Denoising considerations

D.A. Forsyth,

University of Illinois

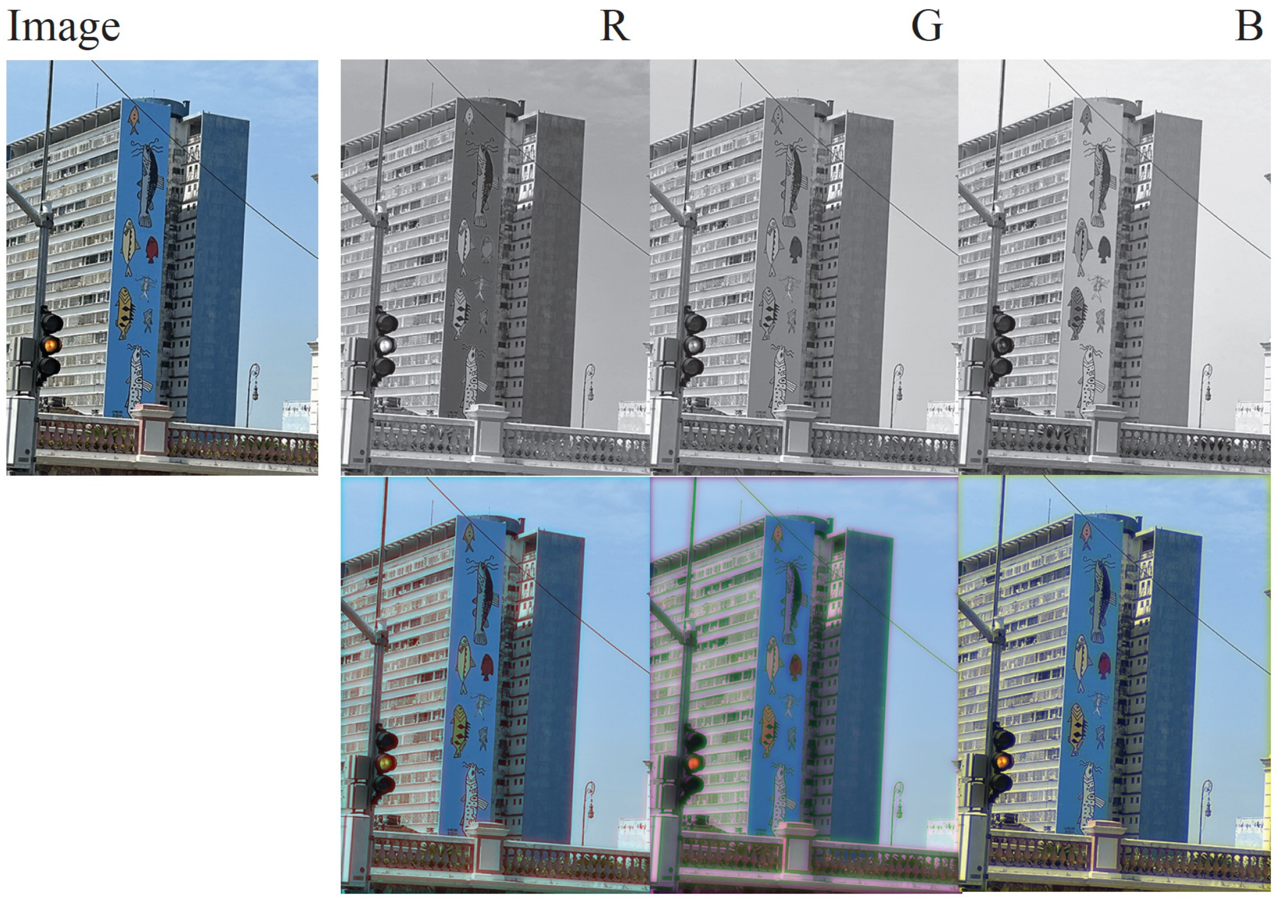
Denoising Images using Optimization

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

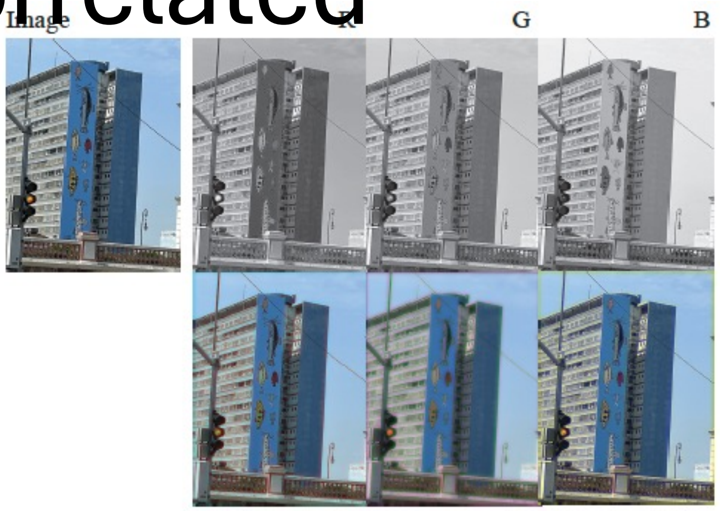
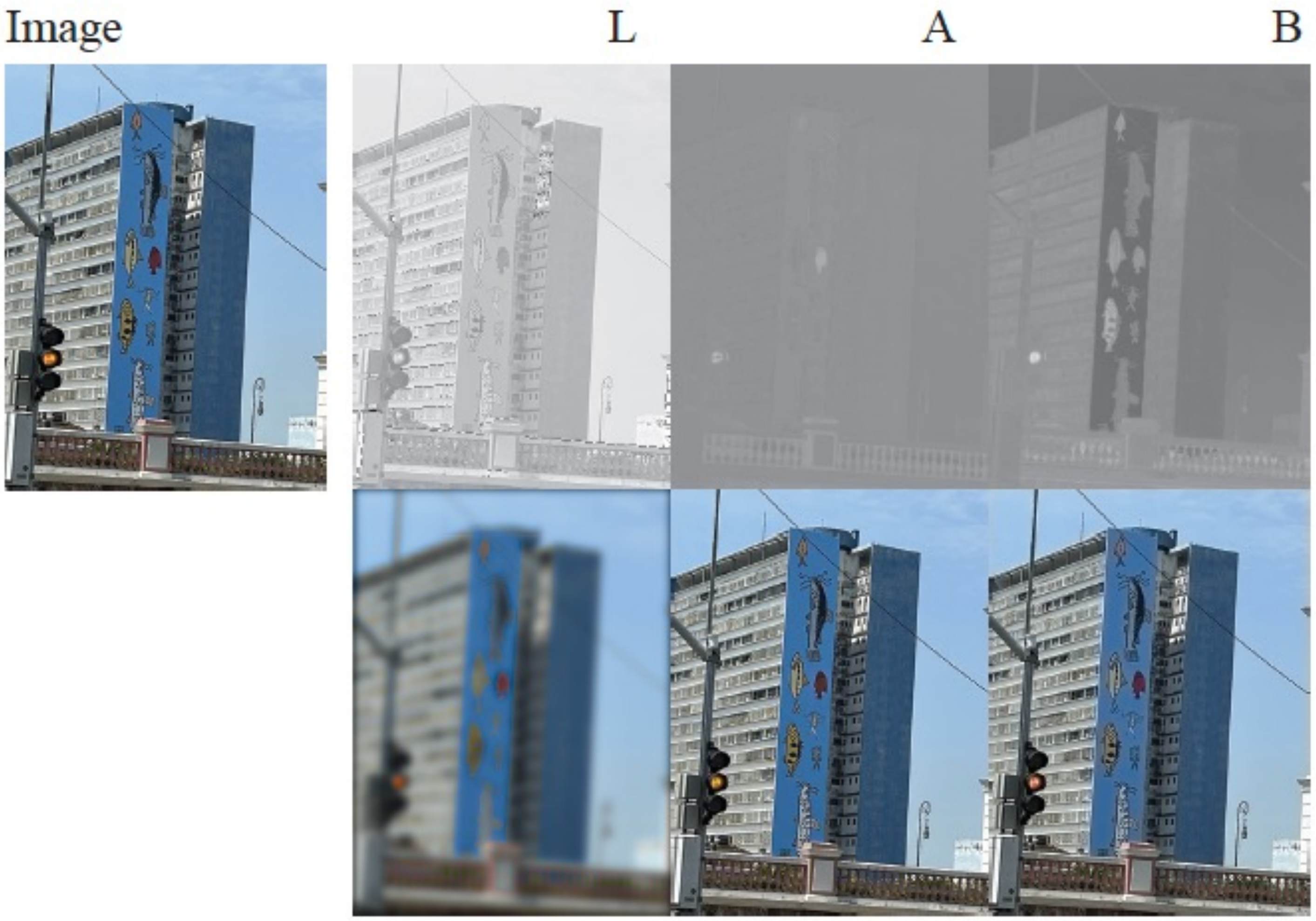


FIGURE 7.1: *RGB color components are heavily correlated, as you can see by looking at images where only one component has been smoothed.. The* **top row** *shows the R, G, and B components of the color image at the* **left**. *The* **bottom row** *shows color images obtained by smoothing one component,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



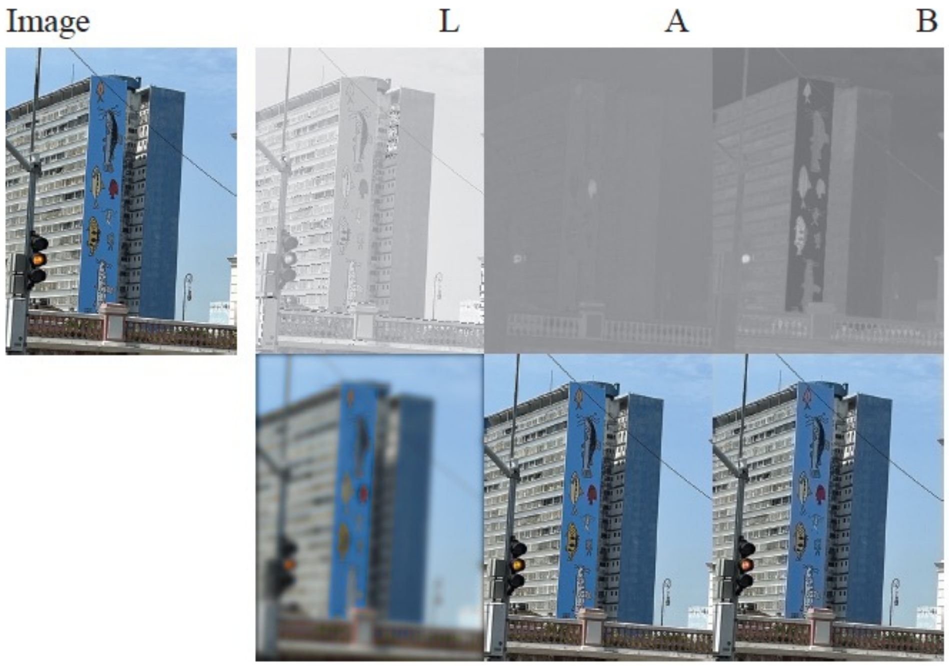


FIGURE 7.2: *Decorrelating the components of a color image before smoothing is important, but one does not need to do this on a per-image basis. The* **top row** *shows the L, A, and B components for this image on the* **right**. *Because these components can be negative, they have been scaled and shifted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

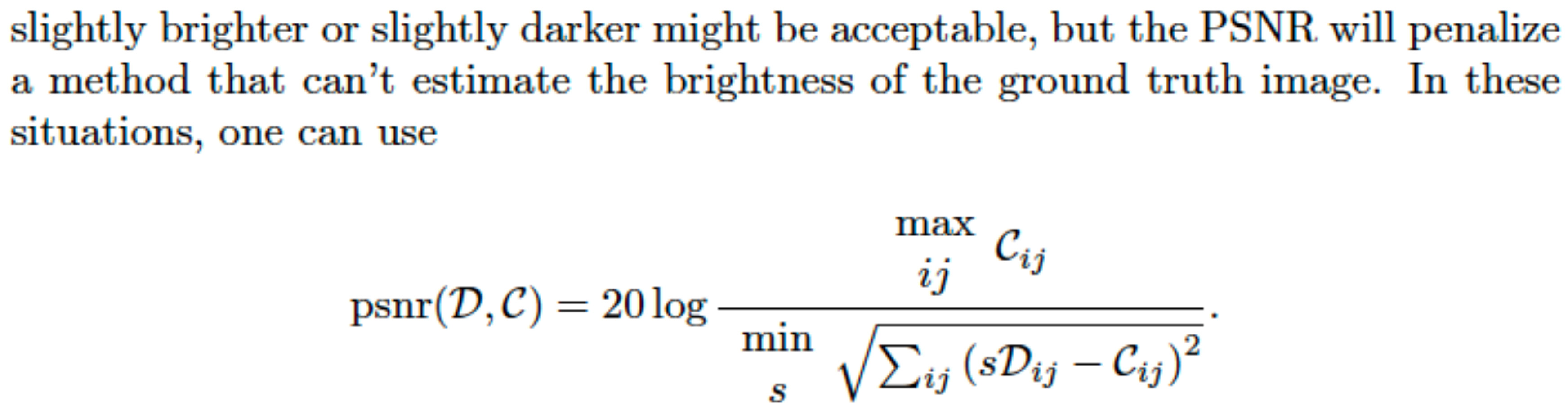
Evaluation: PSNR

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

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

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

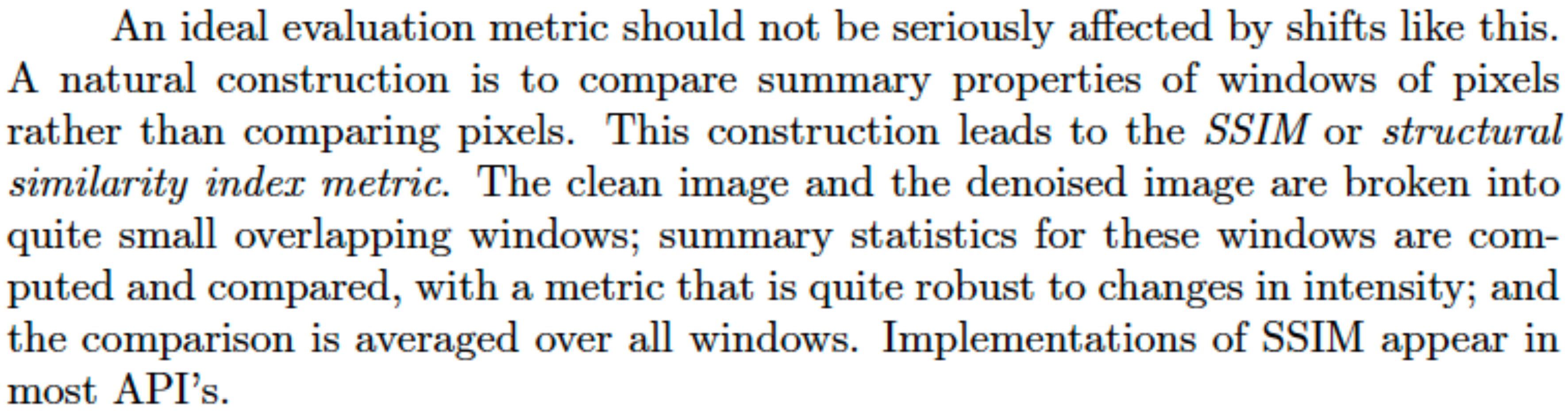


slightly brighter or slightly darker might be acceptable, but the PSNR will penalize a method that can't estimate the brightness of the ground truth image. In these situations, one can use

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

SSIM

- PSNR doesn’t account for small shifts, etc

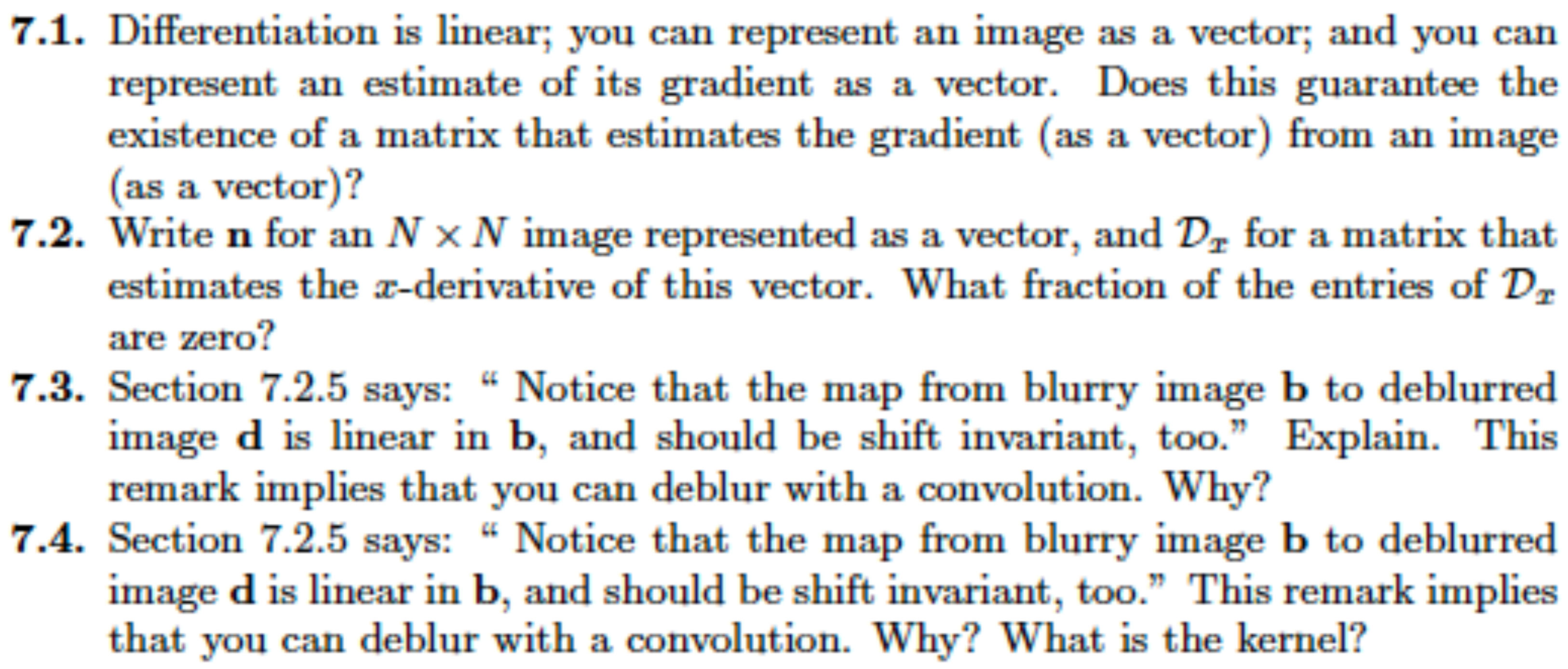


An ideal evaluation metric should not be seriously affected by shifts like this. A natural construction is to compare summary properties of windows of pixels rather than comparing pixels. This construction leads to the *SSIM* or *structural similarity index metric*. The clean image and the denoised image 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.

Think about this…



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 **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 **b** to deblurred image **d** is linear in **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 **b** to deblurred image **d** is linear in **b**, and should be shift invariant, too." This remark implies that you can deblur with a convolution. Why? What is the kernel?