Hough Transforms, Shape Representation

DD2423 Image Analysis and Computer Vision

Mårten Björkman

Robotics, Perception and Learning Division
School of Electrical Engineering and Computer Science

November 12, 2021

Edge detection in short (more next week)

Edge detection in three steps:

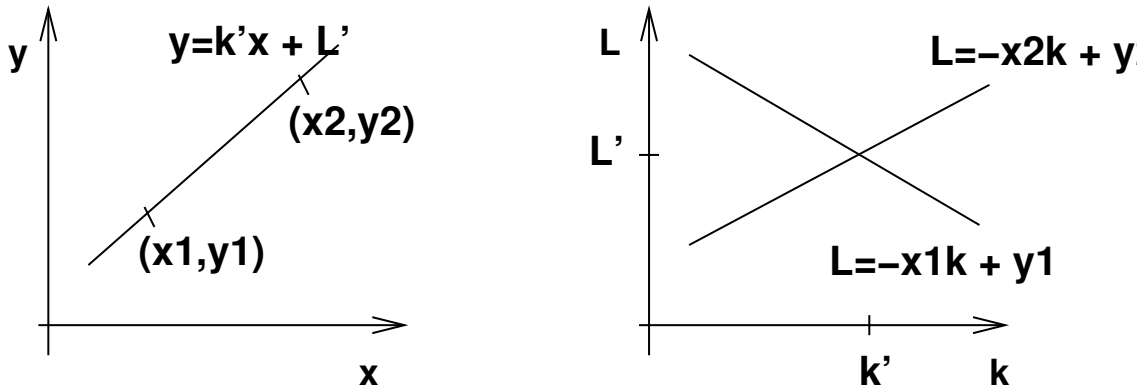
1. Compute image gradients (f_x, f_y) from derivatives.
2. Find edge points by maximizing gradient magnitudes $|(f_x, f_y)|$.
3. Finally, link edge points to edge segments.

How do we go from that to a more compact description? Find lines!

Problem: edge segments might be fragmented into many pieces.

Hough transform for line detection

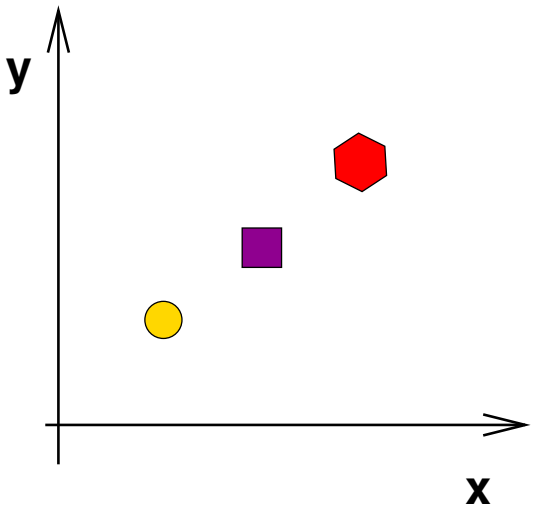
- *Line = a collection of edge points placed along the same direction*.
- Hough Transform is an algorithm that 'detects' all possible lines spanned by the edge points.
- The line *strength* depends on how many points lie on that line.



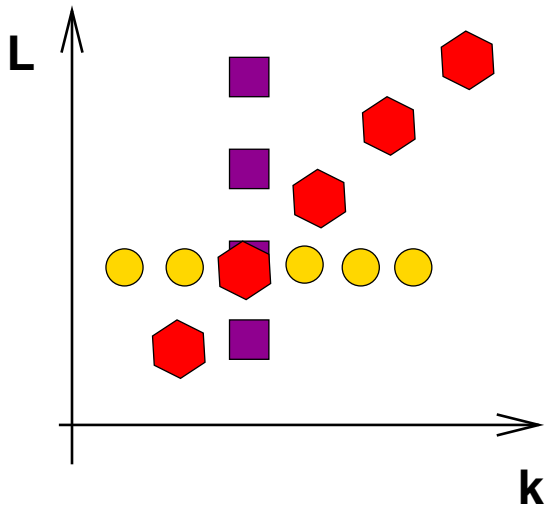
- A point in (x, y)-space is mapped to a line in (L, k)-space.
- A line in (x, y)-space is mapped to a point in (L, k)-space.

From edge points to lines

Discrete space:



edge points



accumulator

- Each point (x, y) can come from many lines in (L, k)-space,

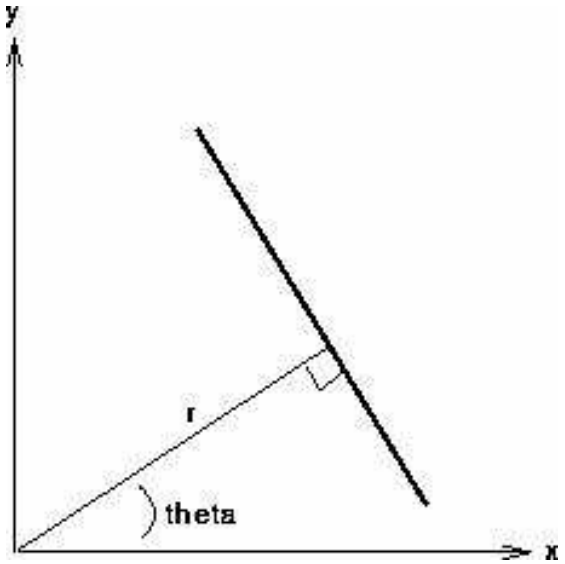
$$L(k) = -kx + y$$

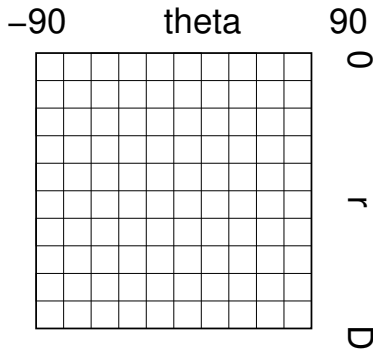
- Idea: the line parameters (L, k) are given by intersection point in accumulator space that gets most votes.

Problem: parameterization in duality space

- Observation: vertical lines correspond to $k \to \pm\infty$.
- Better method: use parameterization $x\cos\theta + y\sin\theta = r$

  θ = orientation of line, r = perpendicular distance to origin

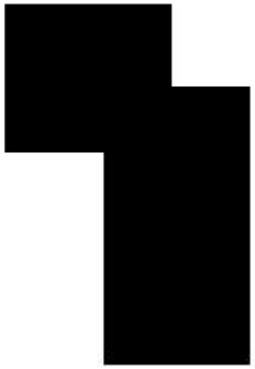


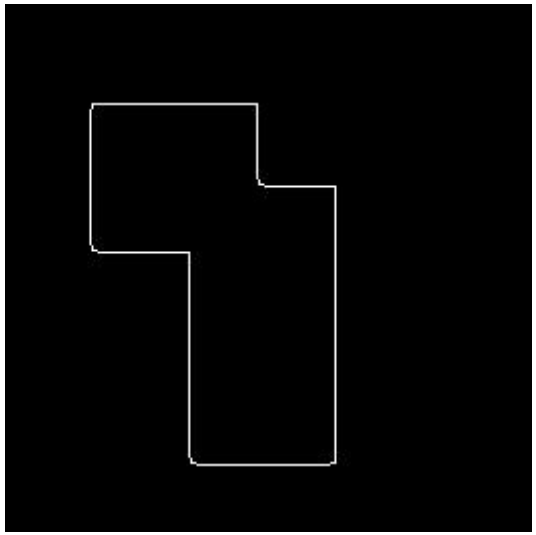


- Point (x, y) in the image $\Rightarrow$ curve $r(\theta) = x\cos\theta + y\sin\theta$ in the Hough (accumulator) space.

Lines represented in Hough space

Which line corresponds to which point in Hough space?

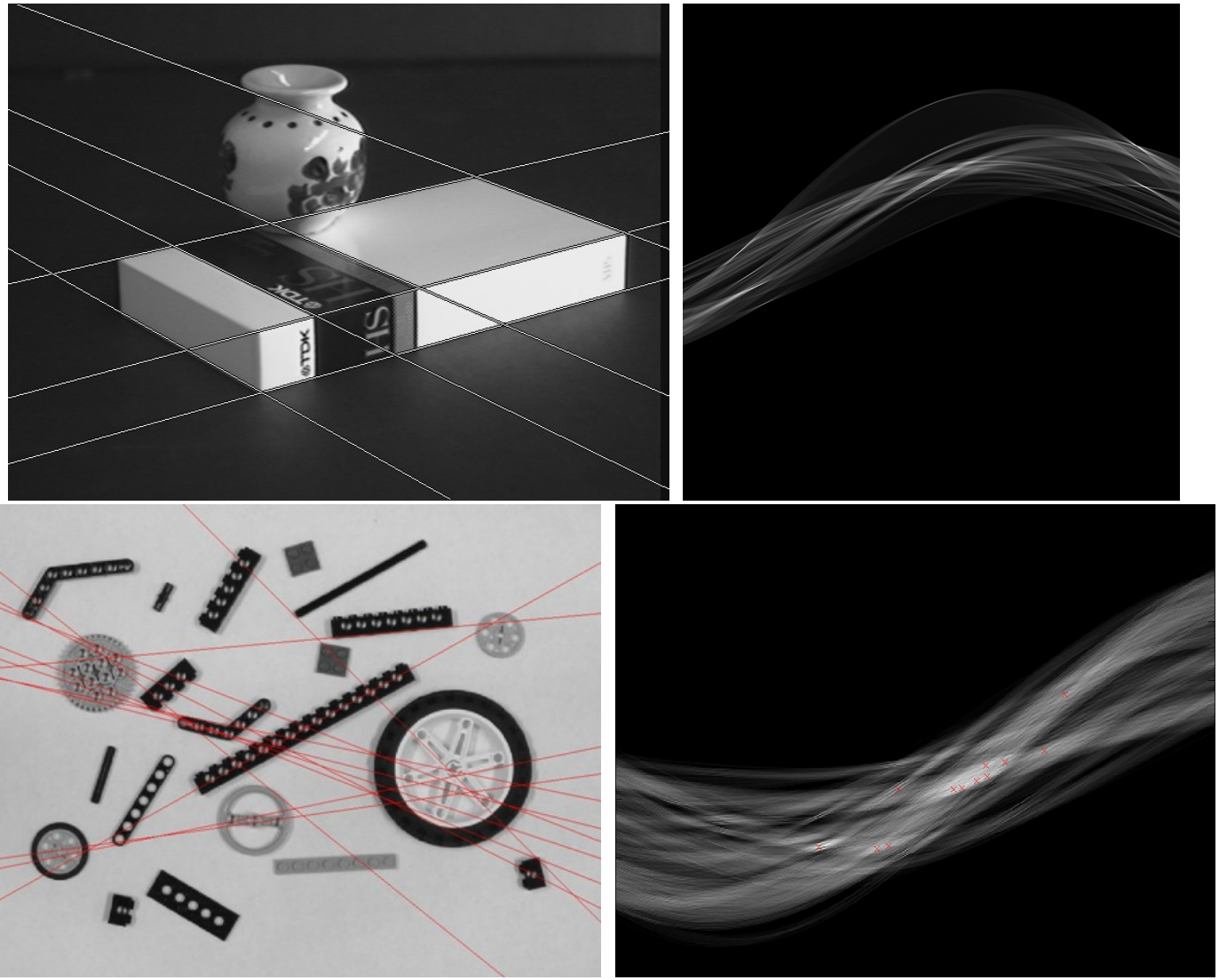




Question: How fine should θ and r be quantized?

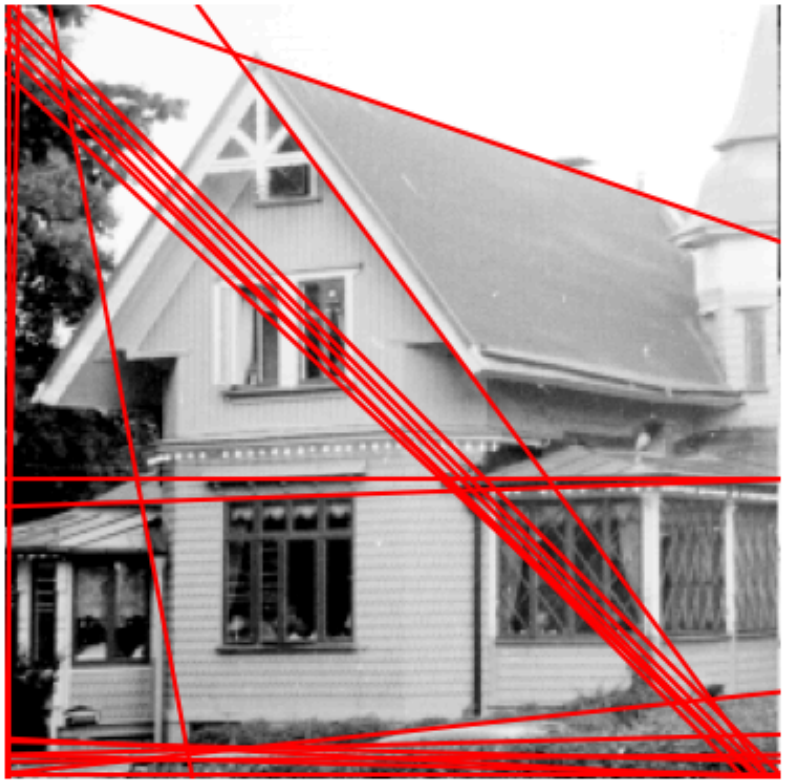
- Too coarse: poor resolution in line directions.
- Too fine: not enough samples in accumulators.

Hough transform (example)



Lower: Edge points may come from texture, not from real lines.

Example from lab 2



Hough transform (improvements)

- Increment with gradient magnitude or some other weighting.

  - Reduces critical dependency on thresholds.
- Use information about gradient direction from edge detection.

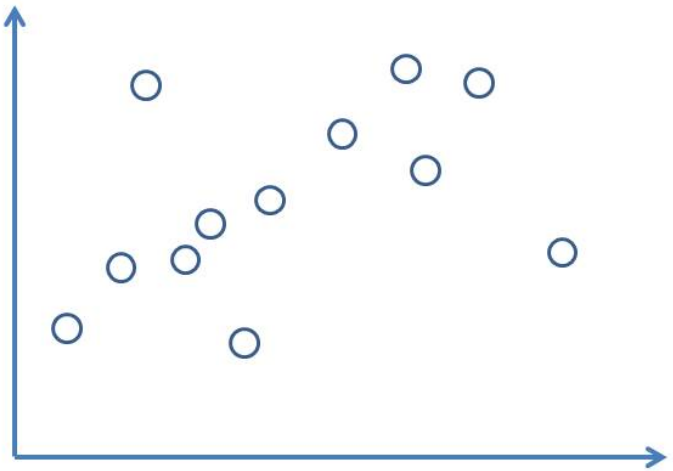
  - Increment only for those r values that seem reasonable.
- Increment accumulator not only point wise but also with some windowing function (Gaussian like)

  - Equivalent to smoothing after voting is done (more effective).

Lane detection using Hough transform

RANSAC: Random Sampling Consensus

- Hough Transforms can be extended
  - Line: 2 unknown parameters $\rightarrow$ 2D Hough space (fine)
  - Circle: 3 unknown parameters $\rightarrow$ 3D Hough space (tricky)
  - Ellipse: 5 unknown parameters $\rightarrow$ 5D Hough space (infeasible)
- Alternative: Random Sampling Consensus (RANSAC)
  1. Randomly pick a minimal set of points (lines=2, circle=3, ellipe=5)
  2. Compute a model (e.g. $y = kx + L$) from the points
  3. Count the number of points 'close enough' to the model.
  4. Repeat 1-3 a given number S of iterations.
  5. Pick the solution with highest number of matching points in 3.



RANSAC (line example)

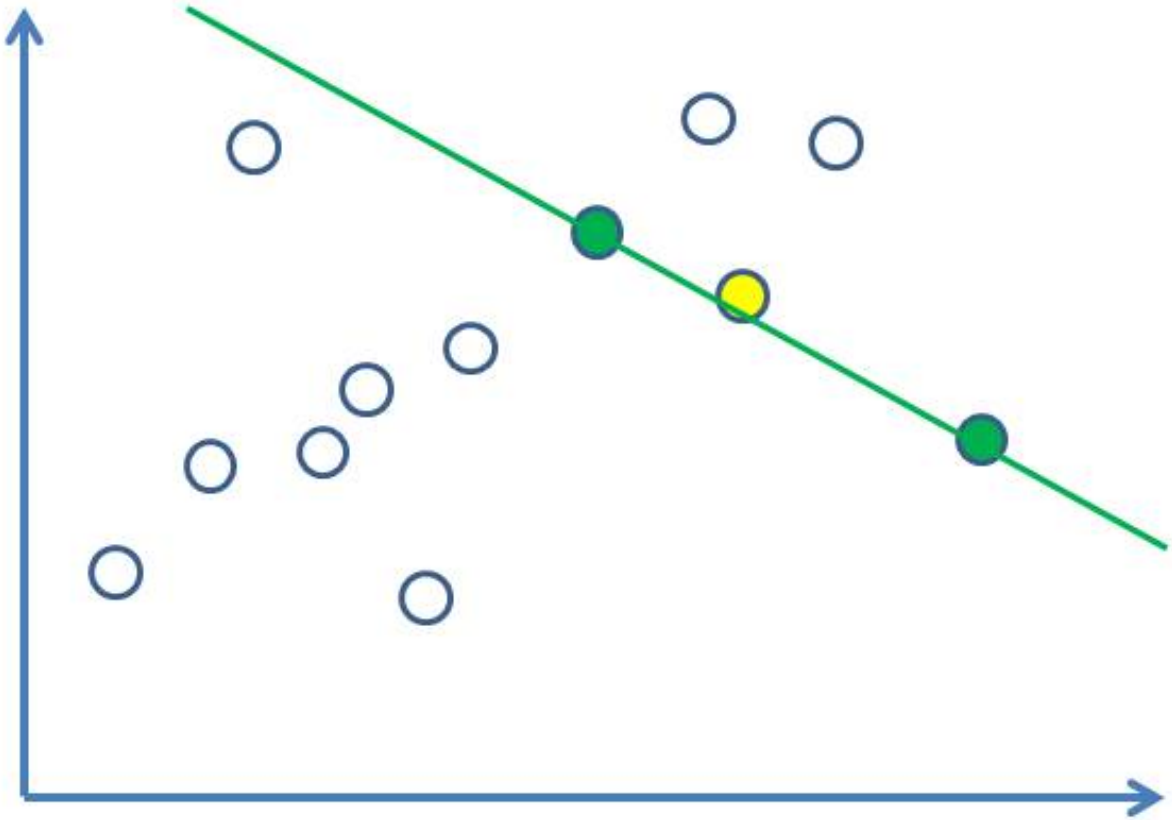
- Step 2: Find the $K = 2$ unknown parameters

$$\begin{cases} kx_1 + L = y_1 \\ kx_2 + L = y_2 \end{cases} \Rightarrow \begin{pmatrix} x_1 & 1 \\ x_2 & 1 \end{pmatrix} \begin{pmatrix} k \\ L \end{pmatrix} = \begin{pmatrix} y_1 \\ y_2 \end{pmatrix} \Rightarrow$$

$$\begin{pmatrix} k \\ L \end{pmatrix} = \frac{1}{x_1 - x_2} \begin{pmatrix} 1 & -1 \\ -x_2 & x_1 \end{pmatrix} \begin{pmatrix} y_1 \\ y_2 \end{pmatrix} = \frac{1}{x_1 - x_2} \begin{pmatrix} y_1 - y_2 \\ x_1 y_2 - x_2 y_1 \end{pmatrix}$$

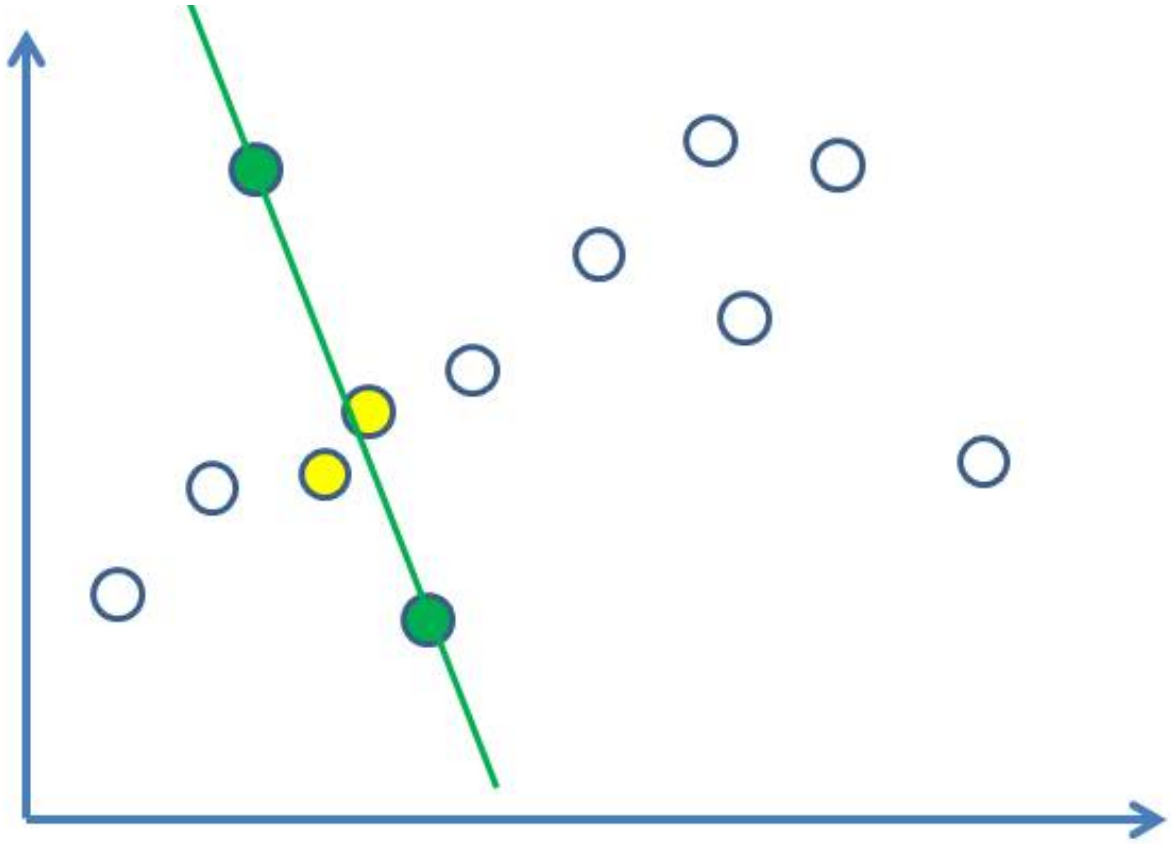
- Step 3: Count the number of points (x_i, y_i) for which $|kx_i + L - y_i| < \varepsilon$, where ε is some threshold (e.g. 2 pixels).

RANSAC (line example)



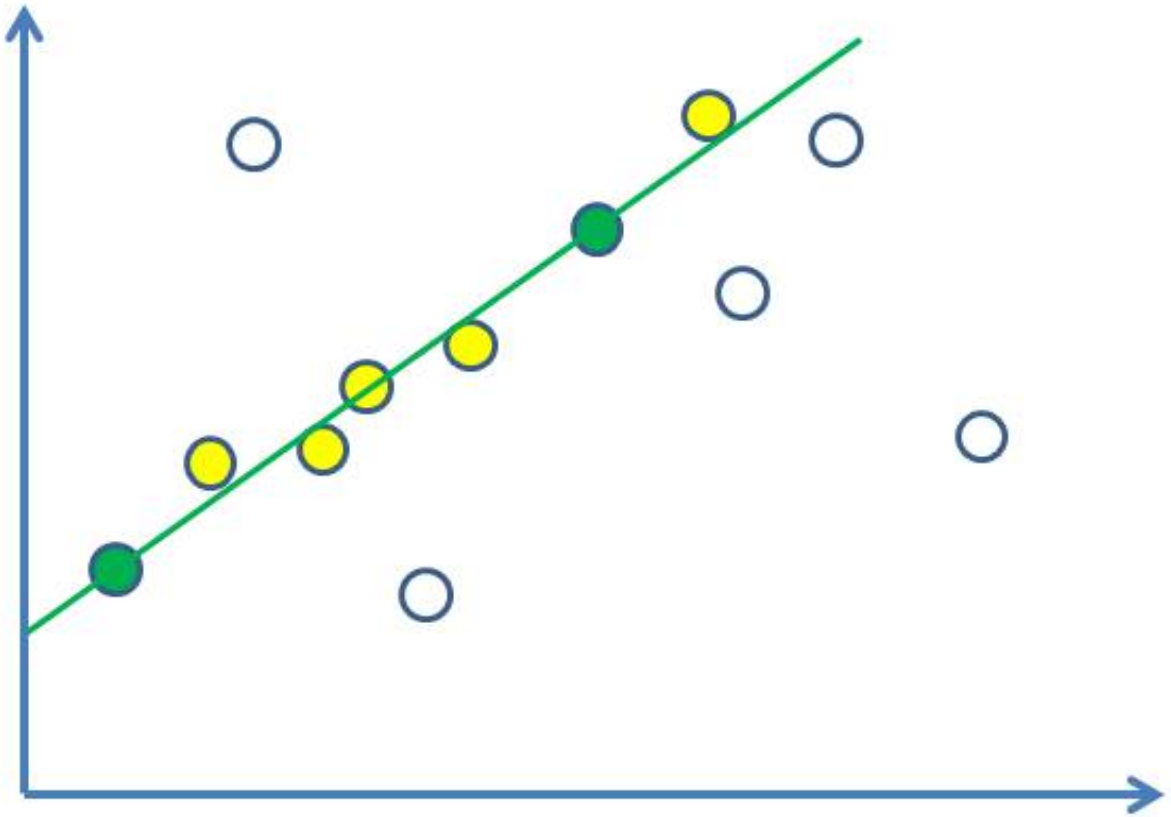
Three inliers, nine outliers.

RANSAC (line example)



Four inliers, eight outliers. Better!

RANSAC (line example)



Seven inliers, five outliers. Even better!

RANSAC: Number of trials?

- All K sampled points must lie on the shape (e.g. line) to be found.
- How many trials are needed for this to happen with a certainty of P, if a fraction of p of points belong to the shape?
- The number of required trials is

$$S = \frac{\log(1-P)}{\log(1-p^K)}$$

Example 1 (line): $p = 10\%, P = 99\% \rightarrow S = 460$

Example 2 (ellipse): $p = 5\%, P = 99\% \rightarrow S = 20^5 = 15 \times 10^6$

- Number of required trials quickly becomes very large.

Matching planes with homographies

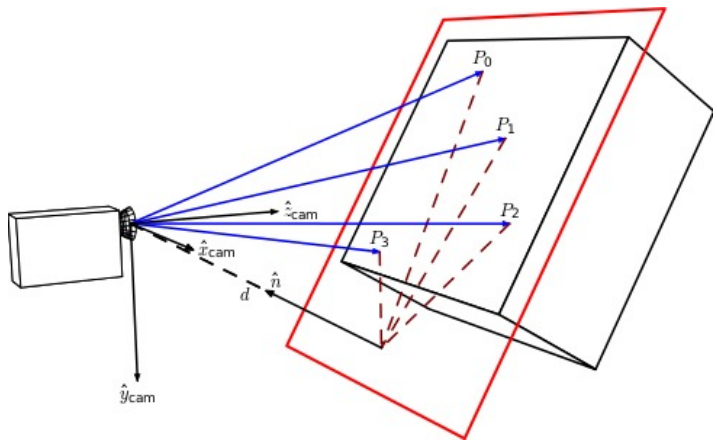
- Projections as normally not invertible.

$$\begin{bmatrix} x_i \\ y_i \\ 1 \end{bmatrix} \simeq \begin{bmatrix} p_{00} & p_{01} & p_{02} & p_{03} \\ p_{10} & p_{11} & p_{12} & p_{13} \\ p_{20} & p_{21} & p_{22} & p_{23} \end{bmatrix} \begin{bmatrix} X_i \\ Y_i \\ Z_i \\ 1 \end{bmatrix}$$

- However, for a plane defined by $Z_i = 0$, it can (usually) be inverted.

$$\begin{bmatrix} x_i \\ y_i \\ 1 \end{bmatrix} \simeq \begin{bmatrix} p_{00} & p_{01} & p_{03} \\ p_{10} & p_{11} & p_{13} \\ p_{20} & p_{21} & p_{23} \end{bmatrix} \begin{bmatrix} X_i \\ Y_i \\ 1 \end{bmatrix}$$

- The relation is a 3×3 homography matrix that is invertible, if the camera center does not lie on the plane.



Matching planes with homographies

- Assume a plane in 3D space is viewed by two different cameras.
- Then a point $(x_i, y_i)^T$ on the plane in one camera can be transformed to a point in the other $(x_i', y_i')^T$ through a homography,

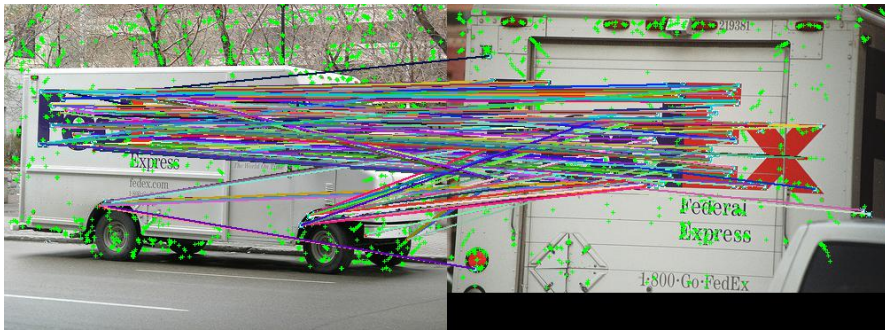
$$\begin{bmatrix} x_i' \\ y_i' \\ 1 \end{bmatrix} \simeq \begin{bmatrix} h_{00} & h_{01} & h_{02} \\ h_{10} & h_{11} & h_{12} \\ h_{20} & h_{21} & h_{22} \end{bmatrix} \begin{bmatrix} x_i \\ y_i \\ 1 \end{bmatrix}$$

- You can see that mapping as a mapping $(x_i, y_i) \rightarrow (X_i, Y_i)$, followed by another mapping $(X_i, Y_i) \rightarrow (x_i', y_i')$.

Matching planes with homographies

1. Detect point features in both image (e.g. SIFT or SURF)

2. Match features between the two images.



3. Use RANSAC to find homography and mismatches (outliers)

Solving for homographies

$$\begin{bmatrix} wx_i' \\ wy_i' \\ w \end{bmatrix} \simeq \begin{bmatrix} h_{00} & h_{01} & h_{02} \\ h_{10} & h_{11} & h_{12} \\ h_{20} & h_{21} & h_{22} \end{bmatrix} \begin{bmatrix} x_i \\ y_i \\ 1 \end{bmatrix}$$

$$x_i' = \frac{h_{00}x_i + h_{01}y_i + h_{02}}{h_{20}x_i + h_{21}y_i + h_{22}}, \quad y_i' = \frac{h_{10}x_i + h_{11}y_i + h_{12}}{h_{20}x_i + h_{21}y_i + h_{22}}$$

$$\begin{cases} x_i'(h_{20}x_i + h_{21}y_i + h_{22}) = h_{00}x_i + h_{01}y_i + h_{02} \\ y_i'(h_{20}x_i + h_{21}y_i + h_{22}) = h_{10}x_i + h_{11}y_i + h_{12} \end{cases}$$

$$\begin{bmatrix} x_i & y_i & 1 & 0 & 0 & 0 & -x_i'x_i & -x_i'y_i & -x_i \\ 0 & 0 & 0 & x_i & y_i & 1 & -y_i'x_i & -y_i'y_i & -y_i \end{bmatrix} \begin{bmatrix} h_{00} \\ h_{01} \\ h_{02} \\ h_{10} \\ h_{11} \\ h_{12} \\ h_{20} \\ h_{21} \\ h_{22} \end{bmatrix} = 0$$

Solving for homographies

$$\underbrace{\begin{bmatrix} x_1 & y_1 & 1 & 0 & 0 & 0 & -x'_1 x_1 & -x'_1 y_1 & -x_1 \\ 0 & 0 & 0 & x_1 & y_1 & 1 & -y'_1 x_1 & -y'_1 y_1 & -y_1 \\ & & & & \vdots & & & & \\ x_n & y_n & 1 & 0 & 0 & 0 & -x'_n x_n & -x'_n y_n & -x_n \\ 0 & 0 & 0 & x_n & y_n & 1 & -y'_n x_n & -y'_n y_n & -y_n \end{bmatrix}}_{A \atop [2n \times 9]} \underbrace{\begin{bmatrix} h_{00} \\ h_{01} \\ h_{02} \\ h_{10} \\ h_{11} \\ h_{12} \\ h_{20} \\ h_{21} \\ h_{22} \end{bmatrix}}_{h \atop [9 \times 1]} = \underbrace{0}_{[2n \times 1]}$$

Define a least square problem: *minimize* $\|Ah - 0\|^2$

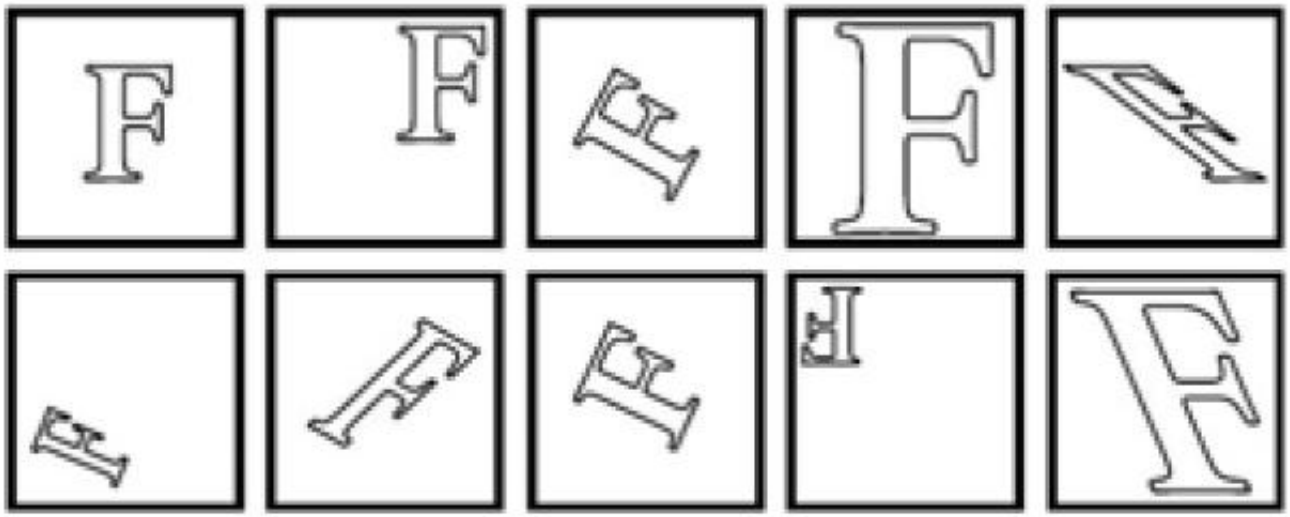
- Since h is only defined up to scale, solve for a unit vector $\hat{h}$.
- Solution: $\hat{h}$ = eigenvector of smallest eigenvalue of $A^T A$.
- At least four points are needed.

Representation of shape

Task: Describe different shapes detected in an image.

- Shape descriptors are 'numbers' that describe a shape.
- A shape may not be reconstructable from a descriptor, but descriptors for different shapes should be different enough for shapes to be discriminated.
- Typical usage:
  - Object recognition
  - Stereo matching

To gain robustness, we often want:



Depending on task, context and images.

- Translation invariance
- Scale invariance
- Rotation invariance (what about 6 and 9?)
- Reflection invariance

Simple (but often efficient) descriptors

- Connected components represented as labeled regions, with attributes such as
  - Size (number of pixels)
  - Bounding box (width and height)
  - Center of gravity
- Compactness =

$$\frac{\text{area}}{(\text{circumference})^2}$$

- Eccentricity =

$$\frac{\text{length of maximum chord A}}{\text{length of maximum chord B} \perp \text{A}}$$

Moment descriptors of regions

- Moments:

$$m_{pq} = \sum_{x,y} x^p y^q f(x,y),$$

$$\text{where } f(x,y) = \begin{cases} 1 & \text{in region} \\ 0 & \text{outside} \end{cases}$$

- Centered moments:

$$\mu_{pq} = \sum_{x,y} (x - \bar{x})^p (y - \bar{y})^q f(x,y)$$

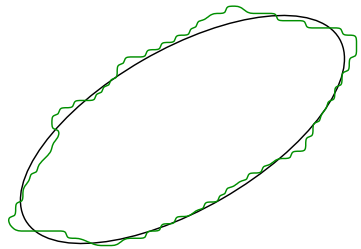
with center of gravity

$$\bar{x} = \frac{m_{10}}{m_{00}}, \quad \bar{y} = \frac{m_{01}}{m_{00}}$$

- Combine moments for rotation, scale invariance ....
- Examples: $\mu_{20} + \mu_{02}$, $\mu_{20}\mu_{02} - \mu_{11}^2$, $(\mu_{30} - 3\mu_{12})^2 + (3\mu_{21} - \mu_{03})^2$

Moments: special case

- Moment descriptors of orders 0-2 $\Rightarrow$ ellipse approximation



- Center of gravity:

$$\bar{x} = \frac{m_{10}}{m_{00}}, \quad \bar{y} = \frac{m_{01}}{m_{00}}$$

- Covariance matrix:

$$C = \begin{pmatrix} C_{xx} & C_{xy} \\ C_{xy} & C_{yy} \end{pmatrix}$$

$$C_{xx} = \mu_{20} = m_{20} - \bar{x} m_{10},\ C_{xy} = \mu_{11} = m_{11} - \bar{x} m_{01},\ C_{yy} = \mu_{02} = m_{02} - \bar{y} m_{01}$$

Moments: special case

- Lengths of main axes of the ellipse:

$$\begin{vmatrix} C_{xx} - \lambda & C_{xy} \\ C_{xy} & C_{yy} - \lambda \end{vmatrix} = 0$$

$$\lambda^2 - \underbrace{(C_{xx} + C_{yy})}_{trace\ C}\lambda + \underbrace{(C_{xx}C_{yy} - C_{xy}^2)}_{det\ C} = 0$$

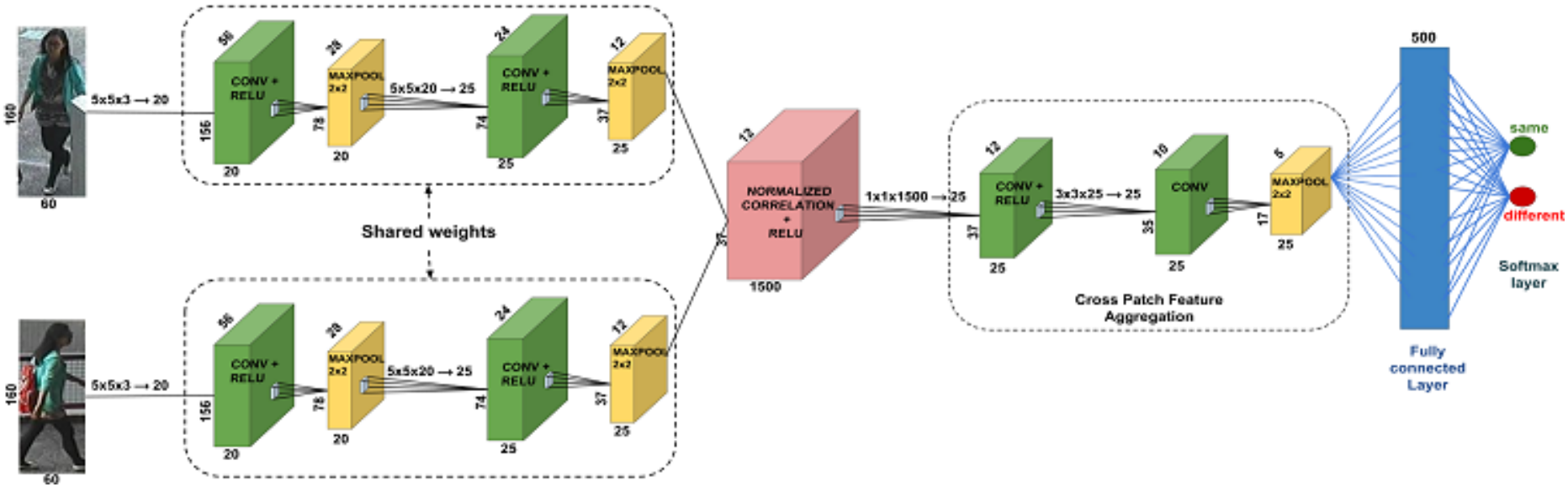
$$\lambda_{1,2} = \frac{\text{trace}(C)}{2} \pm \sqrt{\frac{\text{trace}(C)^2}{4} - \det(C)}$$

- Orientation:

$$\theta = \frac{1}{2}\tan^{-1}\left(\frac{2C_{xy}}{C_{xx} - C_{yy}}\right)$$

- Note: $\lambda_{1,2}$ is rotationally invariant, but orientation θ is not.

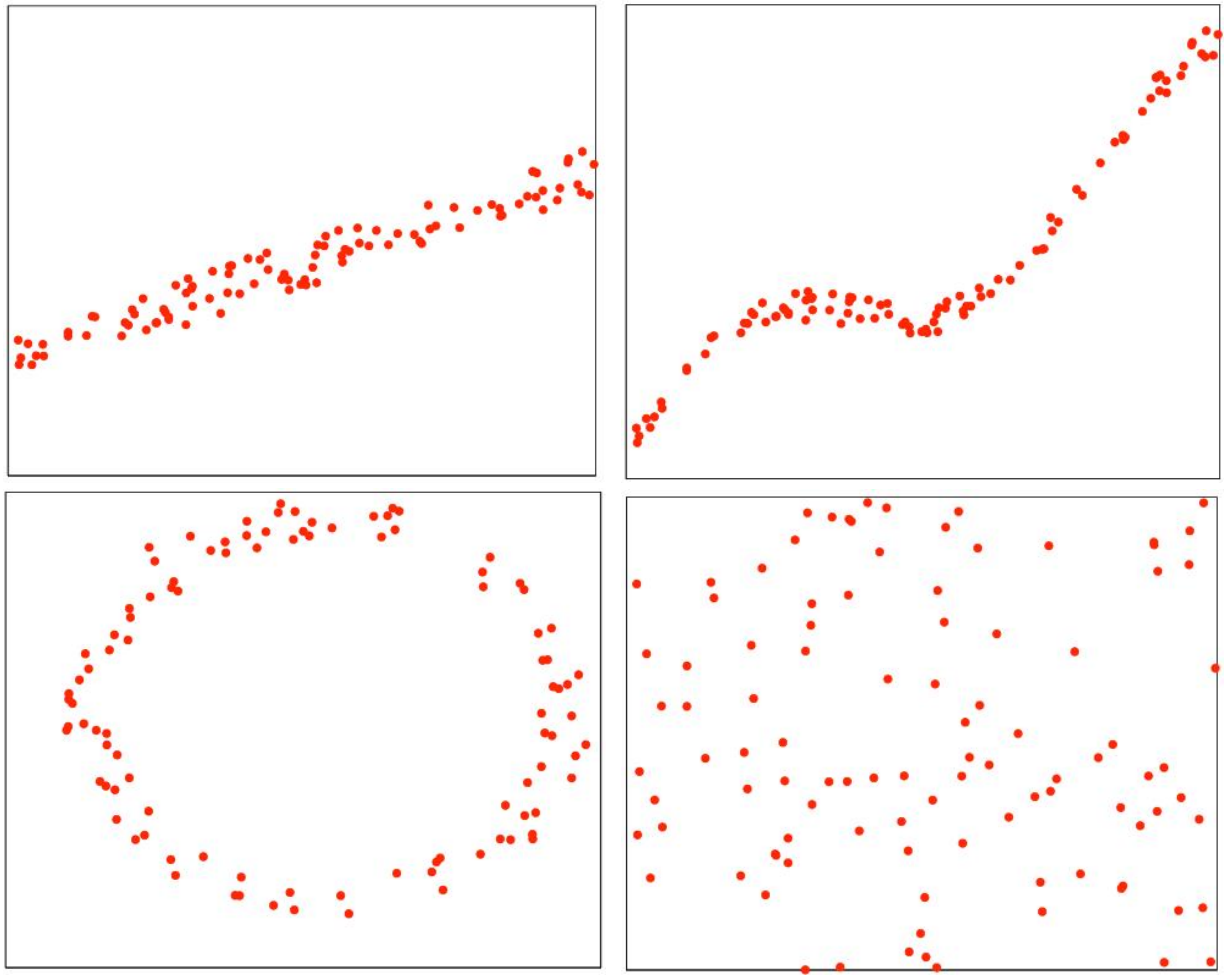
Siemese network for matching



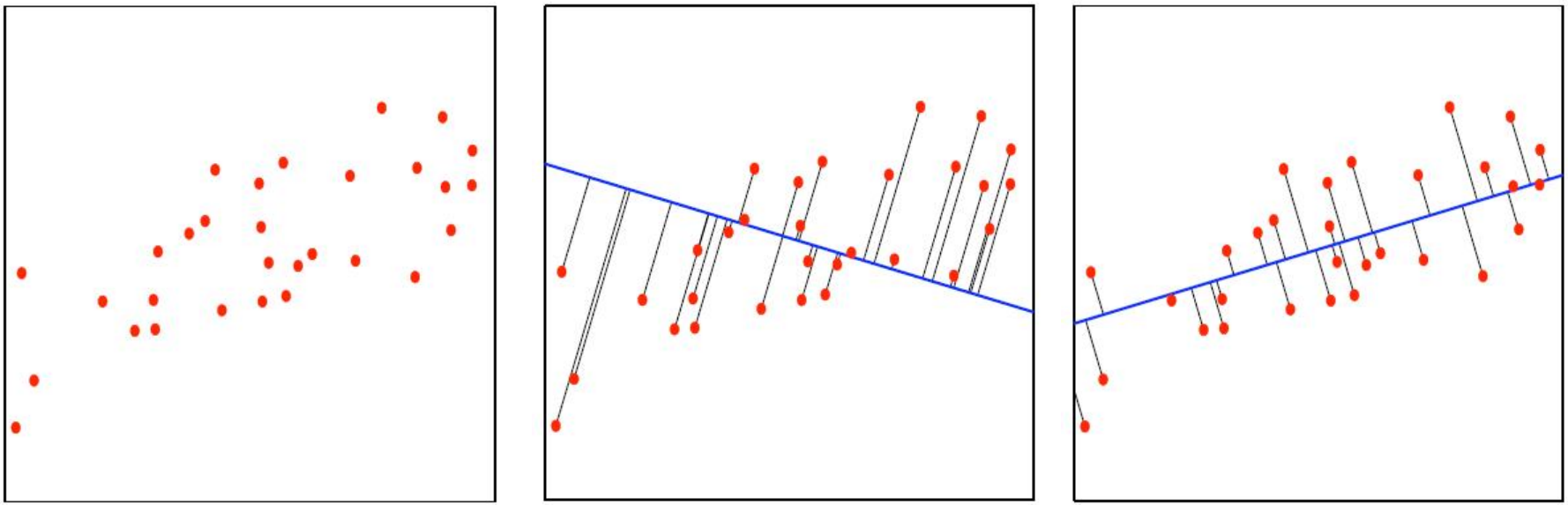
- Compute an image representation (features) with one network
- Correlate features between two images
- Process the correlated features with a second network
- Train with only two known values; same or different.

Dimensionality of data

What is the true dimensionality of this data?



Fit data to low-dimensional model



Dimensionality reduction (embedding)

- Assign instances to real-valued vectors, in a space that is of much lower dimension (even 2D or 3D for visualization).
- Approximately preserve similarity/distance relationships between instances.
- Some techniques:

  Linear: Principal components analysis (PCA)

  Non-linear:

  - Independent components analysis
  - Self-organizing maps
  - Gaussian process latent variable models
  - Deep auto-encoders

Basic idea

- Project high dimensional data onto a lower dimensional sub-space using linear or non-linear transformations.

$$x = \begin{bmatrix} a_1 \\ a_2 \\ \vdots \\ a_N \end{bmatrix} \Rightarrow \text{reduce dimensionality} \Rightarrow y = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_K \end{bmatrix} \quad (K << N)$$

- The goal is to reduce the dimensionality of data, while retaining as much as possible of the variation present in the dataset.

- Find a basis in a lower dimensional sub-space.
- Approximate vectors by projected them to the lower dim space.
  (1) Original space representation

$$x = a_1 v_1 + a_2 v_2 + ... + a_N v_N$$

  where $v_1, v_2, ..., v_N$ is a base in original N-dim space.
  (2) Lower dimensional sub-space representation

$$\hat{x} = b_1 u_1 + b_2 u_2 + ... + b_K u_K$$

  where $u_1, u_2, ..., u_N$ is a base in original K-dim sub-space.
- Note: if $K = N$, then $\hat{x} = x$.

Example: K=N

$$v_1 = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}, v_2 = \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}, v_3 = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \text{ (standard basis)}$$

$$x_v = \begin{bmatrix} 3 \\ 3 \\ 3 \end{bmatrix} = 3v_1 + 3v_2 + 3v_3$$

$$u_1 = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}, u_2 = \begin{bmatrix} 1 \\ -1 \\ 0 \end{bmatrix}, u_3 = \begin{bmatrix} 1 \\ 1 \\ -2 \end{bmatrix} \text{ (another basis)}$$

$$x_u = \begin{bmatrix} 3 \\ 3 \\ 3 \end{bmatrix} = 3u_1 + 0u_2 + 0u_3$$

Thus $x_v = x_u$, but represented with another base.

PCA: Methodology

Suppose $x_1, x_2, ..., x_M$ are $N \times 1$ vectors.

- Step 1: Compute the mean

$$\bar{x} = \frac{1}{M} \sum_{i=1}^{M} x_i$$

- Step 2: Subtract the mean

$$\Phi_i = x_i - \bar{x}$$

- Step 3: From the matrix $A = [\Phi_1 \Phi_2 ... \Phi_M]$, compute

$$C = \frac{1}{M} \sum_{i=1}^{M} \Phi_i \Phi_i^T = A\, A^T (\text{sample covariance matrix}, N \times N)$$

- Step 4: Compute the eigenvalues of C : $\lambda_1 > \lambda_2 > ... > \lambda_N$.

PCA: Methodology

- Step 5: Compute the eigenvectors of C : $u_1, u_2, ..., u_n$. The eigenvectors form a basis for $x - \bar{x}$.

$$x - \bar{x} = b_1 u_1 + b_2 u_2 + ... + b_N u_N = \sum_{k=1}^{N} b_k u_k; \;\; b_k = u_k^T (x - \bar{x})$$

- Step 6 (dimensionality reduction step): Keep only the terms corresponding to the K largest eigenvalues:

$$\hat{x} - \bar{x} = \sum_{k=1}^{K} b_k u_k; \;\; K << N$$

The representation of $\hat{x} - \bar{x}$ in basis $u_1, u_2, ...u_K$ is thus $\begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_K \end{bmatrix}$.

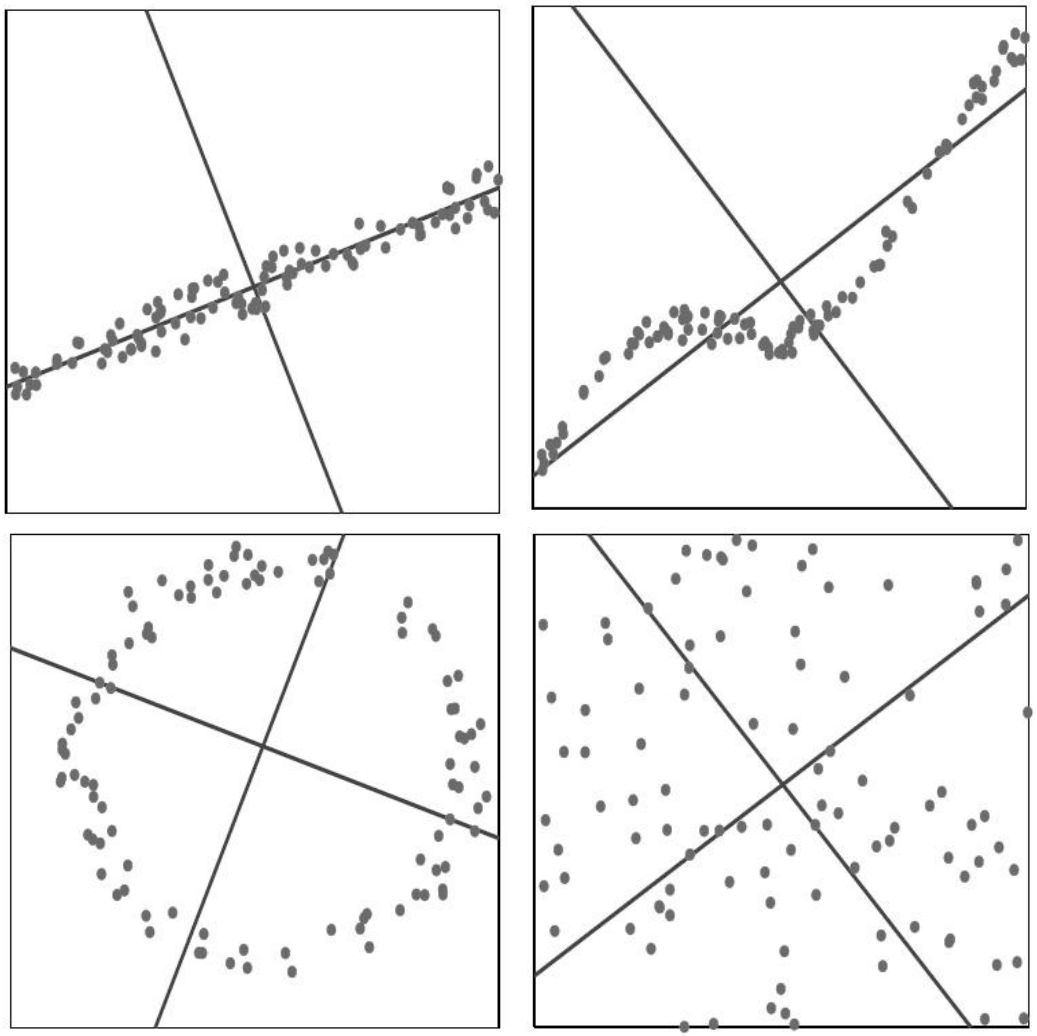
PCA: Methodology

The linear transformation $R^N \rightarrow R^K$ that performs the dimensionality reduction is:

$$\begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_K \end{bmatrix} = \begin{bmatrix} u_1^T \\ u_2^T \\ \vdots \\ u_K^T \end{bmatrix} (x - \bar{x}) = U^T (x - \bar{x})$$

- PCA projects the data along the directions, where the data varies the most.
- These directions are determined by the eigenvectors of the covariance matrix corresponding to the largest eigenvalues.
- The magnitude of the eigenvalues corresponds to the variance of the data along the eigenvector directions.

Example



Think bigger! - Face images

- Assume you have 92×112 grayscale images of faces. Each image is considered as a point in high-dimensional face space.

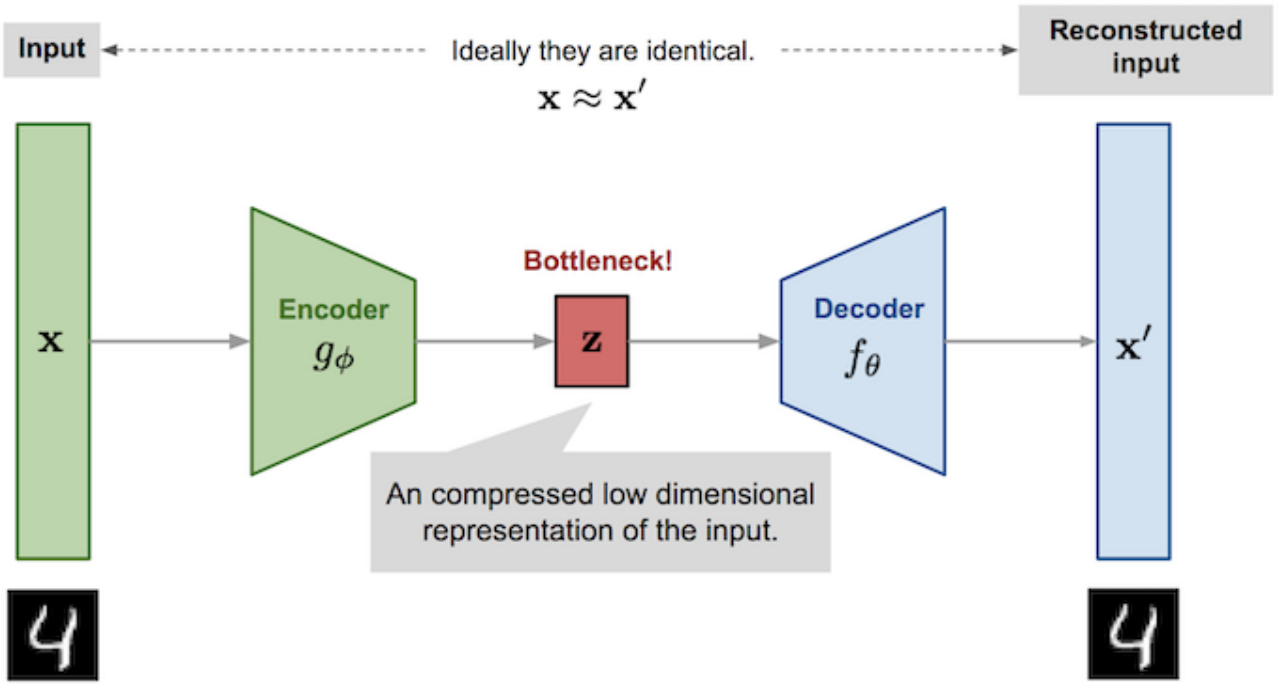
$$\Rightarrow \begin{bmatrix} * \\ * \\ * \\ * \\ * \\ \vdots \\ * \end{bmatrix}$$

- Stack all pixels into a large vector of size $n = 92 \cdot 112 = 10304$ and apply principal component analysis.

Eigenfaces

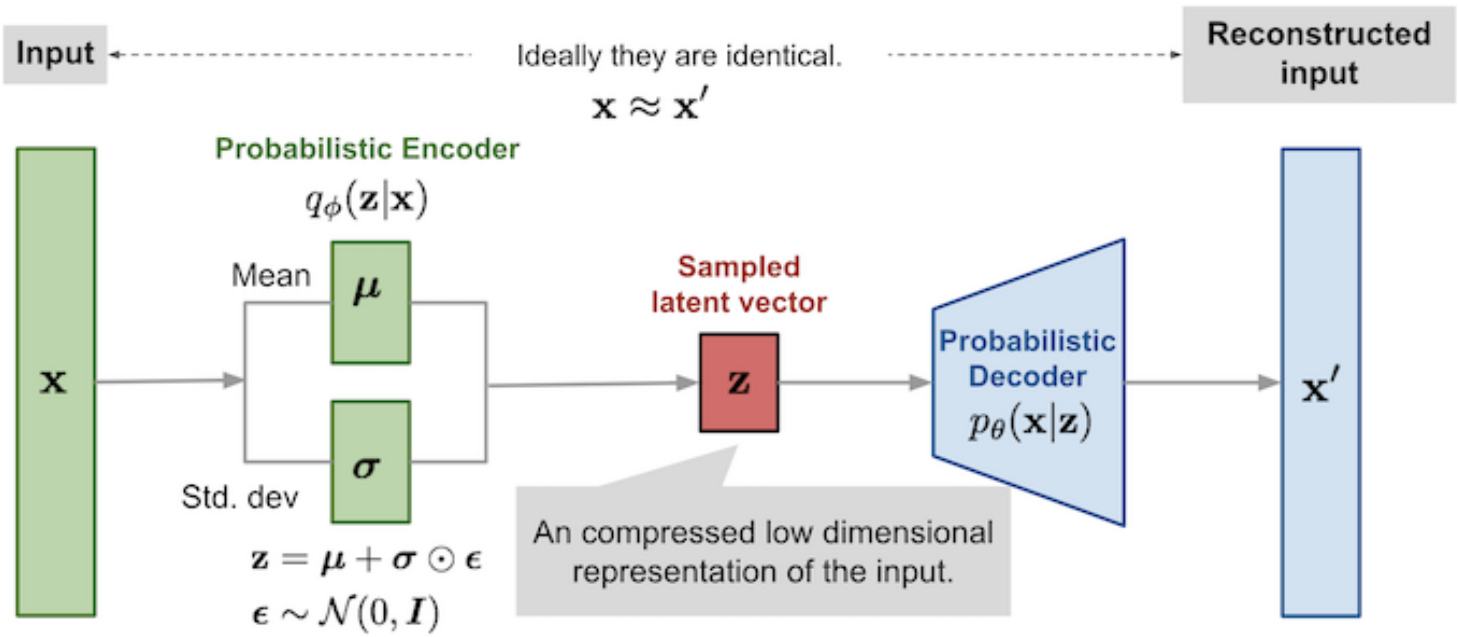
Eigenvectors (eigenfaces) corresponding to the largest eigenvalues for a large set of face images. Upper-left image is the mean.

Autoencoder for image representation learning



- Encoder: Gradually reduce number of neurons to a small latent space
- Decoder: Gradually increase size to that of original image
- Train network by using the input image as the target output
- Use the bottleneck neurons as a low-dimensional image representation

Variational autoencoder



- Problems with a regular autoencoder: We have no control over the shape of the latent space, which might look very strange.
- Solution:
  - Model each input point as a distribution in latent space, $q_\psi(z|x)$).
  - Train with the additional objective that the whole latent distribution should be Gaussian, $z \in \mathcal{N}(0, I)$.
  - Think of many small balls being forced to fit in a larger ball.

Latent space visualization

Move around in latent space and only use the decoder of a variational autoencoder (forces latent data to be of a given distribution)

Summary of good questions

- How does a Hough transform work for lines?
- How many accumulators should you use?
- How does RANSAC work?
- What is a homography?
- In what ways can two shape descriptors be different?
- Can you give two examples of two shape descriptors?
- How do you compute moment descriptors?
- What do you like to preserve with dimensionality reduction?
- How does a PCA work?

Readings

- Gonzalez and Woods: Chapters 10.2 and 11.3 - 11.5
- Szeliski: Chapters 5.2.3, 7.4 and 8.1