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, then to implement one of the more advanced methods. In this paper we discuss how we implemented the equations from a paper based on an unsupervised method that used 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

This paper was motivated by the desire for a hands-off unsupervised image segmentation algorithm. Our primary motivation for wanting a hands-off image segmenter had to do with applications in neural photo editing using introspective adversarial networks as well as image recognition. The idea for using Markov random fields (MRF) for image segmentation is not new, in fact, the paper we decided to implement was written in 2000, but the unsupervised method doesn't seem to have any study since then. The background research we conducted only returned methods for segmenting images based on pretraining an MRF on a class, and then estimating which pixels in a given image are most likely to be of that class. This included training the model to recognize trees and foilage, or roads using labeled images and then segmenting a new image based on the previous learned classes. While this is useful, the hands-off approach can work in far more applications because it does not require labeled data. The drawback of an unsupervised method is that it trades an initial investment of creating a pretrained MRF for a computationally expensive algorithm using Metropolis Hastings to update the MRF parameters.

We investigated numerous methods for performing unsupervised image segmentation, however, ran into many road blocks with many different papers. The difficulty of these methods truly shocked us as we were unable to make any progress with numerous different papers. This paper, although older and full of poor notation, at least provided a simple overview of the algorithm along with the equations necessary to implement it. Anothe benefit of using the method described in this paper is that this paper does not require the user to specify the number of classes or segments. Rather, the final step of this iterative algorithm samples the best number of classes using Bayesian statistics.

2 Data

The image segmentation algorithm that we implemented in this paper is an unsupervised method. This means that the only data involved is the image that is segmented. We use a few different images, the first being a simple noisy grid of 16 different colored squares used in the paper. This served as a bit of a baseline to compare our results to those achieved in the paper. We used a second simple image with only a few clear segments. This image was used largely for testing 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.

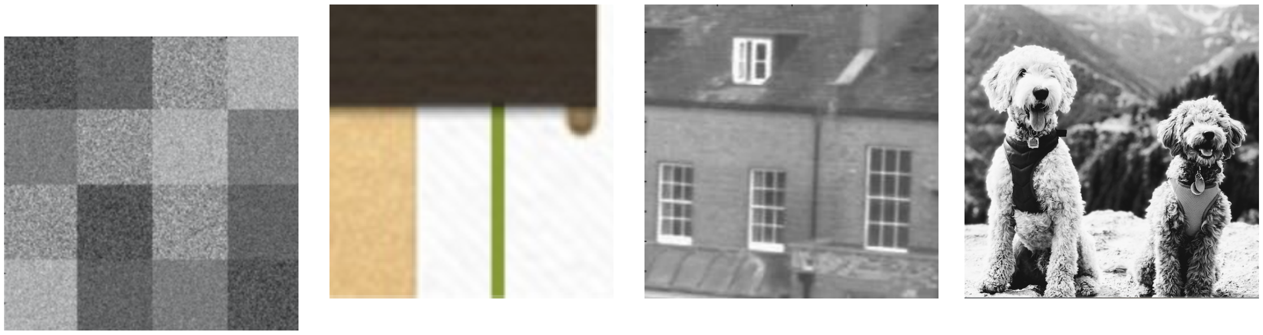


Figure 1: The images used for training

3 Methods

Below is a simple overview of the algorithm proposed and then the modified algorithm we implemented. A more in depth explanations of each step along with descriptions of how we implemented them and the difficulties we encountered at each step follows the algorithms.

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)

Thought the process is simple, each of these steps required extensive coding and interpretation of the equations described in the paper. Many of the equations were especially vague and incldued very poor and inconsistent notation which further complicated the already difficult equations. Sampling the number of classes was the most complex section of the paper and the notation was especially difficult to understand. We were unable to implement this step, however, this was not a major issue, because this only affected how many classes were being used and was a parameter we could fix. Leaving this out, we ended up implementing 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)

Here is an overview of the variables used in the equations:

- 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 class to which pixel s is assigned
- Λ: The set of class labels
- η_s: The set of 4 direct neighbors of cell 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

3.1 Segment the Image with the MRF

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

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, otherwise, +1.

With an initial set of random parameters to define our MRF, we needed to first find the probability of each pixel being in each class. This was relatively simple to compute once the equation had been coded. 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.

3.2 Update 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, 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 calculate the equation 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 a 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.

$$\begin{aligned} a &= \frac{p\left(\mu_c^{new}, \sigma_c^{new} | Y, X\right)}{p\left(\mu_c, \sigma_c | Y, X\right)} \\ &\propto \prod_{s:x_s=c} \frac{\frac{1}{2\pi\sigma_c^{new2} T_t} \exp\left\{-\frac{1}{2T_t}\left(\frac{y_s-\mu_c^{new}}{\sigma_c^{new}}\right)^2\right\}}{\frac{1}{2\pi\sigma_c^2 T_{t-1}} \exp\left\{-\frac{1}{2T_{t-1}}\left(\frac{y_s-\mu_c}{\sigma_c}\right)^2\right\}} \\ &= \prod_{s:x_s=c} \left(\frac{\sigma_c}{\sigma_c^{new}}\right)^3 \left(\frac{T_{t-1}}{T_t}\right) \exp\left\{\frac{-1}{2T_t}\left(\frac{y_s-\mu_c^{new}}{\sigma_c^{new}}\right)^2 + \frac{1}{2T_{t-1}}\left(\frac{y_s-\mu_c}{\sigma_c}\right)^2\right\} \end{aligned} \tag{4}$$

This allowed us to calculate the acceptance ratio for a single pixel before taking the product over all of the pixels of the specified 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 these individual ratios would be very similar, at least across each class, 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.

3.3 Update MRF Parameters

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

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

We immediately ran into the issue of the definition of the exponent $n_{(c,\eta)}$ that is used. 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 what had been previously done. After much deliberation, we realized that this was an extremely convoluted way of iterating over each $s : x_s = c$ as they had done in all previous equations. For no apparent reason they were iterating over η with a counting function that returned $n_{(c,\eta)} = 0$ if $x_s \neq c$ and $n_{(c,\eta)} = 1$ if $x_s = c$. Thus the probabilities corresponding to pixels such that $x_s \neq c$ were set to 1 and essentially 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 of calculating each of the individual acceptance ratios a_s before taking the product over all the ratios. A similar equation is proposed for the likelihod $p(\beta_c^{(1)}, c \in \Lambda|X)$, however, the paper states that the posterior density alculation for $\beta_c^{(1)}$ will not be proper under particular 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.

3.4 Number of classes

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

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

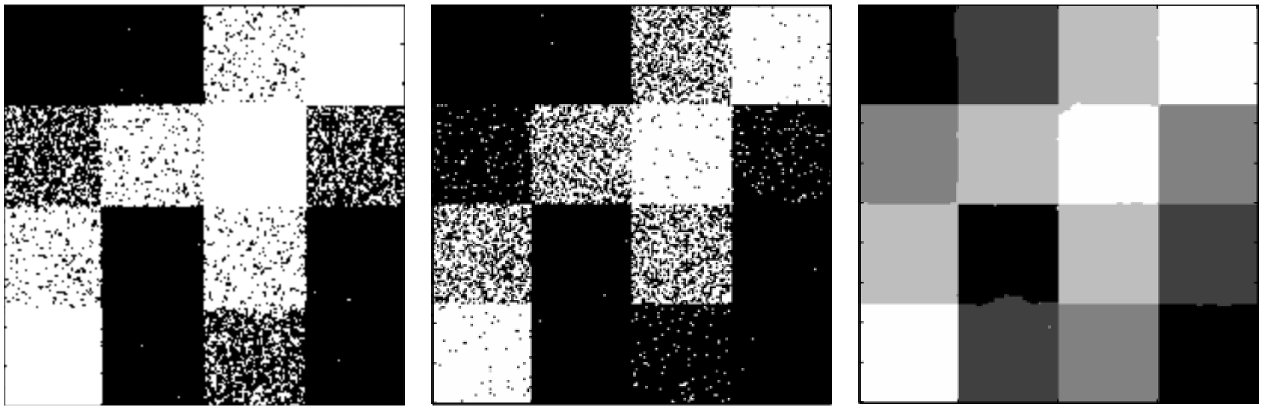
Unfortunately, we were never able to work out the notation in a way that made enough sense to even try and implement this sampling. However, this step only helps find the optimal number of classes for the segmentation at each step and emmitting this step should not affect the other steps or performance 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

We were really shocked that we were able to successfully implement this images segmentation from the paper. We were not able to achieve the same results as the authors of the paper. This could be due to our lack of understanding as to what how the different parameters interact and affect the model. This could be a major factor in determining the variances for the distributions from which the proposed μ_c and σ_c are drawn.

We intially 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. Our model, however, struggled to identify the classes in the same way that the author's method did. This could be due to not having implemented the part step that resamples the number of classes, but we speculate that this has more to do with unknown parameters that we have set that may not be ideal. With this image, we noticed that the segmentations would converge to segmentations with only 2 or 3 classes, even when we had specified 5 or more classes, where certain squares were properly segmented but other squares the model seemed to be 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 authros' results in the paper. We were unable to get our model to produce the final smooth result that the paper achieved as shown in Figure 2. This was the desired output of our model, but unfortunately we were unable to achieve this result.



Our result Author's intermediary result Author's final result

Figure 2: Results comparison for squares image

The similarity of our result and their intermediary result is striking and suggests that our model is very close to working, however, our model never showed signs of smoothing out, even with thousands of iterations.

As for the other images we used, this is what the results looked like. 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.

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 K-means and pixel threshold testing. Throughout our model, we typically used 4 classes from which our model sampled, thus for our K-means we used 4 classes. Here is the K-means segmented output compared to our MRF output.

We also wanted to do some other comparisons, so we utilized the scikit learn image segmentation library to divide segment the imagebased on thresholds. To do this, we first chose to do a 2-class threshold using the mean pixel brightness. All of the pixels brighter than the mean were assigned to one cluster and the rest to the other. We then tried thresholding with 4 classes by assigning labels based on quartiles of pixel brightness. The bottom 25% of pixels belonged to the first class and so on.

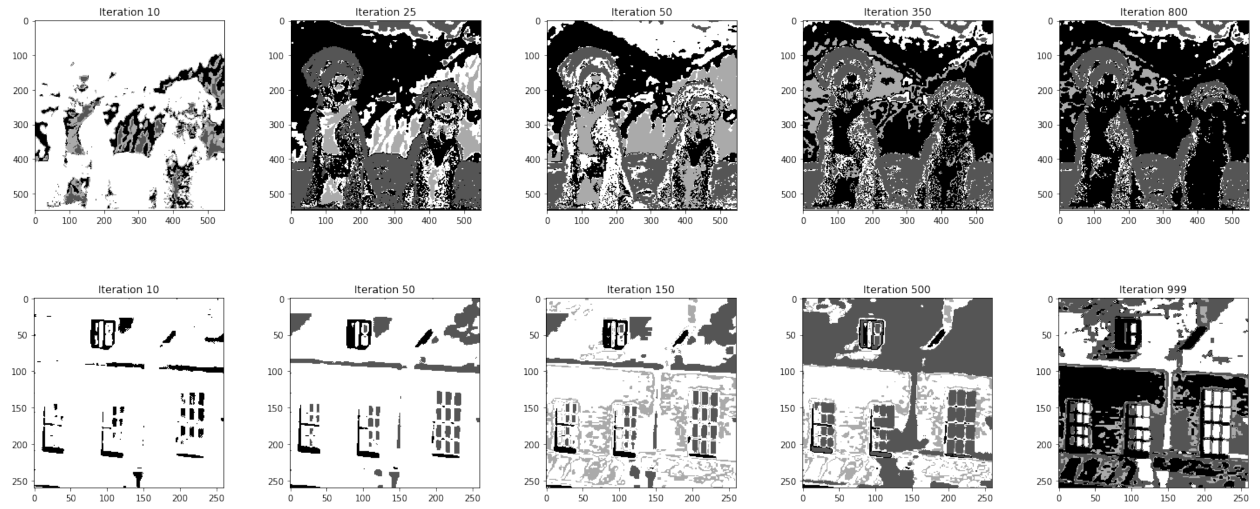
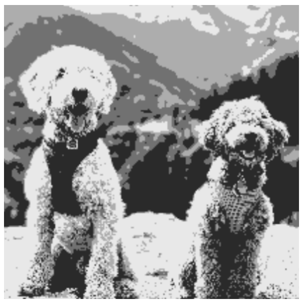
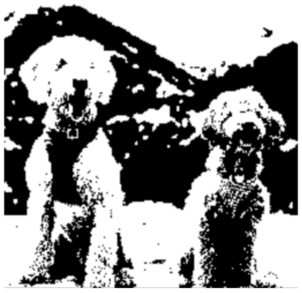


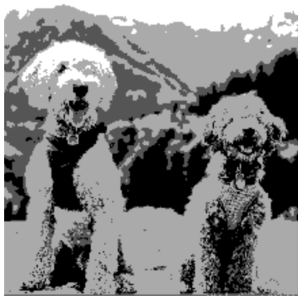
Figure 3: The images used for training



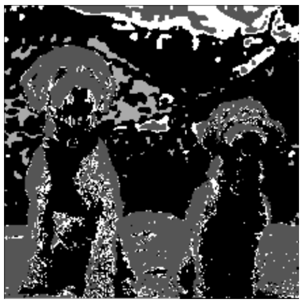
K-means with 4 clusters



2-class pixel thresholding



4-class pixel thresholding



Model results after 800 iterations

Figure 4: Comparison to other methods

6 Analysis

Our methods turned out to be fairly effective. It was able to identify different classes initially and improve them, although it was only a slight improvement. Our model cannot compete with current state of the art methods. However, had our model been able to smooth out classes as we expected it to (see Figure 2), its performance likely would have been on par with newer state of the art methods.

The results we achieved with the squares image reflected exactly the results that the authors had achieved in the paper. Our model seemed to work exactly as theirs had. However, we were unable to tune our model so that it would smooth out each cluster and achieve a result similar to the Author's final result. This was the main issue with our model in all cases.

We saw the same results with the image of the side of a building that was also 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 these 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 among methods, they are represented with different colors making it hard to compare them. In the 4-class pixel thresholding image, we notice that the labels for the dog on the left are split over his face (see 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 is the black labeling of the dog on the right from our model. Our model has grouped together a few clusters and grouped them together, something that none of the other models were able to do. Even then, our model seems to be mostly classified into 2 classes, and is most similar to the 2-class pixel thresholding result, which is similar to the results we were seeing 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. The authors of the paper we used achieved near exact results to those that we found, however, with enough 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 dristributions 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 this seems less likely to us 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.

Overall, we are very happy with the work we were able to do on this project, even though the results are not amazing. 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.