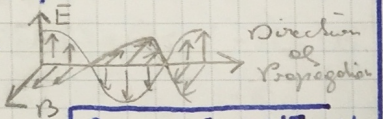
CVIA

Introduction: - Light

Our domain

↳ Optics: i) Geometrical
ii) Physical
iii) QM

↳ EM-Waves:
i) Wavelength
ii) Direction
iii) Amplitude
iv) Phase
v) Polarization



Energy is $\propto |E \times B|$. Indeed zero at some points. → But cannot average over time

Camera may have diffrent sensitivities!

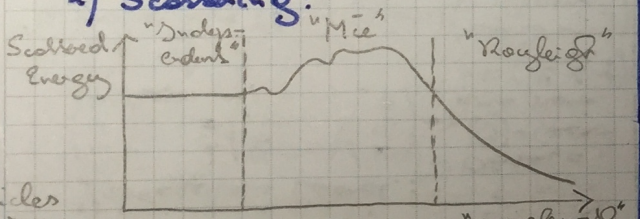
↳ Visible Light:

$\lambda_{BLUE} \sim 500\,nm \rightarrow \lambda_{RED} \sim 750\,nm$

Since Solar Spectrum peaks around 500nm

↳ Light - Matter Interaction:

i) Scattering:



Large Particles

"Independent": Clouds

"Mie": Comparable sizes, e.g. green cloud from volcano

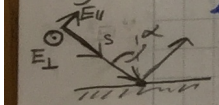
Small Particles ($d < 40\,nm$)

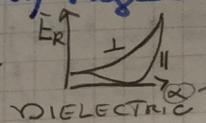
"Rayleigh": Medium with very small particles: Red light goes straight through! Blue light gets scattered!

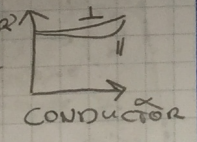
$E_S \propto \frac{1}{\lambda^4}$

Compare "Tyndall"

ii) Reflection:







iii) Refraction: $n_1 \sin\alpha_1 = n_2 \sin\alpha_2$
"Dispersion": $\alpha_2 = f(n_1, n_2, \alpha_1)$ (1)

iv) Absorption:
"Wavelengths are dissipated"
↳ Holes in solar spectrum

v) Diffraction: "Bending"

Illumination Techniques:

↳ Back Lighting: (Inspection)
Light source + diffuser plate behind object → Silhouette (use binary images)

↳ Directional Lighting: (Geo. Detect.)

↳ Diffuse Lighting: Used for shiny samples (prevent sharp shadows)

↳ Polarized Lighting: Distinguish
i) Diffuse vs. Specular
ii) Dielectric vs. Metal
Send polarized light!!!
Check if polarized light is reflected (diffuse will depolarize)
Send unpolarized under Brewster angle → Dielectric will polarize
Metal will stay unpolarized

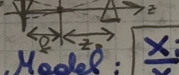
Law of Malus: $I_{(\omega)} = I_0 \cos^2\theta$

↳ Colored Lighting:

↳ Structured Lighting: (3D shape)

↳ Stroboscopic Lighting: (No motion blur)

Cameras:

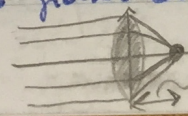


↳ Pinhole Model: $\frac{x_i}{x_o} = \frac{y_i}{y_o} = \frac{f}{-z_o} = -m$

↳ Problem: ↱ Maybe even diffraction!
→ Hole too small: Dark image
→ Hole too big: Unsharp image

↳ Solution: Use a thin lense ("reduces focal length")

↳ Focal length: f
By definition → Rays parallel to principal axis will go through point at distance f

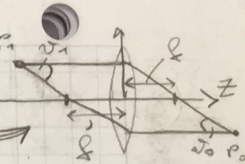


For thin lenses: "Lense maker equation" $\frac{1}{f} = \left[\frac{n_{IN}}{n_{OUT}} - 1\right] \cdot \left(\frac{1}{R_1} + \frac{1}{R_2}\right)$

Study Summary

↳ Gauss thin lense equation:

$$\frac{1}{z_0} - \frac{1}{z_1} = \frac{1}{f}$$



Derived relating y_0 and y_1

↳ Assumptions:
i) Spherical lens surface
ii) Incoming light ∥ axis
iii) Radii ≫ Th. Lenses
iv) Refractive index same on both sides!

Magnification of lense: $m = \frac{f}{(z_0 - f)}$

↳ Depth of Field (DOF):
Disadvantage of lense → for given z_0 only one z_i (= location of image plane) gives sharp image.
In practice there is an interval, where images are sharp (Δz)

↳ Aberations:
Violation of thin lense assumptions!
i) Geometrical:
→ Spherical aberation:
Parallel rays to principal axis don't all hit "focal point"
Can be corrected for!
→ Radial distortion:
Magnification "m" varies radially → Non linear projection
ii) Chromatic:
One solution: use multiple lenses → Snell: Different focal length for different wavelengths

↳ Design Types:
(Both use photoelectric effect to convert EM-radiation to charge)
i) CCD: ↱ Collection of charge
→ 1st Phase: "Integration"
→ 2nd Phase: "Read out"
(Still charges 1 by 1 to "charge amplifier" → charge to voltage conv.)
ii) CMOS: (Can avoid "blooming")
Each photo sensor has its own charge amplifier!
(More noise, lower "fill rate", faster)

↳ Color Cameras:
i) Prism: High-end / expensive
ii) Filter mosaic: Cheap / aliasing effects
iii) Color Wheel: Low framerate / more than 3 colors!

For large z: Say $f/z = c \in \mathbb{R}$
Becomes "Pseudo-Orthographic"
i.e. Projection + Scale

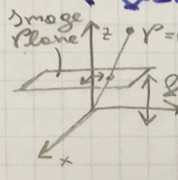
Projection Models:

↳ Orthographic Projections:

Projection lines are parallel & orthogonal to proj.-plane.
E.g. proj. on $z=0$: $\begin{pmatrix}x\\y\\z\end{pmatrix} \to \begin{pmatrix}x\\y\\0\end{pmatrix}$

↳ Perspective Projections: (camera frame)

Cartesian Coord.:
Solve $\begin{pmatrix}u\\v\\f\end{pmatrix} = t\begin{pmatrix}x\\y\\z\end{pmatrix}$

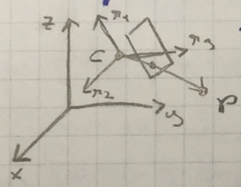


$\begin{pmatrix}u\\v\end{pmatrix} = \frac{f}{z}\begin{pmatrix}x\\y\end{pmatrix}$

Homogenous Coord.:

$\begin{pmatrix}u\\v\\w\end{pmatrix}_{20H} = \begin{pmatrix}f&0&0\\0&f&0\\0&0&1\end{pmatrix}\begin{pmatrix}x\\y\\z\end{pmatrix} \to$ Obtain $\begin{pmatrix}u\\v\end{pmatrix}$-Coord via norm. i.e. $\left(\frac{u}{w}, \frac{v}{w}, 1\right)$

↳ Perspective Projection: (arbitrary frame)



Transformation from world coord. frame to camera frame

$[P]_c = \begin{pmatrix}-r_1-\\-r_2-\\-r_3-\end{pmatrix}\left[\begin{pmatrix}x\\y\\z\end{pmatrix} - \begin{pmatrix}c_1\\c_2\\c_3\end{pmatrix}\right]$

R^T... Projects relative vector on new basis vectors — (Relative Vector $\vec{CP}$)

Cartesian: $[P]_c = R^T(P-C)$

Homogenous: $[P]_c = (R^T \mid -R^T C)\tilde{P}_{3OH}$ (Translation)

So: $\begin{pmatrix}u\\v\\w\end{pmatrix}_{20H} = \begin{pmatrix}f&0&0\\0&f&0\\0&0&1\end{pmatrix} R^T(P-C)$

$R_{x,y}$... # Pixel per unit length
$\begin{pmatrix}x_0\\y_0\end{pmatrix}$... Pixel coord of principal point

↳ Transform to Pixel Coord.:

$\begin{pmatrix}x_i\\y_i\\w\end{pmatrix}_{20H} = \begin{pmatrix}R_x & s & x_0\\0 & R_y & y_0\\0&0&1\end{pmatrix}\begin{pmatrix}u\\v\\w\end{pmatrix}$

So: $\begin{pmatrix}x_i\\y_i\end{pmatrix} = \begin{pmatrix}R_x & s & x_0\\0 & R_y & y_0\end{pmatrix}\begin{pmatrix}f&0&0\\0&f&0\\0&0&1\end{pmatrix} R^T(P-C) = KR^T(P-C)$, K... "Calibration mat."

Well, mathematically speaking this is "shift equivariance" ⚠

Sig.-Sys.-Recap:

$S[\cdot]$... "Functional"

$f(x) \to \boxed{S[f]} \to g(x)$

↳ LSI - Systems:

- Linear: $S[\alpha f_1 + \beta f_2] = \alpha S[f_1] + \beta S[f_2]$
- Shift-invariance: $f'(x) = f(x+\Delta x)$
  $S[f'] = S[f](x+\Delta x)$

Complex Exp(·) = "Eigenfunc."
Proof: $f(x) = e^{i\Omega x}$
$f'(x) = e^{i\Omega(x+\Delta x)} = e^{i\Omega \Delta x} e^{i\Omega x}$

$S[f'] = \begin{cases} L \to e^{i\Omega\Delta x} \cdot S[f](x) \\ SI \to S[f](x+\Delta x) \end{cases}$

So: $S[f](x+\Delta x) = S[f](x) \cdot e^{i\Omega\Delta x}$
for all x & Δx → Set $x = 0$

$S[f](\Delta x) = S[f](0) \cdot e^{i\Omega\Delta x}$ (Eigenvalue)

Rename $\Delta x = x$:

$\boxed{S[e^{i\Omega x}] = S[e^{i\Omega x}](0) \cdot e^{i\Omega x}}$

$\mathcal{F}[f](\Omega)$... "Contribution to signal f of the complex exp. with wave vector Ω"

- Arbitrary Signal f:
  Decompose f in sum of complex exponentials using Fourier-Trafo.

$f(x) = \frac{1}{2\pi}\int_{-\infty}^{\infty} \mathcal{F}[f](\Omega) \cdot e^{i\Omega x}\, d\Omega$

By linearity: Each $e^{i\Omega x}$ is transformed separately in FD

$\boxed{\mathcal{F}[S[f]]_{(\Omega)} = S[e^{i\Omega x}]_{(0)} \cdot \mathcal{F}[f]_{(\Omega)}}$

Proof:
$S[f] = S\left[\frac{1}{2\pi}\int_{-\infty}^{\infty}\mathcal{F}[f]_{(\Omega)} e^{i\Omega x}\, d\Omega\right]$
$= \frac{1}{2\pi}\int_{-\infty}^{\infty}\mathcal{F}[f]_{(\Omega)}\, S[e^{i\Omega x}]_{(0)} e^{i\Omega x}\, d\Omega$
$= \mathcal{F}^{-1}\left[\mathcal{F}[f]_{(\Omega)} \cdot S[e^{i\Omega x}]_{(0)}\right]$
$\Rightarrow \mathcal{F}[S[f]] = \mathcal{F}[f] \cdot S[e^{i\Omega x}]_{(0)}$

"Point-Spread Function" (= Image of a point)

- In spatial domain: Convolution

$\mathcal{F}[S[f]] = \mathcal{F}[f] \cdot S[e^{i\Omega x}]_{(0)} \Leftrightarrow S[f] = f_{(x)} * \mathcal{F}^{-1}[S[e^{i\Omega x}]_{(0)}]$

- Point-Spread-Func.:

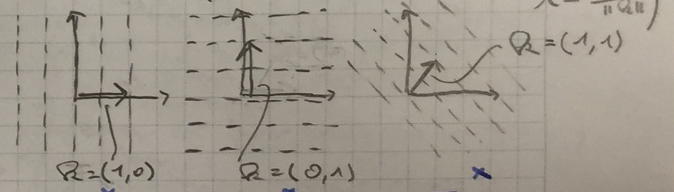
$\boxed{S[\delta] = \mathcal{F}^{-1}[S[e^{i\Omega x}]_{(0)}]}$

Proof: Use $\mathcal{F}[\delta(x-x_0)] = e^{-i\Omega x_0}$

↳ Fourier Transform: — ORTHONORM BASIS

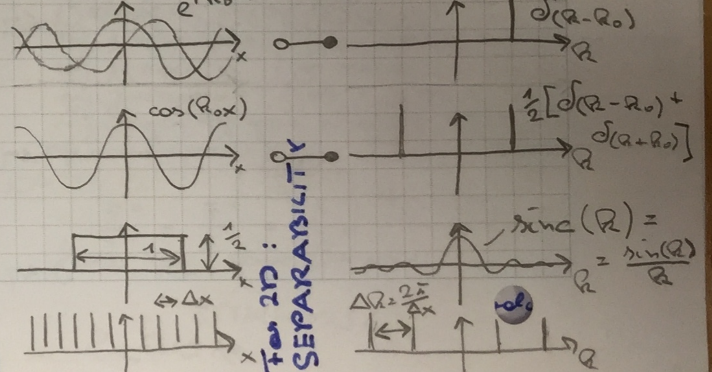
- Goal: Represent $f(x)$ as sum of complex exponentials
- $f(x) = \frac{1}{2\pi}\int_{-\infty}^{\infty}\mathcal{F}(\Omega) \cdot e^{i\Omega x}\, d\Omega$
- It can be shown:
  $\mathcal{F}(\Omega) = \mathcal{F}[f] = \frac{1}{2\pi}\int_{-\infty}^{\infty} f(x) \cdot e^{-i\Omega x}\, dx$
- Properties:
  $\mathcal{F}(\Omega) = \mathcal{F}_R(\Omega) + i\,\mathcal{F}_I(\Omega)$
  $\mathcal{F}_R(\Omega)$ ... even $\mathcal{F}_R(-\Omega) = \mathcal{F}_R(\Omega)$
  $\mathcal{F}_I(\Omega)$ ... odd $\mathcal{F}_I(-\Omega) = -\mathcal{F}_I(\Omega)$

In 2D: (Wave length of planar waves: $\lambda = \frac{2\pi}{\|\Omega\|}$)



$f(x) = \hat{\mathcal{F}}_{(1,0)} + \hat{\mathcal{F}}_{(0,1)} + \hat{\mathcal{F}}_{(1,1)} + \dots$

↳ Important Fourier Pairs: (See Medical Imaging)



For 2D: SEPARABILITY

REMEMBER: (ALWAYS)

Resolution in Space <=> Extent in FD (BW)
Extent in Space (FOV) <=> Resolution in FD

Similar in k-Space: When discretizing: $\frac{2\pi}{\Delta k} \geq FOV$

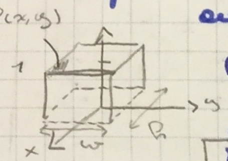
Since $p(x,y)$ is even: Real -> No phase shift; "Low Pass" (phase reversal); NOT: radially symmetric, but even!

Remember "photon buckets" of CCD/CMOS

Sampling:

(We have no model for blooming)

↳ Step 1: Integrate brightness over cell window



$g(x,y) = i(x,y) * p(x,y)$

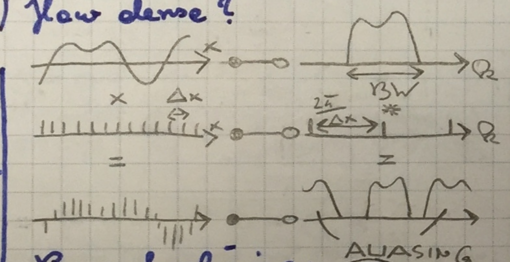
Meaning:
Box function is the "Point-Spread-Func." of our camera model!

In Fourier Domain: $F(k_x,k_y) = I \cdot P$

$P(k_x,k_y) = sinc(k_x) \cdot sinc(k_y)$ (with some scaling)

↳ Step 2: Read out value at pixel center!

i) How dense?



Prevent aliasing: $\frac{2\pi}{\Delta x} > BW$ (Nyquist)

ii) Which range?
- In space: FOV
- In k-space: According to resolution!

Multiplication in FD with box => => Conv. in space with sinc(·)

Reconstruction:

(Bottom right image above -> Apply box -> Sinc is perfect interpolant)

↳ From above: Only possible if
- Signal is band limited
- Sampled above Nyquist limit
- Signal is periodic
- Sampling compatible with signal period.

Never for infinite extent in space

Sinc would need infinite extent -> Often use Gauss! as function or dispersion

↳ Artifacts (from sampling)
- Aliasing (High frequencies are "folded back")
- Leakage (From choosing a FOV)
  In space: Mult. with Box -> Mixing frequencies!

Quantization:

↳ "We store discrete values of brightness"

Preprocessing - Denoising:

↳ Recall Fourier Properties:
"Power Spectrum": $\Phi = \|F(I(x,y))\|^2$

↳ Low Frequ.: Flat regions (Φ big)

↳ High Frequ.: Edges (Φ small)

↳ F("White noise") = "Uniform"
(White noise contains all frequ. equal)

=> SNR high at low frequ.

↳ LINEAR Filters: (Low pass)
Idea: Damp regions with bad SNR

① Ideal Low-Pass:
Multiply in FD with box.
Problem: Ripples (Conv. with sinc)
Unsharp (Edges gone)
=> We do not want sign reversal in spatial domain

② Average-Filter:
Design a convolution mask in spatial domain, s.t. there are no sign reversals. E.g.: Average
Problem: In FD this gives something like $sin(k_x) \cdot sin(k_y)$
=> Not really low pass
("Instead of damping high frequ there is over-/undershooting")

From ① + ② => No sign reversals in either domain!

③ Binomial Filter $\begin{pmatrix}1\\1\end{pmatrix} \otimes \begin{pmatrix}1\\1\end{pmatrix} = \begin{pmatrix}1&2&1\\2&4&2\\1&2&1\end{pmatrix}$ (Rapid, discrete approx of ...)
Does not have sign reversal in either domain. For size -> ∞: -> Gaussian!

low-pass

Problem: Linear filters ALWAYS blur
Why? S[img + noise] = S[img] + S[noise]

↳ NON-LINEAR Filters:

① Naive: Median
Good -> "Odd man out" & Preserve discont.
Bad -> Destroys "detail" (Ausbuchtung)

② Anisotropic Diffusion Filter:
Diffusion Equation: (= "Heat eq.") $\frac{\partial}{\partial t} g = \frac{\partial}{\partial x}(c \frac{\partial}{\partial x} g)$
Solution: Transform into FD (space)

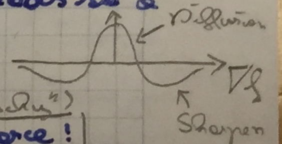
$$\Rightarrow g(t) = g(t=0) * Gauss(\sigma = ct)$$

This solution has to assume c > 0, otherwise problems with Fourier Trafo.
If c < 0 -> Put negative sign into time variable: Run diffusion in reverse time direction (= Sharpen)

Idea: Make sign(c) dependent on ∇g -> For small $|\nabla g|$: Diffusion
For large $|\nabla g|$: Sharpen

E.g.: $c(\nabla g) = exp(-|\nabla g|^2)$

Use chain rule to derive the "new" diffusion equation -> leads to a diff.-coeff. like this:



Improve performance even more (Otherwise: "Patchy")
=> Restraining force!

Preprocessing - Deblurring:

① Unsharp Masking: -> Perfect for Gaussian Blur
Consider blurred image g_b as outcome of the diffusion equation applied to original image g_0.
Replace time derivative with Euler-backw.

$$\frac{g_b - g_0}{\Delta t} = c \nabla^2 g_b$$

$$\Rightarrow g_0 = g_b - c\Delta t \cdot \nabla^2 g_b$$

In practice: used "well-chosen" constant

∇^2 ... LSI-Operator (+ isotropic)
Can be implemented as convolution!
BUT sensitive to noise (2nd-order derivative)
-> Combine with Gaussian -> LoG ~ DoG

If we estimate SNR low (= we have low confidence in the signal) ⟹ Becomes low-pass (conservative)

Usually blurring is a low-pass → Inverse amplifies high freq.!

② **Inverse Filter:** Assume original img was blurred by $f_b = b * f_0$
$\Rightarrow F_b = B F_0$

Original img can be restored by choosing a filter $\boxed{1/B}$

Problem: • Ill-defined for $B=0$
• Huge noise amplification

③ **Wiener Filter:**
Goal: Design a LSI-Filter
We feed the filter with some signal (e.g. blurred img) + noise.
Output is given by convolution:

$\boxed{i = b + n \quad , \quad o = h' * i}$ ← input
(some signal, noise; output, PSF)

Denote: d ... "Desired signal"
Optimize h' (least-squares-sense)

$\boxed{h' = \arg\min \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} (o - d)^2 \, dx\, dy}$

It can be shown:

$\boxed{\mathcal{F}(h') = H' = \frac{\Phi_{id}}{\Phi_{ii}}}$ (Problem: unknown)

where power spectra are FT of cross correlations:

$\Phi_{id} = \mathcal{F}(\int_{-\infty}^{\infty}\int_{-\infty}^{\infty} i(x'-x, y'-y) d(x', y') \, dx' dy')$

Example - Deblurring:
We feed filter with blurred img + noise: $i = h * b + n$ – noise (blurring PSF, orig img = d)
Assume noise and blur are uncorrelated → $\Phi_{nb} = \Phi_{bn} = 0$

$\Phi_{ii} = H^2 \Phi_{BB} + \Phi_{nn}$ → $H' = \frac{H \Phi_{BB}}{H^2 \Phi_{BB} + \Phi_{nn}}$
$\Phi_{id} = H \Phi_{BB}$

Or: $\boxed{H' = \frac{H}{H^2 + 1/SNR}}$, $SNR = \frac{\Phi_{BB}}{\Phi_{nn}}$ (signal-noise-ratio)

Need to know: ⚠
• H... Blurring kernel
• SNR → Estimate according to confidence! (Want conservative: Set SNR low)

Difference to convolution: MIRRORED

Properties:
• $H = 0 \to H' = 0$ (No noise)
• $SNR = \infty \to H' = \frac{1}{H}$ (Only noise)
• $SNR = 0 \to H' = 0$

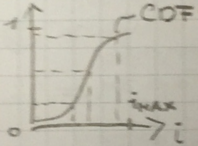
Estimate ⚠

Blurring: Loss of contrast over edges
→ Unsharp masking is kind of contrast enhancer too

Preprocessing - Contrast Enhancement:

Problem: Not whole intensity domain is used, but only some few
Idea: Redistribute intensities, s.t. intensity histogram is flat!
→ All intensities used equally often

So we have an arbitr. distr. $p(i)$ and want to map it to a uniform distribution! → "Inversion Method"



Choose $\boxed{I' = i_{MAX} \cdot CDF(I)}$
New intensities

"Where CDF is flat: $p(i)$ will generate few samples, but since CDF is flat → wide range is mapped to small I' range!"

The math: (Assume continuity)

$I \sim p(i)$, $F_0(I \le i) = \int_0^i p(i')\, di'$
Distribution function of I':
$F_N(I' \le i) = F_N(I \le F_0^{-1}(\frac{i}{i_{MAX}})) =$
$= \int_0^{F_0^{-1}(\frac{i}{i_{MAX}})} p(i')\, di'$

Rule for inverse: $(f^{-1})' = \frac{1}{(f' \circ f^{-1})}$

$\frac{d}{di} F_N(i) = p(F_0^{-1}(\frac{i}{i_{MAX}})) \cdot \frac{d}{di} F_0^{-1}(\frac{i}{i_{MAX}})$
$= p(F_0^{-1}(\frac{i}{i_{MAX}})) \cdot \frac{1}{p(F_0^{-1}(\frac{i}{i_{MAX}}))} \cdot \frac{1}{i_{MAX}}$
$= \frac{1}{i_{MAX}}$ □

Edge Detection:

① Large gradient magnitude

Detect edges by computing $\|\nabla f\|$ (= isotropic) Problem: $\|\cdot\| \neq$ linear
→ But ∂_x / ∂_y = linear

"Sobel-Mask": Finite-Difference-App. + Binomial smoothing
First FD, then smooth

$\frac{\partial}{\partial x} = \begin{pmatrix} 1 \\ 2 \\ 1 \end{pmatrix} (-1 \;\; 0 \;\; 1) = \begin{pmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{pmatrix}$

AD SOBEL:
"High pass" in one direction
"Low pass" in the other

Note: Sobel mask will generate "thick edges"

② Zero Crossings of Laplacian:

Idea: Edges lie at intensity inflection
$\Delta f = 0$ (... "∇f has no sources on edges")
Implementation: "Stencil"
Problem: 2nd order derivatives (= Prone to noise)
⟹ Always combine with Gaussian Smooth
Leads to $L * G * f$ = Laplace of Gauss (LoG)
LoG can be approximated with a DoG (= Difference of Gaussian)
⟹ Leads to 1 pixel thick edges
But: Too many (also flat regions have $\Delta f = 0$)
\# Finding zero crossings

③ Matched Filter: (Let's optimize!)

Define Edge model $m(x)$
Find the optimal linear filter as
$h = \arg\max \{SNR \cdot LOC\}$, $LOC = \frac{1}{E[(x_0 - x_c)^2]}$

Result: $\boxed{h(x) = m(x)}$ (Expected value since we deal with random noise; Edge location $x_c = 0$)

Problem: • Still noisy
• Many local maxima in $h * f$

"Max. Filter"

④ Canny Edge-Detector: ← "State of the art"

Theory: Additional constraint
$h = \arg\max \{SNR \cdot LOC + c \frac{1}{N_{MAX}}\}$ (weight; Average # of local max)

Result: c small: "matched filter"
$\boxed{c \text{ large: Approaches 1st derivative of a Gaussian}}$

E.g.: Sobel mask is kind of Canny (Binomial ⟺ Gaussian, $m = (-1 \;\; 0 \;\; 1)$)

How to implement in 2D???
• Start with Gaussian blur
• Compute Gradient mag. + dir.
• Non-maximum suppression } Post-Proc.
• Hysteresis Thresholding }

OUTPUT: Still "thick" edge

How to compute Gradient?
E.g. Sobel mask → Problem: Discrete pixel grid
Use FD-Approx in 4 dir.
→ Choose maximum as gradient

T_H set too low: Many noise edges
T_L set too high: Edges break up
Result: Thin edges!

ad ④ Canny Edge-Detector:

Non-maximum suppression:
Goal: Generate thin edges
Idea: For each pixel we have now $\|\nabla f\|$ and $\measuredangle \|\nabla f\|$
Compare current pixel with neighbor values in gradient direction (i.e. orthogonal to edge = direction in which edge is smeared).
If neighbor has bigger $\|\nabla f\|$
→ Suppress (= Set value to zero)
Else: Preserve (= Do nothing)

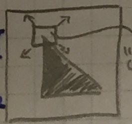
Hysteresis Thresholding:
Goal: Get rid of edges coming from noise!
Idea: Noise-edges will have low $\|\nabla f\|$
Define two thresholds $T_L < T_H$
Step over pixels → 3 cases:
- $\|\nabla f\| > T_H$: Keep value (= Class 1)
- $\|\nabla f\| < T_L$: Set to zero
- $T_L < \|\nabla f\| < T_H$: Keep only if connected to Class 1 pixels

Canny fails at: Crossings / Corners
(Simplified edge model is wrong → non-max suppression is bad)

"Harris" Corner Detector:

Goal: Distinguish
- Flat Regions
- Edges
- Corners

Idea: Look at intensity variations
Changes of intensity upon variation of window



Taylor: $I_{(\vec{x}+\Delta\vec{x})} = I_{(x,y)} + I_x\Delta x + I_y\Delta y$

$$E_{(\Delta x,\Delta y)} = \sum_{x,y} \omega_{(x,y)} \cdot \left[I_{(x,y)} - I_{(x+\Delta x)}\right]^2$$
$$= \sum_{x,y\in\omega} \left[I_x\Delta x + I_y\Delta y\right]^2$$
$$= \sum_{x,y\in\omega} (\Delta x\ \Delta y)\begin{pmatrix} I_xI_x & I_xI_y \\ I_xI_y & I_yI_y\end{pmatrix}\begin{pmatrix}\Delta x\\ \Delta y\end{pmatrix}$$
$$= (\Delta x\ \Delta y)\left[\sum_{\omega}\begin{pmatrix} I_xI_x & I_xI_y \\ I_xI_y & I_yI_y\end{pmatrix}\right]\begin{pmatrix}\Delta x\\ \Delta y\end{pmatrix}$$

Structure Matrix

Same Procedure applies as in "PCA - Long Answer"

Structure Matrix = "Quadratic Form"
→ We can understand this matrix and what it does by looking at EV/EW

- λ_1 & λ_2 ~ small: Flat
- λ_1 ~ small, λ_2 ~ large: Edge
- λ_1 & λ_2 ~ large: Corner

→ Remember this diag. eq. ?!

Don't want to compute λ_1, λ_2, since computational heavy → rather use

$$R = \det H - k(\operatorname{trace} H)^2$$

- $R \ll 0$: Edge
- $R \gg 0$: Corner
- $|R|$ ~ small: Flat

Feature Matching:

(Basis for: Registration, Tracking, Recognition)

Given 2 images, we wanna perform matching / find corresponding parts

Difficulties:
- Different Viewpoints
- Different Illumination
- Different Background
- Occlusions

"Global features used with binary images"

⇒ Feature: "Describe one local character." (↳ Occlusion)

Idea:
(i) Find "Interest Points" using a DETECTOR (e.g. corners, blobs, ...)
(ii) Each interest point is assigned some characteristic of its surrounding by a DESCRIPTOR. (It gets a "feature vector".)
(iii) Use some measure in "feature space" to compare two images in a matching step

The "surrounding" is specified by the PATCH ←

Consider 3 types of transformation between viewpoints:
- Similarity (4 DOF)
- Affine (6 DOF)
- Projective (8 DOF)

INVARIANCE: feature(Trafo(img)) = feature(img)
COVARIANCE: patch(Trafo(img)) = Trafo(patch(img))

⚠

We want:
- Invariance of features
- Covariance of patch / interest pt.

Note: Should also take into account photometric Trafos → work with gradients

↳ Invariants (for geom. trafos)
Similarity → Ratio of lengths/angles
Affinity → Ratio of areas
Projectivity → "Cross Ratio"

Since patches are small → Affinity or even Similarity will do

↳ Patch Selection & Feature Description (examples)

① Edge Corners + Affine Moments:

(i) Obtain corners + edges
(ii) Create parallelograms ("infinitely" many): Start at a corner. Two sides of the parallelogram are obtained by evolving vectors along the edges, s.t. the ratio of areas between vector & edge is const.
(iii) Select one parallelogram of this family [illegible] based on some invariant (e.g. solution of some optimization problem)
} COVARIANT PATCH
iv) Create an invariant feature to describe this patch, e.g. using a combination of "moments": $M_{pq}^{abc} = \iint x^p y^q R^a \cdot G^b \cdot B^c dA$
} INVARIANT FEATURE

② Intensity Extrema + Affine Moments:

(i) Search intensity extrema (interest point)
(ii) Shoot rays away from interest points (in 2D plane)
(iii) Evaluate some invariant func. along these rays
(iv) Search maxima → gives contour
(v) fit ellipse (& double in size)
} Patch
(vi) Describe patch with "affine moments" as above.
} Feature

Note on "affine moments" M_{pq}^{abc}:

One needs to build combinations of them. Can get very complex!
Correct combinations are invariant under affine geometric trafo & photometric trafo in form of scaling the channels

Maximally Stable Extremal Regions (= Blob Detector)

③ MSER + Affine Moments

Covariant Point Detector

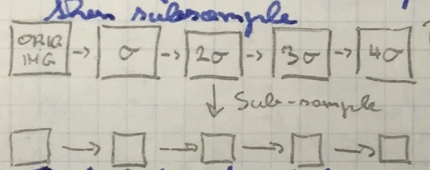
Invariant Features

(i) Start with intensity extremum
(ii) Create series of thresholded img
(iii) Take regions where growth is slowest (→ edges)
(iv) Same as before: Combination of M_{pq}^{abc}
→ Scale invariant feature transform

④ SIFT: (Detector + Descriptor)

SIFT is based on invariants under similarity trafos (works surprisingly well)

1. Create a "Scale Space":
Take original image & subsample
Due to aliasing → always low-pass when subsampling
So we first blur multiple times, then subsample



2. Find interest points:
This is done using the "mexican hat filter", i.e. Laplace of Gaussian. But we use the approximation: Difference of Gauss. This is cheap, as we already have all blurred images! From one octave with 5 img create 4 DoG (Do for all octaves)
→ Compute min/max (over 3 blurred img: 4 → 2)
→ Reject those interest points with low contrast
3. Assign orientation:
Look at region around interest pt.
Create histogram of gradient orientation → pick maximum
4. Create feature vector:
16×16 grid around interest pt.
→ Divide in 16 4×4 grids
In each 4×4 grid create gradient histogram: (*)

(*): For all pixels in 4×4 grid compute gradient directions.
Put in a histogram with 8 classes 0° - 45°, 45° - 90°, ...
Weight histogram with distance to interest pt.
→ Collect 16·8 = 128 numbers (= feature)

Image Decomposition:

↳ Scale Space: (Was used in SIFT)
Idea: Scene has multiple levels of detail
→ Increase efficiency by starting work on lower resolution.
(Can early prick a region of interest)
Implementation: ↦ Nyquist
Before downsampling → Low Pass
Assure no artifacts from downsampling → Only Gauss Blur!
On the fly: DoG-approx of LoG (Zero-crossings ⇒ edges)

↳ Decomposition using different Bases: (= Unitary Trafo of img) i.e. pixel-wise decomp

- Interpret images as large vectors
- Using Dirac impulses as basis (= 2D Shah-Fund) (⇔) "Standard Basis"
- Since a new basis is (by convention) chosen orthonormal, transition between bases is a rotation !!!
  ⇒ Rotation corresponds to unitary operators
- A set of basis images B_{uv} is:
  - ORTHONORMAL: $\sum_x \sum_y B_{ij} B_{ij}^* = \delta_{ij}$
  - COMPLETE: (We deal with those)
    Any other img. can be represented!
    Damn hard on Hilbert Spaces
    Easy on finite Spaces → Reduces to finding enough orthog. img.
  - SEPARABLE: (only nice to have)
    $B_{ij} = \varphi_i \otimes \varphi_j$ → Outer Products
    Allows very fast implementations!
- Given a basis - how to represent img?
  $f_{(x,y)} = \sum_u \sum_v F_{(u,v)} B_{uv}$ → Backward
  $F_{(u,v)} = \sum_x \sum_y f_{(x,y)} B_{uv}^*$ → Forward

Backward: "Linear combination of basis"
Forward: "To obtain $F_{(u,v)}$ project f onto B_{uv}" ⇒ Analog Vector

Proof for Backward-Trafo:
$f = \sum_u \sum_v F_{(u,v)} B_{uv}$
Fourier-Trick: Multiply with $B_{u'v'}^*$ and sum ("Project $F_{(u,v)}$ out")

$$\sum_x \sum_y f B_{u'v'}^* = \sum_x \sum_y \sum_u \sum_v F B_{uv} B_{u'v'}^*$$
$$\sum_x \sum_y f B_{u'v'}^* = \sum_u \sum_v \sum_x \sum_y F B_{uv} B_{u'v'}^*$$
$$\sum_x \sum_y f B_{u'v'}^* = \sum_u \sum_v F_{(u,v)} \delta_{(u-u',v-v')}$$
$$\boxed{\sum_x \sum_y f B_{u'v'}^* = F_{(u',v')}}$$

- Optimal Truncation Property
Problem: We only keep a subset of weights $F_{(u,v)}$
Question: Can we reduce "truncation error" by somehow changing "surviving weights"?
Answer: No! Leave surviving weights unscaled
Why?
You would be trying to get information about the image, that is stored along vectors that are completely independent (orthogonal) of available basis vectors → Dumb idea!
Additional:
By adding more weights: Truncation error always decreases!

↳ Examples: ⓪ Dirac-Basis: (Standard Basis) Perfect localization in Space

① Fourier Transform (DFT)
Works with odd extension → Edge introduced
Basis func: $e^{i\varphi x}$ → Real cos(·)
→ Imag sin(·)
Odd extension → Introduces high frequencies

② Cosine Transform (DCT)
Works with even extension
→ No high frequencies introduced!
Purely real / Separable → Chips av.

③ Haar Trafo:
"Better localization balance"
For high frequ.: spatial extent gets smaller

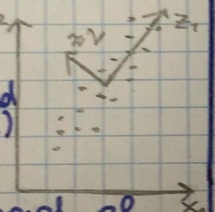
Principal Component Analysis

Image Decomposition - PCA:

- So far: DFT/DCT/... are examples of image independent transforms, i.e. rotation of basis vectors always the same!
- Q: Is there an "optimal basis" for a given class of img.?
- A: YES! PCA for img, called KLT (Karhunen-Loève-T.)
- Define "optimal basis": Such that we capture most inter-class variation with the least amount of basis vecs.
- Definition: "Principal Comp. (PC)" k-th PC is the direction in image space capturing the k-th max. variance & orthogonal to all other PC

(Here "image space" means the vector space containing the image vectors, which contain the pixel values!)

- 2D - Example: (Else: Shift)
  - → Assume data centered in origin (not shown)
  - → 1st PC = z_1 (z_1... New basis vec.)
  - → E.g.: Compression: Store only z_1 - coord. of each data (+ direction of z_1)



- How to find PC in high dimension of image space?
- → Short answer: Solve Eigenvalue-Problem for the data's covariance matrix C
- → Long answer: (SAX: $x \in \mathbb{R}^p$)
  - Img. is vector of pixel intens.
  - Assume every pixel (= every coord.) is a random variable
  - Joint distribution (charact. for class of img, e.g. faces) is $P(x)$... unknown (Image data: $x_1, \dots, x_n$ iid copies $x \sim P(x)$)
  - Assume covariance matrix known! $C = cov(x) \in \mathbb{R}^{p \times p}$

$C = E[(x-\bar{x}_n) \otimes (x - \bar{x}_n)] = E[x \otimes x]$

  - Find PC as the direction $d \cdot x$, s.t. $var(d \cdot x) = max$ (Some linear combination of x)

- → Long answer (continued):
  - Optimization Problem:

$$d = \arg\max_{\|d\|=1} \{ var(d x) \}$$

(Need constraint $\|d\| = 1$, otherwise no finite max)

  - Bilinearity: $var(dx) = cov(dx, dx) = d^T C d$ ("Quadratic Term")
  - Lagrange Multipliers: $\nabla(d^T C d) = \lambda \nabla (d^T d - 1)$ ($d^T C d$... mon. increasing in d → max. found on boundary)

$$C d = \lambda d \rightarrow (C - \lambda I) d = 0$$

  - d... Eigenvector of C: $var(d x) = d^T C d = d^T \lambda d = \lambda$
  - k-th PC is eigenvector of C with k-th maximum eigenvalue!?! SOOO BEAUTIFUL *_*
- Notes:
  - For uncorrelated training img: PCA/KLT → DCT
  - How to obtain image statistics C? Collect image data vectors as columns of matrix $X \in \mathbb{R}^{p \times n}$
    - p... (Img width) × (Img height)
    - n... Number of sample images
    - (i) $n > p$: $C = \frac{1}{n} X X^T$... good estimator
    - (ii) $n < p$: C has rank drop
      - Data lies in subspace (Hyperplane of $\mathbb{R}^p$)
      - Instead find EV of $S = X^T X$, $S \in \mathbb{R}^{n \times n}$ (Also solves computational complexity)
      - If c... eigenvector of S → Xc... eigenvector of C
  - Recall: Variance can be interpreted geometrically as inertia! 100% analogy of PCA to finding "principle axes" of moment of inertia tensor.

Image Decomposition - ICA:

- Main difference to PCA: NOT a unitary Trafo → not only rotation, but a general linear Trafo of the basis → Some math tools are mis read...

Segmentation - Thresholding (= "Identifying entities in an image")

- Input: Image with high foreground-background contrast
- Output: Binary image (i.e. 2 classes)
- General Idea: Look for threshold that separates peaks in histogram
- How to find good threshold? E.g. Otsu criterion:
  - We want large areas with low variance
  - Minimize: $p_1 \sigma_1^2 + p_2 \sigma_2^2$ (Group Size, Within-group-Var())
- Post-Processing Binary Img.
  - (1) Noise reduction - Morphological Operation (Non-Linear)
    - Shift mask over image
    - Look at rank ordered values within mask: $i_{(1)} \leq i_{(2)} \leq \dots \leq i_{(n)}$
    - Assign some value to center i_c:
      - Reduce obj → $i_c = i_{(1)}$ → Erosion
      - Increase obj → $i_c = i_{(n)}$ → Dilation
      - $i_c = i_{(\frac{n+1}{2})}$ → Median
    - Powerful in combination, e.g. "closing" (1st. dilation, 2nd erosion) (Fill gaps → / ← Recorrect size)
  - (2) What if multiple obj are in img?
    - "Connected component analysis" (can make 2 class binary img → multiple class img) (each class has one lable)
    - Define neighborhood of a pixel
    - Do for pixels marked as obj.: (BG has no lable)
      - → Check if neighborhood has lable
      - → No lable found: Start new
      - → 1 lable found: Copy
      - → >1 lable found: "Note equivalence"
- Pros:
  - Simple for further processing (Band-width reduced)
  - Real-time hardware available!
- Cons:
  - Typically no good segmentation
  - Pixel by pixel decision...

In contrast to thresholding: Use spatial information

Segmentation - Edge based:

- Given: Pixels that could belong to an edge (e.g. after Sobel)
- Goal: Find a linked list, that describes thin / connected lines!

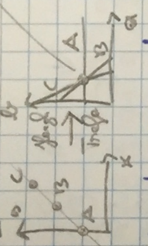
① "Canny edge detector":

- Yields "thin" edges
- Problem: Very often not connected
  Why: Failure due to the "simplified edge" model!

② Hough Transform:

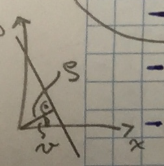
NOTE: HT & HT is image space again!

- Idea: Have parametric model of edge available!
  - → Step over "possible edges" and vote for all param. combinations, s.t. the edge would contain the current pixel
  - → Gives a profile in parameter space
  - → Peak detection!



All points lie on line specified by this point in parameter space

- Example: Straight lines
  - Possible Parameterization $y = ax + b$
  - ⚠ But this is no good! Since $a \in [0; \infty)$ ⚠
  - Better Parameterization $x \cos\vartheta + y \sin\vartheta = \rho$
  - Each point adds sinusoidal in parameter space



- Notes:
  - "Peak Detection" non-trivial
  - Needs a-priori knowledge of shape

③ Snakes - "Active contour model": (= closed line)

Intuition: "Rubber Band"

- Given initial contour (near desired object → Evolve over time to fit object boundary!
- How: Define energy function to be minimized!

In discrete case: "Set of points" (x_i, y_i) — $s \in [0;1]$ "Arc length"

- The details:
- Let contour be given as $\vec{v}(s) = \begin{pmatrix} x(s) \\ y(s) \end{pmatrix}$
- Solution of: $v(s) = \arg\min (E_{EXT} + E_{INT})$
- External energy: "Punish for distance to edges"

$$E_{EXT} = -\int_0^1 \|\nabla I\|^2 \, ds = -\int_0^1 (G_x^2(v(s)) + G_y^2(v(s)))$$
$$\approx -\sum_i G_x^2(x_i, y_i) + G_y^2(x_i, y_i)$$

- Internal energy: "Punish for non-smoothness / stretch / deviation from shape prior / ..."
  Common choice: (linear) deformation energy

$$E_{INT} = \int_0^1 \left(\alpha \left|\frac{\partial v}{\partial s}\right|^2 + \beta \left|\frac{\partial^2 v}{\partial s^2}\right|^2\right) ds$$
$$\approx \sum_i \left(\alpha \|v_{i+1} - v_i\|^2 + \beta \|v_{i+1} - 2v_i + v_{i-1}\|^2\right)$$

(Penalize stretch ↑, Penalize bending ↑)

- How to do the optimization?
  (i) Variational Calculus:
  Functional to be minimized
  $F = \frac{1}{2}\int_0^1 (\alpha \|v_s\|^2 + \beta \|v_{ss}\|^2) - \|\nabla I\|^2 ds$
  → Gateaux-Derivative → Find Euler-Lagrange Equations:
  $-\alpha v_{ss} + \beta v_{ssss} = -\nabla \|\nabla I\|^2$
  Solve using FDM-approximation
  (ii) Greedy Search:
  For each point of contour place a "search grid" & pick pixel where minimized
  → Iterative Procedure
  → Convergence not guaranteed
  (iii) "Dynamic Programming" — No clue what that is ☹

- Pros:
  + "Energy" can be customized, e.g.
  → Stretch punishment makes contour want to go to a single point! Instead punish for deviation from given length $\sum \alpha (\|v_{i+1} - v_i\|)^2 \to \sum \alpha (\|v_{i+1} - v_i\| - \alpha)^2$
  → Define shape prior
  Punish for deviation from "expected shape"
  + Contour always connected!
- Cons:
  - E_{INT} critical
  - Local minima

Somehow inverse approach to edge detection → We try to detect homogeneities

Segmentation - Region based:

- Detect homogenous regions
- Place random seeds → grow regions until "homogeneity criterion" is broken
- Criterion: Non trivial

Segmentation - Statistical Pattern Recognition:

- A sneak overview of ML: (Terminology is not well defined)

SUPERVISED → Classification (→ Generative, Discriminative), Regression
UNSUPERVISED → Dimension Reduction (e.g. PCA), Clustering (Hierarchical, Partitional, Bayesian)

- Supervised:
  Find $f: X \to Y$ on the basis of "examples" / "training data" (= pairs $(x_i, f(x_i) = y_i)$)
  Pipeline:
  Training Data $X_1 \to Y_1$, $X_2 \to Y_2$, ... —Learning→ Model $f(x)$ —Prediction→ Test Data $X_1' \to ?$, $X_2' \to ?$ ...
- Unsupervised:
  "Learning without training data"
  Goals: Compact data, Identify "latent variables"
- Clustering: Organization of unlabeled data into "similar groups" (aka "cluster")
  (*) - Hierarchical: Find successive clusters using previously established clusters (bottom-up OR top-down)
  - Partitional: Find all cluster at once
  - Bayesian: Generate a "Posterior distr."

Also: Classifiers that are not based on a prob. model are called discriminative

- Generative Classification:
  Classifiers based on a model for the joint probability $P(X, Y)$ — These models are learnt from the examples!
- Discriminative Classification:
  Classifiers based on a model for the conditional probability $P(Y | X = x)$ (*)

Note: In a segmentation problem Y takes a finite number of values → Discrete RV. Nevertheless we write integrals! (Think of them being wrt counting measure ^^, i.e. they are sums)

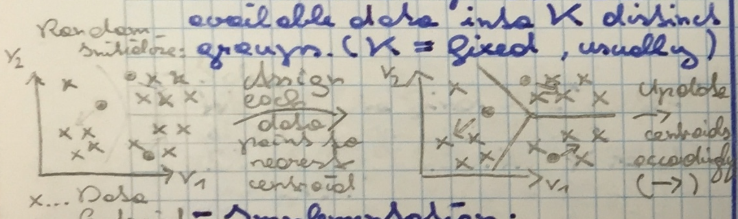
Markov Random Fields

Segmentation - Statistical Pattern Recognition (cont'd):

- The following are **VERY** brief descriptions of how we can use ML for segmentation! (We really just scratch the surface here):

① K-Means Algorithm:
(Unsupervised partitional clustering)

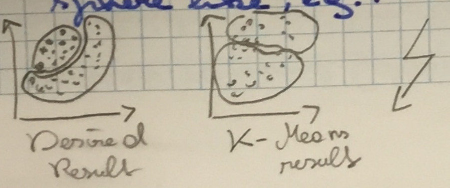
- We don't have training data!
- We want to separate the available data into K distinct groups. (K = fixed, usually)



- Implementation:
  (1) Choose k random data points as initial centroids (= cluster center). Left pic
  (2) Assign each data point to a group by seeking the closest centroid. Right Pic
  (3) Re-compute centroids using the current cluster memberships
  (4) Repeat until converged
- Note:
  → In the example data is specified by a 2D feature vector. In the case of an image → Could choose to use every pixel as data point & its feature vector is its intensity (Similar to thresholding)

  Only if we use every pixel as data point will we get a "classification" for every pixel

  → Sensitive to outliers
  → Cluster will always look sphere like, e.g.:



② Naïve Generative Model:

- Assume now X, Y are correlated RVs
  (X... Input data / feature vector
  Y... Class labels)
- Goal: Find the "map" $h: X \to Y$ (In probabilistic models called "hypothesis h")

  Remember: Decision Theory with squared loss!
- Assume joint density $P(X,Y)$ known
- It can be shown: → Here without proof ^^
  The optimal hypothesis is

  $$h_{opt} = E[Y|X=x] = \int y P(Y|X) dy$$

  (Optimal here means: This hypothesis minimizes the risk, based on square-loss)
- Therefore: We want $P(Y|X=x)$ (called: POSTERIOR)
- Then we could find the optimal hypothesis by doing the integral
- Problem: Integral = hard
- Solution: A reasonable choice is to use the maximum a posterior (MAP)-estimator: $h_{MAP}(x) = \arg\max_y (P(Y|X=x))$
- How do we get the posterior?

  $$P(Y|X=x) = \frac{P(X|Y=y) \cdot P(Y)}{P(X)}$$

  POSTERIOR; likelihood (not 100% sure); PRIOR; Constant
- Naïve approach:
  - "Learn" the distribution $P(X|Y=y)$ from training data (kernel est.)
  - Set Prior = const (all classes equally likely)

  Then MAP reduces to MLE, that is $h_{MAP} = h_{MLE} = \arg\max(P(X|Y=y))$
- Problem: Choosing $P(Y)$ = const means our hypothesis only considers the feature of the current pixel!
  → Basically we do fancy threshold!

How to make connection to my statistics course? ($Y \Leftrightarrow \vartheta$, i.e. we wanna know the class)
→ $X \sim P \in \{P_\vartheta : \vartheta \in \Theta\}$ → "classes"

③ Generative Model - MRF:

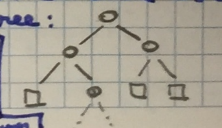
- Main difference: We now choose the PRIOR in a clever way!
- Choose $P(y)$ depending in some "weird way on the "neighborhood"
- Basically no clue what is going on here
- In the end: There is a model for $P(y)$, hence we can compute the joint dist. $P(X,Y)$... that's why it is a generative model!

④ Discriminative Model - KNN:

- Recall: Discriminative model means, our goal is to directly get the posterior. Or even more aggressive directly the maximum!
- Overall goal of discriminative methods: Estimate model parameters, s.t. the error, when predicting the "training data", is minimized!
  → Possible to compute error for training data, since the "ground truth" is provided!
- Procedure of K-nearest neighbors (KNN)
  - Keep all training data in memory
  - During segmentation: Find the K nearest training data points (in feature space)
  - Choose label on basis of the labels of the training data (e.g. majority voting)
- Pro: → Easy to implement/understand
- Con: → Depends on K (e.g. for 2 class segmentation, always use odd K)
  → Keep all training data in memory

⑤ Discriminative Model - Random Forest:

- Forest: "Ensemble of decision trees"
- One tree:



Different Trees: focus on diff. features

- At each internal node: Simple decision (left or right?)
- At each leaf node: Prediction

Both learned! → HOW? DUNNO

(*) Functional: "Function of functions"

Intermezzo: Variational Calculus

- We will consider 3 problems. Each problem tries to find extrema of some functional! (*)

(1) Find f, s.t. $I(f) = \int_{x_1}^{x_2} F(x, f, f')$ with $f(x_1), f(x_2)$ fixed is extreme.

Margin notes: Why? Compare to optimizing a function $f(x_0) \to$ In optimum f' is indep. of x / That is we consider Dirichlet BC

$$\frac{d}{d\varepsilon} I(f + \varepsilon\eta) \overset{!}{=} 0 \quad \forall \eta(x) \text{ "admissible"}, \quad \eta(x_1) = \eta(x_2) = 0$$

So called "Gateaux derivative". Here:

$$\frac{d}{d\varepsilon} I(f + \varepsilon\eta) = \int_{x_1}^{x_2} \frac{d}{d\varepsilon} F(x, f + \varepsilon\eta, f' + \varepsilon\eta')$$

$$= \int_{x_1}^{x_2} \eta \cdot F_f + \eta' \cdot F_{f'} \, dx$$

Wanna get rid of η' -> Int. by parts:

$$\int_{x_1}^{x_2} \eta' \cdot F_{f'} = \eta \cdot F_{f'} \Big|_{x_1}^{x_2} - \int_{x_1}^{x_2} \eta \cdot \frac{\partial}{\partial x} F_{f'}$$

(the boundary term is 0)

Plug this into Gateaux-D.:

$$0 = \int_{x_1}^{x_2} \eta(x) \left(F_f - \frac{\partial}{\partial x} F_{f'}\right) dx$$

Has to hold for all admissible test function $\eta(x)$!

Therefore: $F_f - \frac{\partial}{\partial x} F_{f'} = 0$ — Euler-Lagrange Equation!!!

(2) What happens if we have dependence on another function:

$$I(f) \to I(f_1, f_2) = \int_{x_1}^{x_2} F(x, f_1, f_1', f_2, f_2')$$

Now we want the Gateaux-Derivative in both "variables" (i.e. f_1 & f_2) to vanish!

Margin note: Again we consider the Dirichlet BC, $f_1(x_1), f_1(x_2) \dots$ fixed

Gives 2 equations:

(i) $\frac{d}{d\varepsilon} I(f_1 + \varepsilon\eta, f_2) \overset{!}{=} 0$

(ii) $\frac{d}{d\varepsilon} I(f_1, f_2 + \varepsilon\nu) \overset{!}{=} 0$

For $\eta(x), \nu(x)$ "admissible" i.e. $\eta(x_1) = \eta(x_2) = \nu(x_1) = \nu(x_2) = 0$

Let's see how those derivatives look:

$$\frac{d}{d\varepsilon} I(f_1 + \varepsilon\eta, f_2) = \int_{x_1}^{x_2} \frac{d}{d\varepsilon} F(x, f_1 + \varepsilon\eta, f_1' + \varepsilon\eta', f_2, f_2')$$

$$= \int_{x_1}^{x_2} \eta F_{f_1} + \eta' F_{f_1'} \, dx$$

Exactly the same as in (1), so we know the resulting Euler-Lagrange! (The same happens with (ii)):

(i) $F_{f_1} - \frac{\partial}{\partial x} F_{f_1'} = 0$

(ii) $F_{f_2} - \frac{\partial}{\partial x} F_{f_2'} = 0$

— Euler-Lagrange

(3) What happens if the function now is 2D?

$$I(f(x)) \to I(f(x,y)) = \iint_D F(x, y, f, f_x, f_y)$$

We only have a single Gateaux-D. ($I(\cdot)$ has only one argument)

$$\frac{d}{d\varepsilon} I(f + \varepsilon\eta(x,y)) \overset{!}{=} 0$$

We still consider Dirichlet BC, so $f|_{\partial D}$ "fixed" $\Rightarrow \eta|_{\partial D} = 0$

Also assume: Domain D... closed & simply connected (s.t. Gauss Theorem holds)

$$\frac{d}{d\varepsilon} I(f + \varepsilon\eta) = \iint_D \frac{d}{d\varepsilon} F(x, y, f + \varepsilon\eta, f_x + \varepsilon\eta_x, f_y + \varepsilon\eta_y)$$

$$= \iint_D (\eta F_f + \eta_x F_{f_x} + \eta_y F_{f_y}) \, dy \, dx$$

As above: Wanna get rid of η_x, η_y! Use Gauss Theorem! -> Need divergence

$$\eta_x F_{f_x} + \eta_y F_{f_y} = \nabla \cdot \begin{pmatrix} \eta F_{f_x} \\ \eta F_{f_y} \end{pmatrix} - \left(\eta \frac{\partial}{\partial x} F_{f_x} + \eta \frac{\partial}{\partial y} F_{f_y}\right)$$

$$\iint_D \nabla \cdot \begin{pmatrix} \eta F_{f_x} \\ \eta F_{f_y} \end{pmatrix} dA = \oint_{\partial D} \begin{pmatrix} \eta F_{f_x} \\ \eta F_{f_y} \end{pmatrix} \cdot d\vec{A} = 0$$

It follows: $\iint \eta_x F_{f_x} + \eta_y F_{f_y} \, dA = -\iint \eta \left[\frac{\partial}{\partial x} F_{f_x} + \frac{\partial}{\partial y} F_{f_y}\right]$

$$\iint_D \eta \left[F_f - \frac{\partial}{\partial x} F_{f_x} - \frac{\partial}{\partial y} F_{f_y}\right] dA = 0$$

$$\Rightarrow F_f - \frac{\partial}{\partial x} F_{f_x} - \frac{\partial}{\partial y} F_{f_y} = 0$$

NOTES - OF:

- Info "spreads" from corners (because of smoothness)
- Errors often at boundaries (— " —)
- Confusion with untextured obj. / Changing lighting

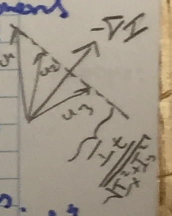
Motion Extraction – Optical Flow:

- Applications:
  - "Change" – Detection
  - Motion compensation
  - Feature tracking, ...
- Optical Flow – Definition: "Projection of 3D motion vectors onto the image plane"
- Idea: Take Lagrangian viewpoint of mechanics: (in some sense at least) → Better: Projected point. We glue ourselves onto a "pixel". As we move along with the pixel, the intensity stays constant! (Only true approx.)

$$\frac{d}{dt} I(x, y, t) = 0 = I_x u + I_y v + I_t$$

↑ Change when following a physical point; ↑ Change when looking at same pixel

- This is the optical flow constraint, rewritten: $\begin{pmatrix} I_x \\ I_y \end{pmatrix} \cdot \begin{pmatrix} u \\ v \end{pmatrix} = -I_t$
- Aperture Problem: Only specifies velocity component along the image gradient!



- We have 2 unknowns & 1 equation → What kind of black magic should we apply to solve this problem??? Answer: Regularization
- Introduce artificial "smoothness constraint"
- Measure for smoothness error:

$$e_s = \iint (u_x^2 + u_y^2 + v_x^2 + v_y^2) \, dx \, dy$$

- Measure for error in optical flow constraint

$$e_c = \iint (I_x u + I_y v + I_t)^2 \, dx \, dy \, !$$

- Minimize: $e_s + \lambda e_c$ (λ large... favour constr.; λ small... favour smooth.)

This is a functional of 2 functions in 2D! Using previous results for the functional

$$I = \iint F(x, y, u, u_x, u_y, v, v_x, v_y) \, dx \, dy$$

(i) $F_u - \frac{\partial}{\partial x} F_{u_x} - \frac{\partial}{\partial y} F_{u_y} = 0$

(ii) $F_v - \frac{\partial}{\partial x} F_{v_x} - \frac{\partial}{\partial y} F_{v_y} = 0$

Solve using FDM

Euler-Lagrange for optical flow!

NOTE: $Z \propto \frac{1}{\text{disparity}}$
→ If Z is large, i.e. disparity is small, then depth difference can no longer be seen! (Human stereo vision works up to 10m)

3D-Acquisition - Overview:

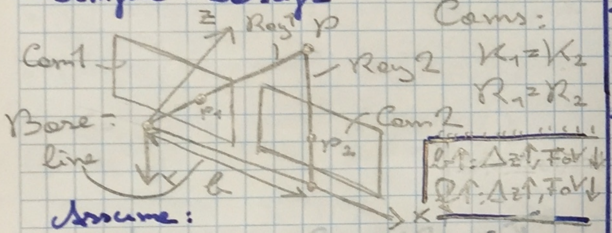
- ACTIVE
  - Uni-Direct.
    - Time Of Flight (TOF)
  - Multi-Direct.
    - Line Scanning
    - Structured Light
- PASSIVE
  - Uni-Direct.
    - Shape from texture, contour,
  - Multi-Direct.
    - Stereo

RECALL: Pinhole-Proj.-Model
$p = KR^T(P - C)$
K... Internal
R & C ... Ext.

3D-Acquisition - Stereo:

• Idea: "Triangulation", i.e. Given the projections of P onto two img.-planes (p_1 & p_2) → Reconstruct the projection lines (Ray_1 & Ray_2) and find their intersection!

• Simple - Setup



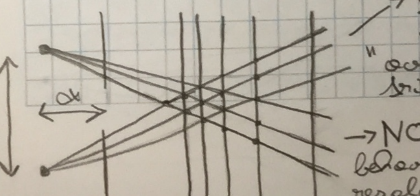
Identical Cams: $K_1 = K_2$, $R_1 = R_2$

Assume:
$$K = \begin{pmatrix} f & 0 & 0 \\ 0 & f & 0 \\ 0 & 0 & 1 \end{pmatrix} \quad R_1 = R_2 = \mathbb{I}, \; C_1 = (0,0,0), \; C_2 = (b,0,0)$$

$p_i = \begin{pmatrix} x_i \\ y_i \end{pmatrix}$

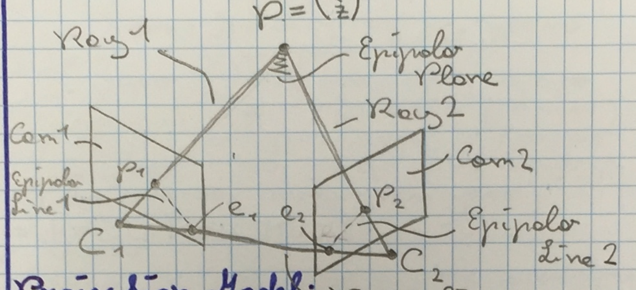
Then the projections are:
$$\begin{pmatrix} x_1 \\ y_1 \end{pmatrix} = \frac{f}{Z}\begin{pmatrix} X \\ Y \end{pmatrix}, \quad \begin{pmatrix} x_2 \\ y_2 \end{pmatrix} = \frac{f}{Z}\begin{pmatrix} X - b \\ Y \end{pmatrix}$$

Solve (by subtracting $x_1 - x_2$):
$$X = \frac{b}{x_1 - x_2} x_1, \quad Y = \frac{b}{x_1 - x_2} y_2, \quad Z = \frac{b}{x_1 - x_2} f$$
$(x_1 - x_2)$ = "Disp."



→ Stereo only possible in "overlapping triangle" (FOV)
→ NON-LINEAR behaviour of depth resolution Δz as function of disparity

• General Setup:
Cam1: K_1 (internal), R_1 & C_1 (external)
Cam2: K_2 (internal), R_2 & C_2 (external)



Projection Model:
$$p_1 = K_1 R_1^T (P - C_1)$$
$$p_2 = K_2 R_2^T (P - C_2)$$
→ p_1 & p_2 are in 2D-H coord.! Cartesian → "Norm."

From this we find the Ray equation:
Ray 1: $P(t) = C_1 + t_1 \cdot R_1 K_1^{-1} p_1, \; t_1 \in \mathbb{R}$
Ray 2: $P(t) = C_2 + t_2 \cdot R_2 K_2^{-1} p_2, \; t_2 \in \mathbb{R}$

Find P as intersection of those!
Problem: Noise/Discretization → Rays might not intersect!
Solution: E.g. find P such that "reprojection error" is minimal

HARD Problem → How to find corresponding p_1/p_2

Epipolar Geometry can help!
Idea: Given the projection p_1, we know → P can only be along Ray 1 → Projection p_2 can only be along the projection of Ray 1 (aka "epipolar line")

The math: Plug eqn. for Ray 1 into projection eqn. for Cam 2
$$p_2 = K_2 R_2^T (C_1 + t R_1 K_1^{-1} p_1 - C_2)$$
$$p_2 = t \cdot \underbrace{K_2 R_2^T R_1 K_1^{-1} p_1}_{} + \underbrace{K_2 R_2^T (C_1 - C_2)}_{}$$
$K_2 \cdot$ "Direction of Ray 1 as seen from Cam 2" = "Vanishing Point of Ray 1"
Projection of center of Cam 1 = "Epipole" e_2

Defining the homography: $A = K_2 R_2^T R_1 K_1^{-1}$
We found: $\boxed{p_2 = e_2 + t\, A p_1}$
Epipole! / Vanishing Point!

This epipolar constraint expresses that p_2 lies on a line segment between "epipole" & "vanishing point"

Rewrite constraint:
$(e_2 \times A p_1) \perp$ "Epipolar Plane"
Since p_2 lies within the epipolar plane, we have: $p_2^T (e_2 \times A p_1) = 0$
$\vec{e_2} \times \vec{a}$

We don't like cross-product → $= e_2[x] \cdot \vec{a}$
Write as the matrix product with $e_2[x]$

Hence: $\boxed{p_2^T e_{2[x]} A p_1 = p_2^T F p_1 = 0}$

F... "Fundamental matrix" (rank(F) = 2)
F works in both directions!
→ $p_2^T F$ ... line in 1st image
→ $F p_1$ ... line in 2nd image

• Notes:
- F can be computed from the knowledge of 8 correspond.
- Often: Use RANSAC
  (i) Have a lot of data points.
  (ii) Choose the minimum amount of data to be able to solve the problem (here 8) ← RANDOM
  iii) Compute parameters
  iv) Check how many data points obey the computed param.! (i.e. compute number_of_outliers)
  v) If fraction of outliers is below some threshold
  → Recompute using all inliers
  → Else: Start from beginning
- Remark on $e_2[x]$:
$$e_2 \times a = \begin{pmatrix} e_x \\ e_y \\ e_z \end{pmatrix} \times \begin{pmatrix} a \\ b \\ c \end{pmatrix} = \begin{pmatrix} e_y c - e_z b \\ e_z a - e_x c \\ e_x b - e_y a \end{pmatrix}$$
$$\Rightarrow e_2[x] = \begin{pmatrix} 0 & -e_z & e_y \\ e_z & 0 & -e_x \\ -e_y & e_x & 0 \end{pmatrix}$$

NOTE - "Vanishing Points": Using homogeneous coordinates one can define the intersection point of 2 parallel lines (point at infinity.) $x_\infty = (\cdot,\cdot,\cdot,0)$ → FACT: Under perspective projection this point will no longer be at infinity! This projection is called vanishing point!

3D - Acquisition - Live Scanning:

Solution to correspondence problem:
Replace 1 of the stereo cams with "projective device" (e.g. Laser)
Laser's "ray" is known & points on cam easily detected, i.e. cam's "ray" easily constructed!
Simply intersect "rays" :)

Specific Object Recognition:

- Goal: Recognize an instance of a class: "My car" not "a car"
- Challenges: (Usual candidates)
  Illumination / Viewpoints changes, occlusion, clutter

① Model based approach:
(= local geom. features)

- Model ("features") of obj. stored in database!
- At recognition: Compare features of query image to features in database
- Critical: ⚠ FEATURES ⚠
  (i) "Hand made" (by designer):
  e.g. Hole, arc, wire frame ...
  ↯: Hypothesis + Verification for both: Type + Pose
  → SLOW
  (ii) Invariant features:
  Decouple "Type + Pose"!
  → Leads to Hybrid methods

② Appearance based approach:
(= global image representation)

- "Model" = "Image of obj itself"
- But compressed img. → PCA
- Training:
  → Sample set of viewing cond.
  → Apply PCA to every single img.
  → Keep dominant PC
  → Sampled viewing cond. will form a manifold in PCA-Space

Example: (*)

(*) Sample changes of viewing points in "azimuth" (→ real obj's turn table) & keep first 3 PCs. Then the manifold will be a curve in 3D-Space!

Encodes object type!

- Recognition:
  → Find "nearest manifold" to test img.
  → Once manifold is found → Find nearest points in that manifold → Gives Pose :)

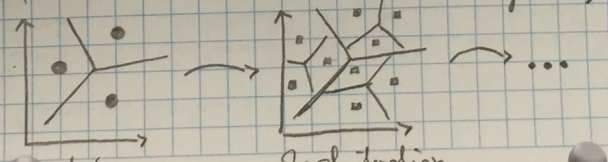
Comparison:

Model	Appearance
Compact model (but hard to get)	Large model (but easy to get)
SLOW	EFFICIENT
Cluster ✓	Cluster ✗

③ Hybrid methods
(= local image representation)

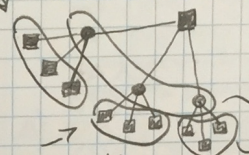
- "Model" = "Local features"
- Recall from FEATURE MATCHING:
  (i) DETECTOR finds interest points
  (ii) DESCRIPTOR extracts feature vector (i.e. describes surrounding)
  (iii) MATCHING: Find closest obj. in database in "feature space"
- Note: Choose invariant features
  → need less img. in data base
- Already looked at (i) & (ii), now look at (iii), the actual recognition step
- FAST MATCHING:
  (Hierarchical vocabulary tree)
  - Goal: Need to do some nearest neighbor search in feature space (for many features)
  - Idea: Build accelerated data structure!
  - How: Hierarchical K-Means clustering (of feature space)

FAVOR Method to build Kd-Tree (with arbitrary planes)



Vocabulary Tree

- Once feature space is hierarchically clustered → Build Tree:
  - Tree Nodes: Cluster Centers
  - Tree Leaves: Cluster Centers of lowest level
  = "Group of similar features"
  = "Visual words"



Here K-Means ran 4 times reaching depth of 2 (with K = 3)

- "Visual Words":
  Group of similar features!
  Lowest level cluster centers!
  Sometimes they correspond to parts (e.g. wheel)
- Inverted File Index:
  Each "visual word" = "leaf of vocabulary tree" = "lowest level cluster center" stores a reference to the images, which contain that particular visual word.
- At recognition step:
  - Extract features from query img.
  - Send them down the vocabulary tree & find visual words the img. contains → Find the query images based on "bag of words"
  - For each visual word:
    Give a vote to all img. that are referenced by that visual word
  - Object is recognized as the one with the most votes!

- ROBUST MATCHING:

For specific object recognition it makes sense to look for spatial consistency of features!

Can hypothesize on a "map" between two images & see how many features agree!

→ Use RANSAC.

Loss = ∞

clean")

Object Category Recognition:

- Goal: Recognize not only "my car", but "any car"
- Challenges: Usual candidates + "Intra-class variation"
- Recognition has two components:
  (i) Classification (Is there a car?)
  (ii) Detection (Where is the car?)

I: CLASSIFICATION:

- Idea: Intra-class variation may be large on global scale, but typically small on local scale → Objects of same category should be made up of "similar visual words"

→ Same approach as with specific object recognition:
BOF: Don't check on spatial consistency!

- When working with visual words:
  Need to create "vocabulary"
  (i) Extract features from training data (many faces...)
  (ii) Cluster the feature space, i.e. find the visual words!
- Question: Size of vocabulary=?
  ⚠ Specific Obj.Rec.: Large vocab.
  ⚠ Obj. category Rec.: Small vocab.
- Once we decided on a vocabulary we represent a category by its

> Histogram of word occurence
> =
> Bag of Words (BoW)

- Why is this cool?
  → BoW is a vector of fixed size (= size of vocab.)!
  No matter how many features there are detected
  → Removes spatial dependence
  (This is both: Strength & Weakness)

- What if we want spatial info?
  → Spatial Pyramids: BoW on regular subdomains of image → then concatenate vectors

II: DETECTION:

- Idea: Sliding Window-Approach
  Slide window over all different size over img. & solve a classification problem at each location (= Brute Force)
- Main requirement for this to work: FAST feature extraction & classification
- Examples for FAST features:
  Ex. HAAR-FILTER
  (i) Compute gradient image
  (ii) Divide image into cells
  (iii) For each cell: Orientation Histogram
  (iv) Concatenate all Histograms
  - Rectangular Integral image filter (Remember Haar-Trafo?)
    Measure response to "Haar Filters"
- Prominent Example:
  Viola-Jones Face Detector:
  (i) Sliding window approach (24×24)
  (ii) Use Haar filters:
  Looks like nose → → Response to "nose bridge"
  Looks like eyes → → Response to "eye-cheek"
  Remember "matched filter"? Highest response to structure, that is similar to filter-structure!
  (iii) If response is above some threshold → Classify as face
  Each such filter is a "weak classifier"
  (iv) We have many Haar filters → many weak classifiers! Use the AdaBoost-Algorithm to find the important ones* and how to combine them optimally
  * + Weights
  (v) Once the important Haar-features + their thresholds + the optimal linear combination of them is known → we can speed up the classification by cascading!

FAST REJECTION if false!

(v) Cascading:
Split Haar Filters into groups!

Window contains face? → 1st 10 Haar f. → Maybe → 2nd 10 Haar f. → Maybe → ...
↓ No ↓ No

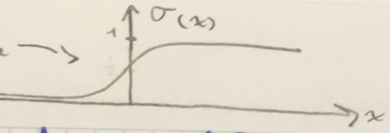
Note: Haar Filter can be evaluated very fast using "Integral img".

Notes on sliding window approach:
+ Easy to implement
+ Empirical success → Viola-Jones
- High computational expense
- Not all objects are box-shaped
- View windows in isolation → context lost

Deep Learning - Single layer Perceptron:

- Supervised / Discriminative Method
- Notation: $x \in \mathbb{R}^d$ ... "Input" / "Feature" (NN: Inputs)
  E.g. pixel intens. of b&w segmented
  $y \in \mathbb{R}^m$ ... "Output"
- Regression: y... Numeric Value
  Classification: y... Categorical
- Goal: Find the map $y = f_v(x)$
  where $f_{(\cdot,v)}$ is some predefined model with parameter v
- How? $\hat{v} = \arg\min_v \sum_{n=1}^{N} L(y_n, f_v(x_n))$
  → Find the best v that minimizes the "Loss" within a training set ("Ground truth") $(y_n, x_n)_{n=1,..,N}$
- Loss & Interpretation of map is task specific:
  → Regression: $f_{(x,v)}$... Numeric value
  E.g. when doing regression: "Price" "an Quality in question"
  $L_{(\cdot,\cdot)} = \| y_n - f_{(x_n,v)} \|_p$
  → Classification:
  Entries should sum to 1 → $f_{(x,v)}$... Vector of size m (m = # classes)
  Each entry is considered a probability!
  Prediction: Index in the class corresponding to largest entry
  If $p=1$: Loss = 0
  If $p=0$: Loss = ∞
  $L(y_n, f_{(x_n,v)}) = -\log(\text{"Prob. given to correct class"})$

One typical choice for the non-linearity (Sigmoid) →

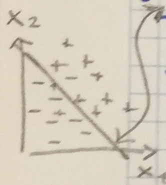


- How do we model $f(x, \vartheta)$? Assume Binary classification! We can choose $m = 1$:
  $f(x,\vartheta) = P(y=1) = 1 - P(y=0)$

$[x_1 \ldots x_d] \to P_{(y=1)}$

Single-Layer Perceptron:
$$f(x,\vartheta) = \sigma(\vartheta^T x + b) = \frac{1}{1+e^{-(\vartheta^T x + b)}}$$

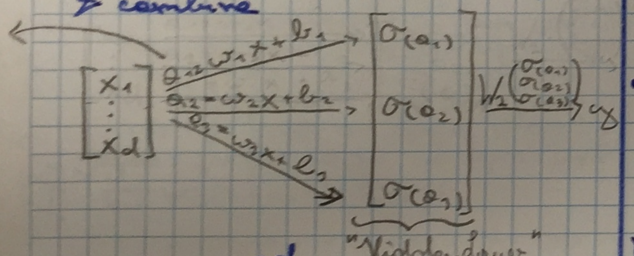
- Activation: $\vartheta^T x + b$, $\vartheta^T \in \mathbb{R}^d$ (= Affine Subspace = Hyperplane)
- Non-linearity: σ ("Activation") $\sigma(\cdot)$ … "Sigmoid" → Maps $a = \vartheta^T x + b$ to probabilities $[0; 1]$
- Decision Boundary: $f(x,\vartheta) = \frac{1}{2}$ for single-layer perceptron: will always be linear!
  Why? Put hyperplane in strictly increasing non-linearity and ask where const. → Same as asking where hyperplane is some other const → (hyper-) line



ACTIVATIONs
a_i … Different Hyper-planes

Deep Learning - Multi-Layer-Perc.: (MLP)

- How to get more complicated decision boundaries? Do multiple single layer perc. & combine

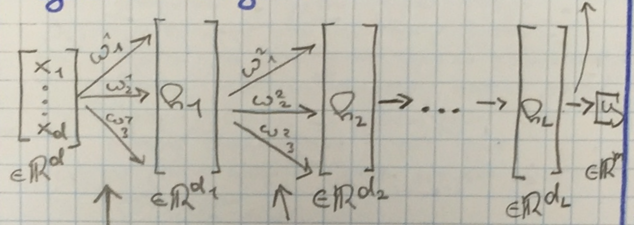


- Put $w_i \in \mathbb{R}^d$ into a matrix.
  $W_1 = [w_1 \ldots w_{d_1}]$ where d_1 = "Width of hidden layer"
- Non-linearity at the end: Basically just scaling
- Non-linearity in hidden layer: IMPORTANT ⚠ "linear model"
  Else we would only get a linear comb. of hyperplanes = Hyperplane!

$$a_{l,i} = w_i^l x + b_i^l$$
(i-th activation of layer l; activation, weights, bias)

For binary dec.: Only 1 final activation a_L

- General Layout:



$W_1 = (w_1^1 \; w_2^1 \ldots w_{d_1}^1)$ $W_2 = (w_1^2 \; w_2^2 \ldots)$

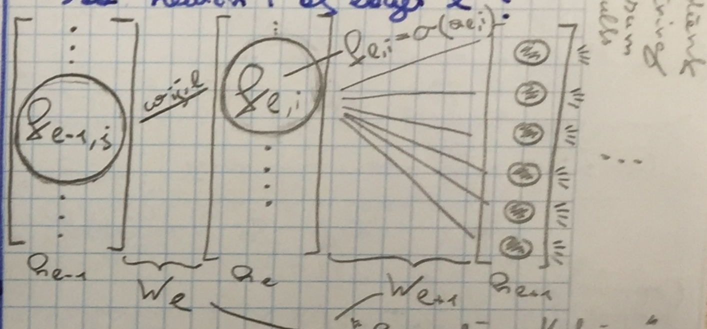
- Number of parameters only from $x \to h_1$:
  - $d \times d_1$ weights
  - d bias
  } $d \cdot d_1 + d$
- Notes:
  - MLP can do linear decision boundary
  - Width $(d_1, d_2, ..)$ & Depth (# hidden layer) = "design choice"
  - Hidden layers (up until h_{L-1}): "Automatic feature extraction"
  - $d_i > d$: Map to higher dimension (sometimes easier to separate classes in higher dim.)
  - $d_i < d$: "Compression", potential info loss
  - Last layer: Linear ("logistic") model (h_L) on extracted features!
- How to do more than 2 classes? Let y have higher dim. (More than one a_L)
  How to map to probabilities?
  "Soft max.-func.": $P_{(y=k)} = \frac{e^{a_{L,k}}}{\sum_i e^{a_{L,i}}}$

Deep Learning - MLP Training:

- 3 Sets:
  (i) Training Set: Estimate Param. (W_l, b)
  (ii) Validation Set: Estimate Hyper-Param. (i.e. Width & Depth)
  (iii) Test Set: Estimate Accuracy
- Overfit: Too big difference in Training & Test Set → Need more training data! (Or less parameter…)

RECALL:
$$\hat{\vartheta} = \arg\min_{\vartheta} \sum_n \mathcal{L}(y_n, f(x_n, \vartheta))$$
← Sum over training data (x_n, y_n)

- Goal of Training: Find parameter that minimize loss-func.
- Problem: $\mathcal{L}(\vartheta)$ … probably complicated optimization surface
- Nevertheless → Use (stupid) gradient descent:
$$\vartheta_i^{t+1} = \vartheta_i^t + \eta \frac{\partial \mathcal{L}}{\partial \vartheta_i} = \vartheta_i^t + \eta \sum_n \frac{\partial \mathcal{L}_n}{\partial \vartheta_i}$$
(Compute gradient for every training data point & sum results)
- How does this look for $w_{ij,l}$ - The connection from "neuron" j in layer $l-1$ to "neuron" i of layer l?



"Connection Matrices"

Chain Rule:
$$\frac{\partial}{\partial w_{ij,l}} \mathcal{L} = \frac{\partial \mathcal{L}}{\partial h_{l,i}} \cdot \frac{\partial h_{l,i}}{\partial a_{l,i}} \cdot \frac{\partial a_{l,i}}{\partial w_{ij,l}}$$
$= \sigma'(a_{l,i})$, $h_{l-1,j}$

What's that? $h_{l,i}$ is used as input to all activations $a_{l+1,k}$ of the following layer $(l+1)$!!! Need chain rule again:
$$\frac{\partial}{\partial h_{l,i}} \mathcal{L} = \sum_k \frac{\partial \mathcal{L}}{\partial h_{l+1,k}} \cdot \frac{\partial h_{l+1,k}}{\partial a_{l+1,k}} \cdot \frac{\partial a_{l+1,k}}{\partial h_{l,i}}$$
Sum over all neurons of layer $l+1$ — "AHA" o.O — $\sigma'(a_{l+1,k})$, $w_{ik,l+1}$

- Important Result: Gradient wrt weights in layer l depends on "stuff" from layer $l+1$!
- This is the reason for so-called Backpropagation!
- Often defined:
$$\delta_{l,i} = \frac{\partial \mathcal{L}}{\partial h_{l,i}} \cdot \sigma'(a_{l,i})$$
"Error signal"

When do you even use y_n? Of course only in $\mathcal{L}$ / $\nabla\mathcal{L}$! (E.g. clarification: Which vector comp. to put in the log(?))

- **Summary for Training MLP:**
  - → Weights: $\frac{\partial \mathcal{L}}{\partial w_{i,j,\ell}} = \delta_{\ell,i} \cdot z_{\ell-1,j}$
  - → Bias: $\frac{\partial \mathcal{L}}{\partial b_{i,\ell}} = \delta_{\ell,i}$
- **Backpropagation of Error Signal:**

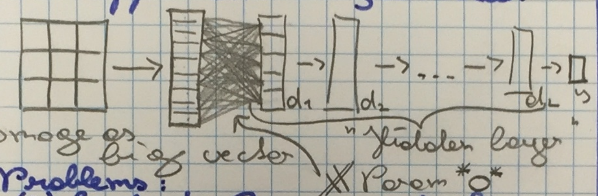
$$\delta_{\ell,i} = \left(\sum_k \delta_{\ell+1,k}\, w_{k,i,\ell+1}\right) f'(a_{\ell,i})$$

Sum over error signals $\delta_{\ell+1,k}$ in following layer & multiply with connection weights !!!

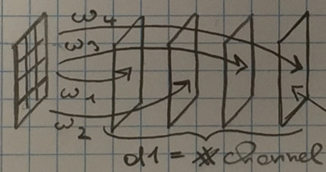
- **Implementation:**
  - (i) Put training example x_n in NN & compute outputs (feed-forward pass)
  - (ii) Having all activations available compute all error signals starting from the last layer (Backpropagation)
  - (iii) Using error signals, one can compute the gradients w.r.t. all weights
  - (iv) Do this for all training examples & sum gradients!
  - (v) Do the gradient descent iteration step
- **Notes:**
  - "Stochastic gradient descent" (SGD): Don't use all training data to compute gradients, but only a subset → update weights more often → FASTER
  - We use an iterative scheme to compute the weights → need some initial values! How? No matter what you do → Don't choose them const. Would lead to complete symmetry in network → "one neuron thick" network… Instead: Random initialization
- Main question after reading of material on NN: Why does this work? o.O

Deep Learning - Convolutional NNs:

- How to use MLP/NNs for image analysis? We need the input to be a vector!
- Naive approach: "fully connected"

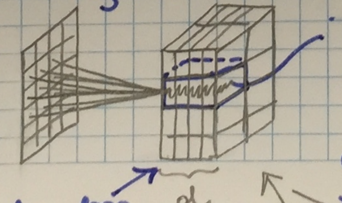


- Problems:
  - (i) Leads to huge amount of param. E.g. 64×64 img. & width of first layer $d_1 = 128$: # $\approx 0.5 \cdot 10^6$ (even though 64×64 → 128 severe compression?)
  - (ii) Images have "local statistics": What does it help to know that there's a street lantern in the background, if we want to identify a man/woman in the img.?
- **Solution: Convolutional Architecture**



Each activation is an image, obtained from convolution. This is a channel

- Collection of channels = Layer
- Non-linearity applied pixel-wise
- Only one convolution kernel for each channel: "weight sharing"
- Solves problem of many param.: Only need d_1 ($k_1 \times k_2$) ← Kernel size e.g. 3×3
- Can choose d_1 small, why?

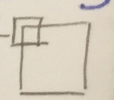


This vector of length d_1 gathers only local info! Obtained from multiple convolutions!

d_1 different kernels for d_1 diff. activations/channels. ← Tensor

Note: "Deeper Layer" gather more global information! → "Receptive Field"

- So convolutional architecture: "Larger layers with sparser connections"
- "Padding": Math. "valid" convolution: Reduces img. size. If want to keep exact size: "Pad" img. Use 0 when kernel outside domain

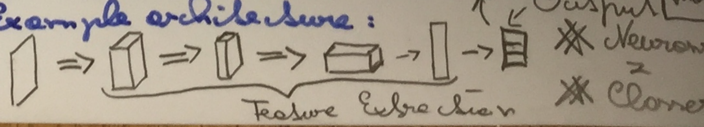


- **Example: CNN for Recognition**
  Goal: Given an image, classify the whole img. as face, car, …
  - → The output should be a vector with size = # classes
  - → There are 2 problems:
    - (i) Dimension Reduction: How to get from huge layers to this small vector
    - (ii) Translation invariance: Wanna identify an image as face no matter if it is in the upper left or lower right corner! ← $\mathcal{M}$ = "the face"

ad 1.: Dimension Reduction
→ Reduction through valid convolution: Slow reduction or huge masks (= many param.)
Very fast dim. reduction → "Stride": When doing convolution, shift mask not only one, but multiple pixels! Usually stride of 2 in all directions (half image size) → In both direct.
BUT: Info-Loss

ad 2.: Translation Invariance
Translation operation
→ Invariance: $f(T \circ img) = f(img)$
→ Equivariance: $f(T \circ img) = T \circ f(img)$ → for segmentation/localization
CNNs have native equivariance!
"Pooling": Remember morphological op.? Do the same here! Shift mask over image and represent whole domain by one value, e.g. Max-Pooling
Often + Stride. Gives partial translation invariance!

Example architecture:



• Example : CNN for Segmentation

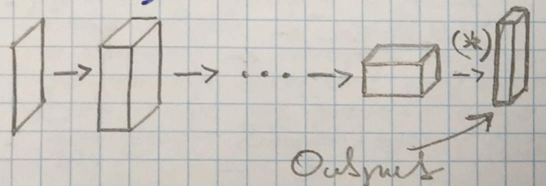
→ Output is not a vector of size = * classes, but it is an image!

→ Naive implementation: Pixelwise classification

Does work, not very good at boundaries, can be slow at test time!

(Shift mask and do classific. for center pixel!)

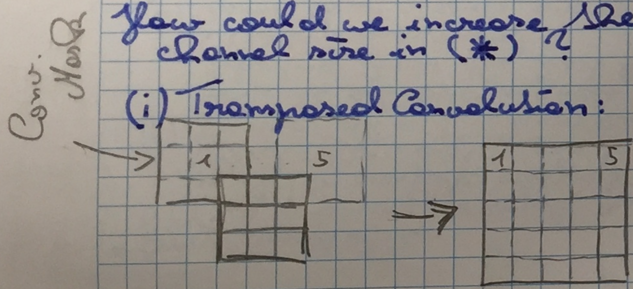
→ Fully convolutional Network



Output channels are of same size as input images.
There are as many output channels as classes (I think)
→ Do pixelwise soft-max

How could we increase the channel size in (*)?

(i) Transposed Convolution:



(ii) Upsampling + Convolution
Self explanatory

Two competing factors in segment.:

(i) Integrate large context:

Automatic through "Receptive Field"

(ii) Distinguishing neighbouring boundary pixels