Acceleration Strategies for Gaussian Mean-Shift Image Segmentation

Miguel Á. Carreira-Perpiñán
Dept. of Computer Science & Electrical Eng., OGI, Oregon Health & Science University
miguel@csee.ogi.edu

Abstract

Gaussian mean-shift (GMS) is a clustering algorithm that has been shown to produce good image segmentations (where each pixel is represented as a feature vector with spatial and range components). GMS operates by defining a Gaussian kernel density estimate for the data and clustering together points that converge to the same mode under a fixed-point iterative scheme. However, the algorithm is slow, since its complexity is $\mathcal{O}(kN^2)$, where N is the number of pixels and k the average number of iterations per pixel. We study four acceleration strategies for GMS based on the spatial structure of images and on the fact that GMS is an expectation-maximisation (EM) algorithm: spatial discretisation, spatial neighbourhood, sparse EM and EM–Newton algorithm. We show that the spatial discretisation strategy can accelerate GMS by one to two orders of magnitude while achieving essentially the same segmentation; and that the other strategies attain speedups of less than an order of magnitude.

The mean-shift algorithm is a hill-climbing algorithm that operates as follows. Given a data set $\{\mathbf{x}_n\}_{n=1}^N \subset \mathbb{R}^D$, define a kernel density estimate

$$p(\mathbf{x}) = \frac{1}{N}\sum_{n=1}^{N} K\left(\left\|\frac{\mathbf{x}-\mathbf{x}_n}{\sigma}\right\|^2\right) \tag{1}$$

where $K(t)$ is a kernel function (e.g. $K(t) = e^{-t/2}$ for the Gaussian). Then, rearranging the stationary-point equation $\nabla p(\mathbf{x}) = \mathbf{0}$ one can easily obtain the iterative scheme $\mathbf{x}^{(\tau+1)} = \mathbf{f}(\mathbf{x}^{(\tau)})$ with

$$\mathbf{f}(\mathbf{x}) = \sum_{n=1}^{N} \frac{K'\left(\left\|\frac{\mathbf{x}-\mathbf{x}_n}{\sigma}\right\|^2\right)}{\sum_{n'=1}^{N} K'\left(\left\|\frac{\mathbf{x}-\mathbf{x}_{n'}}{\sigma}\right\|^2\right)}\mathbf{x}_n \tag{2}$$

(where $K' = dK/dt$), called *mean-shift algorithm*. The fixed points of $\mathbf{f}$ are the stationary points of the density p. The mean-shift algorithm can be shown to converge to modes of the kernel density estimate under mild conditions. The algorithm originates in [8] (though not in the form above) and has been subsequently developed by others [1, 4, 5].

The mean-shift algorithm can be applied to clustering by declaring each mode of the kernel density estimate as representative of one cluster, and assigning a data point $\mathbf{x}_n$ (or indeed any point $\mathbf{x} \in \mathbb{R}^D$) to the mode it converges to, $\mathbf{f}^\infty(\mathbf{x}_n)$. Since the algorithm does not depend on parameters such as step sizes, the clustering is uniquely defined given the kernel density estimate, i.e., given the bandwidth σ. The algorithm has been proven particularly successful in image segmentation [5] where each data point $\mathbf{x}_n$, i.e., each pixel, is represented by *spatial* and *range* features, e.g. (i, j, I) or (i, j, L^*, u^*, v^*) where (i, j) is the pixel location in the image and I and (L^*, u^*, v^*) the pixel value in a greyscale or colour image, respectively.

Different kernels give rise to different versions of the mean-shift algorithm. Here, we focus exclusively on *Gaussian mean-shift* (GMS), where the kernel density estimate is a Gaussian mixture and the algorithm can be written in the following, elegant form [1]:

$$p(n|\mathbf{x}^{(\tau)}) = \frac{\exp\left(-\frac{1}{2}\left\|(\mathbf{x}^{(\tau)}-\mathbf{x}_n)/\sigma\right\|^2\right)}{\sum_{n'=1}^{N}\exp\left(-\frac{1}{2}\left\|(\mathbf{x}^{(\tau)}-\mathbf{x}_{n'})/\sigma\right\|^2\right)} \tag{3a}$$

$$\mathbf{x}^{(\tau+1)} = \textstyle\sum_{n=1}^{N} p(n|\mathbf{x}^{(\tau)})\mathbf{x}_n \tag{3b}$$

i.e., the new iterate $\mathbf{x}^{(\tau+1)}$ is the data average under the posterior probabilities given the current iterate $p(n|\mathbf{x}^{(\tau)})$. GMS produces better segmentations than the Epanechnikov kernel [5] but requires a large bandwidth ($\sigma \approx \frac{1}{5}$ of the image side) and is far slower to converge. Indeed, the Epanechnikov kernel has finite support and so convergence occurs in a finite number of steps [5]. However, GMS is an expectation-maximisation (EM) algorithm [3, 2] where the E step (eq. (3a)) computes the posterior probabilities $p(n|\mathbf{x})$ and the M step (eq. (3b)) updates the iterate $\mathbf{x}$. Thus, its convergence has linear order [2, 11], requiring many iterations to attain good accuracy. Besides, each iteration costs $\mathcal{O}(N)$ for each data point, so that clustering the whole data set costs $\mathcal{O}(kN^2)$ where k is the average number of iterations. This is particularly expensive for image segmentation where the number of pixels N is large.

Our objective in this paper is to study ways of accelerating GMS for image segmentation. We propose four different strategies which can significantly reduce the computational cost while obtaining almost the same segmentation. They exploit the grid structure of image data and the fact that GMS is an EM algorithm. For simplicity, we focus on a scalar bandwidth σ, the same for all kernels.

The rest of the paper is organised as follows. We introduce the acceleration strategies and their evaluation in terms of computational cost and clustering error (section 1), describe each strategy in detail (sections 2–5) and give experimental results with different bandwidths (section 6). Finally, we discuss the results and related work (section 7).

1. Overview of strategies and their evaluation

Let us first determine the bottlenecks of the GMS algorithm. We have a data set of N points (pixels) in D dimensions (e.g. 3 for greyscale images, 5 for colour images, higher if using texture or edge features). Assume that the average number of iterations per data point (per pixel) is k, so that the complexity of the algorithm is $\mathcal{O}(kN^2D)$.

Bottleneck 1 The average number of iterations k is large, typically around 100 (depending on the convergence tolerance and the bandwidth σ). This high number of iterations is spent not only near the mode of convergence but also in slow crawls up ridges of $p(\mathbf{x})$.

Bottleneck 2 The cost per iteration is large, about $2ND$ multiplications. This is required to obtain the posterior probability $p(n|\mathbf{x})$ for each data point (the E step, ND multiplications) and to obtain the next iterate $\mathbf{x}^{(\tau+1)}$ (the M step, ND multiplications).

A successful acceleration technique for GMS must address one or both bottlenecks. We propose the following four different strategies (we will call `ms` the exact GMS algorithm). `ms1`, *spatial discretisation*, divides the spatial domain of the image into cells of subpixel size and forces all points projecting on the same cell to converge to the same mode; it reduces the total number of iterations. `ms2`, *spatial neighbourhood*, uses a subset of points (rather than all N) in eq. (3), namely the nearest neighbours in the spatial domain rather than in the full D-dimensional feature space; it reduces the cost per iteration. `ms3`, *sparse EM*, is based on Neal and Hinton's idea [13] of interleaving full E steps with partial E steps, where only a fraction (the plausible set) of the posterior probabilities $p(n|\mathbf{x})$ are updated; it reduces the cost per iteration for most iterations. `ms4`, *EM–Newton*, starts with EM steps and later switches to Newton's method, which has quadratic convergence; it reduces the total number of iterations.

Any acceleration technique for GMS should be evaluated with respect to computational cost and segmentation error.

`ms`:	1
`ms1`:	1
`ms2`:	e
`ms3`:	2 if full step, e if partial step
`ms4`:	1 if EM step, $\left(1+\frac{D+1}{4}\right)$ if Newton step, $\left(\frac{3}{2}+\frac{D+1}{4}\right)$ if EM step after failed Newton step

Table 1. Cost per iteration (in number of multiplications relative to `ms`, the exact Gaussian mean-shift algorithm) for each of the accelerated methods, assuming N data points in D dimensions (with $N \gg D$). $e \in (0,1]$ is the fraction of the data set used (neighbours for `ms2`, plausible set for `ms3`). For `ms2` e is constant, while for `ms3` e varies across iterations. The number of exponentials for each method generally scales in the same proportion as the number of multiplications, so we consider only the latter in the table.

The memory cost is modest for all our strategies so we focus on the time cost. Rather than measuring this as running time, which depends on the actual hardware and software (and is certainly misleading in our Matlab implementation), we measure it in normalised iterations, where an exact GMS iteration (i.e., $2ND$ multiplications) equals 1. The cost of each strategy's iteration is determined in the next sections and summarised in table 1.

Our gold standard is the segmentation produced by GMS (for a given bandwidth) without using any pre- or post-processing (such as removing small clusters), irrespective of the perceptual quality of the segmentation. We use a simple percent measure of clustering error $P \in [0,100]$ obtained by matching corresponding clusters between the two segmentations, counting the number of misclustered pixels over the whole image and dividing by N. This simple error measure works well since the strategies generally produce very similar segmentations, having the same number of clusters and differing only in a few pixels near cluster boundaries.

Clustering errors can arise for two reasons. First, if the iterative scheme for the approximate method does not converge to a mode of $p(\mathbf{x})$, then the convergence points will not coincide with the modes, and there may be more or fewer convergence points than modes. This is the case for `ms1` and `ms2`. However, the error can be kept under control depending on the parameter of the method (at a higher computational cost). Second, even if the accelerated iterative scheme does converge to a mode of $p(\mathbf{x})$, the mode of convergence for a given starting point $\mathbf{x}_n$ may be different from that using exact GMS, in particular for points near cluster boundaries. This is the case for `ms3` and `ms4`.

2. `ms1`: spatial discretisation

We consider each pixel in the spatial domain as a unit square centred at location (i,j) and subdivided into an $n \times n$ grid of cells (fig. 1). The parameter of this method is n. The idea is that all points $\mathbf{x} \in \mathbb{R}^D$ (the feature space) whose

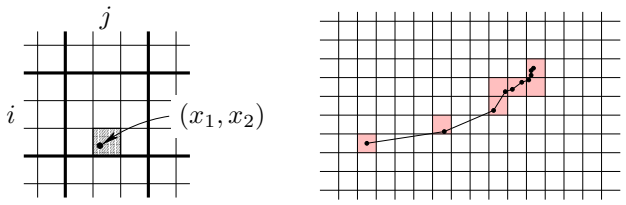

Figure 1. *Left*: discretisation of the spatial domain for $n = 3$. Every pixel (i, j) is subdivided into $n \times n$ cells. Every point $\mathbf{x} \in \mathbb{R}^D$ whose spatial projection (x_1, x_2) falls in a given cell (such as the one showed) has the same fate as all other points projecting on that cell. *Right*: a path $\mathbf{x}^{(0)}, \mathbf{x}^{(1)}, \cdots \in \mathbb{R}^D$ visits cells in the spatial domain; iterates from a later pixel stop iterating when they reach a visited cell.

spatial projection (x_1, x_2) belongs to the same cell share the same fate, i.e., converge to the same mode. Thus, if we start GMS from a data point $\mathbf{x}_n$ and at some iteration $\mathbf{x}^{(\tau)}$ projects on a cell that we have previously visited while iterating GMS for some other data point $\mathbf{x}_m$, then we stop iterating, and mark every iterate of $\mathbf{x}_n$'s path as visited and as converging to $\mathbf{x}_m$'s mode.

The intutition why this idea should work is the following. Consider a greyscale image for simplicity, with features $\mathbf{x} = (i, j, I) \in \mathbb{R}^3$. The image defines a 2D manifold because the intensity I is a function of the spatial location (i, j). Since the GMS iterates must lie in the convex hull of the data points [1] and moreover lie in high-density areas (since EM increases the density monotonically), the iterates will typically lie very near the said manifold. Thus, the iterate paths that go through a given location (i, j) do so at approximately the same intensity $I(i, j)$. This means that the paths are well approximated by their 2D projections in the spatial domain. Naturally, this situation does not apply with non-image data; there the discretisation should involve all D variables (which suffers from the curse of dimensionality).

The number of modes found by `ms1` is less than or equal to the number of modes found by GMS. Clustering errors occur when paths eventually converging to different modes travel temporarily close together, thus competing for the same cells (usually along cluster boundaries). The error can be reduced as much as desired by using a fine enough discretisation (high enough n), since the total number of iterates for all paths is finite given the convergence tolerance.

The order in which pixels are selected for GMS affects the result, because future paths depend on which cells have been already visited. We have found that selecting first a collection of pixels distributed as a uniform grid over the image and then the rest of pixels attains a lower error than selecting pixels rowwise, with no change in computational cost (using a random order also lowers the error but has a larger variance).

Each iteration in `ms1` is an exact GMS iteration, so the total cost is the number of iterations performed. The savings in `ms1` come from the fact that the long crawls along density ridges and upon convergence near a mode (which consume a large number of iterations) are done only for the first few pixels. Essentially, once a path has been travelled once, it is not travelled again; unlike with exact GMS, where almost identical paths are travelled by many different pixels converging to the same mode [2], and a given cell may be visited many times during one path.

3. `ms2`: spatial neighbourhood

We approximate the E and M steps at each iteration by using a subset of the data: the sums over all n in eqs. (3a)–(3b) are replaced with sums over $n \in \mathcal{N}(\mathbf{x}^{(\tau)})$. Here, $\mathcal{N}(\mathbf{x}) = \{n \in \{1, \dots, N\} : \|(x_1^*, x_2^*) - (x_{n1}, x_{n2})\|_\infty \leq r\}$ consists of a spatial neighbourhood of $\mathbf{x}$ given by the data points corresponding to pixels within distance r (using the ∞-norm) of $\mathbf{x}$'s nearest pixel location (x_1^*, x_2^*). $\mathcal{N}(\mathbf{x})$ contains $(2\lfloor r \rfloor + 1)^2$ pixels (where $\lfloor \cdot \rfloor$ is the floor function) for pixels both in the interior of the image and close to the boundary (by shifting the neighbourhood inside the image as needed). Thus, the amount of computation is the same for any pixel: one iteration of `ms2` costs $e \in (0, 1]$ iterations of GMS, where $e = (2\lfloor r \rfloor + 1)^2 / N$ is the proportion of the data set used in the iteration. Importantly, note that the cost of finding the nearest neighbours in the spatial domain is negligible, unlike that of finding nearest neighbours in the whole space. The parameter of this method is r (or equivalently e).

The computational savings of this method arise from the reduction of the cost per iteration. The overall speedup depends on the total number of iterations carried out, which is investigated experimentally in section 6. The `ms2` iteration does not converge to a mode of $p(\mathbf{x})$, though the error should be small if the neighbourhood size significantly exceeds the bandwidth. The number of modes may be less than, equal to or greater than that of GMS.

4. `ms3`: sparse EM

The sparse EM algorithm [13] is an accelerated EM algorithm that preserves the convergence guarantee of EM. We will describe it in the context of the GMS algorithm for simplicity. The idea consists of recasting the EM algorithm as an alternate maximisation of the free energy function $F(\tilde{\mathbf{p}}, \mathbf{x})$ defined below in an enlarged parameter space with $N + D$ variables $(\tilde{\mathbf{p}}, \mathbf{x}) \in \mathbb{R}^{N+D}$, where $\mathbf{x} \in \mathbb{R}^D$ is the usual GMS iterate, and $\tilde{\mathbf{p}} = (\tilde{p}_1, \dots, \tilde{p}_N)^T$ represents a distribution over $\{1, \dots, N\}$ such as $p(n|\mathbf{x})$; let us call $\mathbf{p}_\mathbf{x}$ the posterior distribution, i.e., $\mathbf{p}_\mathbf{x}(n) = p(n|\mathbf{x})$. The free energy function is defined as

$$F(\tilde{\mathbf{p}}, \mathbf{x}) = \log p(\mathbf{x}) - \mathrm{D}\left(\tilde{\mathbf{p}} \| \mathbf{p}_\mathbf{x}\right) \tag{4}$$

where $p(\mathbf{x})$ is the usual density of eq. (1) and $\mathrm{D}(\cdot\|\cdot)$ the Kullback-Leibler divergence. The key observation [13] is that the maxima of F correspond to maxima of $p(\mathbf{x})$ and vice versa. The EM algorithm is an alternate maximisation of F: the E step maximises F with respect to $\tilde{\mathbf{p}}$ for fixed $\mathbf{x}$ (which results in $\tilde{\mathbf{p}} = \mathbf{p}_{\mathbf{x}}$, as is obvious from eq. (4)) and the M step maximises F with respect to $\mathbf{x}$ for fixed $\tilde{\mathbf{p}}$ (which can be seen by taking partial derivatives of F). A second key observation is to realise that even though the maximisation with respect to $\tilde{\mathbf{p}}$ (the E step) has a closed form solution, it is a computationally costly solution since we must compute $\tilde{\mathbf{p}}_n = p(n|\mathbf{x})\ \forall n \in \{1,\dots,N\}$, at a cost of $\mathcal{O}(N)$. In the spirit of incremental algorithms, or inexact searches in optimisation, it may be more efficient to maximise with respect to $M < N$ components of $\tilde{\mathbf{p}}$ in the hope that we make significant progress towards an maximum at a lower computational cost. We call this a partial E step, as opposed to a (usual) full E step where all components of $\tilde{\mathbf{p}}$ are updated. Note that no matter which components or how many of them we update, we will increase F (even if $p(\mathbf{x})$ temporarily decreases) and—as long as all components are updated periodically— we will eventually converge to a maximum of F and thus a maximum of $p(\mathbf{x})$.

Computational savings occur when we do many partial steps updating the same set $S \subset \{1,\dots,N\}$ of components (called *plausible set*). The partial step updates the posterior probabilities for $n \in S$ while the full step, which is taken at infrequent iterations, selects a new plausible set and updates all components (as in a usual E step). The cost for a partial step is then roughly $2ED$ multiplications where E is the number of components in S, while the cost for a full step is the usual $2ND$ multiplications plus the cost of determining the plausible set S (see below). The algorithm's details (omitted here for lack of space) are in [13].

In summary, the sparse EM algorithm will infrequently run a full iteration where all component probabilities are updated (and both F and $p(\mathbf{x})$ increase) at a slightly higher cost than a usual EM iteration; and frequently a partial iteration where only a portion of the components are updated (and F increases but not necessarily $p(\mathbf{x})$) at the proportional cost of the usual EM iteration. The overall cost depends on the total number of iterations, but generally is smaller than for EM.

We have now two important issues left: the choice of the plausible set S at a full step, and the decision whether to take a full step. The intuition behind the plausible set is, as in `ms2`, that most of the probability mass (and thus most of the effect on the iterate) is due to a relatively small number of components, those closer to the current iterate. We choose as many components (in decreasing order of posterior probability) as necessary to account for a total probability $1-\epsilon$ with $\epsilon \in [0,1)$, i.e., $\sum_{n\in S} p(n|\mathbf{x}) \geq 1-\epsilon$. Thus, the number of components (and the fraction of data used $e = E/N$) varies after each full step. We have observed experimentally that this adapts better to the particular problem than choosing the E nearest neighbours of $\mathbf{x}$ for fixed E. The parameter of this method is ϵ. Given the posterior probabilities, the cost of computing the plausible set is roughly the cost of sorting the posterior probabilities, which requires $\mathcal{O}(N \log N)$ comparisons. We take the sorting cost to be similar to the cost of a usual E step (ND multiplications), so that the cost of the full and partial steps in units of usual EM steps is 2 and e, respectively. The sorting cost might be reduced by noting that the old list of components is partially ordered at the current full E step, and using a sorting algorithm tailored for partially sorted lists.

The decision when to take a full step, i.e., when to update the plausible set, is as follows. We run partial steps until the distance between consecutive iterates is less than 10^{-3}, or we reach 20 steps; then we take a full step. We found this rule to work best over different ϵ, σ and images. Finally, note that `ms3` always starts and ends at a full E step.

5. `ms4`: EM–Newton

This combines GMS (an EM algorithm) with Newton's method. A similar idea was proposed in [1], where gradient ascent was combined with Newton's method. Experience with the EM algorithm in general [11] and with GMS [1] suggests that the EM algorithm is efficient at quickly moving from the starting point to a point where $p(\mathbf{x})$ is high, but then it slows down considerably, particularly near a mode due to its linear convergence rate. In that case, taking instead a Newton step can make faster progress towards the mode. This will be particularly noticeable near the mode, where Newton's method quadratic convergence will pin down the solution to machine precision in just a few steps. However, away from the mode the Newton step may not improve as much as the EM step, and may even go downhill or be undefined or too long if the Hessian is not positive definite.

Let us derive the Newton step for the kernel density estimate (1). The gradient and Hessian of the density are:

$$\mathbf{g}(\mathbf{x}) = \frac{p(\mathbf{x})}{\sigma^2}\sum_{n=1}^{N} p(n|\mathbf{x})(\mathbf{x}_n - \mathbf{x}) = \frac{p(\mathbf{x})}{\sigma^2}(\mathbf{x}_{\mathrm{EM}} - \mathbf{x})$$

$$\mathbf{H}(\mathbf{x}) = \frac{p(\mathbf{x})}{\sigma^2}\overline{\mathbf{H}}(\mathbf{x}),\ \text{with}$$

$$\overline{\mathbf{H}}(\mathbf{x}) = -\mathbf{I} + \frac{1}{\sigma^2}\sum_{n=1}^{N} p(n|\mathbf{x})(\mathbf{x}_n - \mathbf{x})(\mathbf{x}_n - \mathbf{x})^T$$

where $\mathbf{x}_{\mathrm{EM}}$ is the EM step from (3b). The Newton step is

$$\mathbf{x}_{\mathrm{N}} = \mathbf{x} - \mathbf{H}^{-1}(\mathbf{x})\mathbf{g}(\mathbf{x}) = \mathbf{x} - \overline{\mathbf{H}}^{-1}(\mathbf{x})(\mathbf{x}_{\mathrm{EM}} - \mathbf{x}). \quad (5)$$

We see that, in computing the Newton step, we get the EM step for free. Given the EM step, the cost of the New-

ton step is roughly the cost of computing the $\overline{\mathbf{H}}$ matrix, namely $\frac{ND(D+1)}{2}$ multiplications, because the cost of solving the linear system in $\overline{\mathbf{H}}$ is $\mathcal{O}(D^3)$ which is negligible when $N \gg D$, the case in image segmentation and most clustering applications.

The crucial issue in the combination EM–Newton is when to try the Newton step. We enable Newton steps when the current sequence of iterations slows down. Specifically, at iteration $\tau+1$ and subsequent iterations we try a Newton step if $\left\|\mathbf{x}^{(\tau)} - \mathbf{x}^{(\tau-1)}\right\| < \theta$ for fixed θ. If the Newton step fails, i.e., $p(\mathbf{x}_{\mathrm{N}}^{(\tau+1)}) < p(\mathbf{x}^{(\tau)})$ (or the Hessian is close to singular), then we revert to the EM step and disable Newton steps until the EM steps slow down again. Thus, the first step (or first few steps) is always an EM step, and the final few steps are always Newton steps, achieving quadratic convergence. The parameter of this method is θ.

The cost of an iteration depends on its type: an EM iteration costs $2ND$ multiplications; a successful Newton iteration costs $2ND + \frac{ND(D+1)}{2}$ multiplications; and a failed Newton iteration reverting to the EM iteration costs $3ND + \frac{ND(D+1)}{2}$ multiplications (the additional cost being due to the computation of $p(\mathbf{x})$ for both $\mathbf{x}_{\mathrm{N}}$ and $\mathbf{x}_{\mathrm{EM}}$).

6. Experimental results

We evaluate the performance of the 4 acceleration strategies in terms of normalised iterations (so that iterations from different methods can be directly compared; see table 1) and percent clustering error P with respect to GMS. We test each method on several greyscale and colour images. In both cases, we prescale the greyscale or (L^*, u^*, v^*) values so that we can use an isotropic kernel with bandwidth σ and obtain good segmentations with GMS. Thus, each component of the feature vector $\mathbf{x}$ has now pixel units, as does σ. For each image we try several different σ (to obtain different numbers of clusters). We run each method with the same convergence criterion (that the change $\left\|\mathbf{x}^{(\tau+1)} - \mathbf{x}^{(\tau)}\right\|$ in the iterate be smaller than 10^{-3} pixels) and the resulting convergence points are considered to be the same mode if they lie less than 1 pixel apart. We obtained similar results over several greyscale and colour images, so we report numerical results only for the `cameraman` image of fig. 2.

For `ms2`, the parameter r is relative to the smallest image side, e.g. for `cameraman` 100×100 a value $r = 0.31$ means a neighbourhood radius of $0.31 \times 100 = 31$ pixels, and a proportion of data points used $e = (2\lfloor r \rfloor + 1)^2/N = 0.40$. For `ms4`, θ is relative to the bandwidth σ, e.g. for $\sigma = 16$ a value $\theta = 0.1$ means a distance $0.1 \times 16 = 1.6$ pixels between consecutive iterates.

We ran every method for a range of values of its parameter (fig. 3). For `ms3` the error is always near zero (because, unlike `ms2`, sparse EM converges to the true modes no matter the value of ϵ) and the computational cost is lowest at $\epsilon \approx 10^{-4}$; besides, the iterations ratio curve is very flat, so using 10^{-3} or 10^{-5} makes little difference. For the other methods, their respective parameter trades off error vs cost. We set the optimal value for every method's parameter as that which minimises the iterations ratio subject to achieving an error $P < 3\%$ (or as small as possible); see fig. 5. This ensures an almost perfect segmentation. For `ms1` this results in $n \approx 3$–6 and an iterations ratio of 0.01–0.1 (10–100× speedup), depending on the image and σ. Using larger n further reduces the error at a slightly higher computational cost. For `ms4` there is considerable freedom to choose θ (despite the jagged curve of fig. 5), because although the cost decreases with θ while the error increases with θ, the change is very small; $\theta = 0.1$ works well and results in speedups 1.5×–6×. The error increase with θ is expected since this means more Newton steps are taken instead of EM steps. For `ms2`, the iterations ratio is approximately equal to the fraction $e = (2\lfloor r \rfloor + 1)^2/N$ of data points used (where r is the spatial neighbourhood side). This indicates that the average number of iterations per pixel is about the same as for `ms`. The error P decreases as e increases. Thus, by using a high enough r (or e) we can reduce P to an acceptable level. However, it seems difficult to select an optimal r in advance given a new image and σ value, and the error can be large if using a slightly too small r. The reason is that `ms2` does not converge to the true modes of the density. Significant speedups (up to 5×) occur for small σ but not for large σ.

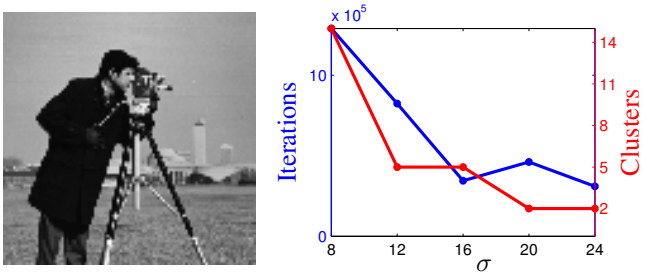

Figure 2. Original image `cameraman` 100×100, total number of iterations (blue) and number of clusters (red) for `ms` as a function of the bandwidth σ.

Fig. 4 shows some optimal segmentations, all being very similar to the `ms` segmentation, and the modes found by all methods being essentially the same (thus the same number of clusters), with a few misclustered pixels only. The picture showing the number of iterations for each pixel, which depends strongly on σ, indicates which image regions take longest (red) or shortest (blue), and indirectly reflects the flatness of the density $p(\mathbf{x})$. Often (but not always) clusters are identifiable in this picture for `ms`, as they are as peaks in the histogram of iterations. The iterations pictures for `ms2` and `ms3` tend to be very similar to that of `ms`, and like-

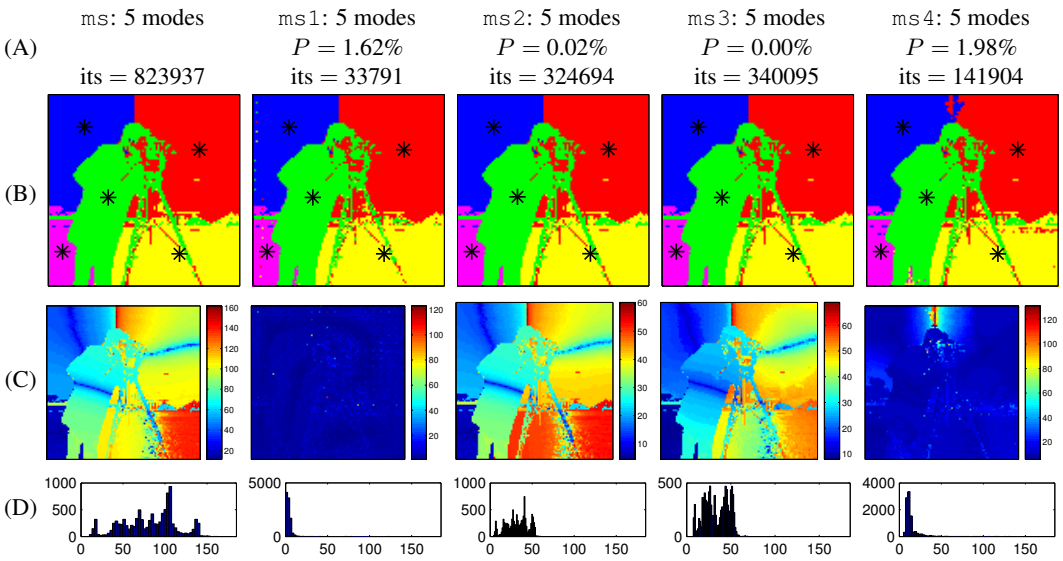

Figure 4. Segmentation results for each method under its optimal parameter value for $\sigma = 12$. For each method we give: (A) the number of modes, error P and number of iterations; (B) the colour-coded segmentation with modes marked $*$; and the distribution of the number of iterations at each pixel, (C) over the image and (D) as a histogram.

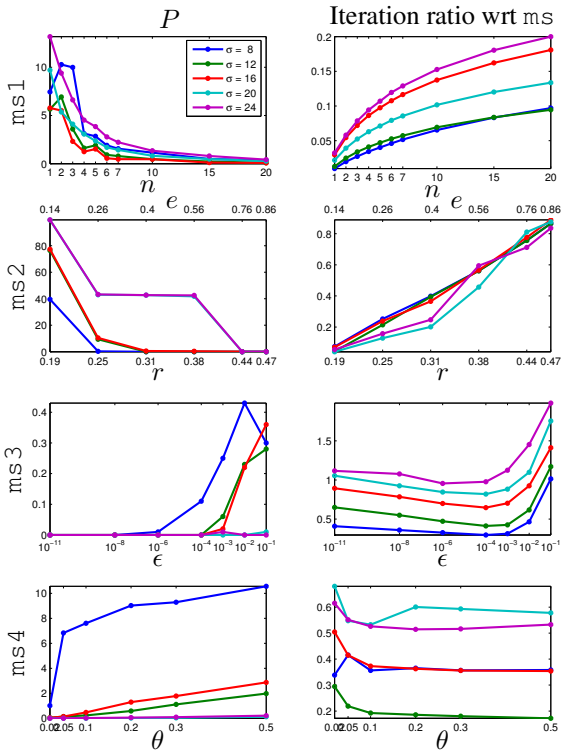

Figure 3. Clustering error P (percent) and computational cost for each method as a function of its parameter: ms1 (n), ms2 (r or e, shown in the upper X axis), ms3 (ϵ) and ms4 (θ). Each of the curves in each plot corresponds to a different bandwidth σ (see legend). In this figure, the iterations curves are ratios with respect to the number of iterations of ms, i.e., inverse speedups (thus, a ratio of 0.5 means the method took half the number of iterations as ms, or a 2× speedup).

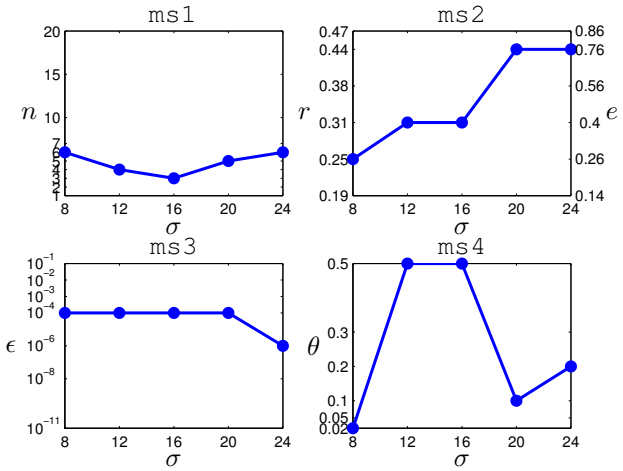

Figure 5. Optimal parameter value for each method as a function of the bandwidth σ.

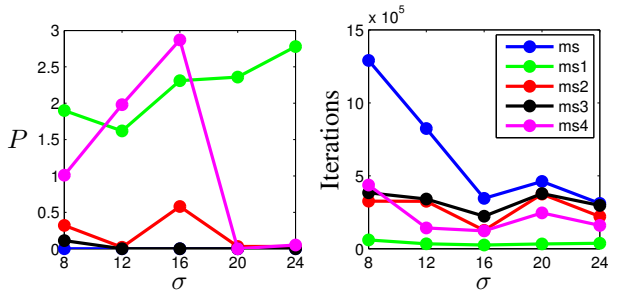

Figure 6. Clustering error P (percent) and number of iterations for each method under its optimal parameter value, as a function of the bandwidth σ.

wise their histograms look like a horizontally compressed version of that of ms. The reason is that these methods are the most similar to ms, being based on using a data subset but otherwise running EM-like steps. The picture for ms1 is completely different: only a handful of points run the average number of ms iterations (the pixels that were selected first, distributed as a grid over the image, visible as light spots); the overwhelming majority run a very small number of iterations (less than 4), as shown in the histogram as well, which peaks at 1.

Fig. 6 shows the error P and number of iterations for every method under its optimal parameter value, as a function of σ. All methods can attain significant speedups with a small error; the speedups are larger when ms takes more iterations (for small σ). The best strategy by far is ms1, with speedups 10×–100×, followed by ms4, with speedups 1.5×–6×. ms3 gives a modest speedup (from 3× for low σ to no speedup for large σ), although it is the safest strategy in terms of low P. Finally, ms2 gives similar or slightly larger speedups than ms3 but can incur a large error.

Fig. 7 shows the following regarding ms1. (1) Of the n^2N cells available, only $f(n)N$ (where $f(n)$ is a sublinear function of n) are used, i.e., visited by an iterate, independently of the image and σ. (2) The average number of iterations per pixel k (= total number of iterations $/N$) approximately equals $f(n)$, independently of the image size N. The sublinear nature of $f(n)$ is also apparent in fig. 3. The sparse usage of cells results from most iterates lying around the modes and ridges leading to modes. These are travelled by the first few paths, sparing later paths from travelling them again. The remaining areas are sparsely populated by iterates, typically a single one per pixel. Since the total number of iterations roughly equals the number of visited cells (excluding the few cells which are visited more than once, usually by the first few pixels), the average number of iterations becames $f(n)$. Thus, ms1's effective memory cost is less than nN cells (with $n \approx 3$–6) and its computational cost is $\mathcal{O}(kN^2)$, like that of GMS, but where the average number of iterations per pixel is $k \approx f(n)$, i.e., 2–4, instead of 30–150 as for GMS.

To study the effect of image size, we ran all experiments with half-size images and correspondingly half-size σ. The results were very similar to the full-size images (same optimal parameters, similar curves) but the number of iterations was about $4\times$ smaller, indicating that the average number of iterations per pixel is the same.

We also ran all experiments with a more accurate convergence criterion, 10^{-6} pixels rather than 10^{-3}. The resulting curves were almost identical except that the number of iterations for ms, ms2 and ms3 roughly doubled, while for ms1 and ms4 it remained approximately equal. The reason is that the former methods have a linear rate of convergence (being essentially EM steps), while ms4 has quadratic convergence and ms1 runs to convergence only a handful of points. Thus, ms1 and ms4 can be ran at a higher accuracy level for free (though for image segmentation a tolerance of 10^{-3} is generally good enough).

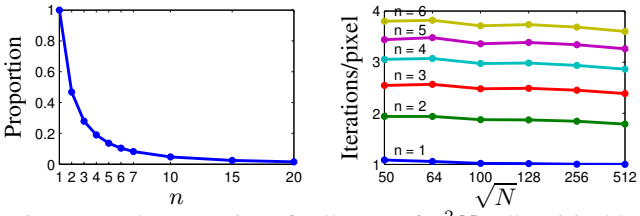

Figure 7. *Left*: proportion of cells (out of n^2N cells) visited by ms1, as a function of the discretisation level n, for $\sigma = 16$ (very similar curves were obtained for other bandwidths/images). The curve decreases faster than $1/n$, so less than nN cells are visited. *Right*: average number of iterations per pixel for ms1 as a function of image size N. Each curve corresponds to a different n and is roughly constant, with a value $f(n)$ that grows sublinearly with n.

7. Discussion and related work

The acceleration strategies we have proposed can be classified in 3 types: discretisation methods (ms1), neighbourhood methods (ms2 and ms3) and hybrid EM–Newton (ms4). Neighbourhood methods reduce the computational cost per iteration while the other methods reduce the number of iterations. All these strategies are new except ms3, which is the sparse EM algorithm of [13], though our adaptation to GMS and our rule when to take full steps is new. Our general conclusions regarding performance are as follows. (1) With parameters set optimally, all methods can obtain nearly the same segmentation as GMS with significant speedups; near-optimal parameter values can easily be set in advance for ms1, ms3 and ms4, and somewhat more heuristically for ms2. (2) ms1 attains the largest speedup by far (10×–100× with clustering error $< 3\%$, depending on the image and σ). It reduces the computational cost $\mathcal{O}(kN^2)$ of GMS by reducing k from 30–150 (the average number of iterations per pixel with GMS) to 2–4, yielding speedups of one to two orders of magnitude for a typical image. Besides, if desired the error can be further reduced at a small additional cost. (3) ms4 ranks second best, with 1.5×–6× speedups. (4) The neighbourhood methods attain more modest speedups (1×–3×); of these, ms3 attains the lowest (near-zero) error of all strategies, while ms2 can result in unacceptably large errors for a suboptimal setting of its parameter. For the larger bandwidths, the neighbourhood methods do not improve over GMS, since the neighbourhood size becomes comparable to the data set size. (5) Some of these methods are orthogonal to each other (e.g. ms1 and ms4) so it is possible to use them in combination and obtain larger speedups.

Earlier proposals of neighbourhood methods with mean-shift [6, 9] or EM algorithms [12] have focused on fast, approximate algorithms for range search such as kd-trees or locality-sensitive hashing. However, range search algorithms scale poorly with the dimension (though they do work well up to, say, dimension 5) and with the number of neighbours requested. The latter is particularly problematic with GMS, which requires bandwidths of the order of 0.1–0.3 the image side and thus very large neighbourhoods. In contrast, `ms2` avoids the range search cost altogether (using instead a fixed pixel neighbourhood), while `ms3` searches for neighbours only at full EM steps, which are infrequent.

Another approach to accelerate GMS is to use a faster but altogether different optimisation algorithm on the Gaussian kernel density estimate (e.g. a quasi-Newton method [14]). While this is a valid idea and may produce good clusterings, it is also likely that the segmentation will differ significantly from that produced with GMS. Preserving the modes' attraction basins was our rationale to use a hybrid EM–Newton algorithm (`ms4`) that starts with EM steps (which direct the iterate towards the right mode) and switches to Newton steps later on, achieving quadratic convergence (which provides a high accuracy at almost no additional cost). In fact, forcing the very first step in `ms4` to be a Newton step rather than an EM step consistently increases the error between 1 and 10 percentage points (the error happening mainly at pixels near cluster boundaries).

The fast Gauss transform (FGT) [7, 10] approximately evaluates a sum of N Gaussians such as eq. (1) at M points in $\mathcal{O}(M+N)$ time rather than the naive $\mathcal{O}(MN)$. The FGT could be combined with any of our techniques, since its improvement is orthogonal to them. Unfortunately, the constant in its computation order grows exponentially with the dimension D (essentially, the number of monomials in a polynomial series in D variables grows exponentially with D for a fixed approximation error). For image data set sizes ($N \lesssim 10^6$), the FGT is competitive with naive evaluation for $D \leq 3$ only. An improved FGT has been recently proposed [15] which handles up to $D = 10$.

8. Conclusion

We have proposed four strategies, all very easy to implement, to accelerate Gaussian mean-shift segmentation (using spatial and range features). The best of these, based on discretising the spatial domain, achieves 10×–100× speedups with a very small error (controlled by the discretisation level). This will facilitate the practical application of GMS, which attains better segmentations than finite-support kernels but is much more computationally costly in a naive implementation. In practice, one may often postprocess the segmentation (e.g. to remove small clusters), which may allow to run the accelerated method at a coarser discretisation level and further reduce the computational cost. The other strategies produced much smaller speedups: up to 6× for the EM–Newton algorithm and up to 3× for neighbourhood-based strategies (spatial neighbourhood, sparse EM). All the methods are readily extended to the case where the bandwidth is adaptive (dependent on the pixel) and non-isotropic. The sparse EM and EM–Newton algorithms are also applicable to clustering non-image data.

Acknowledgements This work was partially supported by NSF CAREER award IIS–0546857.

References

[1] M. Á. Carreira-Perpiñán. Mode-finding for mixtures of Gaussian distributions. *IEEE Trans. PAMI*, 22(11):1318–1323, Nov. 2000.

[2] M. Á. Carreira-Perpiñán. Gaussian mean shift is an EM algorithm. *Submitted*, 2005.

[3] M. Á. Carreira-Perpiñán and C. K. I. Williams. On the number of modes of a Gaussian mixture. In L. Griffin and M. Lillholm, editors, *Scale Space Methods in Computer Vision*, vol. 2695 of *Lecture Notes in Comp. Sci.*, pages 625–640, 2003.

[4] Y. Cheng. Mean shift, mode seeking, and clustering. *IEEE Trans. PAMI*, 17(8):790–799, Aug. 1995.

[5] D. Comaniciu and P. Meer. Mean shift: A robust approach toward feature space analysis. *IEEE Trans. PAMI*, 24(5):603–619, May 2002.

[6] D. DeMenthon. Spatio-temporal segmentation of video by hierarchical mean shift analysis. In *Statistical Methods in Video Processing Workshop (SMVP 2002)*, Copenhagen, Denmark, June 1–2 2002.

[7] A. Elgammal, R. Duraiswami, and L. S. Davis. Efficient kernel density estimation using the fast Gauss transform with applications to color modeling and tracking. *IEEE Trans. PAMI*, 25(11):1499–1504, Nov. 2003.

[8] K. Fukunaga and L. D. Hostetler. The estimation of the gradient of a density function, with application in pattern recognition. *IEEE Trans. Info. Theory*, IT–21(1):32–40, Jan. 1975.

[9] B. Georgescu, I. Shimshoni, and P. Meer. Mean shift based clustering in high dimensions: A texture classification example. In *ICCV*, pages 456–463, 2003.

[10] L. Greengard and J. Strain. The fast Gauss transform. *SIAM J. Sci. Stat. Comput.*, 12(1):79–94, Jan. 1991.

[11] G. J. McLachlan and T. Krishnan. *The EM Algorithm and Extensions*. John Wiley & Sons, New York, 1997.

[12] A. W. Moore. Very fast EM-based mixture model clustering using multiresolution kd-trees. In *NIPS*, pp. 543–549, 1999.

[13] R. M. Neal and G. E. Hinton. A view of the EM algorithm that justifies incremental, sparse, and other variants. In M. I. Jordan, editor, *Learning in Graphical Models*, pages 355–368. MIT Press, 1998.

[14] C. Yang, R. Duraiswami, D. DeMenthon, and L. Davis. Mean-shift analysis using quasi-Newton methods. In *ICIP*, pages 447–450, 2003.

[15] C. Yang, R. Duraiswami, N. A. Gumerov, and L. Davis. Improved fast Gauss transform and efficient kernel density estimation. In *ICCV*, pages 464–471, 2003.