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.
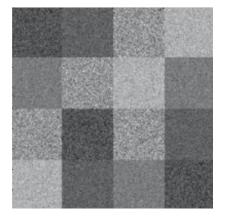

Figure 1: The images used for training

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 a 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:

$$p\left(\mu_c, \sigma_c | Y, X\right) \propto \prod_{s:x_s=c} p\left(y_s | \mu_c, \sigma_c\right) p_r\left(\mu_c\right) p_r\left(\sigma_c\right)$$

$$= \frac{1}{\sigma_c \left(2\pi\sigma_c^2 T_t\right)^{n_c}} \exp\left\{-\frac{1}{2T_t}\sum_{s:a_s=c}\left(\frac{y_s - \mu_c}{\sigma_c}\right)^2\right\} \tag{2}$$

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.

$$a = \frac{p\left(\mu_c^{new}, \sigma_c^{new} | Y, X\right)}{p\left(\mu_c, \sigma_c | Y, X\right)}$$

$$\propto \prod_{s:x_s=c} \frac{\frac{1}{2\pi\sigma_c^{new2}T_t} \exp\left\{-\frac{1}{2T_t}\left(\frac{y_s - \mu_c^{new}}{\sigma_c^{new}}\right)^2\right\}}{\frac{1}{2\pi\sigma_c^2 T_t} \exp\left\{-\frac{1}{2T_t}\left(\frac{y_s - \mu_c}{\sigma_c}\right)^2\right\}} \tag{4}$$

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:

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

We immediately 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.

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

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

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

When calculating this term, the products over all $s \in \Omega$ create problems. In a relatively small 100 x 100 pixel image, this is a product over 10,000 terms. So, 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 at each step and ommitting 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

5   Analysis

Our methods turned out to be fairly effective. They don't par with current state of the art technologies but they definitely do well for something done by a few undergraduate students in a single project, not research. The model we've created does a fairly decent job of segmenting images on par with what the original authors achieved, though sometimes in a few extra iterations. Looking at how it works, I do think it would work in a GAN for neural photo editing to maybe train the GAN to focus itself more closely on one area while keeping the rest of the image the same, but even with faster computers, it seems it still takes too long for any practical applications. Given enough computing power it can be used.

6   Conclusion

The initial intent of getting a completely unsupervised method of image segmentation using Markov Random Fields was successful and could probably be improved upon for efficiency. If given the opportunity to try image segmentation again, I would probably opt to use a neural network as they tend to require an initial investment of training time but can then be used to segment an image very quickly. Something we could do is have more informed priors for each parameters. We could also implement for color images (may be?)