Inpainting using patches

D.A. Forsyth

University of Illinois at Urbana Champaign

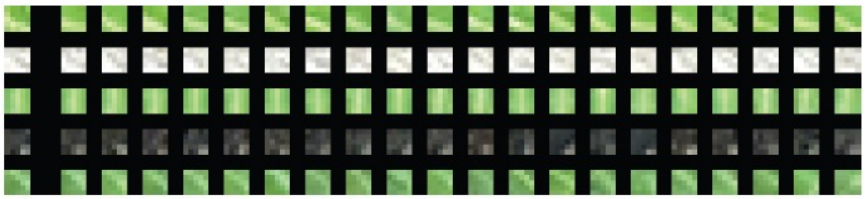
Another property of images

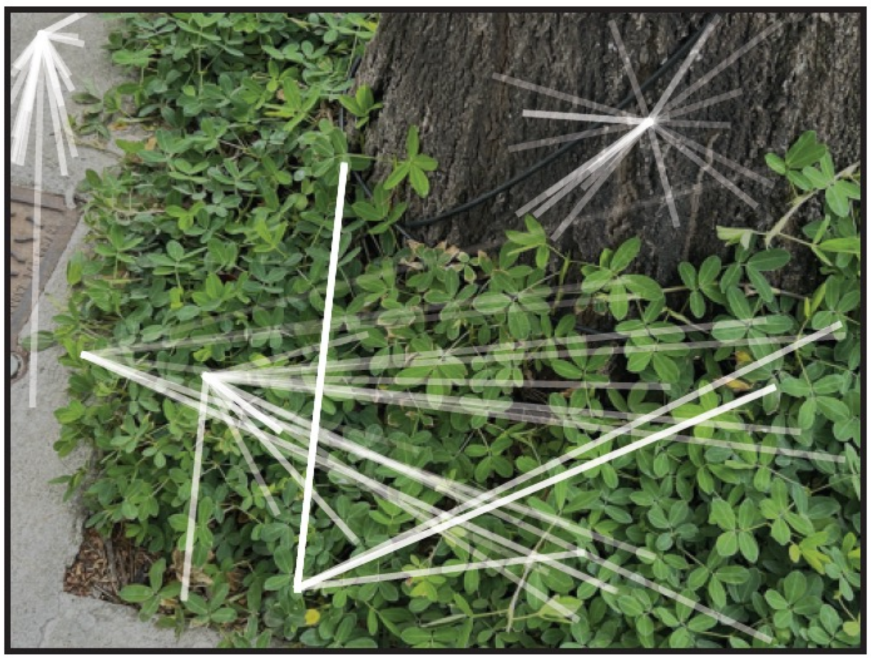
- They’re built out of patches that tend to repeat

- So there is often a patch some distance away that looks like the patch you’re looking at
  - even for quite big patches

Denoising images using patches

5 x 5





Images tend to be made of repeating patches

Denoising images using patches

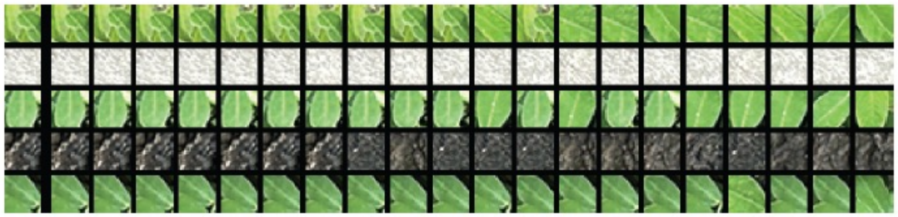
11 x 11

Images tend to be made of repeating patches

This is true even for quite big
(so quite distinctive)
patches

Denoising images using patches

21 x 21



Images tend to be made of repeating patches

This is true even for quite big
(so quite distinctive)
patches

Inpainting

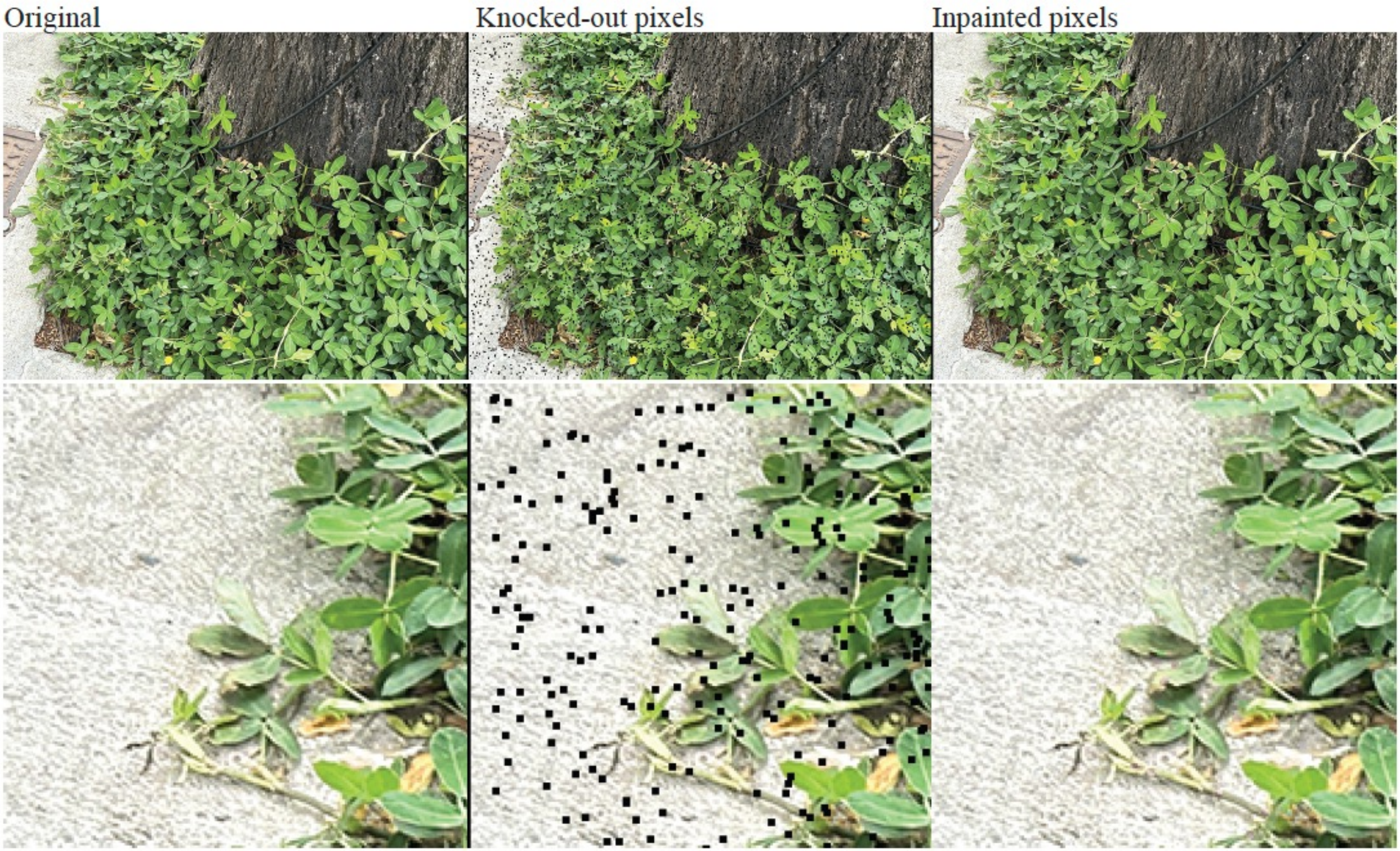
- Simplest case:
  - some fraction of pixels has been “knocked out” at random
  - (perhaps set to zero)
  - and you know which pixels

- Fix:
  - take window around knocked out pixel
  - find closest match in the image
  - take the center pixel from matching window

Original Knocked-out pixels Inpainted pixels

Inpainting for bigger knockouts

- Assume a kxk window gets ”knocked out” (k small, but k>1)
- Procedure works if you are careful about matching
- - eg don’t match to windows that have knocked out pixels


Original
Knocked-out pixels
Inpainted pixels

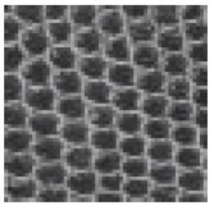
Incremental inpainting

Now imagine that the process that knocks out pixels doesn't just choose pixels at random, but has some some kind of spatial structure. For example, you might have an image with writing on it, and want to replace the writing. Alternatively, the image might have one more more large holes in it.

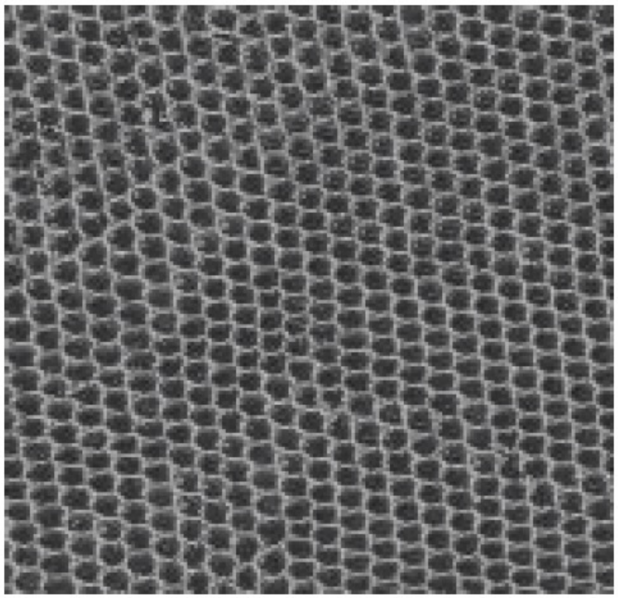
The pixel inpainting procedure above will work, but some details need to change. When isolated pixels are knocked out, you expect that the patch around the pixel is known. If the image has a large hole in it, this no longer applies. Fixing a pixel requires you have at least some known pixel values close to it. Choose such a pixel, and match the patch using the known pixels only. You can do this with a mask that zeros the contribution of knocked out pixels to the SSD. This produces a pool of matches. Now estimate the value of the pixel using this pool. For the moment, choose the center of the best match. Place this value in the image, and you now have an image with a slightly smaller hole in it, so you should be able to find more candidate pixels for replacing. In this *incremental reconstruction* approach, the order in which you visit pixels and the size of the patch becomes important and can quite strongly affect the result.

Incremental inpainting makes big images from small

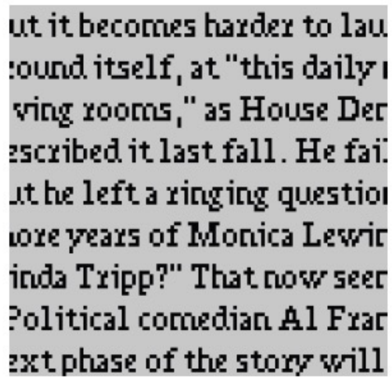
Example



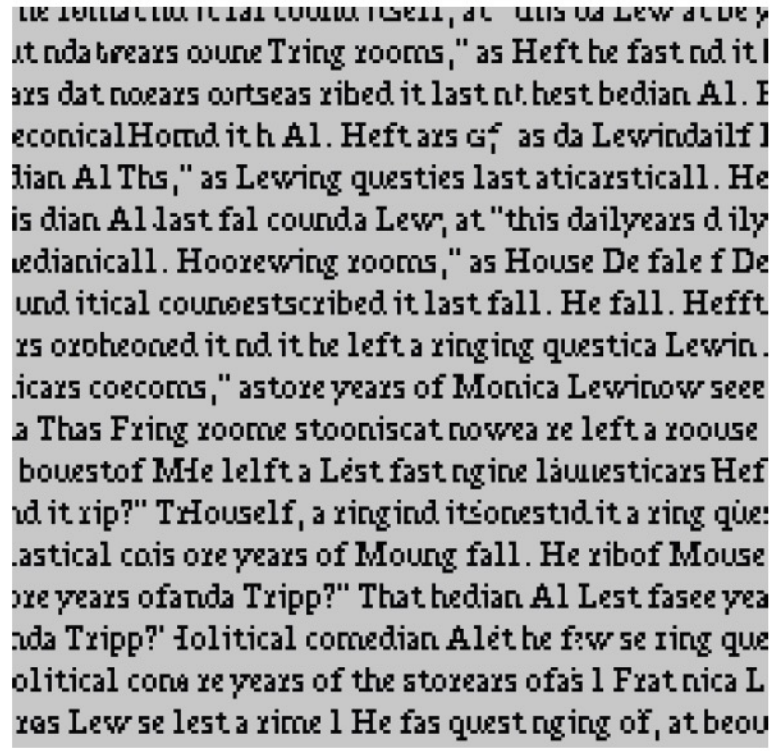
Synthesized



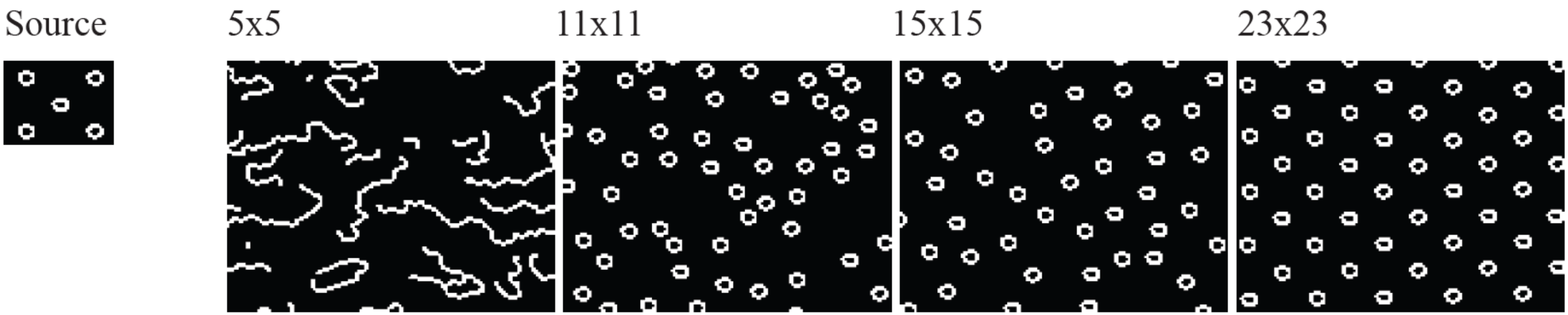
Example



Synthesized



The size of the patch matters



But this is a weird noise model…

- You *know* which pixels are wrong
- What about additive gaussian noise?
- The principle here is that there are other patches in the image that are “like” the patch you are interested in
- Idea – estimate pixel value using
  - a weighted sum *over all patches* weighted by similarity
  - of patch to neighborhood around pixel

Non-local means

The approach is easily formalized. Write $K(\mathbf{p}_{ij}, \mathbf{p}_{uv})$ for a function that compares an image patch $\mathbf{p}_{ij}$ around the i, j'th pixel with the image patch around the u, v'th pixel. This function should be large when the patches are similar, and small when they are different. A useful estimate of the pixel value $\mathbf{x}_{ij}$ at i, j is then

$$\sum_{uv \in \text{image}} \frac{K(\mathbf{p}_{ij}, \mathbf{p}_{uv})\mathbf{x}_{uv}}{\sum_{kl \in \text{image}} K(\mathbf{p}_{ij}, \mathbf{p}_{lm})}.$$

Notice that the weights sum to one. The estimate clearly depends quite strongly on the choice of K.

The gaussian Kernel: One natural choice uses SSD between patches. Write $\mathrm{NSSD}(\mathbf{p}_{ij}, \mathbf{p}_{uv}))$ for the sum of squared differences between the two patches normalized to deal with the number of pixels in the patch (exercises), write σ for some scale chosen to work well, note that I have suppressed the size of the patch, and

use

$$K_{\mathrm{NSSD}}(\mathbf{p}_{ij}, \mathbf{p}_{uv}) = e^{\frac{\left(-\mathrm{NSSD}_{(\mathbf{p}_{ij}, \mathbf{p}_{uv})}\right)}{2\sigma^2}}.$$

The method described here is sometimes known as *non-local means.* As described, it is very slow (quadratic in the number of pixels). Methods to speed it up remain difficult, and are out of scope (**exercises**). As Figure 9.8 shows, non-local means can suppress a great deal of noise without blurring edges.

Additive gaussian noise, 0.1 | gaussian smoothing, σ=2 | gaussian smoothing, σ=4

PSNR=18.4 | PSNR=18.0

LAB

3x3 PSNR: 24.7

7x7 PSNR: 21.5

RGB

3x3 PSNR: 24.5

7x7 PSNR: 25.3

The Bilateral Filter

The gaussian kernel weights down patches that are different from the target patch, but pays no attention to the distance between patches. A natural extension, known as the *bilateral filter*, downweights patches based on their distance. This gives

$$K_{\text{bilat}}(\mathbf{p}_{ij}, \mathbf{p}_{uv}) = e^{\frac{\left(-\text{NSSD}(\mathbf{p}_{ij}, \mathbf{p}_{uv})\right)}{2\sigma^2}} e^{\frac{\left(-\left[(i-u)^2+(j-v)^2\right]\right)}{2\sigma_d^2}}$$

where σ_d controls the rate at which a patches contribution falls off with distance. The bilateral filter admits significant speedups (**exercises**).

Things to think about

10.1. Do you expect to observe every possible set of RGB pixel values in images? why?

10.2. Why do you need a masked normalized SSD to inpaint missing regions?

10.3. Section 10.1.3 has: "Just tiling the texture won't work. The patches may not join up properly" – Explain.

10.4. You wish to inpaint an hole in an image. Why does the order you choose to fill in pixels matter?

10.5. You wish to inpaint an hole in an image. Suggest a good order in which to visit pixels to inpaint. Would this work for every case?

10.6. You want to increase the size of a patch of texture as in Section 10.1.3. Why is it important to choose from several patches at random?