DOCUMENT PAGE CLASSIFICATION AND NONLINEAR DIFFUSION FILTERING FOR IMAGE SEGMENTATION AND NOISE REMOVAL

A Dissertation
Submitted to the Faculty
of
Purdue University
by
Xiaogang Dong

In Partial Fulfillment of the Requirements for the Degree of Doctor of Philosophy

May 2007

Purdue University
West Lafayette, Indiana
To my lovely wife,
Lingmin Zeng,
and my dear parents,
Zhaohong Wang
& Jiahui Dong
TABLE OF CONTENTS

<p>| LIST OF TABLES                      | v  |
| LIST OF FIGURES                    | vi |
| ABSTRACT                           | viii |
| 1 Multiscale Segmentation with Vector-Valued Nonlinear Diffusions on Arbitrary Graphs. | 1  |
| 1.1 Introduction.                  | 1  |
| 1.2 SIDEs for Vector-Valued Images Defined on an Arbitrary Domain. | 3  |
| 1.2.1 Scalar-Valued Diffusions on an Arbitrary Domain. | 3  |
| 1.2.2 Vector-Valued SIDEs on an Arbitrary Domain. | 8  |
| 1.3 Characterizing the Performance of a Segmentation Algorithm. | 11 |
| 1.4 Segmentation of Color Images.   | 11 |
| 1.4.1 Experiment 1: A Simple Shape. | 12 |
| 1.4.2 Experiment 2: A Complicated Shape. | 13 |
| 1.5 Texture Segmentation.          | 14 |
| 1.5.1 Feature Extraction.          | 14 |
| 1.5.2 Parameter Settings.          | 15 |
| 1.5.3 Experiment 1: Two Textures, Straight Boundary. | 18 |
| 1.5.4 Experiment 2: Two Textures, Random Boundary. | 18 |
| 1.5.5 Experiment 3: Comparison to [21]. | 18 |
| 1.5.6 Experiment 4: Comparison to [26]. | 21 |
| 1.6 Segmentation of Natural Images. | 22 |
| 1.7 Conclusions, Discussion, and Future Research. | 24 |
| 1.8 Acknowledgments.               | 26 |
| 2 Circle-Valued Nonlinear Diffusions with Application to Texture Segmentation | 27 |</p>
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1 Introduction</td>
<td>27</td>
</tr>
<tr>
<td>2.2 $S^1$-Valued Nonlinear Diffusion on an Arbitrary Domain</td>
<td>28</td>
</tr>
<tr>
<td>2.2.1 Diffusions on an Arbitrary Domain: Notation</td>
<td>28</td>
</tr>
<tr>
<td>2.2.2 $S^1$-Valued Nonlinear Diffusion</td>
<td>29</td>
</tr>
<tr>
<td>2.3 Texture Segmentation</td>
<td>33</td>
</tr>
<tr>
<td>2.3.1 Feature Extraction</td>
<td>33</td>
</tr>
<tr>
<td>2.3.2 Segmentation Examples</td>
<td>33</td>
</tr>
<tr>
<td>2.4 Conclusions</td>
<td>35</td>
</tr>
<tr>
<td>3 Approximate Methods for Constrained Total Variation Minimization</td>
<td>36</td>
</tr>
<tr>
<td>3.1 Problem Statement</td>
<td>36</td>
</tr>
<tr>
<td>3.2 Notation</td>
<td>38</td>
</tr>
<tr>
<td>3.3 Algorithm Description</td>
<td>39</td>
</tr>
<tr>
<td>3.4 Comparison to SOCP</td>
<td>45</td>
</tr>
<tr>
<td>3.5 Comparison to a Related Image Denoising Method [81]</td>
<td>46</td>
</tr>
<tr>
<td>4 A Document Page Classification Algorithm in Copy Pipeline</td>
<td>50</td>
</tr>
<tr>
<td>4.1 Introduction</td>
<td>50</td>
</tr>
<tr>
<td>4.2 Algorithm Overview</td>
<td>52</td>
</tr>
<tr>
<td>4.3 Algorithm Details</td>
<td>53</td>
</tr>
<tr>
<td>4.3.1 Color vs. Mono Classifier</td>
<td>53</td>
</tr>
<tr>
<td>4.3.2 Text vs. Nontext Classifier</td>
<td>54</td>
</tr>
<tr>
<td>4.3.3 Text/Mix vs. Picture/Photo Classifier</td>
<td>56</td>
</tr>
<tr>
<td>4.3.4 Picture vs. Photo Classifier</td>
<td>58</td>
</tr>
<tr>
<td>4.4 Experiment Results</td>
<td>59</td>
</tr>
<tr>
<td>4.5 Conclusions</td>
<td>60</td>
</tr>
<tr>
<td>LIST OF REFERENCES</td>
<td>61</td>
</tr>
<tr>
<td>Table</td>
<td>Description</td>
</tr>
<tr>
<td>-------</td>
<td>------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>1.1</td>
<td>Statistics of the errors (in percentages of the total number of pixels) in the color image segmentation experiments.</td>
</tr>
<tr>
<td>1.2</td>
<td>Statistics of the errors (in percentages of the total number of pixels) in the two-texture segmentation experiments.</td>
</tr>
<tr>
<td>1.3</td>
<td>Dependence of the errors on PARAMETER1 for the top image in Fig. 1.7(a).</td>
</tr>
<tr>
<td>1.4</td>
<td>Dependence of the errors on PARAMETER2 for the top image in Fig. 1.7(a).</td>
</tr>
<tr>
<td>1.5</td>
<td>Dependence of the errors on PARAMETER3 for the top image in Fig. 1.7(a).</td>
</tr>
<tr>
<td>1.6</td>
<td>The comparison of error percentages for 12 test images.</td>
</tr>
<tr>
<td>2.1</td>
<td>Statistics of the misclassified pixels(%) in Exp. 1.</td>
</tr>
<tr>
<td>2.2</td>
<td>Statistics of the misclassified pixels(%) in Exp. 2.</td>
</tr>
<tr>
<td>4.1</td>
<td>Classification rates for the training suite. Harmful misclassifications are indicated in boldface.</td>
</tr>
</tbody>
</table>
## LIST OF FIGURES

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.1</td>
<td>(a) A graph and its partition $S$ into three regions. (b) An image $u \in U_S$ where $U_S$ is the space of all images which are constant over each region of $S$. (c,d,e) Three vectors which form an orthonormal basis for $U_S$.</td>
<td>5</td>
</tr>
<tr>
<td>1.2</td>
<td>(a) A SIDE energy function $E$. (b) Its derivative $E'$.</td>
<td>9</td>
</tr>
<tr>
<td>1.3</td>
<td>(a) A test image; (b) its noisy version (normalized); (c) detected boundary, superimposed onto the noise-free image.</td>
<td>12</td>
</tr>
<tr>
<td>1.4</td>
<td>(a) A test image; (b) its noisy version (normalized); (c) detected boundary, superimposed onto the noise-free image.</td>
<td>13</td>
</tr>
<tr>
<td>1.5</td>
<td>A pair of Gabor functions with quadrature phase relation. (a) even impulse response; (b) corresponding frequency response</td>
<td>15</td>
</tr>
<tr>
<td>1.6</td>
<td>Segmentation results for two two-texture images with straight boundaries (top two rows) and two two-texture images with random boundaries (bottom two rows).</td>
<td>17</td>
</tr>
<tr>
<td>1.7</td>
<td>Segmentation of two five-texture images from [21].</td>
<td>19</td>
</tr>
<tr>
<td>1.8</td>
<td>(a) The empirical distribution of the percentage of error pixels for the 100 test images from [21]; (b) Comparison of the percentage of error blocks for our algorithm and the algorithm from [21].</td>
<td>20</td>
</tr>
<tr>
<td>1.9</td>
<td>Our segmentation results for images from [26]. Top row: Fig. 11(e) from [26]; second row: Fig. 11(f) from [26]; third row: Fig. 11(h) from [26]; bottom row: Fig. 11(i) from [26].</td>
<td>22</td>
</tr>
<tr>
<td>1.10</td>
<td>Segmentation of natural images from the Berkeley segmentation dataset [43,44].</td>
<td>25</td>
</tr>
<tr>
<td>2.1</td>
<td>(a) The SIDE energy function $E$. (b) Its derivative $E'$.</td>
<td>32</td>
</tr>
<tr>
<td>2.2</td>
<td>Some segmentation examples.</td>
<td>35</td>
</tr>
<tr>
<td>3.1</td>
<td>(a) Image $u$. (b) Segmentation $S$. (c) Segmentation $S'$.</td>
<td>38</td>
</tr>
<tr>
<td>3.2</td>
<td>(a) Noise-free “peppers” image. (b) Percentage difference between the TV for our split-and-merge algorithm and the optimal TV obtained via SOCP, for many different input PSNR levels (solid line); percentage difference between the TV for our merge only algorithm and the optimal TV (dashdot).</td>
<td>44</td>
</tr>
<tr>
<td>Figure</td>
<td>Page</td>
<td></td>
</tr>
<tr>
<td>--------</td>
<td>------</td>
<td></td>
</tr>
<tr>
<td>3.3</td>
<td>(a) Running time for SOCP; (b) running time for our algorithms.</td>
<td>44</td>
</tr>
<tr>
<td>3.4</td>
<td>(a) Output PSNR as a function of the input PSNR. (b) Output PSNR difference between our algorithm and SOCP.</td>
<td>46</td>
</tr>
<tr>
<td>3.6</td>
<td>(a) Noise-free “airplane” image. (b) Noisy “airplane” image with $\sigma = 0.05$ (PSNR = 26.0dB). (c) Noisy “airplane” image with $\sigma = 0.15$ (PSNR = 16.5dB).</td>
<td>48</td>
</tr>
<tr>
<td>3.7</td>
<td>Running times for our split-and-merge algorithm (solid lines) and the algorithm of [81] (dashed lines) to reach the same levels of PSNR.</td>
<td>49</td>
</tr>
<tr>
<td>4.1</td>
<td>Processing a fax in photo and text modes. Left to right: original, photo mode, text mode.</td>
<td>50</td>
</tr>
<tr>
<td>4.2</td>
<td>One possible implementation of our page classification algorithm in the existing copy pipeline.</td>
<td>52</td>
</tr>
<tr>
<td>4.3</td>
<td>Classification decision tree.</td>
<td>53</td>
</tr>
<tr>
<td>4.4</td>
<td>Color text vs. nontext: final feature decision map. The decision boundary (dashed lines) results in no harmful misclassifications of nontext into text.</td>
<td>58</td>
</tr>
</tbody>
</table>
ABSTRACT


We develop a real-time, strip-based, low-complexity document page classification algorithm, which can be used as a copy mode selector in the copy pipeline. It analyzes the scan images and classifies them into one of eight modes. Modes are the combinations of mono/color and text/mix/picture/photo settings. Mode classification is 29% accurate with misclassifications tending towards benign modes. The benefits of such a copy mode selector include improving copy quality, simplifying user interaction, and increasing copy rate.

We propose a novel family of nonlinear diffusions and apply it to the problem of segmentation of multivalued images defined on arbitrary graphs. These equations generalize previously proposed scalar-valued diffusion equations which have been used for segmenting grayscale images. We demonstrate the effectiveness of our new methods on a large number of color, texture and natural image segmentation tasks. We in addition introduce another extension of our diffusion equations which can process orientation images, i.e., images whose every pixel takes values on a circle.

We analyze a specific scalar-valued diffusion equation and show that it can be used as an approximate method for solving the constrained total variation minimization problem posed by Osher, Rudin, and Fatemi.
1. MULTISCALE SEGMENTATION WITH VECTOR-VALUED NONLINEAR DIFFUSIONS ON ARBITRARY GRAPHS.

1.1 Introduction.

The use of partial differential equations in image processing has been developed over the last twenty years for various tasks such as restoration, reconstruction, and segmentation, among others (see, for example, [1–4] and references therein). The input image is treated as the initial data for a diffusion-like differential equation [5–8]. The unknown in this equation is usually a function of three variables: two spatial variables (one for each image dimension) and scale. This function of three variables is called the scale-space, and is alternatively viewed as a collection of 2D images, one image for every value of the scale parameter.

The scale is sometimes also called time because of the similarity of such equations to evolution equations encountered in physics. In fact, one of the starting points of this line of investigation was the observation [5, 8] that smoothing an image with Gaussians of varying width is equivalent to solving the linear heat diffusion equation with the image as the initial condition. Specifically, the solution to the heat equation at time $t$ is the convolution of its initial condition with a Gaussian of variance $2t$. Gaussian filtering has been used both to remove noise and as a pre-processor for edge detection procedures [9]. It has serious drawbacks, however: it displaces and removes important image features, such as edges, corners, and T-junctions. The interpretation of Gaussian filtering as a linear diffusion led to the design of other, nonlinear, evolution equations, which better preserve these features [1, 7, 10–12]. For example, one motivation for the work in [7] is achieving both noise removal and edge enhancement through the use of an equation which in essence acts as an unstable inverse diffusion near edges and as a stable linear-heat-equation-like diffusion in homogeneous regions without edges.
The anisotropic diffusions introduced in [7,13] were the point of departure for the development of stabilized inverse diffusion equations (SIDEs) in [14]. It was shown in [14] that SIDEs may be viewed as a conceptually limiting case of the Perona-Malik diffusions. As shown in [14], the scale-space of such an equation is a family of piecewise-constant approximations of the original image, with larger values of the scale parameter \( t \) corresponding to approximations at coarser resolutions. Since the approximations are piecewise-constant, the scale-space can also be viewed as a fine-to-coarse family of segmentations of the image, and the process of evolution can be viewed as a region-merging procedure whereby pairs of regions get recursively merged to result in progressively coarser segmentations. SIDEs are therefore naturally suited to the problem of image segmentation. Their effectiveness for the segmentation of grayscale images was shown in [14]. It was also experimentally demonstrated in [14] that SIDEs are robust to noise outliers and blurring, and their optimality for certain estimation problems was proved in [15,16]. In addition, SIDEs lend themselves to faster algorithms than other evolution equations, since region merging reduces the dimensionality of the system during evolution.

Since they produce piecewise-constant approximations, however, the original SIDEs are not well suited to the problem of texture segmentation. This difficulty is common to many texture analysis problems, and is typically overcome with a preprocessing step which extracts features from a textured image [17–31]. The goal of feature extraction is to obtain data which is closer to being piecewise-constant than the original textured image. We use the output of a Gabor filter bank as the feature image, following a large body of literature (see, e.g., [17,20–23,26–29]) which has shown the effectiveness of Gabor features for texture analysis. If there are \( K \) filters in the Gabor filter bank, then each pixel of the feature image has \( K \) values. (We note here that other features can be used, and also that there exist many other situations which can give rise to vector-valued image data, such as color images or outputs from multiple sensors [32].) In Section 1.2, we introduce vector-valued diffusions which can segment such vector-valued images. We propose a novel, flexible way of weighting the features during the evolution which allows us, for example, to weight
fine-scale features more heavily at the beginning of the evolution—i.e., for small values of $t$—and to weight coarse-scale features more heavily during the later stages of the evolution.

We in addition develop a novel general way of introducing into our diffusion equations the information about the shapes of the image regions. This is done in Section 1.2 where our diffusion is interpreted as a gradient descent procedure for a certain energy functional. Both the structure of the underlying space where the gradient descent occurs, and the energy functional itself, are dependent on the image regions.

We illustrate our algorithm in Section 1.4 by applying it to segmenting noisy color images. Section 1.5 provides a thorough experimental evaluation of our algorithm. We show that it outperforms several existing methods in a variety of texture segmentation tasks. We give several examples of segmentation of natural images in Section 1.6.

1.2 SIDEs for Vector-Valued Images Defined on an Arbitrary Domain.

1.2.1 Scalar-Valued Diffusions on an Arbitrary Domain.

We define a real-valued image on an arbitrary finite set $\mathcal{N}$ of points as any function which assigns a real number to every point in $\mathcal{N}$. In our segmentation tasks, it is important to define adjacency relationships on the points in $\mathcal{N}$, and therefore we assume that $\mathcal{N}$ is the set of nodes of an undirected graph $G = (\mathcal{N}, \mathcal{L})$ where the set $\mathcal{L}$ of links consists of unordered pairs of distinct nodes. If $\{m, n\}$ is a link, we say that the nodes $m$ and $n$ are neighbors. For example, $G$ could be a finite 2D rectangular grid where each node has four neighbors: east, west, north, and south, as in Fig. 1.1(a).

We let $N$ be the total number of nodes and, without loss of generality, denote the nodes by the integers $1, 2, \ldots, N$, i.e., we assume that $\mathcal{N} = \{1, 2, \ldots, N\}$. An image $u$ can then be thought of as an $N$-dimensional vector: $u = (u_1, \ldots, u_N) \in \mathbb{R}^N$. We moreover use $u(t)$ to denote a parametric family of images defined on the set $\mathcal{N}$ for all values of a continuous-valued nonnegative scale parameter $t$, and we call the collection of images $\{u(t)\}_{t=0}^{\infty}$ a scale-space.
To describe the specific scale-space that we use in this paper, we need the following further definitions. We say that a set of nodes $R \subset N$ is a region if, for any two nodes $m, n \in R$, there exists a path between $m$ and $n$—i.e., a sequence of links of the form $\{m, m_1\}, \{m_1, m_2\}, \ldots, \{m_{k-1}, m_k\}, \{m_k, n\}$. Two disjoint regions $R_1, R_2$ are called neighbors if there exist two nodes $m \in R_1, n \in R_2$ which are neighbors, i.e., such that $\{m, n\} \in \mathcal{L}$. Given a partition $S = \{R_1, \ldots, R_I\}$ of the set $N$ into $I$ disjoint regions, we let $\text{NBR-PRS}$ be the set of all pairs of neighbor regions in $S$, and we let $\text{NBRS}(R_i)$ be the set of all regions in $S$ that are neighbors of a region $R_i \in S$. For example, a partition $S$ consisting of three regions is shown in Fig. 1.1(a). In this example, each region is a neighbor of the other two.

For any partition $S = \{R_1, \ldots, R_I\}$ of $N$, we let $U_S$ be the set of all piecewise constant images which are constant over each region $R_i \in S$. One such image, for the partition in Fig. 1.1(a), is shown in Fig. 1.1(b). We use $\mu_i(u)$ to denote the intensity of any such image $u$ within region $R_i$. Note that $U_S$ is a vector space. To impose a metric on this space, we define the following inner product.

$$\langle u, v \rangle \triangleq \sum_{i=1}^{I} a(R_i) \mu_i(u) \mu_i(v),$$

(1.1)

where $a(R_i)$ is a positive weight function which enables us to weight the contributions of various regions differently. For example, $a(R_i)$ may encode some information about the shape (or size, or location) of $R_i$.

We consider functionals $\mathcal{E}$ defined for images in $U_S$ which have the following form:

$$\mathcal{E}(u) = \sum_{\{R_i, R_j\} \in \text{NBR-PRS}} b(R_i, R_j) \ E(|\mu_i(u) - \mu_j(u)|),$$

(1.2)

where $b(R_i, R_j)$ is a positive weight function with $b(R_i, R_j) = b(R_j, R_i)$, and $E$ is such that $E(0) = 0$ and its derivative, $E'(x)$, is positive for $x > 0$. The function $b(R_i, R_j)$ allows us to assign different weights to different pairs of regions, according to, for example, the shape or length or the boundary between $R_i$ and $R_j$. 
Given an image \( u(0) \in U_S \), we generate a scale-space \( u(t) \) by solving the following gradient descent procedure for \( t > 0 \):

\[
\dot{u}(t) = -\nabla \mathcal{E}(u(t)),
\]  

(1.3)
where $\dot{u}(t)$ is the derivative of $u(t)$ with respect to the scale parameter $t$, and $\nabla$ stands for the gradient in the space $U_S$ equipped with the inner product (1.1). We now show how to implement this descent equation.

**Lemma 1** For $i = 1, \ldots, I$, let the image $e_i \in U_S$ be the indicator function of the region $R_i$ multiplied by $1/\sqrt{a(R_i)}$, i.e., let

$$
\mu_j(e_i) = \begin{cases} 
\frac{1}{\sqrt{a(R_i)}} & \text{if } j = i \\
0 & \text{if } j \neq i 
\end{cases} 
$$

These images then constitute an orthonormal basis for $U_S$.

**Proof.** Using Eq. (1.1), it is easily shown that these $I$ images are orthonormal. Since $U_S$ is an $I$-dimensional space, they form an orthonormal basis for $U_S$.

The three indicator functions for the regions of Fig. 1.1(a) are shown in Figs. 1.1(c-e). They form an orthogonal basis for the space $U_S$ defined by the partition $S$ of Fig. 1.1(a) (and if $a(R_i) = 1$ then this basis is orthonormal).

**Proposition 1** The gradient descent procedure (1.3) can be equivalently written as follows:

$$
\dot{\mu}_i = \frac{1}{a(R_i)} \sum_{R_j \in \text{NBRS}(R_i)} b(R_i, R_j) \frac{\mu_j - \mu_i}{|\mu_j - \mu_i|} E'(|\mu_j - \mu_i|), \; \text{for } i = 1, \ldots, I,
$$

where $E'$ is the derivative of $E$.

**Proof.** As shown in Lemma 1, the vectors $e_1, \ldots, e_I$ illustrated in Figs. 1.1(c-e) form an orthonormal basis for the space $U_S$. This means that any image $u(t)$ which is an element of the space $U_S$, can be represented as a linear combination of these basis vectors,

$$
u(t) = \sum_{i=1}^{I} c_i(t)e_i,
$$

where the $i$-th coefficient $c_i(t)$ is the inner product of the image with the $i$-th basis vector:

$$
c_i(t) = \langle u(t), e_i \rangle
$$

When there is no possibility of confusion, we abbreviate $\mu_i(u(t))$ as $\mu_i$. 

\[
\text{Eq. (1.1)} \quad \sum_{j=1}^{I} a(R_j) \mu_j(u(t)) \mu_j(e_i)
\]

\[
\text{Eq. (1.4)} \quad a(R_i) \mu_i(u(t)) \frac{1}{\sqrt{a(R_i)}}
\]

\[
= \sqrt{a(R_i)} \mu_i,
\]

where in the last expression we have abbreviated \( \mu_i = \mu_i(u(t)) \). Eq. (1.6) can be substituted into Eq. (1.2) to write the functional \( \mathcal{E} \) in terms of \( c_i \)'s. This yields the following gradient descent in the \( c_i \) coordinates:

\[
\dot{c}_i = -\frac{\partial \mathcal{E}}{\partial c_i} = -\frac{\partial}{\partial c_i} \left\{ \sum_{(R_i, R_j) \in \text{NBR-PRS}} b(R_i, R_j) E \left( \frac{1}{\sqrt{a(R_i)}} c_i - \frac{1}{\sqrt{a(R_j)}} c_j \right) \right\}
\]

\[
= -\sum_{R_j \in \text{NBR}(R_i)} \frac{b(R_i, R_j)}{\sqrt{a(R_i)}} \cdot \frac{c_i/\sqrt{a(R_i)} - c_j/\sqrt{a(R_j)}}{|c_i/\sqrt{a(R_i)} - c_j/\sqrt{a(R_j)}} E'(\frac{|c_i/\sqrt{a(R_i)} - c_j/\sqrt{a(R_j)}|}{E'(\frac{|\mu_j - \mu_i|}{\mu_j - \mu_i})}.
\]

Using Eq. (1.6) again to rewrite this in terms of \( \mu_i \)'s, we get:

\[
\sqrt{a(R_i)} \dot{\mu}_i = \sum_{R_j \in \text{NBR}(R_i)} \frac{b(R_i, R_j)}{\sqrt{a(R_i)}} \cdot \frac{\mu_j - \mu_i}{|\mu_j - \mu_i|} E'(\frac{|\mu_j - \mu_i|}{|\mu_j - \mu_i|}),
\]

which is the same as Eq. (1.5).

Note that the general paradigm of defining scale-spaces through various gradient descent procedures is standard [1,7,8,11,14], and is typically implemented on a four-neighbor grid (see Fig. 1.1(a)) using an energy functional of the form

\[
\sum_{\{k,n\} \in \text{NBR-P\$}} E(|u_k - u_n|).
\]

In this case, the gradient is taken in the whole space \( \mathbb{R}^N \) equipped with the standard inner product. Note that this is a special case of our framework, with each region consisting of a single pixel, with \( a \equiv 1 \) in Eq. (1.1), and \( b \equiv 1 \) in Eq. (1.2). A key novelty of our method consists of generalizing this pixel-based approach to arbitrary regions. Our generalization of the energy functional is Eq. (1.2), the standard inner product is generalized by the inner product (1.1), and the gradient descent occurs in the space \( U_S \) equipped with this new inner product.
1.2.2 Vector-Valued SIDEs on an Arbitrary Domain.

We have assumed so far that the image intensities are scalars. We now generalize our evolution equations (1.3, 1.5) to the case when each image intensity is a $K$-dimensional vector. In this case, an image $\mathbf{u}$ can be thought of as an $N \times K$-dimensional vector:

$$\mathbf{u} = (\tilde{u}_1, \ldots, \tilde{u}_N) \in \mathbb{R}^{N \times K},$$

where each intensity vector $\tilde{u}_n = (u_{n,1}, \ldots, u_{n,K})$ is in $\mathbb{R}^K$. An intensity vector can, for example, be the vector of red, green, and blue intensities for a color image, or correspond to $K$ features extracted from a texture image. Each entry of an intensity vector is called a component.

To define a metric on the space $\mathbb{R}^K$ of all intensity vectors, we use the following inner product between two intensity vectors $\tilde{u}_m$ and $\tilde{u}_n$:

$$\langle \tilde{u}_m, \tilde{u}_n \rangle_w \triangleq \sum_{k=1}^{K} w_k u_{m,k} u_{n,k},$$

(1.7)

where the positive weights $w = (w_1, \ldots, w_K)$ allow us to assign different relative importance to different components of an image. We denote the norm corresponding to this inner product by $\| \cdot \|_w$.

As previously, we fix a partition $\mathcal{S} = \{R_1, \ldots, R_I\}$ of the set $\mathcal{N}$ of nodes, and let $U_S$ be the set of all piecewise constant images which are constant over each region $R_i$ in the partition. We now use $\bar{\mu}_i(\mathbf{u})$ to denote the vector intensity of any such image $\mathbf{u}$ within region $R_i$. We define an inner product on the space $U_S$ by generalizing our definition (1.1) as follows:

$$\langle \mathbf{u}, \mathbf{v} \rangle \triangleq \sum_{i=1}^{I} a(R_i) \langle \bar{\mu}_i(\mathbf{u}), \bar{\mu}_i(\mathbf{v}) \rangle_w,$$

(1.8)

where $\langle \cdot \rangle_w$ is the inner product for the intensity vectors defined in Eq. (1.7).

We again generate a scale-space of images in $U_S$, starting with a given image $\mathbf{u}(0) \in U_S$ and using the descent procedure (1.3) where the functional $E$ is the following generalization of Eq. (1.2):

$$E(\mathbf{u}) = \sum_{\{R_i, R_j\} \in \text{NBR-PRS}} b(R_i, R_j) \ E(\|\bar{\mu}_i(\mathbf{u}) - \bar{\mu}_j(\mathbf{u})\|_w).$$

(1.9)
Proposition 2  The gradient descent procedure (1.3) for vector-valued images can be equivalently written as follows:

\[
\dot{\mu}_i = \frac{1}{a(R_i)} \sum_{R_j \in \text{NBR}(R_i)} b(R_i, R_j) \frac{\mu_j - \mu_i}{\|\mu_j - \mu_i\|} E'(\|\mu_j - \mu_i\|_w), \text{ for } i = 1, \ldots, I. 
\]  

(1.10)

Proof. Similar to the proof of Proposition 1.

We use a SIDE energy function \( E \) [14, 33, 34]—i.e., a function \( E \) such that

\[
E''(x) < 0 \text{ for } x > 0, 
\]

(1.11)

as illustrated in Fig. 1.2(a). In addition, we impose that

\[
\lim_{x \to 0} E'(x) = +\infty, 
\]

(1.12)

as shown in Fig. 1.2(b). For example, \( E(x) = \sqrt{x} \) has the desired shape. With this choice of the function \( E \), Eq. (1.10) is well suited for segmentation since it encourages the merging of pairs of neighbor regions. Specifically, as we presently show, the two properties (1.11) and (1.12) ensure that the solution of Eq. (1.10) is attracted to subspaces of \( U_S \) which have the form \( U_{i,j} = \{v : \tilde{\mu}_i(v) = \tilde{\mu}_j(v)\} \) where \( \{R_i, R_j\} \in \text{NBR-PRS} \). To see this, note that, if the solution \( u(t) \) is very close to one such space \( U_{i,j} \) and is away from all other spaces \( U_{i,j} \), then Eq. (1.12) implies that \( E'(\|\tilde{\mu}_j - \tilde{\mu}_i\|_w) \gg E'(\|\tilde{\mu}_j - \tilde{\mu}_i\|_w) \), and therefore the dynamics for both \( \tilde{\mu}_i \) and \( \tilde{\mu}_j \) will in this case be dominated by the term involving \( E'(\|\tilde{\mu}_j - \tilde{\mu}_i\|_w) \).

Denoting \( \nu_{ij} = \tilde{\mu}_j - \tilde{\mu}_i \), we then have, from Eq. (1.10):

\[
\dot{\nu}_{ij} = -\left(\frac{1}{a(R_i)} + \frac{1}{a(R_j)}\right) b(R_i, R_j) \frac{\tilde{\mu}_{ij}}{\|\tilde{\mu}_{ij}\|_w} E'(\|\nu_{ij}\|_w). 
\]

(1.13)
Let $x = \|\tilde{\nu}_{ij}\|_w$. Then Eq. (1.13) can be shown to imply
\[
\dot{x} = -CE'(x),
\]
where $C = \left(\frac{1}{a(R_i)} + \frac{1}{a(R_j)}\right) b(R_i, R_j) > 0$. Since $E'(x) > 0$, we have that $\dot{x} < 0$, and since $E'' < 0$ we have that
\[
\ddot{x} = -CE''(x)\dot{x} = C^2E''(x)E'(x) < 0,
\]
which means that $x$ will reach zero in finite time. When this happens, $\mu_i = \mu_j$, and we effectively have a partition which is coarser than the partition $S$ since now $R_i \cup R_j$ can be viewed as a single region. This motivates the following multiscale region-merging segmentation algorithm.

1. Set $t_{init} \leftarrow 0$.

2. Given a partition $S$ of $N$ and an image $u(t_{init}) \in U_S$, evolve Eq. (1.10) for $t > t_{init}$ until the time $t_{merge}$ when $\tilde{\mu}_i(u(t_{merge})) = \tilde{\mu}_j(u(t_{merge}))$ (numerically, when $\|\tilde{\mu}_i(u(t_{merge})) - \tilde{\mu}_j(u(t_{merge}))\|_w < \epsilon$ where $\epsilon$ is a small parameter) for some pair of neighbor regions $\{R_i, R_j\}$.

3. Remove regions $R_i$ and $R_j$ from $S$ and add region $R_i \cup R_j$:
\[
S \leftarrow S \setminus \{R_i, R_j\} \cup \{R_i \cup R_j\}.
\]

4. If the desired number of regions is reached, stop. Else, define the values of the functions $a$ and $b$ for the newly formed region, adjust the weights $w_k$ (if desired), assign $t_{init} \leftarrow t_{merge}$, and go to Step 2.

We point out that, while Eq. (1.10) is the gradient descent for the functional $E$ of Eq. (1.9), the solution of interest is not a minimum of $E$, because of the stopping rule in Step 4 of the above algorithm. Indeed, minimizing $E$ is trivial and is achieved by any constant image, i.e., by setting $\tilde{\mu}_i(u) = \tilde{\mu}_j(u)$ for all $i$ and $j$, to result in $E = 0$. Such a constant image will eventually be reached by the above algorithm if no stopping rule is used.
1.3 Characterizing the Performance of a Segmentation Algorithm.

We now describe the methodology for evaluating the performance of our algorithm in the color and texture segmentation experiments of Sections 1.4 and 1.5. If the true region is \( R_i \) whereas the region extracted by our algorithm is \( \hat{R}_i \), we define the mismatch \( M(R_i, \hat{R}_i) \) between the true and estimated regions as the set of all nodes which are in one of these regions but not in their intersection:

\[
M(R_i, \hat{R}_i) \triangleq (R_i \setminus \hat{R}_i) \cup (\hat{R}_i \setminus R_i).
\]

We assume in all our experiments that the target number of regions \( I \) is given. The mismatch between the ground truth partition \( S = \{R_1, \ldots, R_I\} \) and the partition \( \hat{S} = \{\hat{R}_1, \ldots, \hat{R}_I\} \) extracted by our algorithm is defined for any permutation \( \pi \) of the indexes \( 1, \ldots, I \) as follows:

\[
\text{MISMATCH}(S, \hat{S}, \pi) \triangleq \bigcup_{i=1}^{I} M(R_i, \hat{R}_{\pi(i)}).
\]

To characterize the performance of our segmentation algorithm, we choose the best match as follows:

\[
\pi^* \triangleq \arg \min_{\pi} \left| \text{MISMATCH}(S, \hat{S}, \pi) \right|.
\]

Then we define

\[
\text{ERROR-SET}(S, \hat{S}) \triangleq \text{MISMATCH}(S, \hat{S}, \pi^*)
\]

\[
\text{ERROR-PERCENTAGE}(S, \hat{S}) \triangleq \frac{|\text{ERROR-SET}(S, \hat{S})|}{|\hat{S}|} \cdot 100\%,
\]

i.e., the error percentage is defined as the ratio (in percent) of the number of mismatched pixels to the total number of pixels.

1.4 Segmentation of Color Images.

We first illustrate our segmentation algorithm by applying it to color images which are viewed as \( \mathbb{R}^3 \)-valued images with red, green, and blue components. In these experiments, we use \( E(x) = \sqrt{x} \), set \( a(R_i) \) to be the area \( |R_i| \) of region \( R_i \) (i.e., the number of nodes in
Fig. 1.3: (a) A test image; (b) its noisy version (normalized); (c) detected boundary, superimposed onto the noise-free image.

1.4.1 Experiment 1: A Simple Shape.

We start by applying our vector-valued SIDE to the color image in Fig. 1.3. The image in Fig. 1.3(a) consists of two regions: two of its three color components undergo an abrupt change at the boundary between the regions. More precisely, the \{red, green, blue\} component values are \{0.1, 0.6, 0.9\} for the background and \{0.6, 0.6, 0.5\} for the square. Each component is corrupted with independent white Gaussian noise whose standard deviation is 0.4. The resulting image (normalized in order to make every pixel of every component be between 0 and 1) is shown in Fig. 1.3(b). We evolve our vector-valued SIDE on the noisy image, until exactly two regions remain. The final boundary, superimposed onto the initial image, is depicted in Fig. 1.3(c). The algorithm is very accurate in locating the boundary: the error set occupies less than 0.5% of the pixels for this 100 × 100 image. This is a typical result, as confirmed by Table 1.1 which shows segmentation results for 100 different white
Table 1.1: Statistics of the errors (in percentages of the total number of pixels) in the color image segmentation experiments.

<table>
<thead>
<tr>
<th></th>
<th>Min</th>
<th>Max</th>
<th>Avg</th>
<th>Std</th>
</tr>
</thead>
<tbody>
<tr>
<td>Color Experiment 1</td>
<td>0.4%</td>
<td>1.3%</td>
<td>0.7%</td>
<td>0.2%</td>
</tr>
<tr>
<td>Color Experiment 2</td>
<td>0.5%</td>
<td>1.4%</td>
<td>0.8%</td>
<td>0.2%</td>
</tr>
</tbody>
</table>

Fig. 1.4: (a) A test image; (b) its noisy version (normalized); (c) detected boundary, superimposed onto the noise-free image.

Gaussian noise realizations with standard deviation 0.4 added to the noise-free image of Fig. 1.3(a). The average error in these experiments is 0.7%.

1.4.2 Experiment 2: A Complicated Shape.

A similar experiment, with the same level of noise, is conducted for a more complicated shape, whose image is in Fig. 1.4(a). The result of processing the noisy image of Fig. 1.4(b) is shown in Fig. 1.4(c). In this $100 \times 100$ image, 0.7% of the pixels are errors. The results of a Monte-Carlo experiment with 100 different noise realizations are recorded in the bottom row of Table 1.1, showing that the average error is 0.8%.
1.5 Texture Segmentation.

1.5.1 Feature Extraction.

The use of Gabor features [35, 36] is a well established strategy in the texture analysis literature [17, 20–23, 26–29]. We adopt Gabor energy features from [23, 27, 29]. These features depend on three parameters: orientation $\theta$, scale $\sigma$, and frequency $\omega_0$, which we abbreviate as $\rho \triangleq (\theta, \sigma, \omega_0)$. Given an image $f(x, y)$, the Gabor energy feature $u(x, y; \rho)$ at orientation $\theta$, scale $\sigma$, and frequency $\omega_0$ is calculated by filtering the image with a pair of Gabor filters $h_c(x, y; \rho)$ and $h_s(x, y; \rho)$ to obtain the filtered images $g_c(x, y; \rho)$ and $g_s(x, y; \rho)$, respectively. These filtered images are then combined via:

$$u(x, y; \rho) = \sqrt{g_c(x, y; \rho)^2 + g_s(x, y; \rho)^2}.$$  (1.14)

In our experiments, the Gabor filter pair is the following pair of filters with quadrature phase relation [23, 27, 29]:

$$h_c(x, y; \rho) = e^{-0.5\sigma^{-2}(\bar{x}^2+\bar{y}^2)} \cos (\omega_0 \bar{x}),$$
$$h_s(x, y; \rho) = e^{-0.5\sigma^{-2}(\bar{x}^2+\bar{y}^2)} \sin (\omega_0 \bar{x}),$$

where

$$\bar{x} = x \cos \theta + y \sin \theta,$$
$$\bar{y} = -x \sin \theta + y \cos \theta.$$

The frequency responses of these two filters are:

$$H_c(\omega_x, \omega_y; \rho) = \pi \sigma^2 \left\{ e^{-0.5\sigma^2[(\bar{\omega}_x+\omega_0)^2+\bar{\omega}_y^2]} + e^{-0.5\sigma^2[(\bar{\omega}_x-\omega_0)^2+\bar{\omega}_y^2]} \right\},$$
$$H_s(\omega_x, \omega_y; \rho) = \pi \sigma^2 \sqrt{-1} \left\{ e^{-0.5\sigma^2[(\bar{\omega}_x+\omega_0)^2+\bar{\omega}_y^2]} - e^{-0.5\sigma^2[(\bar{\omega}_x-\omega_0)^2+\bar{\omega}_y^2]} \right\},$$

where

$$\bar{\omega}_x = \omega_x \cos \theta + \omega_y \sin \theta,$$
$$\bar{\omega}_y = -\omega_x \sin \theta + \omega_y \cos \theta.$$
Fig. 1.5: A pair of Gabor functions with quadrature phase relation. (a) even impulse response; (b) corresponding frequency response; (c) odd impulse response; (d) corresponding frequency response

Note that \( H_c(\omega_x, \omega_y; \rho) \) is a real-valued, even function and \( H_s(\omega_x, \omega_y; \rho) \) is a purely imaginary-valued, odd function. Both these filters are bandpass filters centered at \( (\omega_0 \cos \theta, \omega_0 \sin \theta) \) in the frequency domain.

Fig. 1.5 shows the intensity map of a pair of Gabor filters in the space and frequency domains. The even Gabor filter \( h_c(x, y; \rho) \) is presented in Fig. 1.5(a), and the corresponding odd Gabor filter \( h_s(x, y; \rho) \) is in Fig. 1.5(c). Fig. 1.5(b) shows the real part of \( H_c(\omega_x, \omega_y; \rho) \) since it is a real function in the frequency domain. Similarly, Fig. 1.5(d) shows the imaginary part of \( H_s(\omega_x, \omega_y; \rho) \).

Feature selection for texture analysis is an active research area. For example, features based on wavelets [18, 19, 26, 30, 37–39] and wavelet packets [24, 40, 41] have been proposed. We emphasize that while we use the Gabor energy features in the texture segmentation experiments presented in this paper, our vector-valued diffusion algorithm of Section 1.2 can be used in conjunction with any set of features.

1.5.2 Parameter Settings.

In order to accurately analyze a repetitive structure such as a texture patch, data points are needed which cover at least a few periods. We therefore assume that in our texture segmentation tasks, the areas of the regions cannot be very small, and design the evolution
(1.10) to encourage the formation of large regions. In all the experiments, the initial segmentation takes every pixel to be a separate region. We assume that the correct number of regions is known, and stop the evolution when this number of regions is reached. At the beginning of the evolution, when most regions are small, we use fine-scale Gabor features. We use coarse-scale Gabor features towards the end of the evolution since, as small regions aggregate into larger ones, coarse-scale features become more informative. In addition, the existence of only large regions towards the end of the evolution allows us to build histograms of Gabor filter outputs within each region as suggested in, for example, [21]. These design considerations motivate the following choices for the parameters.

- To encourage relatively rapid evolution of small regions, we use \( a(R_i) = |R_i| + (|R_i|/100)^3 \) where the area \( |R_i| \) of \( R_i \) is the number of nodes in \( R_i \). For small regions, \( a(R_i) \approx |R_i| \); however, for large \( |R_i| \) the cubic term dominates and slows down the evolution of \( \vec{\mu}_i \).

- We evolve Eq. (1.10) in two stages. Each stage uses a different set of features. In stage 1, we use eight-dimensional feature vectors consisting of Gabor features at a fine scale \( \sigma = 1.5 \) and eight orientations. (When selecting the scales and frequencies, we use a procedure similar to [22] in order to obtain a good coverage of the frequency plane.) In stage 2, we use 325-dimensional feature vectors constructed as follows. For every region, we take 25-bin histograms of 12 Gabor features (three scales \( \sigma = 1.5, 3, 6 \), four orientations per scale), and in addition we take a 25-bin histogram of the original grayscale image.\(^2\) We stop stage 1 and start stage 2 as soon as the area of every region is at least 500 pixels.

- As in the previous section, we set \( E(x) = \sqrt{x} \) and let \( b(R_i, R_j) \) be the length of the boundary between \( R_i \) and \( R_j \).

\(^2\)Note that this two-stage procedure is equivalent to running both stages with all 333 features, but setting the weights \( w_k \) to zero during each stage for the features that are not used during that stage.
Fig. 1.6: Segmentation results for two two-texture images with straight boundaries (top two rows) and two two-texture images with random boundaries (bottom two rows).

Table 1.2: Statistics of the errors (in percentages of the total number of pixels) in the two-texture segmentation experiments.

<table>
<thead>
<tr>
<th>Experiment</th>
<th>Min</th>
<th>Max</th>
<th>Avg</th>
<th>Std</th>
</tr>
</thead>
<tbody>
<tr>
<td>Experiment 1 (straight boundary)</td>
<td>0.2%</td>
<td>3.6%</td>
<td>0.6%</td>
<td>0.4%</td>
</tr>
<tr>
<td>Experiment 2 (random boundary)</td>
<td>0.4%</td>
<td>4.4%</td>
<td>1.2%</td>
<td>0.6%</td>
</tr>
</tbody>
</table>

We have tried many different parameter settings and found that our segmentation results are not very sensitive to changes of the parameters. The results of such experimental sensitivity analysis for one of our experiments are reported in Section 1.5.5.
1.5.3 Experiment 1: Two Textures, Straight Boundary.

We form 90 two-texture test images using all pairs of ten Brodatz textures \cite{42}.\footnote{The Brodatz texture images were obtained from www.ux.his.no/~tranden/brodatz.html.} Each test image is obtained by concatenating two $256 \times 256$ texture images. Our nonlinear diffusion is evolved until two regions remain. The results in Table 1.2 and in the top two rows of Fig. 1.6 show that our segmentations are very accurate, with only 0.6\% average error.

1.5.4 Experiment 2: Two Textures, Random Boundary.

In the second experiment, 100 test images are generated using two Brodatz textures, D4 and D84, separated by random boundaries which are 100 independent realizations of a correlated Gaussian random process. Examples of such boundaries are shown in the two bottom panels of Fig. 1.6(c), and the corresponding test images are shown in Fig. 1.6(a). The results are summarized in Table 1.2 and show that, again, our algorithm produces consistently reliable segmentations, with only 1.2\% average error. In this experiment, it is not easy to visually discern the boundary; the boundaries extracted by our algorithm (Fig. 1.6(b)), however, are very close to the actual ones. The mismatches between the extracted boundary and the actual one are shown in Fig. 1.6(d).

1.5.5 Experiment 3: Comparison to [21].

In [21], a segmentation algorithm was developed which optimizes an objective functional obtained from a statistical model of Gabor coefficients. The database used in [21]\footnote{www-dbv.cs.uni-bonn.de/image/mixture.tar.gz} contains 100 $512 \times 512$ five-texture images two of which are shown in the left column of Fig. 1.7(a). The corresponding segmentations produced by our algorithm are shown in Fig. 1.7(b), and the mismatch between the edges found by our algorithm and the actual edges is in Fig. 1.7(d). The median error percentage on this database reported in [21] is
2.65%; our median error percentage is 1.48%. The mean error percentage, estimated from the histogram given in [21], is about 5%; our mean error percentage is 2.23%. The histogram of the error percentage in Fig. 1.8(a) shows that, for most images, our algorithm’s error percentage is below 5%; very few images result in large errors.\footnote{A large error typically results when a significant segment of an edge is completely missed by the algorithm, as in the top row of Fig. 1.9.}

We note that the algorithm of [21] operates on $8 \times 8$ image blocks rather than on pixels. If we convert our segmentation results to $8 \times 8$ blocks by assigning every block to a single region according to the majority of the pixels in the block, our median and mean error percentages go up to 2.21% and 2.79%, respectively.

We in addition use the top image from Fig. 1.7(a) to illustrate the robustness of our algorithm to parameter changes. We let $a(R_i) = |R_i| + (|R_i|/\text{PARAMETER1})^3$, and run the segmentation algorithm with $\text{PARAMETER1} = 50, 75, 100, 125, 150$ and with all the other parameter values as described in Section 1.5.2. The results shown in Table 1.3 demonstrate very similar performance of the algorithm for all five settings of $\text{PARAMETER1}$. We let $\text{PARAMETER2}$ be the number of bins in the Gabor histograms for Stage 2 of the algorithm,
and run the algorithm with \( \text{PARAMETER}_2 = 15, 20, 25, 30, 35 \) and with all the other parameter values as described in Section 1.5.2. As shown in Table 1.4, the results are identical for these five parameter settings. Similarly, starting stage 2 as soon as the area of every region is at least \( \text{PARAMETER}_3 \) pixels, and setting \( \text{PARAMETER}_3 \) to 400, 450, 500, 550, 600, also yields identical results, as shown in Table 1.5.

Finally, we use this experiment to evaluate the running time of our region-merging algorithm. Excluding the feature extraction stage which is common to most typical texture segmentation algorithms, the average running time of the region merging itself for these 100 \( 512 \times 512 \) images is 120 seconds, with standard deviation 25 seconds, on a Pentium 4 3.0 GHz machine. The average number of iterations performed is 884, with standard deviation 48.

### Table 1.3: Dependence of the errors on \( \text{PARAMETER}_1 \) for the top image in Fig. 1.7(a).

<table>
<thead>
<tr>
<th>PARAMETER(_1)</th>
<th>50</th>
<th>75</th>
<th>100</th>
<th>125</th>
<th>150</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of error pixels</td>
<td>2669</td>
<td>2485</td>
<td>2483</td>
<td>2444</td>
<td>2558</td>
</tr>
<tr>
<td>Error percentage</td>
<td>1.02%</td>
<td>0.95%</td>
<td>0.95%</td>
<td>0.93%</td>
<td>0.98%</td>
</tr>
</tbody>
</table>
Table 1.4: Dependence of the errors on PARAMETER2 for the top image in Fig. 1.7(a).

<table>
<thead>
<tr>
<th>PARAMETER2</th>
<th>15</th>
<th>20</th>
<th>25</th>
<th>30</th>
<th>35</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of error pixels</td>
<td>2483</td>
<td>2483</td>
<td>2483</td>
<td>2483</td>
<td>2483</td>
</tr>
<tr>
<td>Error percentage</td>
<td>0.95%</td>
<td>0.95%</td>
<td>0.95%</td>
<td>0.95%</td>
<td>0.95%</td>
</tr>
</tbody>
</table>

Table 1.5: Dependence of the errors on PARAMETER3 for the top image in Fig. 1.7(a).

<table>
<thead>
<tr>
<th>PARAMETER3</th>
<th>400</th>
<th>450</th>
<th>500</th>
<th>550</th>
<th>600</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of error pixels</td>
<td>2483</td>
<td>2483</td>
<td>2483</td>
<td>2483</td>
<td>2483</td>
</tr>
<tr>
<td>Error percentage</td>
<td>0.95%</td>
<td>0.95%</td>
<td>0.95%</td>
<td>0.95%</td>
<td>0.95%</td>
</tr>
</tbody>
</table>

1.5.6 Experiment 4: Comparison to [26].

There are 12 different test images\(^6\) in [26] which are used to evaluate many texture segmentation algorithms. For each algorithm and each test image, the segmentation in [26] is obtained using a supervised classifier trained with features extracted from training samples of every texture present in the test image. Many different algorithms are obtained by combining different feature extraction steps (Gabor filtering, DCT, wavelets, etc) with different classifiers (learning vector quantization, feed-forward neural net, etc). We note here that our segmentation algorithm is unsupervised and is used with the same parameter settings and features for all our texture segmentation experiments. The two algorithms from [26] whose average error percentages are the lowest, are compared in Table 1.6 with our algorithm. For each image, we also provide the mean and the minimum error percentage over all algorithms evaluated in [26].\(^7\) The average error percentage for our algorithm (rightmost column) is much lower than the best average percentage for [26]. In addition, our algorithm performs well on some images where all algorithms in [26] have very high

\(^6\)www.ux.his.no/~tranden/data.html
\(^7\)Note that the corrected data for [26]’s Fig. 11(i) comes from www.ux.his.no/~tranden/comments.html.
Fig. 1.9: Our segmentation results for images from [26]. Top row: Fig. 11(e) from [26]; second row: Fig. 11(f) from [26]; third row: Fig. 11(h) from [26]; bottom row: Fig. 11(i) from [26].

error percentages. For example, our algorithm’s error percentage for the image shown in the second row of Fig. 1.9 (taken from Fig. 11(f) of [26]) is 5.8%, compared to 35% for the best algorithm in [26].

Segmentation results for four of the 12 images are shown in Fig. 1.9.

1.6 Segmentation of Natural Images.

Segmentation of natural images is important since it is often used as the first stage of image analysis algorithms for various tasks such as database organization and retrieval, classification, detection and recognition of objects in images, compression, etc. Fig. 1.10
Table 1.6: The comparison of error percentages for 12 test images.

<table>
<thead>
<tr>
<th>Number of textures</th>
<th>12(a)</th>
<th>12(b)</th>
<th>12(c)</th>
<th>11(a)</th>
<th>11(b)</th>
<th>11(c)</th>
<th>11(d)</th>
<th>11(e)</th>
<th>11(h)</th>
<th>11(i)</th>
<th>11(f)</th>
<th>11(g)</th>
<th>mean</th>
</tr>
</thead>
<tbody>
<tr>
<td>Two</td>
<td>0.5</td>
<td>0.5</td>
<td>0.8</td>
<td>4.1</td>
<td>4.3</td>
<td>11</td>
<td>13</td>
<td>27</td>
<td>6.1</td>
<td>8.4</td>
<td>5.8</td>
<td>14</td>
<td>8.0</td>
</tr>
<tr>
<td>Five</td>
<td>6.5</td>
<td>0.6</td>
<td>7.2</td>
<td>7.2</td>
<td>21</td>
<td>24</td>
<td>19</td>
<td>19</td>
<td>40</td>
<td>30</td>
<td>38</td>
<td>43</td>
<td>21</td>
</tr>
<tr>
<td>Ten</td>
<td>8.1</td>
<td>0.8</td>
<td>8.2</td>
<td>8.7</td>
<td>19</td>
<td>24</td>
<td>18</td>
<td>17</td>
<td>40</td>
<td>29</td>
<td>36</td>
<td>42</td>
<td>21</td>
</tr>
<tr>
<td>Sixteen</td>
<td>0.7</td>
<td>0.2</td>
<td>2.5</td>
<td>7.2</td>
<td>19</td>
<td>21</td>
<td>17</td>
<td>17</td>
<td>33</td>
<td>28</td>
<td>35</td>
<td>42</td>
<td>18</td>
</tr>
<tr>
<td>Mean over all methods in [26]</td>
<td>15</td>
<td>2.8</td>
<td>11</td>
<td>18</td>
<td>33</td>
<td>35</td>
<td>35</td>
<td>31</td>
<td>47</td>
<td>46</td>
<td>51</td>
<td>60</td>
<td>32</td>
</tr>
</tbody>
</table>
illustrates the performance of our algorithm on several images from the Berkeley segmentation dataset [43, 44]. The desired final number of regions is application dependent; in these examples, the final number of regions is simply hand-selected for each image to produce the results which are the most visually pleasing to the authors. As in Section 1.4, the function \( a(R_i) \) is set to be the area of the region \( R_i \); the remaining parameters as well as the feature extraction are the same as for the texture segmentation experiments, as described in Sections 1.5.1 and 1.5.2.

While precise quantitative evaluation of the performance of our algorithm on such images is beyond the scope of this paper (indeed, analyzing the performance of a segmentation algorithm on a natural image is a challenging open problem [44–46]), note that these segmentations are comparable to the ones produced by recent algorithms such as [38, 41, 47–50]. For example, our algorithm captures the outline of the small birds in the center of the top left image while the algorithm in [47] does not; in the church image, our algorithm accurately captures the outlines of the two crosses while the algorithm in [47] does not. Our segmentations of the leopard and bear images are very similar to those in [47]; on the other hand, in the deer image the algorithm in [47] is able to segment the legs of the small deer while our algorithm is not.

1.7 Conclusions, Discussion, and Future Research.

We proposed a novel family of vector-valued nonlinear diffusion equations for images on arbitrary graphs and applied it to the problems of color and texture segmentation. We demonstrated the effectiveness of our method by comparing it to other algorithms on a large number of texture segmentation tasks. We also illustrated our algorithm by segmenting a number of natural images.

Our method provides a systematic way of feature-based segmentation based on minimizing an energy functional. The information about the structure of the desired regions and their boundaries can be built into the formulation. In addition, there is flexibility in
Fig. 1.10.: Segmentation of natural images from the Berkeley segmentation dataset [43, 44].
utilizing various features: different features can be emphasized at different stages of the procedure.

Recent literature on segmentation comes in four broad (and overlapping) categories: graph cut methods \[38, 49, 51, 52\], statistical methods \[19, 37, 40, 53\], PDE-based methods \[41, 48, 54, 55\], and region merging methods \[48–50, 56–60\]. Our framework falls into the last two categories; however, it is different from previously proposed approaches in several important respects. Most recent PDE-based approaches relate image edges to level sets of a (smooth) solution to a PDE. Our system, on the other hand, evolves the image itself, and explicitly forms regions. This allows for a great flexibility in using any information about the sizes, shapes, locations, and boundaries of regions to help guide the evolution. Unlike the traditional region merging methods, however, the fact that our process is the solution to a system of ordinary differential equations ensures certain stability properties, such as robustness to noise and low sensitivity to parameter changes, as shown in \[14, 15\].

Possible future research questions include probabilistic analysis (to extend the work in \[15\]), the development of systematic training methods for parameter selection, the design of fast approximate numerical schemes, and the exploration of different features.

1.8 Acknowledgments.

We would like to thank Profs. Alan Willsky, Hamid Krim, and David Mumford for many helpful discussions during the early stages of this work, and the anonymous reviewers for many useful and insightful comments.
2. CIRCLE-VALUED NONLINEAR DIFFUSIONS WITH APPLICATION TO TEXTURE SEGMENTATION

2.1 Introduction

A large class of effective image restoration and segmentation methods is based on partial differential equations (PDEs) [2–4, 61]. In the simplest scenario, a grayscale intensity image is taken to be the initial data for a diffusion-like PDE, the PDE is solved forward in time, and the resulting solution is the restored or segmented version of the original image. More sophisticated methods designed to work with textured images, use feature extraction before running a PDE-based filter. The result of feature extraction is an image whose every pixel does not necessarily take values in $\mathbb{R}^1$. Most commonly, the feature vector at every pixel is considered to be an element of $\mathbb{R}^K$ where $K$ is the number of features. In addition, color images can be considered as $\mathbb{R}^3$-valued images. This has motivated the development of vector-valued nonlinear diffusion filters. For example, in [62] we have developed a nonlinear-diffusion-based image segmentation method for vector-valued images.

In addition, sometimes it is useful to have features whose values belong to a non-flat surface such as a circle $\mathbb{S}^1$. For example, the directions of textures [63–66] in texture analysis, hue [67, 68] and chromaticity feature [69, 70] in color image processing, and the orientation of an optical flow [64, 69, 71] in motion analysis, are all features taking values on a circle $\mathbb{S}^1$ or a sphere $\mathbb{S}^2$. Consequently, several diffusion filtering methods for such images have been developed. In [64, 66], a feature defined on $\mathbb{S}^1$ is first mapped into $\mathbb{R}^2$, and then the diffusion is performed in $\mathbb{R}^2$. The first approach where the diffusion is directly applied to $\mathbb{S}^1$-valued features is [65]. Subsequently, [69, 71, 72] extended the method of [65] to features defined on more general non-flat surfaces in $\mathbb{R}^n$ and applied it to various image and optical flow restoration tasks. In [68], a color image segmentation scheme is developed which is based on $\mathbb{S}^1 \times \mathbb{R}^1$-valued anisotropic diffusion of the chromatic channel (i.e., hue.
and saturation) and traditional $\mathbb{R}^1$-valued diffusion of the achromatic channel (i.e., value). The diffused image is then segmented using clustering techniques.

Motivated by this, we develop in this paper a novel segmentation algorithm for $S^1$-valued images. This algorithm is based on a novel family of $S^1$-valued nonlinear diffusions closely related to scalar-valued and vector-valued stabilized inverse diffusion equations which we introduced in [14] and [62], respectively. Such an equation can be viewed as the gradient descent procedure for a global energy functional which encourages the formation of flat regions in an image. In [62], such an energy was constructed by defining an appropriate inner product and a norm in the feature space. Since $S^1$ is not a vector space, we cannot define an inner product on $S^1$; however, it is still possible to define a distance between two feature values. Section 2.2 describes how to do this. We then closely follow the paradigm of [62]: we define the energy functional and the corresponding nonlinear diffusion equation, and describe our image segmentation algorithm based on this diffusion equation. In Section 2.3, we describe our feature extraction procedure and illustrate our algorithm through several texture segmentation tasks. We show that it is more effective than our algorithm from [62] for segmenting images with directional texture patterns when the segmentation is based on the orientation features extracted from the images.

Note that the algorithm proposed here and our algorithm for vector-valued images from [62] can potentially be combined, to yield a procedure applicable to images which take values in $S^{K_1} \times \mathbb{R}^{K_2}$—in other words, feature images where $K_1$ features are $S^1$-valued and $K_2$ features are $\mathbb{R}^1$-valued. Investigating this generalization is a topic for future research.

2.2 $S^1$-Valued Nonlinear Diffusion on an Arbitrary Domain

2.2.1 Diffusions on an Arbitrary Domain: Notation

Our basic notation and terminology are the same as in [62]. We define a scalar-valued image on an arbitrary finite set $\mathcal{N}$ of points as any function which assigns a real number to every point in $\mathcal{N}$. In our segmentation tasks, it is important to define adjacency relationships on the points in $\mathcal{N}$, and therefore we assume that $\mathcal{N}$ is the set of nodes of an
undirected graph $G = (\mathcal{N}, \mathcal{L})$ where the set of links $\mathcal{L}$ consists of unordered pairs of distinct nodes. If $\{m, n\}$ is a link, we say that the nodes $m$ and $n$ are neighbors. For example, $G$ could be a finite 2D rectangular grid where each node has four neighbors: east, west, north, and south.

We let $N$ be the total number of nodes and, without loss of generality, denote the nodes by the integers $1, 2, \ldots, N$, i.e., $\mathcal{N} = \{1, 2, \ldots, N\}$. An image $u$ can then be thought of as an $N$-dimensional vector: $u = (u_1, \ldots, u_N) \in \mathbb{R}^N$. We moreover use $u(t)$ to denote a parametric family of images defined on the set $\mathcal{N}$ for all values of a continuous-valued nonnegative scale parameter $t$, and we call the collection of images $\{u(t)\}_{t=0}^\infty$ a scale-space.

To describe the specific scale-space that we use in this paper, we need the following further definitions.

We say that a set of nodes $R \subset \mathcal{N}$ is a region if, for any two nodes $m, n \in R$, there exists a sequence of links connecting $m$ and $n$. Two regions $R_1, R_2$ are called neighbors if there exist two nodes $m \in R_1, n \in R_2$ which are neighbors, i.e., such that $\{m, n\} \in \mathcal{L}$. We let $\text{NBR-PAIRS}$ be the set of all pairs of neighbor regions, and we let $\text{NBRS}(R_i)$ be the set of all regions that are neighbors of a region $R_i$.

For any partition $S = \{R_1, \ldots, R_I\}$ of the set $\mathcal{N}$ of nodes, we let $U_S$ be the set of all piecewise constant images which are constant over each region $R_i \in S$. We use $\mu_i(u)$ to denote the intensity of any such image $u$ within region $R_i$. Note that $U_S$ is a vector space. To impose a metric on this space, we define the following inner product.

$$\langle u, v \rangle \triangleq \sum_{i=1}^I a(R_i)\mu_i(u)\mu_i(v),$$

where $a(R_i)$ is a positive weight function which enables us to weight the contributions of various regions differently.

### 2.2.2 $S^1$-Valued Nonlinear Diffusion

A typical preprocessing step for the analysis of texture images is feature extraction which yields a feature image. The feature image has the feature value (or, in the case of
vector features, the feature vector) at every pixel location. In certain situations, the feature value is best modeled as a quantity taking values on the circle $S^1$. Equivalently, we may consider such a feature to be a periodic quantity with some period $\Theta$, in the sense that, for any $\theta$, the feature values $\theta$ and $\theta + \Theta$ are identified. For example, the orientation of a texture (measured, for instance, as the angle of the texture’s gradient) is $\pi$-periodic since orientation $\theta$ is the same as orientation $\theta + \pi$. We measure the distance between two $\Theta$-periodic quantities $\theta_1$ and $\theta_2$ as the absolute value of their difference modulo $\Theta/2$. To obtain a formula for this distance, we use the following mapping:

$$r(\theta) = \exp \left( \sqrt{-1} \frac{2\pi}{\Theta} \theta \right),$$

which bijectively maps each interval $(r \cdot \Theta - \Theta/2, r \cdot \Theta + \Theta/2]$ of $\mathbb{R}$ to the unit circle in the complex plane. We define the following signed distance $\zeta(\theta_1, \theta_2)$ between $\theta_1$ and $\theta_2$:

$$\zeta(\theta_1, \theta_2) = -\frac{\Theta}{2\pi} \sqrt{-1} \log \left( \frac{r(\theta_1)}{r(\theta_2)} \right),$$

where $\log$ stands for the principal value of the logarithm, i.e., $-\pi < \log(c) \leq \pi$ for any $c \in \mathbb{C}$. The distance between $\theta_1$ and $\theta_2$ is then defined as $|\zeta(\theta_1, \theta_2)|$. Note that this distance coincides with the usual distance $|\theta_1 - \theta_2|$ in $\mathbb{R}$ when $|\theta_1 - \theta_2| \leq \Theta/2$.

In order to segment feature images, we penalize pairwise distances between the feature values of neighbor regions, by defining the following energy functional for images in $U_S$:

$$\mathcal{E}(u) = \sum_{(R_i, R_j) \in \text{NBR-PAIRS}} b(R_i, R_j) E(|\zeta(\mu_i(u), \mu_j(u))|),
\tag{2.2}$$

where $b(R_i, R_j)$ is a positive weight function, and $E(\cdot)$ is monotonically increasing on $[0, +\infty)$, with $E(0) = 0$.

Given an image $u(0)$ with segmentation $S$, we generate a scale-space $u(t)$ by solving the following gradient descent procedure for $t > 0$:

$$\dot{u}(t) = -\nabla \mathcal{E}(u(t)),
\tag{2.3}$$

where $\dot{u}(t)$ is the derivative of $u(t)$ with respect to the scale parameter $t$, and $\nabla$ stands for the gradient in the space $U_S$ equipped with the inner product (2.1). The following proposition shows how to implement this descent equation.
Proposition 3 The gradient descent procedure (2.3) can be equivalently written as follows:

\[
\hat{\mu}_i(u) = \frac{1}{a(R_i)} \sum_{R_j \in \text{NBR}(R_i)} b(R_i, R_j) \frac{\zeta(\mu_j(u), \mu_i(u))}{[\zeta(\mu_j(u), \mu_i(u))]},
\]

\(E'(\zeta(\mu_j(u), \mu_i(u)))\),

for \(i = 1, \ldots, I\),

where \(E'\) is the derivative of \(E\).

Proof. Let \(\chi_i\) denote the indicator function on \(R_i\), for \(i = 1, \ldots, I\), then \(e_1 = \frac{1}{\sqrt{a(R_i)}} \chi_1\), \(\ldots, e_I = \frac{1}{\sqrt{a(R_I)}} \chi_I\) is an orthonormal basis for \(U_S\). Therefore \(u(t)\) can be represented as a linear combination of the basis vectors \(e_1, \ldots, e_I\),

\[u(t) = \sum_{i=1}^{I} c_i(t)e_i,\]

where the \(i\)-th coefficient \(c_i(t)\) is the inner product of the image with the \(i\)-th basis vector \(e_i\):

\[c_i(t) = \langle u(t), e_i \rangle = \sum_{j=1}^{I} a(R_j) \mu_j(u(t)) \mu_j(e_i) = \sqrt{a(R_i)} \mu_i.\]  

(2.5)

where in the last expression we have abbreviated \(\mu_i = \mu_i(u)\). The above equation can be substituted into Eq. (2.2) to write the functional \(E\) in terms of \(\mu_i\)'s. This yields the following gradient descent in the \(c_i\) coordinates:

\[
\dot{c}_i = -\frac{\partial E}{\partial c_i} \\
= -\frac{\partial}{\partial c_i} \sum_{(R_i, R_j) \in \text{NBR-PNR}} b(R_i, R_j) E \left( \zeta \left( \frac{c_i}{\sqrt{a(R_i)}}, \frac{c_j}{\sqrt{a(R_j)}} \right) \right) \\
= -\sum_{R_j \in \text{NBR}(R_i)} b(R_i, R_j) \frac{\zeta(c_i/\sqrt{a(R_i)}, c_j/\sqrt{a(R_j)})}{\sqrt{a(R_i)}} \cdot E' \left( \zeta \left( \frac{c_i/\sqrt{a(R_i)}}, \frac{c_j/\sqrt{a(R_j)}}{\sqrt{a(R_j)}} \right) \right).
\]

Using Eq. (2.5) again to rewrite this in terms of \(\mu_i\)'s, we get:

\[
\sqrt{a(R_i)} \dot{\mu}_i = \sum_{R_j \in \text{NBR}(R_i)} \frac{b(R_i, R_j)}{\sqrt{a(R_i)}} \cdot \frac{\zeta(\mu_j, \mu_i)}{[\zeta(\mu_j, \mu_i)]} E'(|\zeta(\mu_j, \mu_i)|),
\]

1When there is no possibility of confusion, we abbreviate \(u(t)\) as \(u\).
which is the same as Eq. (2.4).

To develop a segmentation algorithm, we follow our procedure from [62] and supplement the dynamics (2.4) with region merging. However, our rules for merging regions are different from [62] and reflect the fact that our features are periodic. Specifically, two neighbor regions $R_i$ and $R_j$ of $u(t)$ are merged at the earliest time instant $t = \tau$ which satisfies $\zeta(\mu_i(u(\tau)), \mu_j(u(\tau))) = 0$, i.e., $\mu_i(u(\tau)) - \mu_j(u(\tau)) = r \cdot \Theta$ for some integer $r$. After the merge, we initialize the intensity value of the new region as $\mu_{new}(u(\tau^+))$, which satisfies:

1. $\mu_{new}(u(\tau^+)) \in (-\Theta/2, \Theta/2]$,
2. $\mu_{new}(u(\tau^+)) - \mu_i(u(\tau)) = r \cdot \Theta$ for some integer $r$.

When $R_i$ and $R_j$ are merged, they are both removed from the segmentation $S$ and their union $R_{new} \equiv R_i \cup R_j$ is added:

$$S(\tau^+) = S(\tau) \setminus \{R_i, R_j\} \cup \{R_{new}\}$$

In addition, we require that $E''(x) < 0$ when $x > 0$ and $\lim_{x \to 0^+} E'(x) = +\infty$. For example, $E(x) = \sqrt{x}$ has the desired shape, see Fig. 2.1. With this choice of the function $E$, Eq. (2.4) is well suited for segmentation since, as shown in [62], it encourages the merging of pairs of neighbor regions.

Extensions of this framework to other manifolds in $\mathbb{R}^n$ are also possible, through defining a distance on the manifold and following a similar procedure to define the energy functional and the corresponding gradient descent equation.
2.3 Texture Segmentation

2.3.1 Feature Extraction

We use the angle of the gradient at each pixel to extract the orientation feature from a texture image. Specifically, given an image \( f(x, y) \), where \( x \) and \( y \) represent the horizontal and vertical indices of pixels respectively, the orientation \( \theta(x, y) \) at the pixel \((x, y)\) is calculated as:

\[
\theta(x, y) = \arctan \left( \frac{f(x, y + 1) - f(x, y)}{f(x + 1, y) - f(x, y)} \right).
\]

Note that we treat any two gradients with antipodal phase angles as the same in the above equation since orientation \( \theta \) is the same as orientation \( \theta + \pi \) for any texture.

2.3.2 Segmentation Examples

We use \( \theta(x, y) \) as the initial data \( u(0) \), and start the evolution of (2.3) with every pixel being a separate region. When the distance \( \zeta \) between the intensities of two neighbor regions becomes zero during the evolution, these two regions get merged into one as described in the previous section, and the gradient descent equation (2.3) proceeds using the new set of regions.

We compare our algorithm with the algorithm of [62], which was shown to produce state-of-the-art texture segmentation results. In our comparative experiments, the algorithm of [62] is also applied to the scalar-valued feature image \( \theta(x, y) \).

For both algorithms, we use the same parameter settings. An energy function \( E = \sqrt{x} \) is used. The function \( a(R_i) \) that we use is described in [62] and is related to the area of region \( R_i \), i.e., to the number of nodes in \( R_i \). Our function \( b(R_i, R_j) \) is the length of the boundary between \( R_i \) and \( R_j \), i.e., the number of links \( \{m, n\} \) such that \( m \in R_i \) and \( n \in R_j \). In all the experiments below, we assume that the correct number of regions is known, and we stop the evolution when this number of regions is reached.

**Experiment 1: Two Textures, Straight Boundary.** We form 30 two-texture test images using six Brodatz textures [42]. Each test image is obtained by concatenating two \( 256 \times 256 \)
<table>
<thead>
<tr>
<th></th>
<th>Min</th>
<th>Max</th>
<th>Mean</th>
<th>Std</th>
</tr>
</thead>
<tbody>
<tr>
<td>Algorithm of [62]</td>
<td>0.56%</td>
<td>37.81%</td>
<td>8.23%</td>
<td>11.13%</td>
</tr>
<tr>
<td>Proposed algorithm</td>
<td>0.36%</td>
<td>23.17%</td>
<td>3.25%</td>
<td>5.48%</td>
</tr>
</tbody>
</table>

**Table 2.1:** Statistics of the misclassified pixels(%) in Exp. 1.

<table>
<thead>
<tr>
<th></th>
<th>Min</th>
<th>Max</th>
<th>Mean</th>
<th>Std</th>
</tr>
</thead>
<tbody>
<tr>
<td>Diffusion in [62]</td>
<td>0.62%</td>
<td>4.61%</td>
<td>1.42%</td>
<td>0.63%</td>
</tr>
<tr>
<td>Orientation Diffusion</td>
<td>0.36%</td>
<td>1.15%</td>
<td>0.59%</td>
<td>0.13%</td>
</tr>
</tbody>
</table>

**Table 2.2:** Statistics of the misclassified pixels(%) in Exp. 2.

Texture images. An example of test image is shown in Fig. 2.2(a) and its corresponding orientation feature is in Fig. 2.2(b). We run both our new algorithm and the algorithm of [62] until two regions remain. The results in Table 2.1 show that our new algorithm produces better segmentations. The average misclassification rate given by the algorithm in [62] is 8.23%; our new algorithm gives a much lower 3.38% average misclassification rate.

**Experiment 2: Two Textures, Random Boundary.** In this experiment, 100 test images are generated using two Brodatz textures, D68 and D77, separated by random boundaries which are 100 independent realizations of a correlated Gaussian random process. An example of such a boundary is shown in Fig. 2.2(l), and the corresponding test image is shown in Fig. 2.2(h). The results are summarized in Table 2.2 and show that, again, when applied to orientation feature our new algorithm produces better segmentations than the algorithm in [62]. In the example of segmenting Fig. 2.2(h), the boundary extracted by our new algorithm (Fig. 2.2(n)), is very close to the actual one, while the algorithm in [62] makes major mistakes in determining the boundary (Fig. 2.2(k)).
2.4 Conclusions

We have introduced a novel $\mathbb{S}^1$-valued nonlinear diffusion equation and used it to develop an image segmentation algorithm. The effectiveness of our algorithm has been illustrated through several texture segmentation experiments where accurate segmentations have been obtained based on a single orientation feature. Our future work will involve extending our diffusion-based segmentation algorithm to more complicated manifolds such as multidimensional spheres and cylinders, in order to be able to utilize multiple $\mathbb{S}^1$-valued and $\mathbb{R}^1$-valued features and achieve more effective segmentation algorithms.
3. APPROXIMATE METHODS FOR CONSTRAINED TOTAL VARIATION MINIMIZATION

3.1 Problem Statement

Suppose the observed data $f$ is a noisy measurement of an unknown image $g$:

$$f = g + w,$$

where $w$ is zero-mean additive white noise with variance $\sigma^2$. It was proposed in [11] to recover an estimate $\hat{g}$ of $g$ by solving the following problem of constrained minimization of the total variation (TV):

$$\hat{g} = \arg \min_{g: \|f - g\| \leq \sigma} TV(g).$$  \hspace{1cm} (3.1)

A number of more effective noise removal paradigms have since been developed [73, 74]. However, problem (3.1) and related variational and PDE-based methods have been successfully used in a variety of other application areas such as tomographic reconstruction [75, 76], deblurring [77–80], and segmentation [14, 62]. This motivates continued interest in problem (3.1) as well as the need to develop fast algorithms for solving it.

The original formulation of [11] treated continuous-space images for which the total variation is defined as follows:

$$TV(g) = \int |\nabla g|.$$

A numerical procedure was developed in [11] based on discretizing the corresponding Euler-Lagrange equations. Since then, many other numerical schemes have been proposed to approximately solve this and related optimization problems [14, 77–84].

We consider a discrete formulation of the problem in which $f$ and $g$ are both images defined on an undirected graph $\mathcal{G} = (\mathcal{N}, \mathcal{L})$ where $\mathcal{N}$ is a set of nodes (pixels) and $\mathcal{L}$ is
the set of links which define the neighborhood structure of the graph. We use the following definition of the total variation for such discrete images:

$$
\text{TV}(g) \triangleq \sum_{\{m,n\} \in \mathcal{L}} |g_m - g_n|.
$$

(3.2)

The norm we use in Eq. (3.1) is the $\ell_2$ norm:

$$
\|f - g\| = \sqrt{\sum_{n \in \mathcal{N}} (f_n - g_n)^2}.
$$

The optimization problem (3.1) can then be cast as a second-order cone program (SOCP), i.e., the minimization of a linear function over a Cartesian product of quadratic cones [85–87]. The globally optimal solution to this SOCP problem can be obtained with interior point methods whose computational complexity is $O(N^2 \log \varepsilon^{-1})$ where $N = |\mathcal{N}|$ is the number of pixels, and $\varepsilon$ is a precision parameter [83, 87].

The main contribution of the present paper is the development of a suboptimal algorithm which we empirically show to be about 15-100 times faster than a state-of-the-art interior point method, for typical natural images. This algorithm is also empirically shown to achieve values of TV($x$) which are quite close to the globally optimal ones achieved by SOCP. Moreover, the images recovered by the new method and via SOCP are visually very similar. We also experimentally evaluate our algorithm’s performance as a denoising method, using the algorithm of [81] as a benchmark. We show that the two algorithms perform comparably. A specialization of this algorithm to 1D discrete signals was proposed and analyzed in [15]. This specialization was shown to exactly solve problem (3.1) in 1D, in $O(N \log N)$ time, and with $O(N)$ memory complexity. Its variants which have the same complexity were shown in [15] to exactly solve two related discrete 1D problems, specifically, the Lagrangian version of problem (3.1) and the minimization of $\|f - g\|$ subject to a constraint on TV($g$).

We note in addition that, if the graph $G$ is a regular rectangular grid, then each pixel $n$ can be represented by its horizontal and vertical coordinates $i$ and $j$. In this case, another possible discretization of the TV is:

$$
\text{TV}'(g) \triangleq \sum_{i,j} \sqrt{(g_{i+1,j} - g_{i,j})^2 + (g_{i,j+1} - g_{i,j})^2}.
$$

(3.3)
With this definition of the discrete TV, the optimization problem (3.1) can also be cast as an SOCP problem, as shown in [83]. It can therefore also be solved using interior point methods for SOCP.

3.2 Notation

We define a real-valued image on an arbitrary finite set $\mathcal{N}$ of points as any function which assigns a real number to every point in $\mathcal{N}$. For the algorithm described in Section 3.3, it is important to define adjacency relationships on the points in $\mathcal{N}$, and therefore we assume that $\mathcal{N}$ is the set of nodes of an undirected graph $\mathcal{G} = (\mathcal{N}, \mathcal{L})$ where the set $\mathcal{L}$ of links consists of unordered pairs of distinct nodes. If $\{m, n\}$ is a link, we say that the nodes $m$ and $n$ are neighbors. For example, $\mathcal{G}$ could be a finite 2D rectangular grid where each node has four neighbors: east, west, north, and south, as in Fig. 3.1. We say that $R \subseteq \mathcal{N}$ is a connected set if, for any two nodes $m, n \in R$, there exists a path between $m$ and $n$ which lies entirely within $R$—i.e., if there exists a sequence of links of the form $\{m, m_1\}$, $\{m_1, m_2\}, \ldots, \{m_{k-1}, m_k\}, \{m_k, n\}$ with $m_1, m_2, \ldots, m_k \in R$.

If $N = |\mathcal{N}|$ is the total number of nodes, then an image $u$ on $\mathcal{N}$ can be thought of as an $N$-dimensional vector: $u \in \mathbb{R}^N$. We say that a set $\mathcal{S} = \{R_1, \ldots, R_I\}$ is a segmentation of an image $u$ if:
• every \( R_i \) is a connected set of nodes;

• the image intensity within every \( R_i \) is constant, \( u_m = u_n \) for all \( m, n \in R_i \) for \( i = 1, \ldots, I \);

• \( R_1, \ldots, R_I \) are pairwise disjoint sets whose union is \( N \).

We then say that \( R_i \) is a region of \( S \). When convenient, we also say in this case that \( R_i \) is a region of \( u \). For example, two segmentations of the image in Fig. 3.1(a) are shown in Figs. 3.1(b,c). We use \( \mu_R \) to denote the common intensity within region \( R \) and \( |R| \) to denote the number of pixels in \( R \).

Given a segmentation \( S \) of an image \( u \), two regions \( R, R' \in S \) are called neighbors if there exist two nodes \( m \in R, n \in R' \) which are neighbors, i.e., such that \( \{m, n\} \in \mathcal{L} \). The multiplicity \( \lambda_{R,R'} \) of two neighbor regions \( R \) and \( R' \) is the length of the boundary between them—i.e., the number of links \( \{m, n\} \) such that \( m \in R \) and \( n \in R' \). We let \( \text{NBR-PRS}_S \) be the set of all pairs of neighbor regions in \( S \), and we let \( \text{NBRS}_S(R) \) be the set of all regions in \( S \) that are neighbors of a region \( R \in S \). For example, in the segmentation \( S \) of Fig. 3.1(b), each region is a neighbor of the other two, and the multiplicities are \( \lambda_{R_1,R_2} = 3, \lambda_{R_1,R_3} = 3, \) and \( \lambda_{R_2,R_3} = 2 \). In the segmentation \( S' \) of Fig. 3.1(c), \( \text{NBRS}_{S'}(R'_1) = \{R'_2, R'_4, R'_5\} \).

### 3.3 Algorithm Description

We define a dynamical system which generates a family of images \( \{u(t)\}_{t=0}^\infty \), parameterized by a time parameter \( t \). We suppose that the initial data for this system is the observed noisy image, \( u(t = 0) = f \). We let \( S(t) \) be the segmentation of \( u(t) \) such that \( \mu_R(t) \neq \mu_{R'}(t) \) for any pair of neighbor regions \( R, R' \in S(t) \). The output of our algorithm is the image \( u(t^*) \) at such time \( t^* \) that \( \|f - u(t^*)\| = \sigma \). The basic reason for the fact that our algorithm is fast is that it does not explicitly compute the solution \( u(t) \) for any \( t \neq t^* \).

The basic reason for the fact that it achieves values of the TV which are close to globally optimal ones, is the fact that the underlying dynamical system is based on the gradient descent for the TV, as we presently explain. The remainder of the section is devoted to the description of the dynamical system and our algorithm for computing \( u(t^*) \).
We first rewrite Eq. (3.2) for any image \( u(t) \), as follows, using the notation introduced in the previous section:

\[
TV(u(t)) = \sum_{\{R,R'\} \in \text{NBR-PRS}_{S(t)}} \lambda_{R,R'} \cdot |\mu_R(t) - \mu_{R'}(t)|.
\]

It is shown in [62] that if the gradient is taken in the space of all images which are piecewise constant on \( S(t) \), then the gradient descent for the TV is given by

\[
\dot{\mu}_R(t) = \frac{1}{|R|} \sum_{R' \in \text{NBR}_{S(t)}(R)} \lambda_{R,R'} \cdot \text{sgn}(\mu_{R'}(t) - \mu_R(t)),
\]

(3.4)

where \( \mu_R(t) \) denotes the intensity within region \( R \) of image \( u(t) \). This equation is valid as long as \( S(t) = \text{const} \), i.e., as long as \( \mu_R(t) \neq \mu_{R'}(t) \) for every pair of neighbor regions \( R \) and \( R' \). As soon as \( \mu_R(t) \) and \( \mu_{R'}(t) \) become equal for some pair of neighbor regions \( R \) and \( R' \), their respective rates of evolution \( \dot{\mu}_R(t) \) and \( \dot{\mu}_{R'}(t) \) become undefined since the right-hand side of Eq. (3.4) undergoes a discontinuity in this case. To handle this scenario, we supplement Eq. (3.4) with the following rule.

**Region merging: how and when.** Suppose that for some time instant \( t = \tau_{R,R'} \) we have: \( R, R' \in S(\tau^-_{R,R'}) \) and \( \mu_R(\tau^-_{R,R'}) = \mu_{R'}(\tau^-_{R,R'}) \). Then we merge \( R \) and \( R' \) into a new region \( R \cup R' \), with the same intensity:

\[
\begin{align*}
S(\tau^+_{R,R'}) &= S(\tau^-_{R,R'}) \setminus \{R, R'\} \cup \{R \cup R'\}, \\
\mu_{R \cup R'}(\tau^+_{R,R'}) &= \mu_R(\tau^-_{R,R'}) = \mu_{R'}(\tau^-_{R,R'}).
\end{align*}
\]

In addition, as shown through numerical experiments in the next section, it may sometimes be beneficial to split a region into two different regions. We postpone until later the discussion of when our algorithm decides to split a region. Once it does, region splitting occurs as follows.

**Region splitting: how.** Splitting of region \( R \) into \( R' \) and \( R \setminus R' \) at some time instant \( t = \tau_R \) means that new regions \( R' \) and \( R \setminus R' \) are formed, and that they have the same intensity as \( R \):

\[
\begin{align*}
S(\tau^+_R) &= S(\tau^-_R) \setminus \{R\} \cup \{R, R \setminus R'\}, \\
\mu_{R'}(\tau^+_R) &= \mu_{R \setminus R'}(\tau^+_R) = \mu_R(\tau^-_R)
\end{align*}
\]
Time instants when regions are merged and split are called event times. Given a time $T$ and the corresponding segmentation $S(T)$, we define the birth time $b_R$ for every region $R \in S(T)$ as $b_R \triangleq \sup\{ t < T : R \notin S(t) \}$. Similarly the death time $d_R$ of $R$ is defined as $d_R \triangleq \inf\{ t > T : R \notin S(t) \}$. Note that $b_R$ can be either zero, or a time instant when two regions get merged to form $R$, or a time instant when $R$ is formed as a result of splitting another region into two; $d_R$ can be either a time instant when $R$ is merged with a neighbor to form another region, or a time instant when $R$ is split into two other regions.

Let

$$\beta_R = \sum_{R' \in \text{NBR}S_{S(T)}(R)} \lambda_{R,R'} \cdot \text{sgn}(\mu_{R'}(b_R^+) - \mu_R(b_R^-)), \quad (3.5)$$

$$v_R = \frac{\beta_R}{|R|}. \quad (3.6)$$

Note that our split and merge rules are such that the intensity of every pixel is a continuous function of $t$. The continuity property means that the righthand side of Eq. (3.4) is constant for $b_R < t < d_R$, and is equal to $v_R$:

$$\dot{\mu}_R(t) = v_R \quad \text{for } b_R < t < d_R. \quad (3.7)$$

For each region $R \in S(t)$, we therefore have:

$$\mu_R(t) = \mu_R(b_R) + (t - b_R) \cdot v_R, \quad \text{for } b_R < t < d_R. \quad (3.8)$$

Let $\bar{u}_R^0 \triangleq \frac{1}{|R|} \sum_{n \in R} f_n$ be the average of the initial data $f$ over the set $R$. For the regions $R$ with $b_R = 0$, we have $\mu_R(b_R) = \bar{u}_R^0$, and therefore the following holds:

$$\mu_R(t) = \bar{u}_R^0 + t \cdot v_R, \quad \text{for } b_R < t < d_R. \quad (3.9)$$

In order for our algorithm to be fast, it is important that Eq. (3.9) hold not only for regions with $b_R = 0$, but also for all other regions, at all times $t$. Proposition 4 below relies on this property. It is straightforward to show that, if this equation holds for every region for all times $t < \tau_{R,R'}$, and if regions $R$ and $R'$ get merged at the time instant $t = \tau_{R,R'}$, then Eq. (3.9) will also hold for the new region $R \cup R'$. We are now finally in a position to state our strategy for splitting regions.
Region splitting: when. We use two criteria to determine whether a region \( R \) is to be split into two regions \( R' \) and \( R \setminus R' \). First, we check whether this split is consistent with the dynamics (3.8)—in other words, we check that \( R \) and \( R \setminus R' \) will not be merged back together immediately after they are split. Second, we determine if there exists a time instant \( \tau_R \) at which a split can be performed in such a way that, on the one hand, the intensity of every pixel is a continuous function of time, and, on the other hand, Eq. (3.9) is satisfied for the new regions. Note that, since there are \( O(2^{|R|}) \) possible two-way splits of \( R \), searching over all possible splits is not computationally feasible. Instead, we only search over a small number of possible splits, namely, only horizontal and vertical splits that result in at least one of \( R', R \setminus R' \) being a rectangle. This can be efficiently accomplished through an algorithm that walks around the boundary of \( R \).

We now describe the termination of the algorithm. It is based on the following proposition, which can be proved using Eq. (3.9).

**Proposition 4** Let \( \alpha(t) = \|u(t) - u(0)\|^2 \). Then, for \( t \in [0, \infty) \), \( \alpha(t) \) is a monotonically increasing function of time, which changes continuously from 0 to \( \|u(0) - \bar{u}(0)\|^2 \) where \( \bar{u}(0) \) is the constant image whose every pixel is equal to the average intensity of the initial data \( u(0) \). It is a differentiable function of time except at the times of merges and splits, and its rate of change is:

\[
\dot{\alpha}(t) = 2t \sum_{R \in S(t)} \frac{\beta^2_R}{|R|}.
\]

The algorithm starts by checking whether \( \|f - \bar{f}\|^2 \leq \sigma^2 \). If this is true, we stop the algorithm and use \( \bar{f} \) as the output. Otherwise, we initialize \( \alpha(0) = 0 \). Given \( \alpha(\tau_l) \) at the current event time \( \tau_l \), we use the following equation, which is derived from Proposition 4, to calculate \( \alpha(\tau_{l+1}) \) at the next event time \( \tau_{l+1} \):

\[
\alpha(\tau_{l+1}) = \alpha(\tau_l) + (\tau_{l+1}^2 - \tau_l^2) \sum_{R \in S(\tau_l)} \frac{\beta^2_R}{|R|}.
\]
The algorithm keeps running until $\alpha(\tau_{l+1}) > \sigma^2$ for a certain $l$. Then our algorithm’s termination time $t^*$ can be calculated as follows:

$$t^* = \sqrt{\tau_l^2 + \frac{\sigma^2 - \alpha(\tau_l)}{\sum_{R \in S(\tau_l)} \frac{\beta_R}{|R|}}}.$$

We then use Eq. (3.9) to calculate $\mu_R(t^*)$ for each $R \in S(t^*)$, and output $u(t^*)$.

Putting everything together, we have the following outline of the algorithm.

1. **Initialize.**
   If $\bar{f}$ satisfies the constraint, output $\bar{f}$ and terminate. Otherwise, initialize $t_0$ to be zero, $u(t_0)$ to be the data $f$, the initial segmentation to consist of singleton regions, and the neighborhood structure to be the standard four-neighbor grid. Initialize the parameters $|R|$, $b_R$, $\beta_R$, $\mu_R(t_0)$, and $\lambda_{R,R'}$ according to the definitions above.

2. **Find potential merge times.**
   Assuming that the intensity $\mu_R(t)$ of every region evolves according to Eq. (3.9), find, for every pair of neighbor regions $R$ and $R'$, the time $\tau_{R,R'}$ which is the earliest time when these two regions have equal intensities.

3. **Construct the event heap.**
   Store all potential merge events on a binary min-heap [88] sorted according to the merge times.

4. **Merge, Split, or Stop.**
   Extract the root event from the heap. Calculate $\alpha(\tau)$ where $\tau$ is the event time. If $\alpha(\tau) \geq \sigma$, go to Step 7.
   if the event is a merge event
       merge;
   if the event is a split event
       split;

5. **Update the Heap.**
   Decide whether the newly formed regions may be split. If so, add the corresponding
Fig. 3.2: (a) Noise-free “peppers” image. (b) Percentage difference between the TV for our split-and-merge algorithm and the optimal TV obtained via SOCP, for many different input PSNR levels (solid line); percentage difference between the TV for our merge only algorithm and the optimal TV (dashdot).

Fig. 3.3: (a) Running time for SOCP; (b) running time for our algorithms.

6. **Iterate.**
   
   Go to Step 4.

7. **Output.**
   
   Calculate $t^*$; calculate and output $u(t^*)$. 

split events to the event heap. Add the merge events for the newly formed regions to the event heap. Remove all the events involving the discarded regions from the heap.
3.4 Comparison to SOCP

Given a noise-free image $g$ with dynamic range $[0, 1]$, we generate noisy images $f$ by adding white Gaussian noise with standard deviation $\sigma$ ranging from 0.01 to 0.30. The correct value of $\sigma$ is used in the simulations both for our algorithm and for the SOCP-based algorithm. We use the MOSEK software as the solver for SOCP. It implements a state-of-the-art interior point method [85]. Running the comparative experiments on 12 different natural images yields very similar results; we only provide the results for one image, “peppers,” shown in Fig. 3.2(a).

SOCP solver converges to the globally optimal solution whereas this is not necessarily the case for our algorithm. Fig. 3.2(b) shows how close our algorithm gets to the globally optimal value for the total variation, for a range of typical input PSNRs. As shown in the figure, our split-and-merge algorithm (solid lines) essentially finds the globally optimal solution at high PSNRs and is within 7% of the globally optimal solution at low PSNRs. Note also that the merge-only version of our algorithm (dashed lines) is about a factor of three farther from the optimal total variation than the split-and-merge algorithm. While the optimization performance of the split-and-merge algorithm is very similar to that of the SOCP, their running times are drastically different. In order to make the comparison of the running times as fair to the SOCP method as possible, we only calculate its running time to get to the total variation achieved by our algorithm, rather than the running time until convergence. As Fig. 3.3 shows, the running time for our split-and-merge algorithm is 15 to 100 times lower than that for the interior point method. Note also that the merge-only version of our algorithm is about twice as fast as the split-and-merge, and that a similar comparison to SOCP shows that the merge-only algorithm is 30 to 150 times faster than the SOCP method.

1The definition we use for PSNR for a noise-free image $g$ and a distorted image $\hat{g}$ is:

$$PSNR = 10 \log \left( \frac{(\max g)^2 \cdot N}{\|g - \hat{g}\|^2} \right)$$

2Note also that presolving and matrix reordering are not counted towards the running time of SOCP since these procedures are reusable for images with the same size [83].
Fig. 3.4: (a) Output PSNR as a function of the input PSNR. (b) Output PSNR difference between our algorithm and SOCP.

The visual appearance of the output images and the corresponding PSNRs is very similar for the split-and-merge algorithm and the SOCP interior point method. The output PSNR is displayed in Fig. 3.4(a) as a function of the input PSNR. The PSNR differences displayed in Fig. 3.4(b) reveal virtually identical performance at high input PSNRs and fairly small differences at low input PSNRs. Moreover, the visual quality of the output images is very similar, as evidenced by Fig. 3.5. The visual quality of the output images for the merge-only algorithm is very similar as well; however, these images have lower PSNRs, especially for low input PSNRs.

3.5 Comparison to a Related Image Denoising Method [81]

The algorithm developed in [81] is an iterative algorithm to solve the optimization problem (3.1) where the total variation is defined by Eq. (3.3). The algorithm converges to the unique solution of (3.1). Since the algorithm in [81] and our split-and-merge algorithm address different optimization problems, we compare their noise removal performance. Specifically, we compare the running times of these two algorithms when they reach the same PSNR.

The setup of our experiments is similar to the previous section. The noise-free images have the dynamic range [0, 1]. The noisy images are generated by adding white Gaussian
Fig. 3.5: Denoising for the peppers image. First row: noisy images. Second row: restored via our split-and-merge algorithm. Third row: restored via our merge-only algorithm. Fourth row: restored via SOCP.
noise with various standard deviations $\sigma$. The correct value of $\sigma$ is assumed to be known in the simulations. We first apply our split-and-merge algorithm to noisy images and calculate the running times and the PSNRs of the denoised images. We then rerun each simulation using the algorithm of [81], until the same PSNR is obtained. The results for two images, for a range of input PSNRs, are shown in Fig. 3.7. For the “peppers” image, the algorithm of [81] is faster than our algorithm, by a factor of 4-33. For the “airplane” image, the two algorithms perform similarly at high PSNRs whereas our algorithm is up to about 6 times faster at low PSNRs. Since our algorithm is a multiscale segmentation process which works with flat regions, it is better adapted to images such as “airplane” which have large homogeneous regions.
Fig. 3.7: Running times for our split-and-merge algorithm (solid lines) and the algorithm of [81] (dashed lines) to reach the same levels of PSNR.
4. A DOCUMENT PAGE CLASSIFICATION ALGORITHM IN COPY PIPELINE

4.1 Introduction

The general workflow of a digital copier is to take scanned images from its scanner, process these images, and send them to its printer for the physical reproduction. A copier must be able to process many different kinds of originals. These originals may have different types of content, such as text, line art, graphics, and natural photos; they may be printed on different kinds of media, for example, on papers of various levels of quality and brightness; they may be created using different rendering techniques such as halftone or continuous tone. These different kinds of originals may interact in many different ways with various limitations of copy pipeline such as streaks, stray light, color fringing, drift, gamut, moiré, etc. Fixed settings of the copy pipeline would therefore produce varying levels of reproduction quality, depending on the type of the original. To resolve this issue and generate user-preferred reproductions, different configurations of the copy pipeline are required. We call these different configurations “copy modes”.

As illustrated in Fig. 4.1, a much better reproduction is obtained when the original matches its optimal configuration. Consider a photograph and a fax. The optimal configuration for a photo (i.e., the photo mode) has smoothing and a wide tone curve range to achieve noise reduction and color accuracy. The optimal configuration for a fax (i.e., the text mode) has sharpening and a nearly bilevel curve to achieve an enhanced reproduction.

Fig. 4.1.: Processing a fax in photo and text modes. Left to right: original, photo mode, text mode.
The left panel of Fig. 4.1 is the original, and the other two panels are the results of processing in the photo and text modes, respectively. Users prefer the cleaner background, increased contrast, and sharpness of the text mode and do not like the over-smoothed, low-density text and the dirty background in the photo mode. Therefore it is essential to match the originals with the most appropriate copy modes.

Three methods exist to match the originals to the copy modes. The “institutional copier” method prompts the user to describe attributes of the original to determine a matching mode. It requires a trained user. The “defaults to a single mode” method yields poor quality for originals that do not match that single mode. The “sub-menus” method provides users the opportunities of selecting modes through optional sub-menus. Users mostly ignore or are unaware of such options. None of the above three methods satisfy user interaction and copy quality expectations.

In this paper, we propose a novel document page classification algorithm which can be used as a copy mode selector in the copy pipeline. Many existing algorithms segment the page and label each segmented region with an appropriate class label [89–93]. These algorithms require simultaneous access to the entire page image, and visit each pixel multiple times. On the other hand, any algorithm applicable to the copy pipeline must process image data one strip at a time, and never revisit previously processed strips. This makes it impossible to apply the existing approaches [89–93]. Our proposed algorithm operates on strip-based data and makes one pass over each strip. The computational complexity and memory requirements of our algorithm are both very low. These advantages make our algorithm an ideal candidate to be implemented in the existing copy pipeline without altering the hardware.

In the rest of the paper, we describe and illustrate our algorithm: Section 4.2 provides its general overview; Section 4.3 describes its specifics; Section 4.4 contains experimental results.
4.2 Algorithm Overview

Figure 4.2 illustrates a possible implementation of our classification algorithm. The classifier resides between the front end and the back end of the pipeline, and directs the image data to the most appropriate processing mode. The revised copy pipeline not only improves copy quality and simplifies user interaction, but also significantly increases copy rate in at least one case, as follows. The pipeline reconfigures for each page in a multi-page copy job from an automatic document feeder (ADF). When it detects a black-and-white page inside the job, it implements 3-to-1 channel reduction and enters the faster mono pipeline mode.

We work with a specific copy pipeline equipped with eight different copy modes which are all possible combinations of mono/color and text/mix/picture/photo. Mono mode is a configuration specially optimized for mono originals while color mode is optimized for color originals. Text mode is optimized for text, line arts, graphics, handwritten text, and faxes; picture mode is for high dynamic range halftoned originals such as glossy magazine pages; photo mode is for continuous tone natural scenes on photographic paper; mix mode is for all other types of originals. Our goal is to classify the scanned image of the original into the following eight distinct classes: color-text, color-mix, color-picture, color-photo, mono-text, mono-mix, mono-picture, and mono-photo. For simplicity, we just use the names of the copy modes as the names of candidate classes. Color-mix is the default class when the classification is hard to make.

We follow a tree-like decision structure illustrated in Figure 4.3. A mono vs. color classifier is placed at the root level of the decision tree. Then text vs. nontext classifiers,
text/mix vs. picture/photo classifiers, and picture vs. photo classifiers are deployed at the second, third, and fourth level of the decision tree. Note that starting from the second level of the decision tree, the two classifiers on the same level share the same methodology and the only difference between them is some tunable parameters. In the following section, we discuss each of these classifiers.

4.3 Algorithm Details

4.3.1 Color vs. Mono Classifier

We assume that all image data are from 300 dot-per-inch (DPI) scans, gamma-corrected, and represented in RGB color space with 8 bits per channel.

The essence of our color vs. mono classification algorithm is detecting image patches which are close to gray and declaring an image to be mono only if every patch is close to gray. Basing the decision on patches that are too small would not be robust to noise; on the other hand, small color regions may be missed if the patches are too large. We have experimentally found that partitioning an image into $32 \times 32$ blocks works well.

The specifics of our algorithm are as follows. If $R(p) = G(p) = B(p)$ for a pixel $p$, we say the pixel is gray, where $R(p)$, $G(p)$, and $B(p)$ are the pixel's red, green, and
blue intensities, respectively. We define the *colorfulness* $C(p)$ of a pixel $p$ as the following quantity:

$$C(p) = \max[R(p), G(p), B(p)] - \min[R(p), G(p), B(p)].$$

Note that if $C(p) = 0$ the pixel is gray. Note also that the quantity $C(p)$ may be interpreted as the “Manhattan distance” from the point $(R(p), G(p), B(p))$ to the line $R = G = B$ in the RGB space.

The colorfulness $C(b)$ of a block of pixels $b$ is defined as the sum of $C(p)$ over all the pixels $p$ that belong to the block $b$. We partition the image into $32 \times 32$ blocks and define the colorfulness of the image, $C_{\text{image}}$, as the maximum of $C(b)$ over all $32 \times 32$ blocks $b$. We classify an image as color if $C_{\text{image}}$ is larger than a threshold, and we classify it as mono otherwise. The threshold is determined from training data.

### 4.3.2 Text vs. Nontext Classifier

Here nontext refers to all classes other than text, i.e., mix, picture, and photo. We use two properties of text documents in order to distinguish them from documents that contain pictures or photos in addition to (or instead of) text. First, the histogram for a typical text region has peaks that are more narrow and tall than the peaks in a typical picture or photo histogram. For example, the histogram for a patch of sharp mono text would have two large and narrow peaks, one near white and one near black. Second, nontext areas of a text document typically contain only a few colors. (If, in addition to text, a document contains many colors, we would want to classify it as mix.)

We extract *histogram peakiness* features via the following procedure. We partition the image into $8 \times 64$ blocks and calculate 64-bin $R$, $G$, and $B$ histograms for each block. The $k$-span of a histogram is defined as the largest number of consecutive bins in the histogram whose values are greater than or equal to $k$. The $k$-span of an image is defined as the largest $k$-span over all 64-bin $R$, $G$, and $B$ histograms of its $8 \times 64$ blocks. For each image, we form a ten-dimensional feature vector consisting of $k$-spans of the image for $k = 3, 6, \ldots, 30$. 
This feature extraction procedure is used to calculate the *histogram flatness score* for any image as follows. We estimate the means of text and nontext feature vectors as well as a common covariance matrix, based on labeled training data. For any image whose feature vector is $\mathbf{x}$, we then define the histogram flatness score

$$ F = (\mathbf{m}_{\text{nontext}} - \mathbf{m}_{\text{text}})^T \Sigma_F \mathbf{x}, $$

where $\mathbf{m}_{\text{nontext}}$ and $\mathbf{m}_{\text{text}}$ stand for the mean of the nontext and text feature vectors, respectively, $\Sigma_F$ stands for the common inverse covariance matrix of the text and nontext feature vectors, and $T$ denotes the transpose of a vector. A small histogram flatness score implies peakiness of the histogram and suggests that the image is text, whereas a large histogram flatness score indicates that the image is nontext.

In addition to the histogram flatness score, we compute the *color variability score* which is small if there are only a few colors in the nontext regions of the image, and large if there are many colors. In order to identify nontext regions we define a *text edge* to be three consecutive pixels in either horizontal or vertical direction such that their values are monotonically increasing or decreasing and the absolute value of the difference between the first and the third exceeds a threshold. We take the threshold to be $100$. A nontext region is any region that does not contain any text edges.

The computation of the color variability score starts with partitioning the image into $8 \times 8$ blocks, detecting nontext blocks, and computing the mean $R$, $G$, and $B$ values for every nontext block. We then construct 256-bin $R$, $G$, and $B$ histograms of these mean values taken from all nontext blocks and count the largest number of nonzero bins among these three histograms. This number is our color variability score. If it is low, every histogram has only a few nonzero entries, suggesting that non-text regions in the image are cartoon-like. If this score is high, there is at least one histogram with many nonzero bins, indicating the presence of many colors in the image. In this case, copying in the text mode would be inappropriate.

Finally, in order to classify an image, we set thresholds for both the histogram flatness score and the color variability score, based on the training data. We classify an image as
text if both scores are less than their corresponding thresholds. Otherwise, we classify it as non-text, i.e., mix, picture, or photo.

4.3.3 Text/Mix vs. Picture/Photo Classifier

Photos and pictures contain natural scenes and no text. Our strategy for distinguishing such images from mix and text documents is to detect text and other regions that do not look like natural scenes.

We partition the image into $64 \times 64$ blocks and count the number of text edges in each block. We use the same definition for a text edge as in the previous section. The text edge count for an image is then defined as the maximum text edge count among all the $64 \times 64$ blocks. A large text edge count suggests that the image may be in the mix or text category.

In addition, we reuse histograms of the $R$, $G$, and $B$ averages of $8 \times 8$ blocks which, as described in the previous section, have been computed for the text vs. non-text classification task. As in the previous section, we extract the number of nonzero bins for each of the three histograms. In addition, we extract the $k$-spans for each of the three histograms, for three values of $k$: $k = M/8, M/4, M/2$, where $M$ is the maximum of the histogram over its first 230 bins. These twelve features (the number of nonzero bins and the three $k$-spans for each of the three histograms) form a feature vector. We estimate the means of the feature vectors for the two classes from the training data, and we also estimate a common covariance matrix. For any image whose feature vector is $y$, we then define the unnaturalness score:

$$U = (m_{\text{mix/text}} - m_{\text{photo/pic}})^T \Sigma_U y,$$

where $m_{\text{photo/pic}}$ and $m_{\text{mix/text}}$ are the mean vectors for the two classes, and $\Sigma_U$ is the common inverse covariance matrix. The larger the score $U$, the more likely it is that the image is not a photo or a picture.

Finally, in order to classify an image, we set thresholds for both the edge count score and the unnaturalness score, based on the training data. We classify the image as photo or picture if both scores are less than their corresponding thresholds. Otherwise, we classify it as mix or text.
Table 4.1: Classification rates for the training suite. Harmful misclassifications are indicated in boldface.

<table>
<thead>
<tr>
<th>ground-truth</th>
<th>color-text</th>
<th>color-mix</th>
<th>color-picture</th>
<th>color-photo</th>
<th>mono-text</th>
<th>mono-mix</th>
<th>mono-picture</th>
<th>mono-photo</th>
</tr>
</thead>
<tbody>
<tr>
<td>color-text</td>
<td>60%</td>
<td>40%</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>color-mix</td>
<td>-</td>
<td>99%</td>
<td>1%</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>color-picture</td>
<td>-</td>
<td>66%</td>
<td>34%</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>color-photo</td>
<td>-</td>
<td>29%</td>
<td>-</td>
<td>71%</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>mono-text</td>
<td>18%</td>
<td>4%</td>
<td>-</td>
<td>61%</td>
<td>17%</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>mono-mix</td>
<td>1%</td>
<td>11%</td>
<td>-</td>
<td>2%</td>
<td>83%</td>
<td>3%</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>mono-picture</td>
<td>1%</td>
<td>4%</td>
<td>1%</td>
<td>-</td>
<td>66%</td>
<td>29%</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>mono-photo</td>
<td>-</td>
<td>1%</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>41%</td>
<td>-</td>
<td>58%</td>
</tr>
</tbody>
</table>
4.3.4 Picture vs. Photo Classifier

Pictures typically contain halftone noise. Smooth regions that are near midtone are most affected by the halftone noise. We use these regions to distinguish between a picture and a photo.

We partition an image into $8 \times 8$ blocks and define a block $b$ to be a midtone block if at least one of its mean $R$, $G$, and $B$ values is within some interval centered at 128. In our experiments, we take this interval to be $[80, 176]$. The smoothness $S(b)$ of a block $b$ is then defined as follows. For a block that is not a midtone block, $S(b)$ is infinite. For a midtone block, $S(b)$ is the minimum of $S_R(b)$, $S_G(b)$, and $S_B(b)$ where $S_R(b)$ is the sum of the absolute values of the first differences in the horizontal and vertical directions within the block $b$ for the red channel, and $S_G(b)$ and $S_B(b)$ are defined analogously for the $G$ and $B$ channels, respectively. We finally define the smoothness of the image as the minimum $S(b)$ over all the blocks $b$.

We classify an image as photo if its smoothness is below a threshold, and we classify it as picture otherwise. The threshold is determined from training data.
4.4 Experiment Results

We have a training suite containing 891 images with at least 100 images for each class. The ground-truth labels of all images in the training suite are selected by hand. Since not all errors are equally costly, some are considered benign and others harmful. We accordingly select the decision thresholds discussed in Section 4.3. For example, mislabeling a mono image as color is acceptable whereas mislabeling a color image as mono is not. By conservatively setting the decision thresholds, we are able to correctly identify all color images in our dataset as color images.

Another example is illustrated in Figure 4.4. This is a 2-D feature decision plot of color-text vs. color-nontext classification. The $x$-axis is the histogram flatness score and the $y$-axis is the color variability score. All color nontext images are shown in “X”s and color text images are shown in “O”s. Recall that we classify a color image as color-text if both its histogram flatness score and its color variability score are less than the corresponding thresholds. Color-text misclassified as color-nontext is considered benign and the opposite way of misclassification is considered harmful. Therefore we choose two conservative thresholds: 8 for the histogram flatness scores and 200 for the color variability scores. The thresholds are shown by the dashed lines in Figure 4.4. Note that there are no “X”s in the lower-left rectangle defined by the dashed lines—i.e., there are no harmful misclassifications of nontext into text.

Table 4.1 summarizes the final classification results for all eight modes. It contains the empirical conditional probabilities $P(\text{classification result} \mid \text{ground truth})$ for the training suite. Depending on the ground truth, accuracy ranges from 29% to 99%. These numbers are shown in the main diagonal of Table 4.1. We consider the accuracy to be satisfactory because only very few images falls into harmful modes. These harmful misclassifications are indicated in boldface numbers in Table 4.1. All other misclassification are considered benign.
4.5 Conclusions

We have introduced a real-time, strip-based, low-complexity document page classification algorithm and used it as the mode selector in the copy pipeline. The revised pipeline improves copy quality, simplifies user interactions, and increases the copy rate. The classification algorithm analyzes the scan image and classifies it into one of eight classes in the copy pipeline. Modes are the combinations of mono/color and text/mix/picture/photo. Mode classification is 29% to 99% accurate with misclassifications tending towards benign modes.
LIST OF REFERENCES
LIST OF REFERENCES


