CSE 559A: Computer Vision

Fall 2019: T-R: 11:30-12:50pm @ Hillman 60

Instructor: Ayan Chakrabarti (ayan@wustl.edu).
Course Staff: Arushee Agrawal, Annie Lee, Jiahao Li, Xiaochen Zhou

http://www.cse.wustl.edu/~ayan/courses/cse559a/

Sep 12, 2019
GENERAL

Office Hours Change

- Prof. Chakrabarti: Mondays 2-3pm @ Jolley 205
- Annie & Arushee: Wednesdays 2-3pm @ Jolley 217
- Jiahao & Xiaochen: Thursdays 5-6pm @ Jolley 217

No office hours on Tuesday anymore. Office hours today evening as originally scheduled.
GENERAL

Recommend that you ask questions that are:

- More general at office hours
  - Stuff in lectures, or concepts behind the problem set questions
- Very specifically related to specific problems on problem sets on Piazza
  - If you are posting code or if it's about your own solution, make the post private

Not a hard rule, but TAs may especially be reluctant to get too deep into solution specifics with you at office hours.
EFFICIENT COMPUTATION

• Convolution, in the most general case, takes $O(n_x n_k)$ time.
  - $n_x = W_x H_x, n_k = W_k H_k$.

• Convolution in the frequency domain:
  - FFT, point-wise multiply, Inverse FFT
  - FFT/IFFT complexity is $O(n_x \log n_x)$ (Most efficient for power of 2 image size)
  - May be worth it for large kernels
  - Or same image convolved with many different kernels
EFFICIENT COMPUTATION

Separable Kernels

\[ G[n_x, n_y] \propto \exp\left(-\frac{n_x^2 + n_y^2}{2\sigma^2}\right) = G_x[n_x]G_y[n_y] \]

- \(x\)- and \(y\)- derivatives of Gaussian also separable.

- Realize that \(k[n_x, n_y] = k_x[n_x]k_y[n_y] = k_x \ast_{\text{full}} k_y\).

  This is by interpreting \(k_x\) and \(k_y\) as having size \(W_k \times 1\) and \(1 \times H_k\).

- So \(X \ast k = X \ast (k_x \ast k_y) = (X \ast k_x) \ast k_y\). This takes \(W_k + H_k\) operations instead of \(W_k H_k\).

- Often if a kernel itself isn't separable, it can be sometimes expressed as a sum of separable kernels.

  - E.g., Unsharp Mask: \((1 + \alpha)\delta - \alpha G_\sigma\) (don't combine!)

- Could also try to do this automatically using SVD.
Recursive Computation

\[
\begin{array}{ccc}
1 & 1 & 1 \\
1 & 1 & 1 \\
1 & 1 & 1 \\
1 & 1 & 1 \\
\end{array}
\begin{array}{c}
1 \\
1 \\
1 \\
1 \\
\end{array}
\begin{array}{c}
0 \\
0 \\
0 \\
1 \\
1 \\
\end{array}
\begin{array}{c}
1 \\
1 \\
1 \\
1 \\
\end{array}
= \\
\begin{array}{c}
1 \\
1 \\
1 \\
1 \\
\end{array}
\begin{array}{c}
0 \\
1 \\
1 \\
1 \\
\end{array} \
\begin{array}{c}
1 \\
1 \\
1 \\
1 \\
\end{array}
\begin{array}{c}
1 \\
1 \\
1 \\
1 \\
\end{array}
= \\
\begin{array}{c}
1 \\
1 \\
1 \\
1 \\
\end{array}
\begin{array}{c}
-1 \\
-1 \\
1 \\
1 \\
\end{array}
\begin{array}{c}
1 \\
1 \\
1 \\
1 \\
\end{array}
\begin{array}{c}
1 \\
1 \\
1 \\
1 \\
\end{array}
\begin{array}{c}
1 \\
1 \\
1 \\
1 \\
\end{array}
= \\
\begin{array}{c}
-1 \\
0 \\
1 \\
1 \\
\end{array}
\begin{array}{c}
0 \\
0 \\
0 \\
1 \\
1 \\
\end{array} \
\begin{array}{c}
1 \\
1 \\
1 \\
1 \\
\end{array}
\begin{array}{c}
1 \\
1 \\
1 \\
1 \\
\end{array}
\begin{array}{c}
1 \\
1 \\
1 \\
1 \\
\end{array}
\begin{array}{c}
1 \\
1 \\
1 \\
1 \\
\end{array}
\begin{array}{c}
1 \\
1 \\
1 \\
1 \\
\end{array}
\end{array}
\]

- Sometimes can decompose into convolution with sparse kernels.
- Many implementations of convolve2d won't make use of sparsity.
  - But you can write your own.
EFFICIENT COMPUTATION

Smooth and Sub-sample

- Don't smooth and subsample!
- For sub-sampling by two, you're computing 4x as many smooth filter outputs as you need to.

Similarly, using zero-filling + convolution for upsampling is inefficient.
Efficient Computation

**numpy Specifics**

1. In general, prefer algorithms that have lower total number of multiplies / adds.
2. Try to use `scipy.signal.convolve2d` (subject to rule 1). It is optimized for cache reads, parallel execution, etc.
   ```python
   # I import it often as:
   from scipy.signal import convolve2d as conv2
   ```
3. Similarly, avoid `for` loops and use element-wise operations on large arrays, matrix multiplies (`np.matmul / np.dot`), etc.

- Some of these things are faster in python because a single large operation runs natively instead of returning to the compiler. But they're also faster because these operations are often 'atomics' in lower-level libraries too (BLAS), and have been highly optimized for modern hardware.
- Thinking about designing your algorithm in terms of these atomic operations is useful beyond python.
- Some points in problem sets allocated for efficient code.
MULTI-SCALE REPRESENTATIONS

Fourier Transform useful for:

\[ A_k = SD_kS^* \]

Concentrating energy (high manitudes) in fewer coefficients

But is the FT interpretable? Does it tell us something about the image?
MULTI-SCALE REPRESENTATIONS

Fourier Transform useful for:

Diagonalizing Convolution \( A_k = S D_k S^* \)

Concentrating energy (high manitudes) in fewer coefficients

But is the FT interpretable? Does it tell us something about the image?
MULTI-SCALE REPRESENTATIONS

- $F[u, v]$ is intuitively average variation in image at that frequency.
- But averaged across the entire image.
- This isn't useful because images aren't "stationary"
- Different parts of the image, "have different frequencies".

- FT decomposition of different levels (coarse/fine) of variation: but without sense of spatial location.
- Multi-scale representations aim to address this.
Gaussian Pyramid

Useful for analyzing image at multiple scales.

E.g., apply the same method (edge detection / CNN) at multiple levels of the pyramid.
MULTI-SCALE REPRESENTATIONS

Laplacian Pyramid

\[ *G_\sigma \downarrow_2 \]

\[ *G_\sigma \downarrow_2 \]

\[ *(\delta - G_\sigma) \]

\[ *(\delta - G_\sigma) \]
MULTI-SCALE REPRESENTATIONS

Laplacian Pyramid

\[ * G_\sigma \quad * (\delta - G_\sigma) \quad \text{Frequency} \]

Low Pass Filter  
High Pass Filter

Keep Gaussian at Coarsest Level
Laplacian Pyramid

No orientation selectivity: See Steerable Pyramids
http://www.cns.nyu.edu/~eero/steerpyr/
Wavelet Pyramid

Gaussian & Laplacian pyramids are good for analysis, but not really a representation. Not easy / possible to go from pyramid coefficients back to original image (like Fourier Transform).

Consider a 2x2 Pixel Block

\[
L = \frac{a + b + c + d}{2}, \quad H_2 = \frac{c + d - a - b}{2}
\]

\[
H_1 = \frac{b + d - a - c}{2}, \quad H_3 = \frac{a + d - b - c}{2}
\]

\[
\begin{bmatrix}
L \\
H_1 \\
H_2 \\
H_3
\end{bmatrix} = \frac{1}{2} \begin{bmatrix}
1 & 1 & 1 & 1 \\
-1 & 1 & -1 & 1 \\
-1 & -1 & 1 & 1 \\
1 & -1 & -1 & 1
\end{bmatrix} \begin{bmatrix}
a \\
b \\
c \\
d
\end{bmatrix}
\]

Unitary Matrix / Co-ordinate transform
MULTI-SCALE REPRESENTATIONS

Wavelet Pyramid

Can be achieved by Convolution + Downsampling

For every non-overlapping 2x2 Patch in the Image

Still a Co-ordinate Transform!

\[
L = \frac{a + b + c + d}{2}
\]

\[
H_1 = \frac{b + d - a - c}{2}
\]

\[
H_2 = \frac{c + d - a - b}{2}
\]

\[
H_3 = \frac{a + d - b - c}{2}
\]
MULTI-SCALE REPRESENTATIONS

Wavelet Pyramid

\[
\begin{bmatrix}
L \\
H_1 \\
H_2 \\
H_3
\end{bmatrix} = \frac{1}{2}
\begin{bmatrix}
1 & -1 & 1 & -1 \\
1 & -1 & 1 & -1 \\
-1 & 1 & -1 & 1 \\
-1 & 1 & -1 & 1
\end{bmatrix}
\begin{bmatrix}
a \\
b \\
c \\
d
\end{bmatrix}
\]

Can Invert to get Image

\[
L = \frac{a + b + c + d}{2}
\]

\[
H_1 = \frac{b + d - a - c}{2}
\]

\[
H_2 = \frac{c + d - a - b}{2}
\]

\[
H_3 = \frac{a + d - b - c}{2}
\]
MULTI-SCALE REPRESENTATIONS

Wavelet Pyramid

Apply Recursively

Wavelet Transform

\[
L = \frac{a + b + c + d}{2} \quad H_2 = \frac{c + d - a - b}{2}
\]

\[
H_1 = \frac{b + d - a - c}{2} \quad H_3 = \frac{a + d - b - c}{2}
\]

"Harr"

Others based on different filters
MULTI-SCALE REPRESENTATIONS

Wavelet Pyramid

Applications: Analysis, image modeling & restoration, compression
Wavelet Pyramid

Applications: Analysis, image modeling & restoration, compression
Statistics / Estimation Recap

Setting

- There is a true image (or image-like object) $x$ that we want to estimate
- What we have is some degraded observation $y$ that is based on $x$
- The degradation might be "stochastic": so we say $p(y|x)$

Simplest Case (single pixel: $x, y \in \mathbb{R}$):

$$y = x + \sigma \epsilon \rightarrow p(y|x) = \mathcal{N}(y; \mu = x, \sigma^2)$$

Estimate $x$ from $y$: "denoising"
Statistics / Estimation Recap

- $p(y|x)$ is our observation model, called the likelihood function
  - Likelihood of observation $y$ given true image $x$
- Maximum Likelihood Estimator: $\hat{x} = \arg\max_x p(y|x)$

For $p(y|x) = \mathcal{N}(y; x, \sigma^2)$, what is ML Estimate?

$$p(y|x) = \frac{1}{\sqrt{2\pi\sigma}} \exp\left(-\frac{(y-x)^2}{2\sigma^2}\right)$$

$\hat{x} = y$

That's a little disappointing.
Statistics / Estimation Recap

- What we really want to do is maximize $p(x|y)$, not the likelihood $p(y|x)$
  - $p(y|x)$ is the distribution of observation $y$ given true image $x$
  - $p(x|y)$ is the distribution of true $x$ given observation $y$
- Bayes Rule

$$p(x|y) = \frac{p(y|x)p(x)}{p(y)}$$

Simple Derivation

$$p(x, y) = p(x)p(y|x) = p(y)p(x|y) \rightarrow p(x|y) = \frac{p(y|x)p(x)}{p(y)}$$

$$p(y) = \int p(y|x')p(x')dx'$$
Statistics / Estimation Recap

\[ p(x|y) = \frac{p(y|x)p(x)}{\int p(y|x')p(x')dx'} \]

- \( p(y|x) \): Likelihood
- \( p(x) \): "Prior" Distribution on \( x \)
  - What we believe about \( x \) before we see the observation
- \( p(x|y) \): Posterior Distribution of \( x \) given observation \( y \)
  - What we believe about \( x \) after we see the observation
Statistics / Estimation Recap

\[ p(x|y) = \frac{p(y|x)p(x)}{\int p(y|x')p(x')dx'} \]

Maximum A Posteriori (MAP) Estimation

\[ \hat{x} = \arg \max_x p(x|y) \]

- The most likely answer under the posterior distribution
- Other Estimators Possible: Try to minimize some "risk" or loss function \( L(\hat{x}, x) \)
  - Measures how bad an answer \( \hat{x} \) is when true value is \( x \)
- Minimize Expected Risk Under Posterior

\[ \hat{x} = \arg \min_x \int L(x, x')p(x'|y)dx' \]

- Let's say \( L(x, x') = (x - x')^2 \). What is \( \hat{x} \) ?

\[ \hat{x} = \mathbb{E}_{p(x|y)}x = \int x \ p(x|y)dx \]
Statistics / Estimation Recap

\[ p(x|y) = \frac{p(y|x)p(x)}{\int p(y|x')p(x')dx'} \]

Maximum A Posteriori (MAP) Estimation

\[ \hat{x} = \arg \max_x p(x|y) \]

How do we compute this?

\[ \hat{x} = \arg \max_x p(x|y) = \arg \max_x \frac{p(y|x)p(x)}{\int p(y|x')p(x')dx'} \]

\[ = \arg \max_x p(y|x)p(x) \]
Statistics / Estimation Recap

\[ p(x|y) = \frac{p(y|x)p(x)}{\int p(y|x')p(x')dx'} \]

Maximum A Posteriori (MAP) Estimation

\[ \hat{x} = \arg \max_x p(x|y) \]

How do we compute this?

\[ \hat{x} = \arg \max_x p(x|y) = \arg \max_x \frac{p(y|x)p(x)}{\int p(y|x')p(x')dx'} \]

\[ = \arg \max_x p(y|x)p(x) = \arg \max_x \log[p(y|x)p(x)] \]

\[ = \arg \max_x \log p(y|x) + \log p(x) \]

\[ = \arg \min_x -\log p(y|x) - \log p(x) \]

(Turn this into minimization of a cost)
Statistics / Estimation Recap

Maximum A Posteriori (MAP) Estimation

\[ \hat{x} = \arg\min_x - \log p(y|x) - \log p(x) \]

Back to Denoising

- \( p(y|x) = \frac{1}{\sqrt{2\pi\sigma}} \exp\left( -\frac{(y-x)^2}{2\sigma^2} \right) \)
- Let's choose a simple prior:
  - \( p(x) = \mathcal{N}(x; 0.5, 1) \)
Let's choose a simple prior:

\[ p(x) = \mathcal{N}(x; 0.5, 1) \]
IMAGE RESTORATION

Statistics / Estimation Recap

Maximum A Posteriori (MAP) Estimation

\[
\hat{x} = \arg\min_x \left( \frac{(y - x)^2}{2\sigma^2} + C + \frac{(x - 0.5)^2}{2} + C' \right)
\]

Back to Denoising

- \( p(y|x) = \frac{1}{\sqrt{2\pi\sigma}} \exp\left(-\frac{(y-x)^2}{2\sigma^2}\right) \)
- Let's choose a simple prior:
  - \( p(x) = \mathcal{N}(x; 0.5, 1) \)
Statistics / Estimation Recap

Maximum A Posteriori (MAP) Estimation

\[
\hat{x} = \arg\min_x \frac{(y - x)^2}{2\sigma^2} + \frac{(x - 0.5)^2}{2}
\]

Back to Denoising

- \( p(y|x) = \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{(y-x)^2}{2\sigma^2}\right) \)

- Let's choose a simple prior:
  - \( p(x) = \mathcal{N}(x; 0.5, 1) \)
Statistics / Estimation Recap

Maximum A Posteriori (MAP) Estimation

\[ \hat{x} = \arg \min_x \frac{(y - x)^2}{2\sigma^2} + \frac{(x - 0.5)^2}{2} \]

How do you compute \( x \)?

- In this case, simpler option available. Complete the squares, express as a single quadratic \( \propto (x - \hat{x})^2 \).
- More generally, to minimize \( C(x) \), find \( C'(x) = 0 \), check \( C''(x) > 0 \)
- What is \( C'(x) \)?

\[ C'(x) = \frac{x - y}{\sigma^2} + x - 0.5 \]
Statistics / Estimation Recap

Maximum A Posteriori (MAP) Estimation

\[
\hat{x} = \arg\min_x \frac{(y - x)^2}{2\sigma^2} + \frac{(x - 0.5)^2}{2}
\]

\[
C'(x) = \frac{x - y}{\sigma^2} + x - 0.5 = 0
\]

\[
\hat{x} = \frac{y + 0.5\sigma^2}{1 + \sigma^2}
\]

\[
C''(x) = \frac{1}{\sigma^2} + 1 > 0 \quad \forall x
\]

- Means the function is convex (quadratic functions with a positive coefficient for \(x^2\) are convex)
Statistics / Estimation Recap

Maximum A Posteriori (MAP) Estimation

\[ \hat{x} = \arg \min_x \frac{(y - x)^2}{2\sigma^2} + \frac{(x - 0.5)^2}{2} \]

\[ \hat{x} = \frac{y + 0.5\sigma^2}{1 + \sigma^2} \]

- Weighted sum of observation and prior mean
- Closer to prior mean when \( \sigma^2 \) is high
Statistics / Estimation Recap

- Let's go beyond "single pixel images": \( Y[n] = X[n] + \epsilon[n] \)
- If noise is independent at each pixel

\[
p(\{Y[n]\}|\{X[n]\}) = \prod_n p(Y[n]|X[n]) = \prod_n \mathcal{N}(Y[n]; X[n], \sigma^2)
\]

- Similarly, if prior \( p(\{X[n]\}) \) is defined to model pixels independently:

\[
p(\{X[n]\}) = \prod_n \mathcal{N}(X[n]; 0.5, 1)
\]

- Product turns into sum after taking log
Statistics / Estimation Recap

\[
\hat{X}[n] = \arg\min_{\{X[n]\}} \sum_n \frac{(Y[n] - X[n])^2}{2\sigma^2} + \frac{(X[n] - 0.5)^2}{2}
\]

- Note: This is a minimization over multiple variables

- But since the different variables don't interact with each other, we can optimize each pixel separately

\[
\hat{X}[n] = \frac{Y[n] + 0.5\sigma^2}{1 + \sigma^2} \quad \forall n
\]
Multi-variate Gaussians

- Re-interpret these as multi-variate Gaussians

For $X \in \mathbb{R}^d$:

$$p(X) = \frac{1}{\sqrt{(2\pi)^d \det(\Sigma)}} \exp \left( -\frac{1}{2} (X - \mu)^T \Sigma^{-1} (X - \mu) \right)$$

- Here, $\mu \in \mathbb{R}^d$ is a mean "vector", same size as $X$
- Co-variance $\Sigma$ is a symmetric, positive-definite matrix $d \times d$ matrix

When the different elements of $X$ are independent, off-diagonal elements of $\Sigma$ are 0.

How do we represent our prior and likelihood as a multi-variate Gaussian?
Multi-variate Gaussians

- Represent $X$ and $Y$ as vectorized images

\[ p(Y|X) = \mathcal{N}(Y; X, \sigma^2 I) \]

- Here the mean-vector for $Y$ is $X$

- The co-variance matrix is a diagonal matrix with all diagonal entries $= \sigma^2$
Multi-variate Gaussians

\[
p(Y|X) = \frac{1}{\sqrt{(2\pi)^d \det(\Sigma)}} \exp\left(-\frac{1}{2} (Y - \mu)^T \Sigma^{-1} (Y - \mu)\right)
\]

- \( \mu = X, \Sigma = \sigma^2 I \)
- \( \det(\Sigma) = (\sigma^2)^d \)
- \( \Sigma^{-1} = \sigma^{-2} I \)

\[
(Y - \mu)^T \Sigma^{-1} (Y - \mu) = (Y - X)^T \frac{1}{\sigma^2} I (Y - X) = \frac{1}{\sigma^2} (Y - X)^T I (Y - X)
\]

\[
= \frac{1}{\sigma^2} (Y - X)^T (Y - X) = \frac{1}{\sigma^2} \|Y - X\|^2
\]

\[
= \frac{1}{\sigma^2} \sum_n (Y[n] - X[n])^2
\]
But now, we can use these multi-variate distributions to model correlations and interactions between different pixels.

Let's stay with denoising, but talk about defining a prior in the Wavelet domain.

Instead of saying, individual pixel values are independent, let's say individual wavelet coefficients are independent.

Let's also put different means and variances on different wavelet coefficients.

- All the 'derivative' coefficients have zero mean.
- Variance goes up as we go to coarser levels.

\[
p(Y|X) = \mathcal{N}(Y; X, \sigma^2 I) \\
p(X) = \mathcal{N}(Y; 0.5I, I) \\
\hat{X} = \arg \min_X \frac{1}{\sigma} \|Y - X\|^2 + \|X - 0.5\|^2
\]
Let $C = WX$, where $W$ represents the Wavelet transform matrix

- $W$ is unitary, $W^{-1} = W^T$
- $X = W^T C$

Define our prior on $C$:

$$p(C) = \mathcal{N}(C; \mu_c, D_c)$$

- Now, $D_c$ is a diagonal matrix, with entries equal to corresponding variances of coefficients
- Off-diagonal elements 0 implies Wavelet co-efficients are un-correlated
- A prior on $C$ implies a prior on $X$
  - $C$ is just a different representation for $X$
Change of Variables

- We have probability distribution on a random variable $U: p_U(U)$
- $U = f(V)$ is a one-to-one function

\[ p_V(V) = p_U(f(V)) \]

Realize, that $p(\cdot)$ are densities. So we need to account for "scaling" of the probability measure.

\[
\int p_U(U) dU = \int p_U(f(V)) dU \\
\frac{dU}{dV} = \det(J_f) dV
\]

where $\det(J_f)_{ij} = \frac{d u_i}{d v_j}$

So, $p_V(V) = p_U(f(V)) \det(J_f)$

If $U = f(V) = A V$ is linear, $J = A$.

If $A$ is unitary, $\det(A) = 1$. 