CS 283 Final Project: Image Smoothing While Preserving Edges

James Gui

December 11, 2017

Figure 1: Smoothing result of L0 gradient minimization

1 Introduction

In visual imagery, edges create the overall structure and basic building blocks of what we see. Other features like texture and color provide context, but for many vision tasks–like image recognition and segmentation–edges are the visual morphemes for analyzing an image. Thus, a great deal of research has gone into methods of processing images to enhance edges and extract edge information.

In particular, edge-preserving smoothing has many applications in image processing–from enhancing the sharpness of images to cleaning up noise before edge detection. Basic smoothing of this kind can be done through bilateral filtering [TM98]. However, such filtering can still blur salient edges, since it is a local averaging operation.

Xu et al introduced a novel method for image smoothing that relied on solving an optimization problem globally rather than using local operations.[XLXJ11] The key to their formulation was the L_0 norm (number of nonzero values) of the gradient–by constraining this value, they were able to keep salient edges fully intact while achieving a smoothing effect. This method of smoothing contrasts with typical local averaging methods and instead uses a discrete counting metric. The overall effect creates clean, visually distinct edges while maintaining overall color and edge location. For conciseness, I will use the term "we" to refer to Xu et al and am in no way claiming credit for the ideas.

2 Methods

Let S be the smoothing result and I be the input image. To "smooth" the image, the number of non-zero pixel value differences is constrained. The function that counts this number, $C(S)$, is defined as $C(S) = \{\#p|\partial_x S_p| + |\partial_y S_p| \neq 0\}$. In this equation, the ∂ operator is equivalent to the `diff` function in MatLab, or the forward difference of an array.[1] We can think of $C(S)$ as the L_0 norm of the forward differences. The objective function is then

$$\min_S \sum_p (S_p - I_p)^2 \tag{1}$$

with the constraint $C(S) = k$ for some number of pixels k. To generalize this function for many images, whose k values can range widely, we introduce a smoothing parameter λ to the cost function, changing the objective to

$$\min_S \left\{ \sum_p (S_p - I_p)^2 + \lambda \cdot C(S) \right\} \tag{2}$$

A large λ increases the effect that $C(S)$ has on the output, reducing the number of edges in the optimal solution by punishing high numbers of edges. As a result, higher λ gives a smoother solution.

This objective is not easy to solve, however, so we introduce auxiliary variables h, v, solving for them alternately with S to achieve an approximate solution. h corresponds to an estimate of $\partial_x S$, and v corresponds to an estimate of $\partial_y S$. The new objective is

$$\min_{S,h,v} \left\{ \sum_p (S_p - I_p)^2 + \lambda C(h,v) + \beta((\partial_x S_p - h_p)^2 + (\partial_y S - v_p)^2) \right\} \tag{3}$$

In this function, $C(h,v) = \{\#p||h_p| + |v_p| \neq 0\}$, analogous to the L_0 norm of the estimated gradient. By fixing h, v and S alternately, we can find closed-form solutions. β is a parameter controlling for similarity between h, v and the gradients $\partial_x S, \partial_y S$ respectively. In our algorithm, we automatically update β, and as β increases Equation (3) approaches Equation (2).

2.1 Computing S

After fixing h, v, we need only consider the terms that involve S. From Equation 3, we get

$$\min_S \left\{ \sum_p (S_p - I_p)^2 + \beta((\partial_x S_p - h_p)^2 + (\partial_y S - v_p)^2) \right\}, \tag{4}$$

We can solve this function by simply taking the derivative and setting it equal to zero. We have

$$S(1 + \beta(\partial_x^T \partial_x + \partial_y^T \partial_y)) = I + \beta(\partial_x^T h + \partial_y^T v) \tag{5}$$

Solving this matrix equation requires a large matrix inversion, so we instead note that the ∂ operators are actually circulant matrices and can be diagonalized by the Fourier transform. So by

[1] Note that to handle boundary conditions, I padded the image matrix by repeating the values from the boundary instead of with zeros. This gives a better estimate of the gradient at the boundary. Xu et al may have handled the boundary differently.

computing the solution in the Fourier domain, we can rely on component-wise division rather than matrix inversion and therefore increase efficiency from $\mathcal{O}(n^2)$ to $\mathcal{O}(n \log n)$. Thus, we have

$$S = \mathcal{F}^{-1}\left(\frac{\mathcal{F}(I) + \beta(\mathcal{F}(\partial_x)^*\mathcal{F}(h) + \mathcal{F}(\partial_y)^*\mathcal{F}(v))}{\mathcal{F}(1) + \beta(\mathcal{F}(\partial_x)^*\mathcal{F}(\partial_x) + \mathcal{F}(\partial_y)^*\mathcal{F}(\partial_y))}\right) \tag{6}$$

where $\mathcal{F}$ is the Fourier transform operator and all multiplications and divisions are component-wise.

2.2 Computing h and v

Looking at the terms involving h, v, the objective function becomes

$$\min_{h,v}\left\{\sum_p (\partial_x S_p - h_p)^2 + (\partial_y S_p - v_p)^2 + \frac{\lambda}{\beta}C(h,v)\right\} \tag{7}$$

This function reaches its minimum at the following conditions:

$$(h_p, v_p) = \begin{cases} (0,0) & (\partial_x S_p)^2 + (\partial_y S_p)^2 \leq \lambda/\beta \\ (\partial_y S_p, \partial_y S_p) & \text{otherwise} \end{cases} \tag{8}$$

A relatively simple proof of this can be found in the paper, but it is omitted for clarity. Now we can write the full algorithm:

Algorithm 1 L0 gradient minimization algorithm

procedure L0MIN($I, \lambda, \beta_0, \beta_{\max}, \kappa$)
  $S \leftarrow I$
  $\beta \leftarrow \beta_0$
  while $\beta \leq \beta_{\max}$ **do**
    $h \leftarrow \partial_x S$ ▷ Using S, solve for h and v
    $v \leftarrow \partial_y S$
    if $(\partial_x S_p)^2 + (\partial_y S_p)^2 \leq \lambda/\beta$ **then**
      $h_p \leftarrow 0, v_p \leftarrow 0$
    $S \leftarrow \mathcal{F}^{-1}\left(\frac{\mathcal{F}(I)+\beta(\mathcal{F}(\partial_x)^*\mathcal{F}(h)+\mathcal{F}(\partial_y)^*\mathcal{F}(v))}{\mathcal{F}(1)+\beta(\mathcal{F}(\partial_x)^*\mathcal{F}(\partial_x)+\mathcal{F}(\partial_y)^*\mathcal{F}(\partial_y))}\right)$ ▷ Using h and v, solve for S
    $\beta \leftarrow \kappa\beta$
  return S ▷ return smoothed image S

3 Results

My results were very similar to those outlined in the paper. I used MatLab to implement the main algorithm outlined in the paper. I hard-coded their recommended parameter initializations ($\beta_0 = 2\lambda$, $\beta_{\max} = 1E5$, $\kappa = 2$). I found that I could apply the algorithm to remove artifacts from text-based Internet "memes" in addition to clip art. Below are a few examples on pictures I've taken as well as on Internet memes:

Figure 2: Before and after. This 5616x3744 image took about 200 seconds to process.

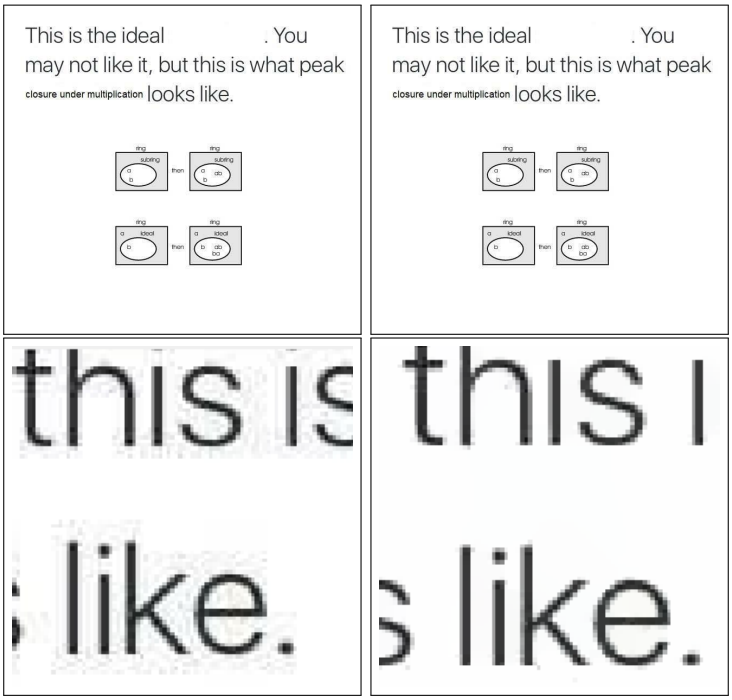


Figure 3: Before and after. This 577x629 image took about 3 seconds to process. Note the removal of compression artifacts in the smoothed image.

Figure 4: Before and after. This 746x750 image took about 5 seconds to process.

This algorithm can also be used to sharpen images via unsharp masking. By smoothing the image and subtracting the smoothed image from the original image, we can simulate passing the

image through a high-pass filter. After adding a fraction of that image to the original, the overall effect is a sharper, more detailed image. A basic equation is shown below:

$$I_{\text{sharp}} = I + \alpha(I - S) \tag{9}$$

Note that while this method certainly works, it enhances noise and artifacts.[GR96] In particular, since the edges of the image are actually sharpened by our smoothing algorithm, may cause gradient reversal. Xu et al fix this by filtering them with Gaussian kernels and optimizing to automatically determine the proper ones. I did not implement this part of their paper.

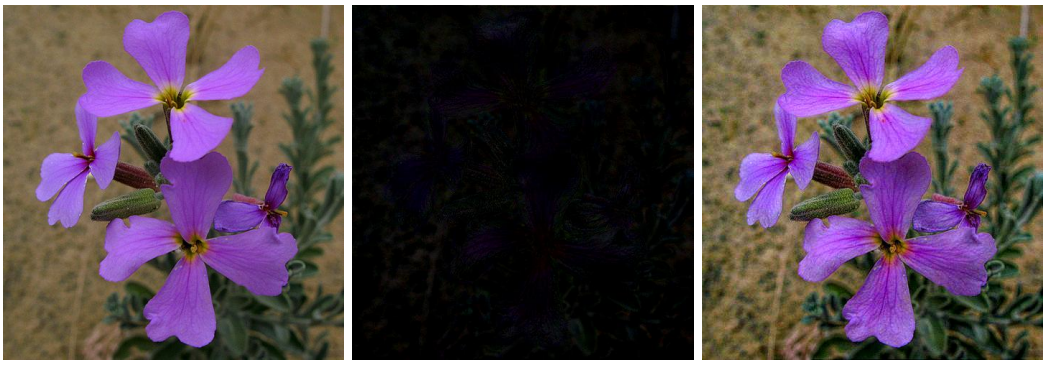


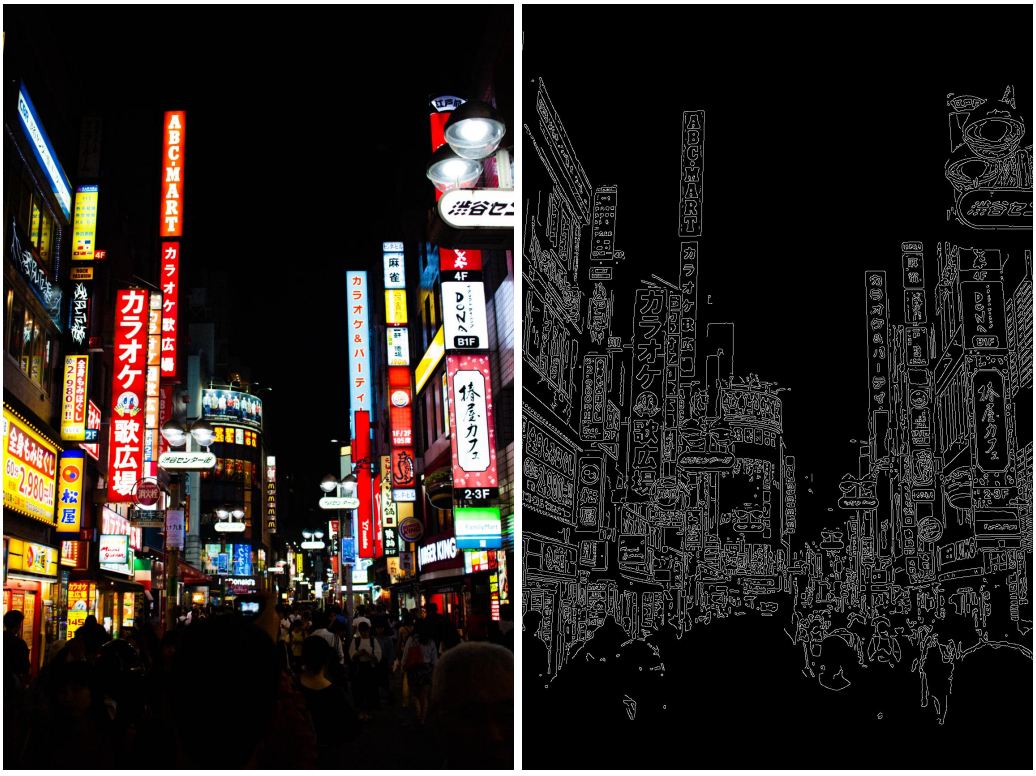
Figure 5: (left to right) Original, Original − Smoothed, Sharpened (photograph from Xu et al)

I also demonstrate the benefit to edge detection that this smoothing method can have below. I use a Canny edge detector on two images–note how the smoothed image has cleaner edges and is robust to noise.

Figure 6: The "cleanup" effect that the smoothing algorithm has can be seen in this set of photos. The top photos are the original, and the bottom photos are the smoothed result

However, there is a limit to the efficacy of this "cleanup". When run on a noisy image or an image already quite clean of noise, the smoothing effect does not really help in edge detection, at least with a Canny detector. I show two examples below:

Figure 7: In this photo, some of the water droplets are large enough to be mathematically considered as "salient edges" via our algorithm and are difficult to smooth out. There is a considerable amount of smoothing, but the result is still quite noisy



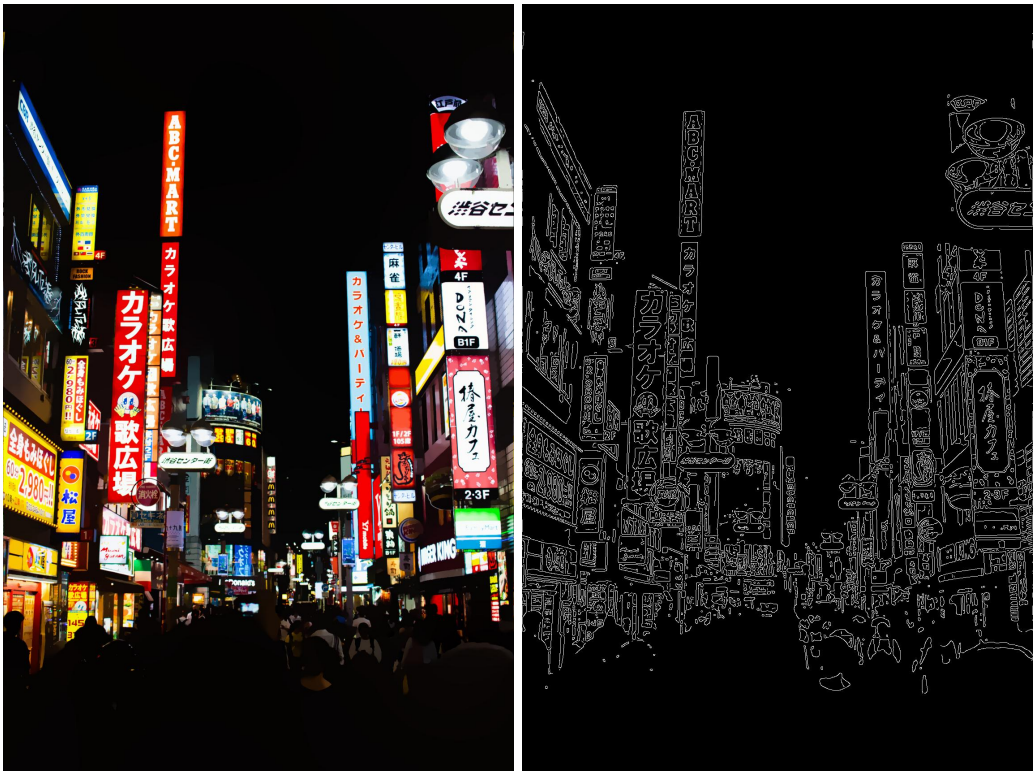


Figure 8: In this photo, the edges are already quite smooth and the overall image is free of noise, so the smoothing effect does not change the image very much.

4 Conclusion

My implementation yielded similar results to the Xu et al implementation, but suffered from slower speeds with larger images. The Xu et al implementation took 150 seconds to process a 3-channel 5615x3744 image, while my implementation took 200 seconds. This could be due to the way I implemented the calculation of S from h, v–I use two calls to the `fft2` function to calculate $\mathcal{F}(\partial_x)^*\mathcal{F}(h) + \mathcal{F}(\partial_y)^*\mathcal{F}(v)$ as well as two calls to the `conj` function; if I had more time, I could possibly compute $\partial_x^T h + \partial_y^T v$ in the image domain and only use one call to `fft2`, which may account for the speed difference. And while the Xu et al implementation is indeed faster, it is still quite slow in processing these images, at least when run on a CPU. To speed up even further (say, for implementation in an image-editing software), running the Fast Fourier Transform on a GPU could make the runtime even better.

As Xu et al note, this smoothing algorithm can work in tandem with local averaging filters, since it relies on the global L_0 norm. I have not demonstrated it here, but using a bilateral Gaussian filter in conjunction with this algorithm can enhance the smoothing effect and eliminate more noise than the filters on their own. For future work, I am curious to see how this algorithm could change if the Roberts Cross operator

$$\begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}, \begin{pmatrix} 0 & 1 \\ -1 & 0 \end{pmatrix}$$

were used to calculate the finite differences instead of the forward difference. I intended to explore this result, but ran out of time in the end.

References

[GR96] Sanjit K. Mitra Tian-Hu Yu Giovanni Ramponi, Norbert K. Strobel. Nonlinear unsharp masking methods for image contrast enhancement. *Journal of Electronic Imaging*, 5:5 – 5 – 14, 1996.

[TM98] C. Tomasi and R. Manduchi. Bilateral filtering for gray and color images. In *Sixth International Conference on Computer Vision (IEEE Cat. No.98CH36271)*, pages 839–846, Jan 1998.

[XLXJ11] Li Xu, Cewu Lu, Yi Xu, and Jiaya Jia. Image smoothing via l0 gradient minimization. *ACM Trans. Graph.*, 30(6):174:1–174:12, December 2011.