Image Segmentation Using Markov Random Fields: An Implementation

Jake Callahan, James Griffin, McKay Harward, Taylor Paskett, Isaac Robbins, Mingyan Zhao

April 22, 2020

Abstract

The goal of this project was to evaluate current use of Markov random fields in image segmentation and then to implement one of the more advanced methods. In this paper we discuss how we implemented an unsupervised method proposed by Barker and Rayner that uses Markov random fields to perform image segmentation. This method employs the Metropolis-Hastings algorithm to find the best parameters for the Markov random field as well as the best number of classes. In short, we coded up a hands-off method of segmenting an input image.

1 Background and Motivation

Image segmentation is one the the most important problems in machine learning today. Industries like medicine, transportation, and manufacturing benefit from the rapid advances we have made in creating image segmentation systems. One of the most interesting and valuable areas of study is unsupervised image segmentation. A powerful and accurate unsupervised image segmentation system has applications in neural photo editing using introspective adversarial networks as well as in image recognition and other computer graphics applications. In this paper, we examine and implement the method of using Markov random fields (MRFs) for unsupervised image segmentation proposed by Barker and Rayner [1].

Markov random fields for image segmentation have been studied for decades, mostly focusing on supervised learning methods. Most of this research centers around segmenting images based on pretraining a Markov random field on a class, and then estimating which pixels in a given image are most likely to be of that class. While this is useful, the more hands-off the approach, the easier it is to apply to new scenarios, as it does not require labeled data.

Unfortunately, research regarding hands-off applications is virtually nonexistent, and when we did find studies of unsupervised image segmentation methods, the required mathematics were far beyond the scope of this class. Other than Barker and Rayner's research, which was published in 2000, we could find no other papers studying the use of Markov random fields for unsupervised

segmentation. The main drawback of this unsupervised method is the tradeoff between the initial investment of creating pretrained Markov random fields and the computational expense of an algorithm using Metropolis-Hastings. However, because of the benefits this method provides–including learning not only the image classes but also how many classes to use–and because of the advances in computing power over the last 20 years, we believe that it is now valuable to revisit these methods in order to determine if applications are possible in the current world of machine learning.

2 Data

As an unsupervised method, the algorithm we outline below trains itself on each image on which it operates. As such, we didn't need (or want) a large dataset for this project. Instead, we curated a small corpus of images by hand that we believed would be useful in providing us information about how well our algorithm was training and making predictions. The first image we used was a simple, noisy grid of 16 different colored squares used by Barker and Rayner. This served as a baseline to compare our results to those achieved in the paper. We used a second simple image with only a few clear segments. We used this image to test the model as it was small and allowed us to run the model quickly. Finally, we used a few photographs to see the final results of our model.

Figure 1: The images used for training

3 Methods

Below is a simple overview of the proposed algorithm followed by the modified algorithm that we implemented. A more in depth explanations of each step, along with descriptions of the associated implementation and difficulties, follow the algorithm.

1. Randomly initialize the parameters for the MRF and number of classes

2. Get the most likely segmentation given the MRF parameters
3. Sample updated noise parameters μ and σ for each class using Metropolis-Hastings
4. Sample updated MRF parameters β_0 and β_1 for each class using Metropolis-Hastings
5. Sample number of classes
6. Repeat steps (2) - (5)

The sampling method follows the same basic pattern: sample new parameters from their prior distributions, calculate an acceptance probability (or likelihood) for those parameters, and then use the Metropolis-Hastings algorithm to accept or reject those parameters.

On the surface, this algorithm is simple. However, many of the equation definitions and notational choices used in the original paper were unclear and required extensive interpretation. Further, some of the equations as defined in the text suffered from underflow and overflow errors and thus required considerable reworking to be suitable for implementation. Sampling the number of classes was the most complex section of the paper and the notation was especially difficult to understand. Therefore, we were unable to implement this step. Because the number of classes was a fixable parameter, this was not a significant issue. Without this step, we implemented the following method.

1. Set the number of desired classes and randomly initialize the parameters for the MRF
2. Get the most likely segmentation given the MRF parameters
3. Sample updated noise parameters μ and σ for each class using Metropolis-Hastings
4. Sample updated MRF parameters β_0 and β_1 for each class using Metropolis-Hastings
5. Repeat steps (2) - (4)

3.1 Variable Definitions

We define the following variables to use in the equations that follow:

- Ω: The image being classified
- s: A pixel of the image Ω
- Y: The grayscale image intensities
- y_s: The grayscale intensity at pixel s in the interval $(0, 1]$
- X: The segmentation labels for the MRF
- x_s: The label to which pixel s is assigned

- Λ: The set of class labels
- η_s: The set of 4 direct neighbors of pixel s
- Ψ: The parameter vector consisting of both noise and MRF parameters
- Θ_c: The noise model parameter vector consisting of μ_c and σ_c
- $\beta_c^{(0)}$: The external field parameter
- $\beta_c^{(1)}$: The inter-pixel interaction strength parameter
- T_t: The annealing temperature at iteration t
- n_c: The number of pixels of class c
- $n_{(c,\eta)}$: Abusive notation explained in Section 3.3

3.2 Segmenting the Image with the MRF

Given an initial set of random parameters to define our MRF, we needed to first find the probability of each pixel being in each class:

$$p\left(x_s = c | y_s, \eta_s, \Psi_c\right) \propto \frac{1}{\sqrt{2\pi\sigma_c^2 T_t}} \exp\left\{-\frac{1}{T_t}\left[\frac{1}{2}\left(\frac{y_s - \mu_c}{\sigma_c}\right)^2 + \left(\beta_c^{(0)} + \beta^{(1)} V(c, \eta_s)\right)\right]\right\} \tag{1}$$

where $V(c,\eta) = \frac{1}{4}\sum_{t\in\eta}(c \oplus x_t)$, where $\oplus$ is an operator defined to take the value -1 if its arguments are equal and 1 otherwise.

We then calculated the probabilities over the entire image for each class and took the argmax, thereby assigning each pixel to the class to which it most likely belongs. The implementation of this process was relatively straightforward and required no adjustments to the equations.

3.3 Updating Noise Parameters

The noise parameters μ_c and σ_c are sampled using Metropolis-Hastings with the following likelihood:

$$\begin{aligned} p\left(\mu_c, \sigma_c | Y, X\right) &\propto \prod_{s:x_s=c} p\left(y_s | \mu_c, \sigma_c\right) p_r\left(\mu_c\right) p_r\left(\sigma_c\right) \\ &= \frac{1}{\sigma_c \left(2\pi\sigma_c^2 T_t\right)^{n_c}} \exp\left\{-\frac{1}{2T_t}\sum_{s:a_{s=c}}\left(\frac{y_s - \mu_c}{\sigma_c}\right)^2\right\} \end{aligned} \tag{2}$$

With a firm grasp on Metropolis-Hastings, we expected updating the noise parameters to be straightforward. However, the suggested T_t from the paper was so large that we ran into immediate underflow errors during computation. We soon realized the issue was the n_c exponent in the

denominator causing our underflow. While segmenting a small 100 x 100 image into 4 classes, n_c should be about 25,000 and possibly much larger. This necessitated that we evaluate the likelihood in a completely different form which we give below.

$$p\left(\mu_c, \sigma_c | Y, X\right) \propto \frac{1}{\sigma_c} \prod_{s:x_s=c} \frac{1}{2\pi\sigma_c^2 T_t} \exp\left\{-\frac{1}{2T_t}\left(\frac{y_s - \mu_c}{\sigma_c}\right)^2\right\} \tag{3}$$

This allowed us to avoid the exponent and the large sum, both of which could cause massive underflow by calculating each part of the product individually. Because this probability would be used only for the acceptance ratio a in the Metropolis-Hastings algorithm, we extended this calculation one step further.

$$\begin{aligned} a &= \frac{p\left(\mu_c^{new}, \sigma_c^{new} | Y, X\right)}{p\left(\mu_c, \sigma_c | Y, X\right)} \\ &\propto \prod_{s:x_s=c} \frac{\frac{1}{2\pi\sigma_c^{new2} T_t} \exp\left\{-\frac{1}{2T_t}\left(\frac{y_s - \mu_c^{new}}{\sigma_c^{new}}\right)^2\right\}}{\frac{1}{2\pi\sigma_c^2 T_t} \exp\left\{-\frac{1}{2T_t}\left(\frac{y_s - \mu_c}{\sigma_c}\right)^2\right\}} \end{aligned} \tag{4}$$

This allowed us to calculate the acceptance ratio for a single pixel before taking the product over pixels of a given class. By doing so, we completely avoided underflow. However, we were only getting acceptance ratios of 0 and ∞. At first this concerned us but we realized that given model parameters, the individual ratios would be very similar (at least across each class) and thus for $a_s > 1$ taking the product would approach ∞ and for $a_s < 1$, the product would approach 0. Having resolved this issue, we were able to update our noise parameters μ_c and σ_c properly.

3.4 Updating MRF Parameters

As in the previous step, the MRF parameters $\beta_c^{(0)}$ are sampled using Metropolis-Hastings with the following probability:

$$\begin{aligned} &p\left(\beta_c^{(0)}, c \in \Lambda | X\right) \\ &= p\left(X | \beta_c^{(0)}, c \in \Lambda\right) p_r\left(\beta_c^{(0)}, c \in \Lambda\right) \\ &= \prod_{(c\in\Lambda,\eta)} \left(\frac{\exp\left(-\frac{1}{T_t}\left[\beta_c^{(0)} + \beta^{(1)} V(c,\eta)\right]\right)}{\sum_{i\in\Lambda} \exp\left(-\frac{1}{T_t}\left[\beta_i^{(0)} + \beta^{(1)} V(i,\eta)\right]\right)}\right)^{n_{(c,\eta)}} \\ &= \prod_{(c\in\Lambda)} \left(\frac{\exp\left(-\frac{1}{T_t}\left[\beta_c^{(0)}\right]\right)}{\sum_{i\in\Lambda} \exp\left(-\frac{1}{T_t}\left[\beta_i^{(0)}\right]\right)}\right)^{n_c} \\ &\times \prod_{(c\in\Lambda,\Psi\eta)} \left(\frac{\exp\left(-\frac{1}{T_t}\left[\beta^{(1)} V(c,\eta)\right]\right) \sum_{i\in\Lambda} \exp\left(-\frac{1}{T_t}\left[\beta_i^{(0)}\right]\right)}{\sum_{i\in\Lambda} \exp\left(-\frac{1}{T_t}\left[\beta_i^{(0)} + \beta^{(1)} V(i,\eta)\right]\right.}\right)^{n_{(c,\eta)}} \end{aligned} \tag{5}$$

We immediately realized that definition of the exponent $n_{(c,\eta)}$ was deeply flawed. The paper defines this as $n_{(c,\eta)} = \#\left(s : x_s = c, \eta_s = \eta\right)$; this notation is abusive and does not follow the paper's

previously established notational conventions. As written, this n function takes in a class c and a set of pixels η, then returns the number of pixels that have label c and neighbors η. However, due to the way η is defined at the beginning of the paper, each pixel will have a unique set of neighbors.

Thus, this n function will return a 1 if there is a pixel in class c with the given set of neighbors η and a 0 if there is not a pixel in class c with that exact set of neighbors. Then for each class, iterating over all η_s with the n function is equivalent to iterating over all $s : x_s = c$ with the indicator function $\mathbb{1}(s, c)_{x_s=c}$. This leads to the exponents on terms corresponding to $s : x_s \neq c$ being set to 0 and excluded from the product. We remedied this equation as follows:

$$p\left(\beta_c^{(0)}, c \in \Lambda | X\right) = \prod_{s:x_s=c} \frac{\exp\left(-\frac{1}{T_t}\left[\beta_c^{(0)} + \beta^{(1)} V(c, \eta_s)\right]\right)}{\sum_{i\in\Lambda} \exp\left(-\frac{1}{T_t}\left[\beta_i^{(0)} + \beta^{(1)} V(i, \eta_s)\right]\right)} \tag{6}$$

This greatly simplified the calculations for updating the MRF parameters $\beta_c^{(0)}$. We used the same method as described in Section 3.2 to calculate each of the individual acceptance ratios a_s before taking the product over all the ratios. A similar equation is proposed for the probability $p(\beta_c^{(1)}, c \in \Lambda | X)$. However, the paper states that the posterior density calculation for $\beta_c^{(1)}$ will not be proper under certain underlying label map configurations. The paper fixes a value *a priori* of 1.5 for $\beta_c^{(1)}$. We follow this practice in our approach.

3.5 Sampling the Number of Classes

The most difficult part of this paper involved sampling the number of classes. The notation throughout this section was very difficult to understand. Below is the equation for the acceptance ratio for splitting a class c into c_1 and c_2.

$$\begin{aligned} &\frac{p\left(X = x^+, \Psi^+, k^+ | Y = y\right)}{p(X = x, \Psi, k | Y = y)} \frac{1}{p_\beta\left(u_1\right) p_\beta\left(u_2\right) p_\beta\left(u_3\right)} \\ &\quad \times \frac{\partial\left(\Psi_{c1}, \Psi_{c2}\right)}{p(\text{segmentation})} \left| \frac{1}{\partial\left(\Psi_c, u_1, u_2, u_3\right)} \right| \end{aligned} \tag{7}$$

Unfortunately, we were not able to calculate this acceptance ratio. The problem arose from the approximation of the first term, which is calculated via

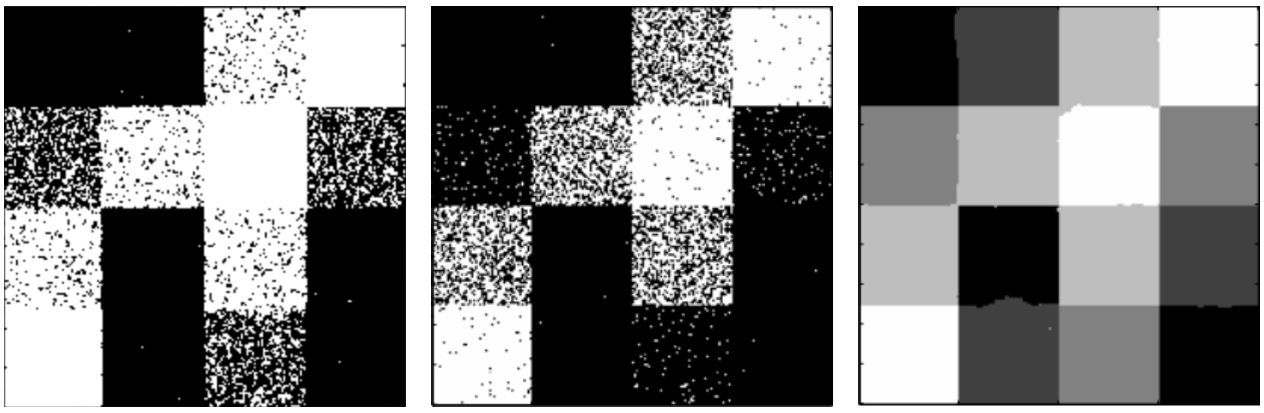
$$\begin{aligned} p(X = x, \Psi, k | Y = y) \propto &\prod_{s\in\Omega} \frac{1}{\sqrt{2\pi\sigma_{x_s}^2}} \exp\left\{-\frac{1}{2}\left(\frac{y_s - \mu_{x_s}}{\sigma_{x_s}}\right)^2\right\} \\ &\times \prod_{s\in\Omega} \frac{\exp\left\{-\left(\beta_{x_s}^{(0)} + \beta^{(1)} V(x_s, \eta_s)\right)\right\}}{\sum_{c\in\Lambda} \exp\left\{-(\beta_c^{(0)} + \beta^{(1)} V(c, \eta_s)\right\}} \\ &\times p_r\left(\beta^{(1)}\right) p_r(k) \prod_{c\in\Lambda} p_r(\mu_c) p_r(\sigma_c) p_r(\beta_c^{(0)}). \end{aligned} \tag{8}$$

When calculating this term, the product over all $s \in \Omega$ creates problems. In a relatively small 100 x 100 pixel image, this is a product over 10,000 terms. This almost always leads to numerical overflow (i.e. $p(X = x, \Psi, k|Y = y) = \pm\infty$) or underflow (i.e. $p(X = x, \Psi, k|Y = y) = 0$). Overflow can be dealt with in certain situations; sometimes we can just leave it be, especially when computing acceptance probabilities, where anything greater than 1 is just taken to be 1. However, in this case, substituting either $\pm\infty$ or 0 into (7) gives $\frac{\infty}{\infty}$ or $\frac{0}{0}$ in the first term, both of which are undefined.

Because of this, we chose not to sample the number of classes in our algorithm. Fortunately, this step only helps find the optimal number of classes for the segmentation and omitting it should not affect the other steps or performance of the algorithm as a whole. With this step omitted from the algorithm, we simply specify the desired number of classes before running it.

4 Results

Out modified image segmentation algorithm was largely successful. We initially tested our model with the noisy squares image from Figure 1. The authors used this image in their paper and achieved great results in segmenting the image. With our model, we noticed that the results would converge to segmentations of only 2 or 3 classes, even when we had specified 5 or more classes. Additionally, certain squares were properly segmented while other squares were not, the noise in the squares indicating that the model was uncertain. This resulted in some squares being solid, where other similar squares were different combinations of the 2 or 3 classes the model was identifying. Below are the results that we were able to achieve compared to the authors' results in the paper. As shown in Figure 2, we were unable to get our model to produce the final smooth result that the paper achieved.



Our result | Authors' intermediate result | authors' final result

Figure 2: Results comparison for squares image

The similarity of our result and the authors' intermediate result is striking and suggests that our model is very close to correct. However, our model never showed signs of smoothing out, even with thousands of iterations.

The results of the other images we used are shown in Figure 3. Again, our model was able to identify major classes of pixels and segment them. However, our model mostly failed when it came to smoothing classes and removing noise. This can be seen in both of the examples below.

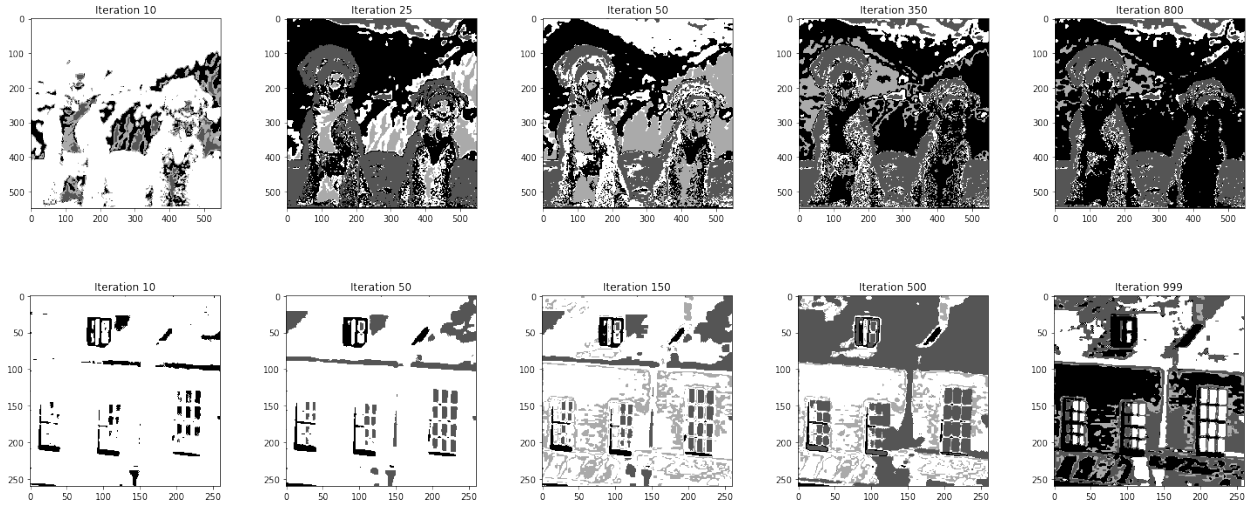


Figure 3: The images used for training

5 Comparison to Other Methods

As part of our analysis, we wanted to compare our model to classical machine learning image segmentation approaches. For this we used the K-means algorithm and Scikit-learn's pixel threshold library which segments images based on pixel brightness. Because we typically used 4 classes to test our model, we used 4 classes in both our K-means and pixel thresholding test. Additionally, we chose to do a 2-class thresholding test using the mean pixel brightness. The results of these comparisons are shown in Figure 4.

6 Analysis

Our model turned out to be fairly effective. It was able to initially identify different class segmentations and improve them, although the improvement was slight. On the square images, our model seemed to work exactly as the authors' models did, achieving very similar results. However, we were

K-means with 4 clusters | 2-class pixel thresholding | 4-class pixel thresholding | Model results after 800 iterations

Figure 4: Comparison to other methods

unable to tune our model so that it would smooth out each cluster and achieve a result similar to the authors' final result. This was the main issue with our model in all cases.

We saw the same results with the building image used in the paper (see Figure 3), but again, our results never smoothed out like theirs did. This is not to say that our model failed completely to smooth out classes. When comparing our model side by side with K-means and thresholding techniques, we noticed that our results were much closer to the results from those methods. It is hard to say whether our model was able to smooth these boundaries better than the other methods, especially because although the classes are very similar across methods, they are represented with different colors making it hard to compare. In the 4-class pixel thresholding image, we noticed that the labels for the dog on the left are split over its face (Figure 4) even though the pixels were very close in intensity. Our model, like the K-means, was able to adjust the classes to account for more similar pixels instead of a more uniform partitioning.

Perhaps the most notable result is our model's black labeling of the dog on the right in Figure 4. Our model has identified several clusters and grouped them together, something that none of the other models were able to do. Even then, our model seems to be mostly classifying using 2 classes, and is most similar to the 2-class pixel thresholding result. This is similar to the results we saw when classifying the squares image.

7 Conclusion

Although our model did work, it seemed to represent striking similarities to other rudimentary techniques like K-means and pixel thresholding. While this is not an issue, we expected that after more iterations, the classes would smooth out and we would see less noise and better groups of classes in our segmentations. Initially we achieved results almost identical to those of the authors. However, with more training, their results seemed to improve to what we would have expected,

while our model did not. This was most likely due to poor choices for our parameters, especially those defining the prior distributions from which other parameters were drawn. Without a stronger understanding of what these parameters were doing, it was hard to know why we were getting poor results. This could also be a result of not having implemented the full method as was done in the paper, although to us this seems less likely than having poor parameters.

Implementing the method for sampling the number of classes in each iteration does not seem feasible given only the information provided in the paper. Thus, for future work to improve this model, a major grid search over many different images could be implemented to try and identify the best parameters for the model. This would require a vast amount of computing power, as there are at least 5 different parameters that are arbitrarily chosen.

Had our model been able to smooth out classes as we expected (see Figure 2), its performance likely would have been on par with current state-of-the-art methods. In its current form, however, it lacks the ability to smooth and denoise images to make it competitive with current practices.

Overall, we are very happy with the work we were able to do on this project, even though the results were somewhat disappointing. Just getting the model to work properly was a great achievement for us as there was a lot of work required to both understand and then implement the equations and methods explained in the paper.

References

[1] S.A. Barker and P.J.W. Rayner. "Unsupervised image segmentation using Markov random field models". In: *Pattern Recognition* 33.4 (2000), pp. 587–602. ISSN: 0031-3203. DOI: https://doi.org/10.1016/S0031-3203(99)00074-6. URL: http://www.sciencedirect.com/science/article/pii/S0031320399000746.