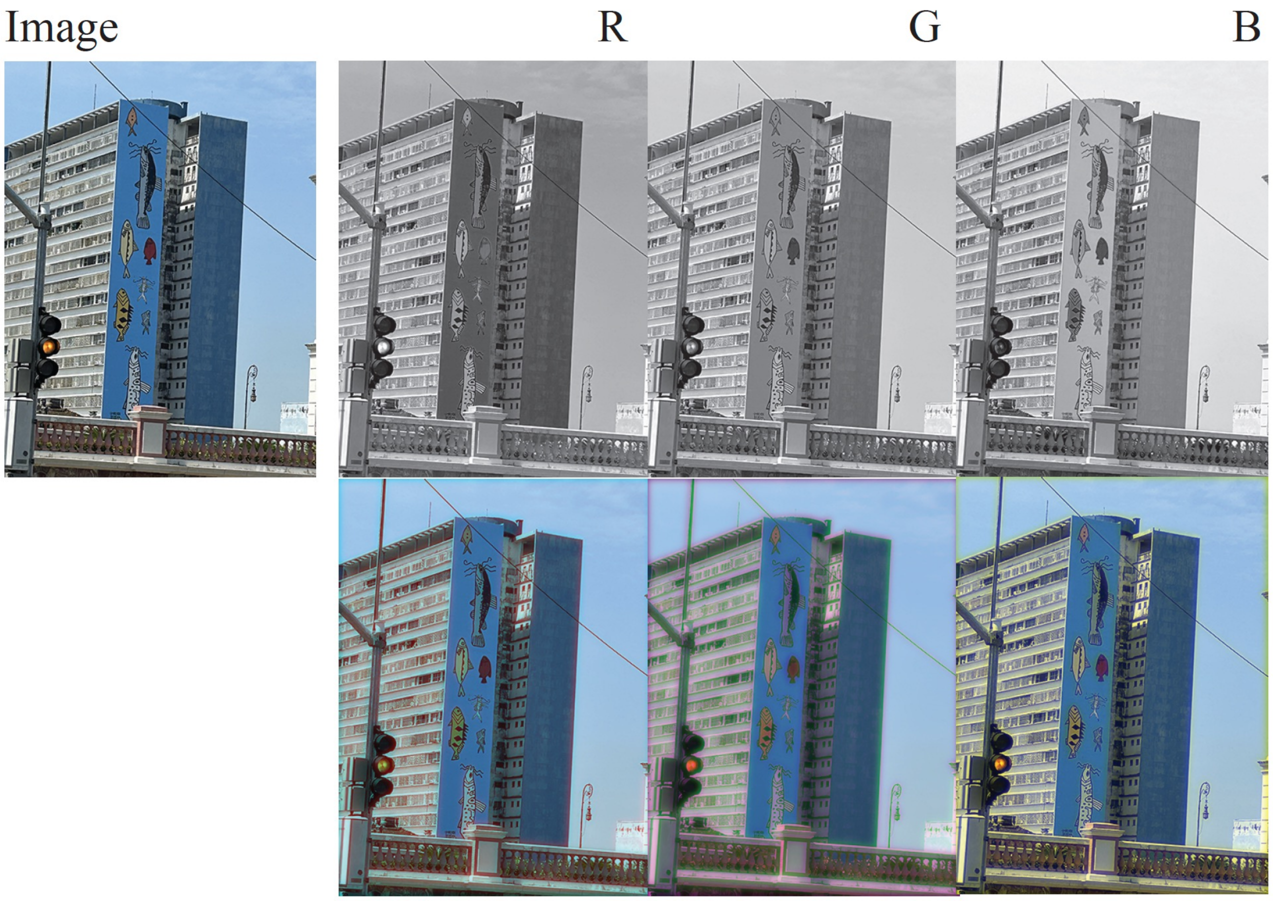
Denoising Images using Optimization

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.

Color images – R, G, and B are strongly correlated



Color images – R, G, and B are strongly correlated

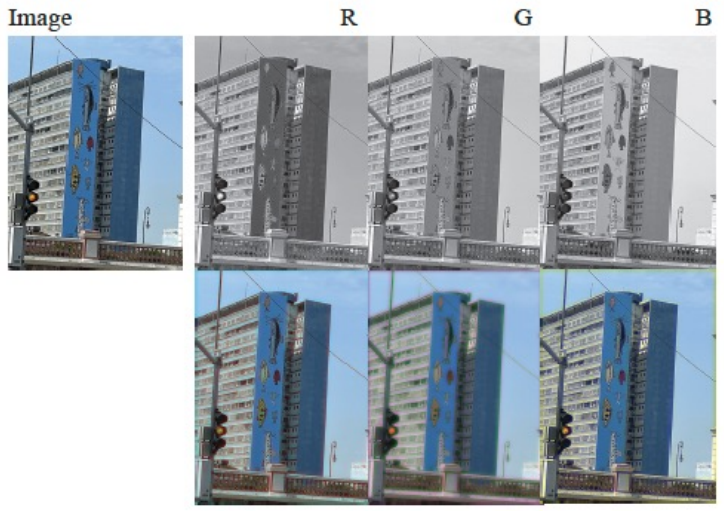
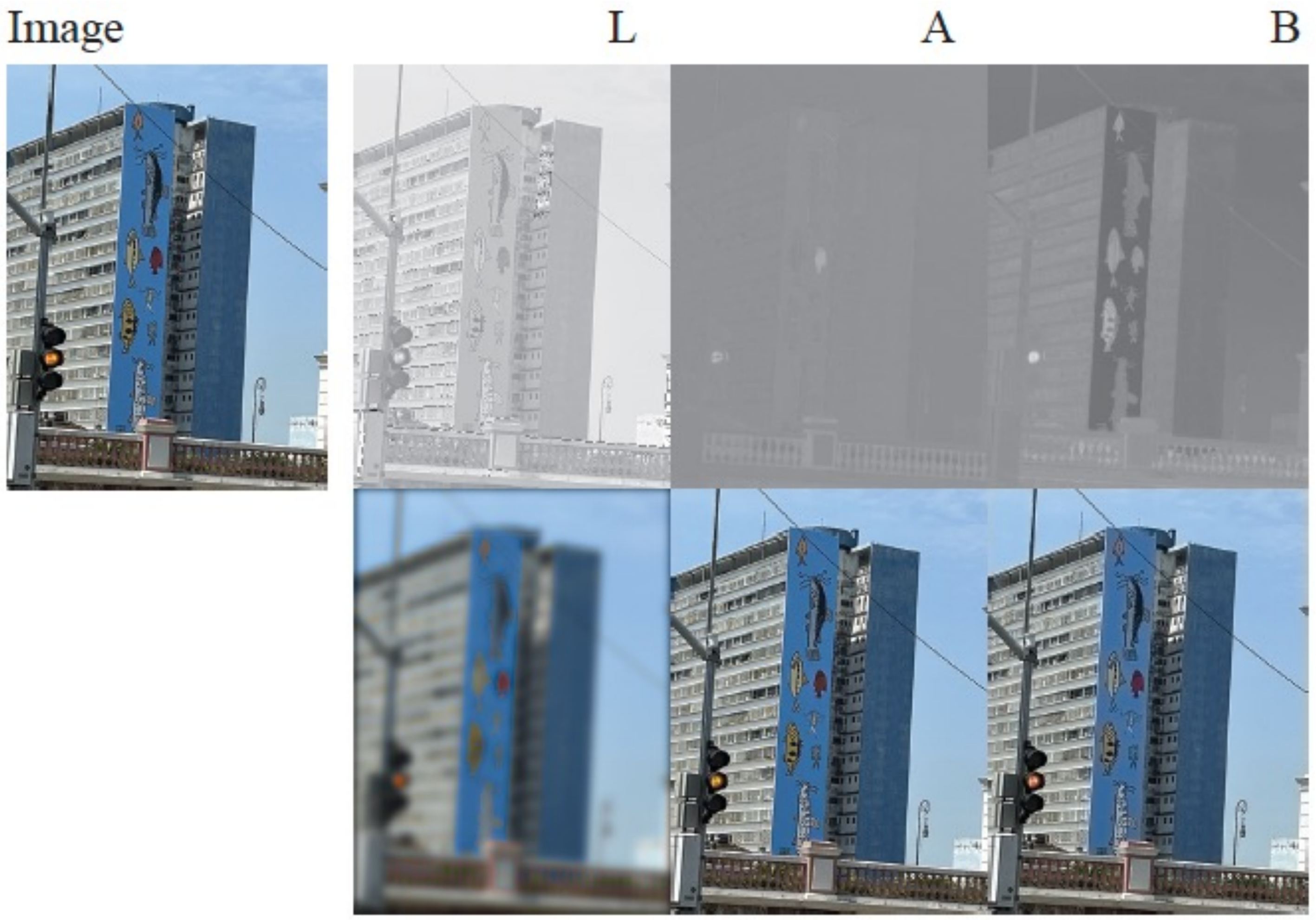


FIGURE 7.1: *RGB color components are heavily correlated, as you can see by looking at images where only one component has been smoothed.. The* **top row** *shows the R, G, and B components of the color image at the* **left**. *The* **bottom row** *shows color images obtained by smoothing one component, 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.*

LAB and smoothing



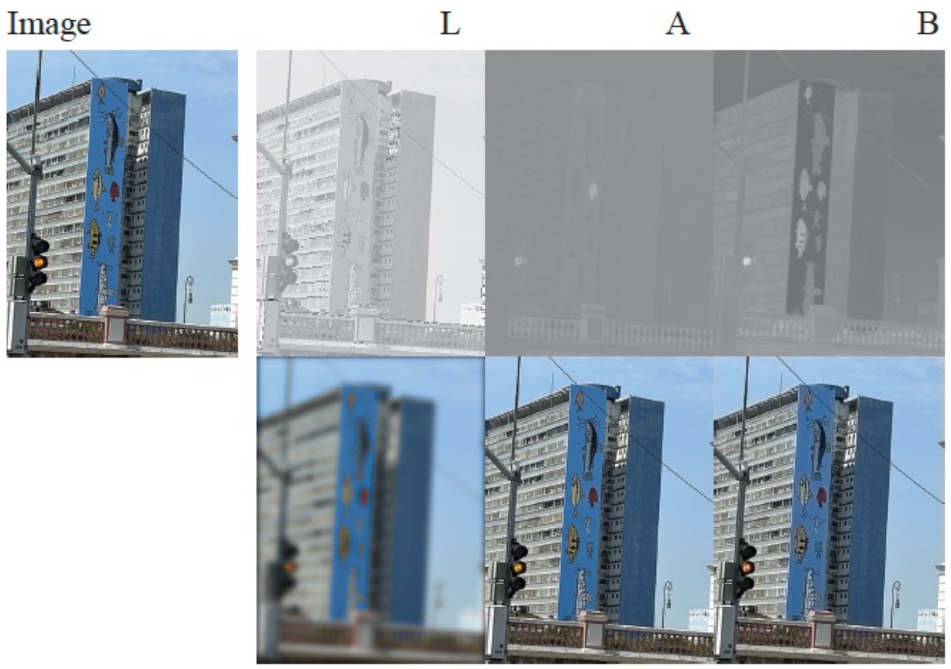


FIGURE 7.2: *Decorrelating the components of a color image before smoothing is important, but one does not need to do this on a per-image basis. The* **top row** *shows the L, A, and B components for this image on the* **right**. *Because these components can be negative, they have been scaled and shifted 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.*

This means you can

- Convert to LAB
- Apply sophisticated denoising to L
- Smooth A, B
- Convert back

Evaluation: PSNR

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). You need to know $\mathcal{C}$ to compute the PSNR, so you can only use PSNR to evaluate if you know the right answer. In some applications, versions of the original image that are uniformly

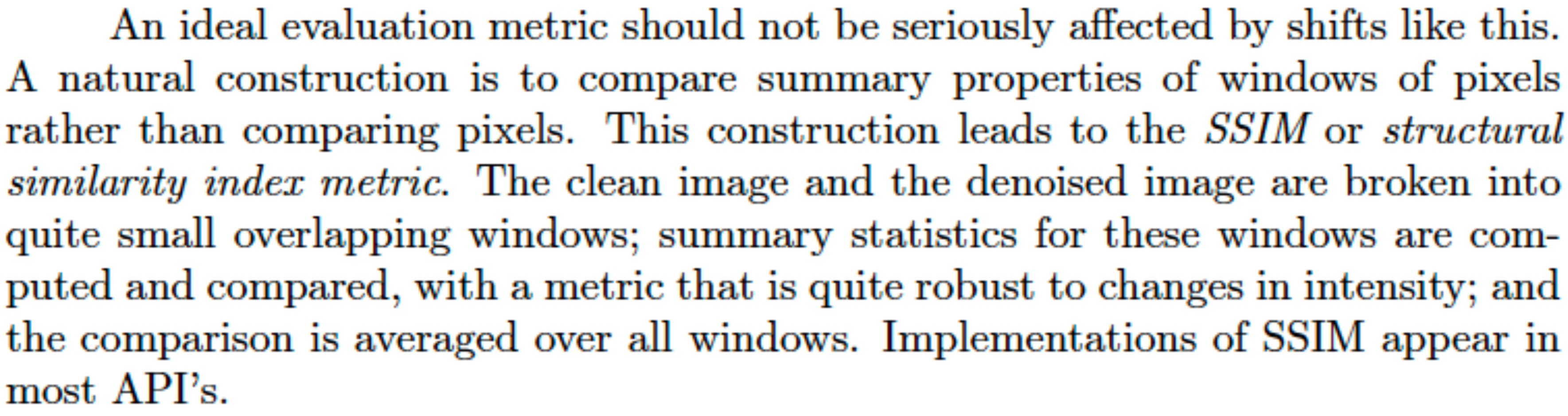
Scaled PSNR

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

$$\mathrm{psnr}(\mathcal{D},\mathcal{C}) = 20\log\frac{\max\limits_{ij}\ \mathcal{C}_{ij}}{\min\limits_{s}\ \sqrt{\sum_{ij}\left(s\mathcal{D}_{ij}-\mathcal{C}_{ij}\right)^2}}.$$

SSIM

PSNR doesn't account for small shifts, etc



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.

LPIPS

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 15 and 16. The reference Implementation of LPIPS is at `https://github.com/richzhang/PerceptualSimilarity`, and many APIs offer LPIPS evaluation.

Denoising by weighted least squares - I

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

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

(the ssd of Section 3.4.2). A good reconstruction could smooth the image over quite long scales in regions where $\mathcal{C}$ is constant. The reconstruction must preserve edges, so the smoothing would need to be over very short scales at edge points. Ideally, smoothing would be along an edge rather than across it. But $\mathcal{C}$ isn't known (otherwise there would be nothing to do). All this suggests that the penalty function needs to look at gradients in $\mathcal{D}$.

Denoising by WLS - II

Straighten

- noisy image N into vector n
- reconstructed image U into vector u

Cost becomes

$$[\mathbf{u} - \mathbf{n}]^T [\mathbf{u} - \mathbf{n}]$$

Denoising by WLS - III

Idea:

reconstructed image should have large gradients only where there is strong evidence in support

(version of ”pixels are like their neighbors”)

Strong evidence:

Use DOG filters to get smoothed gradient of noisy image

Where this is big, gradients in reconstruction should be cheap

Denoising by WLS - IV

Differentiation is linear, so can write matrices so that gradient of u is given by

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

What comes out is stacked x and y derivatives

Denoising by WLS -V

Now write $\mathcal{A}_x(\mathbf{n})$, $\mathcal{A}_y(\mathbf{n})$ for diagonal matrices of weights obtained from the original image. Because these matrices are diagonal, think of them as producing pixel by pixel weights on the cost of a derivative in $\mathcal{D}$. So 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.

Weights could be

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

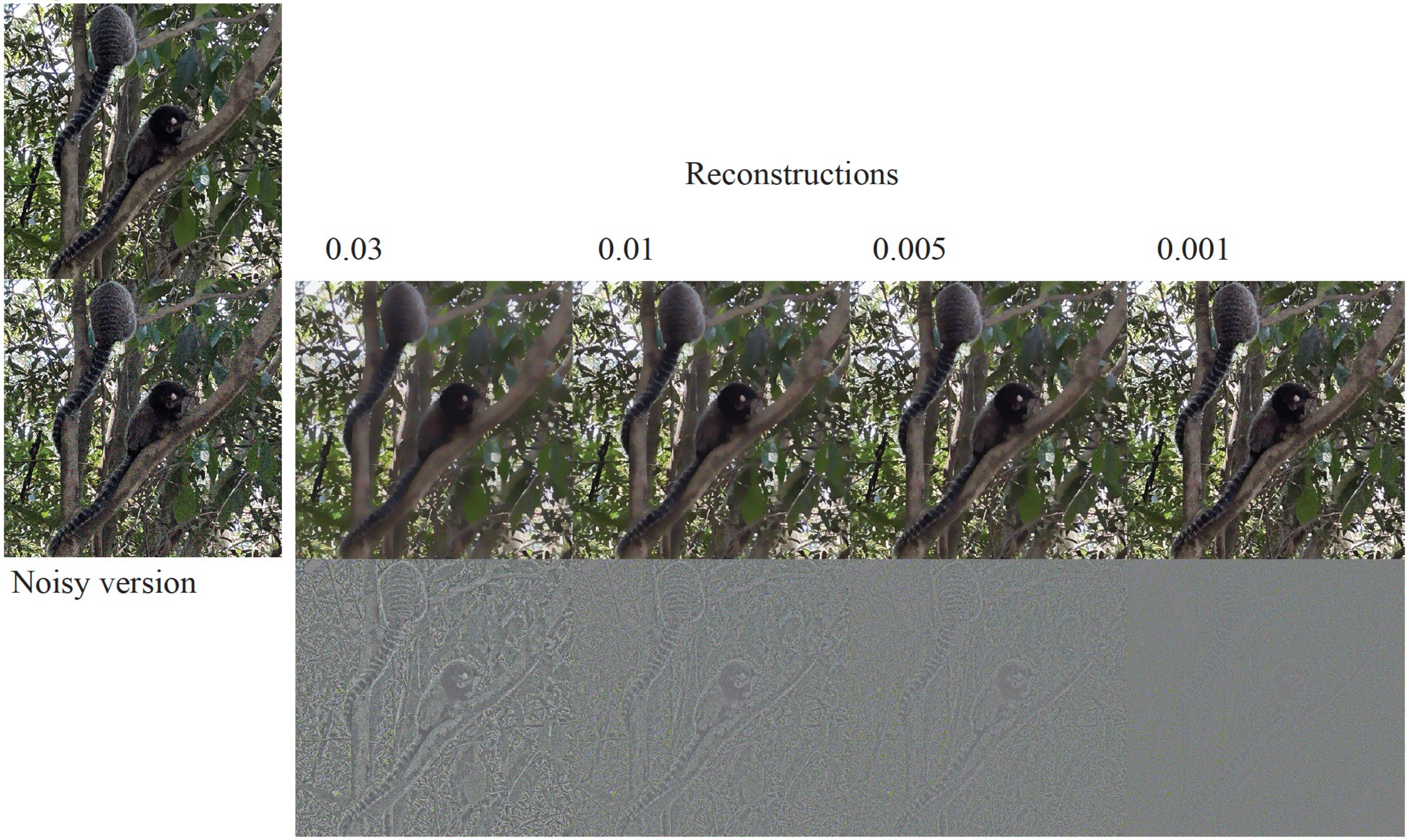
Where w_i is either x or y derivative at i'th location, eps is small

Denoising by WLS - VI

$$\underset{\mathbf{u}}{\operatorname{argmin}} \quad \underbrace{[\mathbf{u}-\mathbf{n}]^T[\mathbf{u}-\mathbf{n}]}_{\text{Be close to noisy image}} + \underbrace{\lambda \mathbf{u}^T\left[\mathcal{D}_x^T\mathcal{A}_x^T\mathcal{A}_x\mathcal{D}_x + \mathcal{D}_y^T\mathcal{A}_y^T\mathcal{A}_y\mathcal{D}_y\right]\mathbf{u}}_{\text{Have big derivatives only if good evidence}}$$

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

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


Original
Reconstructions
0.03
0.01
0.005
0.001
Noisy version
Residuals

Norms - I

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

Q: how to measure the ”size” of the penalty?

$$\downarrow$$

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

Norms -II

The *L2 norm*, defined by

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

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

$\uparrow$

This is squared L2 norm (of what?)

Norms - III

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

Weighted least squares penalized the squared L2 norm of the weighted gradient. Generally, a vector with small L2 norm can have many small, but non-zero, elements. This is because the square of a small number is very small, and the sum of many very small numbers is still small. The weights in weighted least squares tend to mitigate this, because small gradients have large penalty weights. **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.

The L1 Norm

An alternative is to penalize the *L1 norm* of the gradients. The L1 norm of a vector $\mathbf{v}$ is defined by

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

Behavior of L2 norm

A vector with small L1 norm will tend to have zero elements. You can see this by comparing two cases. Write

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

and notice that the $\mathbf{u}$ that minimizes $C_2(\mathbf{u})$ is

$$\frac{1}{1+\lambda}\mathbf{g}.$$

Notice – even if lambda is very big, g ISN'T zero
(it's just small)

Behavior of L1 norm

Now write

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

and think about the $\mathbf{u}$ that minimizes $C_1(\mathbf{u})$. The penalty term isn't differentiable, which creates some inconvenience, but it is a sum over elements of $\mathbf{u}$. Now consider the i'th element of $\mathbf{u}$. If g_i is sufficiently large, then it is easy to show that

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

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

The L1 norm encourages g to have zeros in it!

Total variation denoising - I

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

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

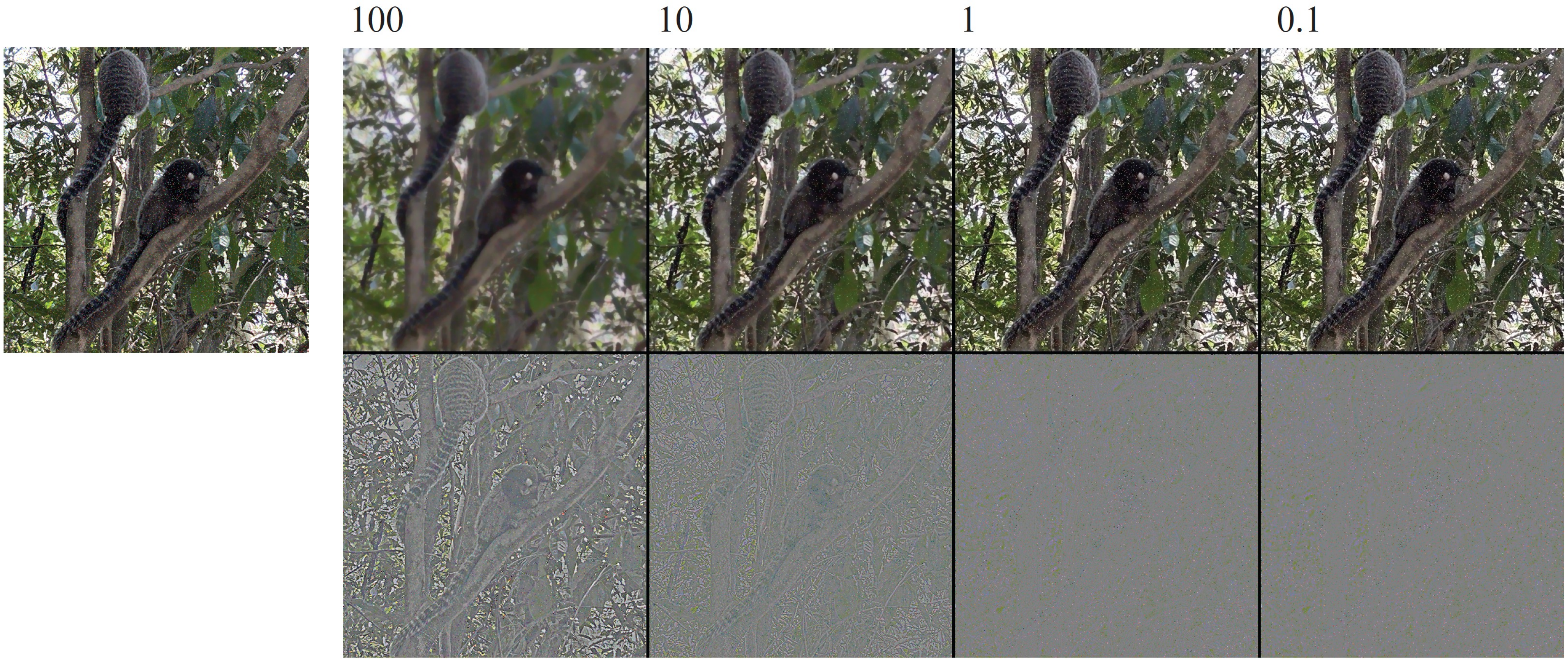
Note this cost function isn't differentiable, but it is convex. The optimization problem for this cost function is well understood, and is relatively easily managed (though beyond our scope). However, you should notice that the penalty encourages zeros in the x and y components of the gradient, which isn't necessarily the same as zero gradients. 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.

Total variation denoising - II

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

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

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.


100
10
1
0.1

Deblurring - I

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 example, take an image and blur it. The result isn't an image, but it is quite close to one. Recall from Section 41.2 that blurring is a linear operation. Write **t** for the true image in vector form, **d** for the deblurred estimate in vector form, **b** for the observed image in vector form, and $\mathcal{B}$ for the linear operator that blurs. Assume $\mathcal{B}$ is known, at least for the moment (**exercises**). Notice **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 **d** that minimizes

$$(\mathcal{B}\mathbf{d} - \mathbf{b})^T (\mathcal{B}\mathbf{d} - \mathbf{b})$$

which would involve solving

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

Deblurring - II

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

But this has some really big eigenvalues!

Regularization

There is a traditional procedure to handle very small eigenvalues in a matrix, known as *regularization*. One seeks a minimum of

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

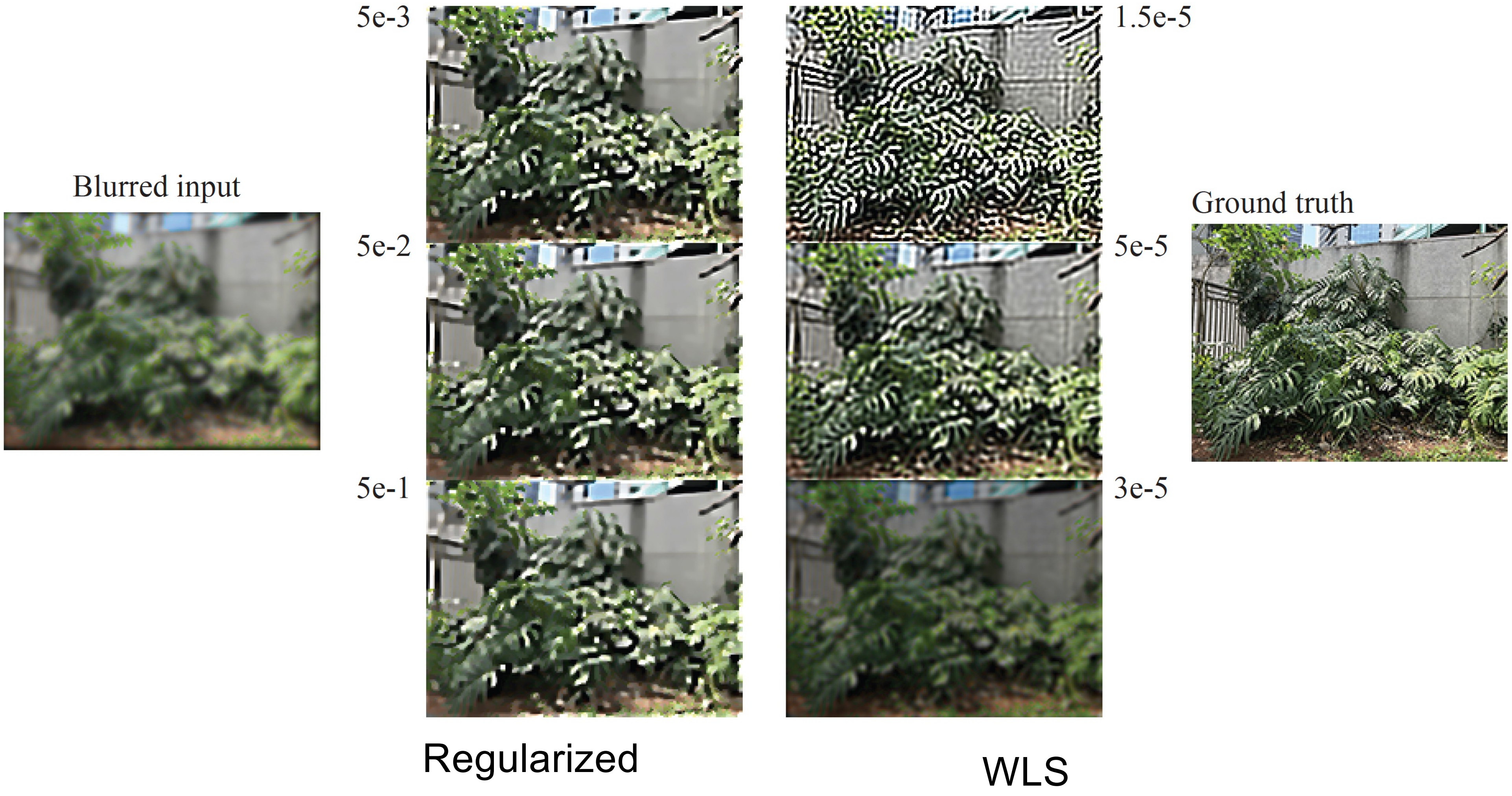
by solving

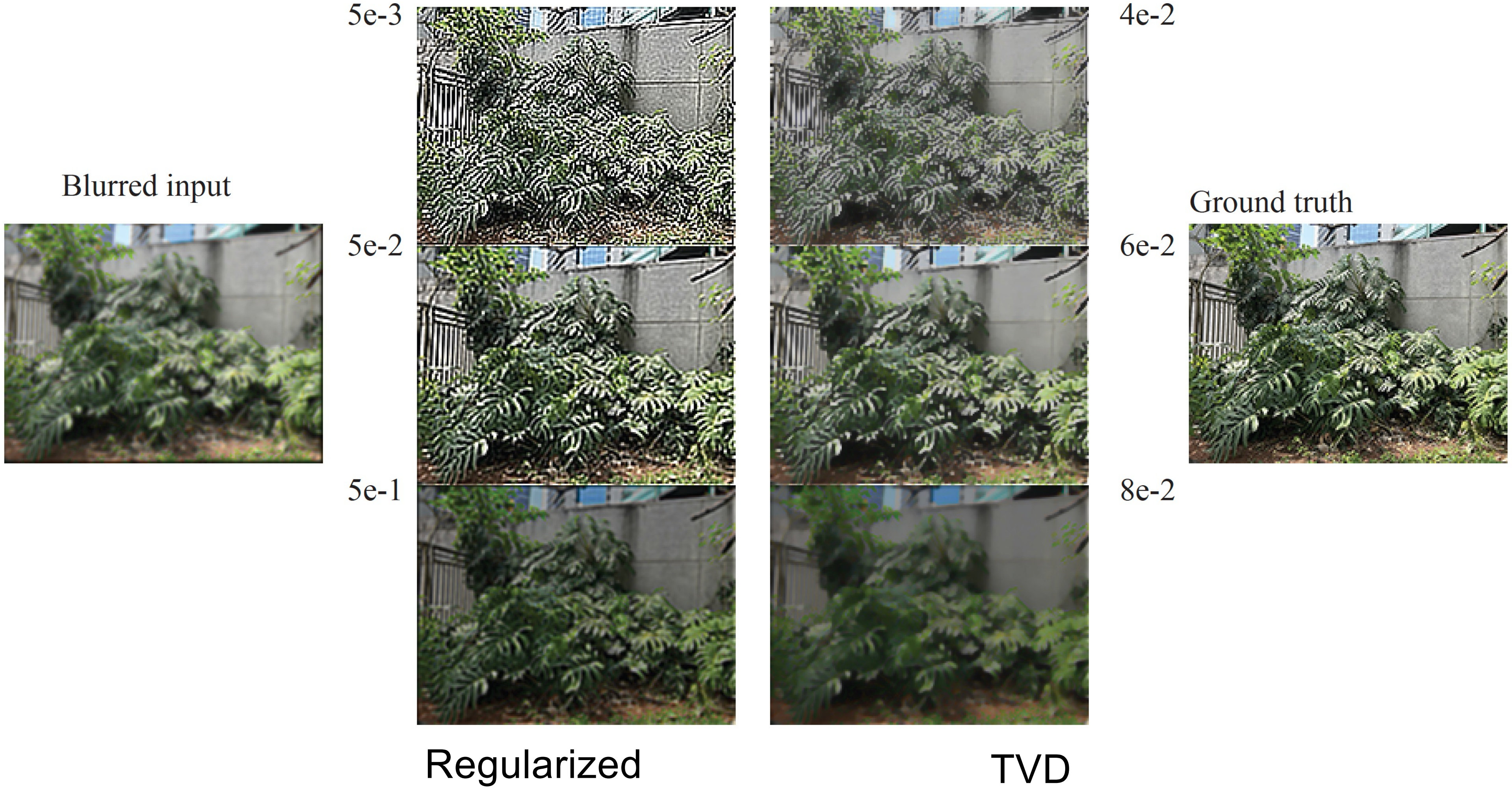
$$(\mathcal{B}^T \mathcal{B} + \lambda \mathcal{I})\mathbf{d} = \mathcal{B}^T \mathbf{b}$$

MLS and TVD

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

Penalty term we used before for MLS or TVD


Blurred input
5e-3
5e-2
5e-1
Regularized
1.5e-5
5e-5
3e-5
WLS
Ground truth


Blurred input
5e-3
5e-2
5e-1
Regularized
4e-2
6e-2
8e-2
TVD
Ground truth

Adding detail with WLS

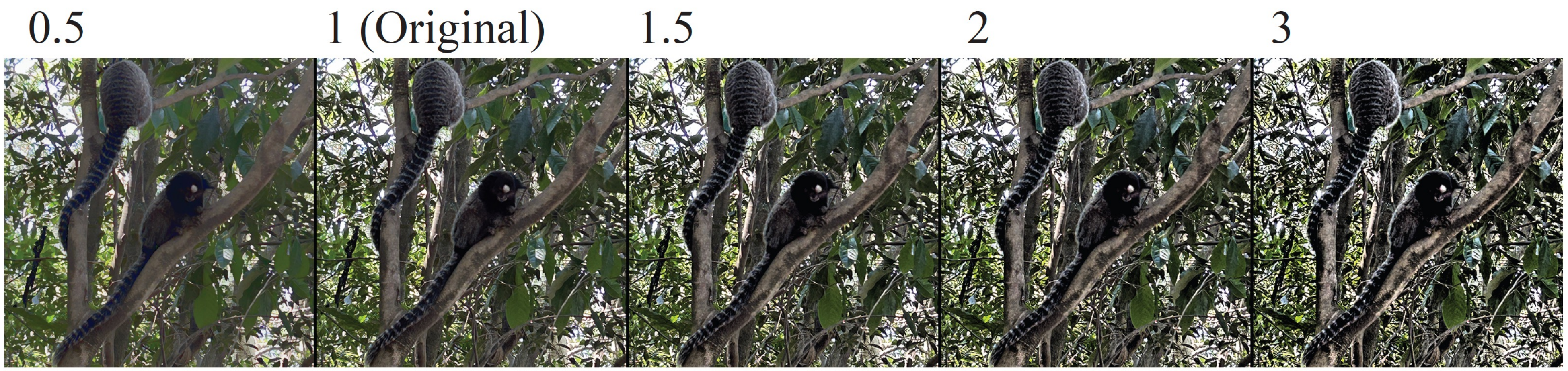
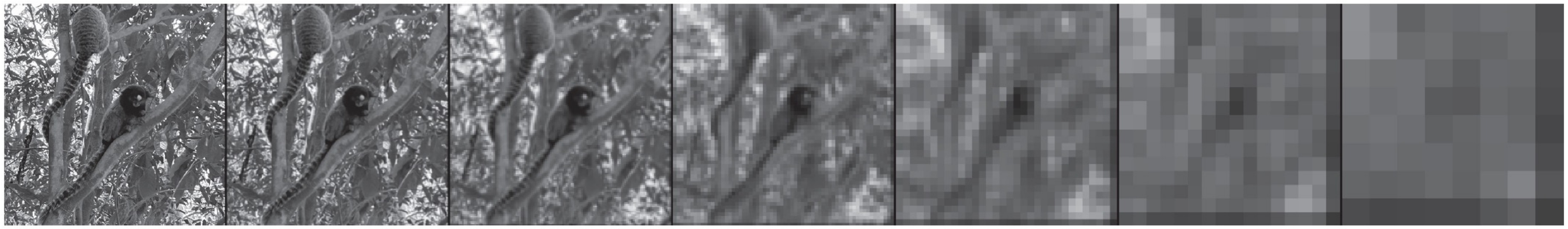


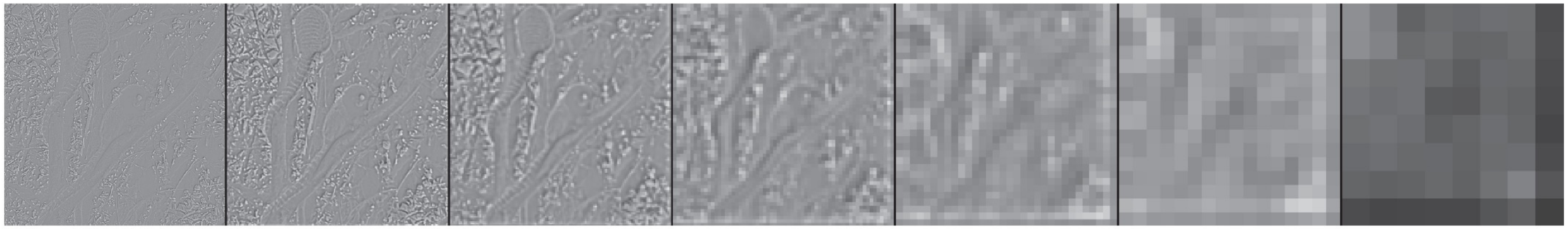
FIGURE 7.11: *For image $\mathcal{I}$, write $W(\mathcal{I})$ for the result of applying weighted least squares. Then these images are $W(\mathcal{I}) + \lambda(\mathcal{I} - W(\mathcal{I}))$ for different values of λ. This deemphasizes detail (**left**) for $\lambda < 1$ and emphasizes detail (**right**) for $\lambda > 1$.* Image credit: *Figure shows my photograph of marmosets in Sao Paulo.*

Trend and detail representations

Gaussian pyramid



Laplacian pyramid



MLS pyramid