Toolbox

CS535
Daniel G. Aliaga

Image Tools

- Features
  - Point, edge, line, corner, SIFT
  - Hough Transform

Edge Detection

- What would you do?

Edge Detection: First Order Operator

- Roberts operator (1963) on image A:

- $G_x = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} * \mathrm{A}$, $G_y = \begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix} * \mathrm{A}$
- $G = \sqrt{G_x^2 + G_y^2}$
- $\theta = \tan^{-1}(\frac{G_y}{G_x})$

(pro: less ops than other methods)

Edge Detection

- Sobel operator (1968) on image A:
- $G_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} * \text{A}, \; G_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix} * \text{A}$
- $G = \sqrt{G_x^2 + G_y^2}$
- $\theta = \tan^{-1}(\frac{G_y}{G_x})$

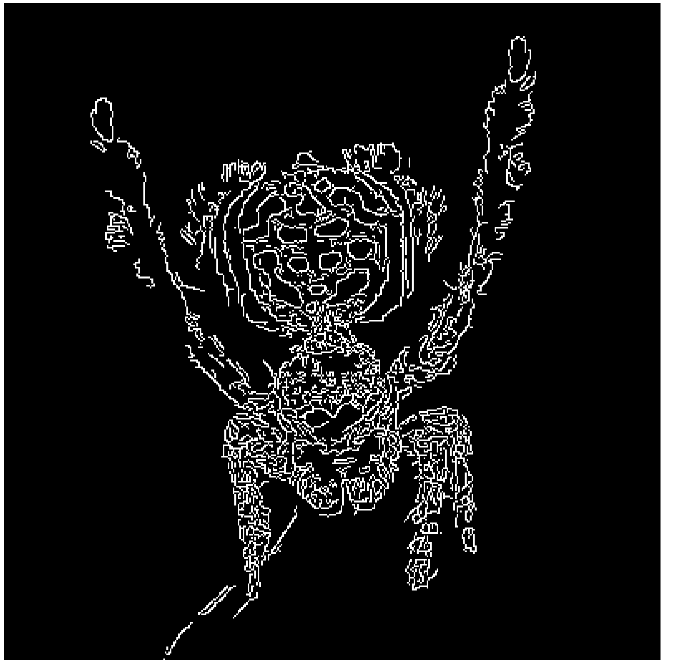
Edge Detection

- Prewitt operator (1970) on image A (different spectral response as compared to Sobel):

- $G_x = \begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{bmatrix}$*A, $G_y = \begin{bmatrix} -1 & -1 & -1 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \end{bmatrix}$*A
- $G = \sqrt{G_x^2 + G_y^2}$
- $\theta = \tan^{-1}(\frac{G_y}{G_x})$

Edge Detection

- Canny Edges (1986)
  - Multi-stage algorithm, uses Sobel/Prewitt (or other) edge detector on a Gaussian filtered image and then has a process of non-maximal suppression



Edge Detection: Second-Order Operator

- Given an image:
  - Gradient (vector)

$$\nabla f(x,y) = \frac{\partial f}{\partial x}\hat{x} + \frac{\partial f}{\partial y}\hat{y}$$

  - Laplacian (scalar) (2nd order)

$$\nabla^2 f(x,y) = \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2}$$

Discrete Laplacian

- $\nabla^2 f(x,y) =$
$$f(x-1,y) + f(x+1,y) + f(x,y-1) + f(x,y+1) - 4f(x,y)$$
- Matrix form = ??

Discrete Laplacian

- $\nabla^2 f(x, y) =$
$$f(x-1, y) + f(x+1, y) + f(x, y-1) + f(x, y+1) - 4f(x, y)$$
- Matrix form =
$$\begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$$

Edge Detection: Second-Order Operator

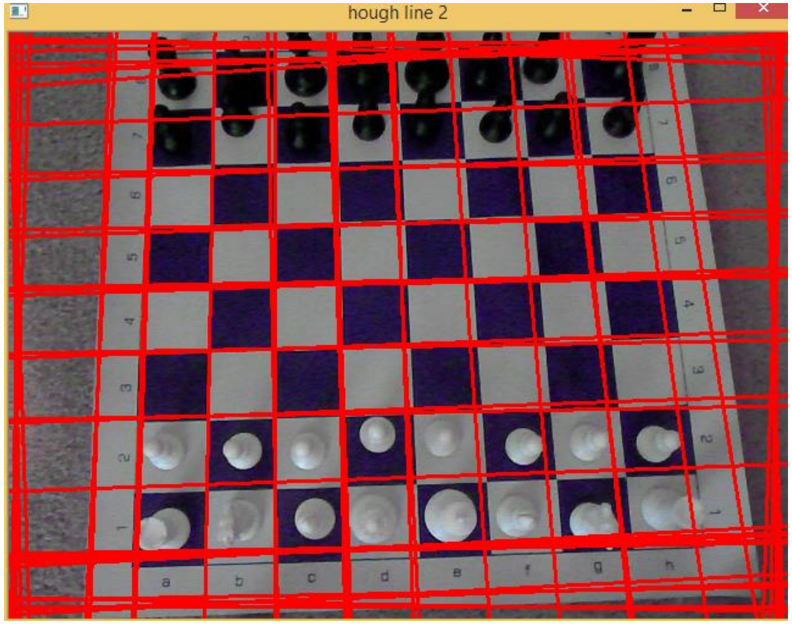
- Laplacian: highlights regions of rapid intensity change

- $L_A = \begin{bmatrix} 0 & -1 & 0 \\ -1 & 4 & -1 \\ 0 & -1 & 0 \end{bmatrix} * A$

(positive Laplacian takes out outward edges; negative Laplacian is possible too)

Edge Detection

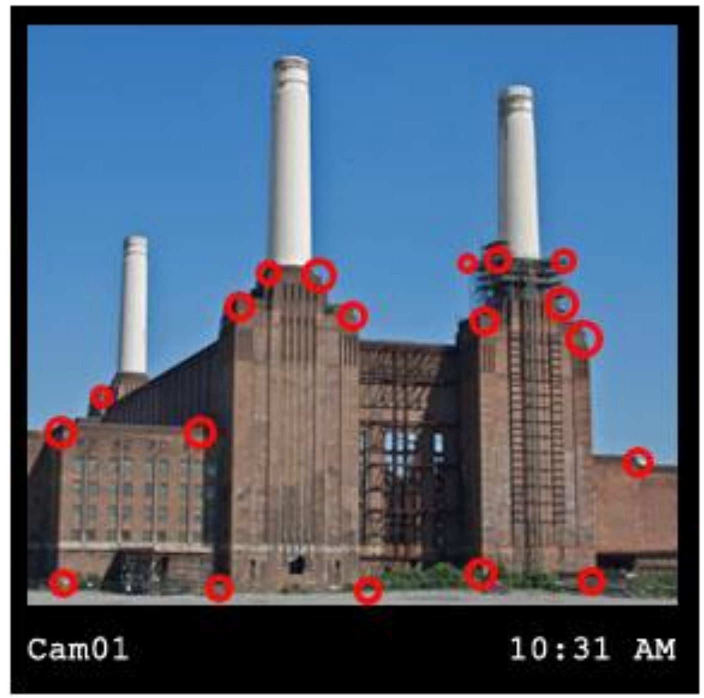
- Hough Transform (1972)
  - Associate with each line segment, a pair (r, θ)
  - Each line segment could be obtained by fitting to results of edge detection
  - Ex: find edges, find strong clusters/points in transform space, then draw lines



Corner Detection

- What would you do?

A: Original image



B: Detected image

Corner Detection

- Harris-Stephens Corner Detector
  - Let the SSD between two patches be:

$$f(\Delta x, \Delta y) = \sum_{(\boldsymbol{x_k}, \boldsymbol{y_k}) \in \boldsymbol{W}} (\boldsymbol{A}(\boldsymbol{x_k}, \boldsymbol{y_k}) - \boldsymbol{A}(\boldsymbol{x_k} + \Delta \boldsymbol{x}, \boldsymbol{y_k} + \Delta \boldsymbol{y}))^{\boldsymbol{2}}$$

  - $\boldsymbol{A}(\boldsymbol{x_k} + \Delta \boldsymbol{x}, \boldsymbol{y_k} + \Delta \boldsymbol{y})$ can be approximated by its Taylor Expansion:
    $= A(x_k, y_k) + A_x(x_k, y_k)\Delta x + A_y(x_k, y_k)\Delta \mathrm{y}$ (A_x, A_y are partial derivatives)

  - Thus, $f(\Delta x, \Delta y) \cong \sum \left(A_x(x_k, y_k)\Delta x + A_y(x_k, y_k)\Delta \mathrm{y}\right)^2$
  - which can be rewritten as

$$f(\Delta x, \Delta y) \approx [\Delta x \ \Delta y] M \begin{bmatrix} \Delta x \\ \Delta y \end{bmatrix}$$

  - Where M is the second-moment tensor (or structural tensor):

$$M = \begin{bmatrix} \sum_{(x,y) \in W} A_X^2 & \sum_{(x,y) \in W} A_x A_y \\ \sum_{(x,y) \in W} A_x A_y & \sum_{(x,y) \in W} A_y^2 \end{bmatrix}$$

Corner Detection

- Harris-Stephens Corner Detector
  - With a structural tensor, the eigenvectors summarize the distribution of the gradient within the associated pixel window
  - To define a strong corner, we want pixels were λ_1 and λ_2 of M are large, and hence f is large
  - $\lambda_1 \gg \lambda_2$ or $\lambda_2 \gg \lambda_1$ means an edge
  - $\lambda_1 \approx \lambda_2$ and large means corner

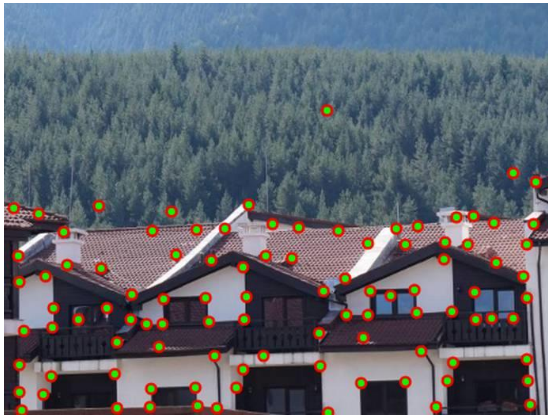
  - One option, compute score:

$$R = \det(M) - k \cdot tr(M)^2$$

k empirically determined, usually $[0.04, 0.06]$

det(M) = $\lambda_1 \lambda_2$ tr(M)=$\lambda_1 + \lambda_2$

R small = flat, R < 0 = edge, R > 0 = corner



Corner Detection

- Shi-Tomasi Detector
  - Similar to Harris but compute $\min(\lambda_1, \lambda_2)$ directly (using characteristic equation)

(claimed to be better, perhaps)

Feature Detection

- Corners
- SIFT: Scale Invariant Feature Transform (1999)
- SURF: Speeded Up Robust Features (2006)
- Deep Learning Based Feature Detection…

SIFT

- Properties:
  - Invariant to spatial rotation, translation, scale
  - Experimentally seen to be less sensitive to small spatial affine or perspective changes
  - Invariant to affine illumination changes

SIFT

- Computational Steps:
  - Scale-space extrema detection
    - local extrema detection using DoG (difference of Gaussians)
    - Compare difference of Gaussians center on a pixel to lower and higher blurs
    - Pick the scale/pixel with highest differences

SIFT

- Computational Steps:
  - Scale-space extrema detection
  - Keypoint localization
    - Similar to Harris Corner Detector, refine location of corners; ignore relatively weak corners

SIFT

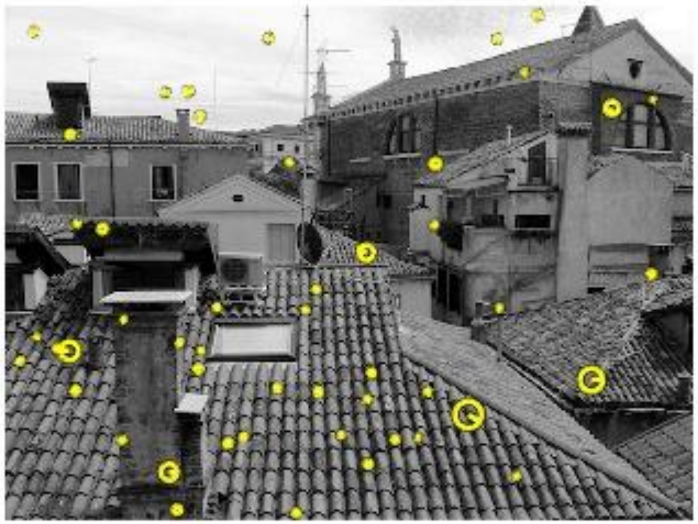
- Computational Steps:
  - Scale-space extrema detection
  - Keypoint localization
  - Compute orientation
    - Use an orientation histogram with 36 bins (or so)

SIFT

- Computational Steps:
  - Scale-space extrema detection
  - Keypoint localization
  - Compute orientation
  - Keypoint descriptor creation
    - Use 16x16 pixel neighborhood to define 4x4 pixel subblocks yields a 128 vector as a descriptor of orientations and normalized to be illumination invariant

SIFT

- Computational Steps:
  - Scale-space extrema detection
  - Keypoint localization
  - Compute orientation
  - Keypoint descriptor creation



Deep Learning Edge Detection

- HED
  - https://arxiv.org/pdf/1504.06375.pdf

- DexiNET
  - https://arxiv.org/pdf/1909.01955.pdf

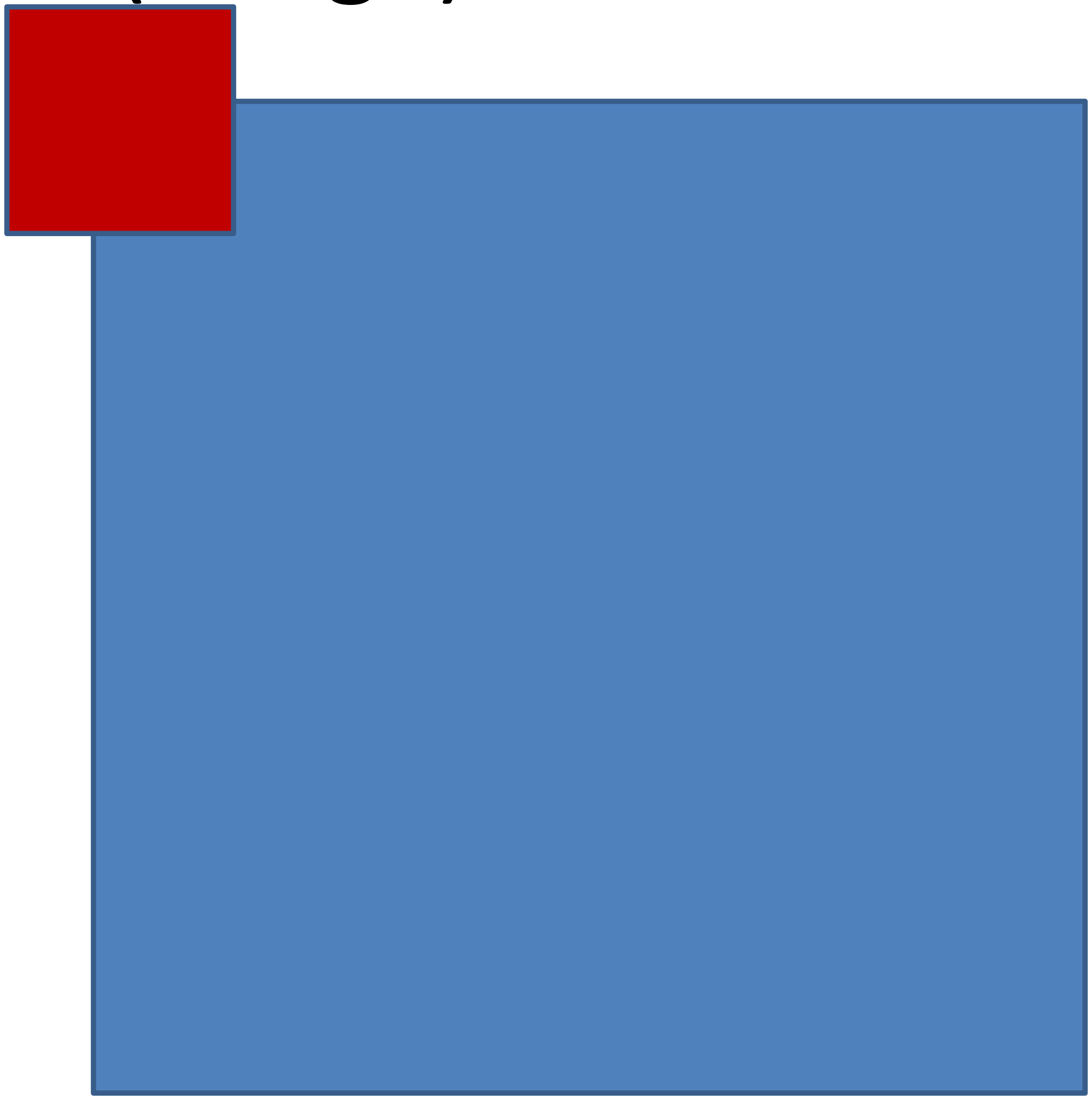
(Image) Convolution

- Convolution
  - Define a kernel
  - “Convolve the image”

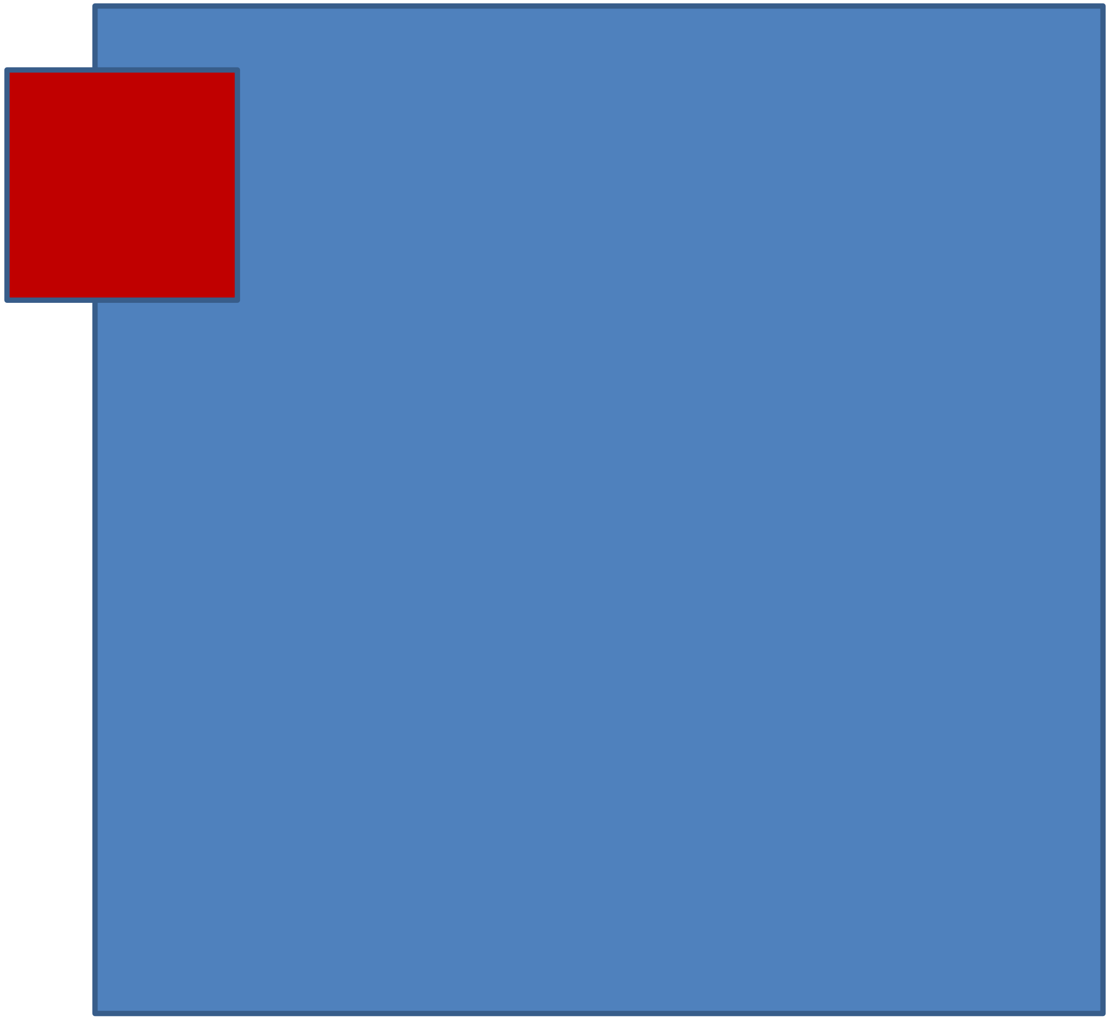
(Image) Convolution

- Kernel: $(1/16)\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$
- What if kernel is not normalized?
- Image: $\begin{bmatrix} p_{11} & \cdots & p_{m1} \\ \vdots & \ddots & \vdots \\ p_{1n} & \cdots & p_{mn} \end{bmatrix}$
- What if image is multi-channel?
- What if kernel falls off the side of the image?

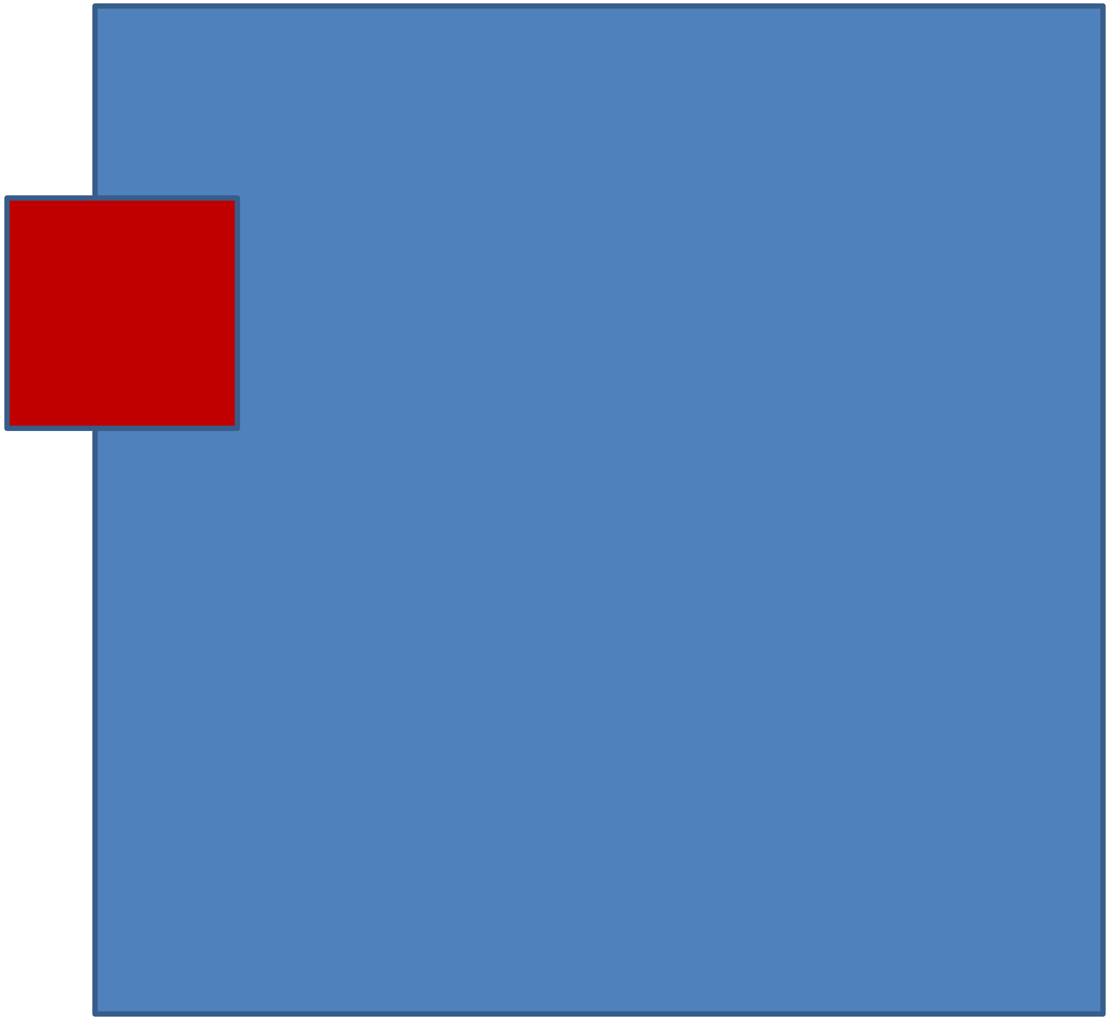
(Image) Convolution



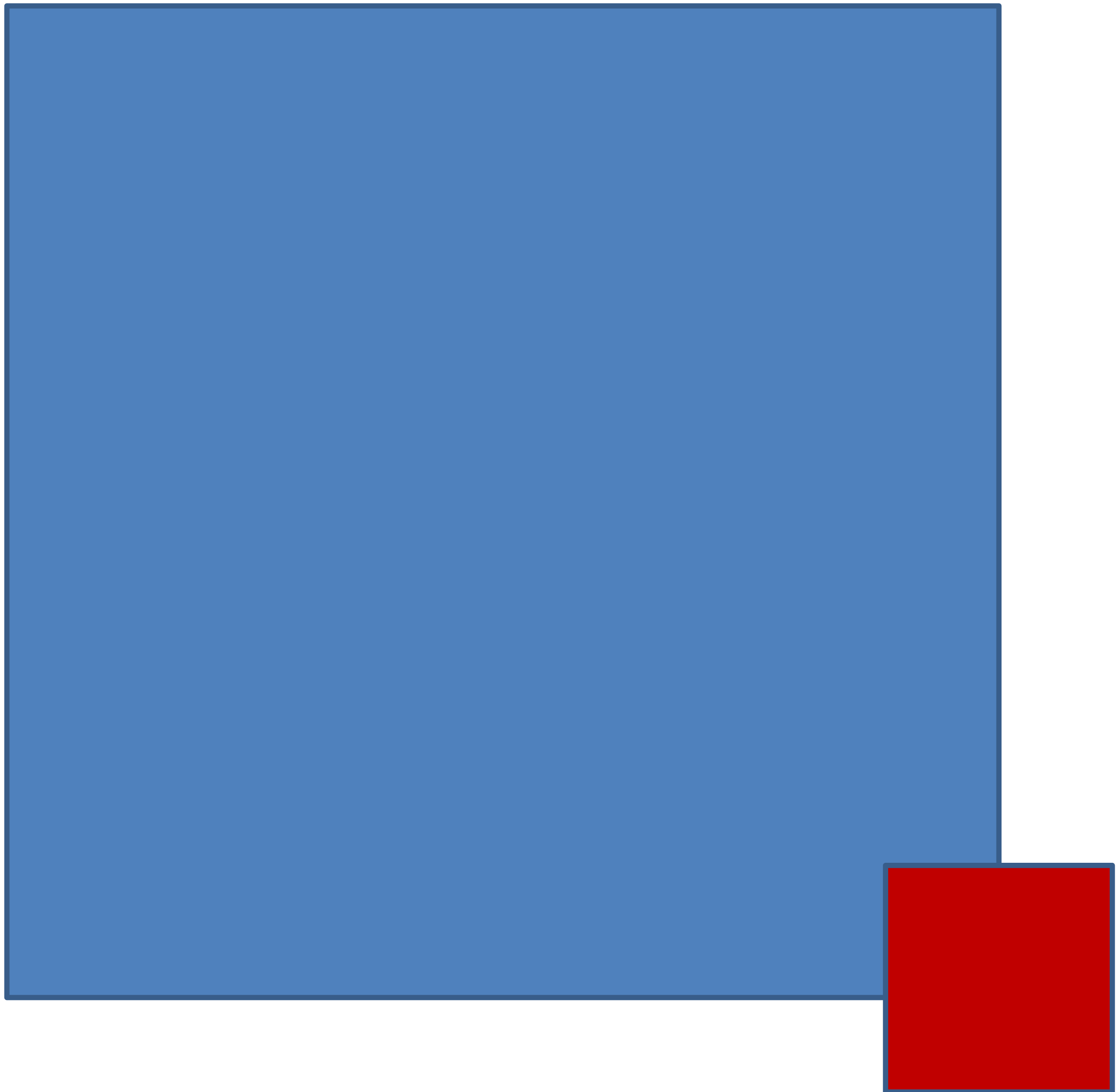
(Image) Convolution



(Image) Convolution



(Image) Convolution



(Image) Convolution

- Recall
  - Convolution in spatial domain = multiplication in frequency domain
  - Thus, low/high frequency filter is a simple multiplication in frequency space
  - Phase component also exists in frequency space so that makes things more complicated…

(Image) Correlation

- Convolution: result of a composition of two signals
- Correlation: measure of coincidence of two signals
  - Subtle difference…
  - Mathematically, the difference is only two signs
  - https://www.youtube.com/watch?v=O9-HN-yzsFQ

- Correlation = measure of similarity?
  - Maybe: Pearson correlation measure

$$\rho_{X,Y} = \frac{\mathrm{E}[(X - \mu_X)(Y - \mu_Y)]}{\sigma_X \sigma_Y}$$

  - Does this work?

Image Similarity Metrics

- Use SIFT/SURF
  - Compute features and see how similar
- L2-norm
  - Per-pixel L2-norm
- Cross correlation
  - Kinda Pearson correlation
- SSIM
- Deep Learning…

Image Similarity Metrics

- SSIM: Structural Similarity Index

$$SSIM(x,y) = \left[l(x,y)^{\alpha} \cdot c(x,y)^{\beta} \cdot s(x,y)^{\gamma}\right]$$

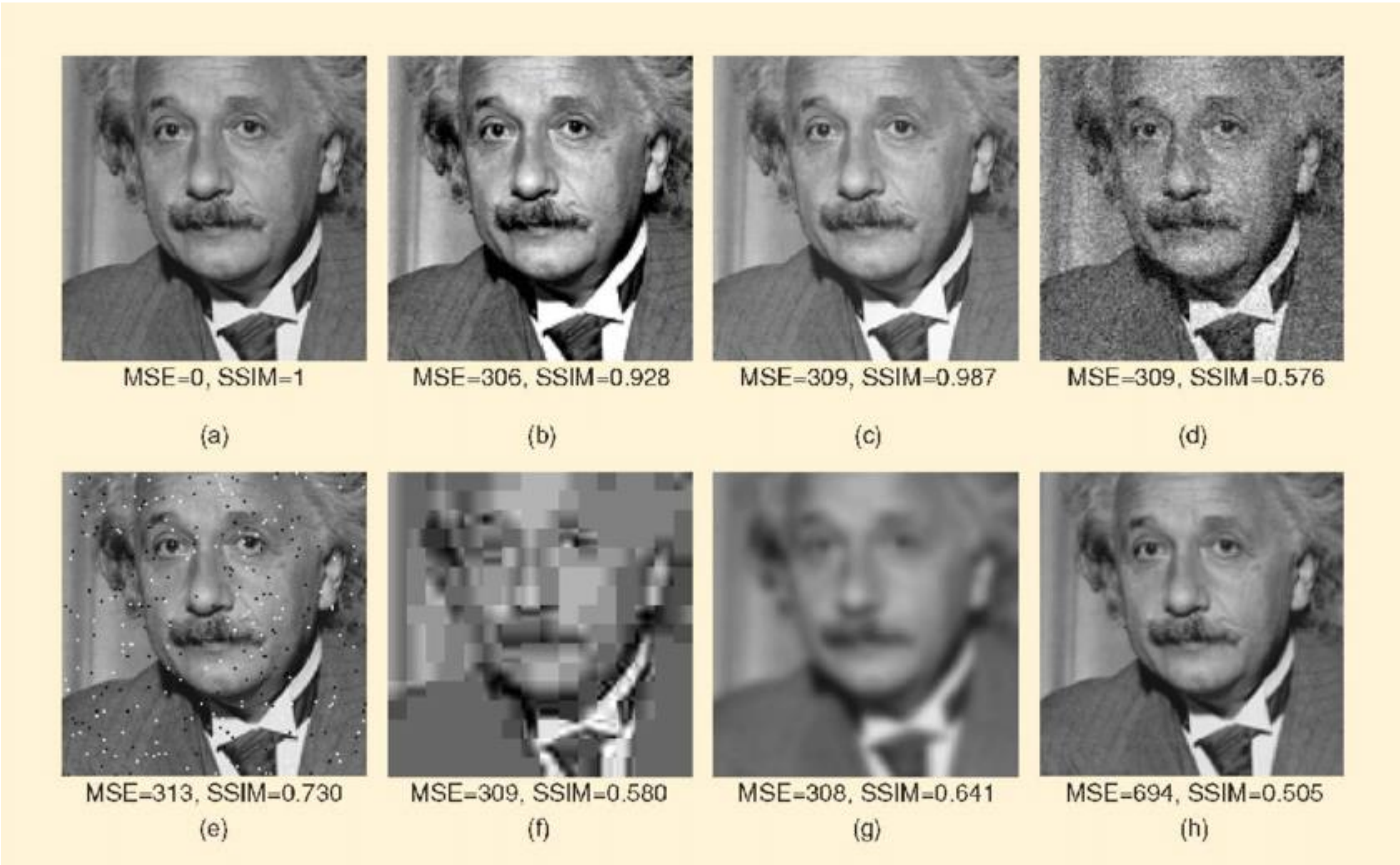
where

$l(x,y)$ measures luminance similarity,

$c(x,y)$ measures contrast similarity, and

$s(x,y)$ measures structure similarity (by covariance)

SSIM

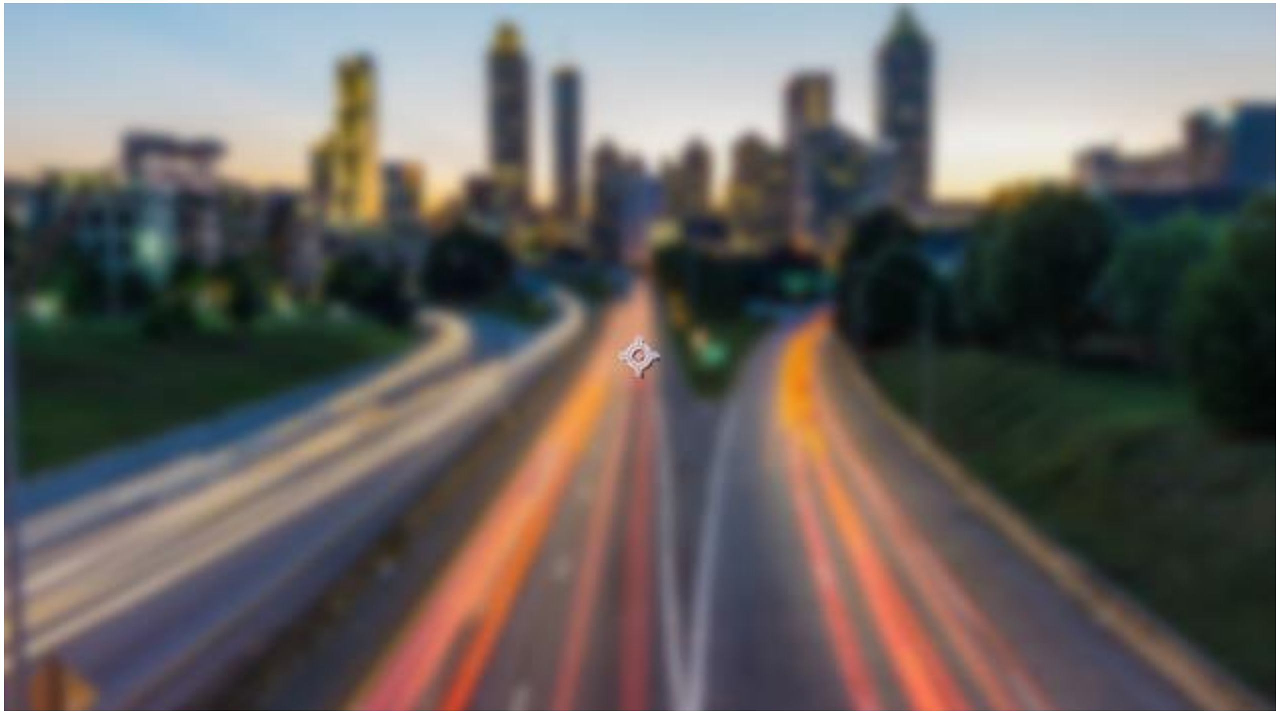

MSE=0, SSIM=1
(a)
MSE=306, SSIM=0.928
(b)
MSE=309, SSIM=0.987
(c)
MSE=309, SSIM=0.576
(d)
MSE=313, SSIM=0.730
(e)
MSE=309, SSIM=0.580
(f)
MSE=308, SSIM=0.641
(g)
MSE=694, SSIM=0.505
(h)

Blurring

- Blur:
  - Box Blur

Blurring

- Gaussian Blur

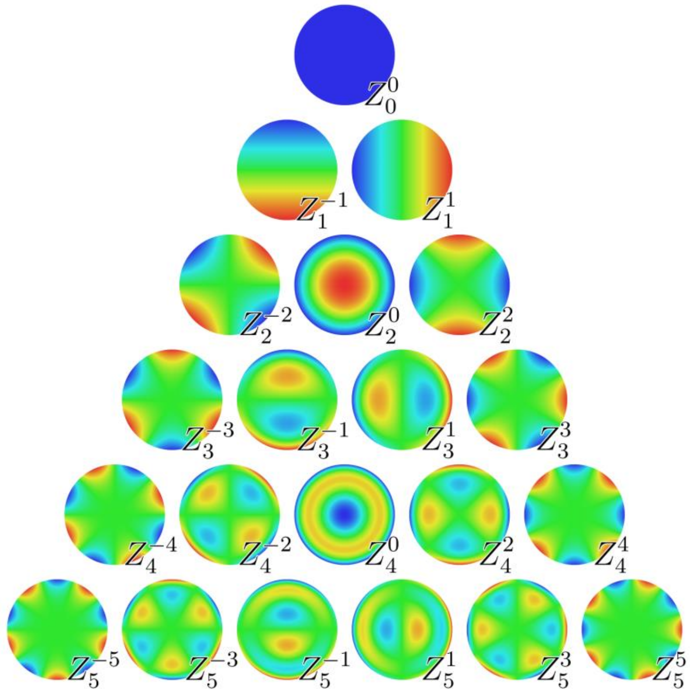


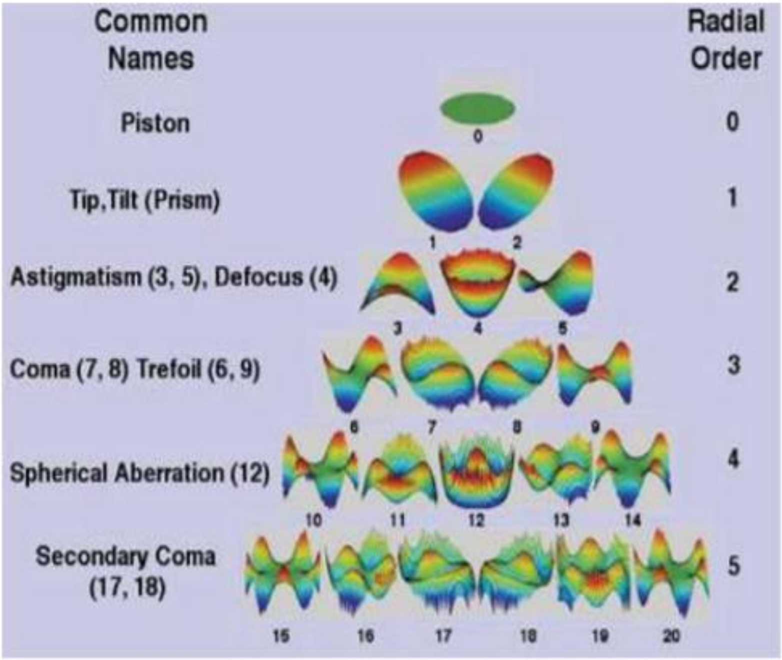
Blurring

- Blur:
  - Radial Blur

Blurring

- Optical Blur:
  - PSF composed of Zernike Polynomials





Blurring

- Basic notion:
  - Blur is basically a PSF (Point Spread Function)

- Basic technique:
  - Apply a spatial blurring using a kernel and convolution

Note: Bilateral Filtering/Blurring

- It is a non-linear, edge-preserving, and noise-reducing smoothing filter
- It replaces the intensity of each pixel with a weighted average of intensity values from nearby pixels but not across edges

Bilateral Filter

- What is the formulation to account for value difference and spatial difference?

Bilateral Filter

- Given image I
- Value difference is $f(x_i, x)$
  - E.g., $\|I(x_i) - I(x)\|$
- Spatial difference is $g(x_i, x)$
  - E.g., $\|x_i - x\|$
- Altogether:

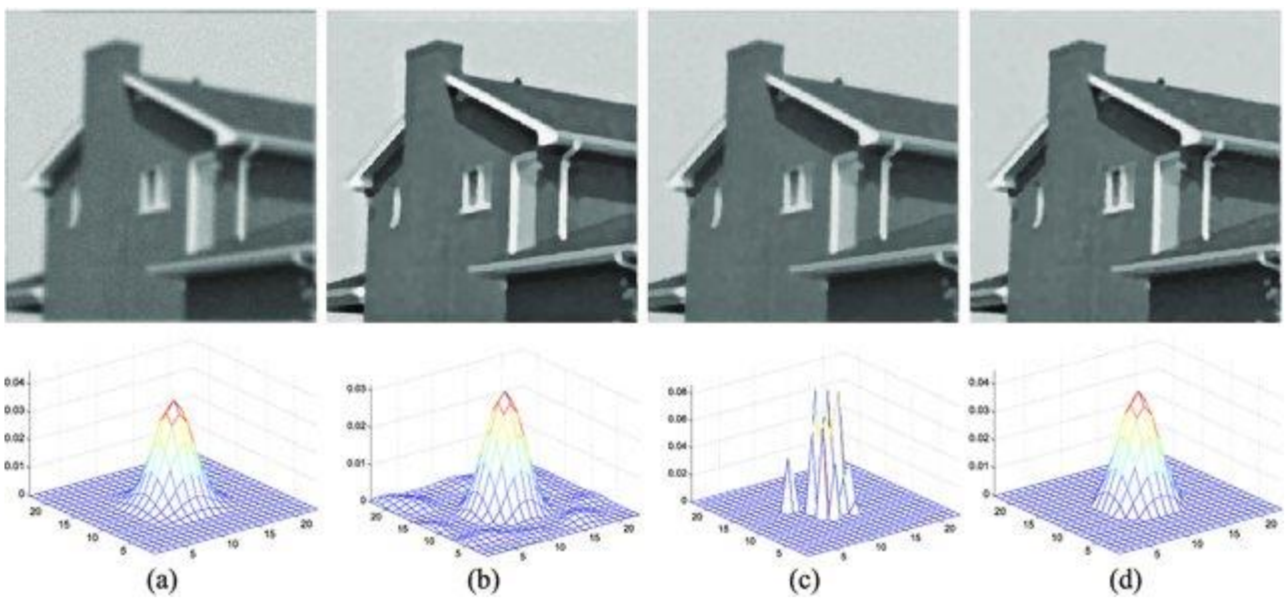
$$I^{\text{filtered}}(x) = \frac{1}{W_p} \sum_{x_i \in \Omega} I(x_i) f_r(\|I(x_i) - I(x)\|) g_s(\|x_i - x\|)$$

Deblurring

- <u>One option </u>is to perform a deconvolution:
  - Non-blind deconvolution
    - The PSF is known

Deblurring

- Another option is to perform a deconvolution:
  - Blind deconvolution
    - The PSF is NOT known



Several variations of blind deconvolution

Human Computation

- https://www.youtube.com/watch?v=tx082gDwGcM
  - Start at 6:45
- Relates to:
  - Citizen science is sometimes described as "public participation in scientific research
  - Crowdsourcing is a less-specific, more public group, to help with the work
  - whereas outsourcing is commissioned from a specific, named group, and includes a mix of bottom-up and top-down processes

Function Solving vs Optimization

- Finding "solutions":
  - Newton's method: $x_{n+1} = x_n - \frac{f(x_n)}{f\prime(x_n)}$
  - Gradient descent: $x_{n+1} = x_n - \alpha_n \nabla F(x_n)$
  - If have no derivatives, use Powell's (conjugate direction) method:
    - Searches in a variety of directions and picks best
  - Linear system of equations: $Ax = b$
    - What is A is not square?
    - …then it is over/under determined

Optimization

- Linear least squares (LLS):
  - LLS is the problem of approximately solving an overdetermined system of linear equations, where the best approximation is defined as that which minimizes the sum of squared differences between the data values and their corresponding modeled values.
  - $x = (A^T A)^{-1} A M^T y$ where y are dependent observations and A are independent observations (note: $(A^T A)^{-1} A^T$ is the Moore-Penrose inverse which is needed because A is not square – else would just be $x = A^{-1} y$

Optimization

- Non-linear least squares (NLLS):
  - Requires successive approximations to solve

$$S = \sum_i W_{ii} \left(y_i - \sum_j X_{ij} \beta_j \right)^2$$

e.g. Levenbu vMar) uses the Jacobian anc

$$f(x_i, p + \delta) \approx f(x_i, p) + J_i \delta$$

PROBLEM: NLLS very sensitive to the presence of outliers (i.e., x_i, y_i pairs that behavior weird, maybe noise)

Optimization

- Random Sample Consensus (RANSAC)
  - Assumes that inliers exist and focuses on determining and using those
  - Randomly select data points and if they fit sufficiently well, use in the iterative optimization

- Rule of thumb:
    - If lots of inliers, use NLLS
    - If lots of outliers, use RANSAC

Optimization

- Convexity: typical assumption which means that objective function is convex

- Fancier optimization methods:
  - ADMM (Alternating Direction Method of Multipliers): optimize by dividing into subproblems
  - and many more…

Randomization-based Algorithms

- Pro: does not need convexity, can handle many dimensions even with lots of local minima
- Con: no guarantees
  - Exception: if PDF of parameters is known and is Gaussian, then it is a maximum likelihood estimation which can essentially be $\approx$ NLLS

Randomization-based Algorithms

- Simulated Annealing
  - Inject noise while during optimization and hope for the best…
- Sequential Monte Carlo (or particle filters)
  - A set of Monte Carlo algorithms, that given some knowledge as to the expected parameter variance, can chose number and range of perturbations, that with some guarantees can field the optimum
  - Fun fact: developed in 1940s by Ulam and von Neumann who used the code name Monte Carlo since the work was secret – think WWII

Randomization-based Algorithms

- Markov Chain Monte Carlo (MCMC):
  – An ensemble of chains is created and walked along
    - Start with a set of points
    - Propose changes to the chains at different temperatures
    - Use acceptance probability to accept some chains (e.g., Metropolis-Hastings method)
    - Keep best chains and repeat
    - Terminate at max iterations or at little change
  – Used often in high-complexity (not-necessarily convex) problems in graphics/vision

Deep Learning

- Has lots of parameters to optimize (100M!)
  - SGD: Stochastic Gradient Descent
  - AdaGrad: Adaptive Gradient Descent
  - ADAM: Adaptive Moment Estimation