Computer Vision

- Goal: enables machine to understand images & videos
  - measurement: compute properties of 3D World (distance, shape...)
  - perception & interpretation: recognize objects, people, activities...

- Course outline:
  1) Image Processing Basic
  2) Segmentation
  3) Local features & Matching
  4) Object Recognition & Categorization
  5) 3D Reconstruction
  6) Deep Learning

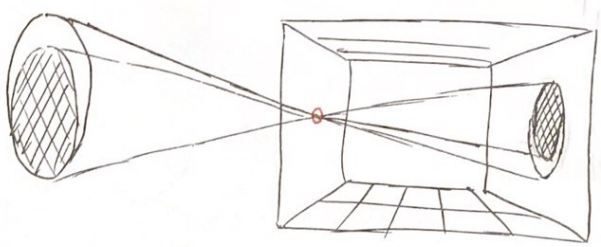
Ng Hieu Duc

CV

Image Formation

Lo1

+) Camera obscura:
"Dark Chamber"
Leonardo Da Vinci 1544

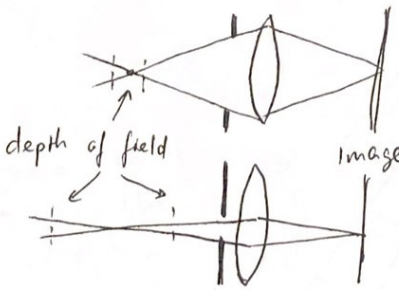


+) Pinhole Camera:

- Pinhole size = Aperture
  - too big - blurring
  - too small - also blur, but because of diffraction
  - but then, image is dark

⇒ Lenses: keep image sharp while capture more light

- The thin lens
- Focus & Depth of Field:



Large Aperture: small depth of field
(only object with the correct distance will be at focus, while background is blur)

Small Aperture: large depth of field
but need more light

- f & field of view: f gets smaller, image ⇒ wide range
  f gets larger, —— telescopic

+) Digital image: Discretize the image into pixels
Quantize light intensities ⇒ pixel values

+) Resolution: No. of pixel (most commonly understand)

+) Color Sensing : Bayer grid – demosaicing

Color image: RGB just 1 of many color space LUV XYZ ...
Grey-scale image

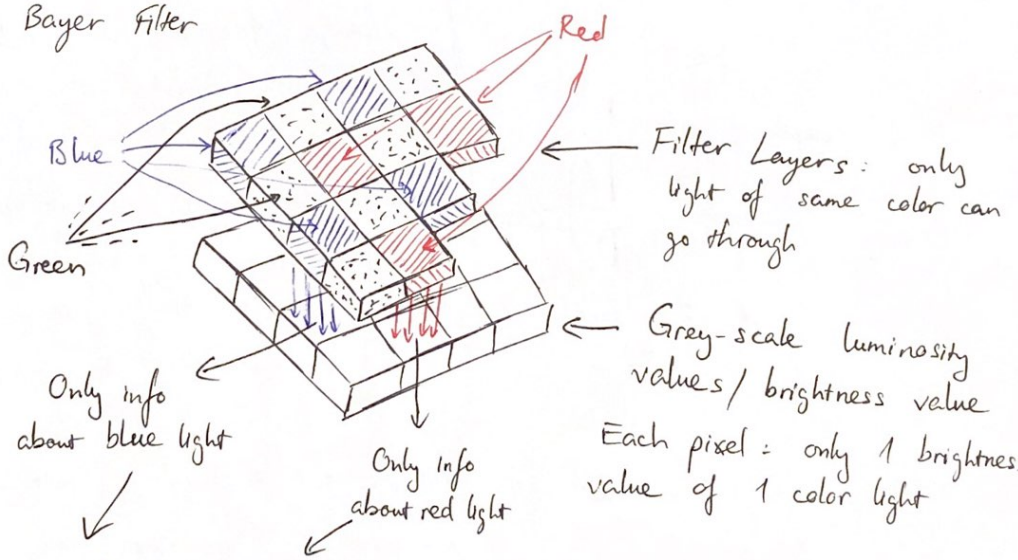
⊛ Demosaicing

Digital camera takes in ~~image~~ light through a filter (Bayer or Xtrans)

⇒ we get a grey-scale image

We need to do Demosaicing based on the filter's pattern to get the color image from the raw image

Example: Bayer Filter



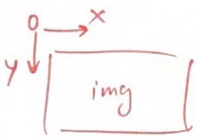
⇒ Use brightness values of surrounding pixels about other 2 colors (a.k.a. demosaicing)

⊛ Raw image has a green-cast

Twice many green as red & blue cause human eyes are twice as sensitive to the green part to other red / blue part.

CV

Image Processing



L02

1. Linear Filters

- Type of noise: { Salt & pepper noise, Impulse noise, Gaussian noise }

noise = randn(size(im)) .* sigma
output = im + noise

Basic Assumption: i.i.d - independent & identically distributed

- Correlation Filter: $G[i,j] = \frac{1}{(2k+1)^2} \sum_{u=-k}^{k} \sum_{v=-k}^{k} F[i+u, j+v]$

Matlab: filter2, imfilter

k is from the window size $(2k+1) \times (2k+1)$

different weights: $G[i,j] = \sum_{u=-k}^{k} \sum_{v=-k}^{k} \underbrace{H[u,v]}_{\text{non-uniform weights}} F[i+u, j+v]$

$G = H \otimes F$

- Convolution: $G[i,j] = \sum_{u=-k}^{k} \sum_{v=-k}^{k} H[u,v] F[i-u, j-v]$

conv2

$G = H * F$

$F_{0,0}$ ... $F_{N,N}$

If $H[u,v] = H[-u,-v] \Rightarrow$ correlation $\equiv$ convolution

Averaging Filter: Ringing Artifacts ?

- Gaussian Filter: $\frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right)$

more noise $\Rightarrow \uparrow \sigma$ $\Rightarrow$ blurring effect

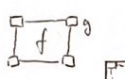
Rule of thumb: set filter width to 6σ

Efficient Implementation: if Filter is separable $\Rightarrow$ do 1D 2 times to have 2D

computational cost: from $O(K^2)$ to $\approx O(2K)$

K: Kernel size

- Boundary issues: { full, same, valid }

output size		
full	$= f + g$	
same	$= f$	
valid	$= f - g$	

Pixel near boundary: { Clip filter (black) $\Rightarrow$ dark border, Wrap around, Copy edge $\Rightarrow$ strong edge response, Reflect across edge }

2. Back-ground

- Taken Fourier Transform of a signal ⇒ Frequency coefficients
  ⇒ Frequency Spectrum

⊗ Duality: The better a func is localized in 1 domain
the worse it is localized in the other

- Effect of Convolution: $f * g \multimap F \cdot G$

A Gaussian has compact support in both domain
⇒ convenient choice for low-pass filter

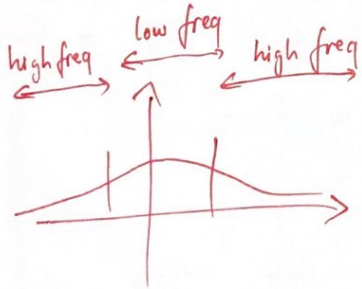
[low pass filter ⇒ smooth
[high pass filter ⇒ edge, line

- Sharpening filter : emphasize noise aswell

3. Non-Linear Filter : replace each pixel by the median of neighbors

- Median Filter: [remove spikes (good for impulse, salt&pepper noise)
[edge preserving (unlike mean filter)

If we increase filter size of Median Filter ⇒ reducing structure
loose details



freq domain
(Fourier)

⊗ Correlation vs Convolution
Conv is better, it has additional nice properties
- commutative : $f * g = g * f$
- associative $(f * g) * h = f * (g * h)$
- Fourier. transform $f * g \multimap F \cdot G$

Both are linear shift invariant (LSI)
$h \circ (f_0 + f_1) = h \circ f_1 + h \circ f_0$

also $f.h \multimap F * H$

L03,

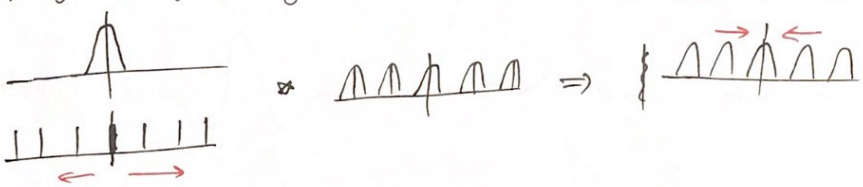
4. Multi - Scale representations

- Image Pyramid: very little over-head (in term of computational cost)

● Fourier Interpretation: Discrete Sampling:

- Sampling in spatial domain is like multiplying with a spike func

⇒ Sampling in frequency domain is like convolving with a spike func



=> When we sampling with lower frequency, the magnitude spectrum will be overlaped (due to duality) ⇒ will not be able to reconstruct signal

+) Nyquist theorem and limit: to recover a certain freq f, at least sample with 2f

⇒ Aliasing artifacts in Graphics: overlapped (because sampling with too low freq)

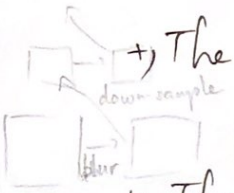
/ey li ət sing/

edges?

⊛ We can not recover high frequencies, but we can avoid artifacts by prior smoothing before resampling

with impulse img
conv reproduces itself
corr reflect itself

+, The Gaussian Pyramid: perform blurring / smooth ⇒ then down-sampling



+, The Laplacian Pyramid: $L_i = G_i - \text{expand}(G_{i+1})$

$G_i = L_i + \text{expand}(G_{i+1})$

$L_n = G_n$

⇒ $L_{0 \to n-1}$ contain high frequency information

Images in Laplacian Pyramid can be compressed further than the corresponding Gaussian Pyramid images

Laplacian ~ Difference of Gaussians

⇒ detect high-freq ≈ edges

the name Laplace ⇒ ?? from a combinations of 2nd derivatives

Laplacian : $\nabla^2 f = \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2}$

$$\frac{\partial^2 f}{\partial x^2} = [f(x+1, y) - f(x, y)] - [f(x, y) - f(x-1, y)]$$

$$= f(x+1, y) + f(x-1, y) - 2f(x, y)$$

$$\Rightarrow \nabla^2 f = f(x \pm 1, y) + f(x, y \pm 1) - 4f(x, y)$$

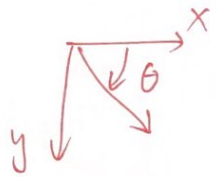
⇒ Laplacian filters:

0	1	0
1	-4	1
0	1	0

5. Filters as templates:

Correlation filtering as Template Matching

6. Image Gradients



- Differentation & Convolution: $\frac{\partial f(x,y)}{\partial x} \approx \frac{f(x+1,y) - f(x,y)}{1}$

$\Rightarrow$ $\boxed{1\ |\ -1}$

Problem: it shifts the image

$\Rightarrow$ Prewitt, Sobel, Roberts Filters

$\begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{bmatrix}$; $\begin{bmatrix} 1 & 1 & 1 \\ 0 & 0 & 0 \\ -1 & -1 & -1 \end{bmatrix}$ ↘ ↙

$M_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}$ $M_y = \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}$

↘ $\begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix}$; $\begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}$

* Gradient direction

$\theta = \tan^{-1}\left(\frac{\partial f}{\partial y} / \frac{\partial f}{\partial x}\right)$

Edge strength

$\|\nabla f\| = \sqrt{\left(\frac{\partial f}{\partial x}\right)^2 + \left(\frac{\partial f}{\partial y}\right)^2}$

- With noise, we need to smooth the image first →

- Derivative Theorem of Convolution:

$\frac{\partial}{\partial x}(h * f) = \left(\frac{\partial h}{\partial x}\right) * f$

Deri. filter vertical ↑ Gaussian filter ↗

$\Rightarrow$ $g * (h * I) = (g * h) * I$

$\Rightarrow$ Use "Derivative of Gaussian" filter

Only need to do 1 filter.. (??)

7, Edge Detection:

L4,

- Criteria for an "optimal" edge detector: robust to noise, good localization, single response

- Primary steps: { smoothing (suppress noise); edge enhancement (filter for contrast); edge localization }

- Effect of σ on derivatives: { large σ ⇒ larger scale edges; small σ ⇒ finer features also }

- Threshold: { high ⇒ less weak details (either relevant or not); low ⇒ more }

⊛ The Canny Edge Detector:

+) Gradient magnitude

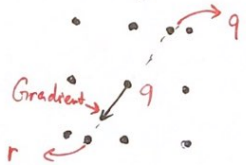
+) Thresholding

+) Non-Maximum Suppression:

Check if pixel is local maximum along gradient direction

(Select single max across width of the edge)

Keep q iff Mag(q) > Mag(p) and Mag(q) > Mag(r)



+) Hysteresis Thresholding: 2 threshold k_{high} & k_{low}

k_{high} start edges, k_{low} continues edge chain (typical: $\frac{k_{high}}{k_{low}} = 2$)

⊛ Edges ≠ Object boundaries

Ex: complex texture, shadow ...

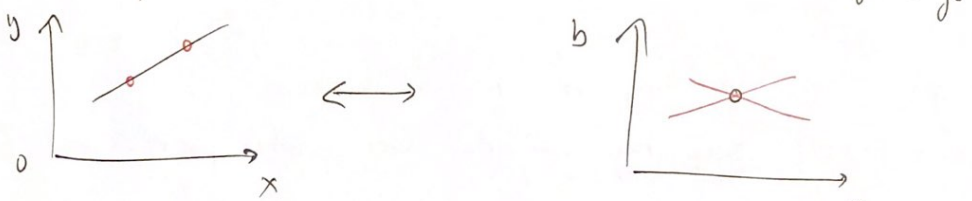
8, Structure Extraction:

- Hough transform: { given points ⇒ what is the line
  - how many lines ?
  - which point belongs to which lines

- Image ⟷ Hough space

$y = m_0 x + b_0$ a line ⟷ a point (m_0, b_0)

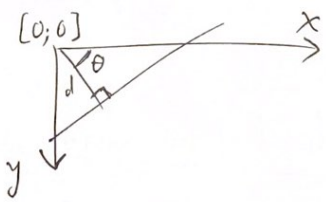
(x_0, y_0) a point ⟷ a line $b = -x_0 m + y_0$



⇒ We can accumulate votes in discrete set of bins

⊛ Problem: vertical lines

⇒ Polar Representation for lines:



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

⇒ Sinusoid segment in Hough space

⊛ Hough Transform Algorithm:

1, Initialize $H[d, \theta] = 0$

2, For each edge point (x, y)

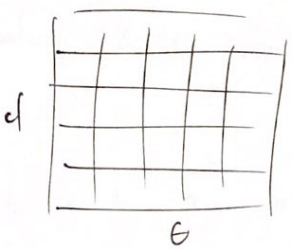
for $\theta = 0 \to 180$ // stride

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

$H[d, \theta] \mathrel{+}= 1$

3, Find values of $(\hat{d}, \hat{\theta})$ where $H[d, \theta]$ is maximal

4, The detected line given by $\hat{d} = x \cos\hat{\theta} + y \sin\hat{\theta}$



In practice, range of

d: $[-D; D]$

θ: $\left[-\frac{\pi}{2}; \frac{\pi}{2}\right]$

Extensions:

1, Use image gradient: $\theta = \tan^{-1}\left(\frac{\partial f}{\partial y} / \frac{\partial f}{\partial x}\right)$ (for lines)

2, Give more votes for stronger edges (gradient magnitude)

3, Change the sampling of $(d, \theta) \Rightarrow$ more or less resolution

less resolution $\Rightarrow$ better to deal with noise, lose some details

more ——— $\Rightarrow$ ..

4, Same procedure for circles, squares, .. (any shape can be parameterized)

circle $\Rightarrow$ 3D Hough space

Tips: +) Smoothing $\Rightarrow$ vote for neighbors

+) Keep tags on the votes, what points vote for which lines..

⇓ Cons: complexity search time ⇑ exponentially with No. model parameters

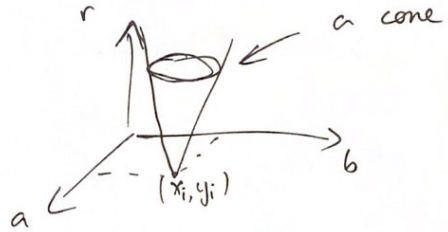
+) Generalized Hough Transform: use define reference point

+) Hough transform for circle:

$$(x_i - a)^2 + (y_i - b)^2 = r^2$$

- Unknown r ⇒ gradient direction

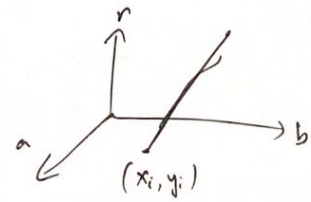
each point to a cone



- Unknown r, know grad_dir

$a = x - r.c\theta$

$b = y + r.s\theta$



Segmentation

L5,

Segmentation as Grouping Problem

- Gestalt Theory:
  The whole is greater than the sum of its parts.
  Relationships among parts can yield new properties / features.
  Gestalt Factors: proximity, similarity, parallelism, closure..
  ⇒ These factors make intuitive sense, but are very difficult to translate to algorithms.

Goals: identify groups of pixels that go together
or find boundaries that separate objects

Segmentation as clustering:

- K-Means Clustering: each pixels as a data point in the feature space NP-hard even with $k = 2$

Problems with K-Means?

K-Means++: as an initialization for centroids c_i

1, Random first center
2, Pick next center with the probability proportional to the distance to previous centers $\|p - c_i\|^2$ ⇒ Kind of prioritize further centers
3, Repeat till k centers

- A variety of feature space choice (intensity, color, texture, position) ⇒ we can group pixels in different ways.

Probabilistic Clustering:

- Expectation - Maximization : compare with K-Means

Model free Clustering : no shape, no number of cluster.

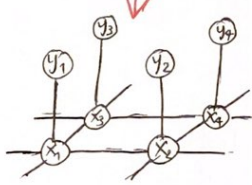
- Mean-Shift Algorithm : only specify window size
  - +) Mode: local maximum of the density of given distribution
  - +) Cluster: all data points in the attraction basin of a mode
  - +) Attraction basin: region for which all trajectories lead to same mode
- Tricks: 1) Assign all points within radius r around final mode
  2) within radius r/c of search path to the mode
- Cons: depends on window size
  computationally expensive, does not scale well with dimension of feature space

Segmentation as Energy Minimization problem

Markov Random Fields

+) Maximize $p(x, y) = \prod_i \phi(x_i, y_i) \cdot \prod_{i,j} \psi(x_i, x_j)$

- $p(x, y)$: x = scene, y = image/pixel
- $\phi(x_i, y_i)$: image-scene compatibility
- $\psi(x_i, x_j)$: scene-scene compatibility



+) Minimize energy formulation:

$$E(x, y) = \sum_i \phi(x_i, y_i) + \sum_{i,j} \psi(x_i, x_j)$$

unary / single-node potentials | pairwise potentials

$$\begin{cases} \phi(x_i, y_i) = -\log \phi(x_i, y_i) \\ \psi(x_i, x_j) = -\log \psi(x_i, x_j) \end{cases}$$

- Many inference algorithms:
  - Gibbs sampling, simulated annealing
  - Iterated conditional modes (ICM)
  - Variational methods
  - Belief propagation
  - Graph cuts

L6, ⊗ Graph cut: ones of many inference algorithm

only for binary energy functions ??

but if we can fit it to our problem, can be implemented very fast

⊗ Unary terms: just normal classifier

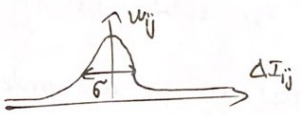
If we have given color model for initial classes: I^s, I^t

$$\Rightarrow \begin{cases} \phi_i(s) \propto \exp\left(-\|I_i - I^s\|^2 / 2\sigma^2\right) \\ \phi_i(t) \propto \exp\left(-\|I_i - I^t\|^2 / 2\sigma^2\right) \end{cases}$$

⊗ Pairwise terms:

Ex: $\psi(x_i, x_j) = w_{ij} \cdot \delta(x_i \neq x_j)$ ⟶ only active when we cut that weight

$$w_{i,j} = \exp\left\{\frac{-\Delta I_{ij}^2}{2\sigma^2}\right\}$$

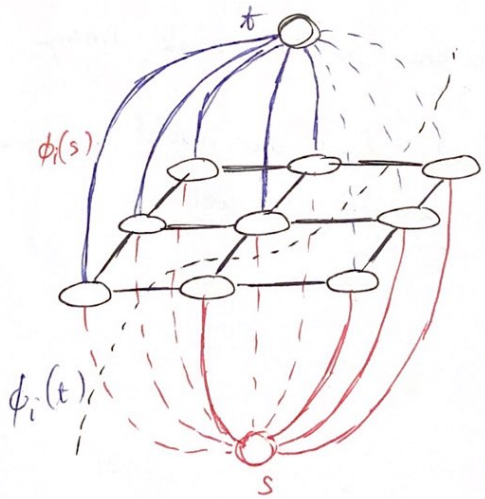


⇓

Cause only when we cut that link between 2 pixels of different classes, that we will loss that weight

⇒ if pixels (in neighboring) is similar ⇒ great weight

not similar ⇒ small weight



⊛ Example:

+) Color potentials: - as unary potentials.

e.g. modeled with MoGs

$$\phi(x_i, y_i; \theta_\phi) = -\log \sum_k \theta_\phi(x_i, k) \cdot p(k \mid x_i)\, \mathcal{N}(y_i; \mu_k, \Sigma_k)$$

+) Edge potentials: - as pairwise potentials.

Contrast sensitive Potts model

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

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

⇒ If there is no label change: $x_i = x_j \Rightarrow$ cost $\varphi = 0$

If there is label change: $x_i \neq x_j$

- If pixel values are indeed different $y_i \neq y_j \Leftrightarrow y_i - y_j \uparrow$
  $\Rightarrow g_{ij}(y) \to 0$ very small
- If not $y_i \approx y_j \Leftrightarrow g_{ij}$ will be great

- β is pre-calculate on the whole image (for each img or all, depends on each situation..)
- θ_ϕ, θ_ψ decide how our model is sensitive to color, or edge.. need to be learned as well.

⊛ s-t - Min cut Problem:

- An st-cut divides the nodes between source & sink
- The cost of the cut: sum of costs of all edges going from S to T (edges that are cut)

⇒ st - min cut is st-cut with minimum cost

↑ Max flow Algorithms:

- Constraints: Edges: Flow < Capacity
  Nodes: Flow in = Flow out
- Algorithms:
  - Find path from source → sink with positive capacity
  - Push max flow through
  - Adjust capacity
  - Repeat till no new path possible

- s-t Graph Cuts can only be applied?

Can only minimize binary energies, that are sub-modular (discrete equivalent to convexity)

- Augmenting Path vs Push-Relabel

all edge weights must ≥ 0 $\forall w_{ij}$

+) α-Expansion Moves:

- Each move add new label & decrease energy
- Choose move that minimize the energy the most

+) α-β swap

Application:

+) Interactive Image Segmentation

+) Iterated Graph Cut

Summary Graph-Cut:

- Applicable for a wide range of problems
- Very efficient algorithm for vision problems
- Become standard

- Can only solve limited class of models
- Only optimal for sub-modular problem
  For multi-label case, only approximate

Object Detection

47,

Classification + Localization = Object Detection < Instance Segmentation

+) Object Recognition Challenges ?

+) Appearance-based recognition: 3D objects can be represented by a set of images appearances, for recognition, it's sufficient to just compare the global representation

+) Identification vs Categorization

Identification	Categorization
same object seen	new object, in only same class
find particular object which we have seen before	

+) Sliding-window based Object Detection — brute force

- Slide window around (1 pixel to the right, down..)
- Over different scale (scale the image, not the window)

+) Detection via Classification: Main Idea

1, Obtain training data (Ex: Car & non-car data)
2, Define features
3, Define classifier

+) Feature Extraction:

- Global Appearance : such as pixel-based appearance; (color or grayscale)

not robust ⇐ sensitive to small shifts, illumination, intra-class variation..

⇒ Gradient-based Representations — invariance to small shifts & rotations

localized histograms offer more spatial info than single global hist.

L8,

HoG (Histogram of Oriented Gradients)

If we want to stick with simple linear classifier (SVM), then our feature representation has to be very good.

Inputs: prepared images with same shape
divide to grid cell (8×8 pixels)

Training phase +) HoG Descriptor Processing Chain:

1) Gamma Compression: $x \mapsto \sqrt{x}$ small performance improvement

2) Compute gradients (both magnitude & direction)

3) Weighted vote in spatial & orientation cells
   From grad_direction ⇒ index of bin
   Vote: NOT '+1', but '+grad_mag'

Trilinear Interpolation is complicate

4) Contrast normalize over overlapping spatial cells
   Each block has 4 cells (2×2)
   Each block overlapped 50% with the next block.
   ⇒ L2 normalization within 4cell, Clipping (0,2), L2 norm again
   Again, just like gamma suppression, the purpose is to reduce the effect of very strong gradients, that could over shadow others.

5) Convert 3D array of histogram into 1D feature vector ⇒ SVM

⊛ The SVM will deal with only 1 type of dimension vector thus, all training inputs, and then the window-shape in testing phase must be the same

⊛ HoG in Testing Phase:

- Is like convolving HoG feature map of test image with the learned template w

- Non-Maximum Suppression: to get single detection
  - Can break ties with noise
  - Also clip the score (> 0,8 ..)
  - Is clustering problem .. could use mean-shift ..

Could loose 1 detection if 2 are too close

- With Sliding-window approach:
  - Move window by 1 cell, not 1 pixel
  - For different scale detection, down/upscale the image, while fixed everything else (cell size ..)

Usually 20-30 rescaled versions
Each with little diff
(??) (20%)

Image Pyramid

Boosting:

N datas
M iterations

Final decisions:

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

$$I(A) = \begin{cases} 1, & \text{if A true} \\ 0, & \text{else} \end{cases}$$

Example: Viola Jones face detector

As example for Adaboost

1) Initialize: $w_n^{(1)} = \frac{1}{N}$ for $n = 1, .., N$

2) For $m = 1, .., M$ iterations

- Train weak classifier $h_m(x)$ to minimize $J_m = \sum_{n=1}^{N} w_n^{(m)} I(h_m(x_n) \neq t_n)$
- Estimate weighted error of h_m on X: $\varepsilon_m = \frac{J_m}{\sum_{n=1}^{N} w_n^{(m)}}$
- Calculate a weighting coefficient for $h_m(x)$: $\alpha_m = \ln\left(\frac{1-\varepsilon_m}{\varepsilon_m}\right)$
- Update weighting coefficients: $w_n^{(m+1)} = w_n^{(m)} \exp\{\alpha_m I(h_m(x_n) \neq t_n)\}$

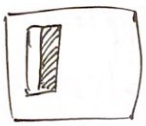
2

⊗ Limitations of Sliding-Window approach:

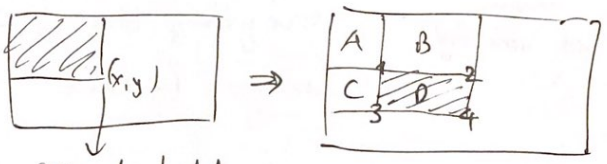
- Not all objects are "box" shape
- Window doesn't consider context. Ex: pedestrian is on the ground only, not in the sky

⇒ That's why we call it brute-force

⊗ Viola-Jones Face Detector:

- "Rectangular" filters:  just rectangular regions with specific position in a patch

- Efficiently computable with Integral image



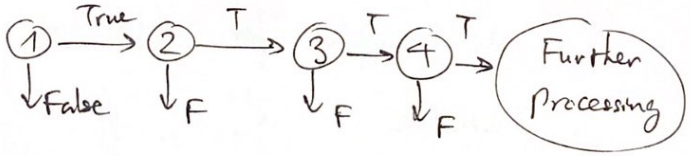
sum of shaded region

D = ① + ④ − ② − ③

- Simple classifier for each filters: output > θ ? just like decision tree

- Cascading Classifiers for Detection:

Use cheap ones with low false negative rates early in the chain



⇒ remove unqualified windows (so we have less computations)

Reject sub-window

⊗ Train with 5k positives, 350M negatives
Test with 38 layer cascade
(1st layer 2 ~~features~~ filters ⟹ last layer 60 61 filters..)

Local feature

2g,

- Motivation: +) Global representations have major limitations
  (pedestrian: head, arm, leg..)
  +) Local feature: more robust to { Occlusions, Articulation, Intra-class variations }
  (half the face is hidden,
  find the eye → face;
  find leg angle; different koala .. but same nose)

- Application: +) Image matching
  +) Image stitching

- Procedure: 1) Find a set of distinctive key-points
  2) Define region around each keypoints
  3) Extract & normalize region content
  4) Compute a local descriptor from normalized region
  5) Match local descriptors / Find corresponding pairs

⊛ Detector Requirements:
- Invariant to Geometric Invariance (translation, rotation, scale,)
- Robust to lightning, noise, blur..
- Local: not global feature ⇒ robust to occlusion, clutter
- Distinctive.
- Efficient: real time
- Quantity

⊛ Many Existing Detectors available:

Hessian & Harris; Laplacian, DoG; Harris- / Hessian- Laplace;

Harris- / Hessian- Affine; EBR & IBR ..

- Corners are repeatable & distinctive (significant changes in all direction)

+) Harris Detector: 2nd Moment Matrix

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

window function

$$M = R^{-1} \cdot \begin{bmatrix} \lambda_1 & 0 \\ 0 & \lambda_2 \end{bmatrix} \cdot R \Rightarrow \begin{cases} \text{both } \lambda_1 \text{ & } \lambda_2 \text{ is great} \Rightarrow \text{corner} \\ \begin{cases} \lambda_1 \text{ is great} \\ \lambda_2 \approx 0 \end{cases} \Rightarrow \text{edge} \end{cases}$$

Eigen value decomposition

- Instead of find λ_i, we can do a cheap approximation:

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

α = const (0,09 → 0,06)

Just need M

$$\text{If} \begin{cases} R > 0 \Rightarrow \text{corner} \\ R < 0 \Rightarrow \text{edge} \end{cases}$$

- Window function $w(x,y)$: uniform - square window: not rotation invariant

⇒ Gaussian window (rotation invariant)

$$\Rightarrow M = g(\sigma) * \begin{bmatrix} I_x^2 & I_x I_y \\ I_x I_y & I_y^2 \end{bmatrix}$$

the window function is to consider the window around a pixel

Because the Harris Detector doesn't say that there is a corner at that pixel but that there is a corner in the window around that pixel

⇒ Harris Detector algorithm summary: 88

1, Image derivatives: I_x, I_y — 1st derivative

Use derivative of Gaussian

2, Square img_der: I_x^2, $I_x I_y$, I_y^2 — σ_1

3, Gaussian filter $g(\sigma)$: $g(I_x^2)$, $g(I_x I_y)$, $g(I_y^2)$ — $\sigma_2 = 2\sigma_1$

4, Cornerness func: $R = \det[M(\sigma_I, \sigma_D)] - \alpha \cdot [\text{trace}(M(\sigma_I, \sigma_D))]^2$

$= g(I_x)^2 g(I_y)^2 - g(I_x I_y)^2 - \alpha\, g(I_x)^2$

$= g(I_x^2).g(I_y^2) - [g(I_x I_y)]^2 - \alpha [g(I_x^2) + g(I_y^2)]^2$

α = const ($0{,}04 \to 0{,}06$)

5, Non-Maximum Suppression & Threshold

⇒ Problem: Harris Detector is rotation invariant but is not scale invariant

Note: with window func., Harris Detector takes into account larger image neighborhood, unlike Hessian Detector

+) Hessian Detector: — 2nd derivative

Intuition: Search for strong derivatives in 2 orthogonal direction will focus on mainly corners & strong textured areas

Also use 2nd Gaussian derivative with σ_1

- $\text{Hessian}(I) = \begin{bmatrix} I_{xx} & I_{xy} \\ I_{xy} & I_{yy} \end{bmatrix}$ ⇒ $\det(\text{Hessian}(I)) = I_{xx} I_{yy} - I_{xy}^2$ — Harris: just corners
- Apply NMS, threshold..
- $\det(H(I)) >$ threshold ⇒ corners

$\sigma^4 \det(H) = \sigma^4 [I_{xx}(\sigma).I_{yy}(\sigma) - I_{xy}^2(\sigma)] > t$

In practice, actually combine both of these 2

If: want precise locations ⇒ Harris
additional locations ⇒ Hessian

$\frac{Tr(H)^2}{Det H} = \frac{(r+1)^2}{r}$ $\quad r = \frac{\lambda_1}{\lambda_2}$

$r > 10$ eliminate (edges)

$r = 1$ good

* σ^4 because the peaks of der_Gaussian is reduce by σ^2

$G = \frac{1}{2\pi\sigma^2} \cdots$ and we have $I_{xx} \cdot I_{yy}$

⇒ σ^4 so that t is independent with σ

40) +) # Scale Invariant Region Selection:

- Exhaustive search over different scale: is stupid

- Scale signature function

- Laplacian of Gaussian – "blob" detector

Define characteristic scale as the scale produces peak of LoG respon

$$L_{xx}(\sigma) + L_{yy}(\sigma) \lessgtr \sigma^2 \Rightarrow \text{list of } (x, y, \sigma)$$

- Efficient Approximation LoG by DoG

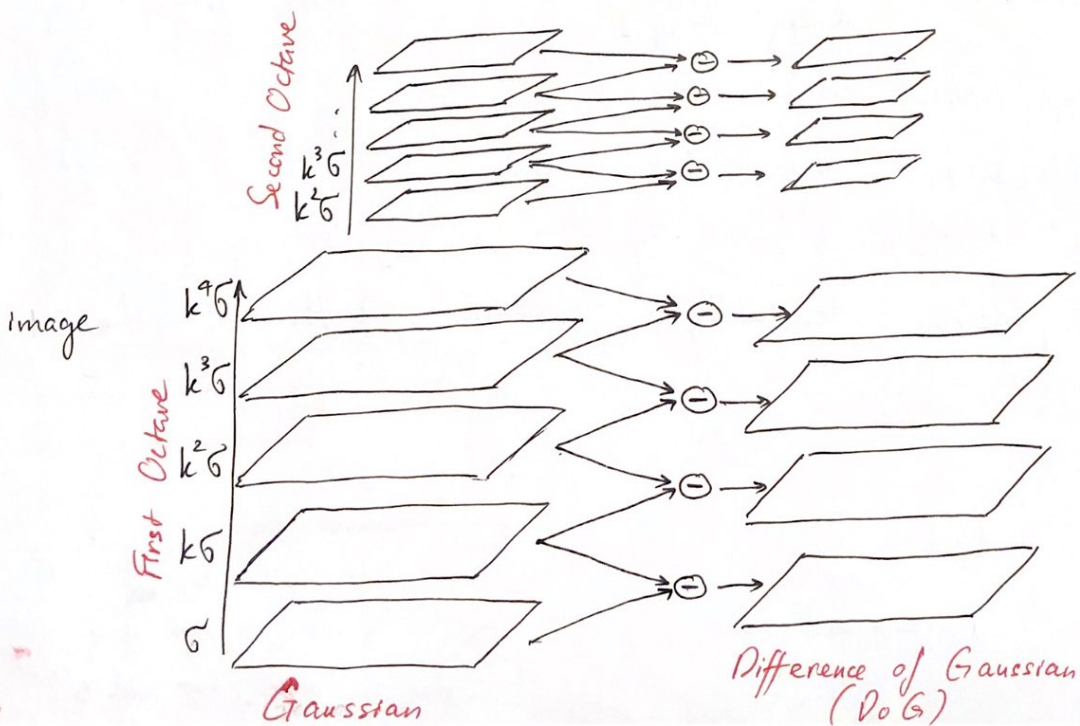
$$L = \sigma^2 \left(G_{xx}(x, y, \sigma) + G_{yy}(x, y, \sigma) \right)$$

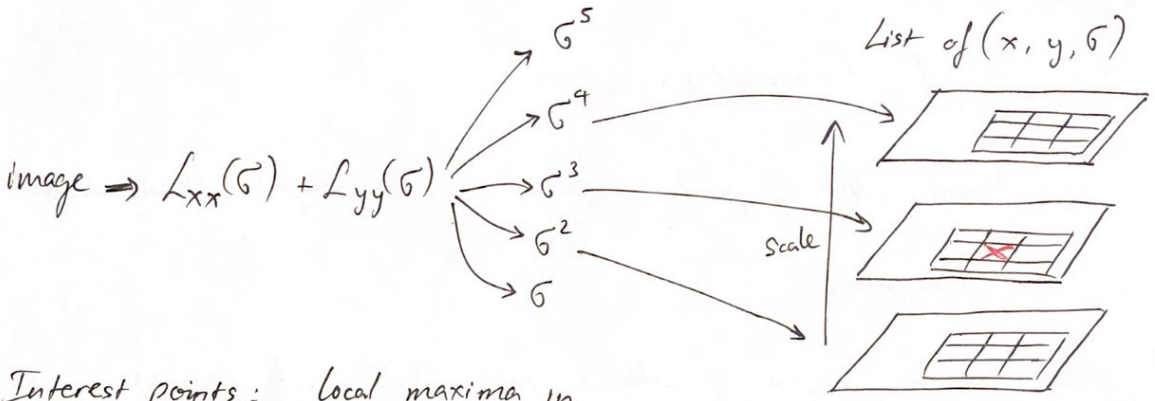
$$DoG = G(x, y, k\sigma) - G(x, y, \sigma)$$

⇒ ## Scale Pyramid:

$\sigma = 1,6$

$k = \sqrt{2}$

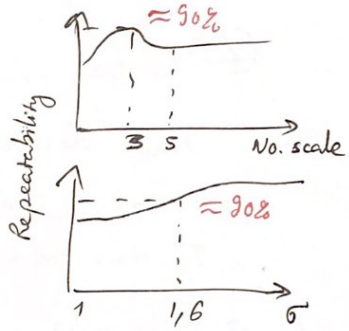




Interest points: local maxima in scale space of LoG (min or max among $27 - 1 = 26$ neighbors)

Tips

- Should choose 3-5 scales sampled per octave
- Init $\sigma = 1,6$



-

		k scale					
	1st	$1/\sqrt{2}$	1	$\sqrt{2}$	2	$2\sqrt{2}$	. .
Octave	2nd	$\sqrt{2}$	2	$2\sqrt{2}$	4	$4\sqrt{2}$	. .
	3rd	$2\sqrt{2}$	4	$4\sqrt{2}$	8	$8\sqrt{2}$	. .

Note: The next octave is 4 times smaller than the previous (2 times in each axis)

⇒ Thus applying σ_i at one octave will be just equal to apply $2\sigma_i$ at the previous octave?

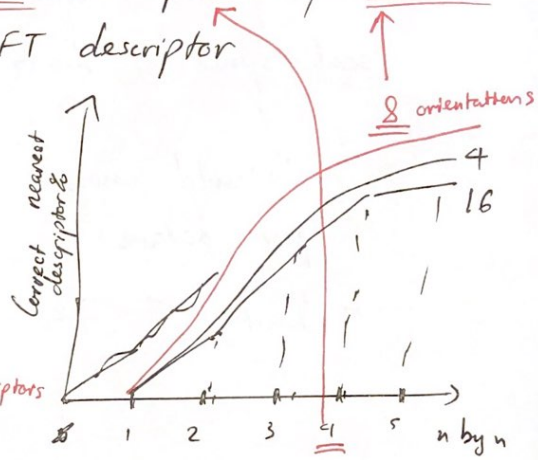
+) Local Descriptors:

- Again, list of pixel intensities .. is stupid, translational variant

⇒ Histograms

① Divide region of interest into 4x4 sub-patches / 8 bins

⇒ 4x4x8 = 128-dim SIFT descriptor



② To be rotational invariant:

Find dominant grad

⇒ rotate to that direction

(If there are 2 dominant grad → create 2 descriptors

③ ⊛ { Normalize
Clip (0,2)
Re-normalize

⊛ Minimum size of the region should be → 32x32 at worst 16x16

+) Key point matching: find nearest neighbor

If the ratio of distance between best & 2nd best is very close (0,8)

⇒ don't pick it

SIFT: Scale Invariant Feature Transform

+) Find consistent configurations:

2 types of problem

- Warping: given { source, transformation } ⇒ find output
- Alignment: given { source, output } ⇒ find transformation

- 2D Affine Transformation:

$$\begin{bmatrix} x' \\ y' \\ w \end{bmatrix} = \begin{bmatrix} a & b & c \\ d & e & f \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ w \end{bmatrix}$$

Translation

$$\begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix}$$

Rotation

$$\begin{bmatrix} c\theta & -s\theta & 0 \\ s\theta & c\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Scaling

$$\begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Shearing

$$\begin{bmatrix} 1 & sh_x & 0 \\ sh_y & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Trans + Rot = Euclidean transf.

Trans. + Rot. + Scaling = Similarity trans.

Trans + Rot + Scaling + Shear = Affine trans

+ Projective warp = Projective transf.

- Fitting a Affine Transformation:

6 variables ⇒ need 3 points

$$\begin{bmatrix} x_i' \\ y_i' \end{bmatrix} = \begin{bmatrix} m_1 & m_2 \\ m_3 & m_4 \end{bmatrix} \begin{bmatrix} x_i \\ y_i \end{bmatrix} + \begin{bmatrix} t_1 \\ t_2 \end{bmatrix} \Leftrightarrow \begin{bmatrix} x_i & y_i & 0 & 0 & 1 & 0 \\ 0 & 0 & x_i & y_i & 0 & 1 \\ & & \vdots & & & \end{bmatrix} \cdot \begin{bmatrix} m_1 \\ m_2 \\ m_3 \\ m_4 \\ t_1 \\ t_2 \end{bmatrix} = \begin{bmatrix} \vdots \\ x_i' \\ y_i' \\ \vdots \end{bmatrix}$$

⇒ Solve with least Squares Estimation

$x = (\text{pseudo inverse of } A) \cdot B$

2

+ Homography: a projective transformation

$$\begin{bmatrix} x' \\ y' \\ z' \end{bmatrix} = \begin{bmatrix} h_{11} & h_{12} & h_{13} \\ - & - & \\ h_{31} & h_{32} & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} ; \quad \begin{bmatrix} x'' \\ y'' \\ 1 \end{bmatrix} = \frac{1}{z'} \begin{bmatrix} x' \\ y' \\ z' \end{bmatrix}$$

$$\Leftrightarrow \quad x' = H . x \quad ; \quad x'' = \frac{1}{z'} x'$$

$$\Leftrightarrow \quad \begin{bmatrix} x_{B_1} & y_{B_1} & 1 & 0 & 0 & 0 & -x_{A_1}x_{B_1} & -x_{A_1}y_{B_1} & -x_{A_1} \\ 0 & 0 & 0 & x_{B_1} & y_{B_1} & 1 & -y_{A_1}x_{B_1} & -y_{A_1}y_{B_1} & -y_{A_1} \\ & & & & \ddots & & & & \end{bmatrix} \begin{bmatrix} h_{11} \\ \vdots \\ h_{32} \\ 1 \end{bmatrix} = [0]$$

$$\Leftrightarrow \quad A . h = 0$$

Do some Singular Value Decomposition shit $\Rightarrow h$

- When we have perpendicular camera axes, $h_{33} \approx 0$
- If not, can set as $A'h' = b$, $h' = [h_{11} \dots h_{32}]^T$ then use pseudo inverse to find H

+ Dealing with Outliers: each transformation as a point (8D)

- RANSAC:

- Alternatively: Generalized Hough transform

Deep Learning

L11, 12, 13)

- Max pooling: robustness to translation
- If use Batch Norm ⇒ no need of biases
  Turn it off to have less params
- If next layer after Batch norm is linear, conv (might be a RELU in the middle), then don't need scale γ of Cause instead, we can multiple the weights later
- Instead of Max pooling, use stride in conv layer, it reduce running time with less calculation
- Global Average Pooling Layer to deal with input of different size, and also reduces params significantly (h x w x d) ⇒ (1 x 1 x d) average of each (h x w)
- Augmentation: better validation results
- VGGnet vs AlexNet: more conv layers with smaller filter size is better than a conv layer with larger size:
  2 @ 3×3 is better than 1 @ 5×5
  less params, while same receptive field
- 1×1 conv layer = "bottleneck" layer, for dimensionality reduction

- GoogLeNet: Inception modul

  No FC layer

- Residual Net: Skip connection ⇒ better grad propagation

- Why don't scale up the gradients when dealing with grad. vanishing?

  Cause the signal-noise ratio, we will also scale up the noise

- Top 1 vs Top 5 accuracy rate?

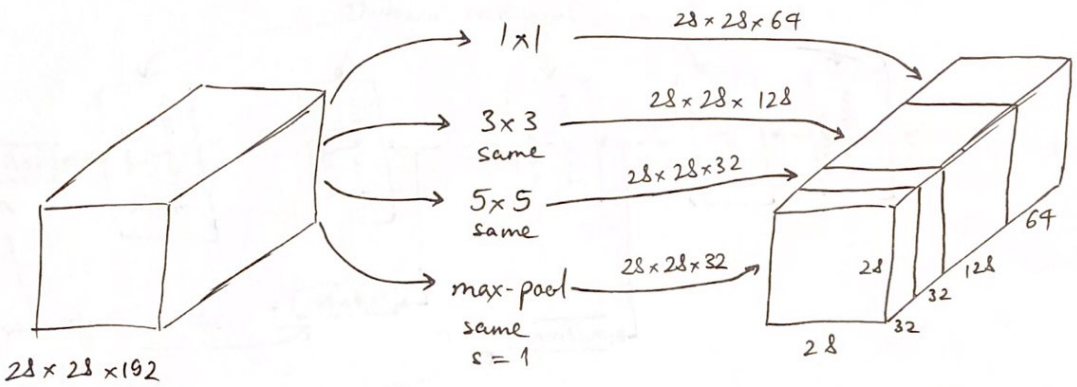
  Example: output: Dog 70%, Cat 20%, Pig 5%, ⇒ Top 1: False; Top 5: Correct

  True Label is "Cat".

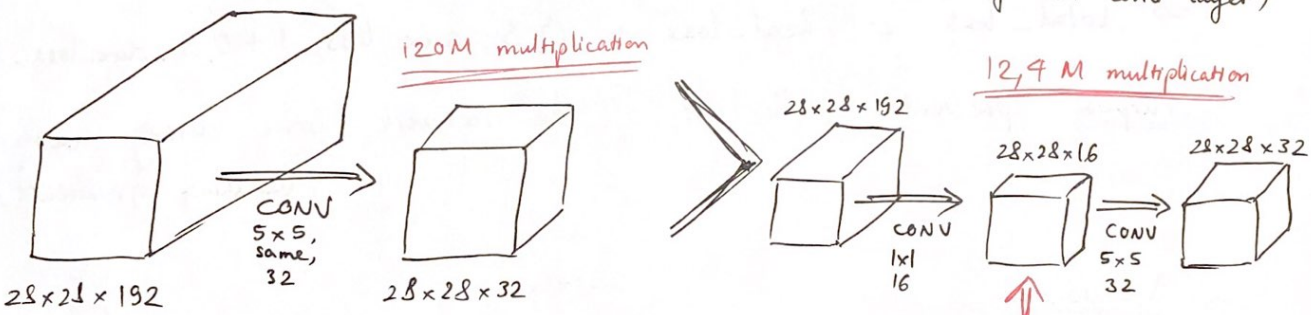
- Transfer learning: small data set: retrain last classifier layers; medium-sized dataset: retrain more layers (fine-tune)

- Residual Net from exercise

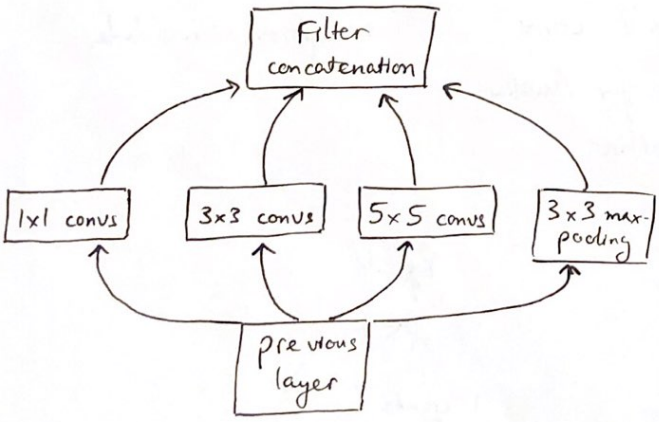
+) GoogLeNet : Inception modul



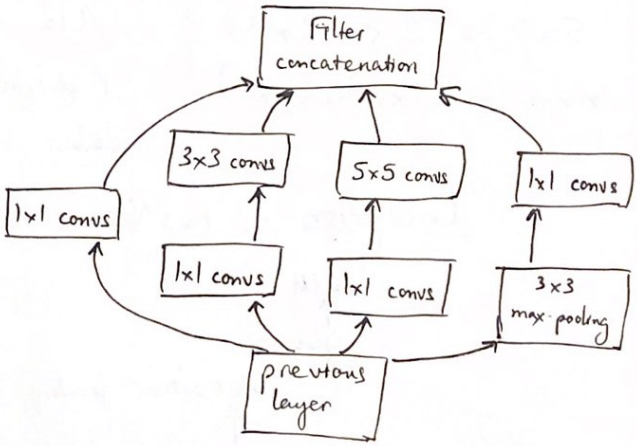
- Naive version vs with dimension reductions (use of 1×1 conv layer)



↓ More computational cost
More parameters

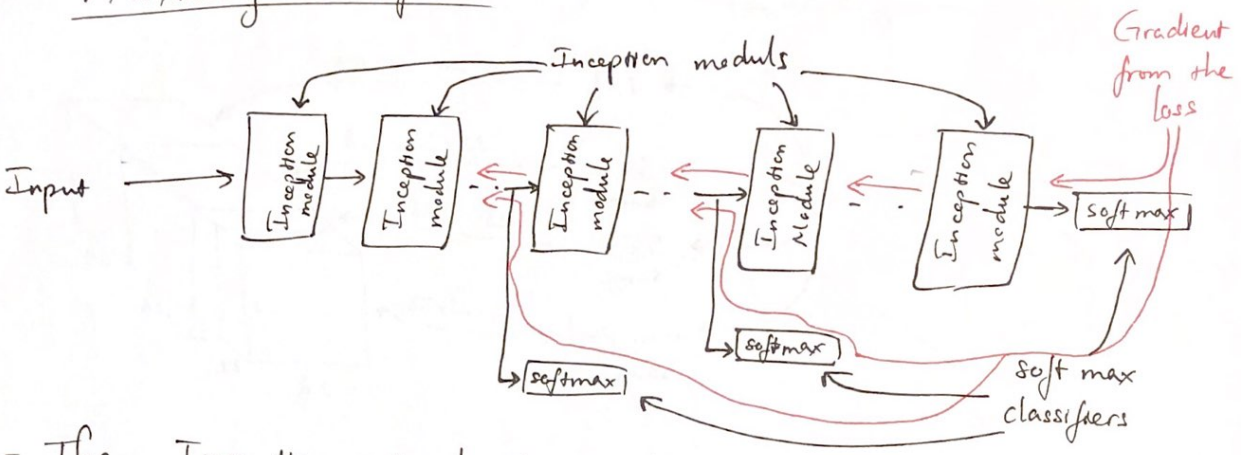


Inception module (Naive version)



Inception module with dimension reductions

– Auxiliary classifiers:



- The Inception network is simply a stack of 9 inception modules
- With 2 extra branches, each branch is applied a softmax to calculate auxiliary loss

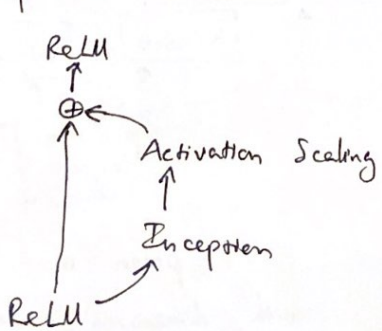
$\Rightarrow$ Total_loss = Real_loss + 0,3 . aux_loss_1 + 0,3 . aux_loss_2

⊗ Purpose: prevent middle part of the network from "dying" out (vanishing gradients)

– Versions:

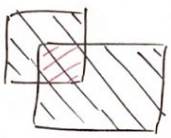
Inception .v2	Inception .v3	Inception .v4
Factorize filter size $5\times5 \Rightarrow 2\times(3\times3)$ $n\times n \Rightarrow (1\times n)$ & $(n\times1)$	RMS Prop Optimizer Add 7x7 conv Batch norm for Auxiliary Label smoothing	Uniformize the Inception modules

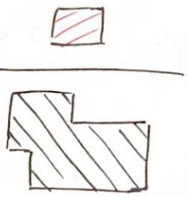
– Inception – ResNet:



+) IoU (Intersection over Union)

2 boxes

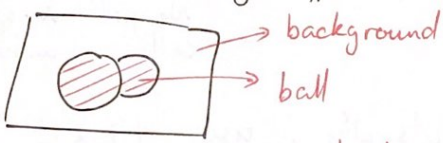


$\Rightarrow$ $IoU = \frac{\text{Intersection}}{\text{Union}} =$ 

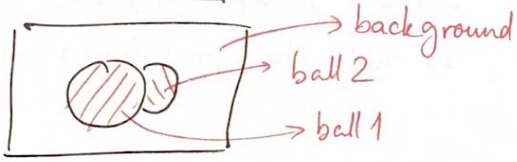
- Use to assess accuracy: correct if $IoU \geq$ threshold
  (commonly threshold = 0,5, or greater)
- Also used in Non max Suppression in YOLO

+) Definition:

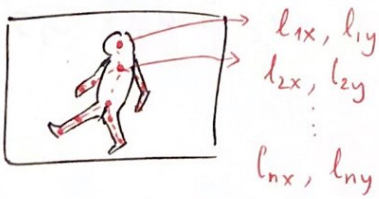
- Classification
- Classification & Localization } 1 object
- Detection : multiple objects, of possibly different classes
- Semantic Segmentation



- Instance Segmentation



+) Landmark detection:



$\Rightarrow$ Supervised training with CNN
(Regression)

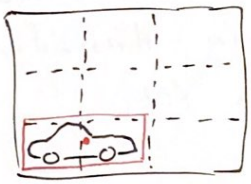
Label key landmark

$\Rightarrow$ Usage: [estimate pose
recognize emotion
. .

Anchor box based detectors:

+) **YOLO:** you only look once

Example: { 3 x 3 grid
2 anchor boxes [1] [2]
3 class (pedestrian, car, motorcycle)



For each grid cell, the out put will be:

$$y = [\underbrace{p_c}_{\text{prob. that there is object}} \underbrace{b_x\ b_y\ b_h\ b_w}_{\text{bounding box coordinate}} \underbrace{c_1\ c_2\ c_3}_{\text{prob. of each class}} \underbrace{p_c\ b_x\ b_y\ b_h\ b_w\ c_1\ c_2\ c_3}_{\text{anchor box 2}}]^T$$

⇒ Total output: $\underbrace{3 \times 3}_{\text{No. grid cell}} \times \underbrace{2}_{\text{No. of anchor box}} \times \underbrace{8}_{5 + \text{No. of classes}}$

- Usually, use 19 x 19 grid cell
  so that the probabilities of 2 same class objects in the same grid cell is small

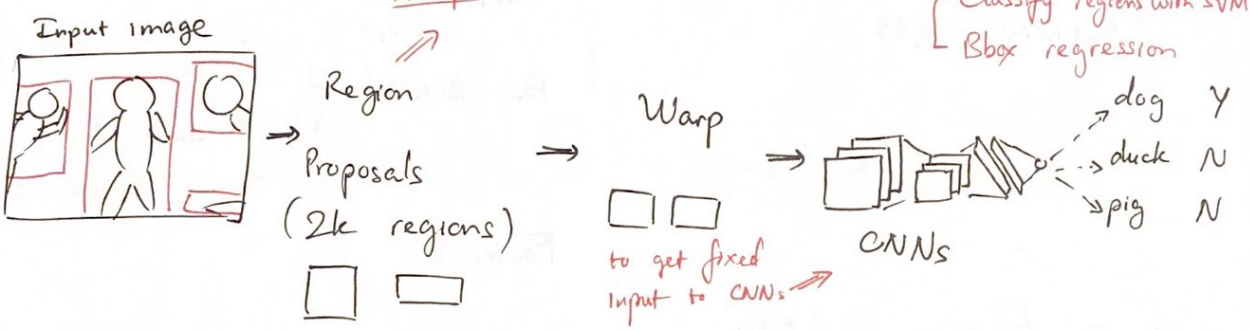
- Filter result with Non-max suppression:
  1) Threshold: remove boxes with low p_c conficlent. ($p_c \leq 0,6$)
  2) Non-max suppression: keep the box with highest confidence while remove boxes with IoU ($\geq 0,5$)
  (Do this seperately for each class)

+) **SSD:** Single shot detector

*) CNNs for Object Detection:

⊛ Regional-~~based~~proposal based Detectors:

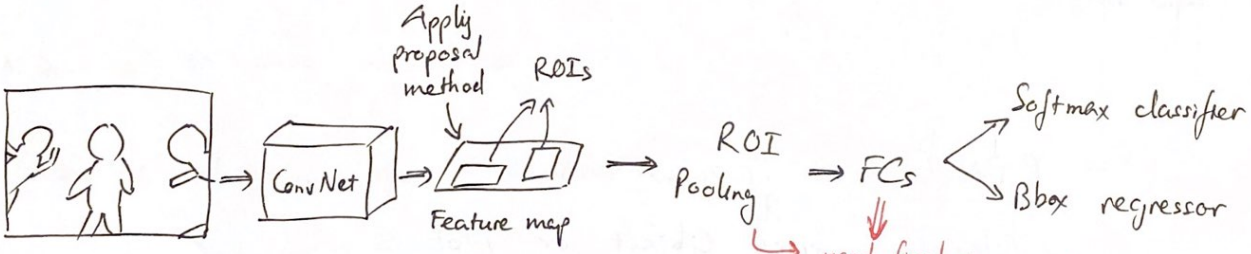
+) R-CNN:



It works .. but there are so much to improve.

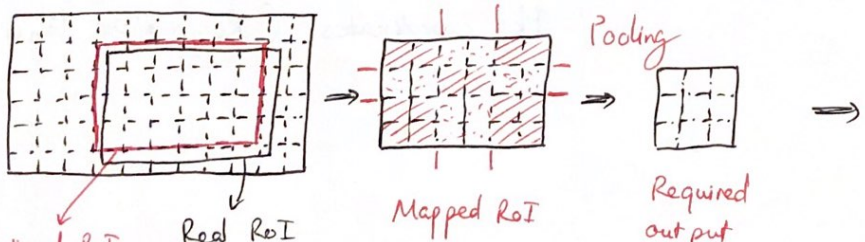
Training 3 days & testing 47s per image are slow

+) Fast R-CNN:



* Still use external proposal method (e.g. Selective Search) to get RoIs

- RoI Pooling:



Also could remove the extra pixels..

Subject to data losses

+) Compared R-CNN vs SPP vs Fast R-CNN:

	Training time (hours)
RCNN	84
SPP	25,5
Fast RCNN	8,75

Test time (secs)

RCNN { 49
47 (Excluding Region Proposal)

SPP { 4,3
2,3 (_____)

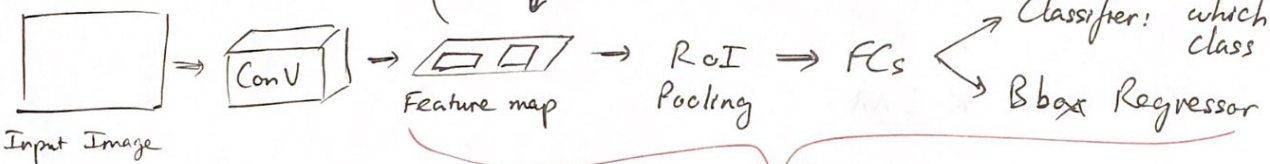
Fast-RCNN { 2,3
0,32 (Exclude ..)

⇒ Problem of Fast R-CNN: runtime dominated by Region Proposals !

Faster RCNN: 0,2

⇒ +) Faster RCNN:

Only this is new RPN → Classifier : Object or not
→ Bbox Regressor
RoIs



The rest is the same as Fast RCNN

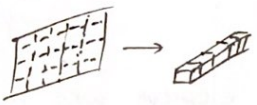
- RPN (Region Proposal Network): is another ConV-net

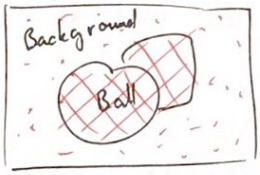
{ Classify error: Object or Not
Bbox error

⇒ With k anchor boxes ⇒ { 2k scores (Classifier layer)
4k coordinates (Regressor layer)

+) Semantic Segmentation: label each pixel with class label but don't care how many objects there are

- Ideas: - Can't use normal FCs on the whole image, cause we will loose pixel location

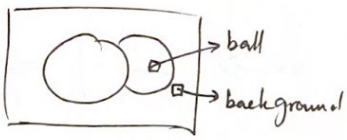




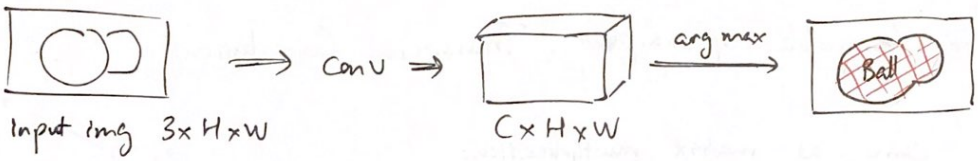
- Sliding window: run small patch through CNN

very inefficient

no reuse of features between shared patches

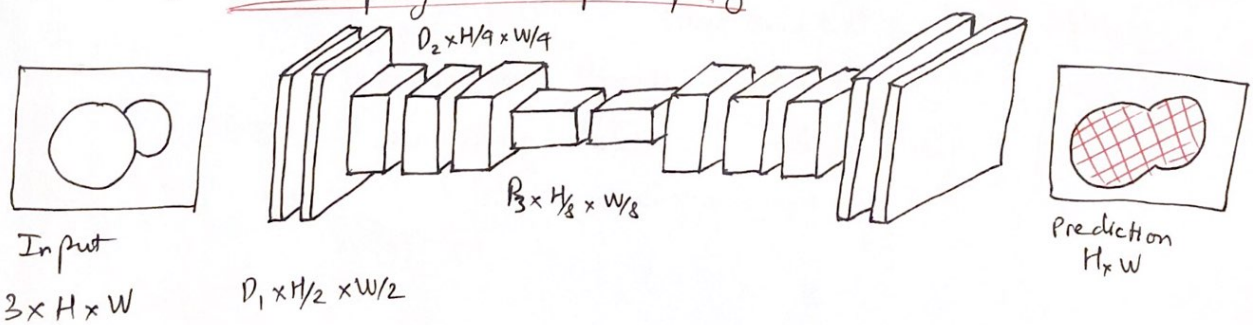


⇒ FCN (Fully Convolutional Net): can process arbitrary image size



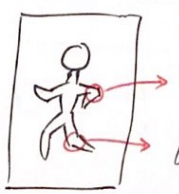
- Problem: conv at original image resolution will be expensive

⇒ downsampling and upsampling inside the network



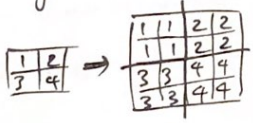
⊗ Application: Human Pose Estimation

- Example: SegNet



+) Upsampling : "Unpooling"

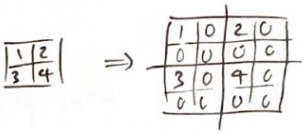
① Nearest neighbor:



Simplest version

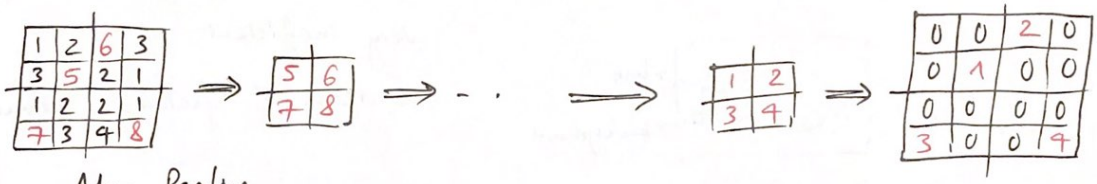
Problem: blocky output structure

② "Bed of Nails":



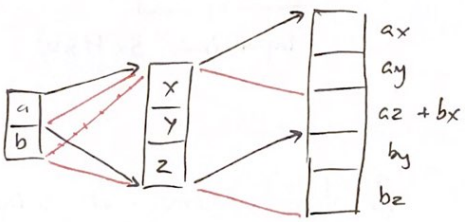
Problem: fixed location for upsampled stimuli

③ Max unpooling: remember which element was max



④ Learnable upsampling: Transpose Convolution

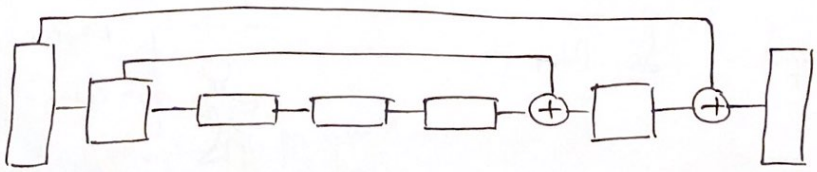
- Conv as matrix multiplication:



- Also called:
  - Up convolution
  - Fractionally strided convolution
  - Backward strided convolution

But not ~~deconvolution~~

+) Extension: Skip Connections ⇒ Example: U-Net



+) Mask R-CNN:

- Object Detection + Semantic Segmentation = Instance Segmentation

Faster R-CNN + FCN = Masked R-CNN

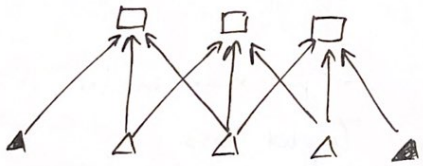
- RoI Align vs RoI Pooling (Fast RCNN)

Solve with bilinear interpolation ⟸ Problem: loss of data, misalignment of pixels

+) Atrous Convolutions: (Dilated)

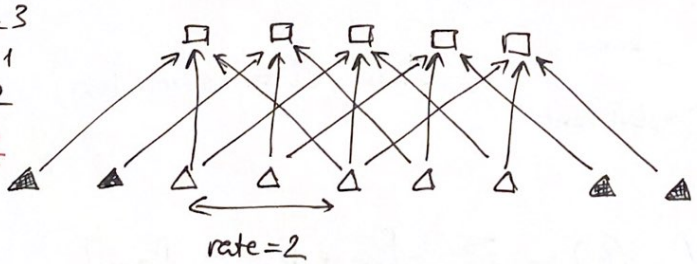
/à trous/ method with holes "méthode à trous"

kernel = 3
stride = 1
pad = 1



- increase receptive field without increasing computation
- like the effect of pooling without loosing resolution

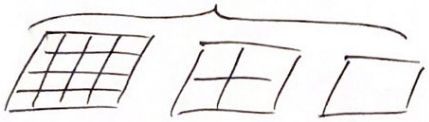
kernel = 3
stride = 1
pad = 2
rate = 2



r as dilation factor

+) Spatial Pyramid Pooling: sampling at many levels then stack all together.

16×256 + 4×256 + 1×256 ⇒ FCs



⇒ lets the network decides which pooling scales to use, or even, how to combine different scales..

⇒ + Atrous Spatial Pyramid Pooling: (ASPP)

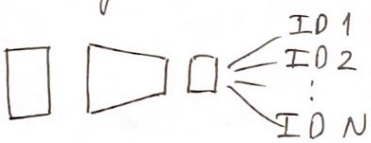
Apply Dilated Conv with r = 6, 12, 18, 24 .. then stack the output

L16/

⊗ CNNs for Matching

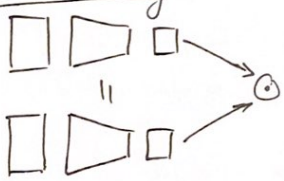
3 Types of Models used for Matching Tasks

- Identification:



Training: Multi Class Classification Loss

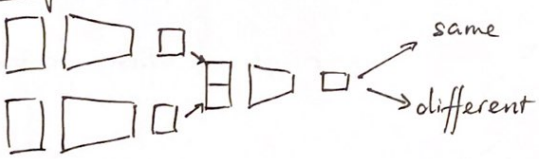
- Embedding



Siamese Network

Large-margin loss or Triplet loss

- Verification:



2-Class Classification loss

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

$D_{a,p}$: distance between a & n

$[..]_+$: $\max(.., 0)$

- If we choose randomly, Anchor & Negative are probably already very different

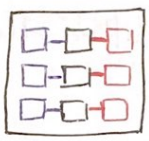
⇒ Need to: Mining Hard / Medium-hard | Triplets

↳ Practical Implementation:

- Offline Hard Triplet Mining: (OHM)

- Embed data with f_θ ⇒ for all data ⇒ very wasteful, high computational cost
- Mine Hard triplets
- Update Embedding f_θ

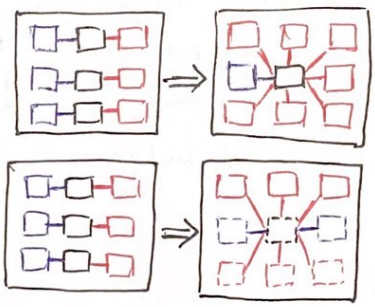
Then do mini-batch learning



- Online Hard Triplet Mining

Idea: 1) Other members of triplets as negative candidates

2) Triplets are constructed only within batch



- Batch All: $d_{j,a,n}^{i,a,p} = D(f_\theta(x_a^i), f_\theta(x_p^i)) - D(f_\theta(x_a^i), f_\theta(x_n^j))$

$$L_{BA}(\theta; X) = \underbrace{\sum_{i=1}^{P}\sum_{a=1}^{K}}_{\text{all anchors } P.K} \underbrace{\sum_{p=1,\, p\neq a}^{K}}_{\text{all pos. } K-1} \underbrace{\sum_{j=1,\, j\neq i}^{P}\sum_{n=1}^{K}}_{\text{all neg. } (P-1).K} \left[m - d_{j,a,n}^{i,a,p}\right]_+$$

P classes, each K images

Use all combinations of triplet within batch

- Batch Hard: For each anchor, within batch, select the hardest pos & neg.

$$L_{BH}(\theta; X) = \sum_{i=1}^{P} \sum_{a=1}^{K} \left[m + \max_{p=1..K} D\left(f_\theta(x_a^i), f_\theta(x_p^i)\right) - \min_{\substack{j=1..P \\ j \neq a \\ n=1..K}} D\left(f_\theta(x_a^i), f_\theta(x_n^j)\right) \right]_+$$

all anchors | hardest positive | hardest negative

- Lifted Embedding: consider all but the anchor-pos. pair as negatives

$$L_L(\theta; X) = \sum_{(a,p) \in X} \left[D_{a,p} + \log \sum_{\substack{n \in X \\ n \neq a \\ n \neq p}} \left(e^{m - D_{a,n}} + e^{m - D_{p,n}} \right) \right]_+$$

- Generalized Lifted Embedding: batch all + lifted embedding

$$L_{LG}(\theta; X) = \sum_{i=1}^{P} \sum_{a=1}^{K} \left[\log \sum_{\substack{p=1 \\ p \neq a}}^{K} e^{D\left(f_\theta(x_a^i), f_\theta(x_p^i)\right)} - \log \sum_{\substack{j=1 \\ j \neq i}}^{P} \sum_{n=1}^{K} e^{m - D\left(f_\theta(x_a^j), f_\theta(x_n^i)\right)} \right]_+$$

all anchors | all positives | all negatives

- Batch hard performs best
  additional improvements with soft margin

- Matching applications: Face Identification very good ⇒ privacy?
  Stereo Matching

3D Reconstruction

L17,

ⓐ Geometric Vision:

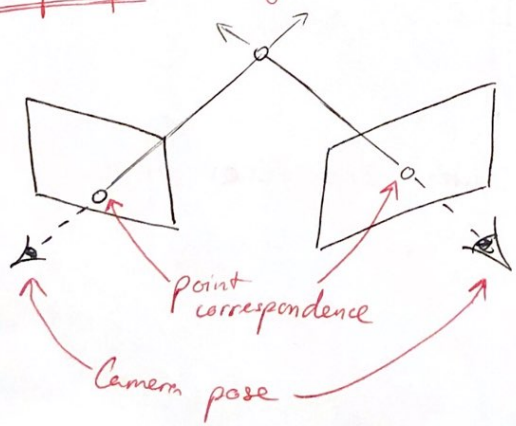
+, Visual Cues (Details)
- Shading
- Texture
- Focus
- Perspective
- Motion

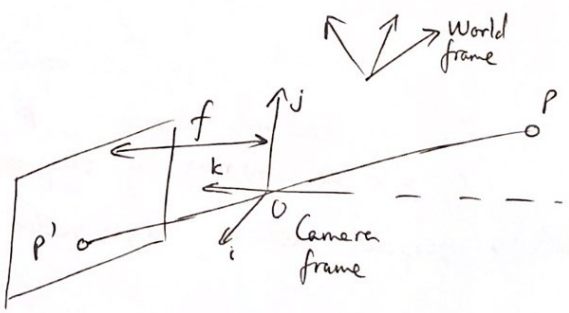
} The focus of lecture

+, Stereo Vision: process of extracting 3D information from multiple 2D views of a scene

ⓑ Epipolar geometry: the geometry of stereo vision

Basic principle: Triangulation





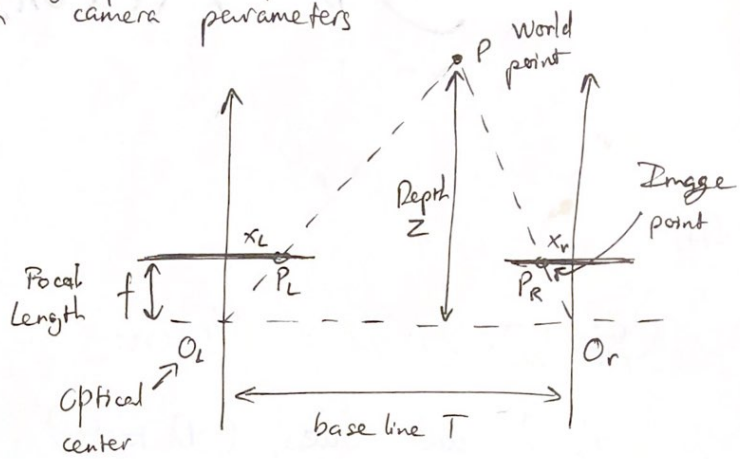
- Extrinsic params: camera frame ⟺ world frame
  rotation matrix, translation vector
- Intrinsic params: Image coord. ⟺ pixel coord.
  focal length, pixel size, . .

- Simple case: [assuming parallel optical axes
[known camera parameters

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

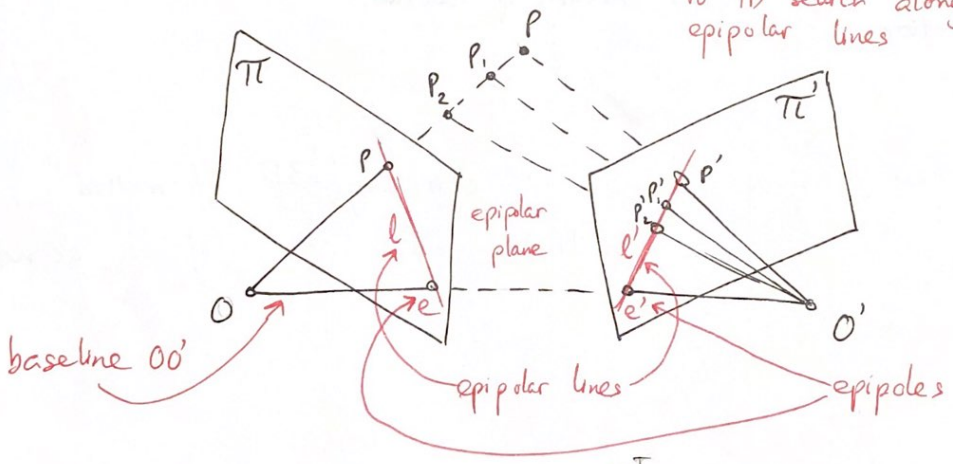
$$\Rightarrow Z = f\frac{T}{x_r - x_l}$$

$x_r - x_l$: disparity



⇒ We can estimate depth of image point from focal length f, baseline T, & disparity $(x_r - x_L)$

- Stereo correspondence constraints: reduces correspondence problem to 1D search along conjugate epipolar lines



$$Ex' \qquad E^T x$$

$$Ee' = 0 \qquad = E^T e$$

E: as essential matrix : $E = T_x \cdot R$

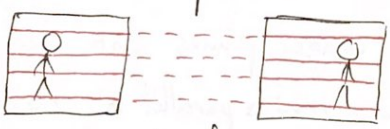
e, e': as positions of epipoles

⇐ epipoles will be eigenvector of E with eigenvalues of 0

- Example:

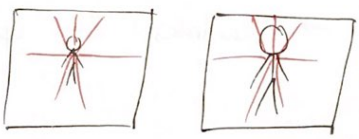
1) Converging cameras: as position of 3D point varies, epipolar lines rotate about the baselines

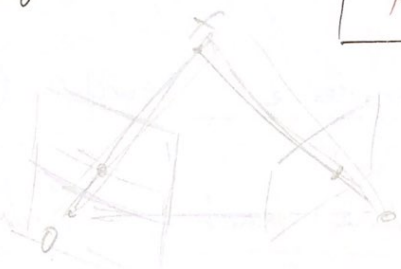
2) Motion Parallel with Image Plane:
the epipolar lines will be all parallel



3) Forward Motion: epipoles are at the same coordinates in both images.

Epipolar lines converge like a star





+) Algebra stuffs:

$$X' = RX + T \quad ; \quad \vec{a} \times \vec{b} = [a_x] \cdot \vec{b}$$

R: rotation matrix; T: translational vector

$[a_x]$: skew matrix corresponding to $\vec{a}$

$$\begin{bmatrix} 0 & -a_z & a_y \\ a_z & 0 & -a_x \\ -a_y & a_x & 0 \end{bmatrix}$$

$$\Rightarrow T \times X' = T \times RX + T \times T = T \times RX$$

$$\Rightarrow X' \cdot (T \times X') = X' \cdot (T \times RX)$$

$$\Rightarrow X' \cdot (T \times RX) = 0$$

$$\Rightarrow X' \cdot (T_x \cdot RX) = 0 \Rightarrow \text{Set } \boxed{E = T_x \cdot R}$$

essential matrix (extrinsic infos)

$\Rightarrow X'^T \cdot E \cdot X = 0$ holds true for all rays p & p' in the same direction as X & X'

$l' = Ep$: epipolar line for point p

$l = E^T p'$: ——— p'

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix}^T \cdot E \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = 0 \quad (x, y \text{ fixed})$$

$$\Leftrightarrow \begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} \cdot \begin{bmatrix} a \\ b \\ c \end{bmatrix} = 0$$

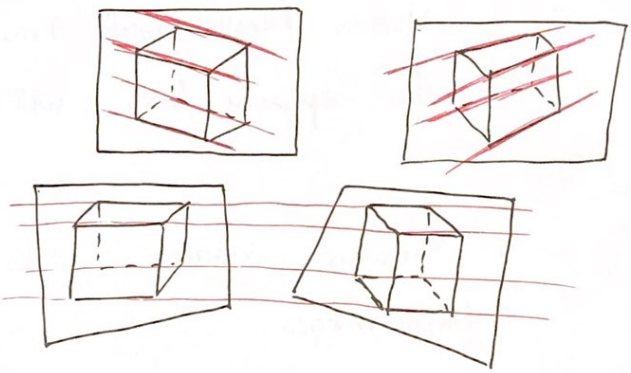
$$\Leftrightarrow ax' + by' + c = 0$$

$\Rightarrow$ Epipolar constraint: $p'^T \cdot E \cdot p = 0$

+) Stereo Image Rectification

Reproject images planes onto a common plane, that is parallel to the baseline

$\Rightarrow$ Scanlines are epipolar lines



+) Correspondence Search (Matching)

In practice, use both

Dense Correspondence Search	Sparse Correspondence Search
- For each pixel, find correspondence - Easy when epipolar lines are scanlines (apply rectification)	- Only for a set of detected features - Use feature description (Harris ?)

Pros

- Simple process
- More depth $\Rightarrow$ useful for surface reconstruction
- Efficiency
- Can have more reliable matches
- Less sensitive to illumination — robust

Cons

Problem with textureless regions / different viewpoints

- Have to know enough to pick good features
- Sparse information

⊛ Stereo Reconstruction

Main steps:
- Calibrate cameras
- Rectify images
- Compute disparity
- Estimate depth

This is just ideal case.
- what if, how can we get extrinsic info from calibration?
- when triangulation failed?

+ Matrix stuff: Solve $\boxed{Ax = 0}$

- By applying SVD: $A = U \cdot D \cdot V^T = U \cdot \begin{bmatrix} d_{11} & & \\ & \ddots & \\ & & d_{NN} \end{bmatrix} \cdot \begin{bmatrix} v_{11} & \cdots & v_{1N} \\ \vdots & \ddots & \vdots \\ v_{N1} & \cdots & v_{NN} \end{bmatrix}^T$

- Solution of $A \cdot x = 0$ is the null space vector of A corresponds to the smallest (last) singular vector of A

⊛ Camera Calibration:

+ Camera Parameters:

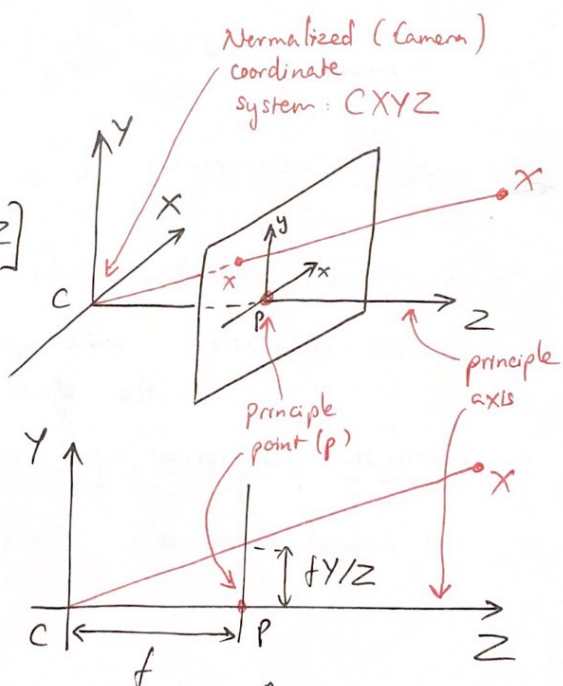
$$[X, Y, Z] \rightarrow [fX/Z,\ fY/Z]$$

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

$$\Leftrightarrow Z \cdot x = P \cdot X_{cam}$$

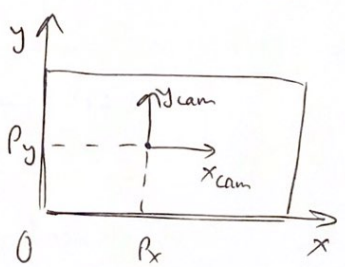
$$= \begin{bmatrix} f & & \\ & f & \\ & & 1 \end{bmatrix} \cdot \left[\begin{array}{ccc|c} 1 & & & 0 \\ & 1 & & 0 \\ & & 1 & 0 \end{array}\right] \cdot X_{cam}$$

Z: unknown normalization factor



- The principle point is not always in the middle of the image

$$\Rightarrow [X, Y, Z] \Rightarrow [fX/Z + p_x,\ fY/Z + p_y]$$



- Pixel Coordinates:

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

pixels/m m pixels

- Camera Rotation & Translation:

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

$\tilde{X}_{cam}$: coords. of point in camera frame; $\tilde{X}$: coords. of a point in world frame; $\tilde{C}$: coords of camera center in world frame

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

$$\rightarrow z \cdot x = K [I|0] \cdot X_{cam} = K \cdot [R \,|\, -R\tilde{C}] \cdot X$$
$$= K \cdot [R \,|\, t] \cdot X$$
$$= P \cdot X$$

⊛ Summary:

+) Intrinsic parameters: $K = \begin{bmatrix} m_x & & \\ & m_y & \\ & & 1 \end{bmatrix} \cdot \begin{bmatrix} f & s & p_x \\ & f & p_y \\ & & 1 \end{bmatrix} = \begin{bmatrix} \alpha_x & s & x_0 \\ & \alpha_y & y_0 \\ & & 1 \end{bmatrix}$

- s: skew, non rectangular pixels, in complicated settings..
- Radial distortion: while opening angle is more than 45° ⇒ the effect is no more negligible.

+) Extrinsic parameters: $[R|t]$ ⇒ Rotation + Translation

+) All: Camera Projection Matrix: $P = K[R|t]$

⊛ DoF:
- General pinhole camera: 9 DoF (3 int + 6 ext) p_x, p_y, f
- CCD Camera with square pixels: 10 DoF ($m_x = m_y \Rightarrow$ 4 int)
- ——— non ———: 11 DoF ($m_x \neq m_y$)
- General camera: 11 DoF ($m_x = m_y$)

↳ Calibrating a Camera:

- Use known "calibration object"
- Algorithm for checkerboard pattern:
  1. Perform Canny Edge Detector
  2. Fit straight lines
  3. Intersect lines to obtain corners

Very good localization $< 1/10$ pixel

- Rule of thumb: number of constraints should exceed number of unknowns by a factor of 5

Ex: 11 params of P ⇒ should have 55 constraints

each point : 2 constraints → should have 55/2 points

But at least, need 11/2 points

$$\lambda x_i = P X_i \Leftrightarrow x_i \times P X_i = 0 \Leftrightarrow \begin{bmatrix} 0 & -X_i^T & y_i X_i^T \\ X_i^T & 0 & -x_i X_i^T \\ -y_i X_i^T & x_i X_i^T & 0 \end{bmatrix} \cdot \begin{bmatrix} P_1 \\ P_2 \\ P_3 \end{bmatrix} = 0$$

$$\Leftrightarrow \begin{bmatrix} 0^T & X_1^T & -y_1 X_1^T \\ X_1^T & 0^T & -x_1 X_1^T \\ \dots & \dots & \dots \\ 0^T & X_n^T & -y_n X_n^T \\ X_n^T & 0^T & -x_n X_n^T \end{bmatrix} \begin{bmatrix} P_1 \\ P_2 \\ P_3 \end{bmatrix} = 0$$

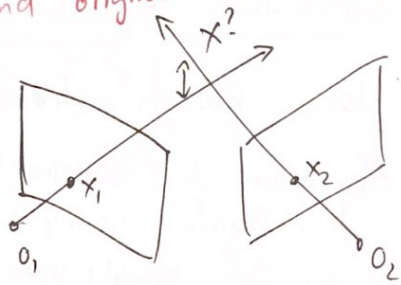
need calibration points in more than one plane

or else, we will get degenerate solution

- After that, still need "matrix decomposition problem" to go from P ⇒ intrinsic, extrinsic parameters

† Revisiting Triangulation:

Find original X



1) Use Linear Algebraic Approach

$[x_{1\times}] \cdot P_1 X = 0$
$[x_{2\times}] \cdot P_2 X = 0$
$\Rightarrow$ Stack
Solve with SVD

2) Nonliner Approach: $\arg\min_X \left[d^2(x_1, P_1 X) + d^2(x_2, P_2 X) \right]$

[most accurate
but no closed-form solution, only iterative algorithm

† Revisiting Epipolar Geometry:

1) Calibrated Case: $x^T \cdot E \cdot x' = 0$; $E = [t_\times] \cdot R$

- E: rank 2, singular, cause e', e is in the null space of E
  $Ee' = 0$, $E^T e = 0$
- 5 dof, up to scale (p_x, p_y, $m_x = m_y$, shew, scale?

2) Uncalibrated Case: $x = K\hat{x}$, $x' = K' \cdot \hat{x}'$

$\Rightarrow x^T \cdot K^{-1T} \cdot E \cdot K'^{-1} \cdot x' = 0 \Leftrightarrow x^T \cdot F \cdot x' = 0$

- Epipolar lines: $Fx' \perp F^T x$.
- $Fe' = F^T e = 0$
- F: singular, rank 2, 7 dof

Fundamental matrix

⊗ Estimate F with Eight Point Algorithm

$x = [u, v, 1]^T$
$x' = [u', v', 1]^T$
$\Rightarrow$ Stack 8 points
$\Rightarrow$ Use SVD

$$\Rightarrow \begin{bmatrix} u \\ v \\ 1 \end{bmatrix}^T \begin{bmatrix} F_{11} & \cdots & F_{13} \\ & & \\ F_{31} & & F_{33} \end{bmatrix} \begin{bmatrix} u' \\ v' \\ 1 \end{bmatrix} = 0$$

$$\Rightarrow [u'u, u'v, \ldots, u, v, 1] \begin{bmatrix} F_{11} \\ \vdots \\ F_{33} \end{bmatrix} = 0$$

If noisy data
$\Rightarrow$ F full rank 3, no epipoles
$\Rightarrow$ Use rank 2 estimation of F with SVD

⊗ Problem: Poor numerical conditioning $u'u \gg v \gg 1$

⇒ The Normalized Eight-Point Algorithm:

1, Center image data at origin
   Scale so mean squared distance = 2 pixels
2, Compute F
3, Enforce rank 2 constraint with SVD on F
   Set smallest singular value to 0
4,

Nonlinear Least-Squared Approach:

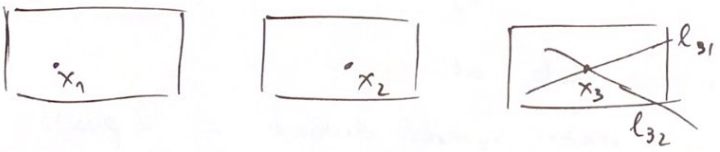
minimize $\sum_{i=1}^{N} \left[d^2(x_i, Fx_i') + d^2(x_i', F^T x_i) \right]$

+) 3D Reconstruction with Weak Calibration:

to estimate world geometry without requiring calibrated camera
only from a redundant set of correspondences

- Procedure:
  - Find interest point
  - Calculate correspondences
  - Compute epipolar geometry
  - Refine

+, Epipolar Transfer:



- Given projections of same point in 2 images: x_1 & x_2

- Failed when the 2 lines stack on each others