CS448f: Image Processing For Photography and Vision

Denoising
How goes the assignment?
The course so far...

• We have a fair idea what image processing code looks like
• We know how to treat an image as a continuous function
• We know how to warp images
• What should we do next?
What are the big problems in Photography and Computer Vision, and how can Image Processing help?
Today: Denoising
How do we know a pixel is bad?

• It’s not like its neighbours
• Solution: Replace each pixel with the average of its neighbors
• I.e. Convolve by

\[
\begin{bmatrix}
1 & 1 & 1 \\
1 & 1 & 1 \\
1 & 1 & 1 \\
1 & 1 & 1 \\
\end{bmatrix}
\]
3x3 Rect Filter
Linear Filters

• Why should a far-away pixel contribute the same amount as a nearby pixel

• Gaussian blur:

\[
Out(\vec{x}) = \frac{\sum_{\vec{x}'} w(\vec{x}, \vec{x}') I(\vec{x}')}{\sum_{\vec{x}'} w(\vec{x}, \vec{x}')}
\]

\[
w(\vec{x}, \vec{x}') = e^{-|\vec{x}' - \vec{x}|^2 / 2\beta}
\]
Gaussian Blur
\[ \beta = 2 \]
beta = 5
Some neat properties:

- It’s radially symmetric and separable at the same time:
  \[ e^{-|\vec{x}' - \vec{x}|^2} = e^{-(x_0 - x'_0)^2} \cdot e^{-(x_1 - x'_1)^2} \]

- Its Fourier transform is also a Gaussian

- It’s not very useful for denoising
Linear Filters

- Convolve by some kernel
- Equivalent to multiplication in Fourier domain
- Reduces high frequencies
  - But we wanted those - that’s what made the image sharp!
  - You don’t always need the high frequencies. The first step in many computer vision algorithms is a hefty blur.
Probabilistically...

• What is the most probable value of a pixel, given its neighbors?
Probabilistically...

- This is supposed to be dark grey:
Let’s ignore color for now
And inspect the distribution
And inspect the distribution

Most probable pixel value
And inspect the distribution

mean = mode = median
Denoising
Denoising
The distribution
The distribution
The distribution

The mean
The distribution

The mode
The distribution

The median
Median Filters

• Replace each pixel with the median of its neighbours
• Great for getting rid of salt-and-pepper noise
• Removes small details
Gaussian Blur:
After Median Filter:
The distribution

- What is the most probable pixel value?
It depends on your current value

• The answer should vary for each pixel
An extra prior

• **Method 1) Convolution**
  – The pixel is probably looking at the same material as all of its neighbors, so we’ll set it to the average of its neighbors.

• **Method 2) Bilateral**
  – The pixel is probably looking at the same material as SOME of its neighbors, so we’ll set it to the average of those neighbors only.
The Bilateral Filter

• How do we select good neighbours?
• The ones with roughly similar brightnesses are probably looking at the same material.

\[
Out(\vec{x}) = \frac{\sum_{\vec{x}'} w(\vec{x}, \vec{x}') I(\vec{x}')} {\sum_{\vec{x}'} w(\vec{x}, \vec{x}')} \\

w(\vec{x}, \vec{x}') = e^{-|I(\vec{x}') - I(\vec{x})|^2 / 2\alpha} \cdot e^{-|\vec{x}' - \vec{x}|^2 / 2\beta}
\]
Gaussian Blur:
After Median Filter:
After Bilateral:
Dealing with Color

• It turns out humans are only sensitive to high frequencies in brightness
  – not in hue or saturation
• So we can blur in chrominance much more than luminance
Chroma Blurring

• 1) Convert RGB to LAB
  – L is luminance, AB are chrominance
    (hue and saturation)
• 2) Perform a small bilateral in L
• 3) Perform a large bilateral in AB
• 4) Convert back to RGB
After Regular Bilateral:
After Modified Bilateral:
Method Noise (Gaussian)
Method Noise (Bilateral)
Leveraging Similarity: Non-Local Means
What color should this pixel be?
What color should this pixel be?
Gaussian Blur: A weighted average of these: (all nearby pixels)
Bilateral filter
A weighted average of these:
(the nearby ones that have a similar color to me)
Non-Local Means
A weighted average of these:
(the ones that have similar neighbors to me)
Non-Local Means:

- For each pixel
  - Find every other (nearby) pixel that has a similar local neighborhood around it to me
  - Set my value to be the weighted average of those

\[
Out(\vec{x}) = \frac{\sum_{\vec{x}'} w(\vec{x}, \vec{x}') I(\vec{x}')}{\sum_{\vec{x}'} w(\vec{x}, \vec{x}')} \\
\]

\[
w(\vec{x}, \vec{x}') = e^{-|P(\vec{x}')-P(\vec{x})|^2/2\alpha} \cdot e^{-|\vec{x}'-\vec{x}|^2/2\beta}
\]
Gaussian Blur:
After Median Filter:
After Bilateral:
After Non-Local Means:
Before:
Method Noise (Gaussian)
Method Noise (Bilateral)
Method Noise (Non-Local Means)
Run-Times: Rect Filter

• For every pixel:
  – For every other nearby pixel:
    • Do a multiply and add
Run-Times: Gaussian Blur

• For every pixel:
  – For every other nearby pixel:
    • Compute a distance-based weight
      – (can be precomputed)
    • Do a multiply and add
Run-Times: Median Filter

• For every pixel:
  – For every other nearby pixel:
    • Add into a histogram
  – Compute the median of the histogram
Run-Times: Bilateral

• For every pixel:
  – For every other nearby pixel:
    • Compute a similarity weight
    • Compute a distance-based weight
      – (can be precomputed)
    • Do a multiply and add
Run-Times: Non-Local Means

• For every pixel x:
  – For every other nearby pixel y:
    • Compute a distance weight
    • Compute a similarity weight:
    • For every pixel z in a patch around y
      – Compare z to the corresponding pixel in the patch around x
      – Add to the similarity term
    • Multiply and add
These methods are all fairly useless for large filter sizes

• ... unless you can speed them up