A comparative study of Unsupervised Texture Segmentation Algorithms

Final Report

Team 4
Rohit Girdhar (201001047)
Nishit Soni (201002026)
Divyansh Agarwal (201001027)

November 27, 2012

International Institute of Information Technology, Hyderabad
Digital Image Processing (CSE478)
Monsoon 2012
Course Project
Supervised by Dr. Jayanthi Sivaswamy and Dr. Anubha Gupta

Contents

Problem Statement

1.1 Problem

To compare two classical unsupervised texture segmentation algorithms based on objective and subjective parameters.

The algorithms we chose are based on Local binary patterns, and an adaptation of a general graph based segmentation algorithm. We also worked on a Gabor filter based segmentation approach.

1.2 Importance

Texture segmentation has various applications in PR, vision like CBIR, recognition etc

1.3 Algorithms References

Algorithm 1
A. K. Jain, F. Farrokhnia, "Unsupervised texture segmentation using Gabor filters, Pattern Recognition, vol. 24, no. 12, pp.1167-1186, 1991
Algorithm 2
Timo Ojala, Matti Pietikainen, "Unsupervised Texture Segmentation Using Feature Distributions, ICIAP '97 Proceedings of the 9th International Conference on Image Analysis and Processing-Volume I , pp 311-318
Algorithm 3
Pedro F. Felzenszwalb and Daniel P. Huttenlocher. Efficient graph-based image segmentation. International Journal of Computer Vision, 59:2004, 2004.

1.4 Current Work/Implementations

We completed the implementation of LBP based texture segmentation algorithm, proposed in [6]. We also propose use of a certain data structure for efficient implementation of parts of the algorithm.

The detailed algorithm used, improvements done and sample results for the same are given in the report. (Chapter 2)

We worked on the algorithm mentioned in [3] (using Gabor Filters), however, found lot of limitations in the algorithm, as well as difficulties in implementation. We present a report on our observations. (Chapter 3)

We have also worked on a generic Graph based algorithm for Segmentation ([1]), and applied it to the task of texture segmentation. (Chapter 4)

We did an in depth objective and subjective comparison of the various approaches, as per benchmarking system detailed in [2]. The results and our observations are attached.

We also explore the applications of these algorithms to the real life problems in segmentation.

Unsupervised Texture Segmentation using Feature Distributions (LBP)

2.1 Algorithm

A region-based algorithm is developed for coarse image segmentation and a pixelwise classification scheme for improving the localization of region boundaries.

2.1.1 A region based algorithm for coarse image segmentation

The texture contents of an image region are characterized by the joint distribution of local binary pattern (LBP) and contrast (C) features.

Local Binary Pattern (LBP): This describes the spatial structure of the local texture. LBP is calculated for the entire image. The original 3x3 neighbourhood of every pixels of the image is threshold by the value of the center pixel. The values of the pixels in the thresholded neighbourhood are multiplied by the binomial weights given to the corresponding pixels and obtained values are summed for the LBP number of this texture unit. Thus, we get an image of same size as original image containing the weights corresponding to each pixel.

Contrast Measure(C): LBP does not address the contrast of the texture. Therefore, we require Contrast Measure(C) which calculates the difference between the average gray-level of those pixels which have value 1 and those which have value 0. The LBP/C distribution is approximated by a discrete two-dimensional histogram of size 256xb, where b is the number of bins for C. For implementation purposes, we have used 8 bins.

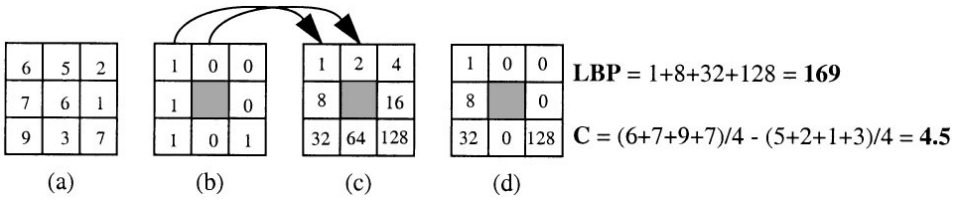


Figure 2.1: Computing LBP/C

G Statistics: The value of the G statistic indicates the probability that the two sample distributions come from the same population: the higher the value, the lower the probability that the two samples are from the same population. We measured the similarity of two histograms with a two-way test of interaction or heterogeneity.

$$\begin{aligned} G =& 2\left(\left[\sum_{s,m}\sum_{i=1}^{n} f_i log f_i\right] - \left[\sum_{s,m}\left(\sum_{i=1}^{n} f_i\right) log\left(\sum_{i=1}^{n} f_i\right)\right]\right. \\ & - \left[\sum_{i=1}^{n}\left(\sum_{s,m} f_i\right) log\left(\sum_{s,m} f_i\right)\right] \\ & \left. + \left[\left(\sum_{s,m}\sum_{i=1}^{n} f_i\right) log\left(\sum_{s,m}\sum_{i=1}^{n} f_i\right)\right]\right) \end{aligned} \tag{2.1}$$

The Segmentation Algorithm essentially consists of two parts: Hierarchical Splitting and Agglomerative merging. Hierarchical splitting is used to divide the image into regions of roughly uniform texture. This is required for the agglomerative merging to be successful. Agglomerative merging procedure merges similar adjacent regions until a stopping criterion is met.

Hierarchical splitting: It recursively splits the original image into square blocks of varying size. Each sub-block has an associated LBP/C histogram, which is required to determine splitting of the block into four sub-blocks based on a uniformity test. We measure the six pairwise G distances between the LBP/C histograms of the four sub-blocks. If we denote the largest of the six G values by G_{max} and smallest by G_{min}, the block is found to be non-uniform and is thus split further into four sub-blocks if a measure of relative dissimilarity within region is greater than a threshold

$$R = \frac{G_{max}}{G_{min}} > X$$

Initially, we divide the original image into rectangular blocks of size S_{max}. It is important not to divide the image in arbitrarily large segments so as to

avoid failure of uniformity test in detecting small texture segment regions. Therefore, starting from S_{max} splitting of blocks is done based on uniformity test until a predetermined S_{min} block size is reached. For, the purpose of implementation we have chosen the value of S_{max} as 64. It is observed that the splitting goes deepest around the texture boundaries.

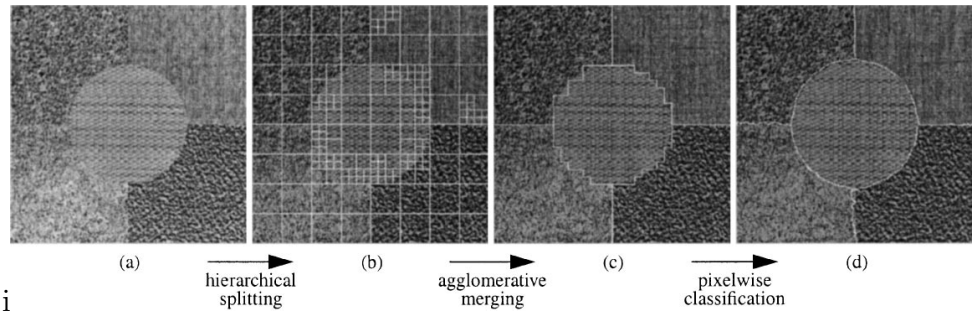
Agglomerative merging: It merges similar adjacent regions until a stopping criterion is satisfied. At a particular stage of the merging, we merge that pair of adjacent segments, which has the smallest merger importance (MI) value.

$$MI = p * G$$

Here, p is number of pixels in smaller of the two regions and G is the distance measure. At each step, the procedure chooses that merger of all possible mergers, which introduces the smallest change in the segmented image. Once the pair of adjacent segments with the smallest MI value has been found, the regions are merged and the two respective LBP/C histograms are summed to be the histogram of the new image region. Merging is allowed to proceed until the stopping rule is encountered

$$MIR = \frac{MI_{cur}}{MI_{max}} > Y$$

The value of threshold Y was determined experimentally. Experimentally, we have observed Y=2.0 gives appropriate results.



i

Figure 2.2: Steps of Segmentation

2.1.2 A pixel based algorithm for refined image segmentation

The hierarchical splitting and agglomerative merging phases results in quite reliable estimates of the different textured regions present in the image. To

further improve the localization of the boundaries a simple pixel wise classification algorithm is used. Treating the LBP/C histograms of the image segments as our texture models we can switch into a texture classification mode. If an image pixel is on the boundary of at least two distinct textures (i.e. the pixel is 4-connected to at least one pixel with a different label), we place a discrete disc with radius r on the pixel and compute the LBP/C histogram over the disc. We compute the G distances between the histogram of the disc and the models of those regions, which are 4-connected to the pixel in question. We relabel the pixel, if the label of the nearest model is different from the current label of the pixel and there is at least one 4-connected adjacent pixel with the tentative new label. The latter condition improves smooth adaptation of texture boundaries and decreases the probability of small holes occurring inside the regions.

2.2 Implementation Details and Improvements

2.2.1 Hierarchical Splitting

We implemented the hierarchical splitting step that divides the image into smaller blocks at regions of possible texture boundaries. Compared to our earlier approach of considering constant size blocks, we now are able to achieve fine grained segmentation of texture.

We observed that the optimal value of X (stopping condition, threshold) for our case is 1.4. Otherwise we followed the algorithm as mentioned in the paper.

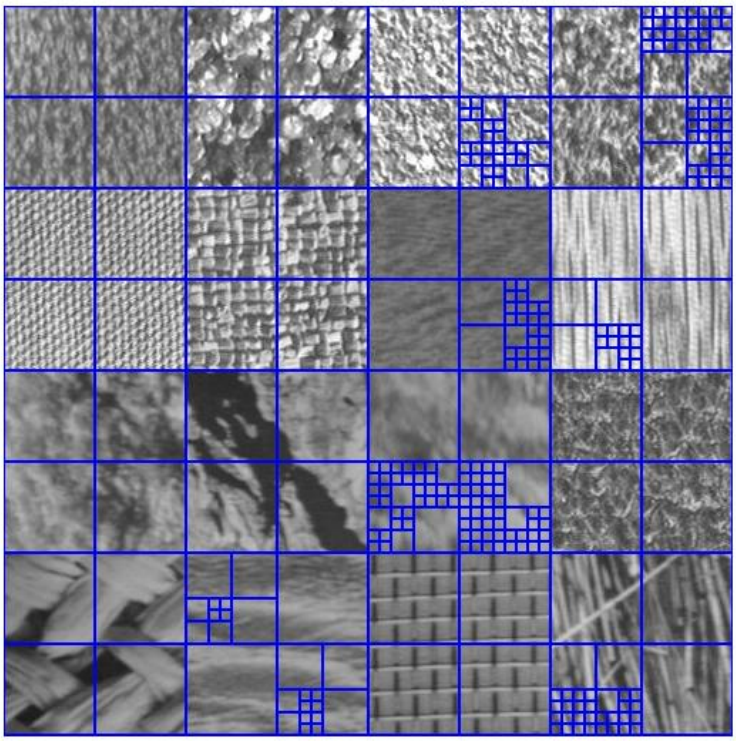


Figure 2.3: Actual heirarchical splitting achieved by our system

2.2.2 Agglomerative Merging

We propose the use of disjoint set data structure for representation of the image for a fast implementation of this step. We observed huge improvement in the speed of the segmenter over the one implemented by us in Interim 1 after using this improved algorithm.

Algorithm

1. Start with each block with itself as different regions.
2. Construct a list of neighbours of each regions intially. (nbrs)
3. Now until merge stopping criterion is met,
   (a) Find the nbrs, i and j with minimum MI (merger index).
   (b) Union the 2 regions (in disjoint-set data structure form)
   (c) Merge the nbrs list of i and j

2.2.3 Pixelwise classification

We essentially followed the algorithm as proposed in [6] and given above. For relabelling the pixels, the paper proposed a circular disk of radius r ($r = 2 * S_{min}$), however, for simplicity and computational efficiency, we use a square of side 4*r+1. We could achieve good result even after the above simplification.

2.3 Limitations

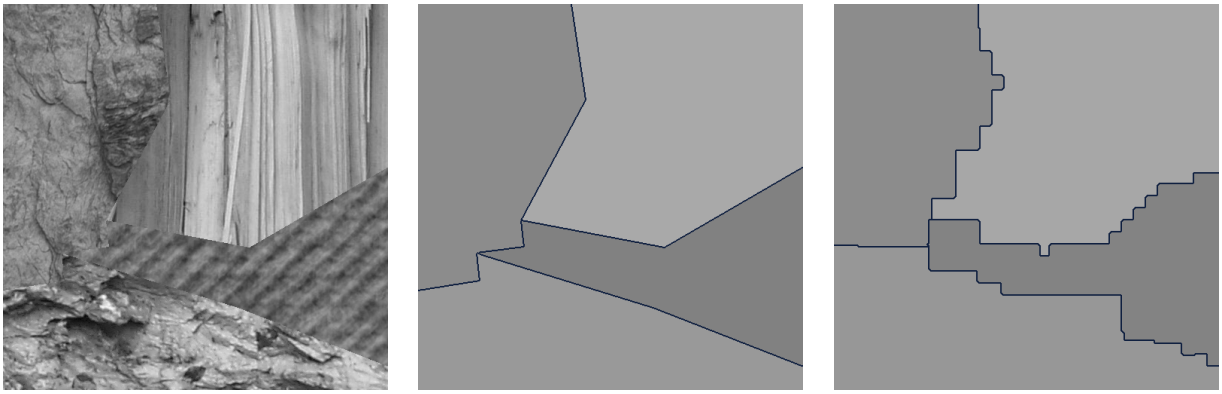
Following are some of the limitations of our system:

- Even after a asymptotically fast implementation, using suitable data structures, the segmenter takes about 5 minutes on a 256x256 image for an acceptable level of segmentation. One reason for the same is that the implementation was done in MATLAB, that usually is slower compare to other programming languages.
- At times, when changing from one dataset to other, we need to tune the parameters again (X and Y, as mentioned above), so as to achieve an optimal performance.

2.4 Sample Results

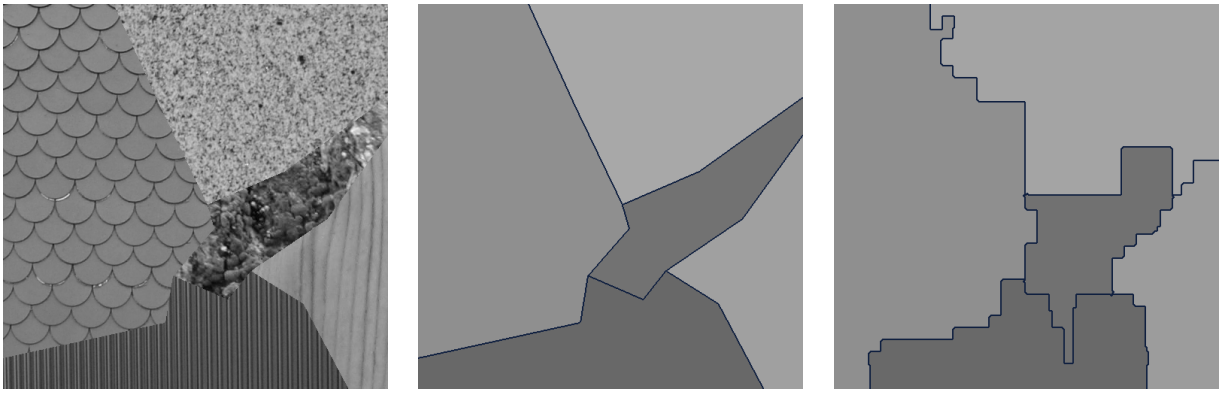
These are some results using our current unsupervised segmentation system. Same color regions depict one texture region (segmented).

All the results were computed at same value of parameters (X = 1.4, Y = 1.6)



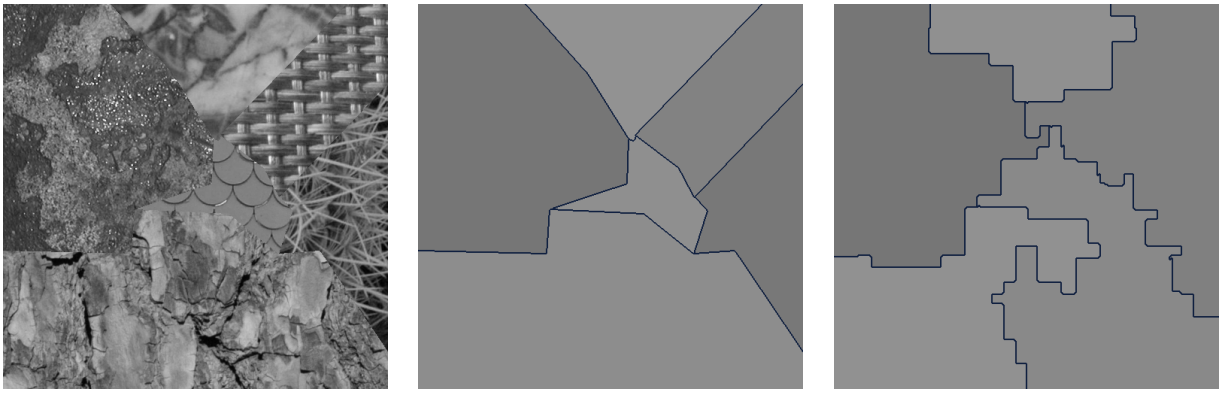
(a) Original Image (b) Ground Truth (c) Segmented Image

Figure 2.4: Grayscale Benchmark image



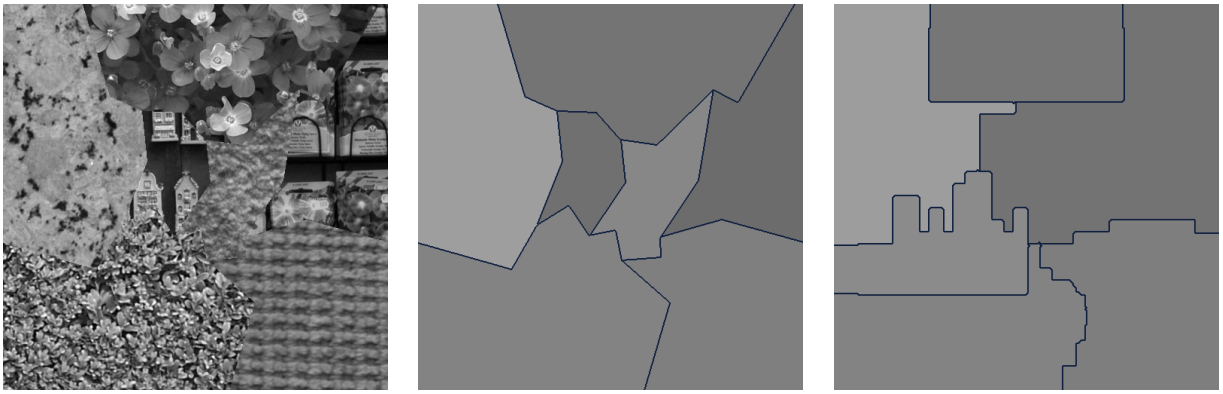
(a) Original Image (b) Ground Truth (c) Segmented Image

Figure 2.5: Grayscale Benchmark image



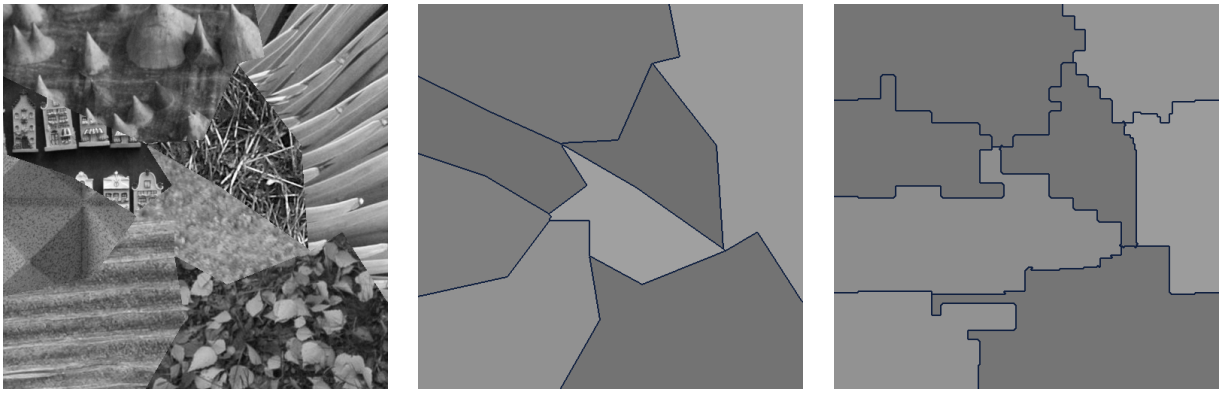
(a) Original Image (b) Ground Truth (c) Segmented Image

Figure 2.6: Grayscale Benchmark image



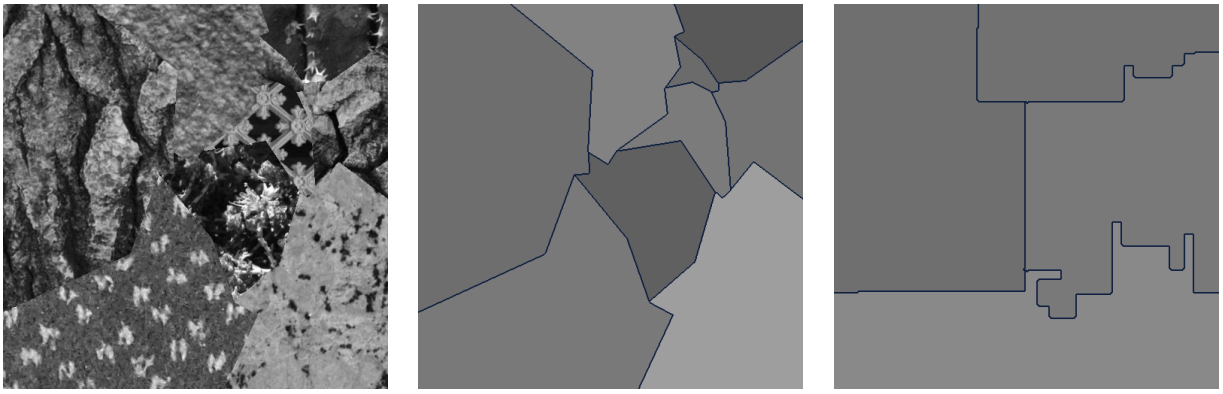
(a) Original Image (b) Ground Truth (c) Segmented Image

Figure 2.7: Grayscale Benchmark image



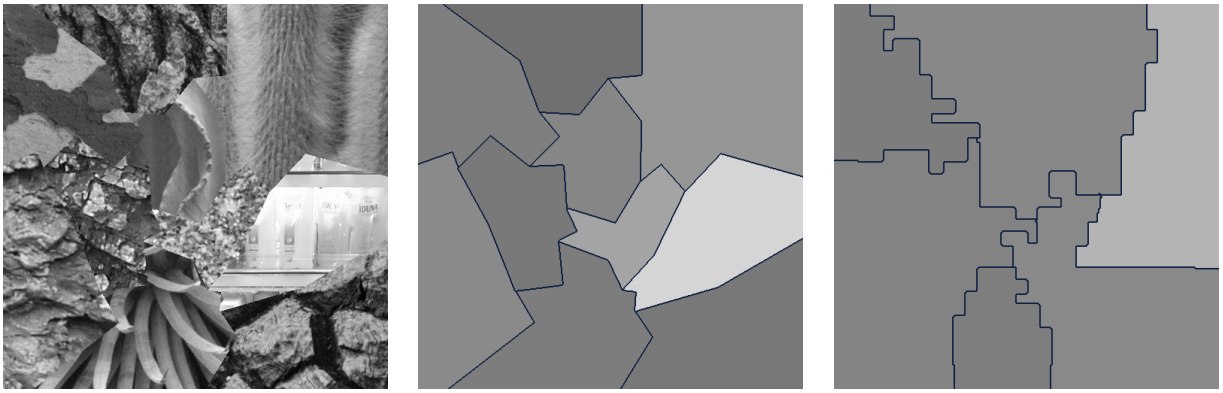
(a) Original Image (b) Ground Truth (c) Segmented Image

Figure 2.8: Grayscale Benchmark image



(a) Original Image (b) Ground Truth (c) Segmented Image

Figure 2.9: Grayscale Benchmark image

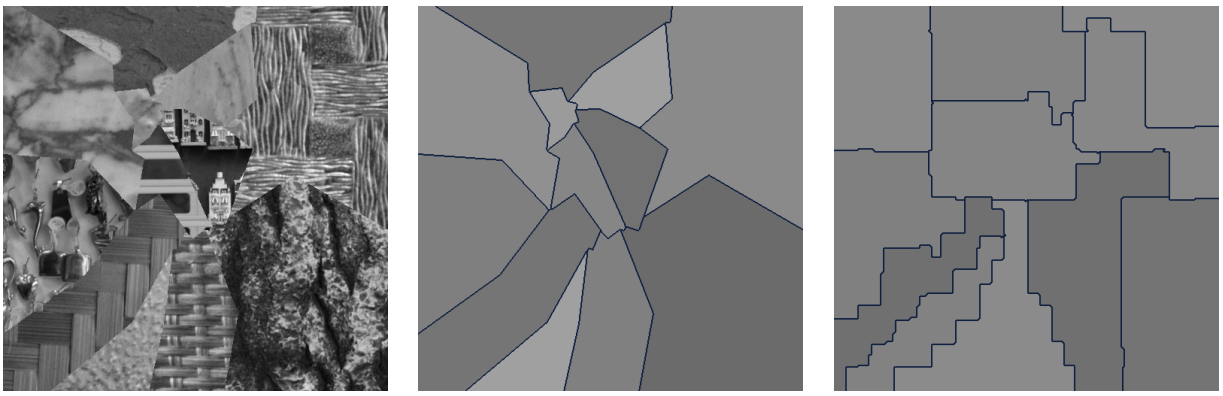


(a) Original Image (b) Ground Truth (c) Segmented Image

Figure 2.10: Grayscale Benchmark image

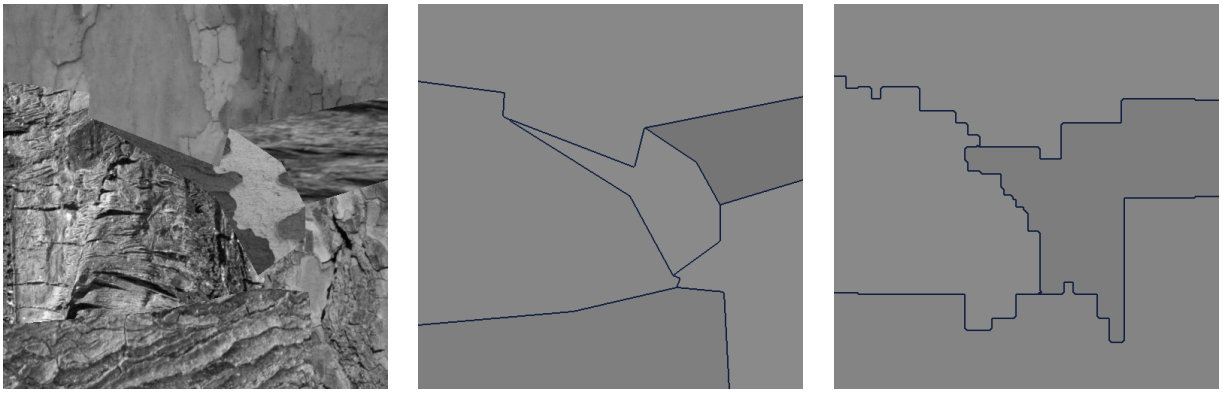
(a) Original Image (b) Ground Truth (c) Segmented Image

Figure 2.11: Grayscale Benchmark image



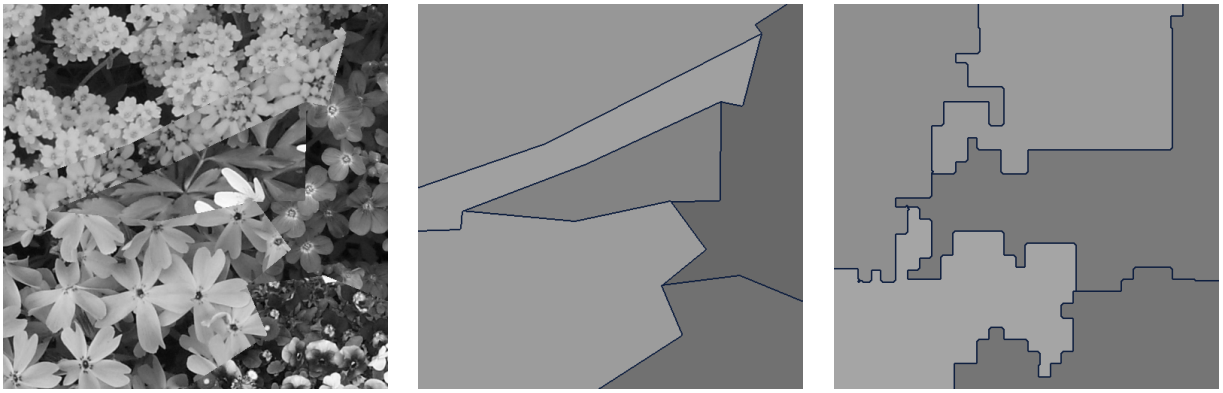
(a) Original Image (b) Ground Truth (c) Segmented Image

Figure 2.12: Grayscale Benchmark image



(a) Original Image (b) Ground Truth (c) Segmented Image

Figure 2.13: Grayscale Benchmark image

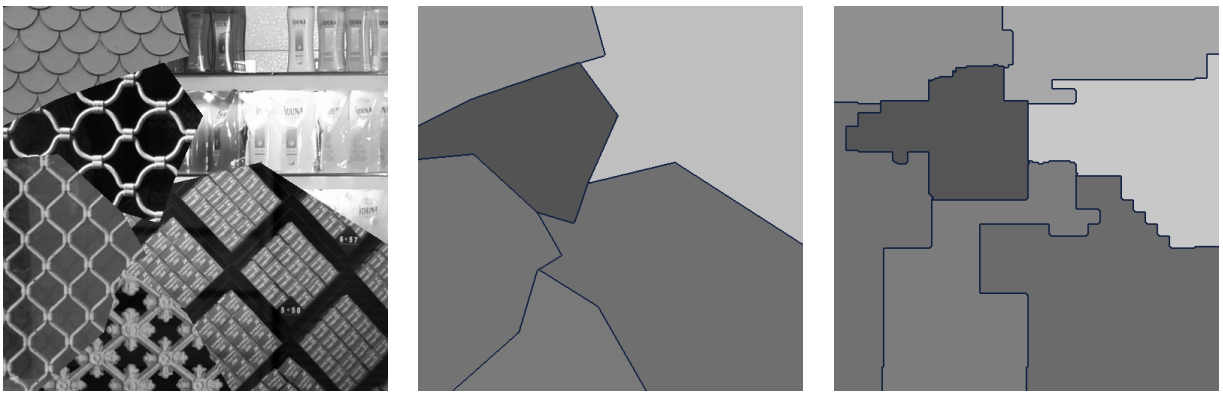


(a) Original Image (b) Ground Truth (c) Segmented Image

Figure 2.14: Grayscale Benchmark image

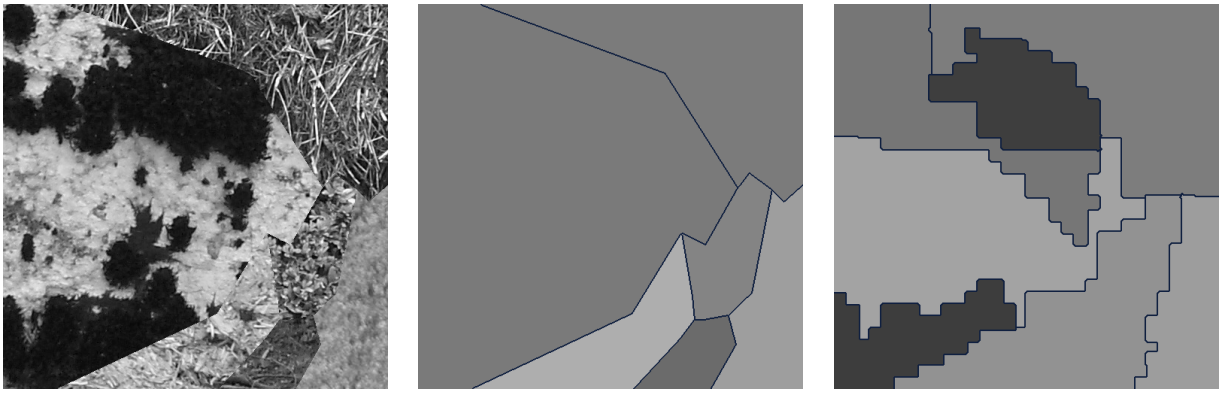
(a) Original Image (b) Ground Truth (c) Segmented Image

Figure 2.15: Grayscale Benchmark image



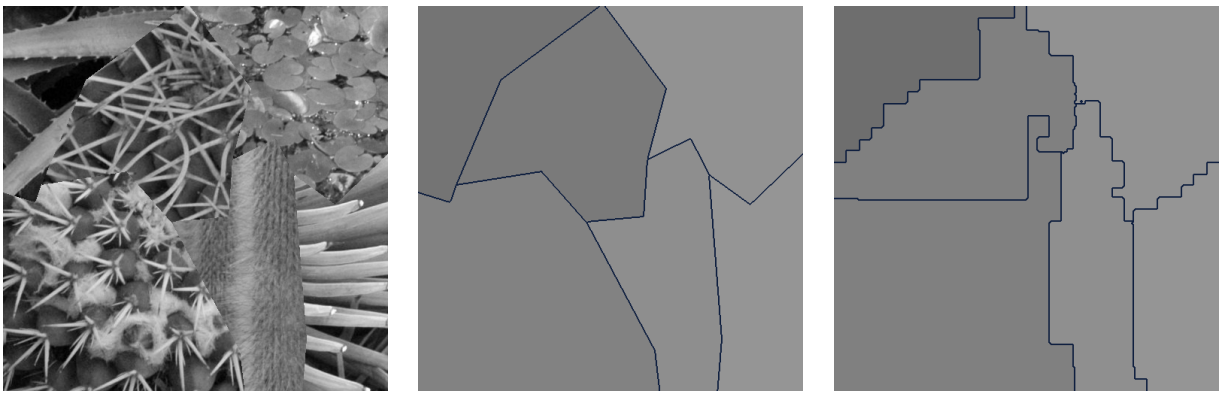
(a) Original Image (b) Ground Truth (c) Segmented Image

Figure 2.16: Grayscale Benchmark image



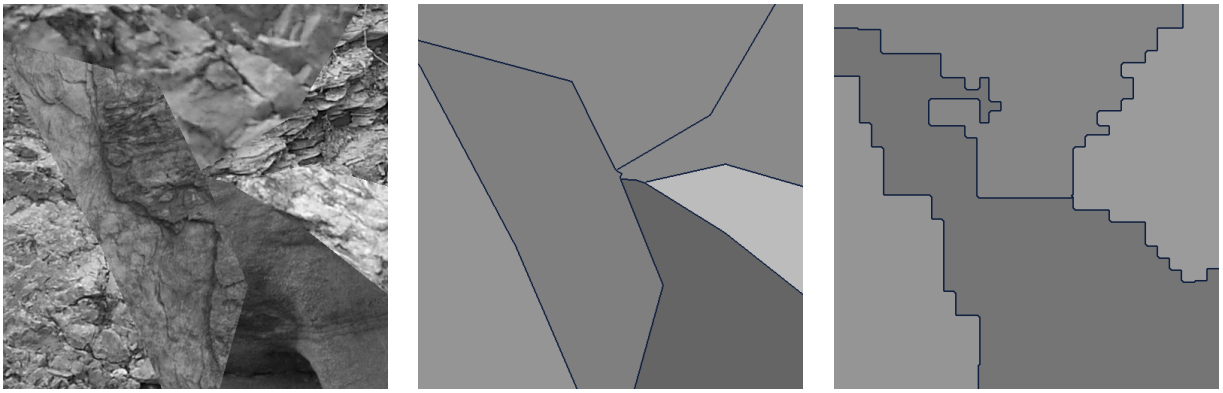
(a) Original Image (b) Ground Truth (c) Segmented Image

Figure 2.17: Grayscale Benchmark image



(a) Original Image (b) Ground Truth (c) Segmented Image

Figure 2.18: Grayscale Benchmark image

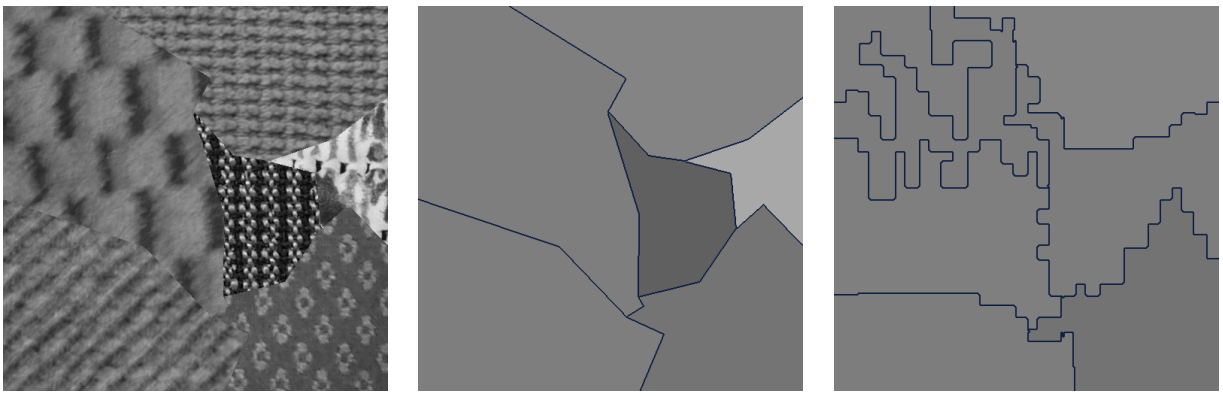


(a) Original Image (b) Ground Truth (c) Segmented Image

Figure 2.19: Grayscale Benchmark image

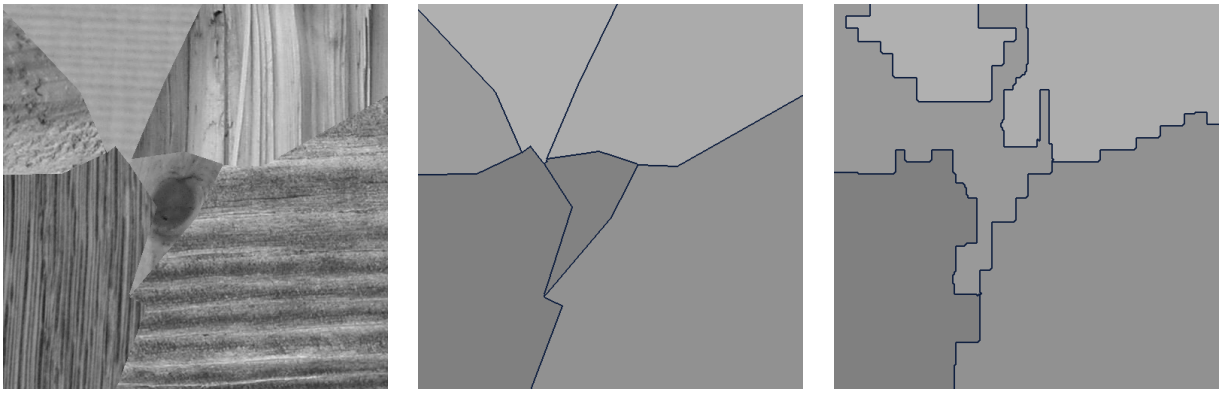
(a) Original Image (b) Ground Truth (c) Segmented Image

Figure 2.20: Grayscale Benchmark image



(a) Original Image (b) Ground Truth (c) Segmented Image

Figure 2.21: Grayscale Benchmark image



(a) Original Image (b) Ground Truth (c) Segmented Image

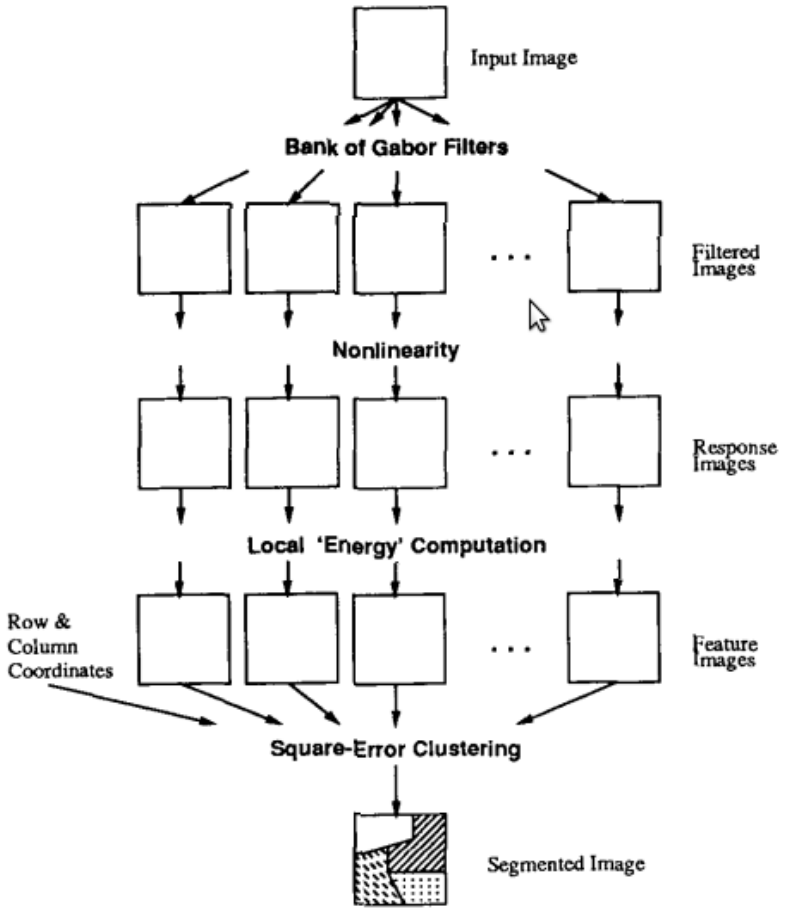
Figure 2.22: Grayscale Benchmark image

Texture segmentation using Gabor Filters

The texture segmentation algorithm by Gabor Filter is inspired by the multi-channel filtering theory for visual information processing in human visual system. The multi-channel filtering approach to texture analysis is intuitively appealing because the dominant spatial-frequency components of different textures are different. An important advantage of the multi-channel filtering approach to texture analysis is that one can use simple statistics of gray values in the filtered images as texture features. This simplicity is the direct result of decomposing the original image into several filtered images with limited spectral information.

3.1 Introduction

In the suggested research paper, a bank of Gabor filters is used to characterize the multiple channels. The filter set forms an approximate basis for a wavelet transform, with the Gabor function as the wavelet. Each (selected) filtered image is subjected to a bounded non linear transformation that behaves as a blob detector. The combination of multi-channel filtering and the non linear stages can be viewed as performing a multi-scale blob detection. Texture discrimination is associated with differences in the attributes of these blobs in different regions. A statistical approach is then used where the attributes of the blobs are captured by texture features defined by a measure of energy in a small window around each pixel in each response image. This process generates one feature image corresponding to each filtered image.


Input Image
Bank of Gabor Filters
Filtered Images
Nonlinearity
Response Images
Local 'Energy' Computation
Row & Column Coordinates
Feature Images
Square-Error Clustering
Segmented Image

3.2 Algorithm

The texture segmentation system involves the following three steps:

- Decomposition of the input image using a filter bank
- Feature extraction, and
- Clustering

3.2.1 Decomposition of input image using a filter bank

The multi-channel is represented with the help of 2-dimensional Gabor Filter. A two-dimensional Gabor function consists of a sinusoidal plane wave of some frequency and orientation, modulated by a two-dimensional Gaussian. Gabor filters has optimal joint localization, or resolution, in both the spatial and the spatial-frequency domains which is highly favourable for texture segmentation because filters with smaller bandwidths in the spatial-frequency domain are more desirable for finer distinctions among different textures. On the other hand, accurate localization of texture boundaries requires filters that are localized in the spatial domain. However, the effective width of a filter in the spatial domain and its bandwidth in the spatial-frequency domain are inversely related. The Gabor filter in the spatial domain is given by:

$$h(x,y) = exp\left\{-\frac{1}{2}\left[\frac{x^2}{\sigma_x^2} + \frac{y^2}{\sigma_y^2}\right]\right\} cos(2\pi u_0 x + \phi)$$

where u_0 and ϕ are the frequency and phase of the sinusoidal planewave along the x-axis (i.e. the 0 degree orientation), and σ_x and σ_y are the space constants of the Gaussian envelope along the x- and y-axis, respectively. A Gabor filter with arbitrary orientation, θ degree, can be obtained via a rigid rotation of the x-y coordinate system. The frequency- and orientation-selective properties of a Gabor filter are more explicit in its frequency domain representation. With $\phi = 0$, the Fourier transform of the Gabor function in (1) is real-valued and given by:

$$H(u,v) = A\left(exp\left\{-\frac{1}{2}\left[\frac{(u-u_0)^2}{\sigma_u^2} + \frac{v^2}{\sigma_v^2}\right]\right\} + exp\left\{-\frac{1}{2}\left[\frac{(u+u_0)^2}{\sigma_u^2} + \frac{v^2}{\sigma_v^2}\right]\right\}\right),$$

where $\sigma_u = 0.5 * \pi * \sigma_x$, $\sigma_v = 0.5 * \pi * \sigma_y$. Such representations are, therefore, referred to as Modulation Transfer functions (MTF)

3.3 Choice of Filter parameters

We implement each Gabor filter as a discrete realization of the MTF in equation (2).We use four values of orientation θ: 0, 45, 90, and 135. For an image array with a width of N, pixels, where N , is a power of 2, the following values of radial frequency u0 are used:

$$1\sqrt{2}, 2\sqrt{2}, 4\sqrt{2}, \ldots, \text{ and } \frac{N_c}{4}\sqrt{2} \text{ cycles/image-width}$$

Note that the radial frequencies are 1 octave apart. The restriction to four orientations is made for computational efficiency. Note that filters with very low radial frequencies (e.g., 1 * $\sqrt{2}$ and 2 * $\sqrt{2}$ cycle/image width) can often be left out, because they capture spatial variations that are too large to correspond to texture. To assure that the filters do not respond to regions with constant intensity, we have set the MTF of each filter at (u,v) = (0,0) to zero. As a result each filtered image has a zero mean. The above parameters determine the filter set, however for computational efficiency, a subset of filtered image can be used. This translates into reduction in number of features. To determine the best subset of the filtered images (filters), we use following criteria: Let r(x,y) be the i^{th} filtered image and R(u,v)be its Discrete Fourier Transform. Since, the amount of overlap between the MTFs of the Gabor filters in our filter set is small, so the total energy E in s(x, y) can be approximated by:

$$E \approx \sum_{i=1}^{n} E_i$$

where

$$E_i = \sum_{x,y} [r_i(x,y)]^2 = \sum_{u,v} \mid R_i(u,v) \mid^2$$

To select a subset of filter, we sort the filters based on their energy and pick as many filter as necessary to account for at least 95% of the intensity variations in s(x, y).

3.3.1 Feature extraction

Firstly, each filtered image is subjected to a non-linear transformation represented as follows:

$$\psi(t) = tanh(\alpha t) = \frac{1 - e^{-2\alpha t}}{1 + e^{-2\alpha t}}$$

where α is a constant. In the experiment, $\alpha = 0.25$ is used resulting in a rapidly saturating, threshold-like transformation. Consequently, the application of the nonlinearity transforms the sinusoidal modulations in the filtered images to square modulations and, therefore, can be interpreted as a blob detector. Now, instead of processing individual blob to obtain its attributes, we simply compute the average absolute deviation (AAD) from the mean in small overlapping windows to obtain the feature image. For, accurate localization of texture boundaries we use Gaussian weighted windows. For a Gabor filter with radial frequency u_0,the average size is given by:

$$T = \frac{N_c}{u_0} \text{ pixels}$$

3.3.2 Clustering

Once feature image is obtained, it is required to group the pixels into a number of clusters representing the texture regions. We use the k-means algorithm. This is essentially a supervised step, as it expects the prior knowledge about the number of textures. The algorithm of k-means is as follows:

1. Initialize centroids of K-clusters randomly.
2. Assign each sample to the nearest centroid.
3. Calculate centroids (means) of K-clusters.
4. If centroids are unchanged, done. Otherwise, go to step 2.

In texture segmentation, neighbouring pixels are very likely to belong to the same texture category so we include the spatial coordinates of the pixels as two additional features to take into account the spatial adjacency information in the clustering process.

3.4 Shortcomings

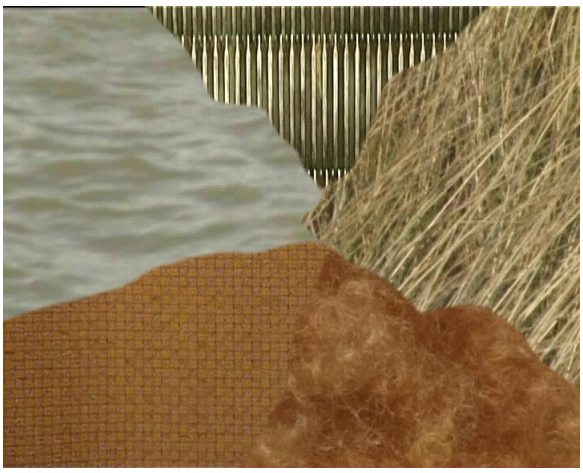
We found that this algorithm did not give desirable results, as shown in following figure. Possible reasons for the same could be:

- The position and bandwidth of Gabor filter in frequency domain needs to be firmly established to properly capture textural information. The Gabor filters are better in supervised texture segmentation where filter locations can be chosen based on empirical information of power

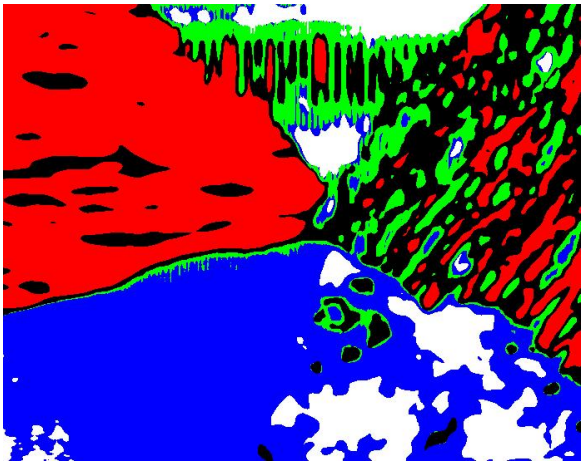
spectrum characteristics of different textures. In unsupervised case, a pool of band filters is chosen which might not be adequate to differentiate different texture due to its inadequacy of representing the different components of a texture.

- Secondly, the algorithm is too complex and requires a lot of fine tuning of various parameters on part of user to obtain a desired and acceptable result.

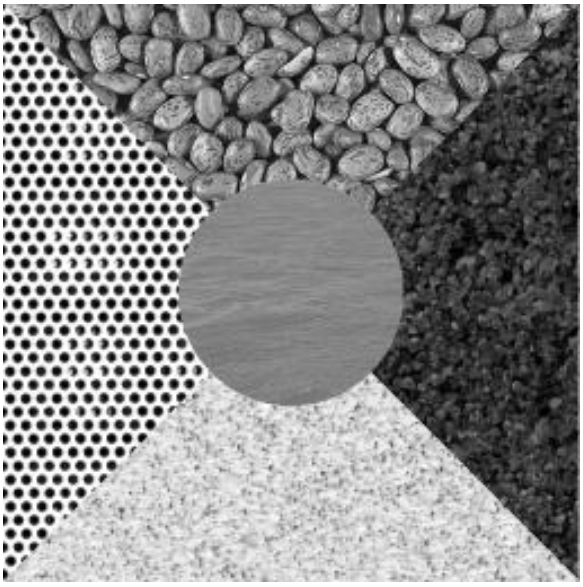
- Thirdly, k-means clustering again makes the algorithm supervised. Also, result is fairly unpredictable for same input as convergence to local minima, many a times leads to counter-intuitive answers. And, also the number of clusters k is an input parameter, therefore an inappropriate choice of k may yield poor results.



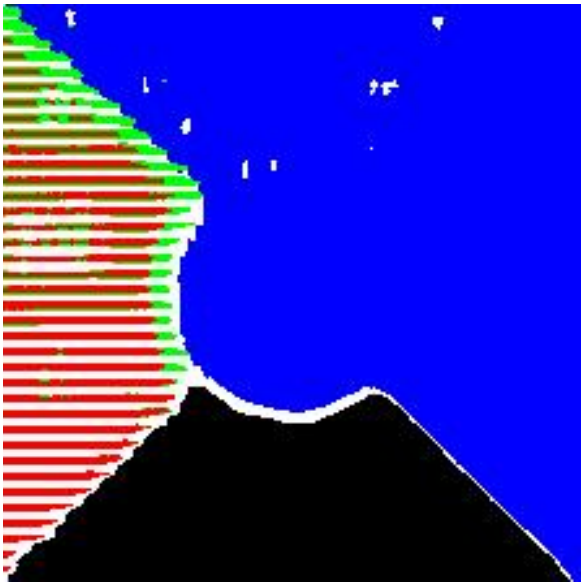
(a) Original Image



(b) Segmented Image



(c) Original Image



(d) Segmented Image

Figure 3.1: Segmentation based on Gabor filter approach with K-means clustering. Clearly, the results are not good.

Efficient Graph based Segmentation

4.1 Introduction

A fast and efficient graph based segmentation algorithm was proposed in [1]. The paper addresses a general problem of segmenting an image into regions. They use a graph based representation for the image and define a predicate for measuring evidence for boundary between two regions. The algorithm is quite fast, as it runs in time nearly linear over the number of edges in the graph.

We attempt to adapt this algorithm for the specific problem of Texture Segmentation.

4.2 Algorithm

The algorithm as mentioned in [1] is summarized as follows:

$G = (V, E)$ is an undirected graph with vertices $v_i \in V$, is the set of elements to be segmented and edges $(e_i, e_j) \in E$ correspond to pair of neighbouring vertices. Initially we start with all pixels of the image as separate vertices. Each edge has a corresponding non negative weight attached to it, which is a measure of the dissimilarity between the 2 nodes.

In the graph-based approach, a segmentation S is a partition of V into components such that each component (or region) $C \in S$ corresponds to a connected component in a graph $G' = (V, E')$, where $E' \subset E$. In other words, any segmentation is induced by a subset of the edges in E. There are different ways to measure the quality of a segmentation but in general we want the elements in a component to be similar, and elements in different components to be dissimilar. This means that edges between two vertices in the same component should have relatively low weights, and edges between vertices in different components should have higher weights.

4.2.1 Comparing similarity in regions

The paper defines a predicate D that tells if there is an evidence for boundary between 2 segments or not. It compares the inter-component differences to the intra-component difference and hence turns out to be adaptive with respect to local characteristics of the data.

The *internal difference* of a component is defined as the largest weight in the Minimum Spanning Tree (MST) of the component.

$$Int(C) = \max_{e \in MST(C,E)} w(e)$$

The *difference between two components* is the minimum weight edge connecting the 2 components.

$$Dif(C_1, C_2) = \min_{v_i \in C_1 v_j \in C_2, (v_i, v_j) \in E} w((v_i, v_j))$$

If there is no edge between the 2, we put Dif as ∞.

The region comparison predicate evaluates if there is evidence for boundary between a pair of components by using a threshold. We define

$$D(C_1, C_2) = \begin{cases} \text{true} & \text{if} \frac{Dif(C_1,C_2)}{MInt(C_1,C_2)} > M \\ \text{false} & \text{otherwise} \end{cases}$$

where

$$MInt(C_1, C_2) = min(Int(C_1) + \tau(C_1), Int(C_2) + \tau(C_2))$$

The function τ controls the degree to which difference between 2 components must be greater than internal difference in order for a boundary to exist.

$$\tau(C) = \frac{k}{|C|}$$

where $\mid C \mid$ is size of C, and k is some constant parameter.

4.2.2 Pseudo Code

The input is a graph G = (V, E), with n vertices and m edges. The output is a segmentation of V into components $S = (C_1, ..., C_r)$.
0. Smooth all the channels of the image using a Gaussian filter to compensate for digitization artefacts.
1. Sort E into $\pi = (o_1, ..., o_m)$, by non-decreasing edge weight.

2. Start with a segmentation S^0 , where each vertex v_i is in its own component.
3. Repeat step 3 for q = 1, . . . , m.
4. Construct S^q given S^{q-1} as follows. Let v_i and v_j denote the vertices connected by the q-th edge in the ordering, i.e., $o_q = (v_i, v_j)$. If v_i and v_j are in disjoint components of S^{q1} and $w(o_q)$ is small compared to the internal difference of both those components, then merge the two components otherwise do nothing (as per the threshold defined above).
5. Return $S = S^m$.

4.3 Implementation Issues

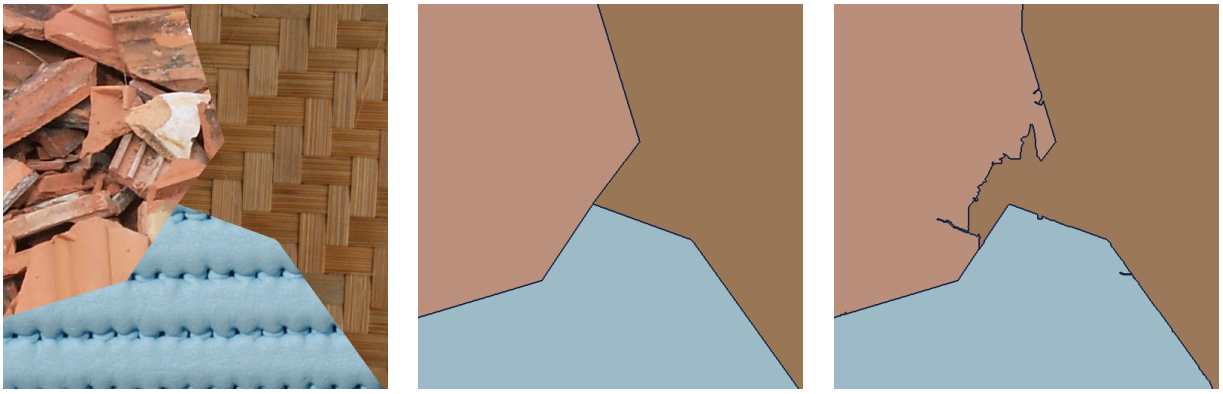
The segmentation S can be maintained as a disjoint set data structure, as used in the LBP algorithm.

The running time of the above algorithm will be $O(mlogm)$ for the Step 0 (sorting), and then step 1-3 take $O(m\alpha(m))$ time, where α is the slow growing inverse Ackerman function.

In order to check whether two vertices are in the same component we use set-find (a function defined for disjoint set DS) on each vertex, and in order to merge two components we use set union.

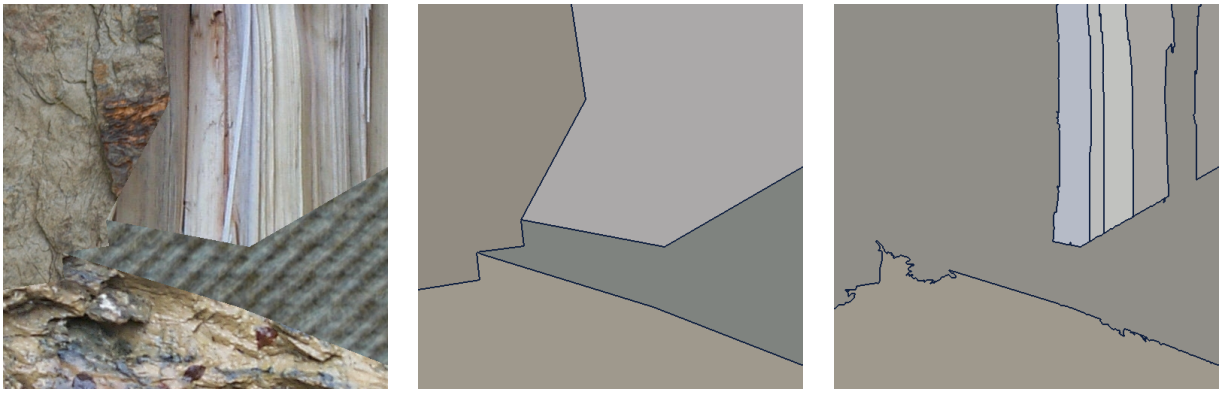
Thus there are at most three disjoint-set operations per edge. The computation of MInt can be done in constant time per edge if we know Int and the size of each component. Maintaining Int for a component can be done in constant time for each merge, as the maximum weight edge in the MST of a component is simply the edge causing the merge. The size of a component after a merge is simply the sum of the sizes of the two components being merged.

4.4 Results



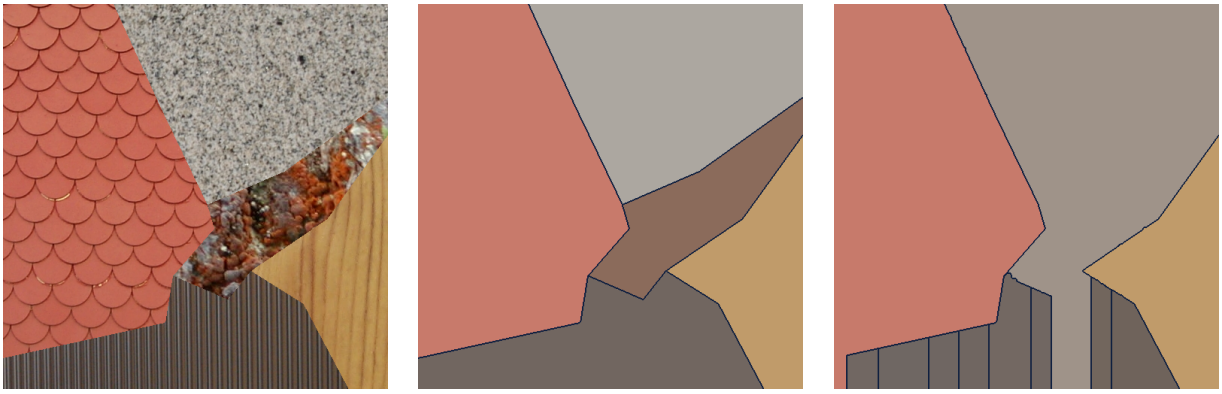
(a) Original Image (b) Ground Truth (c) Segmented Image

Figure 4.1: Color Benchmark image



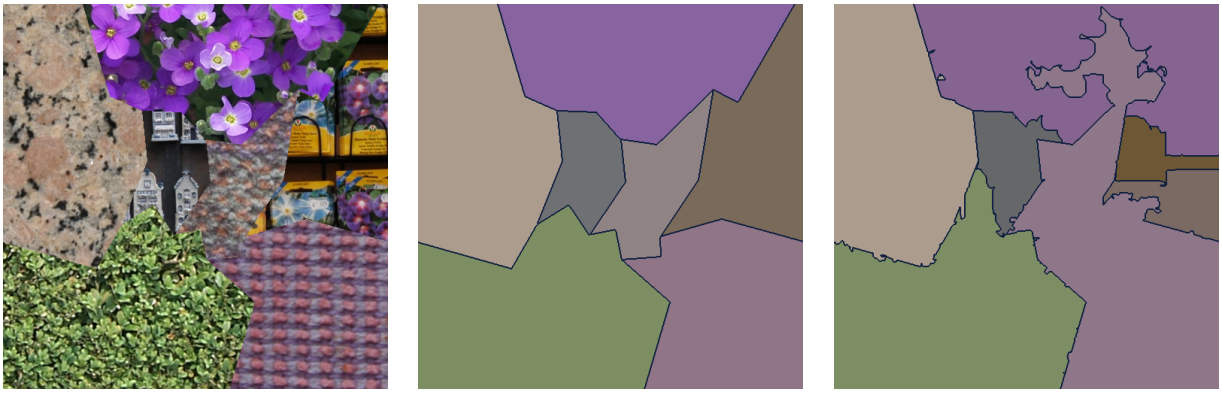
(a) Original Image (b) Ground Truth (c) Segmented Image

Figure 4.2: Color Benchmark image



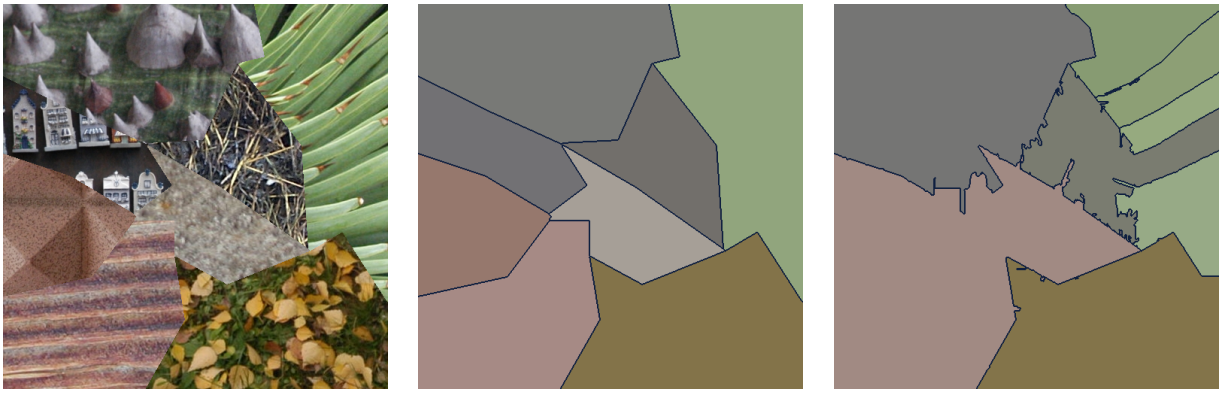
(a) Original Image (b) Ground Truth (c) Segmented Image

Figure 4.3: Color Benchmark image



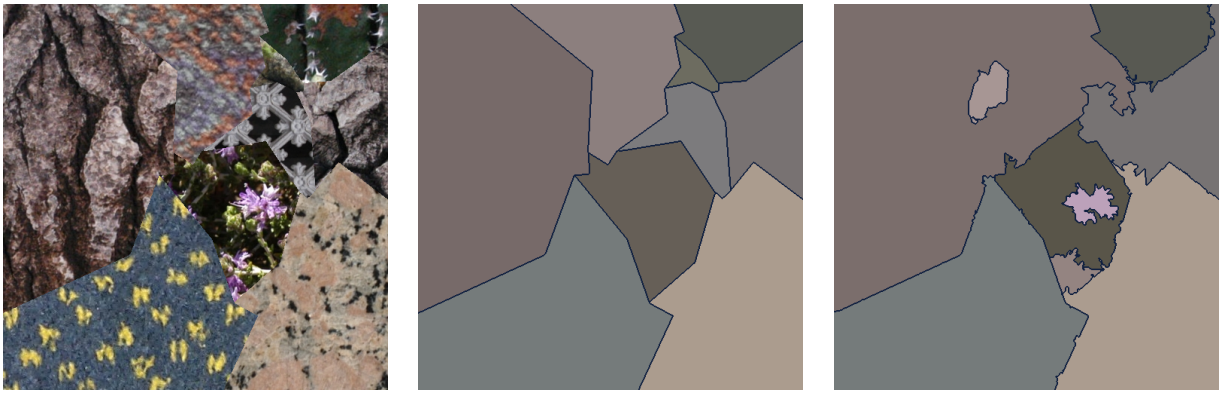
(a) Original Image (b) Ground Truth (c) Segmented Image

Figure 4.4: Color Benchmark image



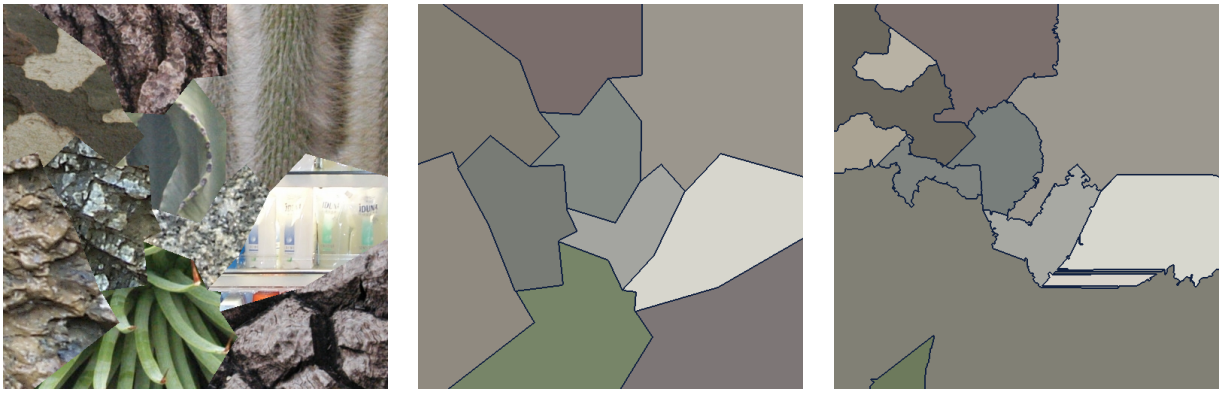
(a) Original Image (b) Ground Truth (c) Segmented Image

Figure 4.5: Color Benchmark image



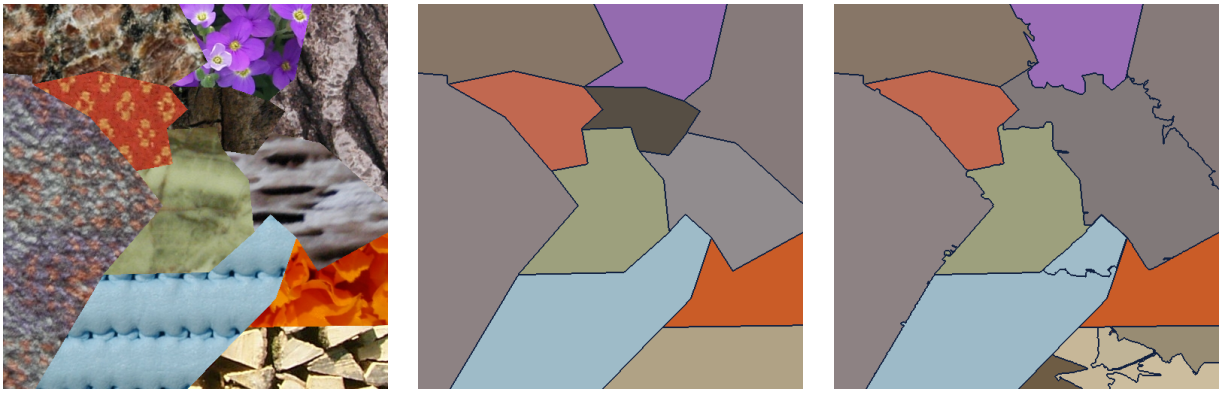
(a) Original Image (b) Ground Truth (c) Segmented Image

Figure 4.6: Color Benchmark image



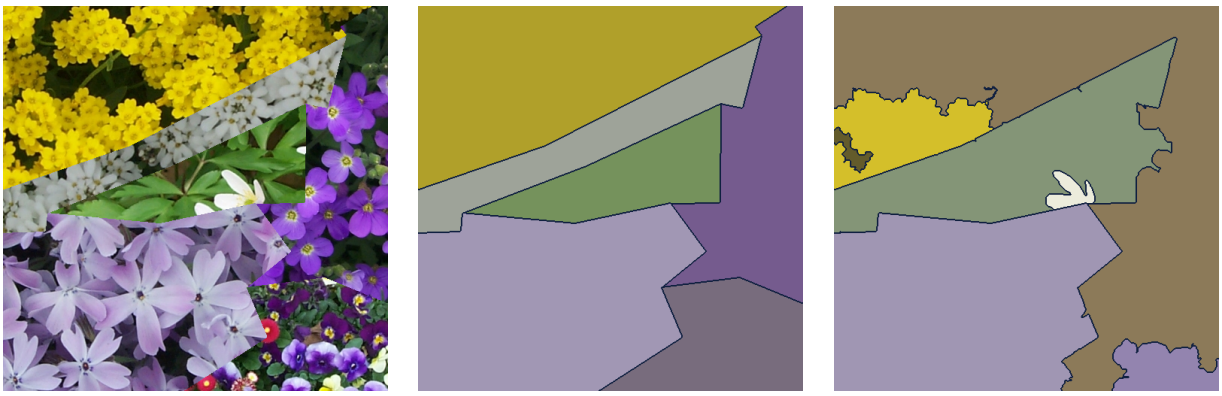
(a) Original Image (b) Ground Truth (c) Segmented Image

Figure 4.7: Color Benchmark image



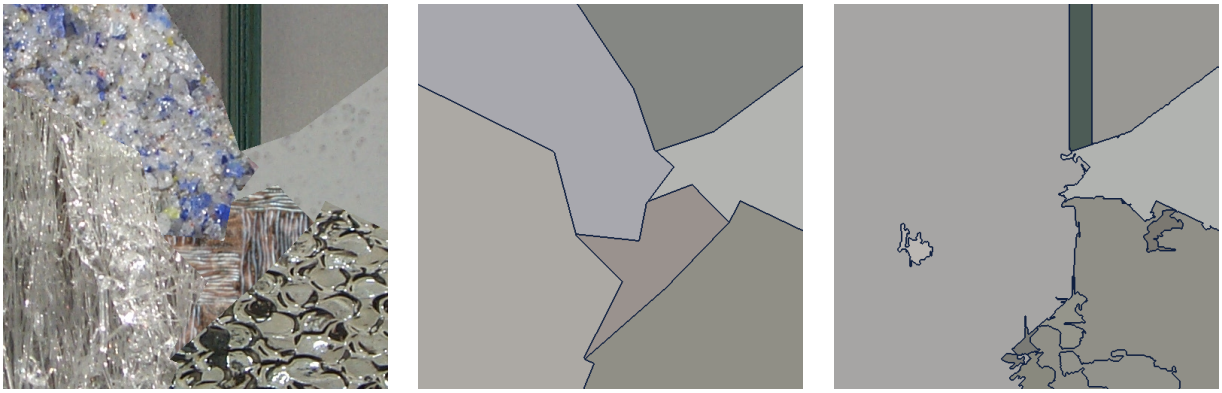
(a) Original Image (b) Ground Truth (c) Segmented Image

Figure 4.8: Color Benchmark image



(a) Original Image (b) Ground Truth (c) Segmented Image

Figure 4.9: Color Benchmark image

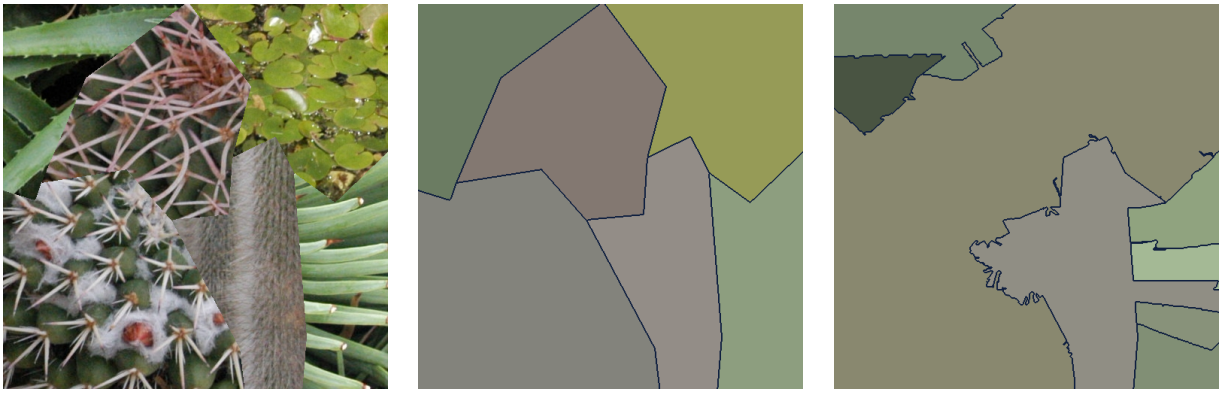


(a) Original Image (b) Ground Truth (c) Segmented Image

Figure 4.10: Color Benchmark image

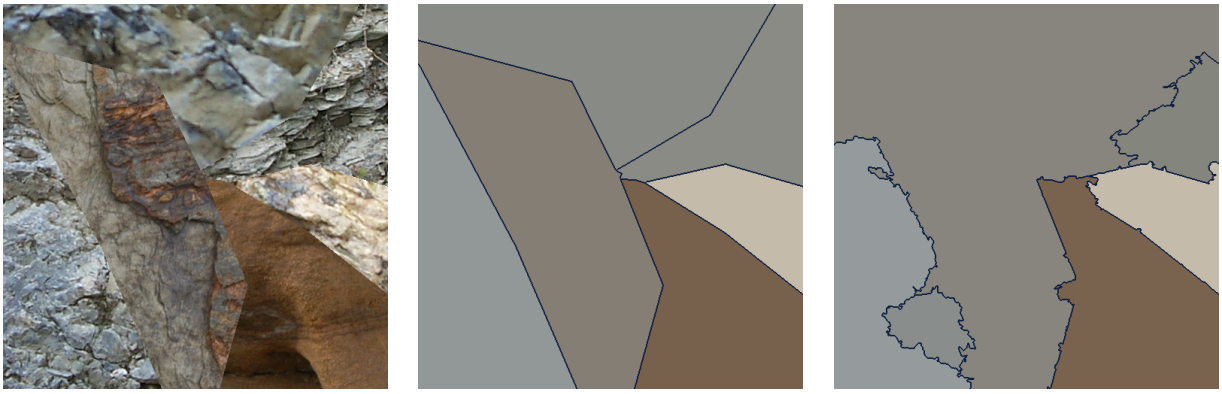
(a) Original Image (b) Ground Truth (c) Segmented Image

Figure 4.11: Color Benchmark image



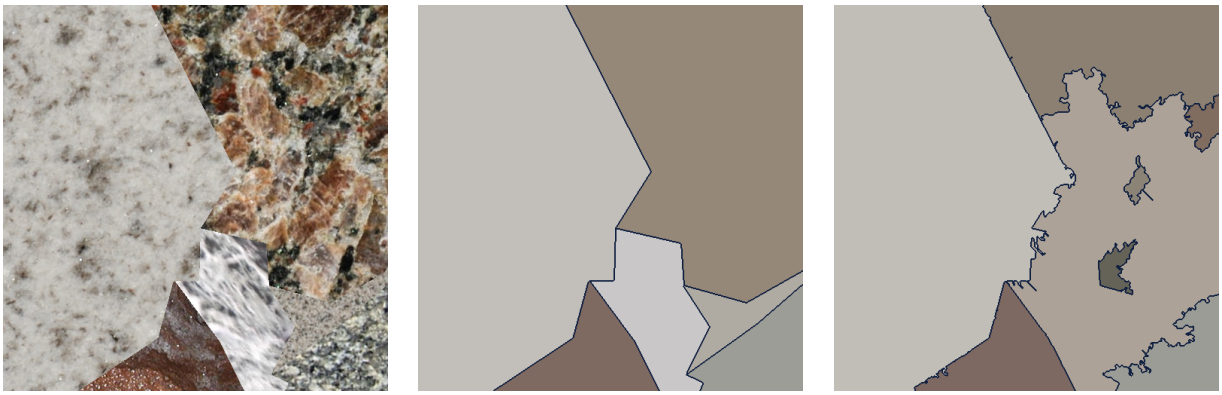
(a) Original Image (b) Ground Truth (c) Segmented Image

Figure 4.12: Color Benchmark image



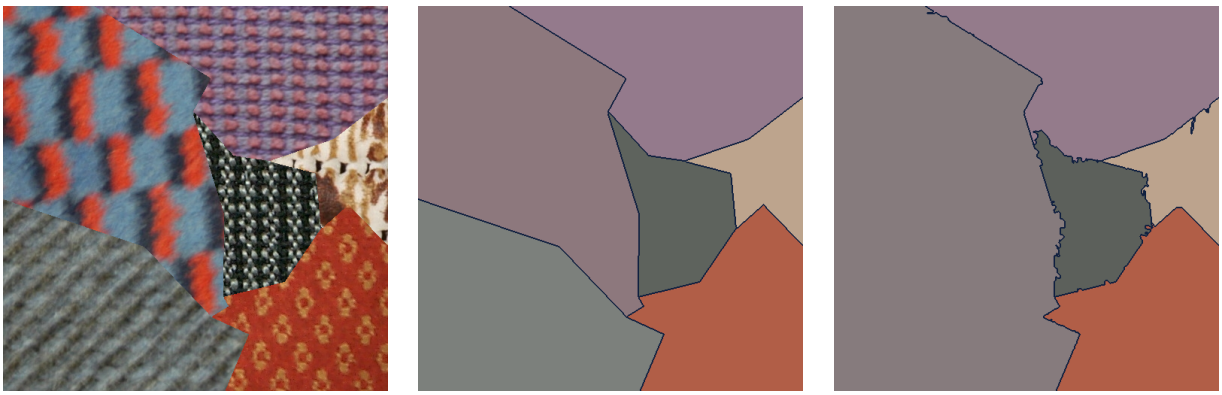
(a) Original Image (b) Ground Truth (c) Segmented Image

Figure 4.13: Color Benchmark image



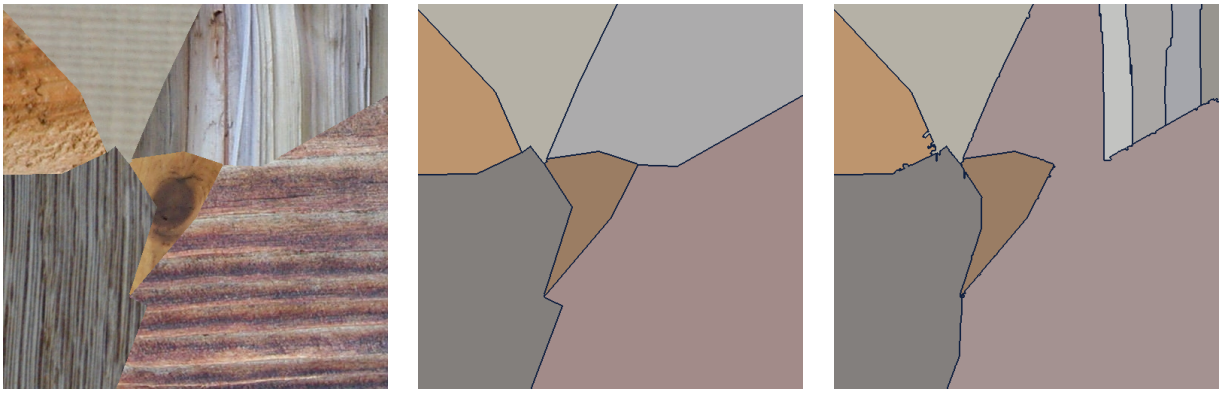
(a) Original Image (b) Ground Truth (c) Segmented Image

Figure 4.14: Color Benchmark image



(a) Original Image (b) Ground Truth (c) Segmented Image

Figure 4.15: Color Benchmark image



(a) Original Image (b) Ground Truth (c) Segmented Image

Figure 4.16: Color Benchmark image

Benchmarking and Evaluation

We found that benchmarking and evaluating a texture segmentation algorithm was itself a research problem. We followed the approach mentioned in [2]. The above approach is available as a web based service for comparing different algorithms to each other at [5]. The portal generates random 512×512 texture mosaics using a large collection of high resolution textures images, using a Voronoi polygon random generator. For each such mosaic, the corresponding ground truth and mask image is also generated. It also has provision for adding noise to the images, to test the noise robustness of the segmenter.

5.1 Method

The evaluation is done on 27 criteria. It mutually compares the ground truth image with corresponding machine segmented regions. The criteria are grouped as

- Region based

  These criteria compare the machine segmented regions with the correct ground truth regions. The overlap acceptance is controlled by a parameter k, as, correct detection occours when, (where R_i (ground truth) and R_j are the regions to be compared)

  $$\mid R_i \cap R_j \mid \geq k \mid R_i \mid$$

  The parameter k can be varied to plot a graph for correct detection vs k.

  Apart from the above, various other parameters for comparison are proposed in [4] like:

- Pixel wise

- Consistency measures

- Clustering comparison criteria

More details regarding the criteria set in [4]. We have used the Region based criteria (Correct Segmentation (CS)) as the parameter to compare results.

Comparison

6.1 Objective Results

6.1.1 Correct Segmentation

LBP based Segmentation

We achieved a Correct Segmentation rate of 42% on the Prague Grayscale dataset. The variation with k is shown: The graph falls rapidly as k (the

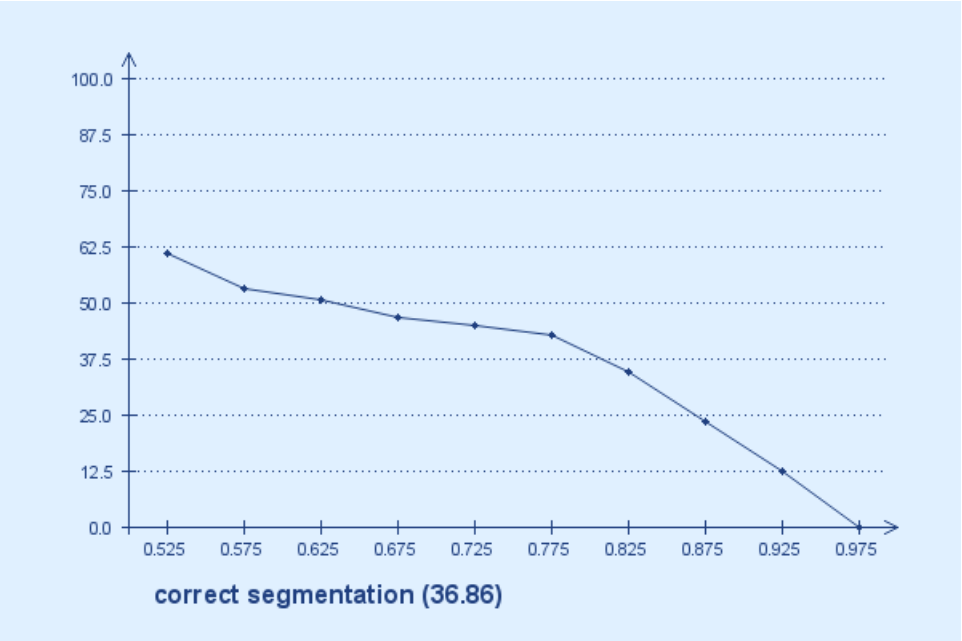


Figure 6.1: Correct segmentation rate vs k for LBP based Segmenter

percentage overlap required to consider a hit) increases. This implies that though, on a coarse level this algorithm could identify regions better than the second algorithm, however, at finer level, the boundaries were not as good as the second (since the fall in the curve for second algorithm is much less).

Our detailed results are available at here.
This algorithm implementation results were ranked **first** among all the grayscale texture segmentation algorithms on Prague Benchmark.

Graph based Segmentation

We achieved a Correct Segmentation rate of 38% on the Prague Color Dataset. The variation of the above with k is shown in the figure:

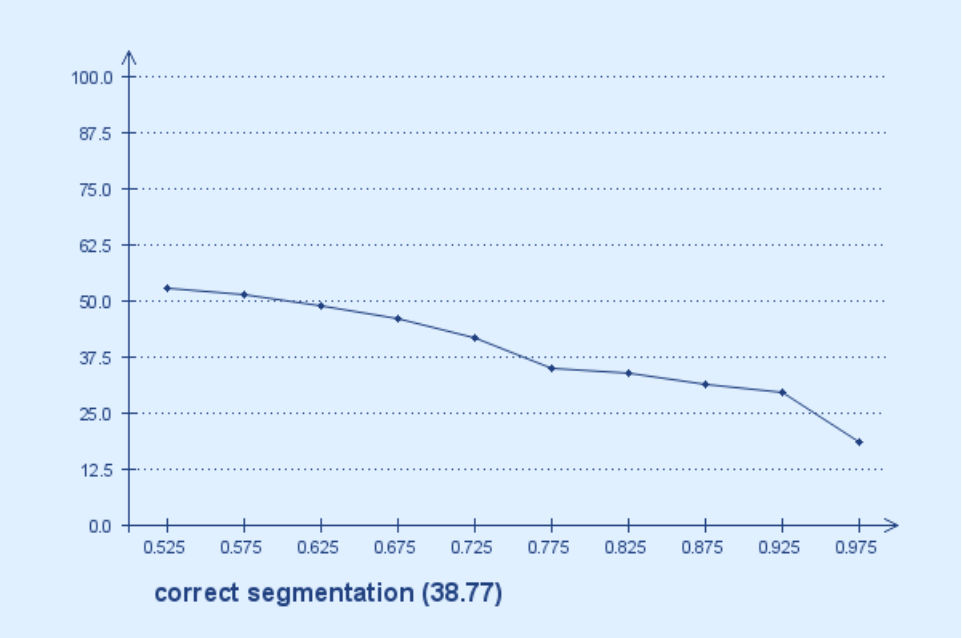


Figure 6.2: Correct segmentation rate vs k for Graph based Segmenter

The graph is more stable than the previous algorithm, that implies that the regions classified still remained classified correctly as the k (required percentage overlap to consider a hit) increased. This implies that Graph algo gave more precise boundaries for regions.

Our detailed results are available at here.
This algorithm implementation was ranked **23** among all the Color based texture segmentation algorithms on Prague Benchmark.

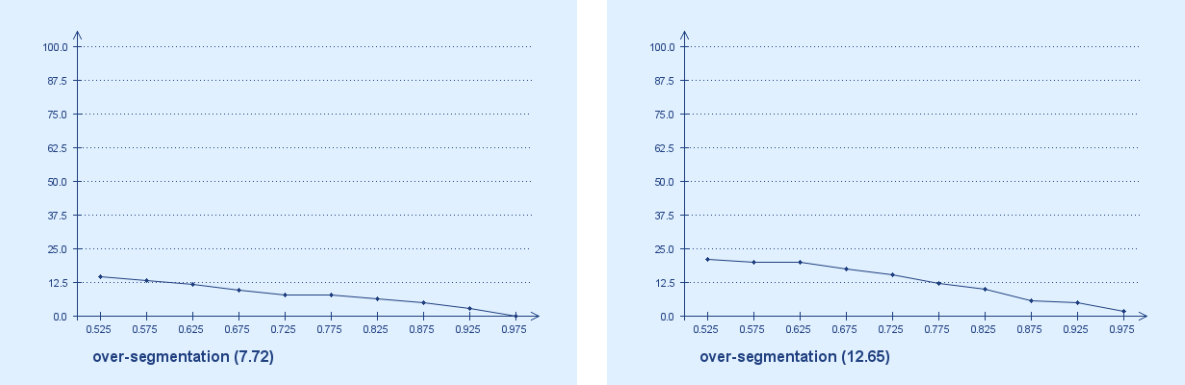
6.1.2 Over Segmentation

Over segmentation was defined as:

$$\sum_{i=1}^{x} |R_m \cap R_{ni}| \geq k|R_m|$$

i.e., over segmentation is said to occur when a region gets segmented into multiple regions.

As we can see from the graphs, LBP based algorithm generally gave lesser over segmentation.



(a) For LBP algorithm (b) For Graph Algorithm

Figure 6.3: Over Segmentation rate vs k

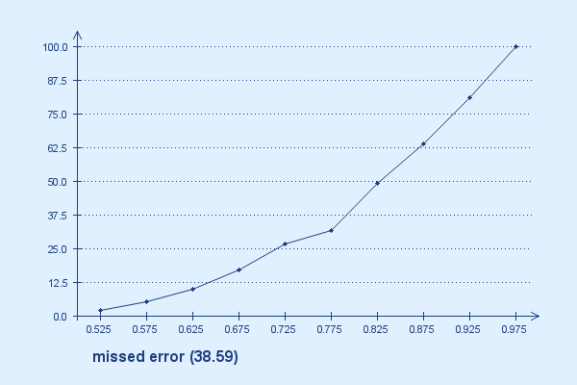
This shows, as Graph based algorithm gave more over segmentation, hence,it was less stable to variations in the texture.

6.1.3 Missed Error

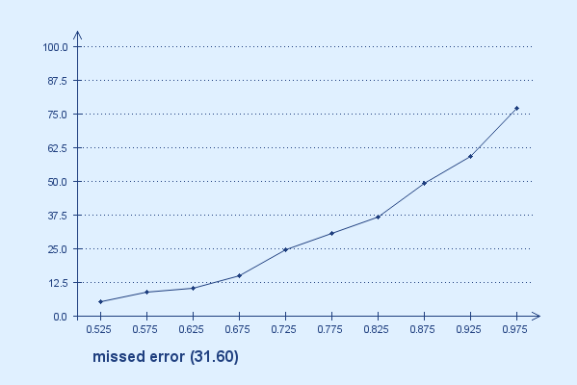
Missed error was defined when:

- R $\notin$ Correct Segmentation
- R $\notin$ Over Segmentation
- R $\notin$ Under Segmentation

As we can see, the missed rate in case of LBP was slightly higher. This implies Graph algorithm, though gave high over/under segmentation, was still able to identify most regions and lead to lesser 'misses' compared to LBP.



(a) For LBP algorithm



(b) For Graph Algorithm

Figure 6.4: Missed rate vs k

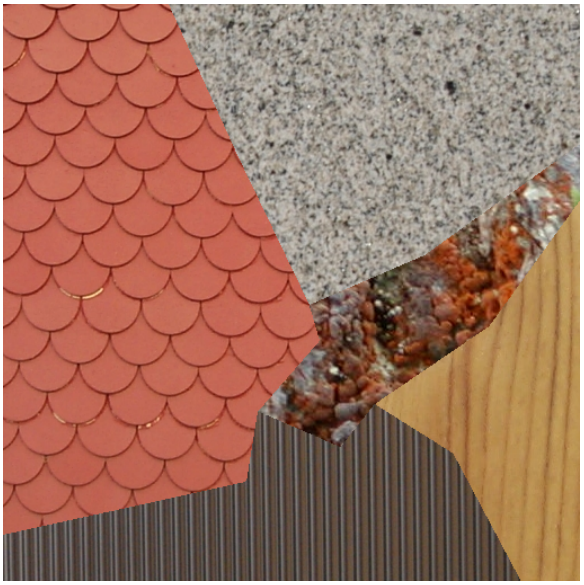
6.2 Observations

6.2.1 Accuracy

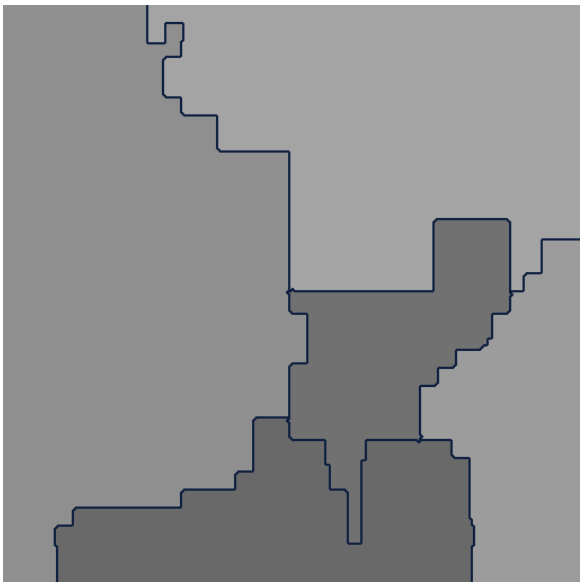
In general, the LBP based segmentation algorithm gave more accurate segmentation results compared to Graph based segmenter. This is because the LBP/C histogram is able to encode the texture information very well.
The pixel level precision in terms of boundary of the segment was much better in case of the Graph based algorithm. This is because the graph algorithm works at pixel level, and also uses color information to make classification decisions for each pixel. LBP algorithm in this case didn't give as good results at it is a split-and-merge algorithm and is constrained by a minimum block size during split stage. Even the pixel-wise classification is not able to achieve very fine boundaries. Its classification is somewhat blocky in nature.

The graph based algorithm was more sensitive to inhomogeneity of texture, while LBP based algorithm responded well to variations in a particular texture pattern.

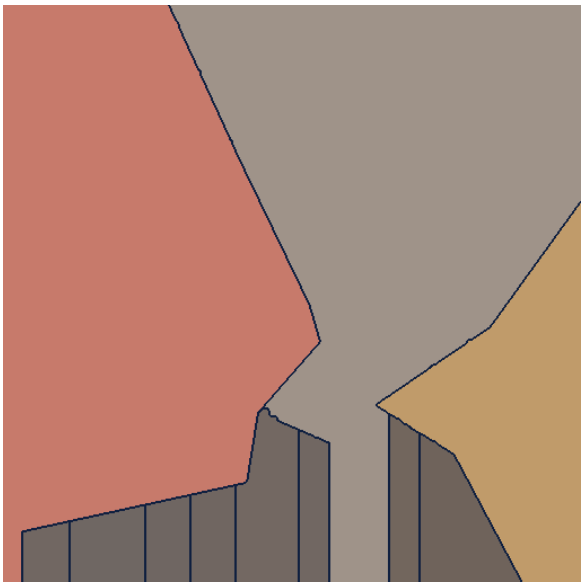
The graph based segmenter required extensive fine-tuning of parameters for each new test image, such as σ for smoothing, the threshold (k) and size of minimum segment to achieve desirable segmentation. This is because, being a pixel level algorithm it does not encode regional information, but works purely on thresholds. LBP on the other hand was more autonomous and didn't require much fine-tuning. It could even encode very complicated and changing textures with ease.



(a) Original Image



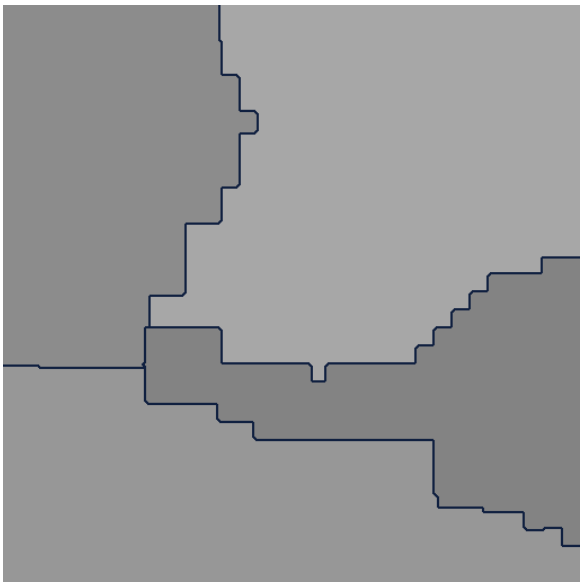
(b) Segmented using LBP algorithm



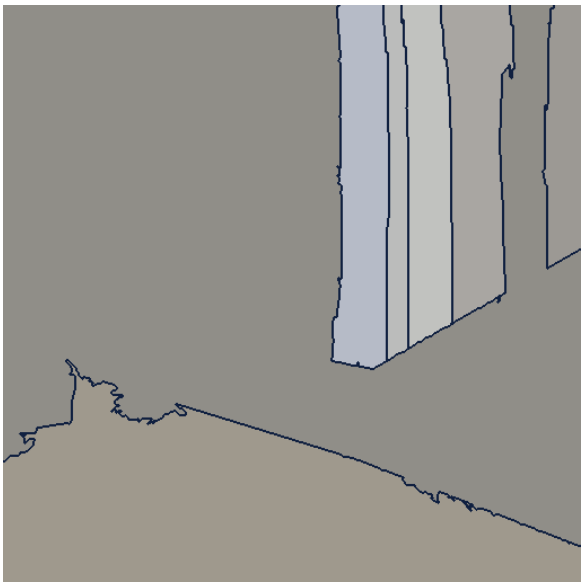
(c) Segmented using Graph based algorithm

Figure 6.5: LBP based algorithm produced more blocky results

(a) Original Image



(b) Segmented using LBP algorithm



(c) Segmented using Graph based algorithm

Figure 6.6: LBP based algorithm is stable against variations in the texture

6.2.2 Speed of implementation

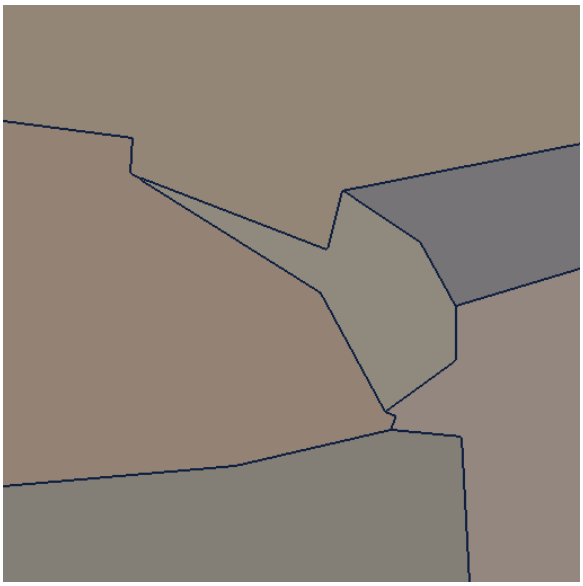
Our MATLAB implementation of LBP based algorithm took on average 4-10 minutes for a 256×256 image for sufficiently fine-grained classification. The C++ implementation of graph based algorithm on the other hand was much faster, running in time linear in the number of pixels in the image, taking less than 5 seconds for even a 512×512 image.

This shows the real world application of the Graph based algorithm particularly for texture based segmentation in real-time videos.

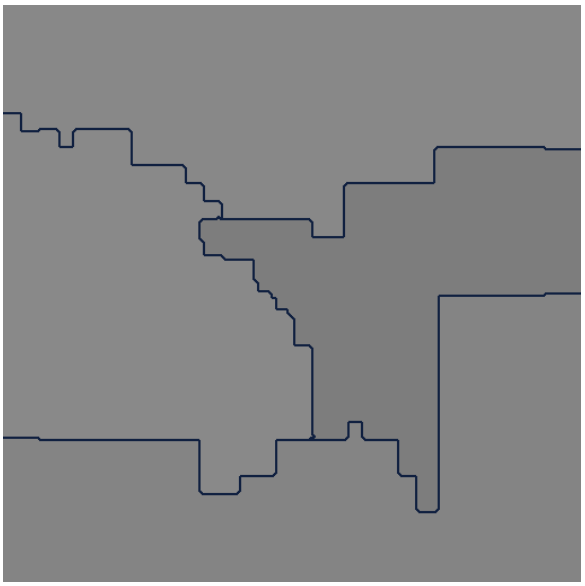
6.2.3 Specific use cases

The graph based algorithm was very efficient and fast, and used the color information in the image very well to produce satisfactory segmentation results. Since most natural images today are coloured, this algorithm can be well suited for object recognition in natural scene images.
The LBP based algorithm was much more stable in terms of encoding actual texture information, and gave on average better performance than graph based even while working on grayscale images and not using any color information. This shows the power of the LBP/C feature in encoding the texture information, and can be considered for use in case of segmentation that includes very complicated and very similar textures (in terms of color, visual perception).

(a) Original Image



(b) Ground Truth

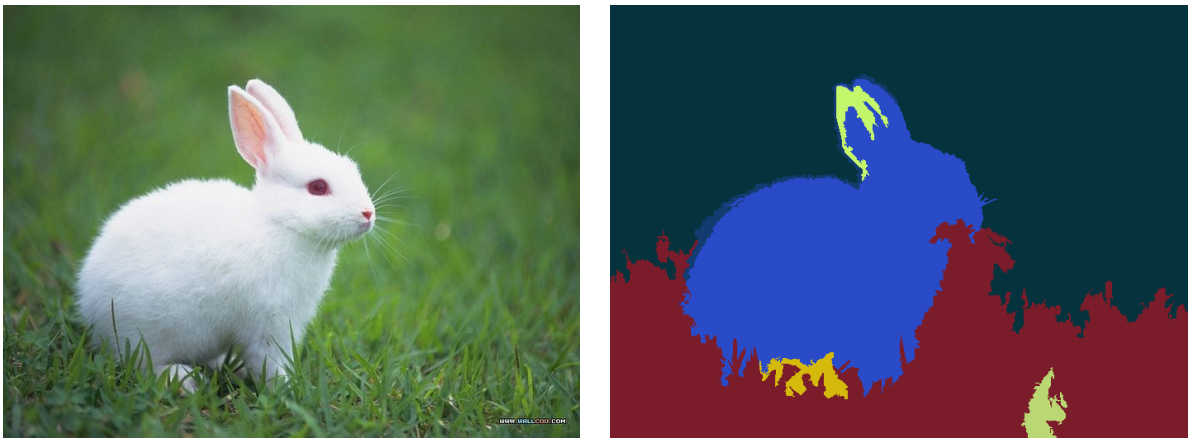


(c) Segmented using LBP based algorithm

Figure 6.7: LBP based algorithm is stable against variations in the texture

Real-life Applications

The graph based segmentation algorithm was tested on various natural images, and it gave good results. Some samples are shown This shows that even after not giving good results on the synthetic images in the Prague dataset, the graph algorithm could still be used in various practical scenarios where texture variations are not very high, and color information is available.



(a) Natural image

(b) Segmenting objects using graph based algorithm

Figure 7.1: Segment the rabbit out of image

(a) Natural image

(b) Segmenting objects using graph based algorithm

Figure 7.2: Segment the road out of image

(a) Natural image

(b) Segmenting objects using graph based algorithm

Figure 7.3: Segment the river out of image

Work Done

- We implemented/analysed image segmentation (mainly texture based segmentation) algorithms proposed in 3 papers.

  We also applied a generic segmentation algorithm (Graph based) for specific task of texture segmentation.

  - LBP based Texture Segmentation
    * Understanding of research paper and background analysis of associated concepts
    * Complete MATLAB implementation of research paper
    * Rigorous testing on dataset of synthetic images and fine-tuning of parameters
    * Objective and Subjective evaluation of results, compared to ground truth
    * Proposed and Implemented a technique for representation of regions for computationally efficient implementation. Here: 2.2.2.
  - Gabor Filter based Segmentation
    * Understanding of basic concepts and research paper on Gabor filter and Wavelet Transform
    * We analysed an existing implementation for Gabor based texture segmentation, however, results were not satisfactory
  - Graph based Texture Segmentation
    * Applied and implemented a generic image segmentation algorithm to the specific task of texture segmentation, by extensive experimenting with the various parameters etc.
    * Rigorous testing on dataset of synthetic images and fine-tuning of parameters
    * Objective and Subjective evaluation of results, compared to ground truth

- Compared the results of the proposed algorithms objectively and subjectively
- Analysed applicability of each in various real life situations

Nature of Work done

- We implemented/analysed 3 texture segmentation algorithms given in [6], [3] and [1].
- One of the algorithms was a generic image segmentation algorithm, we applied it for texture segmentation after extensive experimentation.
- We propose and implemented an algorithmic approach to speed up one algorithm
- We did an exhaustive comparative study based on objective parameters defined in [2], as well as subjective parameters.
- We also went beyond the scope of project, explored the real life applications of the algorithms.

Work Division

- Selection of research papers/background study/selecting testing datasets (All)
- LBP Based Segmentation: Implementation (Rohit and Nishit), Analysis/Testing on Datasets (Divyansh, Nishit)
- Gabor Filter: Understanding/Analysis/Testing (Divyansh, Rohit)
- Graph Based Segmentation: Implementation/Testing on Datasets (Rohit, Divyansh), Analysis (Nishit)
- Comparative Study/Observations (All)

Bibliography

[1] Pedro F. Felzenszwalb and Daniel P. Huttenlocher. Efficient graph-based image segmentation. *International Journal of Computer Vision*, 59, 2004.

[2] Michal Haindl and Stanislav Mikes. Texture segmentation benchmark. In *ICPR*, pages 1–4. IEEE, 2008.

[3] A. K. Jain and F. Farrokhnia. Unsupervised texture segmentation using gabor filters. *Pattern Recognition*, 24:167–1186, 1991.

[4] Stanislav Mikes. The prague texture segmentation benchmark criteria set. `http://mosaic.utia.cas.cz/download/segcrit.pdf`.

[5] Stanislav Mikes. The prague texture segmentation datagenerator and benchmark. `http://mosaic.utia.cas.cz/`, 2008.

[6] Matti Pietikainen Timo Ojala. Unsupervised texture segmentation using feature distributions. *ICIAP 97 Proceedings of the 9th International Conference on Image Analysis and Processing*, 1:311–318, 1997.