Complexity Reduction for "Large Image" Processing

Nikhil R. Pal, *Senior Member, IEEE,* and James C. Bezdek, *Fellow, IEEE*

***Abstract*—We present a method for sampling feature vectors in large (e.g., 2000 $\times$ 5000 $\times$ 16 bit) images that finds subsets of pixel locations which represent c "regions" in the image. Samples are accepted by the chi-square (χ^2) or divergence hypothesis test. A framework that captures the idea of *efficient extension* of image processing algorithms from the samples to the rest of the population is given. Computationally expensive (in time and/or space) image operators (e.g., neural networks (NNs) or clustering models) are trained on the sample, and then extended noniteratively to the rest of the population. We illustrate the general method using fuzzy c-means (FCM) clustering to segment Indian satellite images. On average, the new method can achieve about 99% accuracy (relative to running the literal algorithm) using roughly 24% of the image for training. This amounts to an average savings of 76% in CPU time. We also compare our method to its closest relative in the group of schemes used to accelerate FCM: our method averages a speedup of about 4.2, whereas the multistage random sampling approach achieves an average acceleration of 1.63.**

***Index Terms*—Accelerated fuzzy c-means (AFCM), algorithmic extensibility, complexity reduction in large images, image sampling.**

I. Introduction

LET $F = [f_{ij}]_{I\times J}$ be a digital image with intensity value f_{ij} at pixel (i, j), $f_{ij} \in \mathcal{G} = \{0, 1, \ldots, L-1\}$. The time and space complexity associated with processing F (filtering, segmentation, edge detection, etc.) increases with I, J or L. We call (I, J, L) the *complexity triple* associated with F.

Ten years ago, an 8-bit image with 65 536 pixels, (I, J, L) = (256, 256, 256), was considered "large," and presented time and space problems to the computers of the early 1990s. Today "large" images typically have complexity triples on the order of (2000, 5000, 65 536), which is the complexity triple for many of the images in the *digital database for screening mammography (DDSM)* [1]. As a second example, images taken by the Indian remote-sensing satellite (IRS-1A, IRS-1B) contain 2500 scan lines with 2520 pixels per scan line. Each image requires 6 MB of memory (assuming each pixel requires 1 byte). Normally, for each scene there are four spectral bands resulting in four images, each of size 6 MB. Therefore, each frame consists of 24 MB of data. Segmentation of such an image into regions is often very useful. However, when fuzzy c-means (FCM)-type clustering algorithms are used, for example, if there are five classes then the memory required for just one partition matrix (assuming a 4-byte representation of reals) will be approximately $6 \times 5 \times 4 =$ 120 MB. (Not all segmentation algorithms maintain partition matrices.)

Manuscript received September 10, 2000; revised July 2, 2001 and January 14, 2002. This work was supported by ONR Grant 00014-96-0642. This paper was recommended by Associate Editor L. O. Hall.

N. R. Pal is with the Electronics and Communication Sciences Unit, Indian Statistical Institute, Calcutta 700108, India (e-mail: nikhil@isical.ac.in).

J. C. Bezdek is with the Department of Computer Science, University of West Florida, Pensacola, FL 32514 USA (e-mail: jbezdek@uwf.edu).

Publisher Item Identifier S 1083-4419(02)04807-0.

DDSM images seem big today (so big that printing or displaying a single image at full resolution is almost impossible), but technological advances will soon "shrink" them to reasonable proportions, just as (256, 256, 256) images are manageable now. However, new technology also whets the appetite for new sensors which will produce concomitantly larger and larger complexity triples. The technology cycle will probably continue to limit "large image" processing at full resolution in near real-time speeds for the next few decades. Processing of large data sets such as these cannot be done in a reasonable time. Hence, there is continuing need for methods that effectively "reduce" the complexity triple of large images. Even when large images can be brought into memory, the computations to process such images can be very high. This creates a bottleneck even for "near-real-time" applications, such as online weld-defect detection and control of the welding process for fixing the defects [2], [3].

Processing of very large databases, particularly for data-mining applications, poses many other practical problems. For example, in applications such as document categorization, it is desirable to apply clustering to *huge* data sets (terabytes of data). Usually, such data sets cannot be brought into main memory for processing, and hence, the use of objective function driven clustering algorithms like *fuzzy c-means* (FCM) is either very time-consuming or impossible to use. For such applications, many heuristic algorithms have been developed, which usually make one pass (or very few passes) through the data [5]–[11]. None of the algorithms in [5]–[11] optimize an objective function.

In [5], Fayyad and Smyth provide a nice discussion of the problems faced while processing massive data sets. They suggested an iterative scheme where they first generate a random sample S from the entire data set X. Next, they construct a model M_S (apply a probabilistic clustering algorithm) for S. Then, they apply M_S to the entire data X and find the residual points R from X, which do not fit any of the clusters with a high probability. If R is reasonably large, then clustering can be performed on it again, or else a sample from R can be selected and the entire process can be repeated. The terminal number of clusters may be greater than the initial number of clusters, and the scheme involves various choices such as initial sample size, threshold to decide on high membership, etc.

Ganti *et al.* [9] considered clustering of large databases in "distance space" where the distance between two objects can be obtained by satisfying the triangle inequality. This method is not suitable when we want to compute cluster centroids (as we do when object data are represented by p-dimensional vectors). The authors in [9] propose two algorithms, namely, BUBBLE and BUBBLE-FM. In BUBBLE, the database is scanned only once and each object is inserted into one of the evolving clusters maintained as leaf nodes of a height balanced tree. BUBBLE supports cluster splitting when the distance between a new object and the closest cluster exceeds a threshold. To direct a new object, BUBBLE computes the distance between the new object and all sample objects at all internal nodes encountered on its way to a leaf node. To avoid this time-consuming process, these authors use the fast map (FM) algorithm of Faloutsos and Lin [10] in BUBBLE-FM algorithm.

Recently, Domingos and Hulten [12] proposed a statistical method that can be used to simulate clusters in a large data set using the crisp k-means algorithm based on samples. The method is based on using the Hoeffding bound [13] on the error in estimating the mean of a random variable based on n independent observations. Here, the error is defined as the deviation of the computed mean from the true mean that would be obtained if an infinite number of observations were used. The method can find the number of examples that should be used to attain a desired bound on the error. This interesting method can be used with any data set once the bound is worked out, but the procedure given in [12] is applicable only to the crisp k-means algorithm.

The problem that we are dealing with is a little different from the problem in [12], and a bound similar to the one given for crisp k-means is yet to be worked out. We deal with problems associated with processing large images, and hence, our problem must recognize and account for the *spatial* component associated with feature vectors extracted from an image. We present a general methodology for reducing both the time and space complexity of certain (but not all) image processing algorithms. Our method is also sample-based, but once the sample is selected, it can be used with any learning scheme (including k-means). This aspect of our method makes it attractive, as different types of calculations can de done on the selected sample; i.e., the sample obtained is independent of the processing that will be applied to it.

Our method is useful today, and will continue to be useful as I, J, and L increase. The basic idea begins with statistical hypothesis testing on random samples drawn from F. Simple statistical tests such as the chi-square (χ^2) or divergence test are used to assess the appropriateness of a sample. A sample that passes the test is processed, and the results are then "extended" to the rest of the image.

The remainder of this paper is organized as follows. Section II discusses feature extraction and our image sampling method. Section III defines the class of extensible algorithms for which our method is useful. Section IV is a case study of the application of image sampling and algorithmic extension for image segmentation by FCM clustering. Section V discusses the tradeoff between saved time and lost accuracy when using extensible algorithms. Section VI contains computational examples using satellite images that illustrate our new approach to complexity reduction. Finally, conclusions are given in Section VII.

II. Feature Extraction and Image Sampling

This section discusses feature extraction from two-dimensional images (or sets of them), and sampling methods for sets of feature vectors in images.

A. Features and Feature Extraction

The complexity triple (I, J, L) is associated with a single channel $I \times J$ image F with L gray levels. Many sensors produce suites of t images, $\mathbf{F} = (F_1, F_2, \ldots, F_t)$, that are collocated in time and space, and then (I, J, L) is associated with each of the F_ks. For example, a typical *magnetic resonance image (MRI)* has $t = 3$ gray values associated with spatial location (i, j): spin lattice relaxation time $(T1_{ij})$, transverse relaxation time $(T2_{ij})$, and proton density (ρ_{ij}). Let $\mathbf{f}_{ij} = (T1_{ij}, T2_{ij}, \rho_{ij}) \in \mathcal{G}^3, 1 \le i \le I, 1 \le j \le J$. We call $\mathbf{f}_{ij}$ a pixel-based *intensity vector* associated with (i, j). Many sensors produce multispectral images. For example, satellite images can have as many as $t = 15$ channels; in general, $\mathbf{f}_{ij} \in \mathcal{G}^t$.

Image processing algorithms often use pixel intensity values (f_{ij}) or vectors $(\mathbf{f}_{ij})$. However, many techniques depend on information possessed by features extracted from F (or $\mathbf{F}$). Haralick and Shapiro [14] provide an encyclopedic description of many features used for various image processing operations. Specifically, we may extract a *feature vector* $\mathbf{x}_{ij} = (x_{1,ij}, x_{2,ij}, \ldots, x_{p,ij})^T \in \mathcal{R}^p$ associated with spatial location (i, j) in F or $\mathbf{F}$. $\mathbf{x}_{ij}$ might be built from $\mathbf{f}_{ij}$ alone; or from $\mathbf{f}_{ij}$ and intensity vectors in a spatial neighborhood of (i, j). For example, if $\mathbf{f}_{ij} = (T1_{ij}, T2_{ij}, \rho_{ij})$, we can simply take $\mathbf{x}_{ij} = \mathbf{f}_{ij} \in \mathcal{G}^3$. If we define m_{ij} and sd_{ij} as, respectively, the average and standard deviation of $T1_{ij}$, $T2_{ij}$ or ρ_{ij} over some neighborhood of (i, j), $\mathbf{x}_{ij}$ could be $\mathbf{x}_{ij} = (m_{ij}, sd_{ij}) \in \mathcal{R}^2$; or it could be $\mathbf{x}_{ij} = (T1_{ij}, T2_{ij}, \rho_{ij}, m_{ij}, sd_{ij}) \in \mathcal{G}^3 \times \mathcal{R}^2$, etc.

When $\mathbf{x}_{ij} \in \mathcal{R}^p$ is extracted from $\mathbf{F}$ using $\mathbf{f}_{ij}$ and *some of its neighbors*, $\mathbf{x}_{ij}$ is a *window-based*feature vector. If (i, j) is regarded as a 1×1 window, then f_{ij} and $\mathbf{f}_{ij}$ are special cases of $\mathbf{x}_{ij}$. In the discussions that follow, we assume that each pixel in F (or $\mathbf{F}$) is equipped with a feature vector $\mathbf{x}_{ij}$ (possibly excluding border rows and columns of the image). We denote the feature vectors $\{\mathbf{x}_{ij}\}$ as X. Ignoring border effects, we assume $|X| = IJ$, so X and F are in one-to-one correspondence. Because of this, we sometimes refer to X as "the image," even though it is really a representation of F or $\mathbf{F}$.

B. Image Sampling

We want a sample X_s of X so that the corresponding set of pixels F_s adequately represents the *spatial distribution* of gray values on F. Since there is a one-to-one correspondence between F and X, and our sample should capture the spatial characteristics of F, the selection of X_s can be done by finding a sample $F_s \subset F$ using hypothesis tests.

An image F can be partitioned into a set of c "homogeneous" segments, $F_1, F_2, \ldots, F_c, F_i \bigcap F_j = \phi\ \forall i \ne j, \bigcup_{i=1}^{c} F_i = F$, and $|F_i| = n_i\ (\sum_{i=1}^{c} n_i = IJ)$ [15]. Each segment F_i contains a set of pixels that (hopefully) represents a "meaningful" part of the image, and the pixel values in each segment usually fall within a small range of gray values. In other words, appropriately chosen ranges of gray values correspond to different meaningful spatial segments of the image. We emphasize that our image sampling scheme is *not* intended for, nor is it restricted to, any particular application such as image segmentation. Therefore, the choice of c is not an issue here, and we will not discuss it. However, the concept of image segmentation provides some insight into the image sampling method, so we will explain how the samples can be selected if the *right* value of c and the actual c segments are known. Then we show how to extend the same sampling concept to the case when neither c nor the actual segments are known.

If we know F_i, we can draw a sample $F_{s,i}$ from F_i so that the gray level distribution over $F_{s,i}$ statistically matches that of F_i. This may be done by comparing the histogram of gray values over $F_{s,i}$ with that of F_i. Let P_i and $P_{s,i}$ be the probability distributions of gray values over F_i and $F_{s,i}$, respectively. If P_i and $P_{s,i}$ match statistically, then we can take $F_s = \bigcup_{i=1}^{c} F_{s,i}$ as a representative sample of F.

In most cases, neither c nor the F_is are known, but this does not pose a problem. We impose some pseudosegments on F by dividing $[0, L-1]$ into r cells, $\{C_i\}$, $c \leq r \leq L$. Although the number c of segments in the pattern recognition sense is not known, it is always possible to get a reasonable upper bound for c; i.e., we can always choose a $r > c$. For example, in the case of MR or CT images of a brain, the possible number of tissue types is seven or eight (if there is a tumor); similarly, for satellite images, it is not difficult to get a conservative estimate of the number of land-cover types. Let m_i be the count (frequency) of gray levels in image F corresponding to cell C_i, $\sum_{i=1}^{r} m_i = IJ$, and let $q_i = |F_{s,i}|/N = N_i/N =$ sample probability of a gray value falling in cell C_i; $N = \sum_{i=1}^{r} N_i$. The partitioning of the gray scale imposes a partition on F, $F = \{S_1, \ldots, S_r\}$, $S_i \neq \phi\ \forall i$, $S_i \cap S_j = \phi$, $i \neq j$, and $\bigcup_{i=1}^{r} S_i = F$. If a sample F_s of size N is drawn, and F_s is a good representative of F, then every segment S_i will be well represented in the sample, and N_i/N will be approximately equal to $m_i/IJ\ \forall i$. Next, we test the "goodness" of fit between $q = \{q_i = N_i/N; i = 1, 2, \ldots, r\}$ and $p = \{p_i = m_i/IJ; i = 1, 2, \ldots, r\}$. Once a representative sample of pixels F_s is accepted, we identify the corresponding set of feature vectors X_s associated with pixels in F_s.

Some questions still remain to be answered. *What value of r should be used? Should each cell have equal width?* A related question is: *Why not use $r = c$?* A final question is: *How do we test the match between the two probability distributions, p and q?* Usually, the number of meaningful segments in an image c is between two and ten. If the meaningful segments are known, we can consider each segment as a strata, and use a separate test of hypothesis on each segment to ensure that it is faithfully represented in the sample. However, neither c nor the c segments are known. Therefore, we need to test the hypothesis on a sample drawn from the entire image. Consequently, the use of even $r = 10$ cells to partition G may be too coarse to capture the desired level of spatial detail of the image in the sample. The value of r will impact the level of spatial detail that is supported by the sample. A high value of r usually includes more points in the sample, which provides better results but increases the computational cost. This tradeoff decision must be made by the designer. The best results, in terms of agreement between the parameters estimated from F_s and those from F, can be expected with $r = L$. However, when L is large, this can be a disadvantage. For example, a 16-bit image has $L = 65\,536$, but for practical purposes, taking $r = 256$ should be fine, as 256 segments are much more than the number of meaningful segments expected in an image.

We want to relate the sampling to the spatial distribution of gray values over different segments of the image, because the distinguished characteristics (features) of different segments are functions of the spatial distributions of gray values. For example, two binary textures having exactly the same gray level histograms may look completely different due to different spatial distributions of gray values over the pixels. This is the reason for selecting random samples from the image instead of from the histogram. We discuss problems that we can encounter if we choose the samples from the histogram. Let there be just four gray levels: 1, 2, 3, and 4, with frequencies $n_1 = 10\,000$; $n_2 = 20\,000$; $n_3 = 30\,000$; and $n_4 = 40\,000$, respectively. Now selection of one pixel with gray value 1, two pixels with gray value 2, three pixels with gray value 3, and four pixels with gray value 4 will be enough to satisfy a goodness-of-fit test such as the divergence test. The χ^2 test demands the minimum cell frequency to be five [16], [17]. Therefore, selection of 5, 10, 15, and 20 pixels with gray levels 1, 2, 3, and 4, respectively, would be enough to satisfy the χ^2 test. However, such a sample cannot represent the information content of any nontrivial image. The important point is that an image is not characterized by just its gray values, but also by the coordinates at which different gray values occur.

We represent the characteristics of a pixel by a feature vector, where the feature values are computed based on the neighboring pixels (for a single- or multi-channel image), or the feature values consist of corresponding pixel values of images of different channels (for a multi-channel image). Therefore, the *spatial locations* of the selected pixels along with their gray values are very important. Moreover, we need a faithful representation of each meaningful (unknown) segment of the image in the sample. Therefore, the sampling scheme should have pixels from each meaningful segment. Consequently, the sampling should be tied to the spatial distribution of gray values. This is similar in spirit to a stratified sampling scheme, but in stratified sampling, the different stratas are known. What we *do* know is that each strata *usually* corresponds to a range of gray levels which is associated with a spatial region on the image plane. Therefore, if we draw samples spatially and the gray level distribution in the sample closely matches the gray level distribution over the entire image, then we expect the sample to adequately represent each meaningful segment in the image. Therefore, we define an initial threshold,e.g., 1%, for the number of samples. Thus, in the 4-level image example discussed above, we will first select 1000 pixels. If the selected pixels are not uniformly distributed over the *spatial* lattice, the goodness-of-fit test may not be satisfied, so we will increase the number of samples. Regarding cell width, an unbiased choice would be to use equal widths (except for boundary cells, which could be different). However, the observed data may dictate pooling, which creates unequal widths, even though we intend to have (and start with) equal width cells.

For the goodness-of-fit test, we make the following assumptions.

- F_s is a random sample of N independent and identically distributed observations (spatial locations). Each observation is characterized by its gray value.
- The gray values associated with these locations can be classified into r nonoverlapping categories that exhaust all classification possibilities. That is, the categories are

mutually exclusive and exhaustive. The number of gray values falling into a given category is called the *observed frequency* of the category.

We want the random sample F_s drawn from F to reflect its characteristics [18]. Thus, we test the goodness-of-fit between the observed (based on F_s) and expected (based on F) frequencies for the r categories. This can be done by testing the following hypothesis:

H_0: the sample has been drawn from a population that follows the distribution of gray values in F.

against:

H_-: the sample has not been drawn from a population that follows the distribution of gray values in F.

Agreement between the observed and expected frequencies can be measured using the χ^2 *test* for goodness-of-fit or the *divergence* between two probability distributions [17], [19]. We briefly describe these two test statistics.

Chi-Square (χ^2): The chi-square test statistic (χ^2) measures the agreement (or disagreement) between sets of observed (O_i) and expected (E_i) frequencies. This statistic is computed as

$$\chi^2 = \sum_{i=1}^{r} \frac{(O_i - E_i)^2}{E_i}. \tag{1}$$

Divergence: The divergence between two probability distributions q (obtained from the sample F_s of N independent observations) and p (representing the population F) measures the difficulty of discriminating between the two distributions. Let $p = \{p_1, p_2, \ldots, p_r\}$, $q = \{q_1, q_2, \ldots, q_r\}$, $\sum_{k=1}^{r} p_k = \sum_{k=1}^{r} q_k = 1$. Here, p_k = population probability of the kth cell $= E_k/IJ = m_k/IJ$ and q_k = sample probability of the kth cell $= O_k/N = N_k/N$. The divergence is computed as

$$J(p, q) = N \sum_{j=1}^{r} (p_j - q_j) \log(p_j/q_j). \tag{2}$$

Note that $J(p, q) \geq 0$, with equality if and only if $p_j = q_j$, $j = 1, 2, \ldots, r$.

For large samples, χ^2 or $J(p, q)$ is distributed approximately as χ^2 with $r - 1$ degrees of freedom (DOF). Thus, if the computed value of χ^2 or $J(p, q)$ is equal to or greater than the tabulated value of χ^2 for $r - 1$ DOF at the significance level α, we reject the null hypothesis at that significance level.

The application of χ^2 to a sample of size N can be called "adequate" in most practical applications provided none of the expected frequencies is too small. When the expected frequency of a particular category is less than five, then that category is pooled with an adjacent category [16], [17]. Pooling is carried out until the minimum frequency requirement is met. When categories are pooled, we must recompute the DOF based on the new number of categories. Thus, the number of DOF for the χ^2 and divergence tests may differ for the same F_s.

The χ^2 tests the significance of the discrepancy between the observed and expected frequencies. Thus, the basic objective is to check whether the fit is open to suspicion. Let α be the probability that χ^2 shall exceed any specified value. Then, for the computed value of χ^2, if α is between 0.1 and 0.9, there is no reason to suspect the hypothesis [20]. For such tests, very low values of χ^2 ($\alpha > 0.999$) do not necessarily give more confidence about the hypothesis because generally this situation arises due to the use of asymptotic formulae when the number of DOF is large [20].

However, the present situation is a little different from the usual test for goodness-of-fit. We want to get a sample F_s which is as representative as possible of the population F. In this case, the higher the value of α (even $\alpha > 0.999$), i.e., the lower the value of χ^2, the better the sample is for our purposes. Since it is common to accept the hypothesis if α is greater than 0.05, we may also accept the sample if the computed χ^2 is such that α is greater than 0.05 (acceptance at the 5% level of significance). For the problem at hand, a more conservative approach would be to set the level of significance at a higher value. The higher the value of α, the closer the distributional characteristics of F_s to F. For example, if F_s is selected at $\alpha = 0.95$, then the difference between the two probability distributions is negligible and hence, the difference between the sets of parameters estimated by X_s (corresponding to F_s) and that by X (associated to F) is expected to be negligible as well. Our computational experiments confirm this.

III. Extensible Algorithms

In this section, we divide image processing operations into two families: algorithms which are *efficiently extensible* from X_s to $X - X_s$; and those which are not. Let $\mathbf{A}$: $\mathcal{R}^p \rightarrow \mathcal{R}^q$ be an image processing algorithm. Since X, X_s, and $X - X_s$ are subsets of $\mathcal{R}^p$, we can represent most (but not all) image processing operations with functions of this form. When $\mathbf{x}_{ij} \subset \mathcal{R}^p$ is the feature vector associated with spatial location (i, j) in an input image F, $\mathbf{y}_{ij} = \mathbf{A}(\mathbf{x}_{ij})$ is the result associated with the same location in an output image $\mathbf{A}[F]$, produced by applying $\mathbf{A}$ to every $\mathbf{x} \in X$.

Some functions used in image processing are "one-pass" operators; that is, $\mathbf{A}$ does not depend on parameters that must be estimated with training data before $\mathbf{A}[F]$ is created. One-pass functions are *not* efficiently extensible. An example of this type is the Sobel edge operator [21]. Let $\mathbf{x}_{ij}$ be the vector $(f_1, f_2, f_3, f_4, f_5, f_6, f_7, f_8, f_9)^T \subset \mathcal{R}^9$ of intensities in a 3×3 window whose center pixel is location $(i, j) = 5$, where the window pixels are indexed left to right, then top to bottom, (see Fig. 2, Section VI). Next, define an estimate of the gradient vector of the underlying picture function at (i, j) to be $\mathbf{z}_{ij} \subset \mathcal{R}^2$ as follows:

$$\begin{aligned} z_{1,ij} &= B_1(\mathbf{x}_{ij}) \\ &= (f_9 + 2f_8 + f_7) - (f_1 + 2f_2 + f_3) \end{aligned} \tag{3a}$$

$$\begin{aligned} z_{2,ij} &= B_2(\mathbf{x}_{ij}) \\ &= (f_3 + 2f_6 + f_9) - (f_1 + 2f_4 + f_7). \end{aligned} \tag{3b}$$

$z_{1,ij}$ is an estimate of the gradient at (i, j) in the vertical direction, while $z_{2,ij}$ is the same in the horizontal direction. Equations (3a) and (3b) are compactly represented by writing $\mathbf{z}_{ij} = \mathbf{B}(\mathbf{x}_{ij})$, where $\mathbf{B} = (B_1, B_2)$: $\mathcal{R}^9 \rightarrow \mathcal{R}^2$ and $\mathbf{z}_{ij} = (z_{1,ij}, z_{2,ij})^T$. There are *infinitely many* Sobel edge detectors based on $\mathbf{z}_{ij}$ [22]. The most common Sobel operator uses the Euclidean norm of $\mathbf{z}_{ij}$ to estimate the magnitude of the gradient at pixel (i, j)

$$y_{ij} = N_2(\mathbf{z}_{ij}) = \sqrt{z_{1,ij}^2 + z_{2,ij}^2}. \tag{4}$$

The (Euclidean) Sobel operator A_2 can be defined as the composition of $\mathbf{B}$ followed by N_2

$$y_{ij} = A_2(\mathbf{x}_{ij}) = (N_2 o \mathbf{B})(\mathbf{x}_{ij}). \tag{5}$$

Applying A_2 to the 9-vectors $\{\mathbf{x}_{ij}\}$ extracted from F results in the Sobel edge image $A_s[F] = [y_{ij}]_{I \times J}$. Thresholding $A_2[F]$ produces a standard black and white Sobel edge image.

The application of A_2 to the images X, X_s, and $X - X_s$ produces the images $A_2[X]$, $A_2[X_s]$, and $A_2[X - X_s]$, respectively. If F is $I \times J$, $|X| = (I-2)(J-2)$ because (3) cannot extract 9-vectors for the border rows and columns of F. Nonetheless, if X is extracted from F before the application of A_2, we have $A_2[X] = A_2[X_s] \cup A_2[X - X_s]$. The time it takes to produce $A_2[X]$ by first splitting X into $X_s \cup (X - X_s)$ is actually *longer* than the time needed to compute $A_2[X]$ directly from X (because it takes time to create and test X_s). Consequently, working with subimages, as developed in Section II for "one-pass" functions such as the Sobel operator A_2, cannot reduce (and actually *increases*) their time and space complexity. This is an example from the class of image processing operators which are *not* efficiently extensible from X_s to $X - X_s$.

On the other hand, many imaging operators depend on a set of unknown parameters, e.g., $\boldsymbol{\theta}$, in some parameter space Ω. In this case, multiple passes (usually iterations) through X or some other set of *training data* X_{tr} are needed to estimate (or learn) a "best" $\hat{\boldsymbol{\theta}} \in \Omega$ before the output image can be constructed. We indicate the dependence of $\mathbf{A}$ on $\boldsymbol{\theta}$ and X_{tr} by writing $\mathbf{A}(X_{tr}; \boldsymbol{\theta})$. Suppose we find an optimal (in some well-defined sense) set of parameters $\hat{\boldsymbol{\theta}}$, and once found, the vector $\mathbf{y} = \mathbf{A}(\mathbf{x}; \hat{\boldsymbol{\theta}})$, $\mathbf{x} \notin X_{tr}$, can be calculated noniteratively. We cannot expect that $\mathbf{A}(\mathbf{x}; \hat{\boldsymbol{\theta}}) = \mathbf{A}(\mathbf{x}; \boldsymbol{\theta}^*)$, where $\boldsymbol{\theta}^*$ is the optimal set of parameters found when $\mathbf{A}$ is trained with $X^* = X_{tr} \cup \{\mathbf{x}\}$, but it may be the case that $\mathbf{A}(\mathbf{x}; \hat{\boldsymbol{\theta}}) \approx \mathbf{A}(\mathbf{x}; \boldsymbol{\theta}^*)$. For convenience, we let $\hat{\mathbf{A}} = \mathbf{A}(X_{tr}; \hat{\boldsymbol{\theta}})$, $\mathbf{A}^* = \mathbf{A}(X; \boldsymbol{\theta}^*)$ in what follows. If there are several $\mathbf{x}s \notin X_{tr}$, the time saved by calculating the $\hat{\mathbf{A}}(\mathbf{x})$s noniteratively may be worth the sacrifice in accuracy due to the approximation of $\mathbf{A}^*(\mathbf{x})$ by $\hat{\mathbf{A}}(\mathbf{x})$. We call $\mathbf{A}: \mathcal{R}^p \rightarrow \mathcal{R}^q$ an *extensible function* when

$$\mathbf{A} = \mathbf{A}(X_{tr}; \boldsymbol{\theta}) \text{ depends on } \boldsymbol{\theta} \in \Omega. \tag{6a}$$

$$\hat{\boldsymbol{\theta}} \text{ requires iterative estimation using } X_{tr}. \tag{6b}$$

$$\hat{\mathbf{A}}(\mathbf{x}) \text{ can be calculated noniteratively for } \mathbf{x} \notin X_{tr}. \tag{6c}$$

$$\text{If } \boldsymbol{\theta}^* \text{ is optimal for } X \supset X_{tr} \text{ then } \hat{\mathbf{A}}(\mathbf{x}) \approx \mathbf{A}^*(\mathbf{x}) \, \forall \mathbf{x} \in X. \tag{6d}$$

In the present context, let $X_{tr} = X_s$, the data set obtained by sampling X, as described in Section II. Let t_{X_s} denote the overall time it takes to: 1) construct X_s; 2) estimate $\hat{\boldsymbol{\theta}}$ for $\hat{\mathbf{A}}$; and 3) compute $\{\hat{\mathbf{A}}(\mathbf{x}): \mathbf{x} \in X - X_s\}$. Let t_X be the overall time it takes to: 1) estimate $\boldsymbol{\theta}^*$ for $\mathbf{A}^*$; and 2) compute $\mathbf{A}^*[X]$ (this time can be zero, depending on $\mathbf{A}$). Since there is no guarantee that $\hat{\mathbf{A}}(\mathbf{x}) = \mathbf{A}^*(\mathbf{x})$ for $\mathbf{x} \in X - X_s$, we have at best $\mathbf{A}^*[X] \approx \hat{\mathbf{A}}[X_s] \cup \hat{\mathbf{A}}[X - X_s]$. Our hope is that the loss of accuracy due to extending $\mathbf{A}$ to $X - X_s$ with $\hat{\theta}$ is balanced by a decrease in overall CPU time.

We have pointed out that A_2 would take less time on X than on $X_s \cup (X - X_s)$; therefore, this is an example where $(t_X/t_{X_s}) < 1$. Whenever the ratio is less than one or even close to one, there is little merit in estimating $\hat{\boldsymbol{\theta}}$ and using it to extend $\mathbf{A}$ to $X - X_s$. On the other hand, t_{X_s} can be significantly less than t_X for computationally intensive $\mathbf{A}$s (neural networks (NNs), clustering methods, vector quantization, etc.). For algorithms of this kind, $(t_X/t_{X_s}) \rightarrow \infty$ as $(I, J, L) \rightarrow \infty$, so the loss of accuracy incurred by using $\hat{\mathbf{A}}[X]$ for $\mathbf{A}^*[X]$ may be more than offset by the overall reduction in CPU time.

Since $\mathbf{A}$ at this point is quite general, we cannot give $\mathbf{O}(IJ)$ estimates of t_{X_s} or t_X, so it seems reasonable to use the ratio $(t_X/t_{X_s}) \doteq t_{acc}$ as a measure of the *observed efficiency* of extensible image processing algorithms. Nonextensible algorithms yield $t_{acc} \leq 1$ (as is the case for A_2, the Euclidean Sobel edge operator). Conversely, the efficiency of extending $\hat{\mathbf{A}}$ to $(X - X_s)$ with $\hat{\boldsymbol{\theta}}$ increases as $t_{acc} \rightarrow \infty$. We call t_{acc} the *acceleration factor* achieved by the use of sampling and extension, and say that $\mathbf{A}$ is *efficiently extensible* if and only if its acceleration factor is greater than one

$$\mathbf{A}: \mathcal{R}^p \rightarrow \mathcal{R}^q \text{ is efficiently extensible} \Leftrightarrow t_{acc} > 1. \tag{6e}$$

The questions that need to be answered regarding efficiently extensible imaging operations are: *How much time do they save?* and *How bad is the approximation* $\hat{\mathbf{A}}[X] \approx \mathbf{A}^*[X]$? Generally, the answers will depend on the image to be processed and the $\mathbf{A}$ to be used, as well as (I, J, L). $\mathbf{A}$s that seem amenable to our efficient extension method usually spend a lot of time computing distances in $(X \times X) \subset (\mathcal{R}^p \times \mathcal{R}^p)$, so t_{acc} often depends implicitly on p, the number of features extracted from F. Furthermore, t_{acc} can also depend on the number of classes c as well as the ratio $|X_s|/|X| = N/IJ$. Therefore, in general, t_{acc} is a quite complex function, $t_{acc} = \Phi(I, J, L, p, N, c)$, of at least six integers. Exact complexity analysis may be possible for some $\mathbf{A}$s, but will be impossible for most. The best one may be able to do is measure the acceleration factor t_{acc} for actual trials, and assess the loss due to using output image $\hat{A}[X]$ instead of $A^*[X]$ in terms of disagreement between the two (i.e., average error), as well as visually. This will be demonstrated in Section VI.

Before we present our computational study, we offer some qualitative remarks about efficient extensibility for large-scale images. Our method is most appropriate for operations on images where iterative estimation (learning) is necessary, and where $\hat{\mathbf{A}}$ can be extended to $X - X_s$, as in (6c). Almost all classification functions represented by NNs and fuzzy systems fall into this category. For example, segmentation, edge detection, and classification by any supervised learning scheme [23]–[25] should benefit by extension as long as the approximation in (6d) is acceptable. The same remark applies to most unsupervised learning models used for image processing (e.g., segmentation and edge detection with clustering algorithms, when these are extensible).

Generally, it will be clear from the nature of $\mathbf{A}$ whether sampling F and extending $\hat{\mathbf{A}}$ to $X - X_s$ is a better strategy (when there is a choice) than finding $\mathbf{A}^*$ with X and computing $\mathbf{A}^*[X]$ directly. Furthermore, if F is so large that X (and possibly $X - X_s$) cannot be loaded into memory, *some* type of sampling scheme may be the *only* choice. Now we turn to an example of extensibility based on segmentation with clustering.

IV. FCM as an Extensible Algorithm for Image Segmentation

In this section, we illustrate the scheme outlined in Sections II and III by casting the FCM clustering algorithm in this framework. We use only one subscript to specify a pixel and its associated feature vector. Given a set of feature vectors X extracted from an image, many clustering algorithms can be used to segment F by clustering in X [24], [26]. FCM is frequently used for image segmentation; typical applications include medical [27]–[29] and satellite image analysis [30], [31].

Let $X = \{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N\}, \mathbf{x}_k \in \mathcal{R}^p$, be a finite data set; c $(2 \leq c < N)$ be the number of clusters; $\mathbf{V} = (\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_c)$, $\mathbf{v}_i \in \mathcal{R}^p$ be the set of cluster prototypes; and $U = [u_{ik}]_{c\times N}$ be a fuzzy c-partition of X. The element u_{ik} represents the membership of $\mathbf{x}_k$ in the ith cluster. Fuzzy partition matrices satisfy three constraints

$$0 \leq u_{ik} \leq 1, \qquad \sum_{i=1}^{c} u_{ik} = 1 \quad \forall k = 1, 2, \ldots, N,$$

and

$$0 < \sum_{k=1}^{n} u_{ik} < N \qquad \forall i = 1, 2, \ldots, c. \tag{7}$$

We denote the set of all $c \times N$ matrices that satisfy (7) by M_{fcN}. When $u_{ik} \in \{0, 1\}\ \forall i, k$ in (7), M_{fcN} reduces to M_{hcN}, the *crisp* $c \times N$ c-partitions of X. Each column of $U \in M_{hcN}$ is a crisp label vector in the set $N_{hc} = \{\mathbf{e}_1, \ldots, \mathbf{e}_c\} \subset \{0, 1\}^c$, $\mathbf{e}_i = (0, 0, \ldots, 1, 0, \ldots, 0)^T$, where the "1" occurs at the ith address. Each column of $U \subset M_{fcN}$ is a *fuzzy label vector*; i.e., a vector $\mathbf{u} = (u_{1k}, \ldots, u_{ck})^T$ whose entries satisfy the first two constraints in (7). The parameters that FCM estimates are U and $\mathbf{V}$. These are found by minimizing

$$J_\mu(U, \mathbf{V}) = \sum_{i=1}^{c} \sum_{k=1}^{N} (u_{ik})^\mu \|\mathbf{x}_k - \mathbf{v}_i\|_M^2 \tag{8}$$

where $\mu > 1$ and the inner product induced norm, that is, the distance between $\mathbf{x}_k$ and $\mathbf{v}_i$, is

$$\|\mathbf{x}_k - \mathbf{v}_i\|_M^2 = (\mathbf{x}_k - \mathbf{v}_i)^T M (\mathbf{x}_k - \mathbf{v}_i) \doteq d_{ikM}^2. \tag{9}$$

M in (9) is any $p \times p$ positive definite matrix. First-order necessary conditions for a local extremum of J_μ are well known [4]. The prototype must satisfy

$$\mathbf{v}_i = \frac{\sum\limits_{k=1}^{N} (u_{ik})^\mu \mathbf{x}_k}{\sum\limits_{k=1}^{N} (u_{ik})^\mu}, \qquad 1 \leq i \leq c. \tag{10}$$

To state the necessary conditions for $\{u_{ik}\}$, let $I_k = \{i | 1 \leq i \leq c, d_{ikM} = 0\}$ and $\overline{I}_k = \{1, 2, \ldots, c\} - I_k$. Thus, I_k is the set of indexes of centroids which are identical to some data point $\mathbf{x}_k$. If $I_k \neq \phi$, set $u_{ik} = 0\ \forall i \in \overline{I}_k$ and arbitrarily assign the remaining

$$u_{ik}\text{'s such that } \sum_{i \in I_k} u_{ik} = 1. \tag{11a}$$

$$\text{If } I_k = \phi,\ u_{ik} = \left(\sum_{j=1}^{c} \left(\frac{d_{ikM}}{d_{jkM}} \right)^{2/(\mu-1)} \right)^{-1}$$
$$1 \leq i \leq c,\ 1 \leq k \leq N. \tag{11b}$$

The FCM algorithm consists of guessing $\mathbf{V}$ (or U), and then alternating between (11) and (10) until either $\|U_{new} - U_{old}\|$ or $\|\mathbf{V}_{new} - \mathbf{V}_{old}\|$ is less than a user-specified *termination threshold* ϵ. FCM is an example of *alternating optimization (AO)*. Many other image processing operators are AO algorithms. For example, the E–M algorithm, when used to estimate the parameters of normal mixtures, is a statistically motivated AO method that is used heavily in image processing [32]. All AO algorithms are good candidates for efficient extension when dealing with large images, because they usually require intensive and iterative calculation during the training phase. We mention that Bezdek and Hathaway [40] have recently shown that, under fairly mild conditions on the objective function, *all* AO algorithms converge globally (from any initialization in the constraint space). Reference [40] also contains a statement of the local result, which guarantees q-linear convergence for AO when initialized sufficiently close to a solution. The proofs of these results can be found in [41].

Each calculation of U at (11) is a function of X_s and $\mathbf{V}$, and this is the function we will extend from X_s to $X - X_s$. Let $\mathbf{B}$ be a function that assigns a fuzzy label vector to $\mathbf{x}_k$, given a set of centroids $\mathbf{V}$; i.e., let $\mathbf{B}$: $\mathcal{R}^p \rightarrow N_{fc}$ be

$$\mathbf{B}(\mathbf{x}_k; \mathbf{V}) = (u_{1k}, \ldots, u_{ck})^T, \qquad k = 1, \ldots, N. \tag{12}$$

The value of u_{ik} in (12) is calculated with (11) for $1 \leq i \leq c$. When FCM terminates at an optimal pair $(\hat{U}, \hat{\mathbf{V}})$, $\mathbf{B}(\mathbf{x}; \hat{\mathbf{V}}) \doteq \hat{\mathbf{B}}(\mathbf{x})$ is used to generate fuzzy label vectors for each spatial location (pixel) in $X - X_s$.

Arranging these vectors as a matrix, e.g., $\hat{U}_e$, results in a $c \times (IJ)$ fuzzy partition of X which, after rearrangement of the columns of $\hat{U}_e$ and vectors in X, yields the block matrix

$$\hat{U}_e = \left[\hat{U}_{c\times N} | \tilde{U}_{c\times(IJ-N)}\right]_{c\times IJ}. \tag{13}$$

The first block of $\hat{U}_e$ is the partition $\hat{U}_{c\times N}$ found by applying FCM to X_s. The second block of $\hat{U}_e$ is $\tilde{U}_{c\times(IJ-N)}$, the matrix of memberships extended to $X - X_s$ using $\hat{\mathbf{B}}(\mathbf{x})$.

The matrix $\hat{U}_e$ is then used to approximate U^*, which is a part of an optimal pair $(U^*, \mathbf{V}^*)$ for J_μ at (8) found by applying FCM to *all* of X, instead of just X_s.

The last step needed to produce the desired segmentation of F is to "harden" each column of $\hat{U}_e$ or U^*. The usual (but by no means *only*) way to do this is to use the "max-membership" hardening rule; i.e., define $\mathbf{h}$: $N_{fc} \rightarrow N_{hc}$ as follows. Let $\mathbf{u} =$

$(u_{1k}, \ldots, u_{ck})^T \in N_{fc}$ be the fuzzy label vector for $\mathbf{x}_k \in X$. Then $h(\mathbf{u}) = \mathbf{e}_i \Leftrightarrow u_{ik} = \max_j\{u_{jk}\}$. Thus, $\mathbf{x}_k$ is assigned the cluster in which it has the maximum membership value. For example, if $c = 3$ and the membership vector for $\mathbf{x}_k$ is $\mathbf{u}_k = (0.4, 0.1, 0.5)^T$, then $\mathbf{h}(\mathbf{u}_k) = \mathbf{h}(0.4, 0.1, 0.5) = (0, 0, 1) = \mathbf{e}_3$, so the pixel corresponding to $\mathbf{x}_k$ is assigned to cluster 3. This $\mathbf{h}$ can be used to assign a hard cluster label to each pixel in the image.

In order to get the segmented image we consider c colors (or perhaps c gray levels) equally distributed in $\{0, 1, \ldots, L-1\}$ associated with the index set $\{1, 2, \ldots, c\}$. If the kth pixel has a cluster label i, we replace its pixel value by the ith color. Formally, let $G = \{L_1, L_2, \ldots, L_c\} \subset \mathcal{R}^c$ be the colors, and g: $N_{hc} \rightarrow G$, $g(\mathbf{e}_i) = L_i$, $1 \le i \le c$. Now define the function A: $\mathcal{R}^p \rightarrow \mathcal{R}$ by composition as $A = (g \circ \mathbf{h} \circ \mathbf{B})$. Then $A[X]$ represents a c-region segmentation of F that depends on the prototype set $\mathbf{V}$ as in (10) found by FCM. When $\hat{\mathbf{V}}$ is used, $\hat{A}[X]$ is the extended segmentation of F; when $\mathbf{V}^*$ is used, $A^*[X]$ is the "exact" segmentation of F. Now we are in a position to ask: *How much time is saved by computing* $\hat{A}[X]$ *instead of* $A^*[X]$*?* and *How well does* $\hat{A} \approx A^*$*?* These are the topics of Section V.

V. Trading Accuracy for Time

Two facts about FCM segmentation of digital images are well established: segmentations are good, but FCM is slow. There have been many attempts to accelerate the iterative loop defined by (10) and (11) [31], [33]–[36]. One class of methods abandon the search for $(U^*, \mathbf{V}^*)$, an optimal pair for J_μ on a complete data set. One subset of this type replaces "literal" (or exact) FCM (LFCM) with an approximation to it, and uses all of X. The other type of approximation scheme in this category uses LFCM, but alters the data set processed. Our method is of this latter type, using $\hat{U}_e$ to approximate U^* on all of X. A second class of methods obtain U^* by eventually running LFCM on the full data set, but attempt to reduce t_X by altering the usual random initialization scheme.

Cannon *et al.* proposed an approximate version of LFCM called *accelerated fuzzy c-means (AFCM)*, which is based on lookup tables to approximate distances, and used it to segment thematic mapper images [30], [31]. Hall *et al.* [27] and De La Paz *et al.* [33] used AFCM for segmentation of MRIs. Although AFCM has been used successfully in several applications (and does reduce the CPU time of LFCM by as much as a factor of 10), it has some limitations.

- AFCM does not satisfy the necessary conditions of FCM, and hence, AFCM iterates do not minimize any well-defined objective function.
- AFCM uses lookup tables, and it is not feasible to have complete logarithmic and exponential (lookup) tables stored in memory. Assumptions are used to limit table sizes, which, in turn, reduce the accuracy of intermediate values used by AFCM.

Cheng *et al.* [34] present a method called *multistage random sampling fuzzy c-means* (mrFCM). This scheme cuts down the computation by reducing the number of feature vectors and iterations used in the initializing stages of FCM calculations. LFCM is run on several small subsets of the data in an incremental fashion to get a set of good initial prototypes, and then LFCM is applied to the whole data set. Based on empirical studies, Cheng *et al.* report that mrFCM reduces CPU time on average by about 50%, but no space reduction is possible. Cheng *et al.* did not provide a guideline about how to terminate their incremental process.

Quite independently, but almost at the same time as Cheng *et al.*, Uma Shankar and Pal [35] proposed another multistage scheme for accelerating LFCM they called *fast fuzzy c-means (FFCM)*. FFCM first runs LFCM on a small sample S_0 of the entire data set X. Let the centroids obtained on S_0 be V_0. Now S_0 is enhanced by a small fraction of X to get S_1, and LFCM is run on S_1 to get the set of centroids V_1. FFCM then uses V_0 (and also V_1) to generate the partition matrix U_0^X (and U_1^X) on the entire data set X. Let the partition matrices generated on X in two successive stages be denoted by U_i^X and U_{i+1}^X, respectively. In FFCM, enhancement of the sample and the running of LFCM on the enhanced sample are continued until there is no significant difference between U_i^X and U_{i+1}^X; i.e., $\|U_i^X - U_{i+1}^X\|$ is very small. FFCM runs LFCM a number of times, but the centroids are expected to improve due to the use of a bigger sample at each step, so the number of iterations required by LFCM in increasingly higher stages is expected to decrease.

Unlike AFCM and mrFCM, FFCM never runs iteratively on the whole data set, so it reduces both computational time and storage space. However, FFCM in [35] did not use a statistical criterion to assess the quality of X_s as a good representation of X, as we proposed in Section II. Moreover, FFCM runs LFCM on different samples, while the method proposed in this paper runs LFCM only on one sample. We distinguish FFCM from the method developed here by calling our new technique *extensible fast fuzzy c-means (eFFCM)*.

Kamel and Selim [37] proposed two algorithms that update cluster prototypes or membership values more frequently than LFCM. In one algorithm, the cluster centers are updated after computing the membership values for each input vector. The second algorithm updates membership values after computing each centroid. Kamel and Selim report that, on average, the first algorithm is 1.23 times faster than LFCM, while the second algorithm is 1.07 times faster than LFCM.

Velthuizen *et al.* [29] proposed an algorithm called *split fuzzy c-means (SFCM)*, which is driven by the philosophy underlying incremental partitioning in iterative least square clustering [38]. SFCM starts with $c = 1$ cluster whose centroid is taken as the mean of all data points. The splitting process is initiated by choosing the next cluster centroid $\mathbf{v}_{c+1}$ as the data point having the maximum weighted sum of squared distance from all centroids. Thus

$$\mathbf{v}_{c+1} = \mathbf{x}_k \leftarrow \sum_{i=1}^{c} (u_{ik})^\mu d_{ik}^2 = \max_l \left(\sum_{i=1}^{c} (u_{il})^\mu d_{il}^2 \right). \quad (14)$$

Next, LFCM is run with these $c-1$ cluster centroids as initial prototypes. The process is repeated until the desired number of clusters is created. Velthuizen *et al.* observed that steps prior to the final run of LFCM can be viewed as an elaborate initialization scheme. The last run of LFCM may terminate quickly, but the total time required for SFCM may be more than running LFCM once at the desired number of clusters; and *every* step of

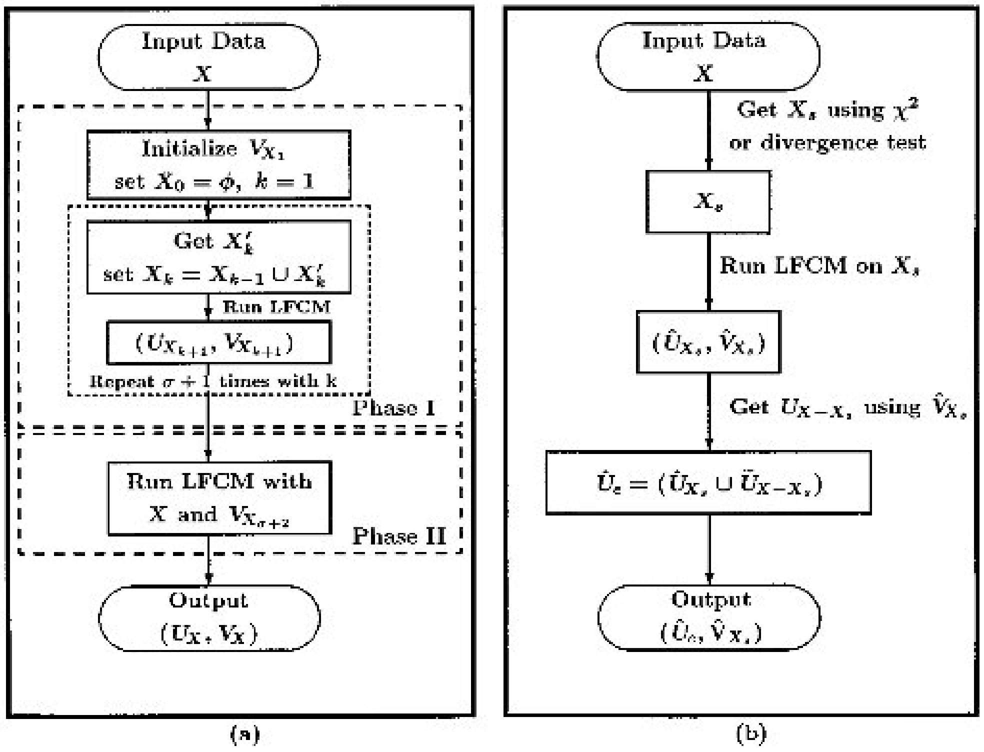
Input Data X
Initialize V_{X_1} set $X_0 = \phi$, $k = 1$
Get X'_k set $X_k = X_{k-1} \cup X'_k$
Run LFCM
$(U_{X_{k+1}}, V_{X_{k+1}})$
Repeat $\sigma + 1$ times with k
Phase I
Run LFCM with X and $V_{X_{\sigma+2}}$
Phase II
Output (U_X, V_X)
(a)
Input Data X
Get X_s using χ^2 or divergence test
X_s
Run LFCM on X_s
$(\hat{U}_{X_s}, \hat{V}_{X_s})$
Get U_{X-X_s} using $\hat{V}_{X_s}$
$\hat{U}_e = (\hat{U}_{X_s} \cup \hat{U}_{X-X_s})$
Output $(\hat{U}_e, \hat{V}_{X_s})$
(b)

Fig. 1. Architectures of mrFCM and eFFCM. (a) The mrFCM scheme. (b) The eFFCM scheme.

SFCM processes all of X. Among the various methods that have been studied to accelerate FCM, the one most similar to ours is mrFCM [34]. Consequently, we use mrFCM as a comparator in the numerical experiments of Section VI.

VI. Algorithms, Protocols, Data Sets, and Experiments

Since we will compare eFFCM to its closest relative (mrFCM), we now give a brief description of mrFCM. Cheng *et al.* [34] proposed a *two-phase* algorithm. The first phase has several *stages*; each stage uses an increasing number of feature vectors and a more stringent stopping condition. The first stage uses randomly initialized cluster prototypes while the ith $(i > 1)$ stage uses the final cluster prototypes from the $(i-1)$th stage. The second *phase* uses the final cluster prototypes produced by phase I and runs LFCM on the entire data set.

Algorithm mrFCM

```
Begin Phase I
Step 1. Choose: $c$, $\mu$, $A$, $\|.\|_{err}$ for LFCM; control parameters $\sigma$ and $\Delta$; termination parameters $\epsilon$ and $\epsilon_0$; and iterate limit $T$. Set $X_0 = \phi$.
Step 2. Choose cluster centers $\mathbf{V}_{X_1}$ randomly.
Step 3.
  For $k = 1$ to $\sigma + 1$:
    1. Select $\Delta$% of the $IJ$ feature vectors without replacement from $X$. A point is selected if no point in its 8-neighborhood is already selected, and the number of trials is less than $2IJ$. If the number of trials is greater than $2IJ$, samples are selected disregarding the 8-neighborhood condition. Call this set $X'_k$.
    2. $X_k = X_{k-1} \cup X'_k$.
    3. Using $X_k$ and initial guess $\mathbf{V}_{X_k}$, iterate LFCM through (11) and (10) for $b = 1, 2, 3, \ldots, T$ until either $b = T$ or $\|U^{(b)}_{X_k} - U^{(b-1)}_{X_k}\| < \epsilon_k$, where $U^{(b)}_{X_k}$ is the partition matrix at iteration $b$, and $\epsilon_k$ is the stopping condition for stage $k$, $\epsilon_k = \epsilon_0 - (k * (\epsilon_0 - \epsilon)/\sigma)$.
    4. $\mathbf{V}_{X_{k-1}} \leftarrow \mathbf{V}^b_{X_k}$, $U_{X_{k+1}} \leftarrow U^b_{X_k}$.
  Next, $k$.
End Phase I
Begin Phase II
Step 4. Initialize LFCM with the final cluster centers ($\mathbf{V}_{X_{\sigma+2}}$) from phase I, and iterate LFCM through (11) and (10) to termination on the whole data set $X$.
End Phase II
```

Fig. 1(a) shows the overall flow of mrFCM. mrFCM is a useful variation of LFCM, but as the authors of mrFCM point out, three critical, user-selected parameters (σ, ϵ_0, and Δ) influence the performance of mrFCM significantly; Cheng *et al.* give no guidelines for their selection. Moreover, mrFCM runs LFCM $\sigma + 1$ times on subsets of X, and then runs LFCM on the entire data set. Therefore, the role played by Steps 3.1–3.4 of mrFCM is really to ensure a good initialization of LFCM. Thus, while mrFCM *may* exhibit time reduction, it is by no means *guaranteed*. Finally, phase II of mrFCM requires exactly the same storage and (possibly) time as LFCM.

In eFFCM, we run LFCM on a small but representative subset X_s of the entire data set X. Initially a random sample F_l from F having l% of IJ points in F is selected, with replacement. If F_l does not pass the hypothesis test, it is enhanced by adding another Δl% of F to F_l. This process of enhancing the sample (with replacement) is continued until the hypothesis test accepts F_s. The sample X_s corresponding to F_s is then used to compute cluster prototypes $\hat{\mathbf{V}}$ with LFCM. This results in the (optimal for X_s) LFCM pair $(\hat{U}, \hat{\mathbf{V}})$. The cluster prototypes $\hat{\mathbf{V}}$ are then used with (12) to generate the remaining $|X - X_s|$ columns in $\hat{U}$ for vectors in $X - X_s$, resulting in $\hat{U}_e$, the fuzzy c-partition of X shown in (13). The matrix $\hat{U}_e$ is *not* part of an optimal pair for J_μ. If desired, a necessary pair $(U_e^*, \mathbf{V}_e^*)$ for J_μ *can* be generated from $\hat{U}_e$ by defining $U_e^* = \hat{U}_e$, and then calculating $\mathbf{V}_e^*$ noniteratively with U_e^* and (10). This way, LFCM is never applied to all of X.

When FCM terminates, take $\hat{U} = U^{(t-1)}$. Since eFFCM runs only once on a small fraction of X, the runtime storage space required for successive estimates of $\hat{U}$ is much less than that needed for estimation of U^* on X. Specifically, this reduces the storage requirement by about $c \times (IJ - N)$ addresses for $U^{(t)}$. For a 2000 × 5000 DDSM image, this reduction typically amounts to about 700 000 addresses, each needing up to 32 bits. The last step of eFFCM computes $\hat{U}_e$ once, and this space is required whether we use eFFCM or LFCM. Fig. 1(b) shows the block architecture of eFFCM.

Algorithm eFFCM

Step 1. Choose c, μ, A, $\|.\|_{err}$ and T for LFCM; and r, l, Δl and α for eFFCM. Here, r is the number of categories as defined in (1) and (2). The initial subset F_s of F contains $(l \times IJ)/100$ pixels from F, Δl is the additional percentage of samples added at each step, and α is the level of significance for the statistical test.

Step 2. $F_s = (l \times IJ)/100$ draws from F (with replacement).

Step 3. Compute χ^2 using (1) or $J(p, q)$ using (2) on F_s.

Step 4. Compare χ^2 or $J(p, q)$ with the tabulated value of α at $(r-1)$ DOF. If the test is significant at level α, then enhance F_s by adding an extra $(\Delta l \times IJ)/100$ draws (with replacement) from F to F_s and go to Step 3. Otherwise go to Step 5. [For χ^2, the DOF (df) may be less than $(r-1)$ if there are categories with expected frequencies less than 5.]

Step 5. Apply LFCM to X_s corresponding to F_s, obtaining the pair $(\hat{U}, \hat{\mathbf{V}})$.

Step 6. Use the cluster prototypes $(\hat{\mathbf{V}})$ generated at Step 5 with (12) to calculate $\hat{U}_e$ on all of X, i.e., $\hat{U}_e = \hat{U}_{X_s} \cup \hat{U}_{X-X_s}$.

A. Performance Comparison

It is not hard to show that LFCM and all of its variants discussed in Section V have the same asymptotic run-time complexity; e.g., $\mathbf{O}(cIJp)$ for IJ feature vectors in $\mathcal{R}^p$ divided into c fuzzy subsets. Asymptotically then, there is no advantage to our sampling and extension scheme over simply running FCM on X. However, the key word is "asymptotically." Since IJ is *always* less than infinity, a better indication of effective speedup is to compare the number of operations used for finite N, I, and J during the iterative phase of these algorithms. Comparing cNp to $c(IJ)p$, we see that the number of operations *per iteration* done by eFFCM as compared to LFCM stands in the ratio N/IJ. Our experiments indicate that $(100N)/IJ = 30\%$ usually produces good results for (256, 256, 256) images. For higher resolution images, like DDSM with $(I, J, L) = (2000, 5000, 65\,536)$, $IJ = 10^7 = 10$ million pixels in X, X_s is likely to be much less than 30% of X. Even for such images, if we assume that X_s is 30% of X, then $N = 0.3\,IJ = 300\,000$ pixels. Although p and c "fall out" in the comparison, each FCM construct adds, subtracts, multiplies, divides, and exponentiates. Thus, we can expect CPU time on X_s to be significantly less than CPU time on X. Furthermore, even though the *asymptotic* complexities are equal for finite data, the larger F, the larger we expect (t_X/t_{X_s}) to become.

For example, if you need 500 iterations to terminate both schemes, eFFCM saves 500 (700 000) $= 3.5 * 10^8$ constructs, each involving (roughly) cp operations. For moderate problems then, $cp = 5(10) = 50$; so eFFCM saves something like $3.5 * 50 * 10^8 = \mathbf{O}(10^{10})$ operations. Even today, with 1 BIP machines, this represents a savings of several thousand seconds (perhaps 30 min per image). Moreover, even if we start with identical initializations on X and X_s, we expect LFCM to take significantly fewer iterations than it would take on X. Therefore, extensible algorithms should (and do) save time, but do they also produce reasonable approximations to the undiluted outputs obtained by running A^* on X?

We compare the performance of eFFCM in four different ways. $\hat{U}_e$ in (13) will never equal U^*, where $(U^*, \mathbf{V}^*)$ is an optimal pair for J_μ on X. One way to compare $\hat{U}_e$ to U^* is to harden the columns of both matrices with $\mathbf{h}$, resulting in the sets $\{\mathbf{h}(\hat{\mathbf{u}}_{e,i})\}$ and $\{\mathbf{h}(\mathbf{u}_i^*)\}$, $i = 1, 2, \ldots, IJ$. Then, we can compare these two sets of labels, and count the number of mismatches. Let

$$E_{acc}\left(\hat{U}_e, U^*\right) = \frac{1}{2}\sum_{i=1}^{IJ} \|\mathbf{h}(\hat{\mathbf{u}}_{e,i}) - \mathbf{h}(\mathbf{u}_i^*)\|_2^2. \tag{15}$$

E_{acc} in (15) is the number of times $\hat{U}_e$ and U^* *disagree*. Before using (15), the rows of U^* may have to be reorganized (equivalently, relabeling of the centroids in $\mathbf{V}^*$) so that comparison is made between corresponding clusters. While U^* (or $\{\mathbf{h}(\mathbf{u}^*)\}$) may itself be unsatisfactory [e.g., $\{\mathbf{h}(\mathbf{u}^*)\}$ might be a segmentation of F that is deemed unacceptable], (15) is a valid measure of the error incurred by using $\mathbf{A}$ to produce $\hat{U}_e$, and taking $\hat{U}_e \approx U^*$. E_{acc} is a valid measure of error when one looks at the clustering results; i.e., the partitioning of X.

Second, we could also compute $E = \|\hat{\mathbf{V}} - \mathbf{V}^*\|$. Here, we calculate *both* $\hat{\mathbf{V}}$ and $\mathbf{V}^*$, so E could be directly computed. This error is not to be confused with the same expression which appears in [12], where $\hat{\mathbf{V}}$ is computed, but $\mathbf{V}^*$ is the "true but unknown" set of prototypes that crisp c-means *would* produce on an infinite number of samples. E is bounded above in [12] by an asymptotic estimate, and it is used to control the number of samples that are actually processed. However, since U is uniquely determined by the set of centroids $\mathbf{V}$, E_{acc} is directly related to E, and there is no need to calculate both in our applications.

The second index of comparison is t_{acc}, which compares how *time efficient* the extensible algorithm is. Third, we also compare FCM objective function values computed directly running LFCM on X with the objective function values computed on X using the terminal $(\hat{U}_e, \hat{\mathbf{V}})$ obtained by eFFCM. Finally, we also render visual judgments about the quality of $\hat{U}_e$ as an approximation to U^*.

Before we turn to the numerical experiments, we point out that it is even possible to get qualitatively *better* results with

f_1	f_2	f_3
f_4	f_5	f_6
f_7	f_8	f_9

Fig. 2. Gray values over a 3 × 3 window.

$\hat{U}_c$ than with U^*. How? Most iterative parameter estimation algorithms optimize an objective function, and the algorithm can get stuck at a local extremum. FCM on the entire set X will almost surely have a more complex search space than J_μ does on the smaller sample X_s. Therefore, the chance of landing at a "better" minimum is higher with X_s (although this is not guaranteed). Moreover, with a small X_s, more runs of the parameter estimation algorithm with different initial conditions can be made leading to parameters that yield the best value of the objective function (for FCM, it would be the best value of J_μ). Therefore, "high" approximation error in terms of (15) does not necessarily mean that we have poor results, because E_{acc} is a measure of relative agreement between $\hat{U}_c$ and U^*. Next, we describe some numerical experiments that illustrate the ideas of sampling and extensibility.

B. Numerical Experiments

Some of our experiments use two derived features: window *average* and window *busyness* [21]. Consider a 3 × 3 window centered at (i, j) with gray levels as indicated in Fig. 2.

- The average gray level $\overline{f}$ over the window centered at the (i, j)th position of the image is

$$\overline{f} = \tfrac{1}{9}\left(\sum_{i=1}^{9} f_i\right). \tag{16}$$

- The busyness B over the window in Fig. 2 is

$$\begin{aligned} B = \tfrac{1}{12}(&|f_1 - f_2| + |f_2 - f_3| + |f_4 - f_5| + |f_5 - f_6| \\ &+ |f_7 - f_8| + |f_8 - f_9| + |f_1 - f_4| + |f_4 - f_7| \\ &+ |f_2 - f_5| + |f_5 - f_8| + |f_3 - f_6| + |f_6 - f_9|). \end{aligned} \tag{17}$$

We used eFFCM to segment two 4-band satellite images: SI-1 and SI-2 from IRS-1A [39]. Figs. 3(a) and 4(a) show the input images SI-1 and SI-2, respectively. For visual clarity Figs. 3(a) and 4(a) present enhanced (histogram equalized) versions of the input images, but for computations we used the original images. Each image has the complexity triple (256, 256, 256)—relatively small images in the year 2001. Our computational exercise consists of two cases.

Case 1: Sample selection is done using gray levels of band-4 (0.77–0.86 μm) from SI-1; and clustering is performed using the extracted features $(\overline{f}, B)$ in (16) and (17). That is, $X \subset \mathcal{R}^2$; $\mathbf{x}_k = (\overline{f}_k, B_k)$, $k = (i, j)$.

Case 2: The sample is selected using the same band-4 gray levels from SI-2; and raw band-3 (0.62–0.68 μm) and band-4 gray level pixel intensity 2-vectors are used for clustering.

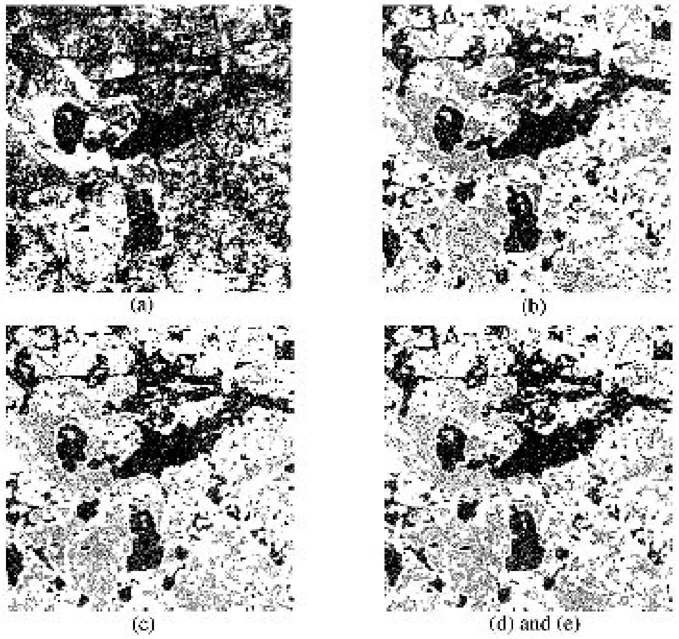

Fig. 3. (a) Input satellite image SI-1. (b) Segmentation of (a) produced by eFFCM when centroids are generated by 9% of the data and only the divergence test is satisfied. (c) Segmentation of (a) produced by eFFCM when centroids are generated by 29% of the data and both the divergence and χ^2 tests are satisfied. (d) and (e) Segmentation of (a) produced by LFCM and mrFCM (which are the same).

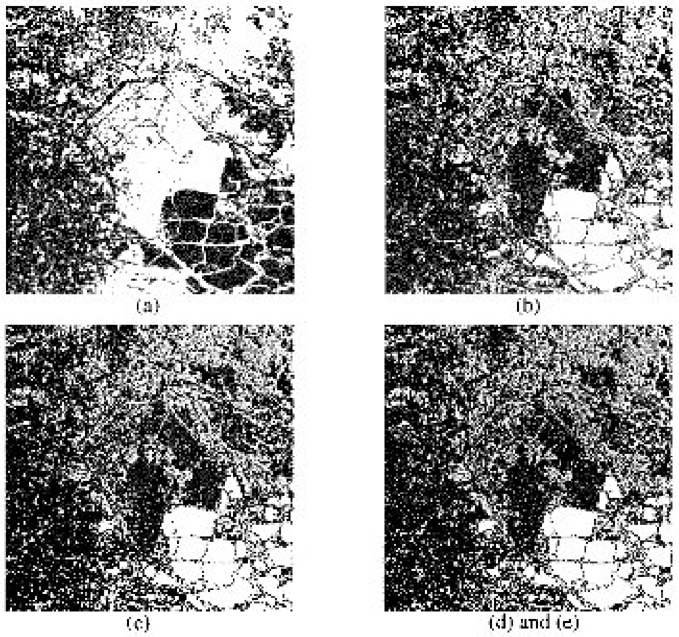

Fig. 4. (a) Input satellite image SI. (b) Segmentation of (a) produced by eFFCM when centroids are generated by 18% of the data and only the divergence test is satisfied. (c) Segmentation of (a) produced by eFFCM when centroids are generated by 21% of the data and both the divergence and χ^2 tests are satisfied. (d) and (e) Segmentation of (a) produced by LFCM and mrFCM (which are the same).

To benchmark the performance of eFFCM, we ran LFCM and mrFCM on X with the same initial cluster prototypes as used for eFFCM. The computational protocols used were $\alpha = 0.95$, $l = 1\%$ initial sample size, $\Delta l = 1\%$ incremental sample size,

TABLE I
COMPARISON OF eFFCM AND mrFCM FOR CASE 1

	eFFCM										mrFCM	
Set No.	% of X used	$J_2(\hat{U}_c, \hat{\mathbf{V}}_{X_s})$	Chi-Square			Divergence			(%) E_{acc}	t_{acc}	(%) E_{acc}	t_{acc}
			Comp.	Table	df (r-1)	Comp.	Table	df (r-1)				
0.a	1%	400230.0	49.05*	36.41	24	57.23	67.50	50	11.51	17.22	0.05	1.05
0.b	2%	393830.7	41.67	46.18	32	52.63	72.14	54	7.82	14.58		
1.a	30%	390542.0	48.36*	47.46	65	52.43	53.47	72	1.20	2.93	0.00	1.42
1.b	33%	390520.1	48.21	48.32	66	50.56	53.47	72	1.00	2.66		
2.a	19%	390517.4	49.15*	44.90	62	51.32	51.74	70	0.79	4.93	0.00	1.20
2.b	38%	390487.7	49.91	50.03	68	49.40	53.47	72	0.36	2.75		
3.a	19%	390637.0	46.33*	44.90	62	49.64	53.47	72	1.50	3.85	0.00	1.38
3.b	23%	390539.4	41.69	44.61	64	45.95	53.47	72	0.40	3.44		
4.a	17%	390560.1	45.26*	44.90	62	48.62	51.74	70	1.31	5.05	0.00	1.46
4.b	33%	390527.6	46.59	48.32	66	45.27	53.47	72	0.68	2.62		
5.a	28%	390524.5	48.81*	47.46	65	50.23	54.33	73	0.80	3.19	0.027	1.19
5.b	35%	390487.2	41.67	50.03	68	42.83	54.33	73	0.16	2.68		
6.a	9%	390706.1	45.71*	41.50	58	48.24	52.60	71	0.83	7.32	0.00	1.35
6.b	29%	390595.4	47.13	47.46	65	49.27	52.60	71	1.08	3.19		
7.a	29%	390502.6	50.76*	47.46	65	51.35	54.33	73	0.43	2.26	0.00	0.93
7.b	35%	390507.5	46.95	50.03	68	47.80	54.33	73	0.59	1.75		
8.a	5%	390606.0	21.57	25.98	43	34.20	47.46	65	1.85	8.63	0.00	1.20
9.a	30%	390482.8	44.92	47.46	65	52.10	53.47	72	0.28	3.86	0.005	1.91
10.a	31%	390602.0	44.30	47.46	65	50.57	53.47	72	0.88	3.11	0.00	2.08

* The values which are significant at 95% (i.e., the values not meeting the acceptance criterion), except for set 0 where it is 5%.

$c = 4$ clusters, $M = I_{p\times p}$ (identity matrix), and $\mu = 2$. eFFCM and LFCM are terminated using the condition

$$\sqrt{\frac{1}{p\times c}\,\|\mathbf{V}^{(t)} - \mathbf{V}^{(t-1)}\|_I^2} < \epsilon = 0.0001$$

where $\mathbf{V}^{(t)}$ is the set of cluster centers generated at iteration t. For mrFCM, we used the following parameters: $\sigma = 7$ stages, $\Delta = 4.5\%$ increment in sample size in every stage, $\epsilon_0 = 0.1$, and $\epsilon = 0.0001$. Cheng *et al.* [34] terminated mrFCM on $\|U^t - U^{t-1}\|_I$. For the sake of a fair comparison, we terminated mrFCM here with the same conditions as LFCM and eFFCM. In other words, we terminated mrFCM with

$$\sqrt{\frac{1}{p\times c}\,\|\mathbf{V}^{(t)} - \mathbf{V}^{(t-1)}\|_I^2} < \epsilon$$

where ϵ is varied as shown in Step 3.3, phase I, mrFCM.

Note that r, the number of categories used to compute the frequency distribution over F_s, is different from c, the number of meaningful segments (clusters) assumed to be in the data. r can be equal to the number of distinct gray values present in the image, or the entire gray scale can be split into r (r less than the number of distinct gray values) nonoverlapping cells. Moreover, r can vary from sample to sample.

In Table I, instance (a) corresponds to the smallest sample for which *either* the χ^2 or divergence passes F_s. If in instance (a) *both* the χ^2 and divergence are insignificant, we do not generate instance (b). However, if F_s in (a) is passed by *only one* of χ^2 or $J(p, q)$, then we augment F_s in (a) by additional samples until both χ^2 and $J(p, q)$ become insignificant. Instance (b) corresponds to this case. For each case, we have tabulated results for several sets of runs with one or two instances for each set. In each table, column 3 is the value of the objective function $J_2(\hat{U}_c, \hat{\mathbf{V}}_{X_s})$ computed on the entire data set with the terminal cluster prototypes generated by eFFCM on the indicated sample (size given in column 2).

We ran LFCM to termination on both X_s and X using the same initial centroids and used U^* and $\hat{U}_c$ with (15) to assess the accuracy of eFFCM relative to LFCM on X. Values of E_{acc} for eFFCM (given as percentages) are reported in column 10 of Table I.

Table I depicts the results obtained for eleven sets of runs for case 1. A typical value of J_2 at termination of LFCM on X is $J_2(U, \mathbf{V}) = 390\,469.8$, which is quite close to the values reported in column 3 of Table I. Set 0 reports the results when X_s is selected at the 5% level of significance. For all other sets, α was 0.95 (i.e., 95% level of significance). The values in column 10 of Table I show that, except for set 0, there is better than 98% agreement between the crisp partitions obtained by hardening the eFFCM and LFCM partitions by maximum memberships, even when only 5% of the data are used (set 8 in Table I). For set 0 (Table I), although we achieve a high acceleration factor, the difference between labels assigned by FFCM and LFCM after hardening the partitions goes up to 11.51%.

We also achieved, without noticeable loss of performance, a time saving of approximately 70% and a space saving of nearly 60%. Column 11 shows values of the acceleration factor t_{acc}. Comparing these to (6e), we see that eFFCM is (apparently) an efficient extension of LFCM. In instance 1.a, for example, eFFCM

TABLE II
COMPARISON OF eFFCM AND mrFCM FOR CASE 2

		eFFCM									mrFCM	
Set No.	% of X used	$J_2(U, V_{X_s})$	Chi-Square			Divergence			(%) E_{acc}	t_{acc}	(%) E_{acc}	t_{acc}
			Comp.	Table	df (r-1)	Comp.	Table	df (r-1)				
1.a	33%	710707.7	33.31	37.29	53	38.58	44.04	61	1.92	2.33	0.10	1.69
2.a	9%	710963.1	30.70*	29.79	44	34.10	37.29	53	1.20	8.90	0.00	3.16
2.b	10%	710850.0	28.34	30.61	45	34.86	37.29	53	1.12	8.35		
3.a	29%	710557.1	43.03*	35.61	51	44.82	44.90	62	0.85	2.70	0.00	1.57
3.b	31%	710521.5	33.54	36.45	52	33.39	44.90	62	1.15	2.56		
4.a	13%	710786.4	35.43*	31.44	46	38.42	38.98	55	0.77	5.65	0.00	1.88
4.b	14%	710703.2	33.41	31.44	46	32.95	38.98	55	0.19	5.67		
5.a	18%	710983.9	38.68*	32.27	47	39.59	40.66	57	0.98	4.56	0.00	1.99
5.b	21%	710968.6	33.931	33.934	49	35.37	41.50	58	0.37	4.37		

* The values which are significant at 95% (i.e. the values not meeting the acceptance criterion).

terminates 2.93 times faster than LFCM, both beginning with the same initialization. Thus, eFFCM completes the entire image in about $(100/2.93) \approx 34\%$ of the time that LFCM requires.

Column 4 is the computed value of χ^2 for the given set. The tabulated χ^2 value is shown in column 5 at the DOF given in column 6. Similarly, computed and tabulated values of divergence, and DOF are reported in columns 7, 8, and 9, respectively.

Columns 12 and 13 present the results obtained with mrFCM. We used the same initial centroids and terminating condition for mrFCM, eFFCM, and LFCM. Since the final phase of mrFCM runs LFCM on the entire data set, unless a very bad initialization ($\mathbf{V}_{X_1}$), is used in Step 2 of phase I of mrFCM, the partitions generated by LFCM and mrFCM with the same initial centroid $\mathbf{V}_{X_1}$ should not be much different. Consequently, the crisp partitions obtained by hardening the LFCM and mrFCM partitions using maximum memberships will be almost the same. This is indeed reflected by the low E_{acc} values reported in column 12 of Table I. The acceleration factor for mrFCM varies from 0.93 to 2.08. Therefore, mrFCM fails to extend LFCM efficiently in the sense of (6e) only once—set no. 7 of case 1. In all other trials, mrFCM is also an efficient extension of LFCM.

Comparing the acceleration factors of eFFCM and mrFCM in Table I, we see that even the lowest acceleration factor for eFFCM (1.75) is nearly equal to the highest acceleration factor (2.08) of mrFCM. For eFFCM, the highest acceleration factor is 8.63 (for 0.a, it is 17.22 where X_s is selected at $\alpha = 0.05$), and out of the 19 instances reported in Table I, the acceleration factor is more than 2.08 for all instances except 7.b. The acceleration factors achieved with mrFCM for case 1 (see Table I) are lower than that reported in [34]. This may be because of the difference in the data sets used or in the termination conditions.

For visual assessment of the performance of eFFCM, we display in Fig. 3 a typical segmented image produced by eFFCM corresponding to set 6 of case 1. Fig. 3(b) is the segmented image produced by eFFCM when the centroids $\hat{\mathbf{V}}$ are produced with 9% of the data points and only the divergence test is satisfied, set 6.a. Fig. 3(c) is the result obtained when both the χ^2 and divergence tests are satisfied, set 6.b. Fig. 3(d) is the result produced by LFCM on X; and Fig. 3(e) [which coincides with Fig. 3(d)] is the segmentation produced by mrFCM. For consistency, we used the same initial prototypes for all four runs. Comparison of Fig. 3(b)–(e) shows that the segmented images are practically identical (visually).

Table II depicts the results for case 2, i.e., when SI-2 is used as the input [see Fig. 4(a)] and gray levels of band-4 are used for sample selection. For case 2, bands (3, 4) intensity values are used as the features. In this case, t_{acc} varies from 2.33 to 8.90 for eFFCM, while acceleration for mrFCM varies between 1.57 and 3.56. Inspection of column 10 in Table II reveals that eFFCM can achieve an acceleration factor of 8.90 with less than 2% degradation in performance as measured by E_{acc}. For this data set, a typical value of J_μ, when LFCM is run on X, is 710 377.1, which is practically the same as those reported in column 3, Table II. As an illustration of segmentations produced for case 2, we show the images for set 5, Table II. Fig. 4(b) and (c) displays results of eFFCM for instances 5.a and 5.b; while Fig. 4(d) and (e) represents the segmented images for LFCM and mrFCM. Again, there is "good" visual agreement between the literal and approximate segmentations.

Discounting the case 0 trials in Table I (the only trials that used $\alpha = 0.05$ as the statistical significance threshold), Tables I and II have 26 eFFCM (and 16 mrFCM) tests. The average size of X_s over the 26 trials was $N = 0.2388IJ$—that is, on average, we need about 24% of the image in order for X_s to be acceptable at $\alpha = 0.95$. The average acceleration of eFFCM was 4.20, compared to an average of 1.63 for mrFCM. Therefore, for an image that takes, for example, 30 min to segment with LFCM, we can expect completion with mrFCM in 18.4 min, while eFFCM cuts the run-time to 7.14 min. On the other hand, the average accuracy of eFFCM at approximating LFCM in 26 trials is $\overline{E}_{acc} = 0.87\%$, whereas mrFCM (almost) always produces complete agreement between pixel labels reproduced by itself and LFCM. Therefore, in a 2000×5000 image, we expect about 87 000 pixel labels (in 10 million) produced by eFFCM and LFCM to disagree. Combining these facts, it seems safe to assert that eFFCM will probably be several times faster than mrFCM, but at a cost in accuracy of perhaps 1% of the LFCM labels found by mrFCM. From this it is clear that eFFCM and mrFCM can be combined to effect a further tradeoff between saved time and accuracy lost when LFCM is a desirable segmentation method but is too costly to run.

VII. Conclusions and Issues for Further Research

A. General Conclusions

There are two main contributions of this paper: 1) the use of simple hypothesis tests, such as χ^2 or divergence, to select subsets of pixels whose intensities are representative of image regions, and whose feature vectors comprise training data for computationally intensive learning models used in image processing; and 2) the introduction of an (empirical) notion of efficient extensibility of imaging operators from the training pixels to the rest of the image. These two ideas are applicable to "large image" processing, and are designed to save time and space by running the "literal" learning algorithms on the training data, and then *approximating* the results which a literal algorithm might obtain on the remaining pixels in the image. This is done, of course, at a sacrifice in accuracy (where "accuracy" means closeness to the output of the *literal version*).

In our proposed scheme, suppose we accept a sample obtained after k-steps enhancement. Therefore, the hypothesis has been tested $k+1$ times before the sample is accepted. Of these $k+1$ tests, in the first k cases, the hypothesis was rejected at the α level of significance. A natural question is: *Is the probability that one of these rejections is by just chance (i.e., it was wrongly rejected when it was actually true or acceptable)?* If we assume the probability of committing a Type-I error as a Bernoulli process, then the probability of committing at least one Type-I error in k trials is $\gamma = (1-(1-\alpha)^k)$. For example, with $\alpha = 0.05$ and $k = 2$, $\gamma = 0.0975$. Notice that γ is greater than $\alpha = 0.05$. If the number of tests to be performed *is known beforehand* then the Bonferroni correction can be used to adjust α downwards so that the overall chance of Type-I error remains α. However, in the present context, this correction cannot be done (because the number of tests to be performed is not known *a-priori*), nor it is necessary. It is not necessary because if the hypothesis is rejected by chance, we will enhance the sample, and enlarging the sample will in turn make a better representation of the image.

Another point worth investigating is the power of the tests used here. Let β be the probability of wrongly accepting a hypothesis when it is false. Thus, β is a function of the alternative hypothesis H_1. The complementary probability $1-\beta$ is the *power* of the test of hypothesis H_0 against the alternative hypothesis H_1 [16]. To get an idea about the power of the tests, a lot of simulations must be done taking different distributions as alternative hypotheses. We leave this for a future work.

Since our problem is to select a sample irrespective of the learning task, estimation of an asymptotic error rate bound as done in [12] is quite difficult, because the definition of error will depend on the learning task. Moreover, Domingos and Hulten [12] computed the loss with respect to centroids obtained using finite samples and infinite samples (10 million data points). In our case, the population is always finite, so we can at best compare the centroids produced by eFFCM on X_s and FCM on the entire X. Nonetheless, securing an asymptotic bound on the error rate for FCM following the method in [12] for crisp c-means is an interesting and useful idea for future research.

B. Conclusions for Acceleration of FCM

We exemplified our sampling and extension methods by applying them to a typical image processing problem—segmentation with the (literal) FCM clustering algorithm, resulting in the new approximation technique eFFCM. Based on our limited experiments, we find that the new method is about 2.5 times faster than mrFCM, and 4.2 times faster than LFCM, at an average cost in changed pixel labels of less than 1%. Our method uses, on average, about one-fourth of the image data during training, and extended segmentations are (visually) indistinguishable from their literal relatives.

To conclude, we make a few more remarks about eFFCM.

1) Unlike other multistage schemes (e.g., mrFCM), eFFCM runs iterative LFCM only once, and on a relatively small subset of X.
2) Unlike AFCM, eFFCM does exact optimization of J_μ for a small subset of X.
3) The χ^2 and divergence tests do not necessarily agree. If either statistic satisfies the hypothesis test, it can be assumed that the sample tested is a good representative of X, so either test can be used.
4) The size of X_s cannot be fixed prior to run-time. The actual time and space reductions achieved by eFFCM depend on the distribution of gray values in F and the particular sample of F chosen. Experimentally, the size of the selected sample is almost always less than one-third of the size of the image, and on average reduces computation time by about 76%.
5) When X is too large to load in host memory, AFCM and mrFCM cannot be used; but eFFCM will provide approximate FCM clustering in X as long as X_s can be mounted in the host.

We feel that these are significant improvements to the utility of FCM for *LARGE* images. As (I, J, L) *decreases*, it becomes more and more attractive to simply run LFCM on the whole image. mrFCM can be usefully modified with our idea so that LFCM is used only twice: once on X_s and then on X, with the terminal cluster prototypes generated by LFCM on X_s as input to Step 5 of mrFCM. To build confidence in our method, more simulations need to be done with images of different sizes and complexity.

Acknowledgment

The authors gratefully acknowledge L. O. Hall for providing the code for mrFCM, which they tailored to suit their requirements. They also thank B. Uma Shankar for helping them with the numerical computation, and the referees for their constructive suggestions.

References

[1] K. Bowyer, D. Kopans, W. P. Kegelmeyer, R. Moore, M. Sallam, K. Chang, and K. Woods, "The digital database for screening mammography," in *Proc. 3rd Int. Workshop Digital Mammography, 58*, 1996. Electronic contact information includes email: ddsm@bigpine.csee.usf.edu and a URL address: http://marathon.csee.usf.edu/Mammography/Database.html.

[2] S. B. Chen, Y. J. Lou, L. Wu, and D. B. Zhao, "Intelligent methodology for sensing, modeling, and control of pulsed GTAW: Part I—Bead-on-plate welding," *Weld. Res. Supplement*, pp. 151–163, June 2000.

[3] ——, "Intelligent methodology for sensing, modeling and control of pulsed GTAW: Part II—Butt joint welding," *Weld. Res. Supplement*, pp. 164–174, June 2000.

[4] J. C. Bezdek, *Pattern Recognition With Fuzzy Objective Function Algorithms*. New York: Plenum, 1981.

[5] U. Fayyad and P. Smyth, "From massive data sets to science catalogs: Applications and challenges," in *Proc. Workshop Massive Data Sets*, J. Kettenring and D. Pregibon, Eds. Washington, DC: National Research Council, July 7–8, 1995.

[6] T. Zhang, R. Ramakrishnan, and M. Livny, "BIRCH: An efficient data clustering method for very large databases," in *Proc. ACM SIGMOD*, 1995, pp. 103–114.

[7] P. Bradley, U. Fayyad, and C. Reina, "Scaling clustering algorithms to large databases," in *Proc. 4th Int. Conf. Knowledge Discovery and Data Mining*. Menlo Park, CA: AAAI Press, 1998, pp. 9–15.

[8] V. Ganti, J. Gehrke, and R. Ramakrishnan, "Mining very large databases," *Computer*, pp. 38–45, Aug. 1999.

[9] V. Ganti, R. Ramakrishnan, J. Gehrke, A. L. Powell, and J. C. French, "Clustering large datasets in arbitrary metric spaces," in *Proc. 15th Int. Conf. Data Engineering*, Sydney, Australia, March 23–26, 1999, pp. 502–511.

[10] C. Faloutsos and K. Lin, "FastMap: A fast algorithm for indexing, data-mining and visualization of traditional and multimedia datasets," in *Proc. ACM SIGMOD Int. Conf. Management of Data*, San Jose, CA, May 22–25, 1995, pp. 163–174.

[11] S. Guha, R. Rastogi, and K. Shim, "CURE: An efficient clustering algorithm for large databases," in *Proc. ACM SIGMOD Int. Conf. Management of Data*, Seattle, WA, June 2–4, 1998, pp. 73–84.

[12] P. Domingos and G. Hulten, "A general method for scaling up machine learning algorithms and its application to clustering," in *Proc. 18th Int. Conf. Machine Learning*, 2001, pp. 106–113.

[13] W. Hoeffding, "Probability inequalities for sums of bounded random variables," *J. Amer. Stat. Assoc.*, vol. 58, pp. 13–30, 1963.

[14] R. Haralick and L. Shapiro, *Computer and Robot Vision*. Reading, MA: Addison-Wesley, 1992, vol. 1.

[15] R. F. Gonzalez and R. E. Woods, *Digital Image Processing*. Reading, MA: Addison-Wesley, 1992.

[16] M. Kendall and A. Stuart, *The Advanced Theory of Statistics, Vol 2: Inference and Relationship*. New York: Macmillan, 1979.

[17] J. D. Gibbons and S. Chakraborti, *Nonparametric Statistical Inference*. New York: Marcel Dekker, 1992.

[18] W. G. Cochran, *Sampling Techniques*, 3rd ed. New York: Wiley, 1977.

[19] S. Kullback, *Information Theory and Statistics*, 3rd ed. New York: Wiley, 1959.

[20] R. A. Fisher, *Statistical Methods for Research Workers*, 12th ed. Edinburgh, U.K.: Oliver & Boyd, 1954.

[21] A. Rosenfeld and A. C. Kak, *Digital Picture Processing*. New York: Academic, 1982.

[22] J. C. Bezdek, R. Chandrasekhar, and Y. Attikiouzel, "A geometric approach to edge detection," *IEEE Trans. Fuzzy Syst.*, vol. 6, pp. 52–75, Jan. 1998.

[23] J. C. Bezdek and D. Kerr, "Training edge detecting neural networks with model-based examples," in *Proc. IEEE 3rd Int. Conf. Fuzzy Syst.*, Piscataway, NJ, 1994, pp. 894–901.

[24] J. C. Bezdek, J. M. Keller, R. Krishnapuram, and N. R. Pal, *Fuzzy Models and Algorithms for Pattern recognition and Image Processing*. Norwell, MA: Kluwer, 1999.

[25] N. R. Pal, T. Cahoon, J. C. Bezdek, and N. R. Pal, "A new approach to target detection for LADAR data," *IEEE Trans. Fuzzy Syst.*, vol. 9, pp. 44–52, Jan. 2001.

[26] A. K. Jain and P. J. Flynn, "Image segmentation by clustering," in *Advances in Image Understanding*, K. Bowyer and N. Ahuja, Eds. Los Alamitos, CA: IEEE Comput. Soc. Press, 1996, pp. 65–83.

[27] L. O. Hall, A. M. Bensaid, L. P. Clarke, R. P. Velthuizen, M. S. Silbiger, and J. C. Bezdek, "A comparison of neural network and fuzzy clustering techniques in segmenting magnetic resonance images of the brain," *IEEE Trans. Neural Networks*, vol. 3, pp. 672–682, Sept. 1992.

[28] J. M. Carazo, F. F. Rivera, E. L. Zapata, M. Radermacher, and J. Frank, "Fuzzy sets-based classification of electron microscopy images of biological macromolecules with an application to ribsomeal particles," *J. Microsc.*, vol. 157, no. 2, pp. 187–203, 1990.

[29] R. P. Velthuizen, L. P. Clarke, S. Phuphanich, L. O. Hall, A. M. Bensaid, J. A. Arrington, H. M. Greenberg, and M. L. Sibiger, "Unsupervised measurement of brain tumor volume on MR images," *J. Magn. Reson. Imag.*, vol. 5, no. 5, pp. 594–605, 1995.

[30] R. L. Cannon, J. V. Dave, J. C. Bezdek, and M. M. Trivedi, "Segmentation of a thematic mapper image using the fuzzy c-means clustering algorithm," *IEEE Trans. Geosci. Remote Sensing*, vol. GE-24, pp. 400–408, Mar. 1986.

[31] R. L. Cannon, J. V. Dave, and J. C. Bezdek, "Efficient implementation of the fuzzy c-means clustering algorithms," *IEEE Trans. Pattern Anal. Machine Intell.*, vol. PAMI-8, no. 2, pp. 248–255, 1986.

[32] M. Morrison and Y. Attikouzel, "An introduction to segmentation of magnetic resonance images," *Aust. Comput. J.*, vol. 26, no. 3, pp. 90–98, 1994.

[33] R. De La Paz, R. Berstein, W. Hanson, and M. Walker, "Approximate fuzzy c-means (AFCM) cluster analysis of medical magnetic resonance image (MRI) data—A system for medical research and education," *IEEE Trans. Geosci. Remote Sensing*, vol. GE-25, pp. 815–824, 1986.

[34] T. W. Cheng, D. B. Goldgof, and L. O. Hall, "Fast clustering with application to fuzzy rule generation," in *Proc. IEEE Int. Conf. Fuzzy Systems*, Tokyo, Japan, 1995, pp. 2289–2295.

[35] B. Uma Shankar and N. R. Pal, "FFCM: An effective approach for large data sets," in *Proc. 3rd Int. Conf. Fuzzy Logic, Neural Nets, and Soft Computing, IIZUKA*, Fukuoka, Japan, 1994, pp. 331–332.

[36] S. Eschrich, J. Ke, L. O. Hall, and D. B. Goldgof, "Fast fuzzy clustering of infrared images," in *Proc. 20th NAFIPS Conf.*, 2001, pp. 1145–1150.

[37] M. S. Kamel and S. Z. Selim, "New algorithm for solving the fuzzy clustering problem," *Pattern Recognit.*, vol. 27, no. 3, pp. 421–428, 1994.

[38] *Interactive Data Language (IDL), Ver. 3.1 Users Guide*. Boulder, CO: Research Systems, Inc., June 1993.

[39] J. A. Richards, *Remote Sensing Digital Image Analysis: An Introduction*, 2nd ed. Berlin, Germany: Springer-Verlag, 1993.

[40] J. C. Bezdek and R. Hathaway, *Some Notes on Alternating Optimization, in Advances in Soft Computing, AFSS*, N. R. Pal and M. Sugeno, Eds. Berlin, Germany: Springer-Verlag, 2002, pp. 289–304. LNAI 2275.

[41] R. J. Hathaway and J. C. Bezdek, "Convergence of alternating optimization," *Comput. Optim. Appl.* in review.

Nikhil R. Pal (M'91–SM'00) received the B.Sc. degree with honors in physics and the M.B.M degree from the University of Calcutta, Calcutta, India, in 1979 and 1982, respectively, and the M.Tech. and Ph.D. degrees in computer science from the Indian Statistical Institute, Calcutta, in 1984 and 1991, respectively.

Currently, he is a Professor in the Electronics and Communication Sciences Unit of the Indian Statistical Institute. From September 1991 to February 1993, July 1994 to December 1994, October 1996 to December 1996, and January 2000 to July 2000, he was with the Computer Science Department of the University of West Florida, Pensacola. He was also a Guest Faculty of the University of Calcutta. His research interests include image processing, pattern recognition, fuzzy sets theory, measures of uncertainty, neural networks, genetic algorithms, and fuzzy logic controllers. He has coauthored a book entitled *Fuzzy Models and Algorithms for Pattern Recognition and Image Processing*, (Norwell, MA: Kluwer, 1999), co-edited a volume entitled *Advances in Pattern Recognition and Digital Techniques, ICAPRDT'99*, (New Delhi: Narosa, 1999), and edited a volume entitled *Pattern Recognition in Soft Computing Paradigm*, (Singapore: World Scientific, FLSI Vol. 2, 2001). He is an Associate Editor of the *International Journal of Fuzzy Systems, International Journal of Approximate Reasoning*, IEEE TRANSACTIONS ON FUZZY SYSTEMS, and IEEE TRANSACTIONS ON SYSTEMS, MAN, AND CYBERNETICS—PART B: CYBERNETICS, and an Area Editor of *Fuzzy Sets and Systems*.

James C. Bezdek (M'80–SM'90–F'92) received the B.S.C.E. degree from the University of Nevada, Reno, in 1969, and the Ph.D. degree in applied math from Cornell University, Ithaca, NY, in 1973.

He is currently a Professor in the Department of Computer Science, University of West Florida, Pensacola. His interests include woodworking, fine cigars, optimization, motorcycles, pattern recognition, gardening, fishing, image processing, snow skiing, computational neural networks, blues music, and computational medicine. He is the Founding Editor of the *International Journal of Approximate Reasoning* and IEEE TRANSACTIONS ON FUZZY SYSTEMS.

Dr. Bezdek is a Fellow of the IEEE and the IFSA.