panikzettel.philworld.de

Computer Vision Panikzettel

Hans Wurst

September 3, 2021

Contents

1 Image Processing

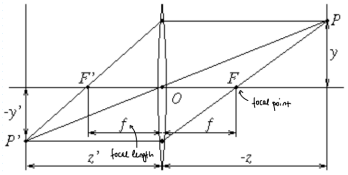
1.1 Image Formation

Lenses: Increasing pinhole size in *pinhole camera* to increase amount of light causes blur. *Lenses* keep image in sharp focus while gathering light from a larger area.

Thin Lens Model: valid if lens thickness is small compared to the radius of curvature. Thin lens equation:

$$\frac{1}{z'} - \frac{1}{z} = \frac{1}{f}$$

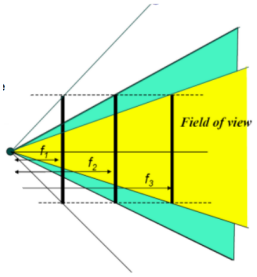
In a thin lens the scene points at distinct depths come in focus at different image planes.



Depth of Field: Distance between image planes where blur is tolerable, a smaller aperture (Blende) increases the range in which the object is approx. in focus.

Field of view depends on focal length f:

- $f \downarrow$: image becomes more *wide angle*, more world points project into the finite image plane
- $f \uparrow$: image becomes more *telescopic*, smaller part of the world projects onto the finite image plane



1.2 Linear Filters

Types of noise (*i.i.d. = "independent, identically distributed"*):

- *Salt and pepper noise*: random occurrences of black and white pixels
- *Impulse noise*: random occurrences of white pixels
- *Gaussian noise*: variations in intensity drawn from a Gaussian distribution

$$f(x,y) = \underbrace{\bar{f}(x,y)}_{\textit{ideal image}} + \underbrace{\eta(x,y)}_{\text{noise process}}, \quad \eta(x,y) \sim N(\mu,\sigma)$$

Correlation Filtering
Replace each pixel by a weighted combination of its neighbors.

$$\begin{aligned} G[i,j] &= \sum_{u=-k}^{k} \sum_{v=-k}^{k} H[u,v]F[i+u,j+v] \\ &= H \otimes F \end{aligned}$$

Convolution Filtering
Flip the filter in both dimensions, then apply correlation.

$$\begin{aligned} G[i,j] &= \sum_{u=-k}^{k} \sum_{v=-k}^{k} H[u,v]F[i-u,j-v] \\ &= H \star F \end{aligned}$$

with averaging window size $2k+1 \times 2k+1$, input image F, output image G, kernel/mask with non-uniform weights H. Convolution is separable into rows and columns 1D filters *(linear)*.
(If $H[u,v] = H[-u,-v]$, then correlation = convolution.)

Gaussian Smoothing
Weigh nearby pixels more than distant ones ($\rightarrow$ "fuzzy blob") using *Gaussian Kernel* with *variance* σ^2:

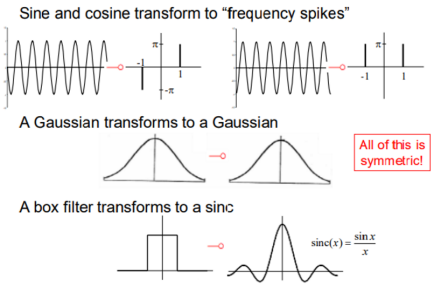
$$G_\sigma = \frac{1}{2\pi\sigma^2} e^{-\frac{(x^2+y^2)}{2\sigma^2}}$$

BOX filter
For every pixel, average every neighbor over the number of neighbors.

Fourier

- Map function onto a *frequency spectrum*.
- The better a function is localized in one domain, the worse it is localized in the other. ("compression" —○ "stretching", vice versa)
- Convolving in *image domain* corresponds to product in *frequency domain*:

$$f \star g \multimap \mathcal{F} \cdot \mathcal{G}$$



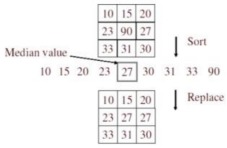
Noise introduces high frequencies: Gaussian as a convenient choice for a *low-pass filter*.

1.3 Nonlinear Filters

Median Filter

Replace each pixel by the *median* of its neighbors.
Properties:

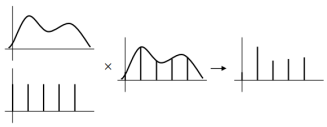
- doesn't introduce new pixel values
- removes spikes: good for impulse, salt and pepper noise
- non-linear
- edge preserving
- better results than Gaussian, even with small kernel



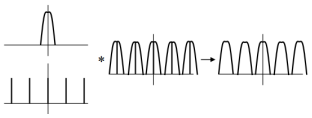
1.4 Multi-Scale Representations

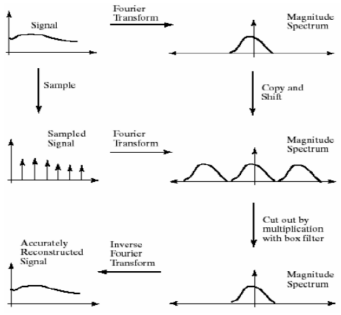
Sampling

Sampling in the spatial domain is like multiplying with a spike function.



Sampling in the frequency domain is like convolving with a spike function.

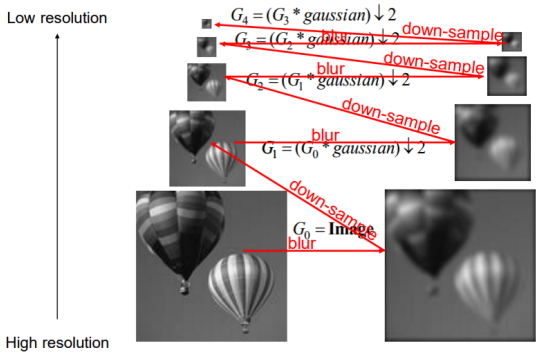




Nyquist theorem: In order to recover a certain frequency f, we need to sample with at least $2f$. This corresponds to the point at which the transformed frequency spectra start to overlap *(Nyquist limit)*.

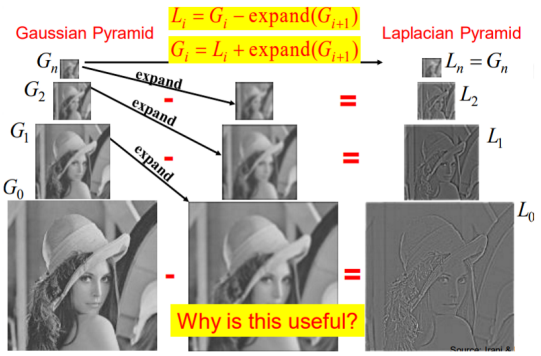
Image pyramids provide an efficient representation for *space-scale invariant* processing.

Gaussian Pyramid



- Create each level from previous one with *"smooth and sample"-principle*.
- Smooth with Gaussians, in part because $G(\sigma_1) * G(\sigma_2) = G(\sqrt{\sigma_1^2 + \sigma_2^2})$.
- Gaussians are low-pass filters, so the representation is redundant once smoothing is performed: No need to store smoothed images at the full original resolution.

Laplacian Pyramid



- Laplacian $\sim$ Difference of Gaussians *(DoG)*: cheap approximation

1.5 Edge Detection

1.5.1 Filters as Templates

Think of filters as a dot product of the filter vector with the image region. Measure the angle between the vectors using $\cos\theta = \frac{a \cdot b}{|a||b|}$. Angle (similarity) between vectors can be measured by normalizing the length of each vector to 1 and taking the dot product. Filters look like the effects they are intended to find, and they find effects they look alike.

1.5.2 Image gradients

For partial derivative filters we get $H_x = (1, 0, -1) = H_y^T$. Images of partial derivatives are slightly shifted, therefore, shouldn't be used.

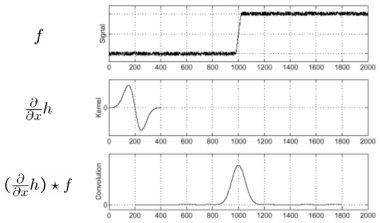
- $\nabla f = [\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}]$: *gradient*, points in the direction of *most rapid intensity change*
- $\theta = tan^{-1}(\frac{\partial f}{\partial y} / \frac{\partial f}{\partial x})$: *gradient direction* (orientation of edge normal)
- $\|\nabla f\| = \sqrt{(\frac{\partial f}{\partial x})^2 + (\frac{\partial f}{\partial y})^2}$: *gradient magnitude* (edge strength)

1.5.3 2D Edge Detection Filters

Gradient filter amplifies noise, *smooth* with Gaussian first.

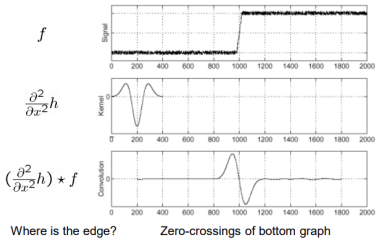
Derivative of Gaussian (DoG) $\frac{\partial}{\partial x}h_\sigma(u,v)$

$$\begin{aligned}\frac{\partial}{\partial x}(h \star f) &= (\frac{\partial}{\partial x}h) \star f \\ &= ((1,0,-1) \star h) \star f\end{aligned}$$



Laplacian of Gaussian (LoG) $\nabla^2 h_\sigma(u,v)$
$(\nabla^2 f = \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2})$

$$\frac{\partial^2}{\partial x^2}(h \star f) = (\frac{\partial^2}{\partial x^2}h) \star f$$

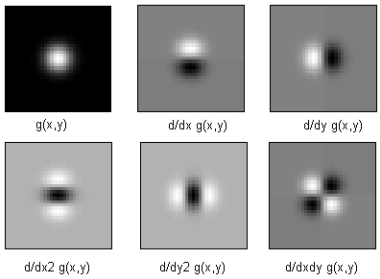


Where is the edge? Zero-crossings of bottom graph

Image Derivatives

G is 1D Gaussian filter, D is 1D derivative of Gaussian filter, and I is the image:

$$\begin{aligned}I_x &= G^T \star (D \star I) \\ I_y &= D^T \star (G \star I) \\ I_{xx} &= G^T \star (D \star I_x) \\ I_{yy} &= D^T \star (G \star I_y) \\ I_{xy} &= D^T \star (G \star I_x)\end{aligned}$$



1.5.4 Canny Edge Detector

An "optimal" edge detector should have *good detection*, *good localization*, and *single response*.

Primary edge detection steps:

1. *Smoothing:* Suppress noise.
2. *Edge enhancement:* Filter for contrast.
3. *Edge localization*
   - Determine which local maxima from filter output are actually edges vs. noise.

- Thresholding, thinning.

Scale:
Use σ of the Gaussian to set the scale at which edges will be later extracted.

Sensitivity:

$$F_T[i,j] = \begin{cases} 1, \text{ if } F[i,j] \geq t \quad (on) \\ 0, \text{ otherwise} \quad (off) \end{cases}$$

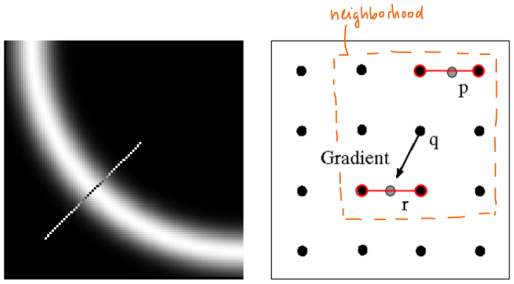
where t threshold, $F[i,j]$ pixel.

Canny Edge Detector steps:

1. Filter image with *derivative of Gaussian*.
2. Find *magnitude* and *orientation* of gradient.
3. *Non-maximum suppression:* Thin multi-pixel wide "ridges" down to single pixel width.

Check, if pixel is local maximum along gradient direction, select single max across width of the edge.

a) Compute interpolated pixels p and r
b) Keep q iff $Mag(q) > Mag(p)$ and $Mag(q) > Mag(r)$



4. *Linking* and (hysteresis) *thresholding*

- Define two thresholds: k_{low} and k_{high} ($k_{high}/k_{low} = 2$)
- Use the high threshold to start edge curves and the low threshold to continue them, until no pixel along the edge is above the low threshold.

1.6 Fitting Techniques

1.6.1 Hough Transform

Many objects are characterized by presence of straight lines.
Voting technique *Hough Transform* answers the three main questions of *Line Fitting*:

- Given points that belong to a line, what is the line?
- How many lines are there?
- Which points belong to which lines?

Idea:

1. Vote for all possible lines on which each edge point could lie.
2. Look for line candidates that get many votes.
3. Noise features will cast votes too, but their votes should be inconsistent.

Hough Space

- set of points $(x, y) \mapsto (m, b)$ such that $y = mx + b$
  (a line in the image corresponds to a point in Hough space)
- point $(x_0, y_0) \mapsto (m, -x_0 m + y_0) = (m, b)$
  (a point in the image corresponds to a line in Hough space)
- two points $(x_0, y_0), (x_1, y_1)$ correspond to $b = -x_0 m + y_0 \stackrel{!}{=} -x_1 m + y_1$
  (two points in the image correspond to the intersection of the two lines, which is a point in Hough space)

Polar Representation for lines ((m, b) representation undefined for vertical lines, infinite values)

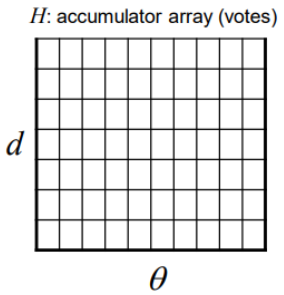
$$x \cdot \cos\theta + y \cdot \sin\theta = d$$

with d the perpendicular distance from line to origin, θ angle d makes with the x-axis. A point in image space corresponds to *sinusoid* segment in Hough space.

Hough Transform Algorithm

Let each edge point in image space *vote* for a set of possible parameters in Hough space. The Hough transform subdivides the Hough space into a *discrete set of bins*. Increase the vote count in each bin that the line passes through. Find peaks in local maxima of the Hough space ($\rightarrow$ Non-maximum suppression filter, is the value in center larger that the values of its 8 neighbors?).

1. Init: $H[d, \theta] = 0$
2. For each edge point (x, y) in the image
   For $\theta = 0$ to 180
   $d = x \cdot \cos\theta + y \cdot \sin\theta$
   $H[d, \theta] += 1$
3. Find the value(s) of $(\hat{d}, \hat{\theta})$ where $H[\hat{d}, \hat{\theta}]$ is maximal
4. Detected line in the image is given by
   $\hat{d} = x \cdot \cos\hat{\theta} + y \cdot \sin\hat{\theta}$



Noise makes the maximum point in Hough space spread over a larger area.

Pros and Cons:

+ all points are processes independently, so can cope with occlusion
+ some robustness to noise, noise points unlikely to contribute consistently
+ can detect multiple instances of a model in a single pass

- complexity of search time increases *exponentially* with number of model parameters
- non-target shapes can produce spurious peaks in parameter space
- *quantization*: hard to pick a good grid size

Extensions

1) Use the image gradient instead of iterating over all possible directions θ:

$$\theta = gradient\ at\ (x,y) = tan^{-1}(\frac{\partial f}{\partial y}/\frac{\partial f}{\partial x})$$

$\rightarrow$ Reduces degrees of freedom on voting space

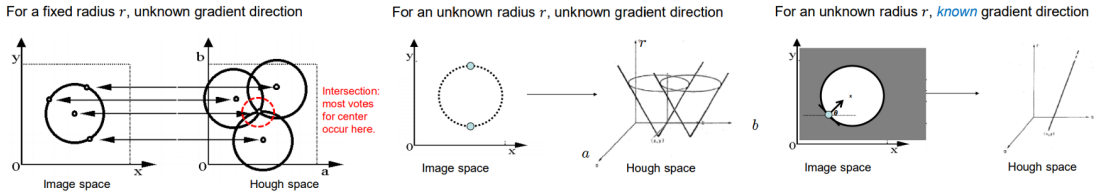
2) Give more votes for stronger edges (use magnitude of gradient).

3) Change the sampling of (d,θ) give more/less resolution.

4) The same procedure can be used with circles, squares, or any other shape.

Extension to Circles

Circle equation with center (a,b), radius r: $(x_i - a)^2 + (y_i - b)^2 = r^2$

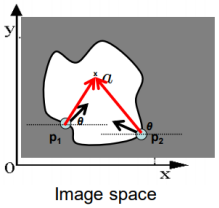


Algorithm:

For every edge pixel (x,y)
  For each possible radius value r
    For each possible gradient direction θ // *or use estimated gradient*
      $a = x - r\cos\theta$
      $b = y + r\sin\theta$
      $H[a,b,r] += 1$

Generalized Hough Transform

For arbitrary shapes defined by boundary points and a reference point, with fixed orientation and scale (e.g. only translation as transformation).



At each boundary point, compute displacement vector: $r = a - pi$.

For a given model shape: store these vectors in a table indexed by gradient orientation θ.

To *detect* the model shape in new image:

- For each edge point
  - Index into table with its gradient orientation θ (edge orientation) and computed *displacement vector* $r = a - p_i$ with reference point $a = (x_c, y_c)$, given

as $r := (r_k, \alpha_k)$, where r_k is the vectors length and α_k the vector orientation.
  - Use retrieved r vectors to vote for position of reference point. The accumulator array has the two coordinates of unknown object center a as axes.
    $x_c = x_i \pm r_k \cos \alpha_k$
    $y_c = y_i \pm r_k \sin \alpha_k$
    $A(x_c, y_c) += 1$
- Peak in this Hough space is reference point with most supporting edges.

1.6.2 RANSAC (RANdom SAmple Consensus)

Alternative strategy for Line Fitting: Sample points and fit a line to them using least-squares regression. RANSAC only returns a "good" result with a certain probability, but this probability increases with the number of iterations.

RANSAC loop

1. Randomly select a *seed group* of points on which to base transformation estimate (e.g., a group of matches).
2. Compute transformation from seed group.
3. Find inliers to this transformation. (inlier = points within a certain distance to the transformation/line)
4. If the number of inliers is sufficiently large, re-compute *least-squares estimate* of transformation on all of the inliers.

→ Keep the transformation with the largest number of inliers!

5. Repeat until some termination criterion is met (e.g. #iterations).

Improve this initial estimate with estimation over all inliers (e.g. with standard least-squares minimization). But this may change inliers, so alternate fitting with reclassification as inlier/outlier.

RANSAC is not limited to fitting lines, it also can be applied to *arbitrary transformation models*. Then the inliers are those points whose transformation error (distance of transformed point to its corresponding point in the other image) is below a certain threshold.

In many practical situations, the percentage of outliers is often very high ($\geq$ 90%), but RANSAC is only applicable with $<$ 50% of outliers. In this case, use Generalized Hough Transform instead.

2 Segmentation

Gestalt factors: (make intuitive sense, but are very difficult to translate into algorithms)

- proximity, similarity, common fate, common region

- parallelism, symmetry, continuity, closure

2.1 Segmentation as Clustering

Best cluster centers are those that minimize *SSD* (Sum of Squared Distances) between all points and their nearest cluster center c_i:

$$\sum_{\text{clusters } i} \sum_{\substack{\text{points } p \\ \text{in cluster } i}} \|p - c_i\|^2$$

2.1.1 k-Means

1. Randomly initialize the cluster centers.
2. Determine points in each cluster: For each point p, find closest c_i. Put p into cluster i.
3. Set c_i to be the mean of points in cluster i.
4. If c_i has changed, repeat Step 2.

k-Means++: prevents arbitrarily bad local minima

1. Randomly choose first center.
2. Pick new center with prob. proportional to $\|p - c_i\|^2$ for $i \in [1, k-1]$ centers
3. Repeat until k centers.

$\rightarrow$ expected error = $\mathcal{O}(\log k) \cdot optimal$

Feature Space

Feature space depends on grouping by pixels:

- intensity similarity (1D intensity value as feature space)
- color similarity (3D color value as feature space)
- texture similarity (24D filter bank responses as feature space)
- intensity+position similarity (simple way to encode both similarity and proximity)

k-Means for Clustering:

1. Collect feature vectors for all pixels in an image.
2. Apply k-Means with predefined k segments/clusters on those vectors.
3. Assign one segment per cluster.

Pros and Cons:

+ simple, fast to compute
+ converges to local minimum of within-cluster squared error (always finds some local minimum)

- setting k
- sensitive to initial centers and outliers
- detects spherical clusters only
- assuming means can be computed
- *NP-hard*, even with $k = 2$

2.1.2 Probabilistic Clustering

Instead of treating the data as a bunch of points, assume, that they are all generated by sampling a continuous function. This function is called a *generative model*, which is defined by a vector of parameters θ, which we want to compute.

Mixture of Gaussians (MoG)

K Gaussian blobs (ellipses) with means $\boldsymbol{\mu}_j$, covariance matrices $\boldsymbol{\Sigma}_j$, dim. D

$$p(\mathbf{x}|\theta_j) = \frac{1}{(2\pi)^{D/2}|\boldsymbol{\Sigma}_j|^{1/2}} exp\{-\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu}_j)^T\boldsymbol{\Sigma}_j^{-1}(\mathbf{x}-\boldsymbol{\mu}_j)\}$$

The likelihood of observing $\mathbf{x}$ is a weighted mixture of Gaussians, where blob j is selected with probability π_j:

$$p(\mathbf{x}|\theta) = \sum_{j=1}^{K} \pi_j p(\mathbf{x}|\theta_j), \quad \theta = (\pi_1, \boldsymbol{\mu}_1, \boldsymbol{\Sigma}_1, ..., \pi_M, \boldsymbol{\mu}_M, \boldsymbol{\Sigma}_M)$$

Expectation Maximization (EM)

Goal: Find blob parameters θ that maximize the likelihood function

$$p(data|\theta) = \prod_{n=1}^{N} p(\mathbf{x}_n|\theta)$$

Idea: Given the Gaussian shape, assign points to clusters; Given the assigned points, approximate Gaussian shape.

Approach:

1. Randomly initialize shape of Gaussian blobs.
2. *E-step:* Given current guess of blobs, compute ownership of each point.
3. *M-step:* Given ownership probabilities, update blobs to maximize likelihood function.
4. Repeat until convergence.

EM Algorithm

- Expectation-Maximization (EM) Algorithm
  - E-Step: softly assign samples to mixture components
    $$\gamma_j(\mathbf{x}_n) \leftarrow \frac{\pi_j \mathcal{N}(\mathbf{x}_n|\boldsymbol{\mu}_j, \boldsymbol{\Sigma}_j)}{\sum_{k=1}^{K} \pi_k \mathcal{N}(\mathbf{x}_n|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)} \quad \forall j = 1,\dots,K, \quad n = 1,\dots,N$$
  - M-Step: re-estimate the parameters (separately for each mixture component) based on the soft assignments
    $$\hat{N}_j \leftarrow \sum_{n=1}^{N} \gamma_j(\mathbf{x}_n) = \text{soft number of samples labeled } j$$
    $$\hat{\pi}_j^{\text{new}} \leftarrow \frac{\hat{N}_j}{N}$$
    $$\hat{\boldsymbol{\mu}}_j^{\text{new}} \leftarrow \frac{1}{\hat{N}_j} \sum_{n=1}^{N} \gamma_j(\mathbf{x}_n)\mathbf{x}_n$$
    $$\hat{\boldsymbol{\Sigma}}_j^{\text{new}} \leftarrow \frac{1}{\hat{N}_j} \sum_{n=1}^{N} \gamma_j(\mathbf{x}_n)(\mathbf{x}_n - \hat{\boldsymbol{\mu}}_j^{\text{new}})(\mathbf{x}_n - \hat{\boldsymbol{\mu}}_j^{\text{new}})^{\mathrm{T}}$$

How to Apply MoG Models?

- Application 1: Unsupervised Clustering
  - Collect feature vectors for all pixels in an image
  - Apply MoG clustering on those feature vectors
  - Assign one segment per cluster

  ⇒ *More expressive than k-means*

  ⇒ *But still does not work too well for segmentation…*
- Application 2: Semantic Segmentation
  - Learn one MoG model per semantic class C_k
  - Perform probabilistic classification by evaluating the posterior

    $$p(C_k|x) = \frac{p(x|C_k)p(C_k)}{\sum_j p(x|C_j)p(C_j)} \quad \text{“posterior probability”}$$

    $$p(x|C_k) = p(x|\theta_{C_k}) = \sum_{j=1}^{M_k} \pi_{k,j}\, p(x|\theta_{k,j}) \quad \text{“per-class likelihood”}$$

  ⇒ *Widely used in practice, e.g., for learning class-specific color models*

MoG Color Models for Segmentation:

1. User marks two regions for foreground and background.

2. Learn a MoG model for the color values in each region.
3. Use those models to classify all other pixels.

Pros and Cons:

+ probabilistic interpretation
+ soft assignments between data points and clusters
+ generative model, can predict novel data points
+ relatively compact storage

- local minima
- initialization (often a good idea to start with some k-means interations)
- need to know number of components (solution: model selection)
- numerical problems are often a nuisance (Ärger)

2.1.3 Model-free clustering

Mean-Shift Algorithm

1. Initialize random seed, and window W.
2. Calculate center of gravity (the *"mean"*) of W: $\sum_{x \in W} xH(x)$ (often Gaussian profile). In this case, H is the height of the corresponding histogram bin.
3. Shift the search window to the mean.
4. Repeat Step 2 until convergence.

To use mean-shift for clustering:

- *Cluster:* all data points in the attraction basin of a mode (= local maximum of the density of a given distribution).
- *Attraction basin:* the region for which all trajectories (Flugbahnen) lead to the same mode

1. Find features (color, gradients, texture, etc.).
2. Initialize windows at individual pixel locations.
3. Perform mean shift for each window until convergence.
4. Merge windows that end up near the same "peak" or mode. (for plateaus)

Speed-ups to mitigate computational complexity:

- Assign all points within radius r of end point to the mode.
- Assign all points within radius r/c of the search path to the mode.

Pros and Cons:

+ *model free:* does not assume any prior shape on data clusters
+ single parameter h (window size)
+ variable number of modes
+ robust to outlies

- output depends on window size
- window size selection is not trivial
- computationally expensive
- does not scale well with dimension of feature space

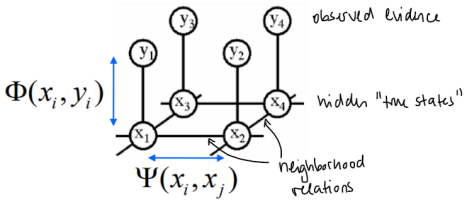
2.2 Graph-Theoretic Segmentation

2.2.1 Segmentation as Energy Minimization

Markow Random Fields (MRF)
MRF are a way to visualize dependencies in random variables.

Joint probability

$$p(x,y) = \prod_i \Phi(x_i, y_i) \prod_{i,j} \Psi(x_i, x_j)$$



with
x: scene $\quad \Phi$: image-scene compatibility func
y: image $\quad \Psi$: scene-scene compatibility func

Alternatively:

- x: true values of the scene, e.g. segment numbers
- y: observed raw pixel values
- Φ: probability, that we observe pixel value y, if the true pixel value is x
- Ψ: probability, that two adjacent pixels have the same or a different label

Maximizing the joint probability is the same as minimizing the negative logarithm of it:

$$-log\, p(x,y) = -\sum_i log\, \Phi(x_i, y_i) - \sum_{i,j} log\, \Psi(x_i, x_j) \Rightarrow \boxed{E(x,y) = \sum_i \phi(x_i, y_i) + \sum_{i,j} \psi(x_i, x_j)}$$

with

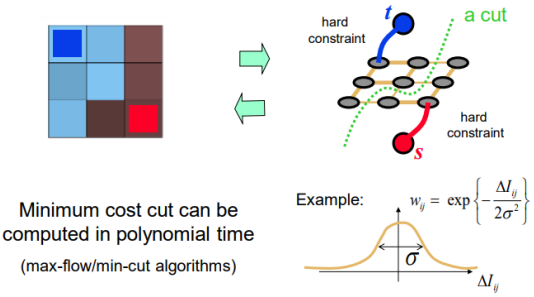
- E *energy function*
- ϕ *single-node/unary potentials*
  - Encode information about given pixel/patch: How likely is a pixel/patch to belong to a certain class?
- ψ *pairwise potentials*
  - Encode neighborhood information: How different is a pixel/patch's label from that of its neighbor?

2.2.2 Graph Cuts for Image Segmentation

Idea: Convert MRF into a source-sink graph.

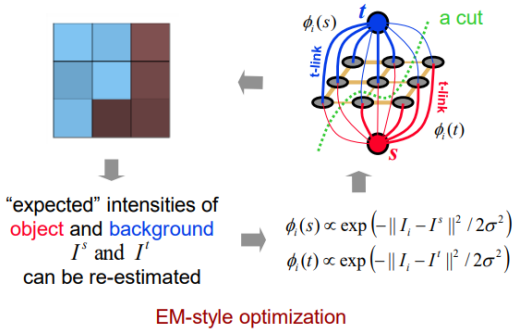
$$E(x,y) = \sum_i \underbrace{\phi_i(x_i)}_{\text{unary terms}} + \sum_{i,j} \underbrace{w_{ij} \cdot \delta(x_i \neq x_j)}_{\text{“Potts model”}}$$

$\delta = 1$, if condition $x_i \neq x_j$ is met, otherwise $\delta = 0$.



Instead of hard labeling, we add edges from t and s to every other node. This approach is called *regional bias*. For t, we use $\phi_i(s)$ and for s we use $\phi_i(t)$ instead of the hard constraints (binary object segmentation). Cutting through an edge with $\phi(t)$ means, that the node remains connected to t.

More generally, regional bias can be based on any intensity models of object and background.



How to set the potentials?

- Color potentials
  - e.g., modeled with a Mixture of Gaussians

$$\phi(x_i, y_i; \theta_\phi) = -\log \sum_k \theta_\phi(x_i, k) p(k|x_i) \mathcal{N}(y_i; \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)$$

- Edge potentials
  - E.g., a “contrast sensitive Potts model”

$$\varphi(x_i, x_j, g_{ij}(\mathbf{y}); \theta_\varphi) = \theta_\varphi g_{ij}(\mathbf{y}) \delta(x_i \neq x_j)$$

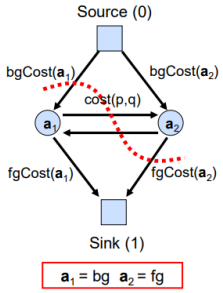
where

$$g_{ij}(\mathbf{y}) = e^{-\beta \|y_i - y_j\|^2} \qquad \beta = \frac{1}{2}\left(\text{avg}\left(\|y_i - y_j\|^2\right)\right)^{-1}$$

- Weight terms θ_ϕ, θ_ψ need to be learned, too!

Code:

```
Graph *g;
For all pixels p
    /* Add a node to the graph */
    nodeID(p) = g->add_node();
    /* Set cost of terminal edges */
    set_weights(nodeID(p), fgCost(p), bgCost(p));
end
for all adjacent pixels p,q
    add_weights(nodeID(p), nodeID(q), cost(p,q));
end
g->compute_maxflow();
label_p = g->is_connected_to_source(nodeID(p));
// is the label of pixel p (0 or 1)
```



Graph-Cuts Energy Minimization

Solve an equivalent graph cut problem:

1. Introduce extra nodes: *source* s and *sink* t.
2. Weigh connections to source/sink *(t-links)* by $\phi(x_i = s)$ and $\phi(x_i = t)$, respectively.
3. Weigh connections between nodes *(n-links)* by $\psi(x_i, x_j)$.
4. Find the minimum cost cut that separates source from sink.

$\Rightarrow$ Solution is equivalent to *minimum of the energy*.

s-t-Mincut Algorithm

Solve the dual maximum flow problem, the maximum flow equals the cost of the st-mincut, which needs to be minimized *(Min-cut/Max-flow Theorem)*. The cost of a st-cut is the sum of cost of all edges going from set S to set T.

Assuming non-negative capacity, we get the algorithm:

When applicable?

$$E(L) = \underbrace{\sum_p E_p(L_p)}_{\text{t-links}} + \underbrace{\sum_{pq \in N} E(L_p, L_q)}_{\text{n-links}} \qquad L_p \in \{s,t\}$$

(Unary potentials / Pairwise potentials)

- s-t graph cuts can only globally minimize binary energies that are submodular. [Boros & Hummer, 2002, Kolmogorov & Zabih, 2004]

E(L) can be minimized by s-t graph cuts $\Leftrightarrow$ $E(s,s)+E(t,t) \le E(s,t)+E(t,s)$ (Submodularity ("convexity"))

- Submodularity is the discrete equivalent to convexity.
  - Implies that every local energy minimum is a global minimum.

  $\Rightarrow$ Solution will be globally optimal.

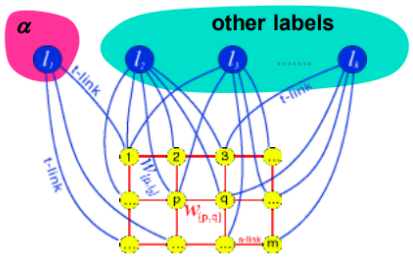
1. Find path from source to sink with positive capacity.
2. Push maximum possible flow through this path.
3. Adjust the capacity of the used edges and record "residual flows" (backwards flow).
4. Repeat until no path can be found.

α-Expansion

Algorithms for non-binary cases are no longer guaranteed to return the globally optimal result, but an approximation. Problem is *NP-hard* with 3 or more labels.

Idea: Break multi-way cut computation into a sequence of binary s-t cuts.

1. Start with any initial solution.
2. For each label "α" in any order:
   a) Compute optimal α-expansion move (s-t graph cuts).
   b) Decline the move if there is no energy decrease.
3. Stop when no expansion move would decrease energy.



Pros and Cons:

+ powerful technique, based on probabilistic model (MRF)
+ applicable for a wide range of problems
+ very efficient algorithms available
+ becoming a de-facto standard

- graph cuts can only solve a limited class of models: submodular energy functions, can capture only part of the expressiveness of MRFs
- only approximate algorithms available for multi-label cases

3 Object Recognition and Categorization

Idea:

- Represent each object (view) by a global descriptor (= feature vector).
- For recognizing objects, just match the descriptors.
- Some modes of variation are build into the descriptor, the others have to be incorporated in the training data.

Identification: Find a particular object. *Categorization:* Recognize any object of a class.

3.1 Sliding-Window Object Detection

Idea: If the object may be in a cluttered scene, slide a window around looking for it. Search over space and scale with the help of a *binary classifier*.

Therefore, we need to:

1. Obtain training data.
2. Define features.
3. Define classifier.

Pros and Cons:

+ simple detection protocol to implement
+ good feature choices critical
+ past successes for certain classes

- high computational complexity
- with so many windows, false positive rate better be low
- non-rigid, deformable objects not captured well with representations assuming a fixed 2D structure
- objects with less-regular textures not captured well with holistic appearance-based descriptions
- if considering windows in isolation, context is lost
- not all objects are "box" shaped

3.2 Gradient-based Representation

Idea: Consider edges, contours, and (oriented) intensity gradients. Summarize local distribution of gradients with *histograms*.

Histograms of Oriented Gradients (HoG)

Divide image window into small spatial regions ("cells"), map each grid cell in the

input window to a histogram, counting the matching gradients per orientations.

3.3 Classifier Construction

Learn a decision rule (classifier) assigning image features to different classes.

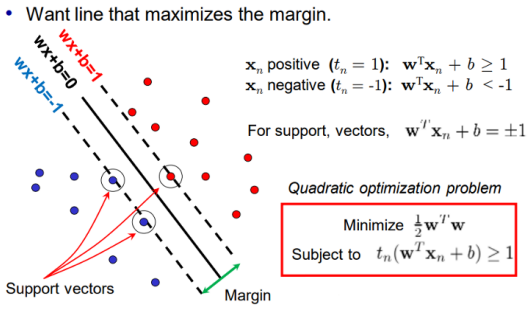
Line equation for a *linear classifier*, which separate positive and negative examples:

$$w_1 x_1 + w_2 x_2 + b = 0 \iff w^T x + b = 0$$

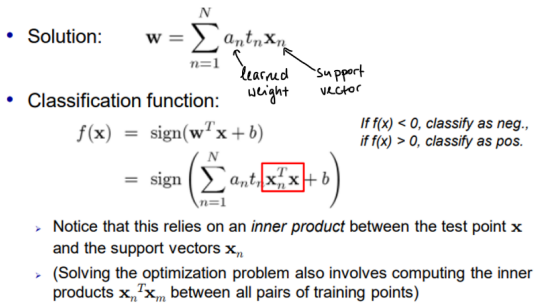
with w being the normal of the line. Then x_n is positive, if $w^T x_n + b \geq 0$, and negative, if $w^T x_n + b < 0$.

3.3.1 Classification with SVMs (Support Vector Machines)

Discriminative classifier based on optimal separating *hyperplane* (line for 2D case). Maximize the *margin* between the positive and negative training examples.



Finding the maximum margin line:



Extension: Non-Linear SVMs

Idea: The original input space can be mapped to some higher-dimensional feature space where the training set is separable ($x \mapsto \phi(x)$). For datasets, which cannot be separated by a *linear* hyperplane in 2D.

Kernel Trick: Instead of explicitly computing the lifting transformation $\phi(x)$, define a kernel function $K(\mathbf{x}_i, \mathbf{x}_j) = \phi(\mathbf{x}_i)^T \cdot \phi(\mathbf{x}_j)$. This gives a nonlinear decision boundary in the original feature space:

$$\sum_n a_n t_n K(\mathbf{x}_n, \mathbf{x}) + b$$

Since the optimization formulation uses the data points only in the form of inner products $\phi(\mathbf{x}_n)^T\phi(\mathbf{x}_m)$, we never need to actually compute the lifting transformation.

That's because we choose an already known kernel function seen below.

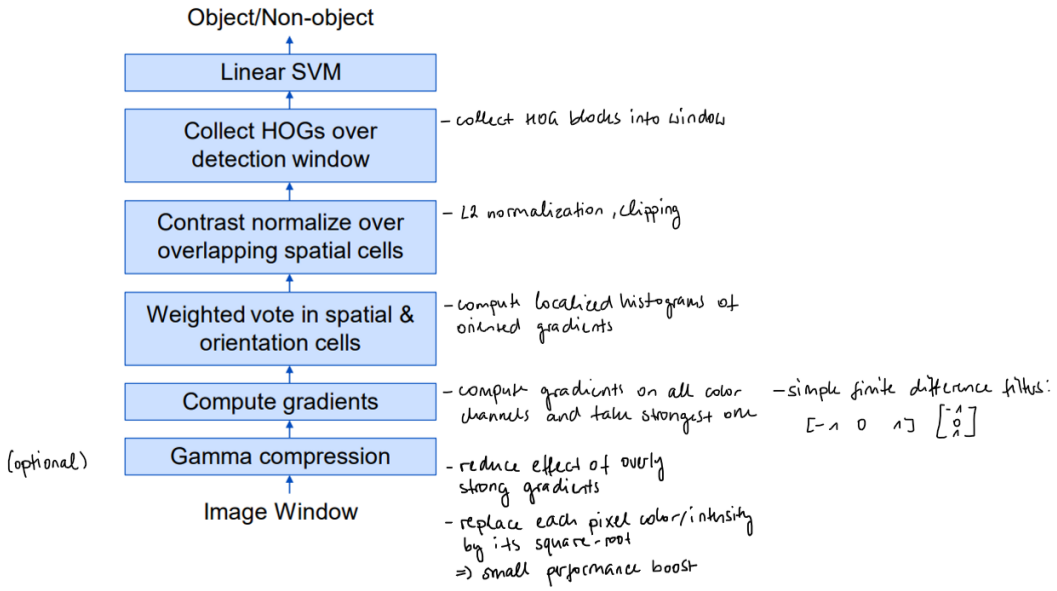
$$\underset{\text{linear}}{K(\mathbf{x}_i, \mathbf{x}_j) = \mathbf{x}_i^T \mathbf{x}_j}, \quad \underset{\text{polynomial of power } p}{K(\mathbf{x}_i, \mathbf{x}_j) = (1 + \mathbf{x}_i^T \mathbf{x}_j)^p}, \quad \underset{\text{Gaussian (Radial-Basis Function)}}{K(\mathbf{x}_i, \mathbf{x}_j) = \exp\left(-\frac{\|\mathbf{x}_i - \mathbf{x}_j\|^p}{2\sigma^2}\right)}$$

SVMs for Recognition:

1. Define your representation for each example.
2. Select a kernel function.
3. Compute pairwise kernel values between labeled examples.
4. Pass this "kernel matrix" to SVM optimization software to identify support vectors and weights.
5. To classify a new example: Compute kernel values between new input and support vectors, apply weights, check sign of output.

HOG (Histograms of Oriented Gradients) Detector

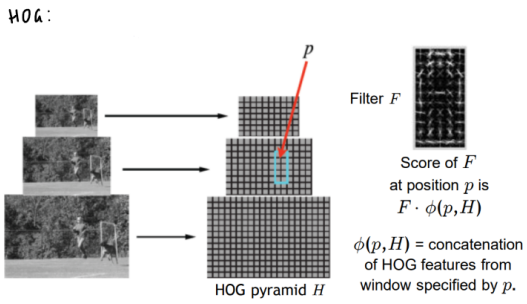
HOG descriptor processing chain:



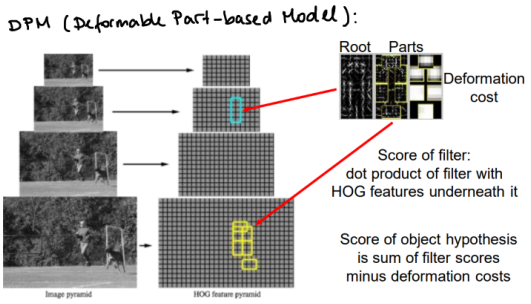
Assume linear SVM classification function $y(x) = w^T x + b$. Then x is the HOG feature map and w the template obtained by the SVM.

After a multi-scale dense scan, we want to suppress non-maximum detections. First, we clip the detection score (the negative). We map each detection to 3D $[x, y, scale]$ space. Subsequently, we apply a robust mode detection, e.g. mean shift.

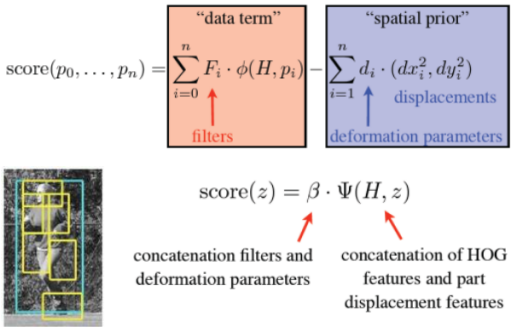
DPM Detector



- Array of weights for features in window of HOG pyramid
- Score is dot product of filter and vector



- Multiscale model captures features at two resolutions

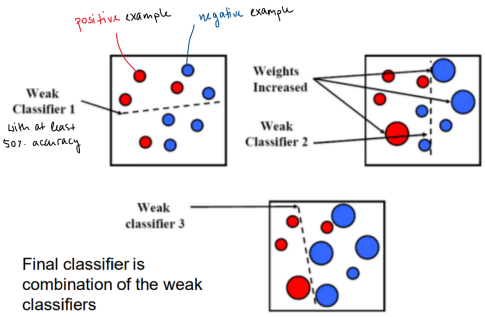


Deformation cost can be directly computed on top of the part filter responses through Generalized Distance Transform. Results into transformed responses.

3.3.2 Classification with Boosting

AdaBoost

Idea: Build a strong classifier H by combining a number of "weak classifiers" $h_1, \cdots, h_M$, which need only be better than chance. At each iteration, add a weak classifier *(sequential learning process)*.



Given: training set $\mathbf{X} = \{\mathbf{x}_1, \cdots, \mathbf{x}_N\}$ with target values $\mathbf{T} = \{t_1, \cdots, t_N\}$, $t_n \in \{-1, 1\}$, associated weights $\mathbf{W} = \{w_1, \cdots, w_N\}$ for each training point

Basic steps:

1. In each iteration, AdaBoost trains a new weak classifier $h_m(\mathbf{x})$ based on the current weighting coefficients $\mathbf{W}^{(m)}$.
2. Adapt the weighting coefficients for each point: Increase w_n if $\mathbf{x}_n$ was misclassified by $h_m(\mathbf{x})$, else decrease.
3. Make predictions using the final combined model

$$H(\mathbf{x}) = sign\left(\sum_{m=1}^{M} \alpha_m h_m(\mathbf{x})\right)$$

Detailed training algorithm:

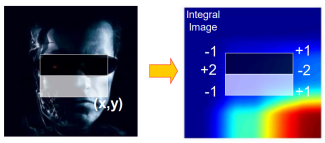
1. Initialization: Set $w_n^{(1)} = \frac{1}{N}$ for $n = 1,\ldots,N$.
2. For $m = 1,\ldots,M$ iterations
   a) Train a new weak classifier $h_m(\mathbf{x})$ using the current weighting coefficients $\mathbf{W}^{(m)}$ by minimizing the weighted error function
   $$J_m = \sum_{n=1}^{N} w_n^{(m)} I(h_m(\mathbf{x}_n) \neq t_n) \qquad I(A) = \begin{cases} 1, & \text{if } A \text{ is true} \\ 0, & \text{else} \end{cases}$$
   b) Estimate the weighted error of this classifier on $\mathbf{X}$:
   $$\epsilon_m = \frac{\sum_{n=1}^{N} w_n^{(m)} I(h_m(\mathbf{x}_n) \neq t_n)}{\sum_{n=1}^{N} w_n^{(m)}}$$
   c) Calculate a weighting coefficient for $h_m(\mathbf{x})$:
   $$\alpha_m = \ln\left\{\frac{1-\epsilon_m}{\epsilon_m}\right\}$$
   d) Update the weighting coefficients:
   $$w_n^{(m+1)} = w_n^{(m)} \exp\{\alpha_m I(h_m(\mathbf{x}_n) \neq t_n)\}$$

Recognition:

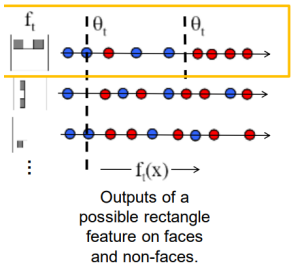
- Evaluate all selected weak classifiers on test data: $h_1(\mathbf{x}), \cdots, h_m(\mathbf{x})$
- Final classifier is weighted combination of selected weak classifiers, as seen in step 3 basic steps.

Viola-Jones Face Detection

Introduce a set of "rectangular" *Haar filters*: Subtract the pixels in the white region from the pixels in the black region. Efficiently computable with *integral image*, which is computed once per image: In every pixel (x, y) in the integral image we store the sum of pixels over the rectangle spanned by the pixel and the top left image corner.



Using AdaBoost for informative feature and classifier selection, we want to select the single rectangle feature and threshold that best separates positive (faces) and negative (non-faces) training examples, in terms of *weighted* error.

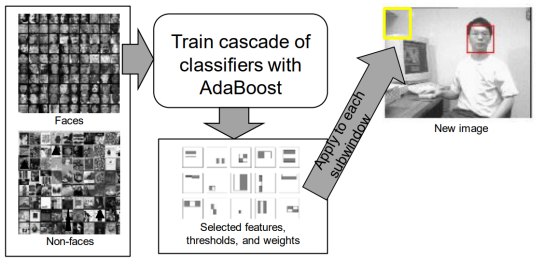


Resulting weak classifier:

$$h_t(x) = \begin{cases} +1 & \text{if } f_t(x) > \theta_t \\ -1 & \text{otherwise} \end{cases}$$

For next round, reweight the examples according to errors, choose another filter/threshold combo.

Even if the filters are fast to compute, each new image has a lot of possible windows to search. For efficiency, in a *cascade fashion* apply less accurate but faster classifiers first to immediately discard windows that clearly appear to be negative.



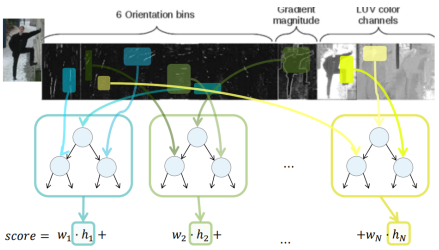
Extension: Integral Channel Features

Generalization of Haar integral image idea from Viola-Jones: Instead of only considering intensities, also take into account *other feature channels* (gradient orientations, color, texture).

Also generalize block computation:

- 1st order features: Sum of pixels in rectangular region *(integral over certain region)*
- 2nd order features: Haar-like difference of sum-over-blocks *(difference between the integrals over two regions)*
- Generalized Haar: More complex combinations of weighted rectangles *(higher order design with multiple such blocks)*
- Histograms: Computed by evaluating local sums on quantized images *(multiple orientation channels, build histogram over corresponding regions, similar to HOG)*

Classifier construct (e.g. VeryFast detector)



- Ensemble of short trees, learned by AdaBoost

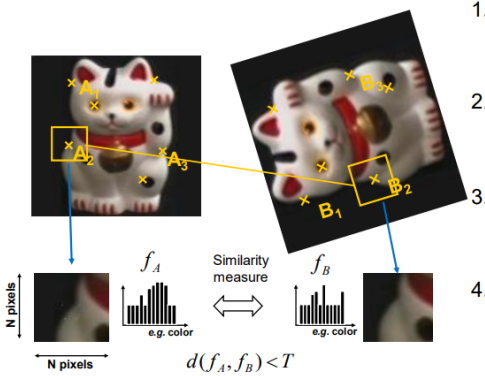
4 Local Features and Matching

4.1 Local Features - Detection and Description

4.1.1 Local Invariant Features

Global representations have major limitations. Instead, describe and match only local regions. Increased robustness to occlusions, articulation, intra-category variations.

General approach:



1. Find a set of distinctive key-points
2. Define a region around each keypoint
3. Extract and normalize the region content
4. Compute a local descriptor from the normalized region
5. Match local descriptors

Requirements:

- Region extraction needs to be *repeatable* and *accurate*.
  - Invariant to translation, rotation, scale changes.
  - Robust or covariant to out-of-plane ($\approx$ affine) transformations.
  - Robust to lighting variations, noise, blur, quantisation.
- *Locality:* Features are local, therefore robust to occlusion and clutter.
- *Quantity:* We need a sufficient number of regions to cover the object.
- *Distinctiveness:* The regions should contain "interesting" structure.
- *Efficiency:* Close to real-time performance.

4.1.2 Keypoint Localization

Look for two-dimensional signal changes: In the region around a *corner*, image gradient has two or more dominant directions.

Harris Detector

Formulation:

- Start from the second-moment matrix M :

$$M = \sum_{x,y} w(x,y) \begin{bmatrix} I_x^2 & I_xI_y \\ I_xI_y & I_y^2 \end{bmatrix}$$

Gradient with respect to x, times gradient with respect to y

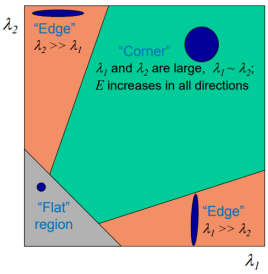
Sum over image region – the area we are checking for corner

$$M = \begin{bmatrix} \sum I_xI_x & \sum I_xI_y \\ \sum I_xI_y & \sum I_yI_y \end{bmatrix} = \sum \begin{bmatrix} I_x \\ I_y \end{bmatrix} [I_x \; I_y]$$

Since M is symmetric, we can execute a eigenvalue decomposition, where λ should be large, when we are looking at an axis-aligned corner:

$$M = R^{-1} \begin{pmatrix} \lambda_1 & 0 \\ 0 & \lambda_2 \end{pmatrix} R$$

We can visualize M as an ellipse with axes lengths determined by the eigenvalues and orientation determined by R.



Corner Response Function, with constant α for fast approximation:

$$R = \det(M) - \alpha \; \text{trace}(M)^2 = \lambda_1\lambda_2 - \alpha(\lambda_1 + \lambda_2)^2$$

Then $R < 0$, if "edge", and $R > 0$, if "corner".

Properties:

- R is invariant to image rotation (ellipse shape, therefore the eigenvalues stay the same)
- not invariant to image scale (due to fixed window size)

Algorithm:

1. Compute second moment matrix *(autocorrelation matrix)*:

$$M(\sigma_I, \sigma_D) = g(\sigma_I) \star \begin{pmatrix} I_x^2(\sigma_D) & I_x I_y(\sigma_D) \\ I_x I_y(\sigma_D) & I_y^2(\sigma_D) \end{pmatrix}$$

   where σ_D is the scale of the Gaussian derivative. Rule of thumb: $\sigma_I \geq 2\sigma_D$. The Gaussian already performs weighted sum $\sum_{x,y} w(x,y)$ of the general second moment matrix $M = \sum_{x,y} w(x,y) M_{I_x, I_y}$, where $w(x,y)$ is a window function.
   a) Image derivatives: I_x, I_y
   b) Square of derivatives: $I_x^2, I_y^2, I_x I_y$
   c) Gaussian filter: $g \star I_x^2, g \star I_y^2, g \star I_x I_y$
2. *Corner Response Function* - two strong eigenvalues:

$$\begin{aligned} R &= \det(M) - \alpha \text{trace}(M)^2 \\ &= (g \star I_x^2)(g \star I_y^2) - (g \star (I_x I_y))^2 - \alpha(g \star I_x^2 + g \star I_y^2)^2 \end{aligned}$$

3. Perform *non-maximum suppression*.

Hessian detector

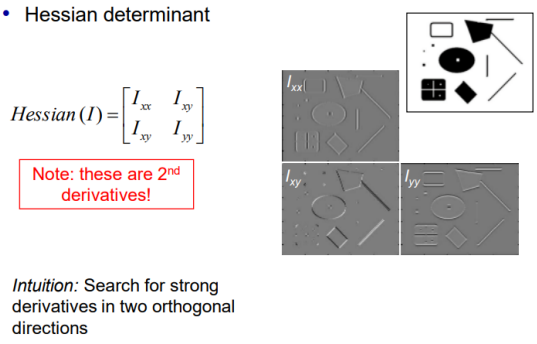
Corner response is obtained with

$$\det(Hessian(I)) = I_{xx} I_{yy} - I_{xy}^2$$

which responses mainly on corners and strongly textured areas.

Properties:

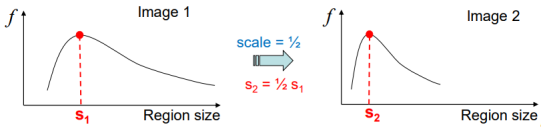
- rotation invariant
- not invariant to image scale



4.1.3 Scale Invariant Region Selection

Automatic Scale Selection

Given are two images of the same scene with a large scale difference between them. Find the same interest points *independently* in each image ($\rightarrow$ *characteristic scale*).



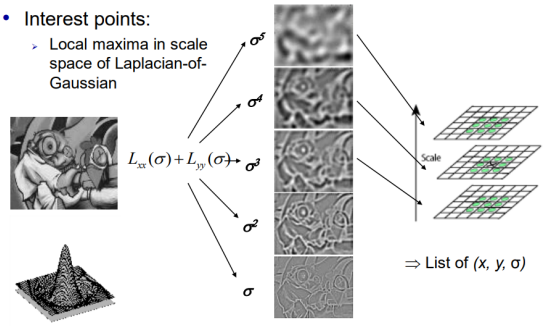
For this, search for maxima of suitable functions *(signature function)* in *scale* and *space*,

which are "scale invariant". Rescale region to fixed size.

Signature Function: Laplacian-of-Gaussian Detector (LoG)

Laplacian-of-Gaussian = "blob" detector.

We define the characteristic scale as the scale that produces peak of Laplacian response.

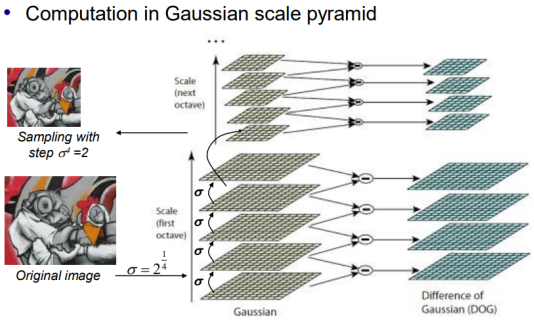


Signature Function: Difference-of-Gaussian Detector (DoG)

We can efficiently approximate the Laplacian with a *Difference of Gaussians*:

$$L = \sigma^2(G_{xx}(x,y,\sigma) + G_{yy}(x,y,\sigma))$$
$$\Rightarrow DoG = G(x,y,k\sigma) - G(x,y,\sigma)$$

The advantages of using this approximations are that we do not need to compute 2nd derivatives and Gaussian are computed anyways *(Gaussian pyramid)*.



Signature Function: Harris-Laplace

1. Initialization: Multiscale Harris corner detection.
2. Scale selection based on Laplacian.
   (same procedure with Hessian → Hessian-Laplace)

4.1.4 Local Descriptors

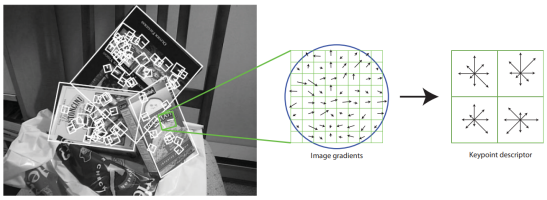
How to *describe* points for matching?

The simplest descriptor would be a list of intensities within a *patch* (write regions as vectors). But small shifts can affect matching score a lot.

SIFT (Scale Invariant Feature Transform)

Descriptor Computation:

- Divide patch into 4x4 sub-patches: 16 cells.
- Compute histogram of gradient orientations (8 reference angles) for all pixels inside each sub-patch.
- Resulting descriptor: 4x4x8 = 128 dimensions.



SIFT aims to achieve robustness to lighting variations and small positional shifts. For rotation invariant descriptors, find *local* orientation (dominant direction of gradient for the image patch). Rotate patch according to this angle, this puts the patches into a *canonical* orientation.

Computation:

1. Compute orientation histogram.
2. Select dominant orientation.
3. Normalize: Rotate to fixed orientation.

Extraordinarily robust matching technique:

+ can handle changes in viewpoint up to 60° out-of-plane rotation
+ can handle significant changes in illumination
+ fast and efficient, can run in real time
- *SURF* as fast approximation of SIFT idea.

4.2 Recognition with Local Features

Image content is transformed into local features that are invariant to translation, rotation, and scale.

Goal: Verify if they belong to a consistent configuration.

- *Warping:* Given a source image and the transformation, what does the transformed output look like?
- *Alignment:* Given two images with corresponding features, what is the transformation between them?

4.2.1 Finding Consistent Configurations

With *homogeneous coordinates* we can express any 2D transformation:

$$\underbrace{\begin{pmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{pmatrix}}_{\text{translation}}, \quad \underbrace{\begin{pmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{pmatrix}}_{\text{scaling}}, \quad \underbrace{\begin{pmatrix} 1 & sh_x & 0 \\ sh_y & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}}_{\text{shearing}}$$

3D rotation around coordinate axes:

$$R_x(\alpha) = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos\alpha & -\sin\alpha \\ 0 & \sin\alpha & \cos\alpha \end{pmatrix}, \quad R_y(\beta) = \begin{pmatrix} \cos\beta & 0 & \sin\beta \\ 0 & 1 & 0 \\ -\sin\beta & 0 & \cos\beta \end{pmatrix}, \quad R_z(\gamma) = \begin{pmatrix} \cos\gamma & -\sin\gamma & 0 \\ \sin\gamma & \cos\gamma & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

Affine transformations are combinations of linear transformations and translations.

4.2.2 Affine Estimation

In alignment, we will fit the parameters of some transformation according to a set of matching feature pairs *("correspondences")*. An affine model approximates perspective projection of planar objects. Assuming we know the correspondences, how do we get the transformation?

Least Squares Estimation

With *Least Squares Estimation* we get for $\mathbf{x}' = M\mathbf{x} + t$

$$\begin{pmatrix} & \cdots & & & & \\ x_i & y_i & 0 & 0 & 1 & 0 \\ 0 & 0 & x_i & y_i & 0 & 1 \\ & \cdots & & & & \end{pmatrix} \begin{pmatrix} m_1 \\ m_2 \\ m_3 \\ m_4 \\ t_1 \\ t_2 \end{pmatrix} = \begin{pmatrix} \cdots \\ x_i' \\ y_i' \\ \cdots \end{pmatrix}$$

Given set of data points (X_i, X_i'), find linear function to predict X's from Xs with $Xa + b = X'$:

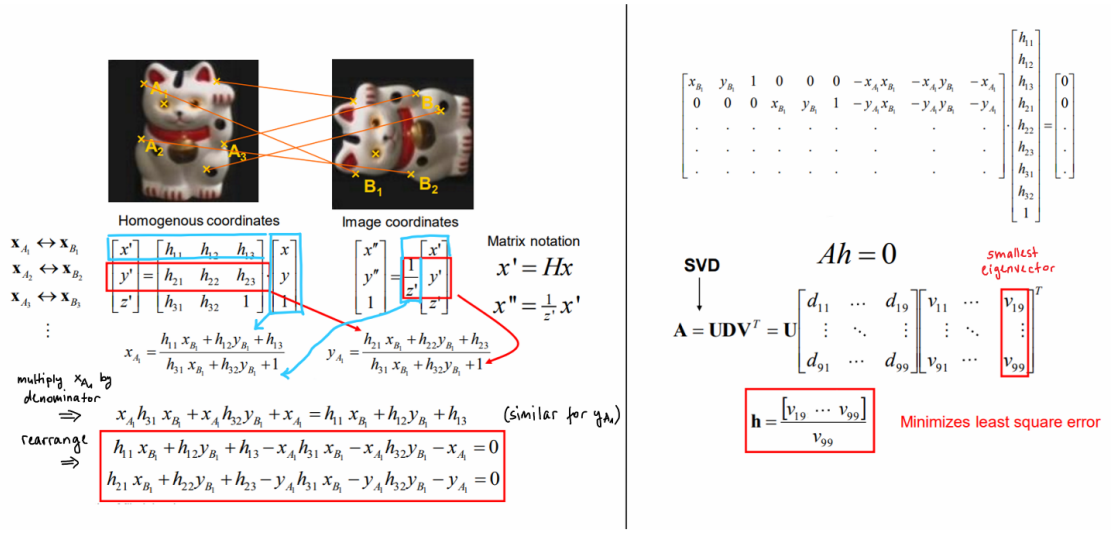
$$\begin{pmatrix} X_1 & 1 \\ X_2 & 1 \\ \cdots & \cdots \end{pmatrix} \begin{pmatrix} a \\ b \end{pmatrix} = \begin{pmatrix} X_1' \\ X_2' \\ \cdots \end{pmatrix} \iff Ax = B$$

Overconstrained problem:
$\min\|Ax - B\|^2 \rightarrow$ *Least-squares minimization*

4.2.3 Homography Estimation

A *projective transform* is a mapping between any two perspective projections with the same center of projection. In a projective transform, a rectangle should map to arbitrary *quadrilateral*. Parallel lines aren't preserved, but need to be straight.

The simplest way to estimate a homography H from feature correspondences is the *Direct Linear Transformation* (DLT) method:



Solution: Null-space vector of A, corresponds to smallest eigenvector.

If v_{99} may be zero, normalize the vector length instead:

$$\mathbf{h} = \frac{[v_{19}, ..., v_{99}]}{||[v_{19}, ..., v_{99}]||}$$

Basic matching algorithm:

1. Detect interest points in two images.
2. Extract patches and compute a descriptor for each one.
3. Compare one feature from image 1 to every feature in image 2 and select the nearest-neighbor pair.
4. Repeat the above for each feature from image 1.
5. Use the lost of best pairs to estimate the transformation between images.

5 Deep Learning

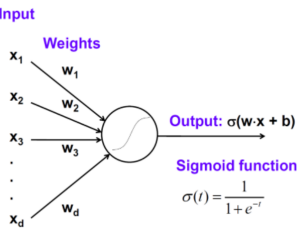
5.1 Neural Networks

Training: Find network weights $\mathbf{w}$ to minimize the error between true training labels t_n and estimated labels $f_{\mathbf{w}}(\mathbf{x}_n)$:

$$E(\mathbf{W}) = \sum_n L(t_n, f(\mathbf{x}_n; \mathbf{W}))$$

Minimization can be done via *gradient descent*, provided f is differentiable. Gradient is determined through *error backpropagation*.

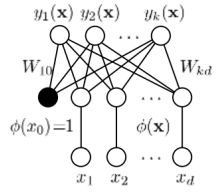
Perceptron:



5.1.1 Background: Deep Learning

Generalized Linear Discriminants:

- Linear classifiers with fixed feature transformation

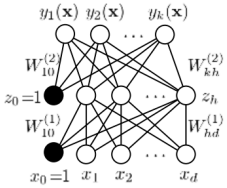


Output layer
Weights
Feature layer
Mapping (fixed)
Input layer

- Outputs
  - Linear outputs $y_k(\mathbf{x}) = \sum_{i=0}^{d} W_{ki}\phi(x_i)$
  - Logistic outputs $y_k(\mathbf{x}) = \sigma\left(\sum_{i=0}^{d} W_{ki}\phi(x_i)\right)$

Multi-Layer Perceptrons (MLP):

- Also learning the feature transformation



Output layer
Hidden layer
Mapping (learned!)
Input layer

- Output

$$y_k(\mathbf{x}) = g^{(2)}\left(\sum_{i=0}^{h} W_{ki}^{(2)} g^{(1)}\left(\sum_{j=0}^{d} W_{ij}^{(1)} x_j\right)\right)$$

$y_k(x) > 0$, if input belongs to target class k. $\phi(x_i)$ can be seen as features. The black node introduces an "offset", the so called *bias* term.

An *MLP* with 1 hidden layer can implement *any* function *(universal approximator)*. However, if the function is deep, a very large hidden layer may be required.

If we leave out the non-linearity $g(\cdot)$, the layers collapse into a single linear function. Therefore, the *non-linearities* make multi-layer representation more powerful.

Nonlinearities

$$\underbrace{\sigma(a) = \frac{1}{1+\exp(-a)}}_{\text{sigmoid}}, \quad \underbrace{tanh(a) = 2\sigma(2a) - 1}_{\text{hyperbolic tangent}}, \quad \underbrace{g(a) = \max(0, a)}_{\text{rectified linear unit (ReLU)}}$$

Gradient Descent

Use *gradient descent* to efficiently adapting *all* weights, not just the last layer.

Set up an *error function*

$$E(\mathbf{W}) = \sum_n L(t_n, y(\mathbf{x}_n; \mathbf{W})) + \lambda\Omega(\mathbf{W})$$

with loss L, regularizer Ω (for ambiguity), $\mathbf{x}_n$ data point and t_n target value. Update each weight $W_{ij}^{(k)}$ in the direction of the (negative) gradient $\frac{\partial E(\mathbf{W})}{\partial W_{ij}^{(k)}}$, which points to the larges *reduction* in the error function.

We can now also apply other loss functions

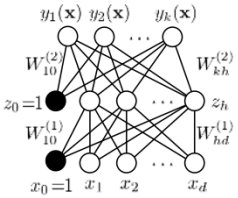
- L2 loss ⇒ Least-squares regression
  $L(t, y(\mathbf{x})) = \sum_n (y(\mathbf{x}_n) - t_n)^2$
- L1 loss: ⇒ Median regression
  $L(t, y(\mathbf{x})) = \sum_n |y(\mathbf{x}_n) - t_n|$
- Cross-entropy loss ⇒ Logistic regression
  $L(t, y(\mathbf{x})) = -\sum_n \{t_n \ln y_n + (1 - t_n)\ln(1 - y_n)\}$
- Hinge loss ⇒ SVM classification
  $L(t, y(\mathbf{x})) = \sum_n [1 - t_n y(\mathbf{x}_n)]_+$
- Softmax loss ⇒ Multi-class probabilistic classification
  $L(t, y(\mathbf{x})) = -\sum_n \sum_k \left\{\mathbb{I}(t_n = k) \ln \frac{\exp(y_k(\mathbf{x}))}{\sum_j \exp(y_j(\mathbf{x}))}\right\}$

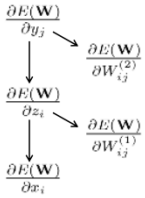
19

Supervised Learning

1. Computing the gradients for each weight.

   Idea: Compute the gradient *layer by layer*. Each layer below builds upon the results of the layer above.
   $\Rightarrow$ *Backpropagation* algorithm





2. Adjusting the weights in the direction of the gradient.
   (Stochastic) Gradient descent: Adjust the weight for one time step further by changing the previous weight by a weighted gradient with respect to that weight evaluated at the current location of the weights:

$$w_{kj}^{(\tau+1)} = w_{kj}^{(\tau)} - \eta \frac{\partial E(\mathbf{w})}{\partial w_{kj}}\bigg|_{\mathbf{w}^{(\tau)}}$$

   with rate/step size η and times index τ. This is applied on *minibatches* of data.

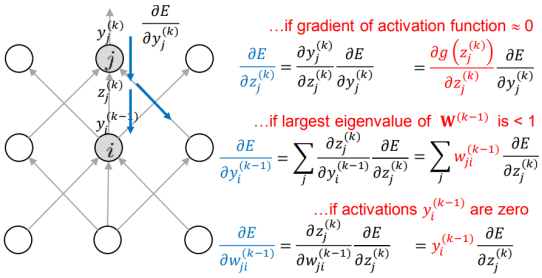
Varnishing gradients problem

In multi-layer nets, gradients need to be propagated through many layers. The *magnitudes of the gradients* are often very different for the different layers, especially if the initial weights are small. Gradients can therefore get very small in the early layers of deep nets.

When designing deep networks, we need to make sure gradients can be propagated throughout the network:

- restricting the network depth
- very careful implementation
- choosing suitable nonlinearities
- performing proper initialization

Backprop steps and where the gradients can varnish:



Weights Initialization

Best practice is to use a *zero-mean distribution* for sampling the initial weights, e.g. a Gaussian or uniform distribution. Compute the variance according to Glorot or He, and plug it into your chosen distribution.

Glorot:

Variance of neuron activations

- Suppose we have an input X with n components and a linear neuron with random weights W that spits out a number Y.
- We *want the variance of the input and output of a unit to be the same*, therefore $n\ \mathrm{Var}(W_i)$ should be 1. This means

$$\mathrm{Var}(W_i) = \frac{1}{n} = \frac{1}{n_{\text{in}}}$$

- Or for the backpropagated gradient

$$\mathrm{Var}(W_i) = \frac{1}{n_{\text{out}}}$$

- As a compromise, Glorot & Bengio propose to use

$$\mathrm{Var}(W) = \frac{2}{n_{\text{in}} + n_{\text{out}}}$$

$\Rightarrow$ Randomly sample the weights from a uniform or Gaussian distribution with this variance.

Designed for *tanh* nonlinearity.

n_{in}: number of *incoming* connection for forward pass
n_{out}: number of *outgoing* connections for backwards pass

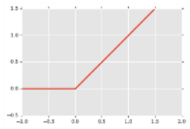
He:

- Extension of Glorot Initialization to ReLU units
  - Use Rectified Linear Units (ReLU)

$$g(a) = \max\{0, a\}$$

  - Effect: gradient is propagated with a constant factor

$$\frac{\partial g(a)}{\partial a} = \begin{cases} 1, & a > 0 \\ 0, & \text{else} \end{cases}$$



- Same basic idea: Output should have the input variance
  - However, the Glorot derivation was based on tanh units, linearity assumption around zero does not hold for ReLU.
  - He et al. made the derivations, proposed to use instead

$$\mathrm{Var}(W) = \frac{2}{n_{\text{in}}}$$

Designed for *ReLU* nonlinearity.

5.2 Convolutional Neural Networks (CNNs)

Convolutional Layers

Feed-forward feature extraction, with classification layer at the end:

1. Convolve input with *learned* filters
2. Non-linearity
3. Spatial pooling
4. (Normalization)

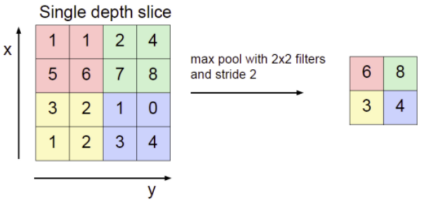
The convolutional filters are learned through *supervised* training and back-propagating the classification error.

But why do we want to use CNNs?

- To avoid huge amounts of parameters, use *convolutions* with *learned kernels*, since the neurons of one layer share the same parameters (the kernels) across different locations.
- All neural net activations are arranged in 3 dimensions. Multiple neurons are all looking at the same input region, stacked in *depth*. (Naming convention: depth $\rightarrow$, width $\nearrow$, height $\uparrow$)
- Convolutional layers can be *stacked*. The filters of the next layer then operate on the full activation volume. Filters are local in (x, y), but densely connected in depth.
- Each activation map is a depth slice through the output volume.

Pooling Layers

Pooling filter responses at different spatial locations, we gain robustness to the exact spatial location of features *(robustness to translations)*. It makes the representation *smaller* without losing too much information.



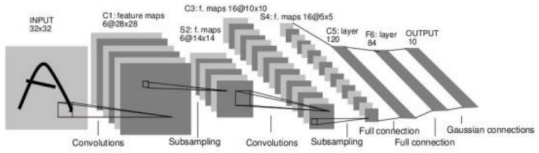
Pooling happens independently across each slice, preserving the number of slices.

5.3 CNN Architectures

5.3.1 LeNet

- 2x conv-conv-pool blocks for feature representation
- FC layers for classification on the end

⇒ successfully used for handwritten digit recognition

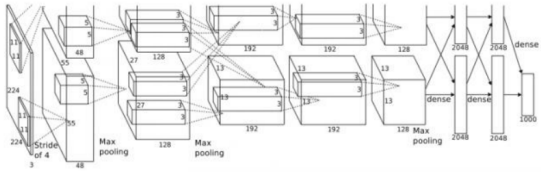


- for other tasks, the two layers of feature computation were not sufficient; also computationally expensive

5.3.2 AlexNet

Similar to LeNet, but

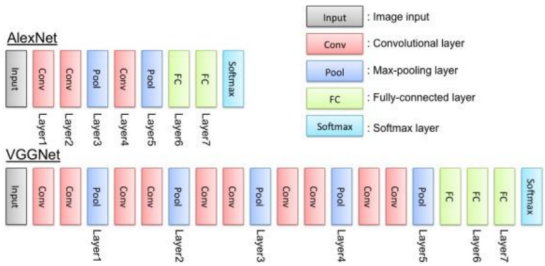
- bigger model (7 hidden layers, 650k units, 60M parameters), more training data (10^6 images instead of 10^3)
- receptive field in the first layer: 11×11, stride 4



⇒ AlexNet almost *halved* the error rate from previous approaches

5.3.3 VGGNet

- deeper network, more convolutional layers with *smaller* filters (+ nonlinearity), with similar amount of FC layers and softmax layer on the end
- receptive field in the first layer: 3×3, stride 1, stacked ones resulting into an efficient 5×5 receptive field, twice 7×7 etc.

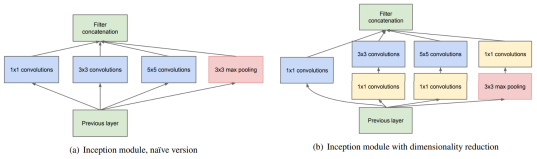


- 138M parameters, but most of them in the FC layers

$\Rightarrow$ same receptive field as AlexNet, but much fewer parameters: $3 \cdot 3^2 = 27$ vs. $7^2 = 49$

5.3.4 GoogLeNet

- *Main ideas:* "inception" module as modular component, learns filters at several scales within each module, 1×1 convs ("bootleneck layers") for dimensionality reduction



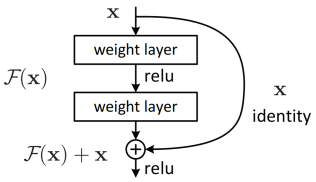
(a) Inception module, naïve version (b) Inception module with dimensionality reduction

- several inception modules in net, with auxiliary classification outputs for training the lower layers (deprecated), 22 layers, no FC layers, only 5M parameters
- VGGNet and GoogLeNet perform at similar level

5.3.5 ResNet

Core component:

- skip connections bypassing each layer
- better propagation of gradients to the deeper layers



5.3.6 Transfer Learning with CNNs

Transfer learning is a machine learning method where a model developed for a task is reused as the starting point for a model on a second task.

1. Train net on ImageNet.
2. If small dataset: Fix all weights (treat CNN as fixed feature extractor), retrain only the classifier.
3. If you have medium sized dataset, *"finetune"* instead: Use the old weights as initialization, train the full network or only some of the higher layers with a smaller learning rate.

5.4 Practical Advise on CNN Training

5.4.1 Data Augmentation

Augment (cropping, zooming, flipping, color PCA) original data with synthetic variations to reduce *overfitting*. Results into much larger training set and robustness against expected variations.

Overfitting: Happens when the model fits too well to the training set. The model starts to recognize specific images in the training set instead of general patterns.

5.4.2 Initialization

When initializing the weights:

- Draw them randomly from a *zero-mean distribution*.
- Common choices in practice: *Gaussian* or *uniform*.
- Common trick: Add a small positive bias ($+\varepsilon$) to avoid units with ReLU nonlinearities getting stuck-at-zero.

When sampling weights from an uniform distribution $[a, b]$:

- Standard deviation is computed as $\sigma^2 = \frac{1}{12}(b-a)^2$.
- Glorot initialization with uniform distribution

$$W \sim U[-\frac{\sqrt{6}}{\sqrt{n_{in}+n_{out}}}, \frac{\sqrt{6}}{\sqrt{n_{in}+n_{out}}}]$$

5.4.3 Batch Normalization

Optimization works best of all inputs of a layer are *normalized*.

Introduce *intermediate layer* that centers the activations of the previous layer per minibatch, resulting in a variance of 1 or mean of 0. I.e., perform transformations on all activations and undo those transformations when backpropagating gradients.

Centering and normalization also needs to be done at test time, but minibatches are no longer available at that point. Therefore, learn the normalization parameters to compensate for the expected bias of the previous layer (usually a simple moving average).

This batch normalization results into much improved convergence.

5.4.4 Dropout

Reduce reliance on individual units by randomly switching off units during training. This changes the net for each data point, effectively training many different variants of the network.

To compensate much larger output responses during testing time, multiply activations with the probability that the unit was set to zero, which reduces the magnitude of the activations.

This results into improved performance.

5.4.5 Learning Rate Schedules

Final improvement step after convergence is reached:

1. Reduce learning rate η by a factor of 10.
2. Continue training for a few epochs.
3. Do this 1-3 times, then stop training.

Turning down the learning rate will reduce the random fluctuations in the error due to different gradients on different minibatches. When turning down the learning rate too soon, further progress will be much slower/impossible after that.

5.5 CNNs for Object Detection

Region Proposal Based Detectors

Avoid dense sliding window with region proposal.

- *R-CNN:* Selective Search + CNN classification / regression
- *Fast R-CNN:* Swap order of convolution and region extraction
- *Faster R-CNN:* Compute region proposals within the network
- *Mask R-CNN:* Detection + instance segmentation + pose estimation

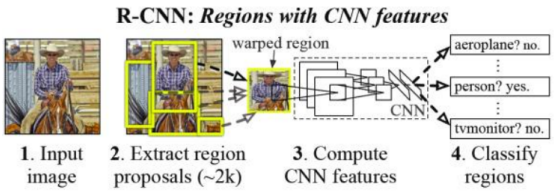
Anchor Box Based Detectors

Perform detection in a single step using grid of anchors boxes.

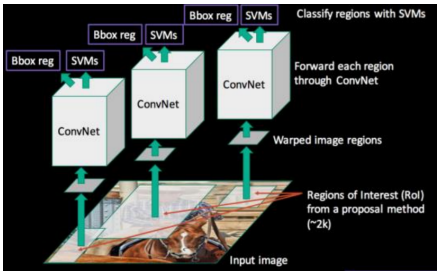
- *YOLO, SSD*

5.5.1 R-CNN

Cut of feature extractor, replace with trained CNN.



Pipeline:



1. Determine RoI of input image from proposal method.
2. Warp image regions.
3. Forward each region through ConvNet.
4. Evaluate ConvNet classification with SVMs.

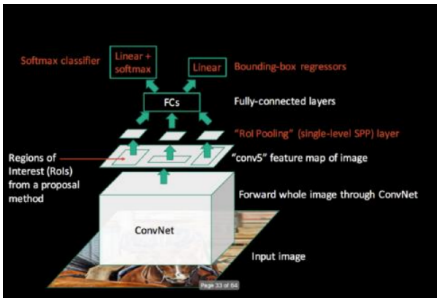
Problems:

- ad hoc training objectives
  - fine-tuned net with softmax classifier *(log loss)*
  - trained post-hoc linear SVMs *(hinge loss)*
  - trained post-hoc bounding-box regressors *(squared loss)*
- training (2 days) and testing (47s/image) are slow
- takes a lot of disk space, need to store all precomputed CNN features for training the classifier

5.5.2 Fast R-CNN

Instead of using ConvNets for every region proposal, apply a ConvNet once on the entire image, and use *RoI pooling*.

Pipeline:



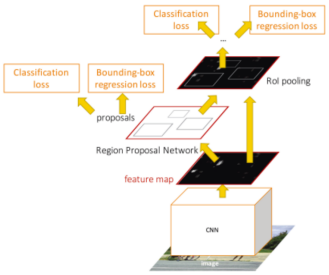
1. Forward whole image through ConvNet.
2. Extract RoIs from a proposal method.
3. "RoI Pooling" layer, resulting into warp.
4. Feed pools into FCs.
5. Linear + softmax classifier, bounding-box regressors.
6. Feed linear + softmax and linear into multi-task loss (log loss + smooth L1 loss)

5.5.3 Faster R-CNN

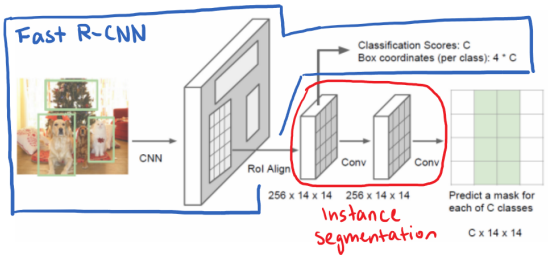
Remove dependence on external region proposal algorithm. Instead, infer region proposals from same CNNs. Results into *feature sharing* and makes object detection *in a single pass* possible.

Faster R-CNN = Fast R-CNN + RPN *(Region Proposal Network)*

One network, *four* losses (joint training):



5.5.4 Mask R-CNN



For detection + instance segmentation, detection + pose estimation.

5.5.5 YOLO/SSD

Directly go from image to detection scores. Results into a very light weight backbone. Subdivide image into a grid. Within each grid cell

1. Start from a set of anchor boxes.
2. Regress from each of the B anchor boxes to a final box.
3. Predict scores for each of C classes (including background).

5.6 CNNs for Segmentation

For *semantic segmentation* label each pixel in the image with a category label. For *instance segmentation* also give an instance label per pixel.

5.6.1 Fully Convolutional Networks (FCN)

Design a network as a sequence of convolutional layers, to make predictions (for segmentation) for all pixels at once.

In *FCNs*, all operations formulated as convolutions. Fully-connected layers become 1x1 convolutions. The advantage of using convolutions is that FCNs can process *arbitrarily sized* images.

Think of FCNs as performing a sliding-window classification. The computation is more efficient, since computations are reused between windows. On the other hand, convolutions at original image resolution will be very expensive.

5.6.2 Encoder-Decoder Architecture

Design net as a sequence of convolutional layers, with *downsampling* and *upsampling* inside the network.

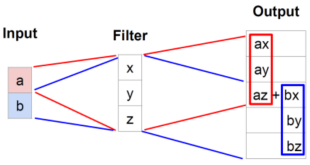
- **Downsamling:** pooling, strided (stride > 1) convolution.
- **Upsampling:**
  - *Nearest-Neighbor:* Spread value of pixel to a patch of one size bigger, results into a blocky output structure.
  - *"Bed of Nails":* Keep one pixel value as the original and pad the other ones in a patch with zero.
  - *Max Unpooling*: Remember which element was maximum after max-pooling, and use this position for "Bed of Nails" upsampling.
  - *Strided Transpose Convolution*

5.6.3 Transpose Convolutions

For a *transpose convolution* with stride 2 and pad 1: Learned filter moves 2 pixels in the output for every one pixel in the input. (stride gives ratio between movement in output and input)

Output contains copies of the filter weighted by the input, summing overlaps in the output. We need to crop one pixel from output to make output exactly 2x input.

1D example:



Express convolution in terms of matrix multiplication (1D, kernel size = 3, stride = 1, pad = 1):

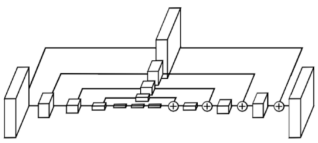
$$x \star a = Xa = \begin{pmatrix} x & y & z & 0 & 0 & 0 \\ 0 & x & y & z & 0 & 0 \\ 0 & 0 & x & y & z & 0 \\ 0 & 0 & 0 & x & y & z \end{pmatrix} \begin{pmatrix} 0 \\ a \\ b \\ c \\ 0 \end{pmatrix} = \begin{pmatrix} ay + bz \\ ax + by + cz \\ bx + cy + dz \\ cx + dy \end{pmatrix}$$

Convolution transpose multiplies by the transpose of the same matrix: $\boxed{x \star^T a = X^T a}$.
When stide = 1, convolution transpose is just a regular convolution (with different padding rules).
When stride > 1, convolution transpose is no longer a normal convolution.

5.6.4 Skip Connections

Since downsampling loses high-resolution information, use *skip connections* to preserve higher-resolution information.

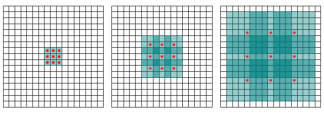
Introduce skip connections when performing pooling, connected just before the corresponding unpooling stage. This transfers some of the input before it was pooled.



5.6.5 Extensions

Dilated Convolutions (Atrous Convolutions)

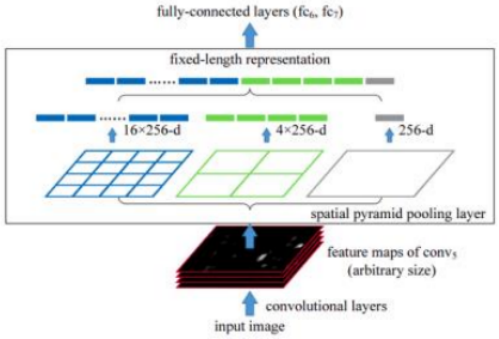
Sample the input at every r-th pixel for the convolution. Increase receptive field without increasing computation. With dilation factor l:



$$y[i] = \sum_{k=1}^{K} x[i + r \cdot k]w[k], \quad (F \star_l k)(\mathbf{p}) = \sum_{\mathbf{s}+l\mathbf{t}=\mathbf{p}} F(\mathbf{s})k(\mathbf{t})$$

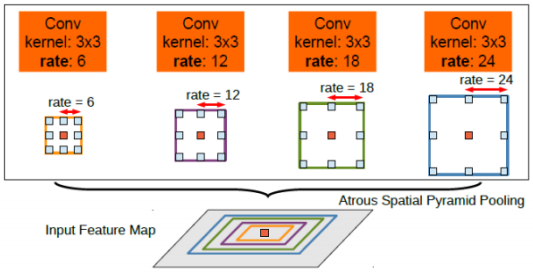
Spatial Pyramid Pooling (SPP)

Multiple convolutional pooling layers at different scales. Improves robustness to varying image scales, and creates a multi-scale feature representation for classification.



Atrous Spatial Pyramid Pooling (ASPP)

Extends SPP concept by using *dilated convolutions* instead of regular ones. Increases receptive field without extra computation.



5.6.6 Examples

- **SegNet**
  - encoder-decoder architecture with skip connections
  - encoder based on VGG-16, decoder is using max unpooling
  - output with K-class softmax classification
- **U-Net**
  - similar idea: encoder-decoder architecture with skip connections
- **DeepLabv3+**
  - uses *atrous spatial pyramid pooling* (ASPP)
  - simple decoder module
  - depth-wise separable convolutions for efficiency

5.7 CNNs for Human Body Pose Estimation

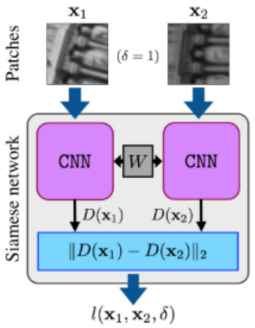
Setup:

1. Annotate images with keypoints for skeleton joints.
2. Define a target disk around each keypoint with radius r.
3. Set the ground-truth label to 1 within each such disk.
4. Infer heatmaps for the joints as in semantic segmentation.

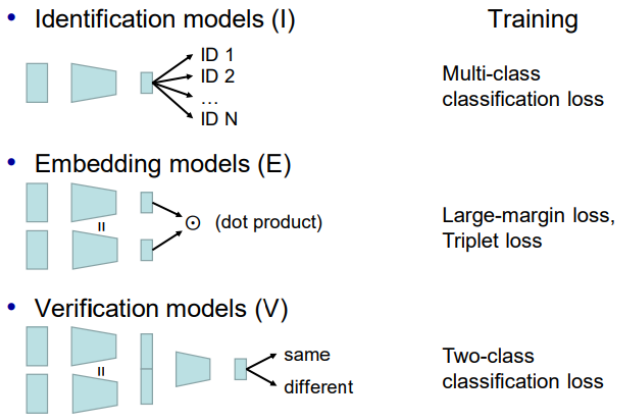
5.8 CNNs for Matching

5.8.1 Siamese Networks

Present the two stimuli to two *identical copies* of a network (with shared parameters). Train them to output similar values iff the inputs are (semantically) similar.



Types of models used for matching tasks:



5.8.2 Triplet loss

To train a network to achieve this, we learn a *discriminative embedding*: Present the network with triples of examples *(negative, anchor, positive)*. Apply *triplet loss* to learn an embedding f that groups the positive example closer to the anchor than the negative one:

$$\|f(x_i^a) - f(x_i^p)\|^2 < \|f(x_i^a) - f(x_i^n)\|^2$$

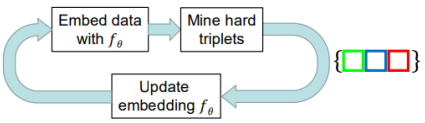
Triplet loss formulation with distance $D_{a,p} = d(x_a, x_p)$ between anchor and positive example and safety margin m as hyperparameter:

$$\mathcal{L}_{tri}(\theta) = \sum_{\substack{a,p,n,\\ y_a=y_p\neq y_n}} [m + D_{a,p} - D_{a,n}]_+ = \sum_{\substack{a,p,n,\\ y_a=y_p\neq y_n}} \max(0, m + D_{a,p} - D_{a,n})$$

Offline Hard Triplet Mining

Mining hard triplets becomes crucial for learning, where the triplet is prone for confusion, because the anchor and negative example are close to each other. A popular solution for that is *offline hard triplet mining*:

1. Process the dataset to find hard triplets, in form of mini-batches.
2. Use those for learning.
3. Iterate.



Considerable effort needed, but a very wasteful design, because a minibatch of the mined triplets contains potentially much more hard triplets than we actually mined.

Online Hard Triplet Mining

Use *online hard triplet mining* instead:

- Each member of another triplet becomes an additional negative candidate. But, we need both hard negatives and hard positives.
- An even better design is to sample K images from P classes for each minibatch. For one anchor, the previously positive and negative example become positive, every other sample in the minibatch (whether previous anchor, positive, or negative) becomes a negative example. The triplets are then only constructed within the minibatch.

Applications

- *Person re-identification*: tracking and surveillance, multi-camera handover.
- *Face identification*
- *Stereo matching*: siamese net for feature extraction, learns an embedding optimized for matching using correlation score, correspondence search in the correlation volume.

5.9 Recurrent Networks

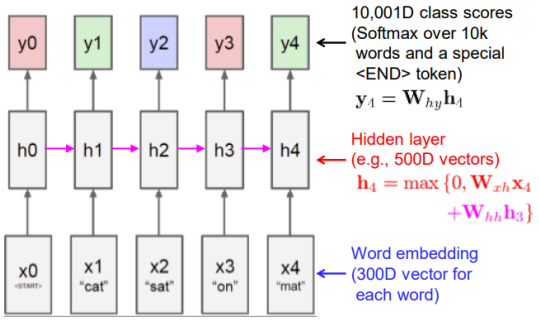
RNNs are regular NNs whose hidden units have additional connections over time. You can *unroll* them to create a net that extends over time. When you do this, keep in mind that the weights for the hidden units are shared between temporal layers.

The net is very powerful, because they combine two properties:

- Distributed hidden state that allows them to store a lot of information about the past efficiently.
- Non-linear dynamics that allows them to update their hidden state in complicated ways.

On the other hand, the training is more challenging, since unrolled nets are deep.

We want to train a language model with $p(\textit{next word} \mid \textit{previous words})$, where p is high for this setting. Pink annotates the recurrent connections between hidden layers.



After training the RNN, we get a language model, which predicts the next word by sampling the output (posterior) of the previous one. From each output y_i, sample next input word x_{i+1}, without end sequence.

Applications

- *Image tagging:* Simple combination of CNN and RNN. Use CNN to define initial state h_0 of an RNN. Use RNN to produce text description for the image. Trained on corpus of images with textual descriptions
- *Video to text description*

6 3D Reconstruction

To reconstruct a 3D structure, we need *multi-view geometry* because structure from one image is inherently ambiguous.

Given several image of the same object or scene, compute a representation of its 3D shape. In particular, given a *calibrated binocular stereo pair*, fuse it to produce a *depth image*.

6.1 Epipolar Geometry and Stereo Basics

The *epipolar geometry* is the *intrinsic* projective geometry between two views. It is independent of scene structure, and only depends on the cameras' internal parameters and relative pose.

Principle: Triangulation, which gives the reconstruction as intersection of two rays. Requires *camera calibration* and *point correspondences*.

Parameters for camera calibration:

- *Extrinsic:* rotation matrix and translation vector (camera frame $\leftrightarrow$ reference frame)
- *Intrinsic:* focal length, pixel sizes, image center point, radial distortion parameters (image coords. relative to camera $\leftrightarrow$ pixel coords.)

Parallel Optical Axes

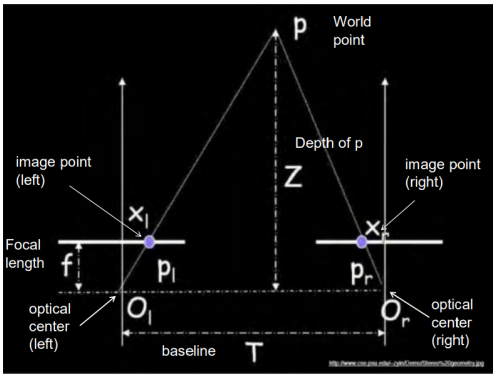
Assume, these parameters are given and fixed.

Task of *depth estimation*: Estimate disparity map $D(x, y)$ from a set of images. For similar triangles (p_l, P, p_r) and (O_l, P, O_r) we get depth associated with point p

$$\frac{T - (x_r - x_l)}{Z - f} = \frac{T}{Z} \quad \Longleftrightarrow \quad Z = f\frac{T}{x_r - x_l}$$

where $x_r - x_l$ *disparity*, the horizontal motion. Then, we can get a point on the other image through the disparity map $D(x, y) := f\frac{T}{Z}$:

$$(x', y') = (x + D(x, y), y)$$



Stereo Correspondences Constraints

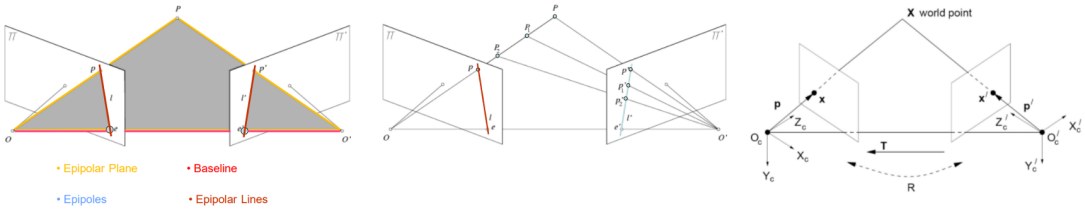
Generally, two cameras do not need to have parallel optical axes.

Geometry of two views allows us to constrain where the corresponding pixel for some image point in the first view must occur in the second view. Here, the *epipolar constraint* is useful, because it reduces correspondences problem to *1D search* along conjugate epipolar lines.

The *epipolar geometry* between two views is essentially the geometry of the intersection of the image planes with the *epipolar planes* having the *baseline* as (fixed) axis.

- *Baseline:* line joining the camera centers.
- *Epipole e*: point of intersection of baseline with the image plane. All epipolar lines intersect at the epipole.
- *Epipolar line l*: intersection of epipolar plane with image plane.
- *Epipolar plane* Π: plane containing baseline and world point. An epipolar plane intersects the left and right image planes in epipolar lines.

Potential matches for p have to lie on the corresponding epipolar line l.



Some useful stuff for computation:

- normalize homogeneous point: $(x_1, x_2, x_3)^T \rightarrow (\frac{x_1}{x_3}, \frac{x_2}{x_3}, 1)^T$
- normalize homogeneous line: $(a, b, c)^T \rightarrow (\frac{a}{\sqrt{a^2+b^2}}, \frac{b}{\sqrt{a^2+b^2}}, \frac{c}{\sqrt{a^2+b^2}})^T$
- perpendicular distance from a point to a line: $d = l \cdot p$ (dot product)

6.1.1 Calibrated Case: Essential matrix

To rotate and translate camera reference frame 1 to get camera reference frame 2, we use the 3D vector $X' = RX + T$.

$$\begin{aligned} X' &= RX + T \\ \underbrace{T \times X'}_{\text{plane normal}} &= T \times RX + T \times T = T \times RX \\ X' \cdot (T \times X') &= X' \cdot (T \times RX) = 0 \end{aligned}$$

Let $E = T_\times R$ be the *essential matrix*, which relates corresponding image points:

$$\begin{aligned} X' \cdot (T \times RX) = 0 = X' \cdot T_\times RX \\ X'^T EX = 0 \end{aligned}$$

Matrix form of cross product:

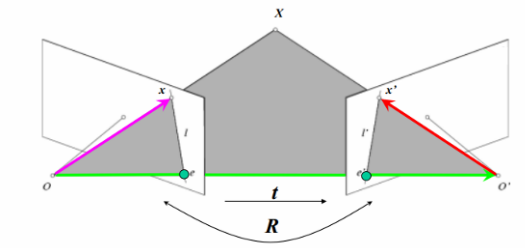
$$a \times b = [a_\times]b = \begin{pmatrix} 0 & -a_z & a_y \\ a_z & 0 & -a_x \\ -a_y & a_x & 0 \end{pmatrix} b$$

This holds for the rays p and p' that are parallel to the camera-centered position vectors X and X', so we have the *epipolar constrain equation*

For epipolar lines from point p and p' respectively we get

$$l' = Ep, \quad l = E^T p'$$

$$\boxed{p'^T Ep = 0}$$



Camera matrix: $[\mathbf{I}|\mathbf{0}]$
$X = (u, v, w, 1)^T$
$x = (u, v, w)^T$

Camera matrix: $[\boldsymbol{R}^T | -\boldsymbol{R}^T\boldsymbol{t}]$
Vector x' in second coord. system has coordinates $\boldsymbol{Rx}'$ in the first one.

The vectors x, t, and Rx' are coplanar

Essential Matrix (Longuet-Higgins, 1981)

Epipolar Constraint

$x \cdot [t \times (Rx')] = 0 \Rightarrow x^T E x' = 0$ with $E = [t_\times]R$

- $E\,x'$ is the epipolar line associated with x' ($l = E\,x'$)
- $E^T x$ is the epipolar line associated with x ($l' = E^T x$)
- $E\,e' = 0$ and $E^T e = 0$:
- E is singular (rank two) : one 0 solution for epipole e
- E has five degrees of freedom (up to scale)

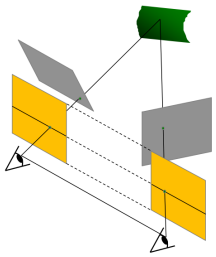
6.2 Stereopsis and 3D Reconstruction

Main steps:

1. Calibrate cameras.
2. Rectify images.
3. Compute disparity.
4. Estimate depth.

6.3 Stereo Image Rectification

When the cameras' optical axes are not parallel, we want to pre-warp the images, such that the corresponding epipolar lines are coincident.



Algorithm:

1. Reproject image planes onto a common plane parallel to the line between optical centers. Pixel motion is horizontal after this transformation.
2. Compute two homographies (3×3 transforms), one for each input image reprojection.

6.4 Disparity

Correspondence problem: Multiple match hypotheses satisfy epipolar constraint, but which is correct?

6.4.1 Dense Correspondence Search

Rectify images first. For each pixel in the first image, find corresponding epipolar line in the right image. Examine all pixels on the epipolar line and pick the best match according to some matching score (e.g. SSD, correlation).

+ simple process
+ more depth estimates, can be useful for surface reconstruction

- breaks down in *textureless* regions
- raw pixel distances can be brittle
- not good with very different viewpoints

6.4.2 Sparse Correspondence Search

Restrict search to sparse set of detected *features*. Rather than pixel values (or lists of pixel values) use *feature descriptor* and an associated *feature distance*. Still narrow search further by epipolar geometry.

We want window large enough to have sufficient intensity variation, yet small enough to contain only pixels with about the same disparity.

+ efficiency
+ can have more reliable feature matches, less sensitive to illumination than raw pixels

- have to know enough to pick good features
- sparse information

Possible Sources of Error

- low-contrast/textureless image regions
- occlusions
- camera calibration errors
- violations of brightness constancy (e.g. specular reflections)
- large motions

6.5 Camera Calibration

Recap: To solve $Ax = 0$, apply SVD

$$A = UDV^T = U \underbrace{\begin{pmatrix} d_{11} & & \\ & \ddots & \\ & & d_{NN} \end{pmatrix}}_{\text{singular values}} \underbrace{\begin{pmatrix} v_{11} & \cdots & v_{1N} \\ \vdots & \ddots & \vdots \\ v_{N1} & \cdots & v_{NN} \end{pmatrix}^T}_{\text{singular vectors}}$$

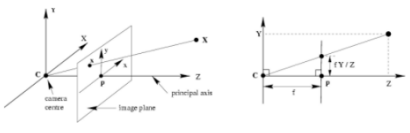
- *singular values* of A = square roots of the eigenvalues of $A^T A$
- Solution of $Ax = 0$ is the *nullspace* vectors of A.
- This corresponds to the *smallest singular vector* of A.

6.5.1 Camera Models/Parameters

In the following, we will look at *camera models*, which are matrices with particular properties that represent the camera mapping.

Pinhole Model

- *Principal axis:* Line from camera center perpendicular to image plane
- *Normalized (camera) coordinate system:* Camera center is at origin and principal axis is z-axis
- *Principal point p:* Point where principal axis intersects the image plane (origin of normalized coordinate system)



$$(X,Y,Z) \mapsto (fX/Z, fY/Z)$$

$$\begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix} \mapsto \begin{pmatrix} fX \\ fY \\ Z \end{pmatrix} = \begin{bmatrix} f & & & 0 \\ & f & & 0 \\ & & 1 & 0 \end{bmatrix} \begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix} = \begin{bmatrix} f & & \\ & f & \\ & & 1 \end{bmatrix} \begin{bmatrix} 1 & & & 0 \\ & 1 & & 0 \\ & & 1 & 0 \end{bmatrix} \begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix}$$

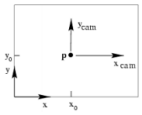
$$x = PX \qquad P = \mathrm{diag}(f, f, 1)[I \mid 0]$$

The origin of camera coordinate system is *principal point*, the origin of the image

coordinate system is in the corner.

Pinhole Model with Origin not in Principle Point

But it may not be, so we define a more general mapping with the *calibration matrix* K:



$$(X,Y,Z) \mapsto (fX/Z + p_x, fY/Z + p_y)$$

$$\begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix} \mapsto \begin{pmatrix} fX + Zp_x \\ fY + Zp_x \\ Z \end{pmatrix} = \begin{bmatrix} f & & p_x & 0 \\ & f & p_y & 0 \\ & & 1 & 0 \end{bmatrix} \begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix} = \begin{bmatrix} f & & p_x \\ & f & p_y \\ & & 1 \end{bmatrix} \begin{bmatrix} 1 & & & 0 \\ & 1 & & 0 \\ & & 1 & 0 \end{bmatrix} \begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix}$$

$$K = \begin{bmatrix} f & & p_x \\ & f & p_y \\ & & 1 \end{bmatrix} \quad \text{Calibration matrix} \quad \boxed{P = K[I \mid 0]}$$

Not Explicit Camera Center

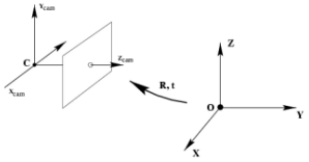
For pixel size $1/m_x \times 1/m_y$, where m_x, m_y pixels per meter in horizontal and vertical direction respectively, we get calibration matrix K:

$$K = \begin{pmatrix} m_x & & \\ & m_y & \\ & & 1 \end{pmatrix} \begin{pmatrix} f & & p_x \\ & f & p_y \\ & & 1 \end{pmatrix} = \begin{pmatrix} \alpha_x & & x_0 \\ & \alpha_y & y_0 \\ & & 1 \end{pmatrix}$$

Camera Rotation and Translation

In general, the camera coordinate frame will be related to the world coordinate frame by a *rotation* and a *translation*.

$\tilde{X}_{cam}$: coordinates of point in camera frame
$\tilde{X}$: coordinates of a point in world frame *(non-homogeneous)*
X: coordinates of a point in world frame *(homogeneous)*
$\tilde{C}$: coordinates of camera center in world frame



In non-homogeneous coordinates:

$$\tilde{X}_{cam} = R(\tilde{X} - \tilde{C})$$

$$X_{cam} = \begin{bmatrix} R & -R\tilde{C} \\ 0 & 1 \end{bmatrix} \begin{pmatrix} \tilde{X} \\ 1 \end{pmatrix} = \begin{bmatrix} R & -R\tilde{C} \\ 0 & 1 \end{bmatrix} X$$

$$x = K[I \mid 0] X_{cam} = K[R \mid -R\tilde{C}] X \qquad \boxed{P = K[R \mid t],} \qquad t = -R\tilde{C}$$

Note: $\mathbf{C}$ is the null space of the camera projection matrix ($\mathbf{PC}$=0)

Summary

(DoF)

- *Intrinsic parameters*
  - principal point coordinates p_x, p_y *(2)*
  - focal length f *(1)*
  - pixel magnification factors m_x, m_y *(1)*
  - skew (non-rectangular pixels) S *(1)*
    (e.g. "picture of picture")
  - radial distortion (preprocessing step)
- *Extrinsic parameters*
  - rotation R *(3)*
  - translation t *(3)*

$$K = \begin{pmatrix} m_x & & \\ & m_y & \\ & & 1 \end{pmatrix} \begin{pmatrix} f & S & p_x \\ & f & p_y \\ & & 1 \end{pmatrix} = \begin{pmatrix} \alpha_x & S & x_0 \\ & \alpha_y & y_0 \\ & & 1 \end{pmatrix}$$

(both relative to world coordinate system)

- *Camera projection matrix* $P = K[R|t]$
  $\Rightarrow$ general pinhole camera: *9 DoF*
  $\Rightarrow$ CCD camera with non-square pixels: *10 DoF*
  $\Rightarrow$ general camera: *11 DoF*

6.5.2 Calibration Procedure

Compute *intrinsic* and *extrinsic* parameters using observed camera data.

Given n points with known 3D coordinates X_i and known image projections x_i, estimate camera parameters, such that $x_i = PX_i$.

Main idea:

1. Place calibration object with known geometry in the scene.
2. Get correspondences.
3. Solve for mapping from scene to image: Estimate $P = P_{int}P_{ext}$.

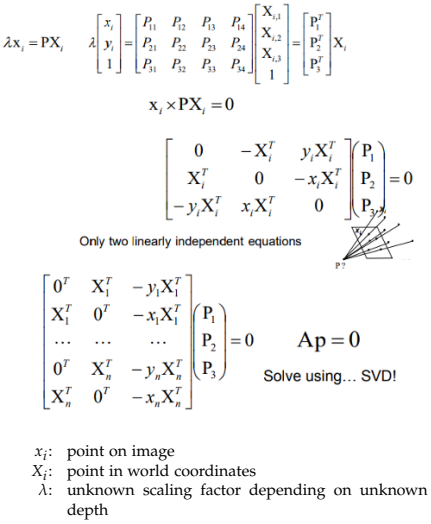
For best results, the calibration points need to be measured with *subpixel accuracy* (depending in exact pattern).

Algorithm for checkerboard pattern:

1. Perform *Canny edge detection*.
2. Fit straight lines to detected linked edges.
3. Intersect lines to obtain corners. If sufficient care is taken, the points can then be obtained with localization accuracy $< 1/10$ pixel

Rule of thumb: Number of constraints should exceed number of unknowns by a factor of 5, thus, at least 28 points are necessary.

DLT Algorithm (Direct Linear Transform)



x_i: point on image
X_i: point in world coordinates
λ: unknown scaling factor depending on unknown depth

Idea: Get rid of unknown scaling factor λ.

Notes:

- 5 1/2 correspondences needed for a minimal solution
- For coplanar points that satisfy $\Pi^T X = 0$ for a plane Π, we will get degenerate solutions $(\Pi, 0, 0)$, $(0, \Pi, 0)$, or $(0, 0, \Pi)$. We need calibration points in more than one plane!

To figure out intrinsic and extrinsic parameters after recovering the numerical from of the camera matrix, use *matrix decomposition*.

6.6 Uncalibrated Reconstruction

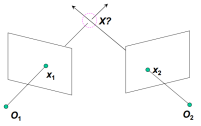
3 main question for two-view geometry:

- *Scene geometry (structure)*: Where is the pre-image of the points in 3D? (→ triangulation)
- *Correspondence (stereo matching)*: How does a point in one image constrain the position of the corresponding point x' in another image? (→ epipolar constraint)
- *Camera geometry (motion)*: What are the cameras for the two views? (→ SfM)

The *fundamental matrix* and the *essential matrix* are the algebraic representations of epipolar geometry, that satisfies some *epipolar constraint* for any pair of corresponding points $x \leftrightarrow x'$ in the two images.

6.6.1 Triangulation

Given projection of a 3D point in several images, find coordinates of the point. Because of errors, the intersection of the two visual rays will never meet exactly.



1) Geometric Approach

Find shortest segment connecting the two viewing rays and let X be the midpoint of that segment.

2) Linear Algebraic Approach

Two independent equations each in terms of three unknown entries of X. Stack them and solve using SVD. This approach nicely generalizes to multiple cameras, unlike geometric approach.

$$\begin{array}{lll} \lambda_1 x_1 = P_1 X & x_1 \times P_1 X = 0 & [x_{1\times}]P_1 X = 0 \\ \lambda_2 x_2 = P_2 X & x_2 \times P_2 X = 0 & [x_{2\times}]P_2 X = 0 \end{array}$$

3) Nonlinear Approach

Find X that minimizes

$$d^2(x_1, P_1 X) + d^2(x_2, P_2 X)$$

Most accurate, but unlike the other two methods, it does not have a closed-from solution (e.g. can't be expressed with a finite number of standard operations).

6.6.2 Uncalibrated Case: Fundamental Matrix

Using $\hat{x}$ and $\hat{x}'$ in matrix space, we apply a normalized coordinate system to get pixel coordinates x and x':

- The calibration matrices K and K' of the two cameras are unknown
- We can write the epipolar constraint in terms of *unknown* normalized coordinates:

$$\hat{x}^T E \hat{x}' = 0 \qquad x = K\hat{x}, \quad x' = K'\hat{x}'$$

$$\hat{x}^T E \hat{x}' = 0 \Rightarrow x^T F x' = 0 \quad \text{with} \quad F = K^{-T} E K'^{-1} \Rightarrow$$

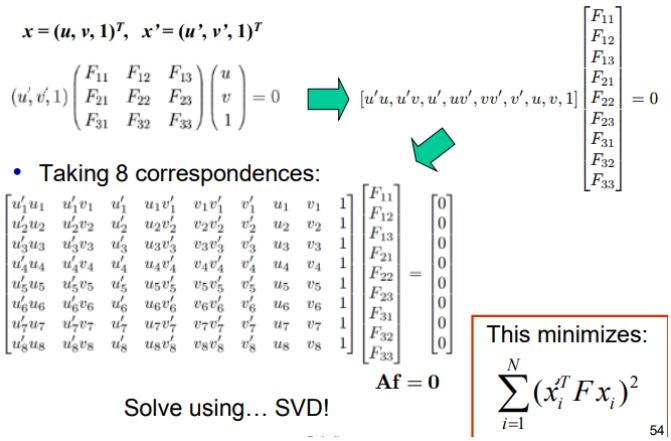
Fundamental Matrix (Faugeras and Luong, 1992)

- $F x'$ is the epipolar line associated with x' ($l = F x'$)
- $F^T x$ is the epipolar line associated with x ($l' = F^T x$)
- $F e' = 0$ and $F^T e = 0$
- F is singular (rank two)
- F has seven degrees of freedom

Geometrically, F represents a mapping from the 2D projective plane of the first image to the pencil of epipolar lines through the epipole e. Thus, it represents a mapping from a 2D onto a 1D projective space, and hence must have rank 2.

To estimate the fundamental matrix F use *eight-point algorithm*.

Eight-Point Algorithm



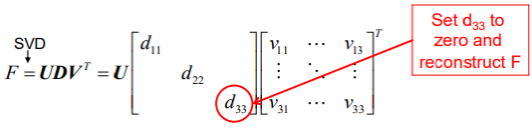
8 points are sufficient, because F has a rank of 2.

Eight-point algorithm has poor numerical conditioning, which can be fixed by rescaling the data.

The solution will usually not fulfill the constraint that F only has rank 2, if we are working on *noisy data*. This means, that there will be no epipoles through which all epipolar lines pass.

Normalized Eight-Point Algorithm

1. Center the image data at the origin, and scale it so the mean squared distance between the origin and the data points is 2 pixels.
2. Use the eight-point algorithm to compute F from the normalized points.
3. Enforce the rank-2 constraint using SVD.



4. Transform fundamental matrix back to original units: If T and T' are the normalizing transformations in the two images, then the fundamental matrix in original coordinates is $T^T F T'$.

Nonlinear Least-Squares (Refinement Approach)

For refining solution, minimize

$$\sum_{i=1}^{N}(x_i^T F x_i')^2 \quad \Rightarrow \quad \boxed{\sum_{i=1}^{N}(d^2(x_i, Fx_i') + d^2(x_i', Fx_i))}$$

6.6.3 Stereo Pipeline with Weak Calibration

We want to estimate world geometry *without* requiring calibrated cameras.

Main Idea: Estimate epipolar geometry from a (redundant) set of point correspondences between two *uncalibrated* cameras.

Procedure to find F *and* correspondences:

1. Find interest points in both images. (e.g., Harris corners)
2. Compute correspondences using only proximity.
3. Compute epipolar geometry.
4. Refine.

RANSAC for robust estimation of F:

1. Select random sample of correspondences.
2. Compute F using them. This determines epipolar constraint.
3. Evaluate amount of support – number of inliers within threshold distance of epipolar line.
4. Iterate until a solution with sufficient support has been found (or max #iterations).
5. Choose F with most support.

6.6.4 Extension: Epipolar Transfer

Given x_1 and x_2 points on the two images with known epipolar geometries F_{13}, F_{23}, point x_3 in third image is the intersection of l_{31} and l_{32}:

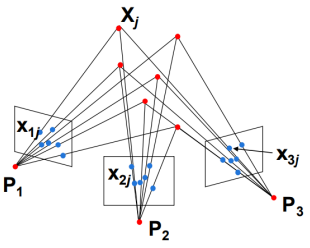
$$l_{31} = F_{13}^T x_1, \quad l_{32} = F_{23}^T x_2$$

Can't be applied, if the motion is strictly in the same plane.

6.7 Structure-from-Motion (SfM)

Given: m images of n fixed 3D points $x_{ij} = P_i X_j$, $i = 1, ..., m$, $j = 1, ..., n$.

Problem: Estimate m projection matrices P_i and n 3D points X_j from the mn correspondeces x_{ij}.



Structure from Motion Ambiguity

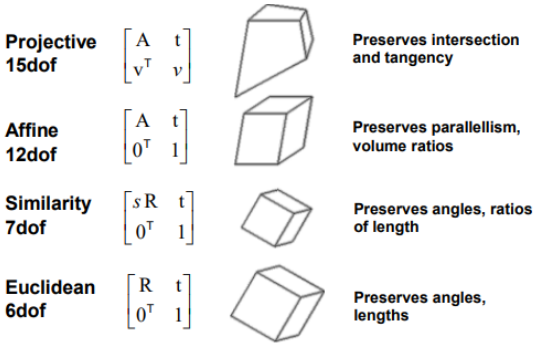
If we transform the scene using a transformation Q (similarity, affine, projective) and apply the inverse transformation to the camera matrices, then the images don't change:

$$x = PX = (PQ^{-1})QX$$

Idea: With no constraints on camera calibration we get a projective reconstruction. Add information to *upgrade* the reconstruction to affine, similarity, or Eulidean.

Hierarchy of 3D Transformations

From most unconstraint on top to the most constraint on the bottom:



Projective Structure from Motion

Given are m images of n fixed 3D points.

$$z_{ij}x_{ij} = P_iX_j,\ i = 1,...,m,\ j = 1,...,n$$

We want to estimate m projection matrices P_i and n 3D points X_j from the mn correspondences x_{ij}. With no calibration info, cameras and points can only be recovered up to a 4×4 projective transformation Q:

$$X \to QX, P \to PQ^{-1}$$

Two-camera case with depths z and z':

- Assume fundamental matrix $\mathbf{F}$ between the two views
  - First camera matrix: $[I|0]Q^{-1}$
  - Second camera matrix: $[A|b]Q^{-1}$
- Let $\tilde{\mathbf{X}} = \mathbf{QX}$, then $z\boldsymbol{x} = [\boldsymbol{I}\,|\,\boldsymbol{0}]\tilde{\boldsymbol{X}}, \quad z'\boldsymbol{x}' = [\boldsymbol{A}|\boldsymbol{b}]\tilde{\boldsymbol{X}}$
- And
$$z'\boldsymbol{x}' = \boldsymbol{A}[\boldsymbol{I}\,|\,\boldsymbol{0}]\tilde{\boldsymbol{X}} + \boldsymbol{b} = z\boldsymbol{A}\boldsymbol{x} + \boldsymbol{b}$$
$$z'\boldsymbol{x}' \times \boldsymbol{b} = z\boldsymbol{A}\boldsymbol{x} \times \boldsymbol{b}$$
$$(z'\boldsymbol{x}' \times \boldsymbol{b}) \cdot \boldsymbol{x}' = (z\boldsymbol{A}\boldsymbol{x} \times \boldsymbol{b}) \cdot \boldsymbol{x}'$$
$$0 = (z\boldsymbol{A}\boldsymbol{x} \times \boldsymbol{b}) \cdot \boldsymbol{x}'$$
- So we have $\boxed{\mathbf{x}'^{\mathrm{T}}[\mathbf{b}_\times]\mathbf{A}\mathbf{x} = 0}$

$\mathbf{F} = [\mathbf{b}_\times]\mathbf{A}$ **b:** epipole $(\mathbf{F}^{\mathrm{T}}\mathbf{b} = \mathbf{0})$, $\mathbf{A} = -[\mathbf{b}_\times]\mathbf{F}$

We can solve for structure and motion when $2mn \geq 11m + 3n - 15$. For two cameras at least 7 points are needed.

Decomposing the fundamental matrix in the two-camera case means that if we can compute the fundamental matrix between two cameras, we can directly estimate the two projection matrices from F. Once we have the projection matrices, we can compute the 3D position of any point X by triangulation.

To obtain both kinds of information at the same time, use *projective factorization*.

Projective Factorization

If we knew the depths z, we could factorize D to estimate M and S. If we knew M and S, we could solve for z.

<u>*Solution:*</u> Iterative estimation with *Sequential Structure from Motion* by alternating the steps above.

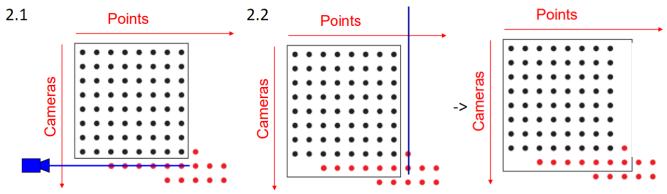
$$\mathbf{D} = \begin{bmatrix} z_{11}\mathbf{x}_{11} & z_{12}\mathbf{x}_{12} & \cdots & z_{1n}\mathbf{x}_{1n} \\ z_{21}\mathbf{x}_{21} & z_{22}\mathbf{x}_{22} & \cdots & z_{2n}\mathbf{x}_{2n} \\ & & \ddots & \\ z_{m1}\mathbf{x}_{m1} & z_{m2}\mathbf{x}_{m2} & \cdots & z_{mn}\mathbf{x}_{mn} \end{bmatrix} = \begin{bmatrix} \mathbf{P}_1 \\ \mathbf{P}_2 \\ \vdots \\ \mathbf{P}_m \end{bmatrix} \begin{bmatrix} \mathbf{X}_1 & \mathbf{X}_2 & \cdots & \mathbf{X}_n \end{bmatrix}$$

Cameras ($3m \times 4$) — Points ($4 \times n$)

$\mathbf{D} = \mathbf{MS}$ has rank 4

Sequential Structure from Motion

1. Initialize motion from two images using fundamental matrix. Initialize structure.
2. For each additional view:
   - Determine projection matrix of new camera using all the known 3D points that are visible in its image - *calibration*
   - Refine and extend structure: Compute new 3D points, re-optimize existing points that are also seen by this camera - *triangulation*
3. Refine structure and motion - *bundle adjustment*



Bundle Adjustment

Non-linear method for refining structure and motion. Minimize mean-square reprojection error

$$E(\mathbf{P}, \mathbf{X}) = \sum_{i=1}^{m} \sum_{j=1}^{n} D(\mathbf{x}_{ij}, \mathbf{P}_i \mathbf{X}_j)^2$$

<u>*Idea:*</u> Seek the *Maximum Likelihood* (ML) solution assuming the measurement noise is Gaussian. It involves adjusting the bundle of rays between each camera center and the set of 3D points.

Considerably improves the results and allows assignment of individual covariances to each measurement. However, it needs a good initialization, and can become an extremely large minimization problem.