Unsupervised Segmentation of Natural Images via Lossy Data Compression

Allen Y. Yang
John Wright
S. Shankar Sastry
Yi Ma

Electrical Engineering and Computer Sciences
University of California at Berkeley

Technical Report No. UCB/EECS-2006-195
http://www.eecs.berkeley.edu/Pubs/TechRpts/2006/EECS-2006-195.html

December 28, 2006

Unsupervised Segmentation of Natural Images via Lossy Data Compression *

Allen Y. Yang[⋆†] John Wright[†‡] Shankar Sastry[⋆] Yi Ma[†‡]

[⋆] Cory Hall, EECS
University of California, Berkeley
Berkeley, CA 94720
Email: {yang, sastry}@eecs.berkeley.edu

[†]Coordinated Science Lab, UIUC
1308 W. Main Steet
Urbana, IL 61801
Email: {jnwright, yima}@uiuc.edu

[‡] Microsoft Research Asia
5/F, Beijing Sigma Center
No.49, Zhichun Road, Hai Dian District
Beijing China 100080

Abstract

In this paper, we cast natural-image segmentation as a problem of clustering texure features as multivariate mixed data. We model the distribution of the texture features using a mixture of Gaussian distributions. However, unlike most existing clustering methods, we allow the mixture components to be degenerate or nearly-degenerate. We contend that this assumption is particularly important for mid-level image segmentation, where degeneracy is typically introduced by using a common feature representation for different textures. We show that such a mixture distribution can be effectively segmented by a simple agglomerative clustering algorithm derived from a lossy data compression approach. Using simple fixed-size Gaussian windows as texture features, the algorithm segments an image by minimizing the overall coding length of all the feature vectors. In terms of a variety of performance indices, our algorithm compares favorably against other well-known image segmentation methods on the Berkeley image database.

1 Introduction

Natural-image segmentation is one of the classical problems in computer vision. It is widely accepted that a good segmentation should group image pixels into regions whose statistical characteristics (of the color or texture) are homogeneous or stationary, and whose boundaries are simple and spatially accurate [10]. Nevertheless, from a statistical viewpoint, natural-image segmentation is an *inherently ambiguous* problem for at least the following two technical reasons[1]:

1. The statistical characteristics of local features (*e.g.*, color, texture, edge, contour) of natural images usually do not show the same level of homogeneity or saliency at the same spatial or quantization scale. This is not only the case for different natural images, but also often the case for different regions within the same image. Thus, one should not expect the segmentation result to be unique [28], and instead should prefer a hierarchy of segmentations at multiple scales.

2. Even after accounting for variations due to scale, different regions or textures may still have different intrinsic complexities, making it a difficult statistical problem to determine the correct number of segments and their model dimensions. For instance, if we use Gaussian distributions to model the features of different textures, the Gaussian for a simple texture obviously has a higher degree of degeneracy (or a lower dimension) than that for a complex texture.

In the literature, many statistical models and methods have been proposed to address some of these difficulties (see [34] for a review). In this paper, we are interested in *unsupervised* image segmentation. Popular methods in this category include *feature-based* Mean-Shift [1], *graph-based* methods [25, 5], *region-based* split-and-merge techniques [23, 31], and global optimization approaches based on either energy functions [33] or *minimum description length* (MDL) [12]. Recent developments have mainly focused on the problem of how to integrate textural information at different scales. For example, one can use more sophisticated *region-growing* or *split-and-merge* techniques [10, 27, 4, 8] to partition inhomogeneous regions;

*This work is partially supported by NSF CAREER IIS-0347456, ONR YIP N00014-05-1-0633, and ARO MURI W911NF-06-1-0076.

[1]It is arguably true that human perception of an image is itself ambiguous. However, we here are concerned about only the ambiguities in computing image segmentation.

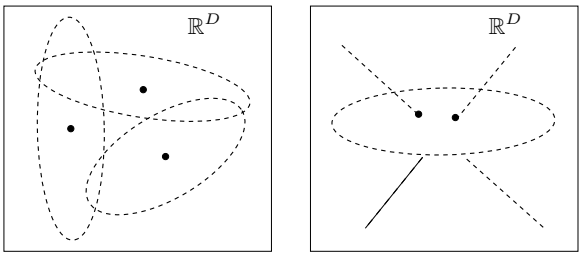

Figure 1: Mixture of regular (left) or degenerate (right) Gaussians.

or one can use *Markov random fields* to model textures or other image cues [16, 23, 28]. For a more detailed survey of these methods, the reader is referred to [13, 7, 24].

Motivations and Contributions: Although the reported performance of image segmentation algorithms has improved significantly over the years, it comes partly at the price of the use of ever more sophisticated feature selection processes, more complex statistical models, and more difficult optimization techniques. In this paper, however, we aim to show that for texture features as simple as fixed-size Gaussian windows (Figure 3), with the choice of a likely more relevant class of statistical models (Figure 1) and its associated agglomerative clustering algorithm (Algorithm 1), one can achieve equally good, if not better, segmentation results as many of the above sophisticated statistical models and optimization methods. Our approach relies on the following two assumptions about natural images:

1. The distribution of texture features in a natural image is (approximately) a mixture of Gaussians that can be *degenerate and of different dimensions* (see Figure 1 right), one for each image segment.

2. At any given quantization scale, the *optimal* segmentation is the one that gives the most compressed representation of the image features, as measured by the number of binary bits needed to encode all the features.

In Section 2, we will show that for features drawn from a mixture of (possibly degenerate) Gaussians, the segmentation that minimizes the coding length is achieved by partitioning the features into their respective Gaussians (degenerate or not). Thus, through compression, the features associated with each segment will be a Gaussian-like cluster.

Be aware that here we are not suggesting compressing the image *per se*. Instead, we compress texture features extracted from the image. Our method bears resemblance to some global optimization approaches, such as using region merging techniques to minimize the MDL cost function [12]. However, the new method significantly differs from the existing maximum-likelihood (ML) or MDL-based image segmentation in two main aspects:

First, we allow the distributions to be *degenerate*, and introduce a new clustering algorithm capable of handling degeneracy. Extant image segmentation methods that segment features based on the cluster centers (*e.g.*, K-Means) or density modes (*e.g.*, Mean-Shift) typically work well at low-level segmentation using low-dimensional color features with blob-like distributions (Figure 1 left) [25]. But at mid-level segmentation using texture features extracted at a larger spatial scale, we normally choose a feature space whose dimension is high enough that the structures of all textures in the image can be *genuinely represented*.[2] Such a representation unavoidably has redundancy for individual textures: The cluster of features associated with one texture typically lies in a low-dimensional submanifold or subspace whose dimension reflects the complexity of the texture (Figure 1 right). Properly harnessed, such low-dimensional structures can be much more informative for distinguishing textures than the cluster means. In this paper, we will see that data compression provides a very effective means of extracting such low-dimensional structures.

Second, we consider *lossy coding* of the image features, up to an allowable distortion. Varying the distortion provides a simple but effective means of considering textural information at different *quantization* scales.[3] Compressing the image features with different distortions, we naturally obtain a hierarchy of segmentations: the smaller the distortion, the more refined the segmentation is (see Figure 7). In a way, the distortion also plays an important role in image segmentation as a measure of the *saliency* of the segments in an image: First, how small the distortion needs to be in order for certain regions to be segmented from the background, and second, how much we can change the distortion without significantly altering the segmentation (see Figure 7 again). Thus, lossy compression offers a convenient framework for diagnosing the statistics of a natural image at different quantization scales for various segmentation purposes.

[2] Here a genuine representation means that we can recover every texture with sufficient accuracy from the representation.

[3] In this paper, we do not consider varying spatial scale as we will always choose a fixed-size window as the feature vector. Nevertheless, as we will demonstrate, good segmentation can still be obtained.

Organization: This paper is organized as follows: Section 2 introduces a new clustering algorithm for minimizing the coding length of data drawn from a mixture of (possibly degenerate) Gaussians. Section 3 discusses how to apply it to image segmentation. Section 4 gives experimental results on the Berkeley segmentation database, and compares to other existing algorithms.

2 Segmentation of Mixtures of Gaussians via Lossy Compression

Once we have assumed that image feature vectors are drawn from a mixture of (possibly degenerate) Gaussians, the problem of image segmentation reduces to that of segmenting such mixed data into multiple Gaussian-like clusters. A popular statistical method for segmenting mixed data is the *expectation-maximization* (EM) algorithm [3, 18], which is essentially a greedy descent algorithm to find the maximum-likelihood (ML) estimate of the mixture of Gaussian distributions [9, 26, 6].

However, notice that here we might be dealing with degenerate Gaussians with unknown dimensions, and furthermore, we do not even know how many of them. Conventional EM-based clustering algorithms do not address these problems, and must be modified to perform well in this domain [6]. In this paper, we introduce a simple clustering method, especially adept at handling unknown number of (possibly degenerate) Gaussians. The new method follows the principle of *lossy minimum description length* (LMDL)[4]:

Principle 1 (Data Segmentation via Lossy Compression) *We define the optimal segmentation to be the one that minimizes the number of bits needed to code the segmented data, subject to a given distortion.*

To apply this principle to our problem, we require an accurate measure of the coding length of data drawn from a mixture of Gaussians. We begin by examining the coding length of data from a single Gaussian. Suppose we are given a random vector $v \in \mathbb{R}^D$ with a multivariate Gaussian distribution $\mathcal{N}(\mu, \Sigma)$, which we wish to encode such that the original vector can be recovered up to a given distortion ε^2, *i.e.*, $\mathbb{E}[\|v - \hat{v}\|^2] \leq \varepsilon^2$. From information theory [2], the average number of bits needed to code v is approximately given by the *rate-distortion function* of the Gaussian:

$$R(\varepsilon) = \frac{1}{2}\log_2 \det(I + \frac{D}{\varepsilon^2}\Sigma), \tag{1}$$

where I is an identity matrix, and Σ is the covariance.[5]

Now consider a set of N i.i.d. samples $V = (v_1, v_2, \ldots, v_N) \in \mathbb{R}^{D\times N}$ drawn from the Gaussian distribution. Let $\mu \doteq \frac{1}{N}\sum_{i=1}^N v_i$, and $\bar{V} \doteq V - \mu \cdot 1_{1\times N}$. As $\hat{\Sigma} = \frac{1}{N}\bar{V}\bar{V}^T$ is an estimate of Σ, an estimate of the rate-distortion function $R(\varepsilon)$ is

$$R(\varepsilon, V) \doteq \frac{1}{2}\log_2 \det\big(I + \frac{D}{\varepsilon^2 N}\bar{V}\bar{V}^T\big). \tag{2}$$

Encoding the N vectors in V therefore requires $N \cdot R(V)$ bits. Since the codebook is adaptive to the data V, we must also represent it with $D \cdot R(V)$ bits, which can be viewed as the cost of coding the D principal axes of the data covariance $\frac{1}{N}\bar{V}\bar{V}^T$. As the data are in general not zero-mean, we need additional $\frac{D}{2}\log_2(1 + \frac{\mu^T\mu}{\varepsilon^2})$ bits to encode the mean vector μ. This leads to the following estimate of the total number of bits needed to encode the data set V:

$$L(V) \doteq \frac{N+D}{2}\log_2 \det\big(I + \frac{D}{\varepsilon^2 N}\bar{V}\bar{V}^T\big) + \frac{D}{2}\log_2(1 + \frac{\mu^T\mu}{\varepsilon^2}). \tag{3}$$

Furthermore, although the above formula is derived for a Gaussian source, the same formula gives an *upper bound* of the coding length for any finite number of samples drawn from a subspace, *i.e.*, a degenerate Gaussian. A detailed proof is provided in the Appendix.

Now let us consider the given data set V as drawn from a mixture of Gaussians. In this case, (3) no longer gives an accurate estimate of the minimum coding length for V. It may be more efficient to code V as the union of multiple disjoint subsets: $V = W_1 \cup W_2 \cup \cdots \cup W_K$. If each subset is sufficiently Gaussian, the total number of bits needed to code V is at most:

$$L^s(W_1, \ldots, W_K) \doteq \sum_{i=1}^K \{L(W_i) + |W_i|\big(-\log_2(|W_i|/N)\big)\}. \tag{4}$$

[4] For a theoretical characterization and comparison of (lossy) ML estimate and (lossy) MDL estimate, one may refer to [14].

[5] Strictly speaking, the rate-distortion function for the Gaussian source $\mathcal{N}(\mu, \Sigma)$ is $R(\varepsilon) = \frac{1}{2}\log_2 \det\big(\frac{D}{\varepsilon^2}\Sigma\big)$ when $\frac{\varepsilon^2}{D}$ is smaller than the smallest eigenvalue of Σ. Thus the equality is good only when the distortion ε is relatively small. However, when $\frac{\varepsilon^2}{D}$ is larger than some eigenvalues of Σ, the rate-distortion function becomes more complicated [2]. Nevertheless, the approximate formula $R(\varepsilon) = \frac{1}{2}\log_2 \det(I + \frac{D}{\varepsilon^2}\Sigma)$ can be viewed as the rate distortion of the "regularized" source that works for all range of ε.

Here the second term counts the number of bits needed to code (losslessly) the membership of the N samples in the K groups, *e.g.*, using the Huffman coding [2]. Notice that the Huffman coding of the membership is optimal only when the membership of the vectors in the K segments is totally random. However, in image segmentation, the membership of pixels is not random – adjacent pixels have higher probability of being in the same segment. In this case, Huffman coding only gives a loose upper bound. Nevertheless, we will demonstrate that minimizing such a function leads to a very simple and effective segmentation algorithm.

To find the optimal segmentation, one essentially needs to compute the coding length for all possible segmentations of the data V, which is a very expensive combinatorial optimization problem. To make the optimization tractable, we propose a *pairwise steepest descent* procedure to minimize the coding length: In the initialization step, each vector v_i is assigned as its own group. At each iteration a pair of groups S_i and S_j is merged such that the decrease in the coding length due to coding S_i and S_j together is maximal. The algorithm terminates when the coding length can no longer be reduced by merging any pair of groups.

Algorithm 1: **(Pairwise Steepest Descent).**

input: the data $V = (v_1, v_2, \ldots, v_N) \in \mathbb{R}^{D\times N}$ and a distortion ε^2.
initialize $\mathcal{S} := \{S_i = \{v_i\} \mid i = 1, \cdots, N\}$.
while $|\mathcal{S}| > 1$ **do**
    choose distinct groups $S_1, S_2 \in \mathcal{S}$ such that
    $L^s(S_1 \cup S_2) - L^s(S_1, S_2)$ is minimal.
    if $L^s(S_1 \cup S_2) - L^s(S_1, S_2) \geq 0$ **then** break;
    else $\mathcal{S} := \big(\mathcal{S} \setminus \{S_1, S_2\}\big) \cup \{S_1 \cup S_2\}$.
end
output: $\mathcal{S}$

Notice that the greedy merging process in Algorithm 1 is similar in concept to classical agglomerative clustering methods, especially Ward's method [30, 11]. However, by using the coding length as a new distance measure between groups, Algorithm 1 significantly improves these classical methods particularly when the distributions are degenerate or the data contain outliers. Nevertheless, as a greedy descent scheme, the algorithm does not guarantee to always find the globally optimal segmentation for any given (V, ε^2).[6] In our experience, the main factor affecting the global convergence of the algorithm appears to be the density of the samples relative to the distortion ε^2.

Extensive simulations have verified that this algorithm is consistently effective in segmenting data that are drawn from a mixture of Gaussian or degenerate subspace distributions. In addition, the algorithm tolerates significant amounts of outliers, and requires no prior knowledge of the number of groups nor their dimensions. Figure 2 shows a few segmentation results of this algorithm on synthesized data sets.

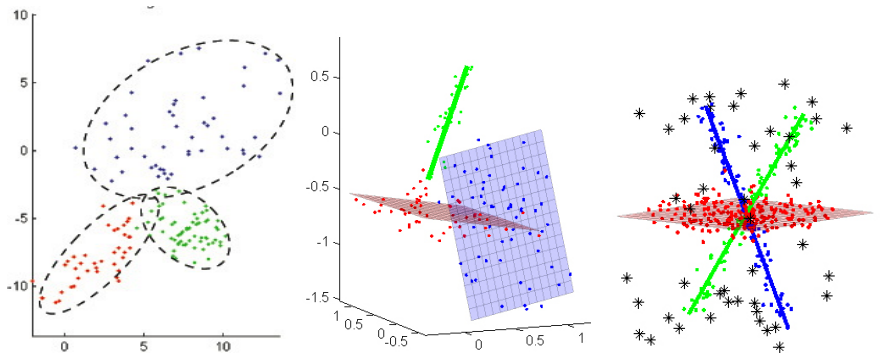

Figure 2: Simulation results (in color) of Algorithm 1 on three different mixture distributions. Left: Three Gaussian distributions in $\mathbb{R}^2$. Middle: Three affine subspaces of dimensions $(2, 2, 1)$ in $\mathbb{R}^3$. Right: Three linear subspaces of dimensions $(2, 1, 1)$ in $\mathbb{R}^3$ with 12% outliers; the algorithm groups all the outliers into one extra Gaussian cluster, in addition to the three subspaces.

In the above experiment, the distortion parameter ε^2 was selected to be close to the true noise variance to achieve best

[6] It may be possible to improve the convergence by using more complicated split-and-merge strategies [29].

results. In practice, there is no universal rule for choosing a good ε^2 for all practical data sets. To apply Algorithm 1 to image segmentation, we need to be able to adaptively choose ε^2 for each image based on its unique texture distributions. We will carefully examine this issue in the next section.

3 Image Segmentation via Lossy Compression

In this section, we describe how the lossy compression-based method in Section 2 is applied to segment natural images. We first discuss what features we use to represent textures and why. We then describe how a *low-level segmentation* is applied to partition an image into many small homogeneous patches, known as *superpixels*. The superpixels are used to initialize the *mid-level texture-based segmentation*, which minimizes the total coding length of all the texture features by repeatedly merging adjacent segments, subject to a distortion ε^2. Finally, we study several simple heuristics for choosing a good ε^2 for each image.

3.1 Constructing Feature Vectors

We choose to represent a 3-channel RGB color image in terms of the $L^*a^*b^*$ color metric, which was specially designed to best approximate perceptually uniform color spaces.[7] While the dependence of the three coordinates on the traditional RGB metric is nonlinear [1], the $L^*a^*b^*$ metric better facilitates representing texture via mixtures of Gaussians. Perceptual uniformity renders the allowable distortion ε^2 meaningful in terms of human perception of color differences, tightening the link between lossy coding and our intuitive notion of image segmentation.

In the literature, there have been two major types of features used to capture local textures. The first considers responses of a 2D-filter bank as texture features [15, 32]. The second applies a $w \times w$ cut-off window around each pixel and stacks the color values inside the window into a vector [22, 21]. In this paper, we choose to use the second method, avoiding the construction of a texture filter bank that arguably relies on the given image set [32]. Each $w \times w$ window is weighted by a 2D Gaussian kernel before stacking to reduce the boundary effect. Figure 3 illustrates this process. In the experiment, we find a 7×7 window provides satisfactory results, although other sizes may also work well.[8] Finally, to reduce the dimensionality, we project the feature vectors into an 8-dimensional space by PCA. This operation preserves all linear structures of dimension less than 8 in the feature space.

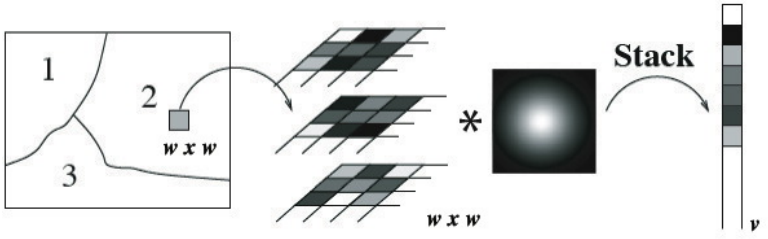

Figure 3: The construction of texture features: A $w \times w$ window of each of the three $L^*a^*b^*$ channels is convoluted with a Gaussian and then all channels are stacked into a single vector v.

3.2 Initialization with Superpixels

Given the feature vectors extracted from an image, one "naive" approach would be to directly apply Algorithm 1, and segment the pixels based on the grouping of the feature vectors. Figure 4 shows one such result. Notice that the resulting segmentation merges pixels near the strong edges into a single segment. This should be expected from the compression perspective, since windows across the boundary of two segments have significantly different structures from the (homogeneous) textures within those segments [12]. However, such a segmentation does not agree well with human perception.

In order to group edge pixels appropriately, we preprocess an image with a low-level segmentation based on local cues such as color and edges. That is, we oversegment the image into (usually several hundred) small, homogeneous regions, known as *superpixels*, which has been generally recommended for all region merging algorithms in [12]. Such low-level segmentation

[7] Equivalently, one can also use the $L^*u^*v^*$ metric.

[8] We did not test window sizes larger than 9 pixels, as the MATLAB implementation will run out of memory processing such texture vectors from a typical 320×240 color image. The current version of MATLAB has a 2GB memory limit imposed by the software.

Figure 4: Two segmentation results of the left original using Algorithm 1 with different ε's. Notice that the pixels near the boundaries of segments are not grouped correctly.

can be effectively computed using K-Means or Normalized-Cuts (NCuts) [25] with a conservative homogeneity threshold. In this paper, we use the publicly available superpixel code [20].

Since the superpixel segmentation respects strong edges in an image (see Figure 5 middle), it does not suffer from the misassignment of edge pixels seen in Figure 4. All pixels in a given superpixel are assigned as one segment initially, forcing the subsequent texture-based segmentation to group boundary pixels together with the interior pixels.

Another benefit from the superpixel preprocessing is a significant reduction in the computation required later to find the optimal segmentation. Using superpixel segments as initial grouping, the algorithm only needs to search amongst several hundred superpixels, instead of all image pixels.

3.3 Minimizing the Coding Length

Taking the superpixels as the initial segments, we then construct texture vectors for the pixels in each superpixel. To reduce the data size, one may also sample only a portion of the pixels to represent the distribution of the texture for each superpixel. Particularly, texture vectors at the boundary of a superpixel may not correctly represent the (homogeneous) texture information in the interior of the superpixel, but a combination of two textures across the superpixel and its neighboring one. Therefore, one may only sample texture vectors from the interior of each superpixel.[9] However, our experiment shows that the heuristic modification may not necessarily improve the overall performance of the algorithm, which will be demonstrated in Section 4. For clarity, we only demonstrate results using all pixels of an image in this section.

After the texture vectors are sampled from an image, one may directly apply Algorithm 1 to compress these texture features and obtain a segmentation. Nevertheless, in order to enforce that the resulting segmentation consists of connected segments, we impose an additional spatial constraint that two segments S_i and S_j can be merged together only if they are adjacent in the 2D image. Thus, we need to construct and maintain a *region adjacency graph* (RAG) G in the clustering process, which is popularly used in other *merge-and-split* type segmentation methods [13]. We represent the RAG using an adjacency list $G\{i\}$ for each segment S_i. Index j is in the set $G\{i\}$ if the segment S_j is a neighbor of S_i. At each iteration, the algorithm searches for a pair of adjacent segments S_i and S_j which lead to maximal decrease in the total coding length. Note, however, that in some applications such as image compression, disconnected segments may be allowed or even desirable. In this case, one can simply discard the adjacency constraint in the implementation.

Figure 5 shows an example of the combined segmentation process. In this example, we find that all feature vectors approximately lie in a 6D subspace in an 8D feature space. Each segment can be well modeled as a 1D to 4D degenerate Gaussian. Figure 6 plots the singluar values of two representative segments. This validates our original assumptions about the distributions of the texture features.

[9] In case a superpixel only consists of boundary pixels, use these pixels anyway.

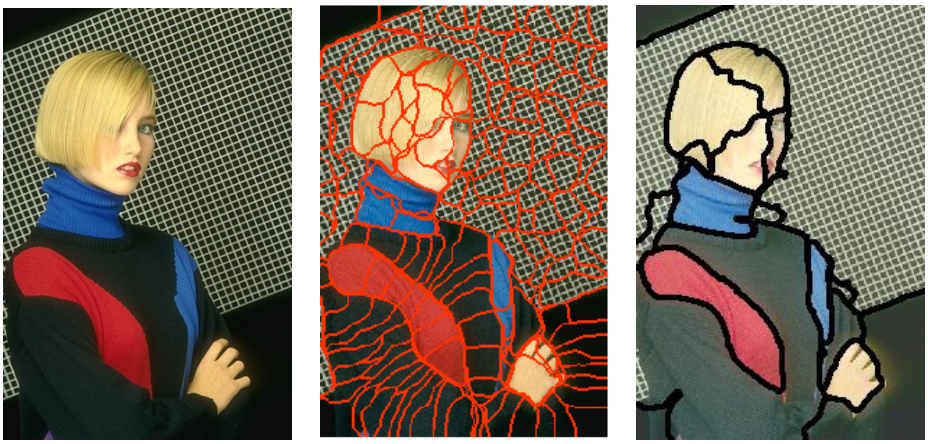

Figure 5: The segmentation pipeline. Left: Original. Middle: Superpixels obtained from low-level segmentation. Right: Segments obtained by minimizing the coding length with $\varepsilon = 0.02$.

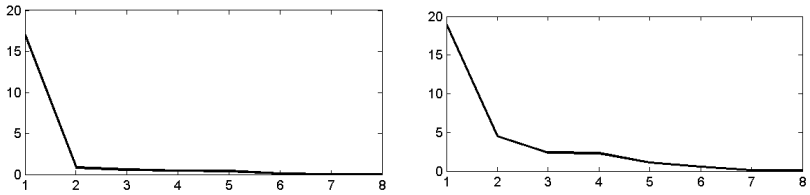

Figure 6: Singluar values of the feature vectors drawn respectively from two image segments in Figure 5 right.

3.4 Choosing the Distortion

As discussed in the introduction, the distortion ε^2 effectively sets the quantization scale at which we segment an image. Figure 7 shows the segmentation of several images under different values of ε. As the figure suggests, a single ε will not give good performance across a widely varying data set such as the Berkeley image segmentation database. Differences in the contrast of the foreground and background, lighting conditions, and image category cause the distribution of the texture features to vary significantly from image to image. For example, segmenting animals from the background tends to require a small ε due to their natural camouflage, whereas human portraits often allow a larger ε, as human faces and clothes differ more strongly from the (man-made) surroundings. In this way, the distortion ε can be linked to the concept of how "salient" an object is in an image and how much "attention" is needed to segment the object.

There are several ways to adaptively choose ε to achieve good segmentation for each image. For example, if a desired number of segments is known *a priori*, we can search a range of ε values for the one that gives the desired number of segments. When such information is not available *a priori*, which is the case for image segmentation, a formal way in information theory to estimate the distortion parameter is to minimize the following cost function:

$$\varepsilon^* \doteq \arg\min_{\varepsilon \in \mathcal{E}} \{L^s(V, \varepsilon) + \lambda N D \log_2(\varepsilon)\}, \tag{5}$$

where λ is a balancing parameter provided by the user. Notice that the first term $L^s(V, \varepsilon)$ decreases as ε increases, and at the same time the second term $ND\log_2(\varepsilon)$ increases. Hence, the expression essentially seeks a balance point between the coding length of the data and the complexity of the model measured by $ND\log_2(\varepsilon)$.

In our experiment, we found that (5) can accurately recover the true value of ε for the simulated mixture Gaussian models by simply setting $\lambda = 1$. However, in terms of image segmentation on real natural images, the estimated ε^* tends to oversegment the images. Our explanation of this performance deterioration between synthesized data and real images is that the distribution of (finite) texture vectors in a real image simply does not satisfy an exact mixture Gaussian model. Therefore, it is not surprising for the theoretical Criterion (5) to use more Gaussian models to approximate this distribution than what human perception may segment the image.

In this work, we choose to heuristically select the scale by stipulating that feature distributions in adjacent regions must be sufficiently dissimilar. One simple measure of dissimilarity, which we adopt in this paper, is the distance between the means

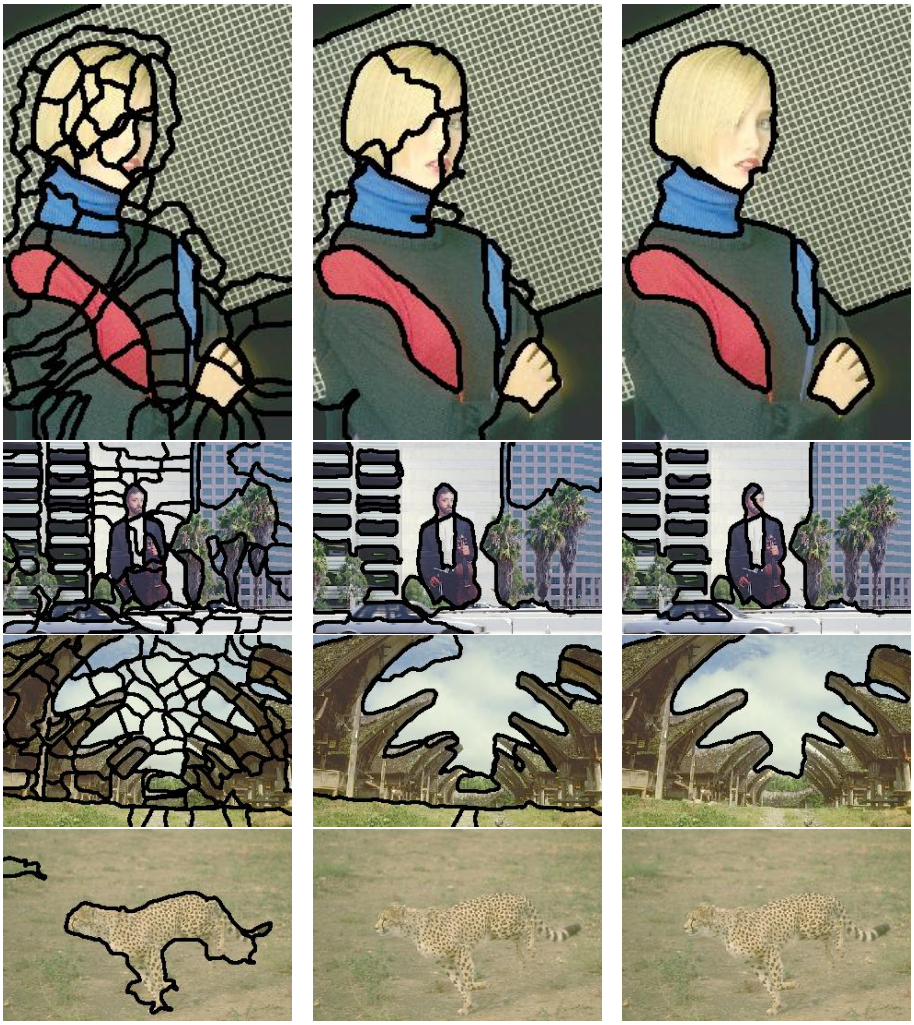

Figure 7: Segmentation results under different ε. Left column: $\varepsilon = 0.001$. Middle column: $\varepsilon = 0.02$. Right column: $\varepsilon = 0.05$. Notice that the algorithm merges all pixels into one group in the last two tests of the Leopard image.

of the *adjacent* segments (as Gaussian clusters). We gradually increase ε until the minimal distance between the means is larger than a preselected threshold γ, giving the most refined segmentation which satisfies the constraint. We note that increasing ε typically causes the number of segments to decrease to achieve a new shortest coding length. We may therefore use the segmentation computed with a smaller ε to initialize the merging process with a larger ε, allowing us to search for the optimal ε efficiently.

It may seem that we have merely replaced one free parameter, ε, with another, γ. This replacement has two strong advantages, however. Experimentally we find that even with a single fixed value of γ the algorithm can effectively adapt to all image categories in the Berkeley database, and achieve segmentation results that are consistent with human perception. Furthermore, the appropriate γ can be estimated empirically from human segmentations, whereas ε cannot. This heuristic thresholding method is similar in spirit to several robust techniques in computer vision for estimating mixture models, *e.g.*, the Hough transform and RANSAC.

The complete segmentation process is specified as Algorithm 2. In terms of speed, on a typical 3GHz Intel PC, the MATLAB implementation of the CTM algorithm on a 320×240 color image takes about two minutes to preprocess superpixels, and less than one minute to minimize the coding length of the features.

Algorithm 2: **(CTM: Compression-based Texture Merging).**

input: Image $I \in \mathbb{R}^{H\times W\times 3}$ in $L^*a^*b^*$ metric, reduced dimension D, window size w, distortion range $\mathcal{E}$, and minimum mean distance γ.

Partition I into superpixels $S_1, \ldots, S_K$. For pixel $p_i \in S_j$, initialize its label $l_i = j$.
Construct RAG $G\{1\}, \ldots, G\{K\}$ for the K segments $S_1, \ldots, S_K$.
Sample $w \times w$ Gaussian windows, and stack the resulting values into a feature vector $v_i \in \mathbb{R}^{3w^2}$.
Replace v_i with their first D principal components.
for all $\varepsilon \in \mathcal{E}$ in ascending order **do**
  for all initial segments S_i, $i = 1, \ldots, K$ **do**
    Compute $L^s(S_i, \varepsilon)$.
    for all $j \in G\{i\}$ **do**
      $U_{ij} \doteq L^s(S_i, \varepsilon) + L^s(S_j, \varepsilon) - L^s(S_i \cup S_j, \varepsilon)$
    end for
  end for
  while $U_{ij} \doteq \max\{U\} > 0$ **do**
    Merge S_i and S_j. Update arrays l, G, L, and U.
    Segment number $K \leftarrow K - 1$.
  end while
  if $\gamma \leq \min_{i,j\in G(i)}\{\|\text{mean}(S_i) - \text{mean}(S_j)\|\}$ **then**
    break.
  end if
end for

output: Final pixel labels $l_1, \ldots, l_{H\times W}$.

4 Experiments

In this section, we demonstrate the segmentation results of Algorithm 2 (CTM) on natural images in the Berkeley segmentation database [17], which also contains benchmark segmentation results obtained from human subjects. Two implementations of CTM are provided. The first one use all image pixels to construct texture vectors, which is denoted as CTM$_+$. The other one only samples from the interior of the superpixels but not at the boundaries, which is denoted as CTM$_-$.

We compare CTM against two unsupervised algorithms that have been made available publicly: Mean-Shift [1] and NCuts [25]. The comparison is based on four quantitative performance measures:

1. The Probabilistic Rand Index (PRI) [24] counts the fraction of pairs of pixels whose labellings are consistent between the computed segmentation and the ground truth, averaging across multiple ground truth segmentations to account for scale variation in human perception.

2. The Variation of Information (VoI) metric [19] defines the distance between two segmentations as the average conditional entropy of one segmentation given the other, and thus roughly measures the amount of randomness in one segmentation which cannot be explained by the other.

3. The Global Consistency Error (GCE) [17] measures the extent to which one segmentation can be viewed as a refinement of the other. Segmentations which are related in this manner are considered to be consistent, since they could represent the same natural image segmented at different scales.

4. The Boundary Displacement Error (BDE) [7] measures the average displacement error of boundary pixels between two segmented images. Particularly, it defines the error of one boundary pixel as the distance between the pixel and the closest pixel in the other boundary image.

Since all three methods are unsupervised, we use both the training and testing images for the evaluation. Due to memory issues with the NCuts implementation in MATLAB, all images are normalized to have the longest side equal to 320 pixels. We ran Mean-Shift with parameter settings (h_s, h_r) chosen at regular intervals of $[7, 16] \times [3, 23]$, and found that on the Berkeley database, $(h_s, h_r) = (13, 19)$ gives a good overall tradeoff between the above quantitative measures. We therefore use this parameter choice for our comparison. For NCuts, we choose the number of segments $K = 20$ to agree with the average number of segments from the human subjects.

To select a proper range for the γ parameter, which is the minimal distance threshold between mean vectors of two adjacent final segments, we collect the segmentation results given by the human subjects in the database. Figure 8 shows the distribution of the mean difference between two adjacent segments from the human segmentation results. Based on the distribution, 92% of adjacent segment pairs have the mean distance larger than 0.03 in the scale of normalized texture vectors. Accordingly, we choose to select γ that yields the best segmentation result from a range $[0.01, 0.03]$. Table 1 shows the average performance of the three methods, as well as that of the human subjects, over the entire database. Three different γ values are being tested, *i.e.*, $\gamma = 0.01$, 0.02, and 0.03.

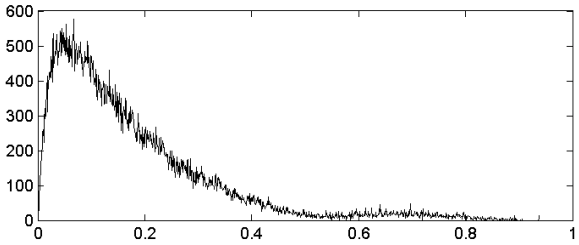

Figure 8: The distribution of the distance between the means of two adjacent segments based on the human segmentation result in the Berkeley database.

Table 1: Average performance on the Berkeley Database (bold indicates best among the algorithms). PRI ranges between $[0, 1]$, higher is better. VoI ranges between $[0, \infty)$, lower is better. GCE ranges between $[0, 1]$, lower is better. BDE ranges between $[0, \infty)$ in the unit of pixel, lower is better. CTM$_-$ represents Algorithm 2 without sampling boundary pixels, and CTM$_+$ represents the algorithm with boundary pixels.

	PRI	VoI	GCE	BDE
Humans	0.8754	1.1040	0.0797	4.994
CTM$_{-,\gamma=0.01}$	0.7719	2.7113	**0.1577**	**8.5027**
CTM$_{+,\gamma=0.01}$	0.7632	2.3076	0.1823	9.4529
CTM$_{-,\gamma=0.02}$	**0.7770**	2.1939	0.1893	8.7085
CTM$_{+,\gamma=0.02}$	0.7548	1.9835	0.1906	10.3205
CTM$_{-,\gamma=0.03}$	0.7663	2.0150	0.1921	9.4193
CTM$_{+,\gamma=0.03}$	0.7370	**1.9043**	0.1824	11.5289
Mean-Shift	0.7550	2.477	0.2598	9.7001
NCuts	0.7229	2.9329	0.2182	9.6038

We have also visually compared the segmentation results from both CTM$_-$ and CTM$_+$ with these γ values. For both CTM$_-$ and CTM$_+$, smaller γ's tend to generate more segments and oversegment the images, and larger γ's tend to generate less segments and hence undersegment the images. A visual comparison of the segmentation results from CTM$_{-,\gamma=0.02}$ and CTM$_{+,\gamma=0.02}$ is provided in Figures 9 – 14. Six different image categories are chosen to partition the Berkeley database into more relevant image groups for rendering, namely, *Landscape* (Figure 9), *Ocean* (Figure 10), *Urban* (Figure 11), *Animals* (Figure 12), *People* (Figure 13), and *Objects* (Figure 14).

Quantitatively, Table 1 shows that both CTM$_-$ and CTM$_+$ outperform Mean-Shift and NCuts in most indices. It is perhaps not surprising that both CTM$_-$ and CTM$_+$ significantly outperform the other two algorithms in terms of VoI, since we are optimizing an information-theoretic criterion. The results show that minimizing the coding length gives a segmentation which is closer to human segmentation in terms of conditional entropy, suggesting that perhaps human perception also approximately minimizes some measure of the compactness of representation.

To compare the difference of the CTM$_-$ and CTM$_+$ algorithms, we turn to the visual results in Figures 9 – 14. Under the same γ threshold, CTM$_-$ in general partitions an image into more detailed regions than CTM$_+$. In the animal category, this capability is important to segment certain animals with camouflage from their background (*e.g.*, the first example in Figure 12). On the other hand, CTM$_-$ also becomes more sensitive to the (gradual) changes of the color in images of sky and water (see the landscape and ocean categories for example). This observation is true for all other γ values.

Finally, we compare the four segmentation indices with different γ values in Table 1. The definition of the GCE and BDE indices leads to more significant penalies toward undersegmentation than oversegmentation. Particularly, GCE does not penalize oversegmentation at all, *i.e.*, the highest score is achived by assigning each pixel as an individual segment. As a result, $\text{CTM}_{-,\gamma=0.01}$ has returned the best GCE and BDE values among all the results in Table 1, but the segmentation in fact heavily oversegments the images, and its VoI value is one of the worst results in the table.

On the other hand, PRI and VoI seem to be more accurate in quantitatively measuring the goodness of an image segmentation comparing to human perception, where PRI leans toward oversegmentation and VoI toward undersegmentation.

To summarize in this comparison, we notice that on one hand, if we tune the algorithms to give the visually best match to human segmentation, none of the algorithms is a clear winner in terms of all four indices; on the other hand, none of the indices seems to be a better indicator of human segmentation than others, which suggests that human segmentation uses much more comprehensive cues. Nevertheless, the extensive visual demonstration and quantitative comparison does serve to validate our hypotheses that the distribution of texture features of natural images can be approximated by a mixture of (possibly degenerate) Gaussians. Also there is still much room to improve upon our method, for instance, using more sophisticated texture features or better heuristics for choosing the distortion. But regardless, the compression-based clustering algorithm should remain a powerful tool for exploiting redundancy and degeneracy for texture segmentation.

5 Conclusion and Discussion

In this paper, we have proposed that texture features of a natural image should be modeled as a mixture of almost degenerate distributions. We have introduced a new lossy compression based clustering algorithm, which is particularly effective for segmenting degenerate Gaussian distributions. We have shown that the algorithm is able to successfully segment natural images by harnessing the natural low-dimensional structures that are present in raw texture features such as Gaussian windows.

In addition, the lossy compression approach allows us to introduce the distortion as a useful parameter so that we can obtain a hierarchy of segmentations of an image at multiple quantization scales. We have proposed a simple heuristic to adaptively determine the distortion for each image if one wants to match the segmentation with that of humans.

The method may provide more relevant image segmentation for applications in lossy image compression. From a computer vision standpoint, the potential for tuning the distortion may also provide a means of addressing other problems such as image saliency detection and image categorization and retrieval. These are some of the challenging problems left open for future investigation.

Acknowledgment

The authors want to thank Parvez Ahammad for his valuable suggestion and literature references. We would also like to acknowledge that the pairwise steepest descent optimization (Algorithm 1) was first suggested by Professor Harm Derksen in the University of Michigan.

Appendix

We provide an alternative justification for the coding length function (3). In Section 2 of the paper, we know that given a set of N i.i.d. samples $V = (v_1, v_2, \ldots, v_N) \in \mathbb{R}^{D \times N}$ drawn from a multivariate Gaussian distribution, an estimate of the total number of bits needed to encode the data set V, subject to the distortion ε^2, is given by:

$$L(V) \doteq \frac{N+D}{2} \log_2 \det \left(I + \frac{D}{\varepsilon^2 N} \bar{V} \bar{V}^T \right) + \frac{D}{2} \log_2 (1 + \frac{\mu^T \mu}{\varepsilon^2}), \tag{6}$$

where $\mu \doteq \frac{1}{N} \sum_{i=1}^{N} v_i$ is the mean and $\bar{V} \doteq V - \mu \cdot 1_{1 \times N}$. In the paper, this estimate is obtained from the rate-distortion function of the Gaussian source. In this section, we show that the same formula gives an *upper bound* of the coding length of V by viewing V as a finite number of samples drawn from a subspace, without explicitly assuming the Gaussian model.

A Zero-Mean Case

For simplicity, we first study the case in which the data V is zero mean, *i.e.*, $\mu = 0$, and we leave the non-zero mean case to the next subsection.

Consider the singular value decomposition (SVD) of the data matrix $V = U\Sigma W^T$, where $U \in \mathbb{R}^{D\times D}$, $\Sigma \in \mathbb{R}^{D\times N}$, and $W \in \mathbb{R}^{N\times N}$. Let $B = (b_{ij}) = \Sigma W^T$. If the data vectors in V lie on a subspace of dimension $d < D$, the first d columns of $U = (u_{ij})$ will form a basis for this subspace. In general, the D columns of U form a basis for the data space, and the column vectors of B give the coordinates of the vectors with respect to this basis.

For coding purpose, we store the approximated matrices $U + \delta U$ and $B + \delta B$. The matrix V can be recovered as

$$V + \delta V \doteq (U + \delta U)(B + \delta B) = UB + \delta UB + U\delta B + \delta U\delta B. \tag{7}$$

When ε is small relative to the data V, $\delta V \approx \delta UB + U\delta B$ as entries of $\delta U\delta B$ are negligible. The squared error introduced to the entries of V are

$$\sum_{i,j} \delta v_{ij}^2 = \mathbf{tr}\big(\delta V \delta V^T\big) \approx \mathbf{tr}\big(U\delta B\delta B^T U^T + \delta UBB^T\delta U^T + \delta UB\delta B^T U^T + U\delta BB^T\delta U^T\big).$$

We may further assume that the coding errors δU and δB are zero-mean independent random variables. Using the fact that $\mathbf{tr}(AB) = \mathbf{tr}(BA)$, the expected squared error becomes

$$\mathbb{E}(\mathbf{tr}\big(\delta V\delta V^T)\big) \quad = \quad \mathbb{E}\big(\mathbf{tr}(\delta B\delta B^T)\big) + \mathbb{E}\big(\mathbf{tr}(\Sigma^2\delta U^T\delta U)\big).$$

Now, let us encode each entry b_{ij} with a precision $\varepsilon' = \frac{\varepsilon}{\sqrt{D}}$ and u_{ij} with a precision $\varepsilon_j'' = \frac{\varepsilon\sqrt{N}}{\sqrt{\lambda_j D}}$, where λ_j is the jth eigenvalue of VV^T.[10] This is equivalent to assume that the error δb_{ij} is uniformly distributed in the interval $\big[-\frac{\varepsilon}{\sqrt{D}}, \frac{\varepsilon}{\sqrt{D}}\big]$ and δu_{ij} is uniformly distributed in the interval $\big[-\frac{\varepsilon\sqrt{N}}{\sqrt{\lambda_j D}}, \frac{\varepsilon\sqrt{N}}{\sqrt{\lambda_j D}}\big]$. Under such a coding precision, it is easy to verify that

$$\mathbb{E}\big(\mathbf{tr}(\delta V\delta V^T)\big) \le \frac{2\varepsilon^2 N}{3} < \varepsilon^2 N. \tag{8}$$

Then the mean squared error per vector in V is

$$\frac{1}{N}\mathbb{E}\big(\mathbf{tr}(\delta V\delta V^T)\big) < \varepsilon^2. \tag{9}$$

The number of bits to store the coordinates b_{ij} with precision $\varepsilon' = \frac{\varepsilon}{\sqrt{D}}$ is

$$\begin{aligned}
&\sum_{i=1}^{D}\sum_{j=1}^{N}\frac{1}{2}\log_2\Big(1 + \big(\frac{b_{ij}}{\varepsilon'}\big)^2\Big) = \frac{1}{2}\sum_{i=1}^{D}\sum_{j=1}^{N}\log_2\Big(1 + \frac{b_{ij}^2 D}{\varepsilon^2}\Big)\\
\le\ & \frac{N}{2}\sum_{i=1}^{D}\log_2\Big(1 + \frac{D\sum_{j=1}^{N} b_{ij}^2}{N\varepsilon^2}\Big) = \frac{N}{2}\sum_{i=1}^{D}\log_2\Big(1 + \frac{D\lambda_i}{N\varepsilon^2}\Big).
\end{aligned}$$

In the above inequality, we have applied the following inequality:

$$\frac{\log(1+a_1) + \log(1+a_2) + \cdots + \log(1+a_n)}{n} \le \log\Big(1 + \frac{a_1 + a_2 + \cdots + a_n}{n}\Big) \tag{10}$$

for nonnegative real numbers $a_1, a_2, \ldots, a_n \ge 0$.

Similarly, the number of bits to store the entries of the singular vectors u_{ij} with precision $\varepsilon'' = \frac{\varepsilon\sqrt{N}}{\sqrt{\lambda_i D}}$ is

$$\begin{aligned}
&\sum_{i=1}^{D}\sum_{j=1}^{D}\frac{1}{2}\log_2\Big(1 + \big(\frac{u_{ij}}{\varepsilon''}\big)^2\Big) = \frac{1}{2}\sum_{i=1}^{D}\sum_{j=1}^{D}\log_2\Big(1 + \frac{u_{ij}^2 D^2\lambda_j}{N\varepsilon^2}\Big)\\
\le\ & \frac{D}{2}\sum_{j=1}^{D}\log_2\Big(1 + \frac{D^2\lambda_j\sum_{i=1}^{D} u_{ij}^2}{N\varepsilon^2}\Big) = \frac{D}{2}\sum_{j=1}^{D}\log_2\Big(1 + \frac{D\lambda_j}{N\varepsilon^2}\Big).
\end{aligned}$$

Thus, for U and B together, we need a total of

$$L(V) = \frac{N+D}{2}\sum_{i=1}^{D}\log_2\Big(1 + \frac{D\lambda_i}{N\varepsilon^2}\Big) = \frac{N+D}{2}\log_2\det\big(I + \frac{D}{N\varepsilon^2}VV^T\big). \tag{11}$$

We thus have proved the statement given in the beginning of this section: $L(V) = (N+D)R(\varepsilon, V)$ gives an upper bound on the number of bits needed to encode V.

[10] Notice that ε_j'' normally does not increase with the number of vectors N, because λ_j increases proportionally with N.

B Non-Zero Mean Case

In the above analysis, we have assumed that the given vectors $V = (v_1, v_2, \ldots, v_N)$ are zero-mean. In general, these vectors may have a non-zero mean. In other words, the points represented by these vectors may lie in an affine subspace, instead of a linear subspace that passes through the origin.

In case V is not zero mean, let $\mu \doteq \frac{1}{N}\sum_{i=1}^{N} v_i \in \mathbb{R}^D$ and define the matrix

$$\bar{V} \doteq V - \mu \cdot 1_{1\times N} = (v_1 - \mu, v_2 - \mu, \ldots, v_N - \mu) \in \mathbb{R}^{D\times N}. \tag{12}$$

The coding length for the zero-mean part $\bar{V}$ obviously follows (11) given in the previous section.

Next, let $\bar{V} = U\Sigma W^T \doteq UB$ be the singular value decomposition of $\bar{V}$, and let $\delta U, \delta B, \delta\mu$ be the error in coding U, B, μ, respectively. Then the error induced on the matrix V is

$$\delta V = \delta\mu \cdot 1_{1\times N} + U\delta B + \delta U B. \tag{13}$$

Assuming that $\delta U, \delta B, \delta\mu$ are zero-mean independent random variables, the expected total squared error is

$$\mathbb{E}\big(\mathbf{tr}(\delta V \delta V^T)\big) \quad = \quad N\mathbb{E}(\delta\mu^T\delta\mu) + \mathbb{E}\big(\mathbf{tr}(\delta B \delta B^T)\big) + \mathbb{E}\big(\mathbf{tr}(\Sigma\delta U^T \delta U)\big). \tag{14}$$

We encode entries of B and U with the same precision as before. We encode each entry μ_i of the mean vector μ with the precision $\varepsilon' = \frac{\varepsilon}{\sqrt{D}}$ and assume that the error $\delta\mu_i$ is a uniform distribution in the interval $\big[-\frac{\varepsilon}{\sqrt{D}}, \frac{\varepsilon}{\sqrt{D}}\big]$. Then we have $N\mathbb{E}(\delta\mu^T\delta\mu) = \frac{N\varepsilon^2}{3}$. Using equation (8) for the zero-mean case, the total squared error satisfies

$$\mathbb{E}\big(\mathbf{tr}(\delta V \delta V^T)\big) \le \frac{N\varepsilon^2}{3} + \frac{2N\varepsilon^2}{3} = N\varepsilon^2. \tag{15}$$

Then the mean squared error per vector in V is still bounded by ε^2:

$$\frac{1}{N}\mathbb{E}\big(\mathbf{tr}(\delta V \delta V^T)\big) \le \varepsilon^2. \tag{16}$$

Now in addition to the $L(\bar{V})$ bits needed to encode U and B, the number of bits needed to encode the mean vector μ with precision $\varepsilon' = \frac{\varepsilon}{\sqrt{D}}$ is

$$\sum_{i=1}^{D} \frac{1}{2}\log_2\Big(1 + \big(\frac{\mu_i}{\varepsilon'}\big)^2\Big) = \frac{1}{2}\sum_{i=1}^{D}\log_2\Big(1 + \frac{D\mu_i^2}{\varepsilon^2}\Big) \le \frac{D}{2}\log_2\Big(1 + \frac{\mu^T\mu}{\varepsilon^2}\Big), \tag{17}$$

where the last inequality is from the inequality (10).

Thus, the total number of bits needed to store V is

$$L(V) \quad = \quad \frac{N+D}{2}\log_2\det\big(I + \frac{D}{N\varepsilon^2}\bar{V}\bar{V}^T\big) + \frac{D}{2}\log_2\Big(1 + \frac{\mu^T\mu}{\varepsilon^2}\Big). \tag{18}$$

Notice that if V is actually zero-mean, we have $\mu = 0$, $\bar{V} = V$, and the above expression for $L(V)$ is exactly the same as before.

References

[1] D. Comanicu and P. Meer. Mean shift: A robust approach toward feature space analysis. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 24:603–619, May 2002. 1, 5, 9

[2] T. Cover and J. Thomas. *Elements of Information Theory*. Wiley Series in Telecommunications, 1991. 3, 4

[3] A. Dempster, N. Laird, and D. Rubin. Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society*, 39(B):1–38, 1977. 3

[4] Y. Deng, B. Manjunath, and H. Shin. Color image segmentation. In *Proceedings of International Conference on Computer Vision and Pattern Recognition*, 1999. 1

[5] P. Felzenszwalb and D. Huttenlocher. Efficient graph-based image segmentation. *International Journal on Computer Vision*, September 2004. 1

[6] M. Figueiredo and A. Jain. Unsupervised learning of finite mixture models. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 24(3):1–16, 2002. 3

[7] J. Freixenet, X. Munoz, D. Raba, J. Marti, and X. Cuff. Yet another survey on image segmentation. In *Proceedings of European Conference on Computer Vision*, 2002. 2, 9

[8] T. Gevers and A. Smeulders. Combining region splitting and edge detection through guided Delaunay image subdivision. In *Proceedings of International Conference on Computer Vision and Pattern Recognition*, 1997. 1

[9] Z. Ghahramani and G. E. Hinton. The EM algorithm for mixtures of factor analyzers. *Technical Report CRG-TR-96-1, Department of Computer Science, University of Toronto*, 1996. 3

[10] R. Haralick and L. Shapiro. Image segmentation techniques. *Computer Vision, Graphics, and Image Processing*, 29(1):100–132, 1985. 1

[11] S. Kamvar, D. Klein, and C. Manning. Interpreting and extending classical agglomerative clustering methods using a model-based approach. Technical Report 2002-11, Stanford University Department of Computer Science, 2002. 4

[12] T. Kanungo, B. Dom, W. Niblack, and D. Steele. A fast algorithm for MDL-based multi-band image segmentation. In *Proceedings of International Conference on Computer Vision and Pattern Recognition*, 1994. 1, 2, 5

[13] L. Lucchese and S. Mitra. Color image segmentation: a state-of-the-art survey. In *Proceedings of the Indian National Science Academy*, 2001. 2, 6

[14] M. Madiman, M. Harrison, and I. Kontoyiannis. Minimum description length vs. maximum likelihood in lossy data compression. In *Proceedings of the 2004 IEEE International Symposium on Information Theory*, 2004. 3

[15] J. Malik, S. Belongie, T. Leung, and J. Shi. Contour and texture analysis for image segmentation. *International Journal on Computer Vision*, 43(1):7–27, 2001. 5

[16] B. Manjunath and R. Chellappa. Unsupervised texture segmentation using Markov random field models. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 13(5):478–482, 1991. 2

[17] D. Martin, C. Fowlkes, D. Tal, and J. Malik. A database of human segmented natural images and its application to evaluating segmentation algorithms and measuring ecological statistics. In *Proceedings of IEEE International Conference on Computer Vision*, pages 416–423, 2001. 9

[18] G. McLachlan and T. Krishnan. *The EM algorithm and extensions*. John Wiley & Sons, 1997. 3

[19] M. Meila. Comparing clusterings: an axiomatic view. In *Proc. International Conference on Machine Learning*, pages 577–584, 2005. 9

[20] G. Mori. Guiding model search using segmentation. In *Proceedings of IEEE International Conference on Computer Vision*, 2005. 6

[21] A. Oliva and A. Torralba. Modeling the shape of the scene: a holistic representation of the spatial envelope. *International Journal on Computer Vision*, 2001. 5

[22] B. Olshausen and D. Field. Natural image statistics and efficient coding. *Network: Computation in Neural Systems*, 7:333–339, 1996. 5

[23] D. Panjwani and G. Healey. Markov random field models for unsupervised segmentation of textured color images. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 17(10):939–954, 1995. 1, 2

[24] C. Pantofaru and M. Hebert. A comparison of image segmentation algorithms. Technical Report CMU-RI-TR-05-40, CMU, 2005. 2, 9

[25] J. Shi and J. Malik. Normalized Cuts and Image Segmentation. In *Proceedings of International Conference on Computer Vision and Pattern Recognition*, pages 731–737, 1997. 1, 2, 6, 9

[26] M. Tipping and C. Bishop. Probabilistic principal component analysis. *Journal of Royal Statistical Society: Series B*, 61(3):611–622, 1999. 3

[27] A. Tremeau and N. Borel. A region growing and merging algorithm to color segmentation. *Pattern Recognition*, 30(7):1191–1204, 1997. 1

[28] Z. Tu and S. C. Zhu. Image segmentation by data-driven markov chain monte carlo. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 24(5):657–673, 2002. 1, 2

[29] N. Ueda, R. Nakan, and Z. Ghahramani. SMEM algorithm for mixture models. *Neural Computation*, 12:2109–2128, 2000. 4

[30] J. Ward. Hierarchical grouping to optimize and objective function. *Journal of the American Statistical Association*, 58:236–244, 1963. 4

[31] S. Yu. Segmentation induced by scale invariance. In *Proceedings of International Conference on Computer Vision and Pattern Recognition*, 2005. 1

[32] S. Zhu, C. Guo, Y. Wu, and Y. Wang. What are textons. In *Proceedings of European Conference on Computer Vision*, 2002. 5

[33] S. Zhu and A. Yuille. Region competition: unifying snakes, region growing, and Bayes/MDL for multiband image segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 18(9):884–900, 1996. 1

[34] S. C. Zhu. Statistical modeling and conceptualization of visual patterns. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 25(6):691–712, 2003. 1

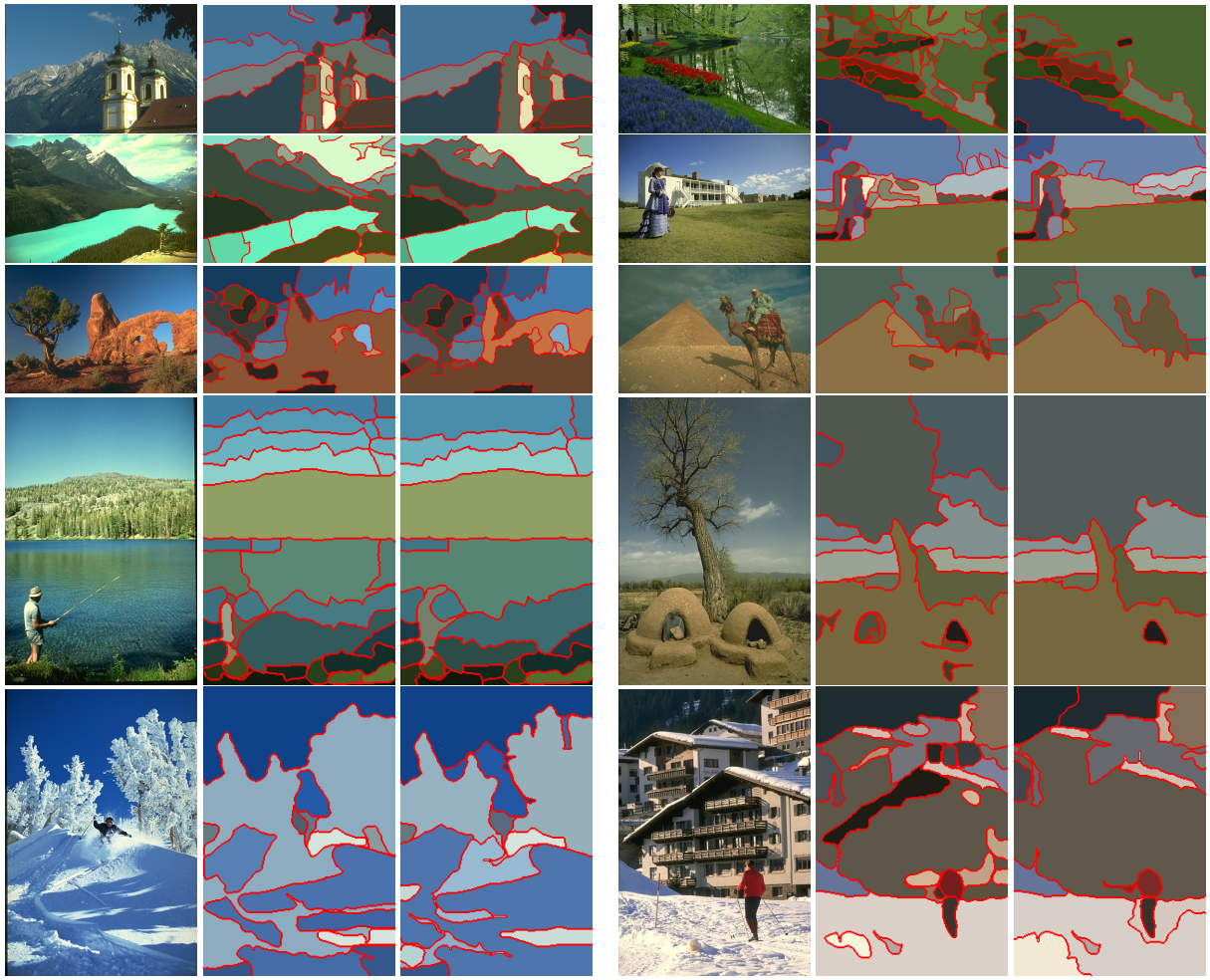

Figure 9: Representative segmentation results in the Category Landscape. Left: Original. Middle: CTM$_-$ with $\gamma = 0.02$ and the boundary pixels excluded. Right: CTM$_+$ with $\gamma = 0.02$ and the boundary pixels included.

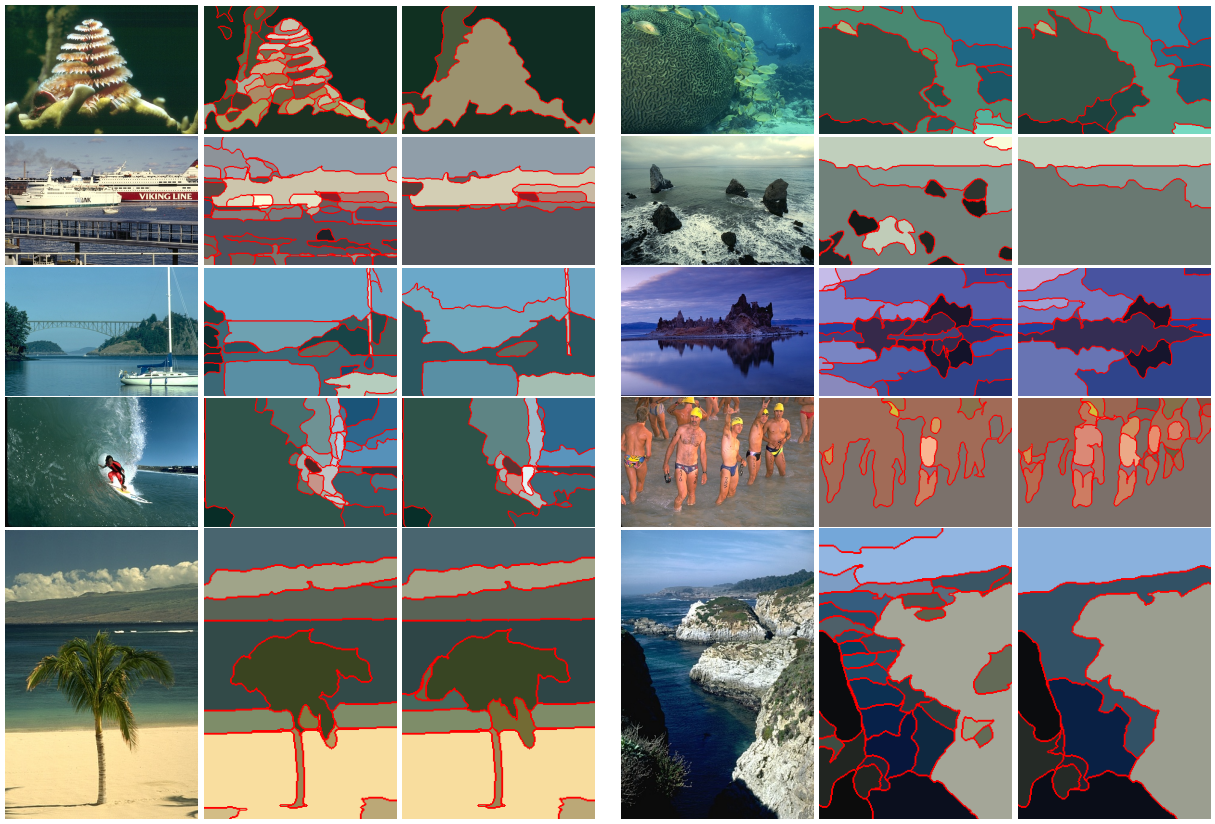

Figure 10: Representative segmentation results in the Category Ocean. Left: Original. Middle: CTM$_-$ with $\gamma = 0.02$ and the boundary pixels excluded. Right: CTM$_+$ with $\gamma = 0.02$ and the boundary pixels included.

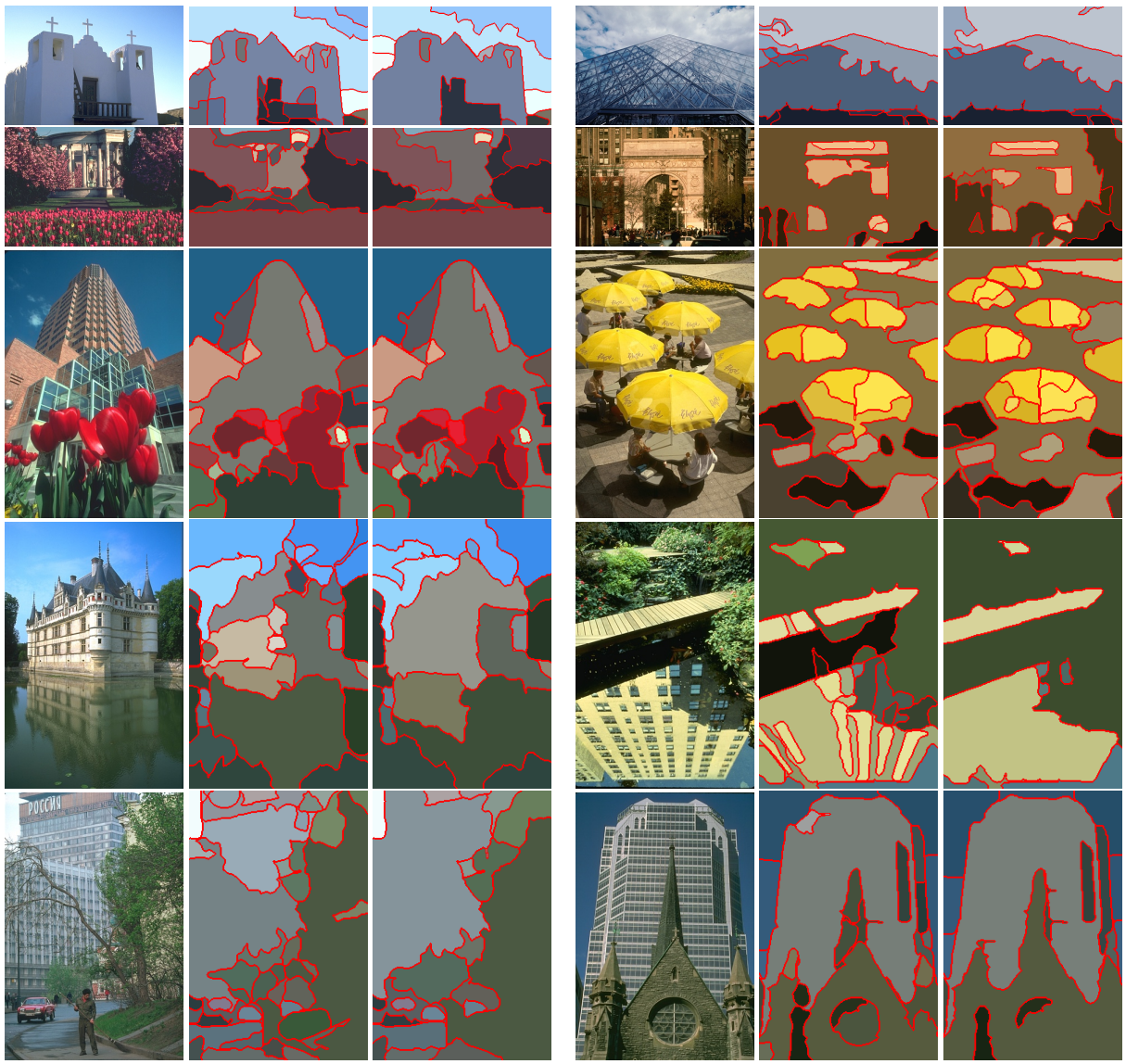

Figure 11: Representative segmentation results in the Category Urban. Left: Original. Middle: CTM$_-$ with $\gamma = 0.02$ and the boundary pixels excluded. Right: CTM$_+$ with $\gamma = 0.02$ and the boundary pixels included.

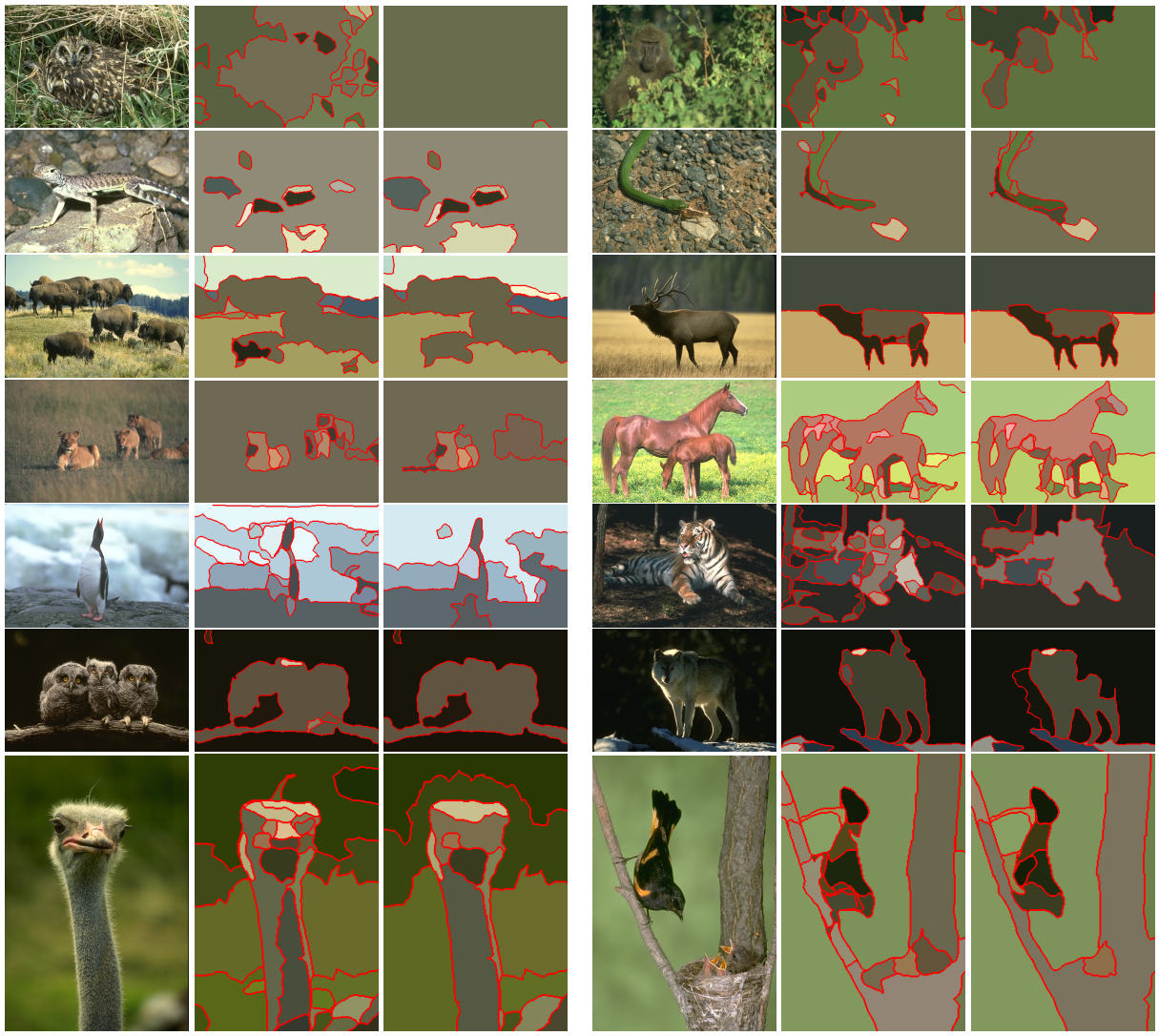

Figure 12: Representative segmentation results in the Category Animals. Left: Original. Middle: CTM$_-$ with $\gamma = 0.02$ and the boundary pixels excluded. Right: CTM$_+$ with $\gamma = 0.02$ and the boundary pixels included.

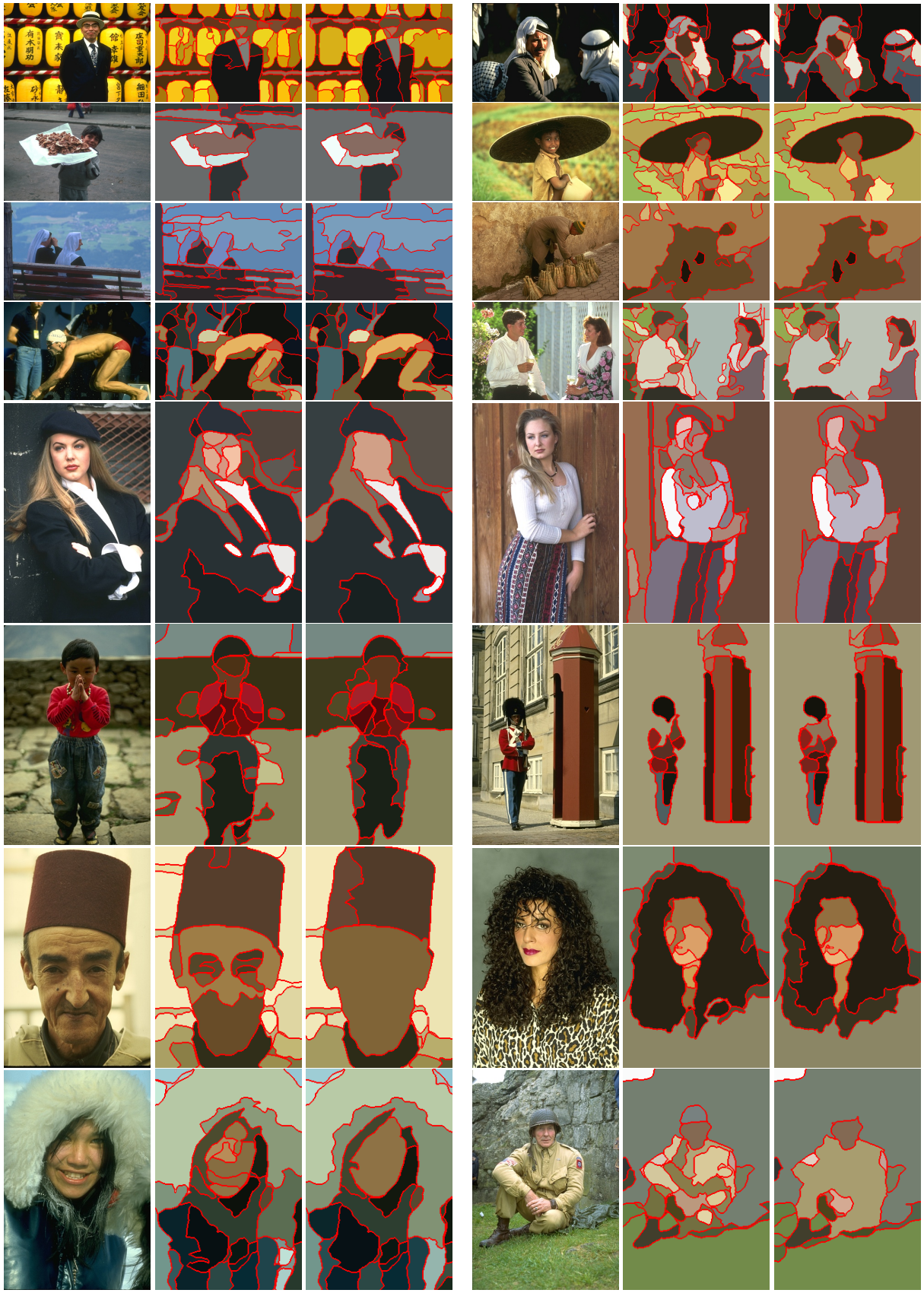

Figure 13: Representative segmentation results in the Category People. Left: Original. Middle: CTM$_-$ with $\gamma = 0.02$ and the boundary pixels excluded. Right: CTM$_+$ with $\gamma = 0.02$ and the boundary pixels included.

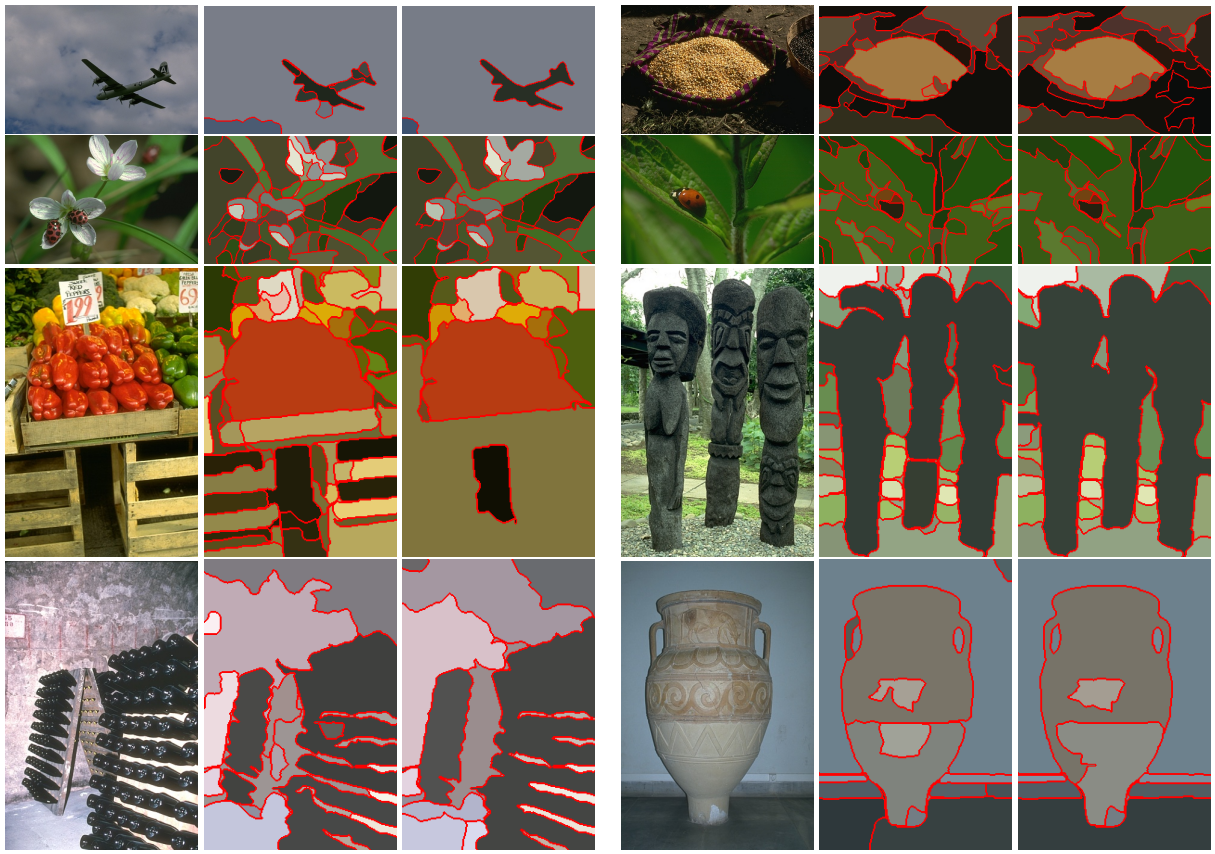

Figure 14: Representative segmentation results in the Category Objects. Left: Original. Middle: CTM$_-$ with $\gamma = 0.02$ and the boundary pixels excluded. Right: CTM$_+$ with $\gamma = 0.02$ and the boundary pixels included.