CHAPTER 7

Application:Denoising Images by Optimization

You are given a noisy image and asked to produce the original. The detailed rules may vary somewhat in different applications. For example, you might have only a rough model of the noise process; you may have a detailed and accurate model of the noise process (though this is unusual); the noise might be deterministic, but depend on parameters you cannot know (this occurs with underwater images); you may need to denoise very few images (or even only one – astronomy applications); you may need to denoise any image, or any image of a particular class; and so on.

Denoising is an important topic for at least two reasons. First, it is extremely useful. Second, the way to denoise an image is to exploit information about what images are "like", so studying denoising strategies is a good way to build intuitions about images. For example, gaussian smoothing (Section 6.2.1) or median filtering (Section 6.2.2) to suppress noise works fairly well because most image pixels "look like" their neighbors.

7.1 CONSIDERATIONS

This chapter uses a master recipe for denoising. Write $\mathcal{N}$ for a noisy image, and think of denoising as finding a denoised image $\mathcal{D}$ that is (a) close to $\mathcal{N}$ and (b) more like a real image. Write

$$\begin{aligned} C(\mathcal{D}) &= [\text{distance from } \mathcal{D} \text{ to } \mathcal{N}] + [\text{unrealism cost for } \mathcal{D}] \\ &= [\text{data term}] + [\text{penalty term}] \end{aligned}$$

and choose a $\mathcal{D}$ that minimizes this cost function. Methods differ mainly by the penalty term, which has a significant effect on how hard the optimization problem is. This framework leads to very strong denoising methods, at the cost of solving what can be a nasty optimization problem.

Procedure: 7.1 *Denoising by Optimization: Master recipe*

Denoise a noisy image $\mathcal{N}$ by finding a denoised image $\mathcal{D}$ that is (a) close to $\mathcal{N}$ and (b) more like a real image. Do this by minimizing the cost function

$$\begin{aligned} C(\mathcal{D}) &= [\text{distance from } \mathcal{D} \text{ to } \mathcal{N}] + [\text{unrealism cost for } \mathcal{D}] \\ &= [\text{data term}] + [\text{penalty term}] . \end{aligned}$$

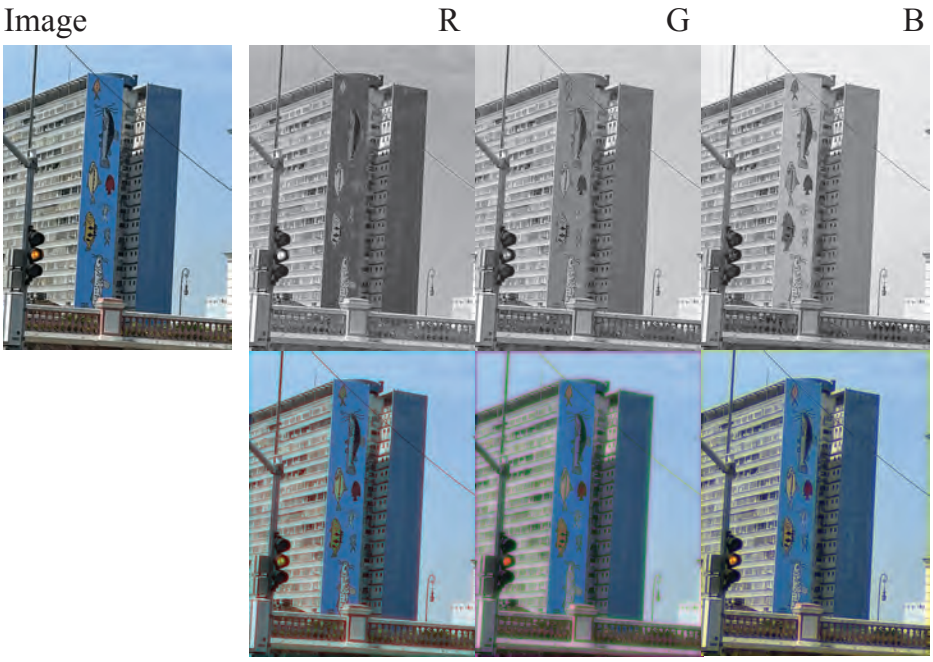

FIGURE 7.1: *RGB color components are heavily correlated, as you can see by looking at images where only one component has been smoothed.. The* **top row** *shows the R, G, and B components of the color image at the* **left**. *The* **bottom row** *shows color images obtained by smoothing one component, then recombining all three. Notice that smoothing any of the R, G, B components alone leads to odd color effects at edges (G is particularly bad).* Image credit: *Figure shows my photograph of a building in downtown Manaus.*

7.1.1 Denoising Color Images

Representing color is a broad subject, and Section 29.3 is a brief introduction. The particular color representation one uses can be important in denoising. Section 8.1 showed that gradients in R, G and B tend to appear in the same place. This is the result of two effects: the R, G and B components of an image are highly correlated (meaning that RGB is not a particularly good color representation for many applications); and isoluminant changes in color really are rare.

Correlation between R, G and B means that, given (say) the true R and G components of an image, you can do a fair job of predicting B. This means that adjusting the components independently is dangerous: adjustments to R and G should be reflected in B, say. The effect is easily observed. Smoothing one of R, G, or B (but not the others) results in strange color effects (**exercises** , Figure 7.1).

Correlation between RGB components is an experimental fact about the world. Remarkably, one can choose a color space that largely decorrelates any image rather well. LAB is such a color space. It has three attractions: first, it

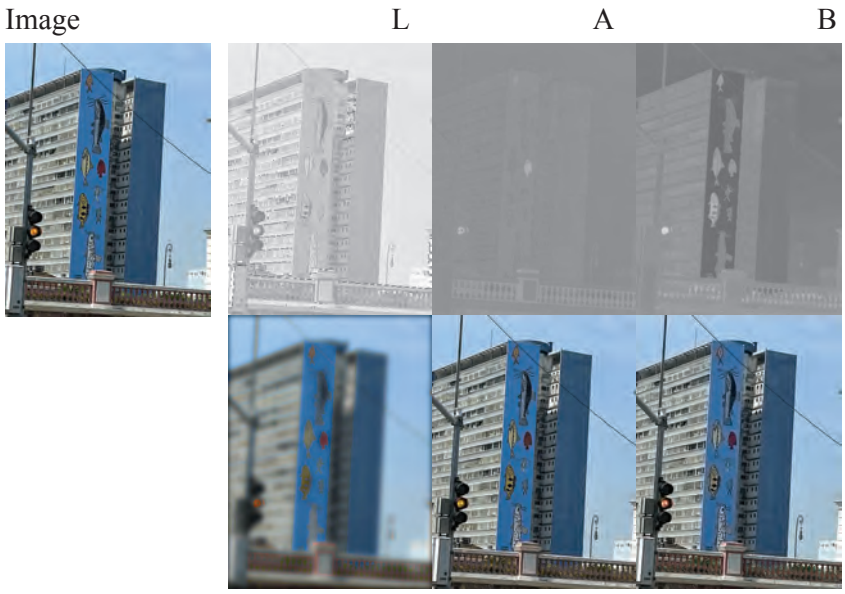

FIGURE 7.2: *Decorrelating the components of a color image before smoothing is important, but one does not need to do this on a per-image basis. The* **top row** *shows the L, A, and B components for this image on the* **right**. *Because these components can be negative, they have been scaled and shifted so that a zero value is mid gray, the largest value is bright and the smallest is dark (the same scale has been applied to each component so you can see relative sizes). The* **bottom row** *shows color images obtained by smoothing one component, then recombining all three. Smoothing L results in a blurry color image; smoothing A or B alone largely has no effect. This means one can use sophisticated methods on the L component and just smooth the A and B components.* Image credit: *Figure shows my photograph of a building in downtown Manaus.*

splits images into an intensity component (L) and two color components (A) and (B), which largely do not depend on intensity; second, these components are largely decorrelated spatially; and third, short distances in the color space are rather a good representation of human perception of the size of color changes. Section 29.3.2 offers more detail on the construction of LAB.

Humans are much better at perceiving spatial detail in intensity than they are in color (this has to do with a variety of physiological details about how the human color system works). In an appropriately chosen color space, the color components can be heavily smoothed without affecting human perception of the image all that much (Figure 7.2), as long as the intensity component is not touched. For each denoising method, I will apply the following strategy to deal with color images: decompose into LAB; use sophisticated denoising on L; and just use a smoothed version of A and B (because mostly people don't notice irregularities in the color components, as above).

7.1.2 Evaluating by Comparing to Ground Truth

An important part of denoising is knowing how well a procedure actually works. For the moment, assume that you have access to a collection of test image pairs, where you have a noisy version and the clean version. You can then evaluate by comparing the denoised noisy image with the clean image. The main question is what number to compute from the pair.

One standard evaluation statistic is the mean *PSNR* or *peak signal-to-noise ratio*. For each pair $(\mathcal{N}, \mathcal{C})$ of noisy version - clean version, first denoise the noisy image to get $\mathcal{D}$. Now compute the PSNR for the pair $(\mathcal{D}, \mathcal{C})$, using

$$\text{psnr}(\mathcal{D}, \mathcal{C}) = 20 \log \frac{\max_{ij} \mathcal{C}_{ij}}{\sqrt{\sum_{ij} (\mathcal{D}_{ij} - \mathcal{C}_{ij})^2}}$$

and average that PSNR over pairs. The PSNR has some good properties: as $\mathcal{D}$ gets closer to $\mathcal{C}$, the PSNR gets larger; and $\text{psnr}(s\mathcal{D}, s\mathcal{C}) = \text{psnr}(\mathcal{D}, \mathcal{C})$ for $s > 0$ (so you can't change the PSNR by scaling the images).

In some applications, versions of the original image that are uniformly slightly brighter or slightly darker might be acceptable, but the PSNR will penalize a method that can't estimate the brightness of the ground truth image. In these situations, one can use

$$\text{psnr}(\mathcal{D}, \mathcal{C}) = 20 \log \frac{\max_{ij} \mathcal{C}_{ij}}{\min_{s} \sqrt{\sum_{ij} (s\mathcal{D}_{ij} - \mathcal{C}_{ij})^2}}.$$

PSNR doesn't really take human perception into account, so that reconstructions with quite good PSNR might look bad to a user, and reconstructions with quite bad PSNR might look good. For example, imagine the reconstruction is the original image, but shifted by one pixel to the left (obtain the missing column by copying its neighbor). The PSNR might be quite bad, but the reconstruction would be good and would look good.

An ideal evaluation metric should not be seriously affected by shifts like this. A natural construction is to compare summary properties of windows of pixels rather than comparing pixels. This construction leads to the *SSIM* or *structural similarity index metric*. The clean image and the denoised image are broken into quite small overlapping windows; summary statistics for these windows are computed and compared, with a metric that is quite robust to changes in intensity; and the comparison is averaged over all windows. Implementations of SSIM appear in most API's.

Human observers have a variety of preferences that SSIM does not fully account for. For example, humans like sharp edges without ringing but can be relaxed about whether the edge is in the right place. As another example, humans are surprisingly good at perceiving lines, and dislike edges that are close to, but not on, a line. The *LPIPS* or *Learned Perceptual Image Patch Similarity* metric is an attempt to deal with this. The clean image and the denoised image are

broken into overlapping windows; deep network features are computed for windows; a weighted difference is computed for these features; and the comparison is averaged over all windows. The features are learned using procedures quite like that of Chapters 17 and 18. The reference Implementation of LPIPS is at `https://github.com/richzhang/PerceptualSimilarity`, and many APIs offer LPIPS evaluation.

7.1.3 Evaluating without Comparison

Sometimes it is difficult to find pairs of clean images and denoised versions. For example, you might not have clean versions of the noisy images; if you don't trust simulations of the noise model, you won't have pairs. Evaluation in this case involves telling whether a denoised image is "like an image", using an *image quality metric*. This is sometimes called *blind evaluation*. These metrics measure how much something that purports to be a natural image is "like an image." You should suspect that this is extremely hard to do accurately. If you could measure how much something is "like an image", then denoising might be straightforward – just make small adjustments to it until it really is a natural image.

Metrics tend to take the following form. Choose some patches in the image; for these, compute some feature vectors; fit a probability model to these feature vectors; and compare the parameters of the fitted model to the parameters of a model fitted to a large number of natural images. Choosing some patches, rather than all, seems to improve the accuracy of the measure, likely because different images contain different numbers of "boring" patches – patches of, say, constant color – and this might bias the model. This recipe was established by the *NIQE* or *Natural Image Quality Evaluator* metric. Implementations of NIQE appear in most APIs.

> **Remember this:** *An important master recipe for denoising is to find something that is (a) close to the original noisy image and (b) like a real image. Telling how much something is like a real image is difficult. You should denoise images in LAB or a similar color representation; typically, use a sophisticated algorithm on the intensity component, and smooth the color components with a gaussian. Evaluate a denoising procedure either by comparing results with ground truth (useful comparisons are PSNR; SSIM; and LPIPS) or using an image quality metric (like NIQE).*

7.2 DENOISING BY OPTIMIZATION

For this Chapter, the data term in the master recipe is

$$\sum_{ij} (\mathcal{D}_{ij} - \mathcal{N}_{ij})^2$$

(the ssd of Section 4.3.1). The penalty term is much more interesting. The crucial rule of thumb – pixels tend to be like their neighbors – suggests that the penalty

FIGURE 7.3: *A color image (***upper left***), with additive gaussian noise (***lower left***), denoised by penalizing the gradient as in Section 7.2.1 at various settings of λ. The denoised images are in the* **top row***, with residual (noisy image - reconstructed image) in the* **bottom row***. The residual is shown on a scale where positive values are light, negative values are dark, and 0 is mid-gray. The noisy image was transformed to LAB; the L component was smoothed with WLS; and the A and B components were smoothed with a gaussian kernel. Notice the strong color noise in the residual. The image noise is independent in R, G and B – so colors can change sharply – and the denoiser is very effective at suppressing this effect. Notice how, as λ is increased, the image becomes rather blurry (compare Figure* **??** *and Figure* **??***).* Image credit: *Figure shows my photograph of marmosets in Sao Paulo.*

function needs to look at gradients in $\mathcal{D}$.

7.2.1 Penalizing the Gradient

The simplest use of gradients would be to denoise $\mathcal{N}$ by looking for $\mathcal{D}$ that is (a) close to $\mathcal{N}$ in squared error; and (b) mostly has small gradients. Rearrange all images into vectors, so $\mathcal{N}$ is vector $\mathbf{n}$, and so on. This representation gives a convenient expression for the gradient. There are matrices $\mathcal{D}_x$ and $\mathcal{D}_y$ that compute the gradient from the image (rearrange the finite differences of Section 5.1.4, or the derivative of gaussians of Section 6.2.3, **exercises**). The gradient of $\mathcal{D}$ can be represented by

$$\begin{pmatrix} \mathcal{D}_x \\ \mathcal{D}_y \end{pmatrix} \mathbf{d}$$

(where the vector of x-derivatives is stacked on the vector of y-derivatives). In turn, the sum of gradient magnitudes for $\mathcal{D}$ is given by

$$\mathbf{d}^T \left[\mathcal{D}_x^T \mathcal{D}_x + \mathcal{D}_y^T \mathcal{D}_y\right] \mathbf{d}.$$

You can find a $\mathcal{D}$ that is (a) close to $\mathcal{N}$ in squared error; and (b) mostly has small gradients by finding

$$\underset{\mathbf{u}}{\text{argmin}} \ \frac{1}{2} \left[\mathbf{u} - \mathbf{n}\right]^T \left[\mathbf{u} - \mathbf{n}\right] + \frac{\lambda}{2} \mathbf{u}^T \left[\mathcal{D}_x^T \mathcal{D}_x + \mathcal{D}_y^T \mathcal{D}_y\right] \mathbf{u}$$

for some λ that weights the significance of the gradient term against the error term. Now write $\mathcal{K} = \left[\mathcal{D}_x^T \mathcal{D}_x + \mathcal{D}_y^T \mathcal{D}_y\right]$. At the minimum,

$$\left(\mathcal{I} + \lambda \mathcal{K}\right) \mathbf{d} = \mathbf{n}$$

(differentiate and set to zero **exercises**). The solution is very like what you would get if you simply smoothed the image with a gaussian (Figure **??**).

7.2.2 Weighted Least Squares

The problem with penalizing the gradient is that image should have strong gradients in some places. Suppressing these graidents doesn't make the image closer to a real image. Ideally, if there is good evidence in support of a large gradient in the denoised image, that large gradient should not be penalized. The *weighted least squares filter* (almost always WLS) seeks a $\mathcal{D}$ that is (a) close to $\mathcal{N}$ in squared error; (b) mostly has small gradients; and (c) has high gradients when $\mathcal{N}$ does.

WLS weights the cost of a gradient in the reconstructed image using the gradients of $\mathcal{N}$. At locations where the gradient of $\mathcal{N}$ is large, it should be cheap to have a large gradient in $\mathcal{D}$. WLS achieves this using diagonal matrices of weights obtained from $\mathcal{N}$. Write $\mathcal{A}_x(\mathbf{n})$ for the weights on the x-derivative and $\mathcal{A}_y(\mathbf{n})$ for the weights on the y-derivative.

At a location where the value of $\mathcal{A}_y$ is small, $\mathcal{D}$ could have a large y-derivative, but at locations where the value is large, $\mathcal{D}$ must have a small y-derivative. The key question here is the choice of $\mathcal{A}_x$ and $\mathcal{A}_y$. A large derivative in $\mathcal{D}$ should only occur when the derivative of $\mathcal{N}$ is large and reliable. Small derivatives $\mathcal{N}$ are untrustworthy, and should not appear in $\mathcal{D}$. So at pixels where $\mathbf{n}$ has a small x derivative, the relevant term in $\mathcal{A}_x$ should be big. Similarly, when $\mathbf{n}$ has a large x derivative, the relevant term in $\mathcal{A}_x$ should be small. A fair choice is to form $\mathbf{w} = \mathcal{D}_x \mathbf{n}$, and then the i, i'th element of $\mathcal{A}_x$ is

$$\frac{1}{|w_i|^\alpha + \epsilon}$$

where α is some power, typically between 1.2 and 2, and ϵ is a small positive constant to avoid division by zero; $\mathcal{A}_y$ follows.

The new image $\mathbf{d}$ (a vector version of the denoised image $\mathcal{D}$) should then solve

$$\underset{\mathbf{u}}{\text{argmin}} \ \left[\mathbf{u} - \mathbf{n}\right]^T \left[\mathbf{u} - \mathbf{n}\right] + \lambda \mathbf{u}^T \left[\mathcal{D}_x^T \mathcal{A}_x^T \mathcal{A}_x \mathcal{D}_x + \mathcal{D}_y^T \mathcal{A}_y^T \mathcal{A}_y \mathcal{D}_y\right] \mathbf{u}$$

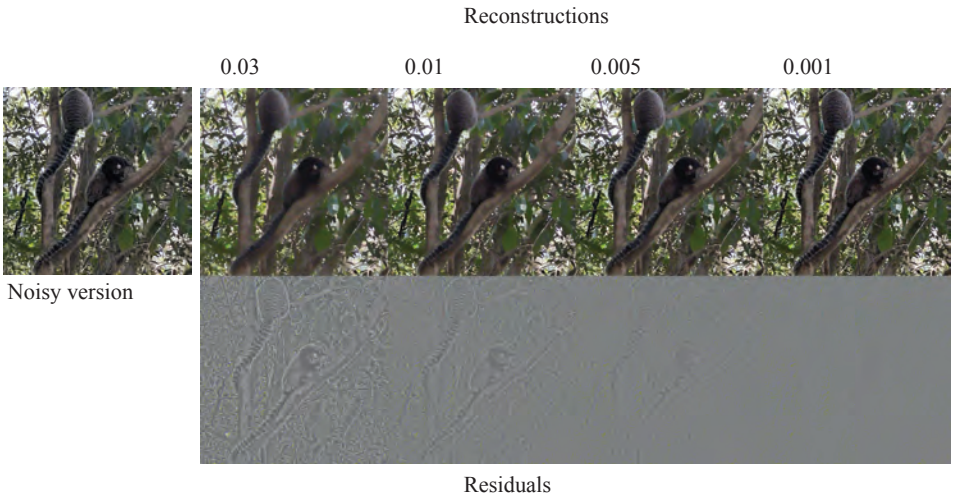

FIGURE 7.4: *The color image of Figure* **??** *with additive gaussian noise (***left***), denoised using weighted least squares (WLS; Section 7.2.2) at various settings of* λ*. The denoised images are in the* **top row***, with residual (noisy image - reconstructed image) in the* **bottom row***. The residual is shown on a scale where positive values are light, negative values are dark, and 0 is mid-gray. The noisy image was transformed to LAB; the L component was smoothed with WLS; and the A and B components were smoothed with a gaussian kernel. Notice the strong color noise in the residual. The image noise is independent in R, G and B – so colors can change sharply – and the denoiser is very effective at suppressing this effect. Notice how, as* λ *is increased, the image does not become blurry (compare Figure* **??***). Edges are largely preserved, but detail is lost with increasing* λ*.* Image credit: *Figure shows my photograph of marmosets in Sao Paulo.*

where the first term pushes $\mathbf{u}$ to be like $\mathbf{n}$, the second term controls the derivatives of $\mathbf{u}$ and λ is some weight balancing the two terms. Write $\mathcal{L} = \left[\mathcal{D}_x^T\mathcal{A}_x^T\mathcal{A}_x\mathcal{D}_x + \mathcal{D}_y^T\mathcal{A}_y^T\mathcal{A}_y\mathcal{D}_y\right]$; then solving this problem is a matter of solving

$$\mathcal{F}(\lambda)\mathbf{d} = (\mathcal{I} + \lambda\mathcal{L})\mathbf{d} = \mathbf{n}.$$

WLS denoising tends to produce less blurry images than just suppressing large gradients (Figure 7.4).

There are several elements in this recipe that will recur. Notice that $\mathbf{d}$ is a *nonlinear* function of $\mathbf{n}$ (because elements of $\mathbf{n}$ are used in the construction of $\mathcal{L}$). Assume $\mathcal{L}$ is known; then $\mathbf{d}$ is linear in $\mathbf{n}$. You should think about this process as one in which $\mathbf{n}$ is used to choose a different linear operation to map $\mathbf{n}$ to the value of $\mathbf{d}$ at each pixel.In principle, one could insist that $\mathcal{L}$ be formed from the denoised image $\mathbf{u}$, but that would result in a very nasty optimization problem indeed.

Natural variants of this procedure involve estimating an "easy" denoised version $\hat{\mathbf{d}}$ of the original image, then using that to form $\mathcal{L}$.

7.2.3 L2 and L1 Norms

The cost function for Section 7.2.1 penalizes the square of the *L2 norm* of the derivative (definition in a box).

Definition: 7.1 *The L2 norm*

The L2 norm of a vector $\mathbf{v}$ is given by

$$\| \mathbf{v} \|_2 = \sqrt{\mathbf{v}^T \mathbf{v}}.$$

Warning: It is quite common to refer to the *square* of the L2 norm as the L2 norm. I will try not to do this, because it's wrong, but you'll bump into this in the literature rather often.

Vectors with small L2 norm tend to have many small components, implying that the cost function encourages gradients to have small components. Ideally, the cost would encourage gradients with many zero components, so regions have constant brightness if possible, and a very slow ramp in intensity might be replaced with a constant value. You can achieve this by penalizing the *L1 norm* of the gradients (definition in a box).

Definition: 7.2 *The L1 norm*

The L1 norm of a vector $\mathbf{v}$ is defined by

$$\| \mathbf{v} \|_1 = \sum_i | v_i |.$$

A vector with small L1 norm will tend to have zero elements. You can see this by comparing two cases. Consider choosing a vector $\mathbf{u}$ that is (a) close to some reference vector $\mathbf{g}$ and (b) has small L2 norm. You find this vector by minimizing

$$C_2(\mathbf{u}) = \frac{1}{2} [\mathbf{u} - \mathbf{g}]^T [\mathbf{u} - \mathbf{g}] + \frac{\lambda}{2} \mathbf{u}^T \mathbf{u}.$$

The $\mathbf{u}$ that minimizes $C_2(\mathbf{u})$ is

$$\left(\frac{1}{1 + \lambda} \right) \mathbf{g}.$$

In turn, this means that you can't force $\mathbf{u}$ to have zeros by making λ large (except when it is infinity, which isn't a useful case).

Now consider choosing a vector $\mathbf{u}$ that is (a) close to some reference vector $\mathbf{g}$ and (b) has small L1 norm. You find this vector by minimizing

$$C_1(\mathbf{u}) = \frac{1}{2} [\mathbf{u} - \mathbf{g}]^T [\mathbf{u} - \mathbf{g}] + \lambda \| \mathbf{u} \|_1.$$

Actually minimizing this expression is a little tricky. The penalty term isn't differentiable, which creates some inconvenience, but it is a sum over elements of $\mathbf{u}$,

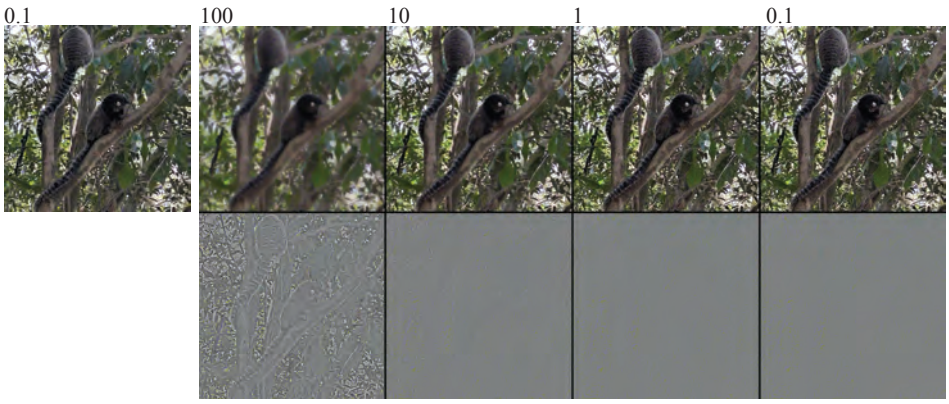

FIGURE 7.5: *The color image of Figure* **??**, *with additive gaussian noise at* $\sigma = 0.1$ *denoised using total variation denoising (TVD; Section 7.2.4) at various settings of* λ. *The denoised images are in the* **top row**, *with residual in the* **bottom row**. *The residual is shown on a scale where positive values are light, negative values are dark, and 0 is mid-gray. The noisy image was transformed to LAB; the L component was smoothed with TVD; and the A and B components were smoothed with a gaussian kernel. Notice the strong color noise in the residual. The image noise is independent in R, G and B – so colors can change sharply – and the denoiser is very effective at suppressing this effect. Notice how, as* λ *is increased, the image does not become blurry (compare Figure* **??***). Edges are largely preserved, but detail is lost with increasing* λ. Image credit: *Figure shows my photograph of marmosets in Sao Paulo.*

so you can minimize term by term. Now consider the i'th element of $\mathbf{u}$. If $|g_i|$ is sufficiently large, then it is easy to show that

$$u_i = \frac{g_i}{1+\lambda}$$

(**exercises**). Now consider what happens when $g_i = \lambda$. If $u_i = 0$, then the cost will be $\lambda^2/2$, but if $u_i = \epsilon > 0$ where ϵ is small, the cost will be $(1/2)(\lambda^2 + \epsilon^2)$. This analysis implies correctly that if $-\lambda < g_i < \lambda$, $u_i = 0$. In turn, using an L1 norm as a penalty on the gradients tends to cause the reconstruction to have many zero gradients

Remember this: *Penalizing the L1 norm tends to produce vectors with numerous zeros.*

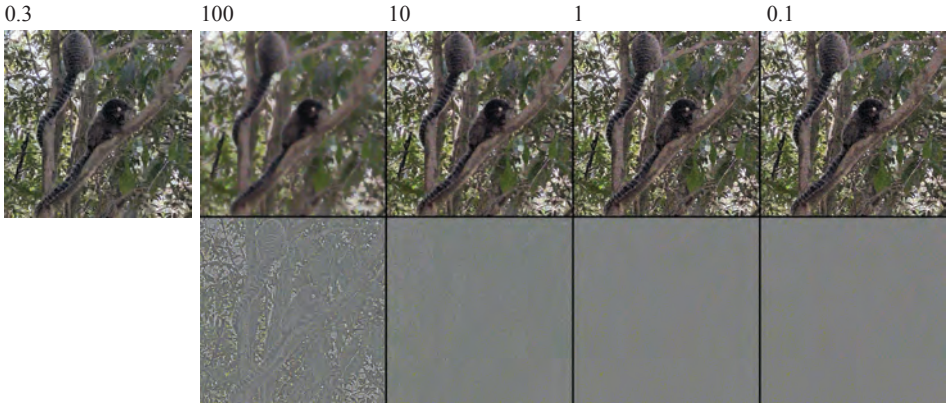

FIGURE 7.6: *The color image of Figure 7.4, with additive gaussian noise at $\sigma = 0.3$, denoised using total variation denoising (TVD; Section 7.2.4) at various settings of λ. Details in caption of Figure 7.5.*

7.2.4 Total Variation Denoising

In *total variation denoising*, the penalty is an L1 norm to the gradient. There are a variety of ways of doing this. In one approach, one seeks

$$\underset{u}{\text{argmin}} \ \frac{1}{2}\left[\mathbf{u}-\mathbf{g}\right]^T\left[\mathbf{u}-\mathbf{g}\right] + \lambda\left[\|\mathcal{D}_x\mathbf{u}\|_1 + \|\mathcal{D}_y\mathbf{u}\|_1\right].$$

Note this cost function isn't differentiable, but it is convex. The optimization problem for this cost function is well understood, and is relatively easily managed (though beyond our scope). However, you should notice that the penalty encourages zeros in the x and y components of the gradient, which isn't necessarily the same as zero gradient magnitudes. One could get a solution where the zeros in the x components are not aligned with the zeros of the y components, so the penalty is biased against some gradient directions but not others. Most strong image APIs will have a total variation denoising implementation (I used SciPy's implementation).

An alternative formulation requires a bit more notation. Write $d_{x,i}(\mathbf{u})$ for the i'th component of $\mathcal{D}_x\mathbf{u}$, and so on. Then solve

$$\underset{u}{\text{argmin}} \ \frac{1}{2}\left[\mathbf{u}-\mathbf{g}\right]^T\left[\mathbf{u}-\mathbf{g}\right] + \lambda\left[\sum_i \sqrt{d_{x,i}(\mathbf{u})^2 + d_{y,i}(\mathbf{u})^2}\right]$$

which is also not differentiable. Solutions require rather more elaborate work than solutions for the previous formulation, and tend to be somewhat slower, but are not biased.

7.2.5 Image Deblurring with a Regularizer

Denoising takes something that isn't quite an image and finds an image that is very like it. Many phenomena can produce something that isn't quite an image. For

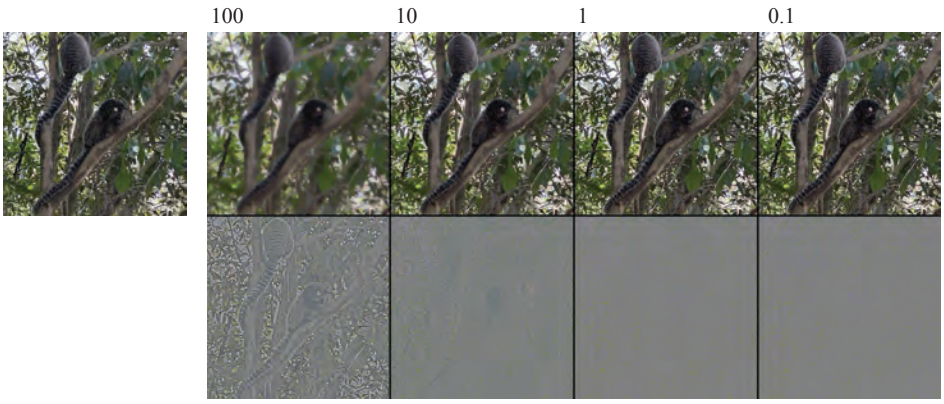

FIGURE 7.7: *The color image of Figure 7.4, with a variant of poisson noise where a randomly chosen pixel in a randomly chosen color channel is flipped, denoised using total variation denoising (TVD; Section 7.2.4) at various settings of λ. The denoised images are in the* **top row**, *with residual in the* **bottom row**. *The residual is shown on a scale where positive values are light, negative values are dark, and 0 is mid-gray. The noisy image was transformed to LAB; the L component was smoothed with TVD; and the A and B components were smoothed with a gaussian kernel. Notice the strong color noise in the residual. The image noise is independent in R, G and B – so colors can change sharply – and the denoiser is very effective at suppressing this effect. Notice how, as λ is increased, the image does not become blurry (compare Figure* **??**). *Edges are largely preserved, but detail is lost with increasing λ.* Image credit: *Figure shows my photograph of marmosets in Sao Paulo.*

example, take an image and blur it. The result isn't an image, but it is quite close to one. Recall from Section 21.3 that blurring is a linear operation. Write $\mathbf{t}$ for the true image in vector form, $\mathbf{d}$ for the deblurred estimate in vector form, $\mathbf{b}$ for the observed image in vector form, and $\mathcal{B}$ for the linear operator that blurs. Assume $\mathcal{B}$ is known.Notice $\mathbf{b}$ is not *exactly* the blurred image. At the very least, there is some error from the numerical representation, and there might be some small noise present, too. Then

$$\mathbf{b} = \mathcal{B}\mathbf{t} + \xi$$

(where ξ is a vector of very small errors) and least-squares suggests choosing $\mathbf{d}$ that yields something close to $\mathbf{b}$ when you blur it. This means choosing

$$\underset{u}{\text{argmin}} \ (\mathcal{B}\mathbf{u} - \mathbf{b})^T (\mathcal{B}\mathbf{u} - \mathbf{b}) .$$

If you differentiate and set to zero, $\mathbf{d}$ must be a solution to

$$\mathcal{B}^T\mathcal{B}\mathbf{d} = \mathcal{B}^T\mathbf{b}.$$

The least squares solution is not reliable, because $\mathcal{B}$ is a smoothing operator – say, convolve with a gaussian for concreteness. This means $\mathcal{B}^T\mathcal{B}$ *must* have some

FIGURE 7.8: **Left** *shows an image blurred with a gaussian,* $\sigma = 1$*;* **center left** *shows regularized least squares reconstructions for different values of the regularization constant;* **center right** *shows deblurred images, using the WLS strategy of Section* **??***; and* **right** *shows the ground truth image. Notice that there must be many very small eigenvalues in* $\mathcal{B}$*, because when the regularization constant is small, the reconstruction is almost unrecognizable (the black and white "snakeskin" pattern is a set of very high frequency components that are easily smoothed out – which is why they correspond to small eigenvalues). Increasing the regularization constant helps control these patterns, but increasingly darkens the reconstruction. The WLS method requires an estimate of the gradient, which I took from the least-squares reconstruction. WLS helps control these components, but the best reconstruction certainly isn't perfect. Scoring whether a reconstruction is "like" an image is difficult.* Image credit: *Figure shows my photograph of a delicious monster in Sao Paulo.*

small eigenvalues, because smoothing suppresses some patterns (which is the whole point of smoothing). Note that $\mathcal{B}^T\mathcal{B}$ must also have some eigenvalues fairly close to one, because there are some patterns that change very little when smoothed. The patterns that are suppressed by smoothing must be exaggerated by $\left(\mathcal{B}^T\mathcal{B}\right)^{-1}$, and may be very heavily exaggerated. This is a serious problem, because the reconstruction is

$$\begin{aligned}\left(\mathcal{B}^T\mathcal{B}\right)^{-1}\mathcal{B}^T\mathbf{b} &= \left(\mathcal{B}^T\mathcal{B}\right)^{-1}\mathcal{B}^T\left[\mathcal{B}\mathbf{t}+\xi\right] \\ &= \mathbf{t}+\left(\mathcal{B}^T\mathcal{B}\right)^{-1}\mathcal{B}^T\xi.\end{aligned}$$

Now even if ξ is very small, the term $\left(\mathcal{B}^T\mathcal{B}\right)^{-1}\mathcal{B}^T\xi$ is likely large, because $\left(\mathcal{B}^T\mathcal{B}\right)^{-1}$ has some large eigenvalues, and $\mathcal{B}^T\xi$ is likely to have some components in the direction of the corresponding eigenvectors.

There is a traditional procedure to handle very small eigenvalues in a matrix,

FIGURE 7.9: **Left** *shows an image blurred with a gaussian, $\sigma = 3$;* **center left** *shows regularized least squares reconstructions for different values of the regularization constant;* **center right** *shows deblurred images, using the WLS strategy of Section* **??***; and* **right** *shows the ground truth image (more details in the caption of Figure 7.8). This reconstruction problem is extremely difficult for these methods.* Image credit: *Figure shows my photograph of a delicious monster in Sao Paulo.*

known as *regularization*. You look for

$$\underset{u}{\text{argmin}}\ C(\mathbf{u}) = \underset{u}{\text{argmin}}\ (\mathcal{B}\mathbf{u} - \mathbf{b})^T (\mathcal{B}\mathbf{u} - \mathbf{b}) + \lambda \mathbf{u}^T \mathbf{u}.$$

Differentiating and setting to zero shows that $\mathbf{d}$ solves

$$(\mathcal{B}^T\mathcal{B} + \lambda\mathcal{I})\mathbf{d} = \mathcal{B}^T\mathbf{b}$$

for some choice of $\lambda > 0$, chosen to get good results. Notice that the map from blurry image $\mathbf{b}$ to deblurred image $\mathbf{d}$ is linear in $\mathbf{b}$, and should be shift invariant, too **exercises** .

7.2.6 Deblurring with the Master Recipe

You can see regularization either as penalizing solutions that are too big, or as constructing a matrix that is close to $(\mathcal{B}^T\mathcal{B})^{-1}$ but does not have small eigenvalues. Alternatively, you can see regularization as a rather crude version of the cost function for Section 7.2.2, where the penalty term discourages images that are "too big", so again you are minimizing a cost function. This cost function has a familiar form:

$$\begin{aligned} C(\mathbf{u}) &= [\text{Term comparing } \mathcal{B}\mathbf{u} \text{ to } \mathbf{b}] + [\text{Term evaluating realism of } \mathbf{u}] \\ &= [\text{data term}] + [\text{penalty term}] \end{aligned}$$

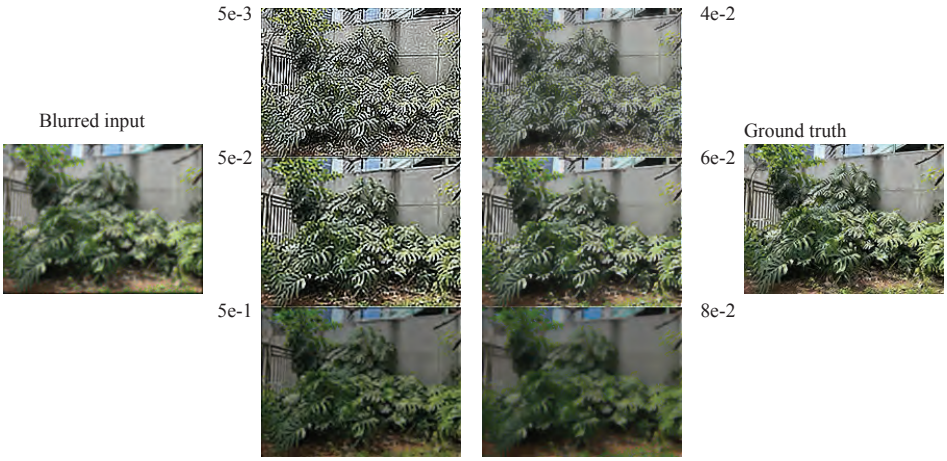

FIGURE 7.10: **Left** *shows an image blurred with a gaussian, $\sigma = 1$;* **center left** *shows regularized least squares reconstructions for different values of the regularization constant;* **center right** *shows deblurred images, using the TVD strategy of Section* **??***; and* **right** *shows the ground truth image (more details in the caption of Figure 7.8). TVD can control the problem high frequency components, with an appropriate choice of weight.* Image credit: *Figure shows my photograph of a delicious monster in Sao Paulo.*

meaning it is possible to apply the master recipe (Section 7.2). I will apply the methods of Sections 7.2.2 and 7.2.4 according to the master recipe, but other choices are possible.

Weighted least squares: The main question to deal with in using weighted least squares is how to form $\mathcal{A}_x$ and $\mathcal{A}_y$. One strategy is to form a regularized least squares estimate $\hat{\mathbf{u}}$, so

$$(\mathcal{B}^T\mathcal{B} + \lambda\mathcal{I})\hat{\mathbf{u}} = \mathcal{B}^T\mathbf{g}$$

then use this estimate to form $\mathcal{A}_x$ and $\mathcal{A}_y$. The problem then becomes large-scale linear algebra. Figures 7.8 and 7.9 show some results.

Total variation denoising: One natural strategy to use total variation denoising for upsampling and deblurring is to apply total variation denoising to a least squares prediction. Doing anything else requires substantial optimization tricks, sketched in the **exercises** . Figures 7.10 and 7.11 show some results.

Denoising by controlling high spatial frequency components is quite helpful, but deblurring exposes difficulties with the approach. Compare Figures 7.11 and 7.9 with Figures 7.10 and 7.8, and notice how much harder it is to deblur a really blurry image than it is to deblur a slightly blurry image. This is because there is very much less image evidence of high spatial frequencies (**exercises**) and so the corresponding eigenvalues of $(\mathcal{B}^T\mathcal{B})^{-1}$ are very large and very hard to control. Further, all the methods we have seen require engaging with inconvenient optimization problems. Chapters 21.3 and 21.3 describe much richer classes of procedure that can be used for deblurring.

FIGURE 7.11: **Left** *shows an image blurred with a gaussian,* $\sigma = 3$*;* **center left** *shows regularized least squares reconstructions for different values of the regularization constant;* **center right** *shows deblurred images, using the TVD strategy of Section* **??***; and* **right** *shows the ground truth image (more details in the caption of Figure 7.8). TVD has a much harder time controlling these components than for Figure 7.10.* Image credit: *Figure shows my photograph of a delicious monster in Sao Paulo.*

Remember this: *A variety of different denoising methods can be derived from the master recipe by using different penalties. Finding a solution involves solving problems of varying levels of difficulty. An L2 penalty on the gradient of the reconstructed image means the reconstructed image is a linear function of the orginal image and is easy to find, but tends to produce blurry reconstructions. Weighted least squares allows the reconstruction to have large gradients when the noisy image does. The reconstruction is no longer a linear function of the noisy image, but is still relatively easy to find, and not to be so blurry. Total variation denoising applies an L1 penalty to the gradient. The resulting optimization problem is tricky, but efficient solutions are known. TVD tends to produce reconstructions that have zero gradient at many locations. The master recipe applies to deblurring as well, but now optimization problems need to be regularized.*

7.3 YOU SHOULD

7.3.1 remember these definitions:

7.3.2 remember these procedures:

7.3.3 be able to:

- FOO

EXERCISES

QUICK CHECKS

7.1. Differentiation is linear; you can represent an image as a vector; and you can represent an estimate of its gradient as a vector. Does this guarantee the existence of a matrix that estimates the gradient (as a vector) from an image (as a vector)?

7.2. Write $\mathbf{n}$ for an $N \times N$ image represented as a vector, and $\mathcal{D}_x$ for a matrix that estimates the x-derivative of this vector. What fraction of the entries of $\mathcal{D}_x$ are zero?

7.3. Section 7.2.5 says: " Notice that the map from blurry image $\mathbf{b}$ to deblurred image $\mathbf{d}$ is linear in $\mathbf{b}$, and should be shift invariant, too." Explain. This remark implies that you can deblur with a convolution. Why?

7.4. Section 7.2.5 says: " Notice that the map from blurry image $\mathbf{b}$ to deblurred image $\mathbf{d}$ is linear in $\mathbf{b}$, and should be shift invariant, too." This remark implies that you can deblur with a convolution. Why? What is the kernel?

7.5. Section 7.2.6 says that it is harder to deblur a really blurry image than it is to deblur a slightly blurry image, because some eigenvalues of $(\mathcal{B}^T\mathcal{B})^{-1}$ are very large and very hard to control. Explain.

7.6. Assume you know an image is blurred using a gaussian kernel, *and* you know the σ of the kernel. You could deblur using the convolution theorem. What might go wrong if you do?

7.7. Assume you know an image is blurred using a gaussian kernel, *and* you know the σ of the kernel. The convolution theorem explains why it is much harder to deblur a heavily blurred image than it is to deblur a lightly blurred image. Explain.

7.8. Is the cost function

$$C_1(\mathbf{u}) = \frac{1}{2}\left[\mathbf{u} - \mathbf{g}\right]^T \left[\mathbf{u} - \mathbf{g}\right] + \lambda \| \mathbf{u} \|_1$$

differentiable?

7.9. Imagine you ignore the question of differentiability, and minimize

$$C_1(\mathbf{u}) = \frac{1}{2}\left[\mathbf{u} - \mathbf{g}\right]^T \left[\mathbf{u} - \mathbf{g}\right] + \lambda \| \mathbf{u} \|_1$$

by gradient descent. You will find the estimated solution you get does not have many zeros in it, even though you are using an L1 norm. Explain what happened.

7.10. Section 7.2.4 has: "However, you should notice that the penalty encourages zeros in the x and y components of the gradient, which isn't necessarily the same as zero gradients." Explain.

LONGER PROBLEMS

7.11. **(a)** Show that

$$\frac{1}{2}\left[\mathbf{u} - \mathbf{n}\right]^T \left[\mathbf{u} - \mathbf{n}\right] + \frac{\lambda}{2}\mathbf{u}^T \left[\mathcal{D}_x^T\mathcal{D}_x + \mathcal{D}_y^T\mathcal{D}_y\right] \mathbf{u}$$

is minimized by $\mathbf{u}$ such that

$$(\mathcal{I} + \lambda\mathcal{K})\,\mathbf{u} = \mathbf{n}$$

(b) How would you solve this linear system?

7.12. You wish to find the $\mathbf{u}$ that minimizes

$$\frac{1}{2}\left[\mathbf{u}-\mathbf{g}\right]^T\left[\mathbf{u}-\mathbf{g}\right]+\lambda\|\,\mathbf{u}\,\|_1.$$

(a) Show this isn't differentiable. Where is it not differentiable?
(b) Show the cost function is a sum over elements of $\mathbf{u}$ and so can be minimized term by term.
(c) Show that if $u_i > 0$, the term $(1/2)(u_i - g_i)^2 + \lambda|\,u_i\,|$ is differentiable in u_i.
(d) Use this to show if $|\,g_i\,|$ is sufficiently large, then

$$u_i = \frac{g_i}{1+\lambda}$$

(e) Show that if $0 < g_i < \lambda$ then $u_i = 0$. Do this by showing if $u_i = 0$, then the cost will be $\lambda^2/2$, but if $u_i = \epsilon > 0$ where ϵ is small, the cost will be $(1/2)(\lambda^2 + \epsilon^2)$.
(f) Show that if $-\lambda < g_i \leq 0$ then $u_i = 0$

7.13. Show that the $\mathbf{u}$ that minimizes

$$C(\mathbf{u}) = (\mathcal{B}\mathbf{u}-\mathbf{b})^T(\mathcal{B}\mathbf{u}-\mathbf{b}) + \lambda\mathbf{u}^T\mathbf{u}$$

is obtained by solving

$$(\mathcal{B}^T\mathcal{B}+\lambda\mathcal{I})\mathbf{d} = \mathcal{B}^T\mathbf{b}.$$

(a) Can $\mathcal{B}^T\mathcal{B}+\lambda\mathcal{I}$ have an eigenvalue smaller than λ?

PROGRAMMING EXERCISES

7.14. Obtain a collection of at least 100 RGB images. You will use these to evaluate a method to choose the weight λ in weighted least squares, for denoising images with different noise models. The exercise assumes your images have pixel values in the range 0 to 255/256 (scale your images if they don't).
(a) Implement a weighted least squares denoiser. You should use a strong linear algebra API (I used NumPy), and use its sparse matrix procedures. Your denoiser should accept a noisy image and a value of λ and produce a denoised version.
(b) Set up a range of at least 5 different values of λ (typically, a geometric series like $0.001, 0.005, 0.025, 0.125, 0.625$ is appropriate). Split your images into a training set (80%) and a validation set (20%). Use the training set to find the value of λ that gives the best average PSNR for images corrupted with additive gaussian noise with $\sigma = 0.01$, $\sigma = 0.1$ and $\sigma = 0.3$. How well does the training set PSNR predict the validation set PSNR? Does the same λ work for different values of σ.
(c) Repeat the previous experiment using the following noise model. At each pixel, with probability $p/2$ the R, G, and B values are flipped to 255/256; with probability $p/2$, to zero; otherwise, it is left alone. Use $p = 1/900$, $p = 1/100$ and $p = 1/25$.

7.15. Obtain a collection of at least 10 RGB images and reproduce the effects of Figure 7.1. Divide the RGB space into 4096 boxes ($16 \times 16 \times 16$) and count the pixels into these boxes. This is a color histogram. Use this histogram as a way to identify colors that are "rare" in the original images but appear in the versions with one smoothed component.
(a) Does smoothing one component produce rare colors?

(b) Can you find another method to evaluate the color effects produced by smoothing just one component?

(c) Is it always the case that smoothing G produces more significant color effects than smoothing R or B?

(d) Check your method(s) for identifying color effects by comparing what happens when you smooth one of RGB and what happens when you smooth one of LAB (which should produce less pronounced effects).