Comparison of Texture Segementation Algorithms for Electron Microscopy Images

Andrew Kennard
Biophysics Program
Stanford University

***Abstract*—Image segmentation is a key bottleneck in analysis of electron microscope (EM) images, made challenging by the fact that many features in EM images differ not in intensity but in texture. I have explored several options for texture segmentation of images, based on feature vectors generated from local histograms of filtered versions of the image, which are then classified using a Support Vector Machine (SVM). I find that while these algorithms perform better than chance, they do not outperform a naive thresholding algorithm. More work must be done to optimize feature vectors for better separation and to improve the SVM classification for better segmentation.**

I. Introduction

Transmission Electron Microscopy (TEM or EM in this report) allows biologists to observe their specimens with unparalled level of detail. In addition to the clear benefit of enhanced spatial resolution due to a smaller diffraction limit, EM images also reveal many different cellular structures simultaneously, unlike fluorescence microscopy images, which typically label at most a few separate structures. Thus EM provides a holistic, high-resolution view of cells or tissues.

This benefit comes with the cost of increased image complexity and greater difficulty for automatic processing of EM images. Unlike in a fluorescence microscopy image, functionally distinct structures in an EM image may not differ merely in grayscale intensity but rather in more complex features such as image texture. This requires more sophisticated algorithms to parse, and in many cases segmentation of EM images is still performed manually, creating a significant bottleneck in the analysis workflow.

A. Related Work

Several previous attempts have been made to classify objects in EM images based on their texture. Proenca *et al.* used several indicators of texture based on spatial correlations in grayscale intensity, such as image entropy or local variance, to identify virus particles in EM images [1]. These indicators supplemented other approaches to segmentation, and were made simpler by the fact that viruses have extremely well-defined size, shape, and texture. It is unclear from this work if texture could be used to segment more unusually shaped regions.

Gorai *et al.* used a combination of several feature descriptors to identify textures in EM images: Local Binary Patterns and Gabor filters [2]. They found that a combination of filters was most effective at segmentation of textured regions. However, the particular structures of interest in that paper were again small and fairly distinct from the rest of the cell. A robust algorithm for any general structure in an EM image has not emerged, and so individual algorithms must be developed for given structures of interest.

B. Project Goal

In my research I perform a lot of electron microscopy, and I want to develop a segmentation algorithm that will correctly pick out the structures I am interested in from EM images. In order to develop a texture segmentation algorithm, I used several TEM images that I have taken of my cell of interest, a fish skin cell. These cells are studied for their ability to move (in order to heal wounds), and I am particularly interested in the dense meshwork that the cell polymerizes to power its movement (Fig. 1). In sections through one of these cells, the meshwork appears as dark swaths on one side of the cell. I would like to have an algorithm that would allow me to identify these dark regions specifically, avoiding other lower-intensity regions that have different texture on the other side of the cell. This is the goal of the three segmentation algorithms I explore in this report. I will describe the approach of each algorithm and identify the tunable parameters for each (and the choices for these parameters that I made). I will discuss the efficacy of each of these methods, and discuss possible directions to take to improve the algorithms.

II. Description of the Algorithms

One challenge in all of these algorithms was that the size of the images from the electron microscope is huge—several thousand pixels in each dimension. This dramatically increased the run time for many of the algorithms I was testing. In order to proceed more rapidly, I identified a subset of each image that was most relevant; in the images tested here, I used a subsection of an image that managed to capture most of a cell, which had a good representation of different textures, in particular the dark meshwork texture I am interested in. I furthermore reduced the size of the image in 2 by bilinear interpolation between pixels. I determined based on the Fourier transform of the image that the image was sufficiently oversampled that I would not be throwing away too much valuable high-resolution information by downsampling by a factor of two.

To simplify the following discussion I define the term *local histogram* in a specific sense: a local histogram is a histogram of the intensity values of a window of some size centered on

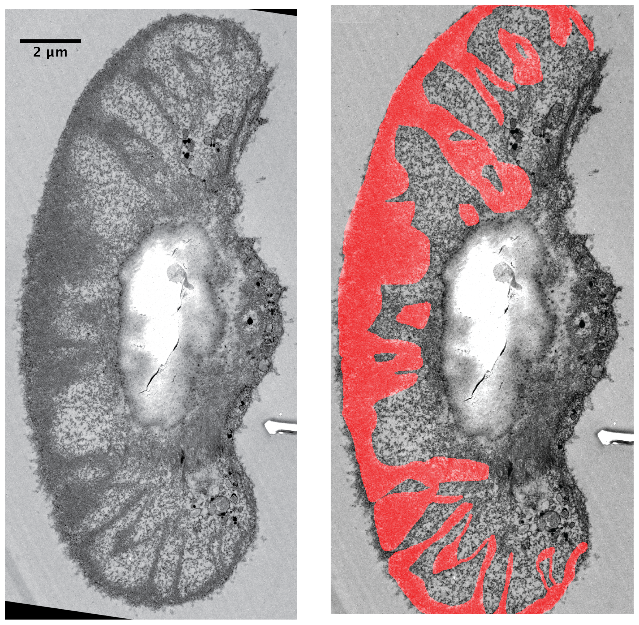

Fig. 1. Example EM image of a fish keratocytes. I am interested in the dark dense meshwork in the fan-like area of the cell (manually segmented in red on the right). Scale bar 2μm.

a particular pixel. Thus a local histogram can be associated with each pixel and provides some information about the distribution of image values around that pixel.

A. *Adaptive Thresholding*

The first algorithm I implemented was a naive algorithm based on thresholding the image. I employed adaptive thresholding to correct for uneven illumination in the image. Because intensity of different structures in the cell varied significantly, I used a large window size to ensure that the same structure in different areas was thresholded the same way. To ensure smooth differences in thresholding and to avoid discontinuities in thresholding due to different thresholds in different local windows, I thresholded a large window (linear dimensions equal to 1/10th the image's linear dimensions), but I only used the result in a smaller sub-window (1/40th the images linear dimension). I tried different values for this parameter, but the result was reasonably robust to the choice of parameter, as long as the sizes of the large window and small window were sufficiently different.

I found that employing Otsu's method was sufficient for effective thresholding out the meshwork from the area surrounding it. As is typical for adaptive thresholding, I had to choose a cutoff for the variance in (normalized) grayscale intensity within a window in order to avoid performing thresholding on a "blank" region of image. I chose this parameter ad-hoc to be 0.005 (grayscale values normalized from 0 to 1), and I found that this supressed thresholding of noise while still thresholding all "signal" regions.

As expected, many other things were also classified as 'foreground' by this thresholding approach, including small regions of dark intensity near the meshwork. I cleaned up the thresholded image using morphological closing (because the 'dark' region was the region of interest) and small region removal. I chose the size of the (square) structuring element and the cutoff size for small regions by eye after looking at the size of some representative objects I knew I wanted to exclude.

B. *Local Binary Pattern*

The next method I implemented was to incorporate a feature known as a Local Binary Pattern (LBP) [3], [4], which has been used successfully to identify textures in faces and even in some EM images. To calculate LBP, a set of 8 pixels around a reference pixel is chosen, and the grayscale intensity of these pixels is compared to the reference pixel. The intensity of each of these 8 pixels can either be higher or lower than the reference pixel's intensity, leading to an 8-bit "local binary pattern" associated with each pixel in the image. This 8-bit number for each pixel can be mapped to a new grayscale image, where now each pixel's intensity value encodes some information about the spatial variation in grayscales in the original image. Hopefully this new LBP image will represent texture more tangibly than the original image. By only looking at relative intensity, LBP also has the advantage of being invariant to changes in global intensity, e.g. due to illumination differences.

The number of possible LBPs can be reduced from 256 to 58 by grouping together LBPs corresponding to "uniform" differences in intensity. These are defined as LBPs in which the binarized relative intensity only changes once as one moves around the center pixel (i.e. moving clockwise aroudn the center pixel, 01110000 and 11000000 are uniform patterns while 01100110 is not). An advantage of collecting together all uniform patterns is that this allows the uniform LBPs (uLBPs) to be roughly rotation invariant (uniform patterns rotated by some angle will still be uniform).

I calculated LBPs using 8 pixels approximately 30 pixels away from a reference pixel. I chose this distance because it corresponded to a characteristic size scale in the image for which textures I recognized (such as the speckled texture) and wanted to distinguish from the meshwork texture. I also converted these LBPs to uLBPs using a look-up table (found online on the website Quant Geeks). I tried to code LBP myself, but found that my implementation was very inefficient. I then used code from MATLAB Central File Exchange written by Nikolay S. to calculate LBP efficiently.

The result of finding uLBPs on the image in Fig. 1 is shown in Fig. 2. Each grayscale value corresponds now to information about the relative intensity of nearby pixels. It can be hard to see differences in this image, but by looking closely one can see that regions of the meshwork texture have a finer pattern of LBP than the "background" regions which have more speckled texture. I found that noise in the image had a noticeable effect on the LBP result: it made the speckled LBP image look even more speckled. To mitigate this, I applied a 3x3 median filter prior to calculating LBPs. I found that this helped, but that

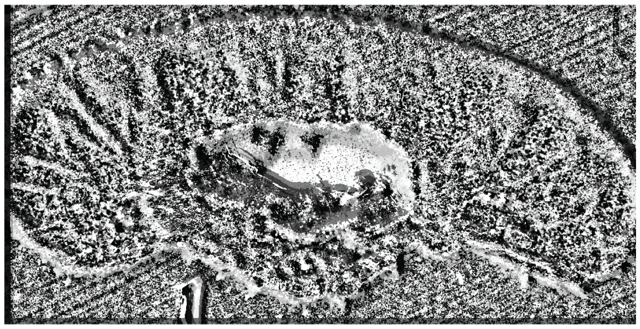

Fig. 2. The result of finding Local Binary Patterns (LBP) in an EM image. Grayscale value correspond to different LBPs, i.e. different arrangements of relative grayscale intensity around each pixel.

windows of any larger size started to mask the differences in signal between my desired foreground and background.

I then calculated a feature vector by counting up the occurences of each uLBP in a local window (a 58-bin histogram). The window size over which this histogram was calculated was fixed to be 31 pixels. I chose this size because it was large enough to capture representative samples of texture while small enough to detect small patches of texture, but this parameter could be further tuned as necessary. Because this window size is fairly large, I calculated an *integral histogram* for the image prior to performing the window operation. Similar to an integral image, each pixel in an integral histogram aggregates data from pixels above and to the left of it; however, instead of just summing up the intensity of those pixels, in an integral histogram one keeps running tallies in a set of specified bins, and updates it so that each pixel contains a *histogram* of the pixel intensities above and to its left. Thus an integral histogram will be a muldimensional array.

Once I had the feature vector, I needed to train a classifier to distinguish the foreground features from background features in the high-dimensional feature space. For this I used the Support Vector Machine method included in VLFeat. SVM is a rapid and simple-to-implement machine learning software, good for binary classification (which is exactly what is required for image segmentation). Each feature for image segmentation corresponds to a single pixel; in the training image I wanted to segment I had over 1 million pixels, so I had plenty of data with which to train an SVM. I had previously manually segmented out regions of texture of interest; this formed my ground truth for training. I fed into the algorithm every pixel in the training image, with its associated foreground or background label.

A limitation of the VLFeat SVM command is that it does not have a simple way to use nonlinear kernels which are typically implemented in other versions of SVM (which I did not realize at first). Thus I was constrained to train only a linear SVM classifier. This has only one hyperparameter to tune, which is the regularizer parameter. To choose the best value of this hyperparameter I performed 5-fold cross-validation on the entire training data set, choosing logarithmically spaced values of the regularizer hyperparameter; I chose the value that had the best average accuracy in the cross-validation test.

One thing I found surprising is that the SVM training converged very slowly; I kept raising the maximum number of allowed iterations, and the SVM algorithm would always terminate by maxing out the number of iterations. I am not very familiar with the ins and outs of SVM, so I am not sure what this suggests about my problem, but it is possible that there were other aspects of the learning procedure that needed to be optimized; using non-linear kernels would be a good start.

To test the algorithm, I used a second image which I had also manually segmented. I applied the exact same procedure for extracting feature vectors, and then applied the trained SVM classifer to get the predicted class for each pixel in the testing image. While a threshold of 0 in SVM score is conventionally used to get the predicted classification of testing data, I also varied the threshold across the range of scores obtained from the testing data to compute ROC curves.

C. Local Spectral Histogram

The final algorithm I implemented uses local spectral histograms [5]. The basic idea of the method is to construct a feature vector for each pixel by applying a filter bank to the image and then concatenate together local histograms from the response of each filter. The hope is that each filter will contain some small amount of information about the texture in a region, and combined together the responses of all different filters should provide good discriminatory information about local texture.

I chose a set of 7 filters based on [5]. The first is simply the intensity of the unfiltered image. I also chose 4 Gabor filters at different orientations (multiples of 45°). A Gabor filter is a complex sinusoid at a particular orientation multiplied by a Gaussian envelope. They are a very flexible filter widely used for detecting features of particular sizes and orientations. Gabor filters have many parameters to tune: the orientation of the sinusoid, the overall size of the Gaussian, and the aspect ratio of the Gaussian in the two orthogonal directions. I tuned all these parameters until I obtained filtered images that seemed by eye to maximize the difference between the foreground and other parts of the cell. I used a symmetric Gaussian with sinusoids of different wavelengths, and a overall Gaussian standard deviation of 17 pixels, chosen to roughly match the structure of the textures. The Gabor filters did a nice job of eliminating small dark regions in the background that may have been confused with foreground.

I also used two Laplacian-of-Gaussian (LoG) filters with different sized Gaussians. Both did a good job of distinguishing flat regions from variable regions, as expected, but the smaller LoG filter captured more detail in the variable regions while the large filter caused the intensity of flat regions to be brighter than the variable regions.

I then calculated local histograms from each filter. Based on [5] I chose 11 bins for my histograms. Bins were created by evenly partitioning the range of grayscale values in each filtered image. In principle this might cause problems if there are outlier pixels that are very bright or very dark which would reduce the effective contrast in the image; histogram equalization prior to binning might help this issue. I chose the same window size (31x31 pixels) for computing local histograms as for the LBP method above; the rationale is the same.

Because the window for the logal histograms was fairly large, I first calculated integral histograms for each filter bank, then concatenated these to get the full feature vector (which is 77-dimensional). To train the classifier, I again used SVM with a linear kernel on all the pixel features derived from the training image. I used 5-fold cross-validation to choose the hyperparameter and tested the classifier on the same testing image as for LBP.

III. Results

Fig. 3 shows the segmentation results in red for the testing image. Even though the testing image was very similar to the training image, the algorithms varied widely in their ability to detect the desired foreground. All of the methods were able to distinguish the dark meshwork from the speckled texture immediately surrounding it, without segmenting that area.

The thresholding method was the most successful at identifying as much of the ground truth foreground as possible (indicated by the recall score of 73%, see Table I). However, this came at the cost of also mislabeling a lot of structures, particularly in the back of the cell, as foreground.

In contrast to the thresholding method, the LBP method identified very few pixels as foreground. However, the pixels chosen by the LBP algorithm were more likely than those of the other methods to be correct (precision of 46%, Table I).

The performance of the LSH algorithm was intermediate between these two others. It identified more foreground pixels than the LBP algorithm, but not as many as the thresholding algorithm. Furthermore, it still falsely identified as foreground many of the darker pixels in the middle of the cell.

ROC curves are a useful way to measure the performance of a classification algorithm. In a ROC curve, the true positive rate and false positive rate are compared as a cutoff for classification (here the cutoff for the foreground score computed by the SVM) is varied. On this plot, a perfect algorithm would have a ROC curve that stays very close to the point where the true positive rate is 100% and the false positive rate is 0, while an algorithm producing a ROC curve that follows the $y = x$ axis is considered no better than random chance. Fig. 4 shows ROC curves for the LBP and LSH algorithms. It can be seen that these algorithms do better than chance at correctly identifying foreground pixels, but that they still have a ways to go in order to be part of a fully automated EM image segmentation pipeline.

Table 2 etc etc

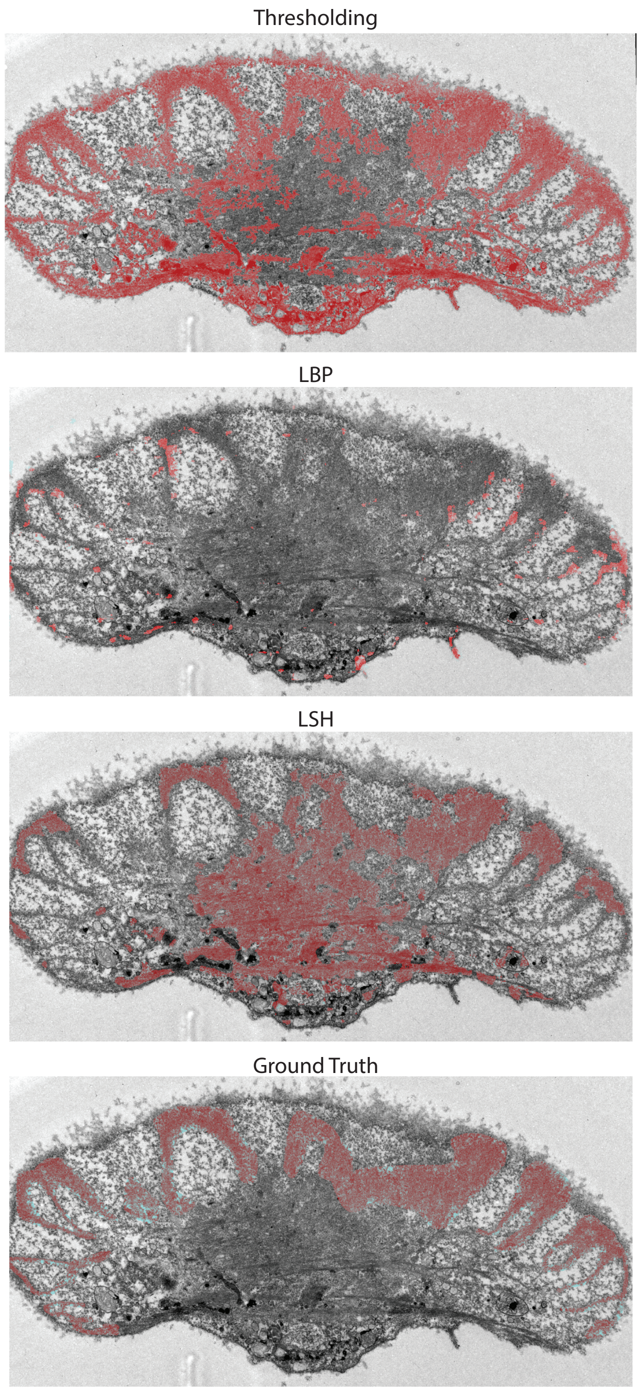

Fig. 3. Results of applying segmentation optimized on a training image to the test image shown; foreground pixels are segmented in red.*LSH*, Local Spectal Histogram method; *LBP*, Local Binary Pattern method. The ground truth for this image is also shown.

IV. Discussion

The use of texture to segment EM images presented in this report was a mixed success. While all the algorithms could distinguish the meshwork from the speckled area around it, they were not able to distinguish meshwork from functionally

Algorithm	Precision (%)	Recall (%)
Threshold	38	73
LBP	46	7
LSH	34	47

TABLE I

PRECISION-RECALL VALUES FOR THE DIFFERENT ALGORITHMS FOR SEGMENTING THE TEST IMAGE. PRECISION IS $TP/(TP+FP)$ AND RECALL IS $TP/(TP+FN)$, WHERE TP IS TRUE POSITIVES, FP IS FALSE POSITIVES, AND FN IS FALSE NEGATIVES.

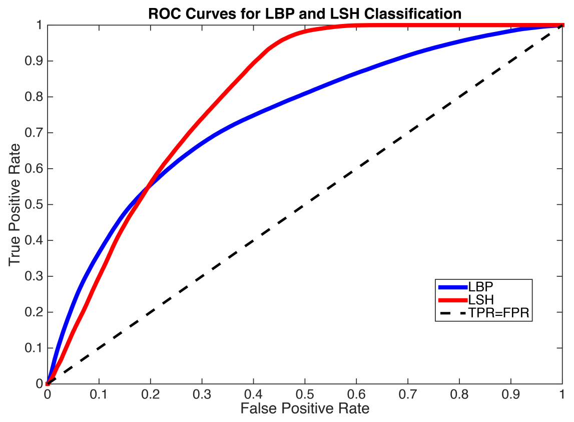

Fig. 4. ROC curves for the LBP and LSH algorithms. The black dashed line shows the expectation of purely random segmentation.

distinct but texturally similar regions at the rear of the cell. This may be a limitation of the information inherent in the image itself; while we know from biological studies that the meshwork is functionally distinct from other areas of the cell, this may not be easily resolvable through this particular imaging technique. However, it is also the case that there are methods (clear in hindsight) that could be used to improve the feature vectors and their classification, which will be described below.

A. Future Directions

There are several ways I can improve these algorithms in the future. The first and most obvious choice is to change the SVM method I used. I was not aware of the many implementations of SVM available for MATLAB that provide more options for improving the results of classification. Two things that would help would be to use non-linear kernels, such as the radial basis function (rbf) kernel, which would provide more flexibility in defining the manifold separating the foreground and background features. Another parameter I could tune in the SVM would be the tolerance to misclassification; given the close proximity of foreground and background pixels in feature space, I could raise the tolerance for misclassification in order to get

Another option for improving the robustness of the learning algorithm would be to incorporate information from more training images. While each image has many "observations" (pixels) to use for training, they are not independent; since features are calculated from sliding windows, nearby pixels will have very similar histograms. It is not clear *a priori* how this might affect the classification. I could instead choose foreground and background pixels at random from a larger set of training images. This would reduce dependence in the training set, and also expose the classifier to more variety (different illumination/contrast/acquisition settings). It would also allow me to have a more equal number of foreground and background training examples

A third option for improvement would be to normalize features in some way prior to training. Without normalization, it's possible that feature dimensions large in magnitude but small in distinction are dominating the spread in the data, leading to poor classification. The easiest way to normalize is to subtract the mean of all observations and divide by the standard deviation in each dimension.

Of course, even if I implement all these corrections, it is still possible that there just isn't enough separation of foreground and background in feature space to perform adequate classification. If this is the case, I will need to revise my feature vectors. I have far more observations than dimensions, so I could easily increase the dimensionality of the feature space by adding additional filters to the filter bank for LSH, for example. This might be required to improve separation. I could also try a better method for optimizing features for distinguishing foreground and background pixels. My current method was to inspect the filtered images by eye and tune the filter parameters to maximize visual distinction. Instead, I could try directly measuring the difference in local histograms for known foreground and background pixels using standard metrics, e.g. the χ^2 distance; I could then optimize parameters to maximize the χ^2 distance between the local histograms for known foreground and background pixels.

Realistically, the difference between some of the dark homogeneous textures in the images selected for testing and training are fairly slight; it is possible that my interpretation of the images is somewhat incorrect, and the distinction I am making in texture does not have basis in the image. If this is the case, I would have to bring in additional imaging information to help distinguish structures with similar texture in different parts of the cell. By correlating EM images with fluorescent images labeling particular components, I can identify functionally different regions that nevertheless have similar texture in the EM image. This may also greatly aid the automatic segmentation attempted here.

Once the segmentation algorithm is improved for the particular texture of interest to me, it will be interesting to see how well the feature space defined for one texture can also classify other textures. Hopefully the same space could be used to identify many different textures in the same image to efficiently access the rich information contained within EM images.

V. APPENDIX A: NOTE ON CODE

Except where explicitly stated, all code written here is my own (using MATLAB functions where appropriate). To make things easier, full functions of other people's code that I have used (the LBP set of functions from Nikolay S on Matlab Central) are in a separate folder (OtherPeoplesCode), included for completeness.

References

[1] M. C. Proença, J. F. M. Nunes, and A. P. A. de Matos, "Texture indicators for segmentation of polyomavirus particles in transmission electron microscopy images." *Microscopy and microanalysis*, vol. 19, no. 5, pp. 1170–82, 2013. [Online]. Available: http://www.ncbi.nlm.nih.gov/pubmed/23773502

[2] A. Gorai, K. Cetina, and L. Baumela, "A comparative study of local binary pattern descriptors and Gabor filter for electron microscopy image segmentation," *International Conference on Parallel, Distributed, and Grid Computing*, pp. 76–81, 2014.

[3] T. Ojala, M. Pietikäinen, and T. Mäenpää, "Multiresolution gray-scale and rotation invariant texture classification with local binary patterns," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 24, no. 7, pp. 971–987, 2002.

[4] S. Liao, M. W. K. Law, and A. C. S. Chung, "Dominant local binary patterns for texture classification." *IEEE Transactions on Image Processing*, vol. 18, no. 5, pp. 1107–1118, 2009.

[5] X. Liu and D. Wang, "Image and Texture Segmentation Using Local Spectral Histograms," *IEEE Transactions on Image Processing*, vol. 15, no. 10, pp. 3066–3077, 2006.