)$, where k is a one-dimensional *profile* and c is a normalization; for example, for the Gaussian distribution, the profile takes the form $k(z) = e^{-z}$. Denoting $g = k'$, the mean shift iteration is then given by

$$x \leftarrow \frac{\sum_{i=1}^n x_i g\left(\left\|\frac{x-x_i}{h}\right\|^2\right)}{\sum_{i=1}^n g\left(\left\|\frac{x-x_i}{h}\right\|^2\right)} \equiv M(x). \tag{4.1}$$

Iterating an infinite number of times is guaranteed to bring x to the mode in whose basin of attraction it lies. The advantage of this procedure over ordinary gradient ascent is the lack of need to set a time-step parameter (note that no such parameter is present in the above expression); practically, the mean shift tends to converge in a very small number of steps, typically around five.

In order to use the mean shift algorithm for segmentation or clustering on the set $\{x_i\}_{i=1}^n$, one may run the iterations starting at each data point. Denoting $M^1(x) = M(x)$, $M^2(x) = M(M(x))$, and so on, we map each point x_i to $M^\infty(x_i)$ (though recall that, typically, $M^5(x_i)$ will do the job). Since there are a much smaller number of modes than there are points, this is a form of segmentation or clustering, which works very well in practice in a number of applications [35, 8, 21].

4.2. Fast mean shift. How can we incorporate our more compact KDE into the mean shift clustering algorithm? We use the following three-step procedure:

1. *Sampling.* Take m samples of the density f to yield $\{\hat{x}_j\}_{j=1}^m$. Form the new density $\hat{f}(x) = \sum_{j=1}^m K_{\hat{h}}(x, \hat{x}_j)$.
2. *Mean shift.* Perform mean shift on each of the m samples: $\hat{x}_j \to \hat{M}^\infty(\hat{x}_j)$. Here, $\hat{M}$ indicates that we use $\hat{f}$ (rather than f) for the mean shift.
3. *Map backwards.* For each x_i, find the closest new sample $\hat{x}_{j^*}$. Then $x_i \to \hat{M}^\infty(\hat{x}_{j^*})$.

In step 1, we construct the reduced KDE, and in step 2, we perform mean shift on this smaller KDE. Given these modes in the reduced KDE, the question is how to map backwards to the original data. We deal with this in step 3 by mapping from each point in the original data (x_i) to the closest point in the reduced data ($\hat{x}_{j^*}$), and from there to the mode to which that point flows ($\hat{M}^\infty(\hat{x}_{j^*})$).

The key speed-up occurs in the second step; instead of using all n samples to compute the mean shift, we can use the reduced set of m samples. In the next section, we will quantify the precise theoretical speed-up that this entails; for the moment, note that in a naive implementation it can lead to a speed-up of n/m, which can in practice be greater than 1000.

Note that there is a variant to this algorithm in which in step 2 each of n points undergoes mean shift using the reduced KDE, and then step 3 is eliminated. This latter algorithm is more well-founded theoretically, as it eliminates step 3, which is a kind of heuristic. However, it can be shown to have a time complexity which is slower by a factor of ξ, where ξ is the average number of mean shift iterations until convergence. While ξ is generally small compared to m and n, it is still in the range of 5–10, leading to a nonnegligible difference between the two algorithms.

4.3. Complexity analysis. The key aspect in the computation of complexity is the speed of the nearest neighbor search. Note that in each mean shift iteration (see (4.1)) one must compute the nearest neighbors out of the n samples x_i to the point in question, x. If the kernel has finite support, we need to compute exactly those neighbors that lie within the support, that is, with $\|x - x_i\| \leq h$. If the kernel has infinite support, such as in the case of a Gaussian kernel, we simply truncate the Gaussian to some desired tolerance; e.g., we look for neighbors x_i with $\|x - x_i\| \leq \gamma h$, with γ chosen by the user (e.g., $\gamma \approx 2$–3).

Suppose that we have a data structure which permits proximity queries in time $q(n)$ and requires a preprocessing time of $p(n)$ to compute; we will look at examples of such structures shortly, but for now, we will leave them as general. In this case, each mean shift iteration requires $O(q(n))$ time to compute, and assuming, as is the case in practice, that $O(1)$ iterations are required for convergence, then the cost of running the algorithm on all n data points (i.e., the overall data reduction algorithm) is

$$T_{orig}(n) = O(p(n) + nq(n)).$$

How much faster is our proposed algorithm? Looking back at section 4.2, we may break down the complexity for each step. Step 1, the sampling step, is $O(m)$. We have already computed the complexity of step 2, the mean shift step; this is simply the above expression, but specified for m rather than n samples, i.e., $O(p(m) + mq(m))$. In step 3, we must map backwards; that is, for each of the n original samples, we must find the closest among the m new samples. Using our data structure (whose preprocessing time we have already accounted for in step 2), this requires $O(nq(m))$. The total is then

$$\begin{aligned} T_{reduce}(n, m) &= O(m + p(m) + mq(m) + nq(m)) \\ &= O(p(m) + nq(m)). \end{aligned}$$

Comparing this with the expression for $T_{orig}(n)$, and since $n = \Omega(m)$, we have clearly reduced the complexity. By how much depends on the precise details of the data structure used, and we now attempt to deal with this issue.

In the simplest case, we have no special data structure. Then $p(n) = 0$, and the query time is linear in the number of elements $q(n) = O(n)$. In this case, $T_{orig}(n) = O(n^2)$, while $T_{reduce}(m, n) = O(nm)$, so the speed-up is a factor of n/m. In practical cases of interest, $n/m > 100$, so this is quite an impressive speed-up.

Now we may use more complex data structures for search. Voronoi diagrams are not very useful when the dimension in which the data lives is $d > 3$ or so, as the preprocessing time is $p(n) = O(n^{\lceil d/2 \rceil})$. Other popular data structures, such as kd-trees, also have space complexity exponential in d [2]. There are a variety of approximate nearest-neighbor algorithms, such as those based on locality sensitive hashing [2], which can lead to more efficient searches if we are willing to find neighbors which are close but not the closest. Indeed, if we choose an approximation factor of $c \geq 1$, that is, if we find points whose distance from the query is within a factor of c of the closest, then we will have $T_{orig}(n) = O(n^{1+1/c^2})$ and $T_{reduce}(n, m) = O(nm^{1/c^2})$, leading to a speed-up of $O((n/m)^{1/c^2})$; see, for example, [2]. Typically, we might take $c = 1.3$, leading to a speed-up of about 100 if $n/m \approx 1000$. (Note, however, that such data structures often have excellent theoretical properties while being somewhat more difficult to implement in practice; see [2].)

4.4. Variant: Soft segmentation or cartoons. We propose the following variant to the fast mean shift clustering algorithm, which leads effectively to a soft segmentation. Note that only the third step has changed; we reproduce the first two for clarity.

1. *Sampling.* Take m samples of the density f to yield $\{\hat{x}_j\}_{j=1}^m$. Form the new density $\hat{f}(x) = \sum_{j=1}^m K_{\hat{h}}(x, \hat{x}_j)$.
2. *Mean shift.* Perform mean shift on each of the m samples: $\hat{x}_j \to \hat{M}^\infty(\hat{x}_j)$. Here, $\hat{M}$ indicates that we use $\hat{f}$ (rather than f) for the mean shift.
3. *Weighted map backwards.* For each x_i and each $\hat{x}_j$, compute a weight between them according to $w_{ij} \propto K_h(x_i, x_j)$, such that the weights sum to 1: $\sum_j w_{ij} = 1$. Then $x_i \to \sum_{j=1}^m w_{ij} \hat{M}^\infty(\hat{x}_j)$.

Such a procedure will yield, instead of a piecewise constant segmentation, a piecewise smooth or cartoon-like segmentation.[4] This is similar in spirit to the Mumford–Shah scheme [26], which seeks a piecewise smooth approximation to the original image. Note that it may also be useful for smoothing images. We show examples of both soft and hard segmentation in section 5.

5. Segmentation experiments.

5.1. Clustering on a synthetic data set. We begin by demonstrating the performance of the proposed technique on a synthetic data set consisting of two-dimensional data samples from a mixture of three Gaussian distributions; see Figure 1. In this example, 10^5 samples were generated in 100 independent trials. We compare the clustering error (the misclassification rate) and the speed-up factor of the proposed sampled mean shift, with subsampling factors of 10, 100, and 1000, versus the standard mean shift. Both the clustering error and the speed-up factor were averaged over 100 trials. The results of this experiment are summarized in

[4]Note that if the kernel has finite support, then for a particular choice of h it is possible that all weights will be 0. Thus, soft segmentation using a kernel with infinite support, such as a Gaussian, is recommended.

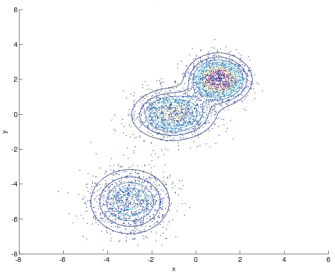

Figure 1. *The data set for the synthetic clustering example.*

Table 1

Comparative results of clustering using the standard mean shift ($S = 1$) *versus the proposed sampled mean shift with subsampling factors of* $S = 10$, 100, *and* 1000.

	$S = 1$	$S = 10$	$S = 100$	$S = 1000$
Speed-up factor	1	15	162	1540
Mean error (percent)	1.59	1.31	1.40	1.53

Table 2

Speed-up factors for image segmentation.

Image	Image size	Sampling m/n	Speed-up factor
Parrot	844×636	1024	401
Castle	960×784	1024	1160
Girl	551×735	1024	1543
Beach	562×602	64	45
House	271×396	64	185
Field	516×542	64	244

Table 1. Notice that two of the clusters have quite an overlap between them; this explains the fact that the misclassification error for the standard mean shift is not lower than for the proposed sampled mean shift.

5.2. Image segmentation. In the image segmentation experiment, we compare the performance of the proposed fast mean shift method with the standard mean shift on six images. The feature vectors are taken to be five-dimensional and are constructed from the three L*a*b* color vectors of the individual pixels in the image and of the two corresponding spatial coordinates (x and y). Table 2 summarizes the timing information, and four of the six images[5] are shown in Figure 2.

In the first three examples, the sampling factor was taken to be $n/m = 1024$. Our complexity analysis indicates that we ought to expect a speed-up factor of about 1024; in fact, despite using fast nearest-neighbor queries (based on kd-trees, which despite the arguments in [2] tend to work well in low dimensions), the speed-up factor in two of the three cases exceeded this sampling factor, as shown in Table 2. In the latter three examples, the sampling factor was taken to be $n/m = 64$. In this case, we expect a speed-up factor of about 64; again, the speed-up factor in two of the three cases exceeded this sampling factor, as shown in

[5]Two of the images are not displayed due to copyright issues.

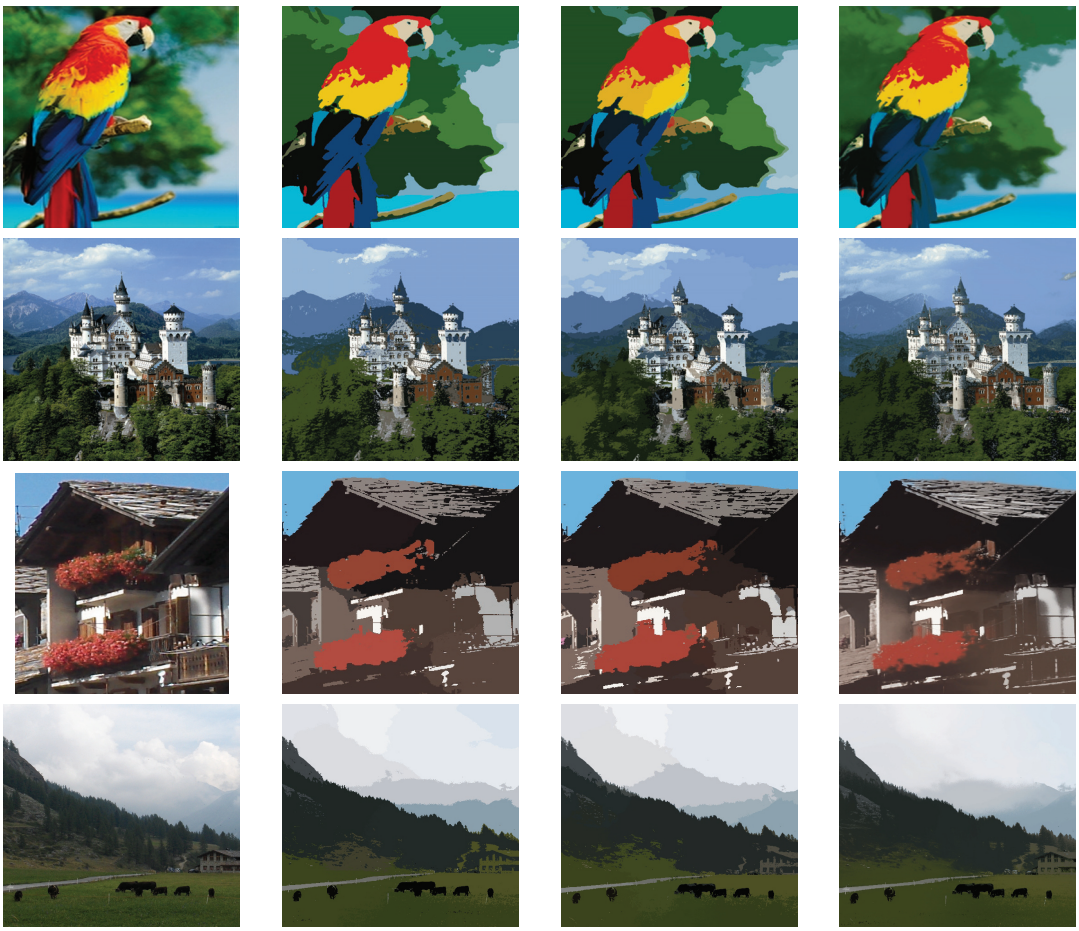

Figure 2. *Image segmentation example:* (a) *the original image,* (b) *the result of standard mean shift segmentation,* (c) *the result of fast mean shift segmentation, and* (d) *the soft segmentation variant of the fast mean shift. Note similar hard segmentation results in* (b) *and* (c).

Table 2. Note that the spread in speed-up factors can be attributed in part to differences in the underlying KDEs, which in turn affects the exact number of mean shift iterations required for convergence. In our theoretical analysis we simply took this to be $O(1)$, but it may vary somewhat from example to example.

In Figure 2, we show the results of the standard mean shift procedure (second column), the fast mean shift procedure (third column), and the soft segmentation variant of the fast mean shift procedure (fourth column). Clearly, hard segmentation results obtained using the original and the fast mean shift methods look quite similar. As expected, the soft segmentation is a kind of piecewise smooth cartoon of the original image, with small details removed from the original image. Therefore it may be suitable for noise reduction or smoothing tasks.

5.3. A comparison of absolute running times. We now compare the absolute running time of our algorithm to another widely used, fast segmentation scheme: the graph-based

Table 3

Timing results for image segmentation. Both algorithms were run on 10 images, each of size 7 megapixels.

Method	Mean running time	Standard deviation of running times
Proposed method	5.67s	0.12s
Felzenszwalb and Huttenlocher [12]	148.98s	5.66s

algorithm of Felzenszwalb and Huttenlocher [12]. We ran the two algorithms on 10 images, each of size 7 megapixels, culled from Flickr. Images of this size have become common, and it is instructive to see how well the proposed algorithm performs on such images, as well as how it compares with the method of [12]. For the latter algorithm, we have used the code which is publicly available at http://people.cs.uchicago.edu/~pff/segment/.

Both methods were run on the same machine, an Intel Core 2 Duo CPU, with clock speed of 2.53 GHz and 4 GB of RAM, running Windows Vista. For both methods, the measured running times do not include loading the input image or writing the output image. For the proposed algorithm, we used a subsampling factor of $n/m = 6400$, which led to good quality results. For the method of [12], we used the following parameter values: the smoothing parameter σ was chosen as 0.5; k, the value for the threshold function, was set to 500; and the minimum component size enforced by postprocessing was taken to be 20. The running time statistics are contained in Table 3, and examples of three of the images[6] and their respective segmentations are shown in Figure 3. Our method is clearly faster[7] than the method of [12] and, with a mean time of 5.67 seconds, should be practical for many applications involving large images.

We also compare our method with another fast mean shift method, that of Paris and Durand [28]. In this case, we simply report the timing results mentioned in their paper [28] for an 8 megapixel image.[8] They report timing results of 29.3 seconds for their core algorithm, and 14.8 seconds for a variant with an "on-the-fly" segmentation hierarchy. The time can be further reduced to just over 3 seconds if the feature space is reduced from five to four dimensions via principal components analysis, at a loss of some accuracy. While it is hard to precisely compare our timing results with those of Paris and Durand, it should be noted that they are within the same ballpark.

5.4. Varying the sampling factor n/m. In order to understand the effect that varying the sampling factor has on segmentation, we ran the fast mean shift algorithm with various sampling factors n/m on the parrot image from the previous section. The images are shown in Figure 4. Note that the algorithm does relatively well even up to a sampling factor of

[6]We acknowledge Flickr users "Tambako the Jaguar" (parrot image), "r3tta" (bottle tree image), and "Rita Willaert" (face image).

[7]It should be noted that the mean time we report for the method of [12] is considerably higher than that reported in [28], in which a time of about 18s is reported for an 8 megapixel image. Two comments are in order. First, we simply downloaded the publicly available code, compiled it, and ran it without modification; the mean running time of 148.98s is what we recorded. Second, even with the faster time of 18s reported in [28], our algorithm is still faster by more than a factor of 3, although the image used in [28] is slightly larger (8 versus 7 megapixels) and the algorithms have been run on two different machines (but with similar clock speeds).

[8]Note that their experiments were run on a machine with a comparable clock speed of 2.6GHz, but with an AMD Opteron chip.

(a) (b) (c)

Figure 3. *Results on large (7 megapixel) images.* (a) *Original image.* (b) *Segmentation using the proposed algorithm.* (c) *Segmentation using the graph-based method* [12]*; note that the colors in this image are meant only to distinguish segments.*

(a) (b) (c) (d) (e)

Figure 4. *Effect of varying the sampling factor* n/m. (a) $n/m = 64$, (b) $n/m = 256$, (c) $n/m = 1024$, (d) $n/m = 4096$, *and* (e) $n/m = 16{,}384$.

$n/m = 4096$, as shown in (d); as the image itself contains a bit more than 5×10^5 pixels, this corresponds to using only $m = 131$ randomly selected samples to construct the KDE, which is quite remarkable. (Note that at this level of sampling, it is possible—though not certain—to get *more* segments in the segmentation. This is due to the fact that with so few samples, there is a greater chance that kernels centered at the samples do not overlap, leaving "empty space" in the feature space. This is the case in Figure 4(d).) The fast algorithm fails at $n/m = 16{,}384$; this is not surprising, as this corresponds to using only 32 randomly selected samples to construct the KDE. Indeed, at this high a subsampling rate, the algorithm produces quite different segmentations from one run to the next.

5.5. Quantifying the performance of fast mean shift. In the following experiment we used 121 images from the Berkeley Segmentation Data Set [25] to quantify similarities between the original and fast mean shift segmentations, with a factor of $n/m = 64$. We calculated two measures: the Rand index (RI) [29], and the global consistency error (GCE) [25]. The RI is a nonparametric measure that works by counting pairs of pixels having compatible label relationships in the two segmentations; it is a measure of *similarity*. The GCE belongs to the region differencing–type measures that compute the overlap of the cluster associated with each pixel in one segmentation and the corresponding (closest) cluster in the other segmentation; it is a measure of *dissimilarity*. The GCE quantifies the consistency between image segmentations of different granularities. Both measures are normalized to be in the range $[0, 1]$. Since the RI is a measure of similarity, the closer the value of RI is to 1, the better is the match between two segmentations. By contrast, GCE measures the dissimilarity between two segmentations; therefore a value closer to 0 corresponds to a better match between segmentations.

The results of this experiment are summarized in the histograms in Figure 5. These results show quantitatively that the original mean shift and the proposed fast mean shift yield quite similar segmentation results: most of the RI values are clustered above 0.85, while most of the GCE values are clustered below 0.25. Out of the 121 images used, we show two example images with their segmentations in Figure 6. The top row shows a typical image, for which the original and the fast mean shift segmentations are close: RI = 0.91 and GCE = 0.12. The bottom row shows one of the worst cases from the point of view of the similarity of the two segmentations; here RI = 0.55 and GCE = 0.15. Clearly, this is a very difficult segmentation example, and both methods fail to segment the image properly. A low value of RI in this

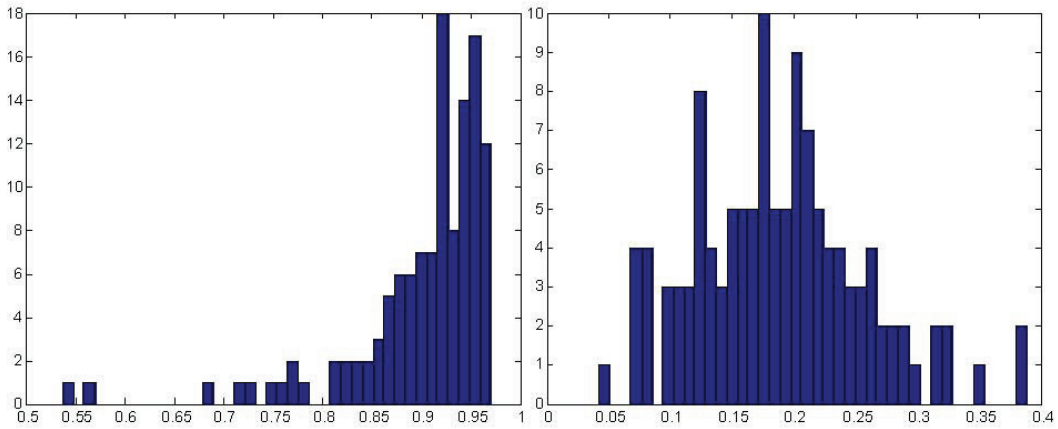

Figure 5. *Histograms of the two performance measures, evaluated on* 121 *images from the Berkeley Segmentation Data Set. Left: Rand index (RI), which measures similarity (high values are better). Right: Global consistency error (GCE), which measures dissimilarity (low values are better).*

Figure 6. *Examples from the Berkeley Segmentation Data Set. Left: Original images. Middle: Original mean shift segmentation. Right: Fast mean shift segmentation. The top row shows a typical example, with* $RI = 0.91$ *and* $GCE = 0.12$. *The bottom row shows one of the worst examples, with* $RI = 0.55$ *and* $GCE = 0.15$.

case can be explained by the differences in the segmentation boundaries. Since the image is quite homogeneous, even small variations in the algorithm parameters can lead to significant differences in segmentations.

Finally, it is worth noting that different random samplings in the fast mean shift algorithm lead to results that are nearly identical to those shown in Figure 5.

5.6. Video segmentation. Figure 7 shows results of video segmentation for two consecutive frames from a video sequence. In this experiment we compared the use of our fast mean shift method in two configurations. In the first (the middle row), a frame-by-frame segmentation was performed. In the second configuration, data from a sliding time window

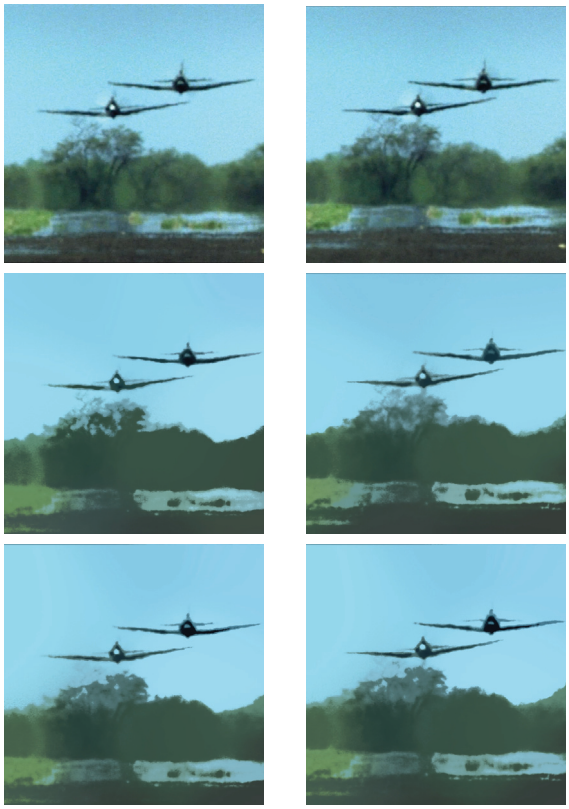

Figure 7. *Video segmentation example. Top row: Two original frames from a video sequence. Middle row: The result of the frame-by-frame segmentation using the proposed method. Bottom row: The result of the sliding* 10*-frame window-based segmentation using the proposed method.*

of 10 frames[9] was combined into a single large data set and used to perform the first two steps (the sampling and the mean shift) in the algorithm of section 4.2, using a subsampling factor of 1024. (Note that in this example, it is possible to use a subsampling factor of 5000, though the results are slightly less clean.) Then, the third step (the mapping backwards) from section 4.2 was applied to each frame. In both cases, the feature vector consists of the three L*a*b* color vectors of the individual pixels in the image, and of the two corresponding spatial coordinates (x and y).[10] The results of the "windowed" version look cleaner; for example, this version does not misclassify the color of the aircraft, as is the case in the frame-by-frame procedure. Note that one might also augment the feature vector in the batch case to include the time axis as an additional feature axis; this might be useful for applications such as target tracking. Clearly, the proposed fast method opens up some new possibilities for dealing with large spatiotemporal data sets.

[9]Note that the video sequence itself is longer than 10 frames; however, for any given mean shift segmentation, a sliding window of 10 frames was used.

[10]It would be natural to add a sixth coordinate to the feature vector, namely, the time t. We have found that such an addition, while natural, does not improve the resulting segmentation much, if at all.

6. Application: Multiscale graph structures. In this section, we show how to use the fast mean shift algorithm to construct a multiscale graph structure corresponding to a particular image. Having a multiscale structure on image graphs can potentially be very useful, due to the number of computer vision problems which can be posed as graph problems and solved using graph algorithms such as graph cuts [23]. Examples of these problems include stereo, semiautomatic segmentation, and optical flow, as well as many problems that can be posed as maximum a posteriori estimation over Markov random fields. A multiscale graph structure can be used to help speed up the solution to these problems in the usual multiscale way: the problem is first solved on the coarsest scale, the solution to this problem is then used to initialize the next scale up, and so on. In addition to making the solution faster, such an approach can also lead in some instances to more accurate solutions. See [1] for an example of another sort of multiscale graph structure, based on algebraic multigrid techniques.

Moreover, multiscale graph structures, in particular of the type to be described here, have gained much currency recently as preprocessing for more sophisticated segmentation or image labeling algorithms. Current algorithms such as [30, 39, 22, 20] use several different "oversegmentations" (often generated by mean shift) at different scales; the idea is that using a multiscale approach provides more information, and hence aids in the image labeling problem. Thus the fast construction of the sort of structure we discuss below is potentially quite useful.

Note that the graph structure that we propose is related to the multiscale clustering described by Leung, Zhang, and Xu in [24]. Our motivation here is not so much the definition of the graph structure (though we do provide a characterization of the structure in Theorem 6.1 that was not explicit in [24]) but a discussion of fast computation of the graph structure via the fast mean shift algorithm. Note also that the notions of the underlying graph structure, and, in particular, the weights, are not discussed in [24].

While the multiscale graph structure proposed here could be used in a variety of algorithms, as we have noted above, we do not pursue this line of inquiry here. Rather, the particular graph construction is described mainly for its illustrative value.

6.1. A multiscale graph structure. We define a continuous multiscale graph structure using mean shift as follows. As before, our data consists of the set of feature vectors $\{x_i\}_{i=1}^n$, where, in most applications of interest, each data point corresponds to a pixel in an image. Denote by $f_h(x)$ the KDE with bandwidth matrix $\mathbf{H} = h^2\mathbf{I}$, and if x is a mode of f_h, let $B(x)$ be the closure of the basin of attraction of x, i.e., $B(x) = Cl(\{y \in \mathbb{R}^d : M^\infty(y) = x\})$. The graph corresponding to bandwidth h is denoted by $G_h = (V_h, E_h, W_h)$, where

$$V_h = \{x \in \mathbb{R}^d : x \text{ is a mode of } f_h(\cdot)\},$$
$$E_h = \{(u, v) : u, v \in V(h), B(u) \cap B(v) \neq \emptyset\},$$

and the edge weights are given by

$$W_h(u, v) = \int_{x \in B(u)} \int_{y \in B(v)} K_h(x, y) dx dy.$$

The weights may be used, for example, if one wishes to apply a spectral-based technique such as normalized cuts [31] as part of a more sophisticated segmentation scheme.

The definitions of the vertex set and edge set are quite natural in terms of mean shift: the vertex set is just the set of modes, and the edge set is the set of pairs of modes whose basins of attraction are adjacent in the feature space. The edge weights are also defined in a reasonable fashion by aggregating the similarity between points in neighboring basins of attraction, though many other definitions are possible as well. In practice, we will approximate the integral in the edge weight definition by its corresponding sum.

This continuous multiscale graph structure has the following desirable properties: we move from a graph whose vertex set comprises the original pixels to a graph with a single vertex. This is very much what we would expect from a multiscale structure. Alternatively, we may think of this property in terms of mean shift segmentations at different bandwidths, without regard to the graph structure. In this case, the property says that the mean shift segmentations obtained for $h = 0$ and $h \to \infty$ have, respectively, one segment per pixel and one segment for the whole image. Formally, we have the following theorem.

Theorem 6.1. *Given the multiscale graph structure as defined above, the graph corresponding to the limit as $h \to 0$ has the property that $V_0 = \{x_i\}_{i=1}^n$, while the graph corresponding to the limit as $h \to \infty$ has the property that $|V_\infty| = 1$.*

Proof. First, examine the case of the limit as $h \to 0$. Let $\xi = \min_{i \neq j} \|x_i - x_j\|$; then for all $h < \xi$, f consists of n nonoverlapping "bumps." (They are strictly nonoverlapping only in the case of finite support kernels; if the kernels have infinite support, then this property is easily seen to be attained as $h \to 0$ and the kernels approach δ-functions.) It is then the case that f has exactly n modes, and corresponding to each is a basin of attraction. Furthermore, the centers of the bumps are the data points x_i, so that the modes are indeed the x_i. Thus, $V_0 = \{x_i\}_{i=1}^n$.

Now examine the case of the limit as $h \to \infty$. We assume the case of radially symmetric kernels with infinite support, though a proof may be supplied in the general case as well; that is, we take $f(x) = \sum_{i=1}^n k(\|\frac{x-x_i}{h}\|^2)$. In this case, the derivative of f is easily shown to satisfy $f'(x) \propto \sum_{i=1}^n (x - x_i) k'(\|\frac{x-x_i}{h}\|^2)$. Thus, any critical point of f, i.e., any point for which $f'(x) = 0$, will satisfy

$$x = \frac{\sum_{i=1}^n k'(\|\frac{x-x_i}{h}\|^2) x_i}{\sum_{i=1}^n k'(\|\frac{x-x_i}{h}\|^2)}. \tag{6.1}$$

As $h \to \infty$, $k'(\|\frac{x-x_i}{h}\|^2)$ approaches a value independent of x; e.g., for the Gaussian kernel, the profile is $k(z) = e^{-z}$, so that this value is $k'(0) = -1$. Thus,

$$\frac{k'(\|\frac{x-x_j}{h}\|^2)}{\sum_{i=1}^n k'(\|\frac{x-x_i}{h}\|^2)} \to \frac{1}{n}.$$

Substituting back into (6.1) gives that, as $h \to \infty$, the solution to $f'(x) = 0$ is $x = \frac{1}{n} \sum_{i=1}^n x_i$, and this is the only solution to (6.1). That is, there is a *single* critical point.

Now, it remains to show that this critical point is a local maximum. To do this, we will invoke an argument based on algebraic topology. However, in order for this argument to apply, we must have a compact domain for f, which is clearly not the case as f's domain is $\mathbb{R}^d$. We therefore use the so-called one-point compactification [27], in which we add a single point at

∞ to the domain, yielding a compact space $\mathcal{D}$ which is easily seen to be homeomorphic to the d-dimensional sphere S^d.

The original function $f : \mathbb{R}^d \to \mathbb{R}$ was shown to have a single critical point; the new function $\tilde{f} : \mathcal{D} \to \mathbb{R}$ can be shown to have a second critical point, at ∞. To see this, note that, as before, $\tilde{f}'(x) \propto \sum_{i=1}^n (x - x_i) k'(\|\frac{x-x_i}{h}\|^2)$, so that $\tilde{f}'(x) = 0$ at $x = \infty$ (the function $k'(\|\frac{x-x_i}{h}\|^2)$ goes to 0 as $x \to \infty$ much faster than x itself goes to ∞). So the second critical point is indeed the newly added point at ∞.

Finally, we must classify these two critical points as maxima, minima, or any of $d-1$ different types of saddle points. This may be achieved with the aid of the weak Morse inequalities [19], which state that $\beta_k \leq m_k$ for $k = 0, \ldots, d$, where β_k is the kth order Betti number of the domain $\mathcal{D}$, and m_k is the number of critical points of order k (a minimum has order 0, a maximum has order d, and the $d-1$ different types of saddle points lie in between). Now, recall that $\mathcal{D}$ is homeomorphic to the d-dimensional sphere S^d; a well-known result from algebraic topology [27] states that $\beta_0(S^d) = \beta_d(S^d) = 1$, while $\beta_k(S^d) = 0$ for all $k = 1, \ldots, d-1$. Thus, the weak Morse inequalities state that $m_0 \geq 1$ and $m_d \geq 1$, while $m_k \geq 0$ for all other k. However, we have just shown above that there are exactly two critical points, i.e., that $\sum_{k=0}^d m_k = 2$. The only feasible solution to the combination of this sum and the weak Morse inequalities is $m_0 = 1$ and $m_d = 1$. Now, clearly the point at ∞ is a local minimum, as $f(\infty) = 0$ and in general $f(x) \geq 0$; thus, the remaining point at $x = \frac{1}{n}\sum_{i=1}^n x_i$ must be a local maximum. Then finally, $|V_\infty| = 1$, as desired. ■

Now, to convert from a continuous multiscale graph structure to a discrete one, we simply choose a fixed number of bandwidths, i.e., $G_\ell = G_{h_\ell}$, where the levels run from $\ell = 0$ (finest) to $\ell = L$ (coarsest). In particular, in d-dimensional space, it is natural to choose $h_\ell \approx 2^{\frac{1}{d}} h_{\ell-1}$, so that the *volume* of coverage doubles at each stage. In this case, we expect $L \approx \log n$; using a line of argument similar to that of section 4.3, we may show that computing this discrete multiscale graph structure using ordinary mean shift requires $O(Ln^2) = O(n^2 \log n)$ time. (This assumes a naive nearest neighbor data structure; see section 4.3 for a more in-depth discussion.)

To use the fast mean shift algorithm, we make two observations. The first is that for the initial level ($\ell = 0$) of the multiscale graph structure, we may use $m_0 \ll n$ samples; this is precisely what we have shown in sections 2 and 5, from the theoretical and experimental points of view, respectively. The second observation is that for all subsequent levels, we may use even fewer samples, due to the increasing bandwidths of these levels. We have that $h_\ell \approx 2^{\frac{1}{d}} h_{\ell-1}$, and we may convert this statement about bandwidths to one about the number of samples to use at each level of the graph, using the result of section 2.4 for bandwidth selection. In particular, if m_ℓ is the number of samples required at level ℓ, this leads to $m_\ell \approx 2^{-\frac{d+1}{d}} m_{\ell-1}$ or, for d large, $m_\ell \approx \frac{1}{2} m_{\ell-1}$. Again, following the logic of section 4.3, we have the complexity of the fast version of the discrete multiscale graph structure as $O(\sum_{\ell=0}^L n 2^{-\ell} m_0) = O(nm_0)$.

Note that the speed-up factor in computing the multiscale graph structure is even better than that of mean shift itself; it is $(n/m) \log n$, versus n/m for the mean shift speed-up. In cases of interest where $n \approx 10^7$ or 10^8, the $\log n$ factor is not insubstantial (i.e., 20–25).

6.2. An example. In Figure 8, we show part of a multiscale graph structure for an image of a seashore, computed using the fast technique described above. The four images we show

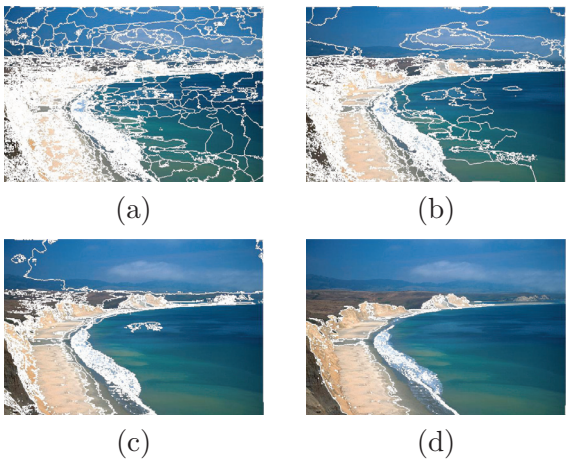

Figure 8. *A visualization of the multiscale graph structure.* (a) $h = 8$, $|V| = 226$; (b) $h = 16$, $|V| = 59$; (c) $h = 32$, $|V| = 11$; *and* (d) $h = 64$, $|V| = 3$. *See accompanying discussion in text.*

are for bandwidths $h = 8$, 16, 32, and 64. The number of vertices for these graphs are 266, 59, 11, and 3, respectively. We visualize the graphs by surrounding each contiguous region in white; note, however, that these contiguous regions are *not* the segments themselves, and often a segment (and hence a vertex) consists of multiple such contiguous regions. This is due to the fact that the graphs are not necessarily planar. This is true in our example, as our feature vector is color (L*a*b*) rather than texture, without the addition of position; thus, the sandy textured region looks as though it contains many nodes, whereas in fact it contains only a few noncontiguous nodes. Even with this effect, one can see the graph simplification as the bandwidth increases.

7. Conclusion.

We have presented a fast version of the mean shift algorithm, based on computing a pared-down KDE using a sampling procedure. We have theoretically demonstrated the closeness of the pared-down KDE to the original KDE, as well as the superior complexity of the fast mean shift algorithm compared to the standard mean shift. We have experimentally verified the algorithm's speed and have shown its potential utility in clustering large data sets, such as video, and in the computation of complex data structures such as the multiscale graph structure presented. It is our hope that the algorithm can find immediate application in fast segmentation of existing large data sets, such as medical images.

REFERENCES

[1] S. Alpert, M. Galun, R. Basri, and A. Brandt, *Image segmentation by probabilistic bottom-up aggregation and cue integration*, in Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'07), 2007.

[2] A. Andoni and P. Indyk, *Near-optimal hashing algorithms for approximate nearest neighbor in high dimensions*, Comm. ACM, 51 (2008), pp. 117–122.

[3] D. Barash and D. Comaniciu, *A common framework for nonlinear diffusion, adaptive smoothing, bilateral filtering and mean shift*, Image Vision Comput., 22 (2004), pp. 73–81.

[4] Y. Cheng, *Mean shift, mode seeking, and clustering*, IEEE Trans. Pattern Anal. Mach. Intell., 17 (1995), pp. 790–799.
[5] C. Christoudias, B. Georgescu, and P. Meer, *Synergism in low level vision*, in Proceedings of the 16th International Conference on Pattern Recognition, IEEE Computer Society, Washington, DC, 2002, pp. 150–155.
[6] D. Comaniciu, *An algorithm for data-driven bandwidth selection*, IEEE Trans. Pattern Anal. Mach. Intell., 25 (2003), pp. 281–288.
[7] D. Comaniciu and P. Meer, *Mean shift: A robust approach toward feature space analysis*, IEEE Trans. Pattern Anal. Mach. Intell., 24 (2002), pp. 603–619.
[8] D. Comaniciu and V. Ramesh, *Mean shift and optimal prediction for efficient object tracking*, in Proceedings of the IEEE International Conference on Image Processing, Vol. 3, 2000, pp. 70–73.
[9] D. Comaniciu, V. Ramesh, and P. Meer, *Kernel-based object tracking*, IEEE Trans. Pattern Anal. Mach. Intell., 25 (2003), pp. 564–575.
[10] D. DeMenthon, *Spatio-temporal segmentation of video by hierarchical mean shift analysis*, in Proceedings of the Statistical Methods in Video Processing Workshop, 2002.
[11] L. Devroye and L. Györfi, *Nonparametric Density Estimation: The L_1 View*, John Wiley & Sons, New York, 1985.
[12] P. Felzenszwalb and D. Huttenlocher, *Efficient graph-based image segmentation*, Int. J. Comput. Vision, 59 (2004), pp. 167–181.
[13] D. Freedman and P. Kisilev, *Fast mean shift by compact density representation*, in Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'09), 2009, pp. 1818–1825.
[14] K. Fukunaga and L. Hostetler, *The estimation of the gradient of a density function, with applications in pattern recognition*, IEEE Trans. Inform. Theory, 21 (1975), pp. 32–40.
[15] B. Georgescu, I. Shimshoni, and P. Meer, *Mean shift based clustering in high dimensions: A texture classification example*, in Proceedings of the Ninth International Conference on Computer Vision, IEEE Computer Society, Washington, DC, 2003, pp. 456–463.
[16] J. Goldberger, H. Greenspan, and J. Dreyfuss, *Simplifying mixture models using the unscented transform*, IEEE Trans. Pattern Anal. Mach. Intell., 30 (2008), pp. 1496–1502.
[17] I. Grabec, *Self-organization of neurons described by the maximum-entropy principle*, Biol. Cybernet., 63 (1990), pp. 403–409.
[18] H. Guo, P. Guo, and H. Lu, *A fast mean shift procedure with new iteration strategy and re-sampling*, in Proceedings of the IEEE International Conference on Systems, Man and Cybernetics (SMC'06), Vol. 3, 2006, pp. 2385–2389.
[19] A. Hatcher, *Algebraic Topology*, Cambridge University Press, Cambridge, UK, 2002.
[20] X. He, R. Zemel, and D. Ray, *Learning and incorporating top-down cues in image segmentation*, in Proceedings of the European Conference on Computer Vision, Lecture Notes in Comput. Sci. 3951, Springer-Verlag, Berlin, 2006, pp. 338–351.
[21] K. Kim, K. Jung, and J. Kim, *Texture-based approach for text detection in images using support vector machines and continuously adaptive mean shift algorithm*, IEEE Trans. Pattern Anal. Mach. Intell., 25 (2003), pp. 1631–1639.
[22] P. Kohli, L. Ladicky, and P. Torr, *Robust higher order potentials for enforcing label consistency*, in Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'08), 2008.
[23] V. Kolmogorov and R. Zabih, *What energy functions can be minimized via graph cuts?*, IEEE Trans. Pattern Anal. Mach. Intell., 26 (2004), pp. 147–159.
[24] Y. Leung, J. Zhang, and Z. Xu, *Clustering by scale-space filtering*, IEEE Trans. Pattern Anal. Mach. Intell., 22 (2000), pp. 1396–1410.
[25] D. Martin, C. Fowlkes, D. Tal, and J. Malik, *A database of human segmented natural images and its application to evaluating segmentation algorithms and measuring ecological statistics*, in Proceedings of the Eighth International Conference on Computer Vision (ICCV 2001), IEEE Computer Society, Washington, DC, 2001, pp. 416–423.
[26] D. Mumford and J. Shah, *Optimal approximations by piecewise smooth functions and associated variational problems*, Comm. Pure Appl. Math., 42 (1989), pp. 577–685.

[27] J. Munkres, *Elements of Algebraic Topology*, Addison–Wesley, Menlo Park, CA, 1984.
[28] S. Paris and F. Durand, *A topological approach to hierarchical segmentation using mean shift*, in Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'07), 2007.
[29] W. Rand, *Objective criteria for the evaluation of clustering methods*, J. Amer. Statist. Assoc., 66 (1971), pp. 846–850.
[30] B. Russell, A. Efros, J. Sivic, W. Freeman, and A. Zisserman, *Using multiple segmentations to discover objects and their extent in image collections*, in Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'06), 2006, pp. 1605–1614.
[31] J. Shi and J. Malik, *Normalized cuts and image segmentation*, IEEE Trans. Pattern Anal. Mach. Intell., 22 (2000), pp. 888–905.
[32] B. Silverman, *Density Estimation for Statistics and Data Analysis*, Chapman & Hall/CRC, London, 1986.
[33] H. Traven, *A neural network approach to statistical pattern classification by 'semiparametric' estimation of probability density functions*, IEEE Trans. Neural Netw., 2 (1991), pp. 366–377.
[34] A. Vedaldi and S. Soatto, *Quick shift and kernel methods for mode seeking*, in Proceedings of the European Conference on Computer Vision (ECCV), 2008, pp. 705–718.
[35] J. Wang, B. Thiesson, Y. Xu, and M. Cohen, *Image and video segmentation by anisotropic kernel mean shift*, in Proceedings of the European Conference on Computer Vision, Lecture Notes in Comput. Sci. 3024, Springer-Verlag, Berlin, 2004, pp. 238–249.
[36] P. Wang, D. Lee, A. Gray, and J. Rehg, *Fast mean shift with accurate and stable convergence*, J. Mach. Learn. Res., 2 (2007), pp. 604–611.
[37] J. Wu and C. Chan, *A three-layer adaptive network for pattern density estimation and classification*, Int. J. Neural Syst., 2 (1991), pp. 211–220.
[38] C. Yang, R. Duraiswami, N. Gumerov, and L. Davis, *Improved fast Gauss transform and efficient kernel density estimation*, in Proceedings of the Ninth IEEE International Conference on Computer Vision, 2003, pp. 664–671.
[39] L. Yang, P. Meer, and D. Foran, *Multiple class segmentation using a unified framework over mean-shift patches*, in Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'07), 2007.