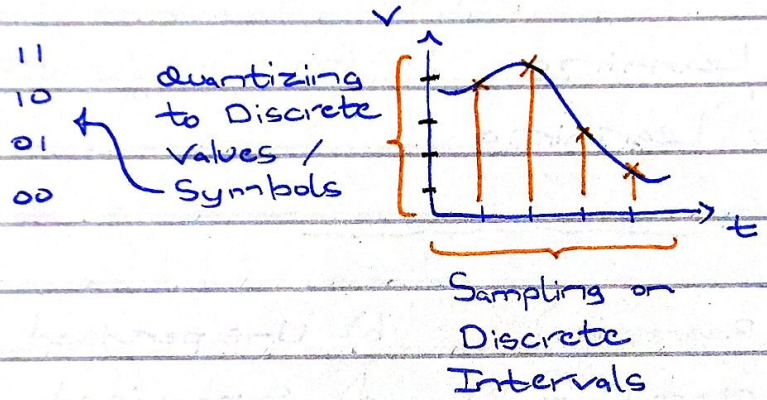
12 / 09 / 23

• Computer Vision (698125704)

→ Image : $(x, y, z) \rightarrow f(x, y)$
└ amplitude is intensity

↓

Digital Everything is discrete and quantized.



→ Number of bits to store an image :

$$b = M \times N \times k$$

$M \times N$: x & y pixels
k : bit depth

⇝ i.e. $2^8 = 256$ levels
{0 - 255}

→ Edges ⇝ where abrupt transitions occur

19 / 09 / 23

Computer Vision

$I_{Bin} \in [0, 1]$

$I_{Gray} \in [0, 256)$ or $[0, 255]$

I_{RGB} ~ combination of $\{R, G, B\} \in [0, 256)$

Histogram

↳ tells us the frequency of occurrence of pixel values

- increasing contrast results in even spread of histogram

~ Noise : $\hat{I}(x,y) = I(x,y) + n(x,y)$

↳ represents estimate true noise

$\therefore n(x) = e^{-x^2/2\sigma^2}$

Derivative

$$\frac{df}{dx} = \lim_{\Delta x \to 0} \frac{f(x) - f(x - \Delta x)}{\Delta x} = f'(x) = f_x$$

↳ for discrete nature, smallest $\Delta x = 1$

$$\frac{df}{dx} = \lim_{\Delta x \to 1} \frac{f(x) - f(x-1)}{\Delta x} = f(x) - f(x-1)$$

$$\frac{df}{dx} = f'(x) = \text{Backward Difference}$$

~ Backward Difference (1D) [-1 1]

$f(x) - f(x-1)$

~ Forward Difference (1D) [1 -1]

$f(x) - f(x+1)$

subtract

~ Central Difference (1D) [-1 0 1]

$f(x+1) - f(x-1)$

Derivatives in 2D

Gradient $\nabla f(x,y) = \begin{bmatrix} \partial f(x,y)/\partial x \\ \partial f(x,y)/\partial y \end{bmatrix} = \begin{bmatrix} f_x \\ f_y \end{bmatrix}$

magnitude $= \sqrt{f_x^2 + f_y^2}$

direction $= \tan^{-1}(f_y/f_x)$

$f_x = \frac{1}{3}\begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{bmatrix}$ $f_y = \frac{1}{3}\begin{bmatrix} 1 & 1 & 1 \\ 0 & 0 & 0 \\ -1 & -1 & -1 \end{bmatrix}$

averaging, extending idea of central difference

- Drag mask such that it fits I completely

Correlation

$f \otimes h = \sum_k \sum_l f(k,l)\, h(k,l)$ (f: image, h: kernel)

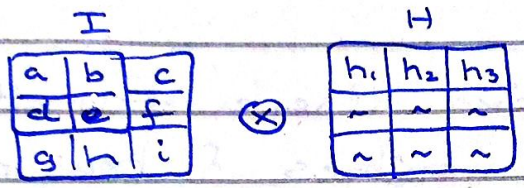
Rest from slides

20/09/23

Computer Vision

Filters

↳ Linear spatial filters perform sum-of-products



$R = (ah_1 + bh_2 + \dots) = I \otimes H$

if [e] is updated by R/n ; n is $3 \times 3 = 9$
its an averaging filter

Box Filter : all coefficients are equal
Weighted Average : give more (less) weight to pixels near (away) from center

- Gaussian Filter

1D : $g(x) = [0.011 \quad 0.13 \quad 0.6 \quad 1 \quad 0.6 \quad 0.13 \quad 0.011]$

- Non-Linear filters are useful when there are "spiked" noisy pixels
  - Min / Median / Max
    - ↳ works well for impulse noise

impulse

Edge Detection

↳ involves mathematical methods to find sharp transitions in images

- step ; roof ; ramp ; spike

↳ first derivative where transitions occur gives spikes { how much intensity was changed }

fails for noisy images → use filters

Gaussian Smoothing

$$g(x,y) = e^{-\frac{1}{2}\left(\frac{x^2+y^2}{2\sigma^2}\right)}$$

As σ is increased ;

- more pixels are used
- noise is more effectively suppressed
- more smoothing occurs

→ Edge Detectors

- Gradient operators
  - Prewit
  - Sobel
- Laplacian of Gaussian — 2nd order
- Gradient of Gaussian

26/09/23

Computer Vision

• Edge Detectors

→ Prewitt ; Sobel

G_x : Vertical edges
G_y : Horizontal edges

Prewitt:

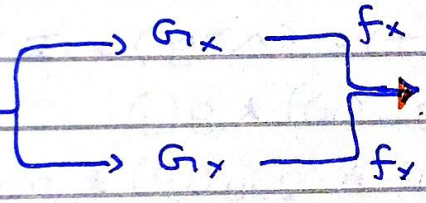
$$G_x = \begin{bmatrix} 1 & 0 & -1 \\ 1 & 0 & -1 \\ 1 & 0 & -1 \end{bmatrix} \qquad G_y = \begin{bmatrix} 1 & 1 & 1 \\ 0 & 0 & 0 \\ -1 & -1 & -1 \end{bmatrix}$$

Sobel:

$$G_x = \begin{bmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{bmatrix} \qquad G_y = \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}$$

central row and column have more weights

Pipeline I



→ Marr Hildreth

- → Smooth image by Gaussian filter → S
- → Apply Laplacian to S
- → Find zero crossings

$$\Delta^2 S = \Delta^2 (g * I) = (\Delta^2 g) * I,$$

where g is Gaussian filter

$$\Delta^2 g = -\frac{1}{\sqrt{2\pi}\sigma^3}\left(2 - \frac{x^2+y^2}{\sigma^2}\right) e^{-\frac{x^2+y^2}{2\sigma^2}}$$

{Check zero crossing from slides}

→ Take magnitude of zero crossing;
$\{a, -b\} \rightarrow |a+b|$
and compare it to user defined threshold τ

Separability of LoG

$h(x,y) = I(x,y) * g(x,y)$ n^2 mults
$h(x,y) = ((I(x,y) * g_1(x) * g_2(y))$ $2n$ mults

$\Delta^2 S = \Delta^2(g * I) = (\Delta^2 g) * I$ n^2 mults
$\Delta^2 S = [(I * g_{xx}(x)) * g(y) + (I * g_{yy}(y)) * g(x)]$ $4n$ mults

We don't X-flip and Y-flip kernels that are symmetric; No point

Quality of an Edge

Robust to noise
Localization
} From slides

27/9/23

Computer Vision

Line Fitting

Least squared error

→ $E = \sum_{i} (mx_i + c - y_i)^2$

↳ From slides

Disadvantages occur when multiple lines and outliers

RANSAC

Robust to outliers - Works by exhaustive randomly sampling data points until the best fit line.

Canny Edge Detector

- Good detection ~ Good localization
- Single response constraint

1. Smooth image with Gaussian filter
   $S = g * I$
2. Compute derivative of S
3. Find magnitude & orientation
4. Apply "Non-Max Suppression"
5. Apply "Hysteresis Threshold"

3/10/23

Computer Vision

- **Non-Max Suppression**

  In the direction of gradient, move along and suppress non-max values to be left with a single point

$$M(x,y) = \begin{cases} |\nabla S|(x,y), & |\nabla S|(x,y) > |\nabla S|(x',y') \; \& \\ & |\nabla S|(x,y) > |\nabla S|(x'',y'') \\ 0, \text{ else} \end{cases}$$

- **Hysteresis Thresholding**

  If ∇ at a pixel $>$ HIGH $\triangleq$ edge pixel
  $<$ Low $\triangleq$ non-edge pixel
  $<$ Low $\;\&\;$ $<$ HIGH ; iterate and check ~ connection with edge pixels ☺
  lt gt

- **Interest Point Detection**

  - Detect distinctive key points
  - Get feature vectors around interest points
  - Match by comparing distance b/w feature vectors

Feature detector must be robust to geometric and photometric differences.

- Edge is not a good interest point
- Expressive structure ~
  - direction of boundary of object changes abruptly
  - intersection between two or more edge segments

Corner is a good interest point

- Harris Corner Detector
  - Corner point can be recognized through a window
  - Large change in intensity in any direction

- SSD (Sum of squared differences)

  In the same fashion as correlation, but now take the difference and square it before summation

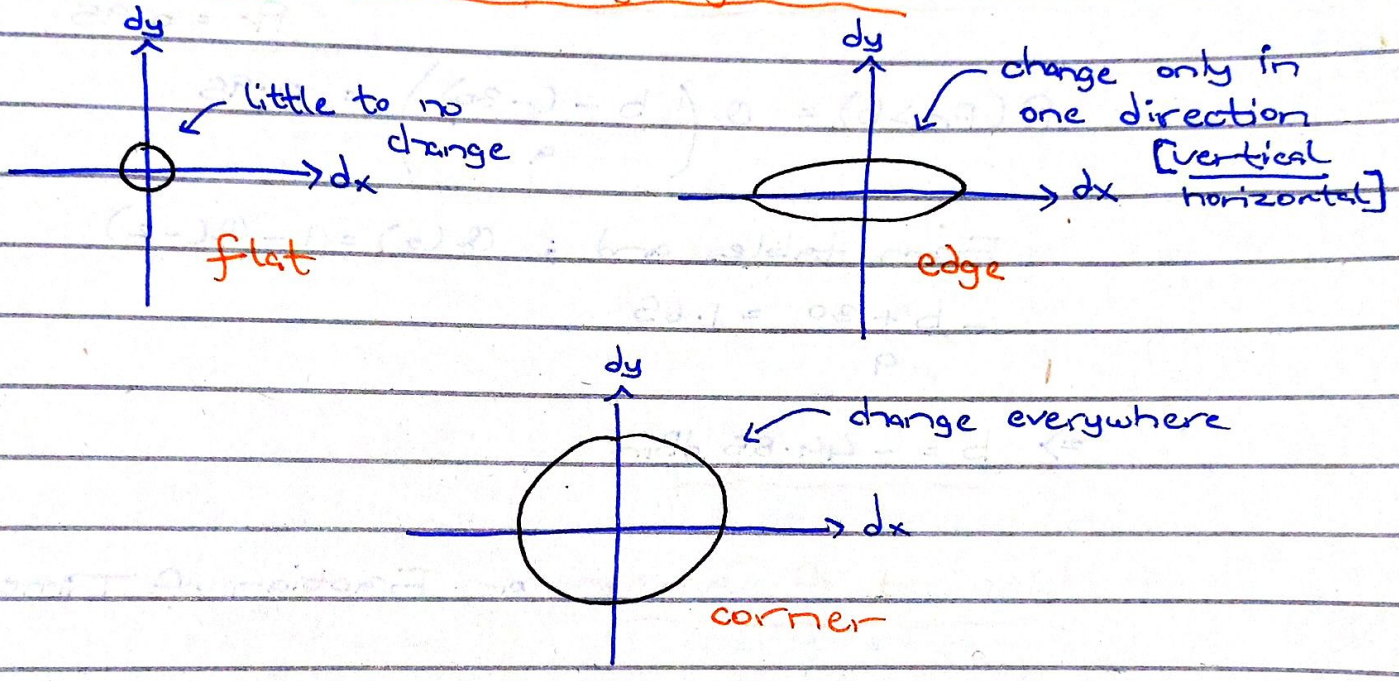
  → lower the SSD ; higher the correlation

$$SSD \underset{max}{min} = \sum_k \sum_l (\mp 2h(i+k, j+l) f[k, l)]$$

4/10/2023

Computer Vision

Distribution of image gradients



Harris Detector

- Weighted SSD:

$$E(u,v) = \sum_{(x,y)\in W} w(x,y)\left[I(x+u, y+v) - I(x,y)\right]^2$$

, where u and v are shifts

$w(x,y)$ ≜ window function {gaussian, uniform}

$E(u,v)$'s shape is same as that of $w(x,y)$

↳ where computation is very slow

- **Taylor Series Approximation**

$$E(u,v) = [u\ v]\left[\sum_{x,y} w(x,y)\begin{bmatrix} I_x \\ I_y \end{bmatrix}[I_x\ I_y]\right]\begin{bmatrix} u \\ v \end{bmatrix}$$

$$= [u\ v]\, M \begin{bmatrix} u \\ v \end{bmatrix}$$

Solving M:

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

- 2×2 shape
  - λ_1 and λ_2 eigenvalues

→ **Eigenvalues & Eigenvectors**

$\det(A - \lambda I) = 0$; $(A - \lambda I)x = 0$

↳ find eigenvalues ↳ find eigenvectors

- **Cornerness**

$$R = \lambda_1\lambda_2 - k(\lambda_1 + \lambda_2)^2 \quad \text{OR}$$

$$= \det(A) - k(\text{trace}(A))^2$$

10/10/23

Computer Vision

Transformations

└ funcs that convert points from one coordinate system to another

- 2D - to - 2D (images)
  └ transforming points from one 2D plane to another 2D plane.

- image registration
  └ process of transforming two images such that features overlap

{ x, y : original coords ; x', y' : transformed coords }

- translation
  └ moves object without deforming it

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} t_x \\ t_y \end{bmatrix}$$

- Scaling
  └ can change length and possibly direction.

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} S_x & 0 \\ 0 & S_y \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

- **reflection**

  special case of scaling

  - flip about y axis $\begin{bmatrix} -1 & 0 \\ 0 & 1 \end{bmatrix}$
  - flip about x axis $\begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}$

- **shearing**

  - horizontal $\begin{cases} x' = x + sy \\ y' = y \end{cases} \Rightarrow \begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} 1 & s \\ 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$
  - vertical $\begin{cases} x' = x \\ y' = sx + y \end{cases} \Rightarrow \begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ s & 1 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$

- **rotation**

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

$x = R\cos\phi \quad ; \quad y = R\sin\phi$

11/10/23

Computer Vision

Affine Transformation

- origin does not necessarily map to origin; there is translation
- Line → Line (Angles don't change ☺)
- parallel → (remain) parallel

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

6 parameters ~ 6 DoF

Projective Transformations

- parallel lines do not necessarily remain parallel; rest are that of affine

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

8 parameters ~ 8 DoF

Order of Transformations

$$x' = Sx$$

$$x'' = Rx' = RSx = (RS)x$$

$$\boxed{x'' = Mx}$$

- Read from right side to get the order of transformations

=> $AB \neq BA$ or $RS \neq SR$

except for some cases such as

$S_x = S_y$ (symmetry)

- Inverse transformations

$$[ABC]^{-1} = C^{-1} B^{-1} A^{-1}$$

↳ order reverses

- rest from Lecture / slides

Example

$$\begin{bmatrix} x \\ y \\ h \end{bmatrix} = \begin{bmatrix} 0 & 2 & 2 & 0 \\ 0 & 0 & 2 & 2 \\ 1 & 1 & 1 & 1 \end{bmatrix}$$

- Translate by $T(2,1)$

$$\begin{bmatrix} x' \\ y' \\ h' \end{bmatrix} = \begin{bmatrix} 1 & 0 & 2 \\ 0 & 1 & 1 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0 & 2 & 2 & 0 \\ 0 & 0 & 2 & 2 \\ 1 & 1 & 1 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} 2 & 4 & 4 & 2 \\ 1 & 1 & 3 & 3 \\ 1 & 1 & 1 & 1 \end{bmatrix}$$

- Translate by T(2,1) then rotate by R(45)

$$\begin{bmatrix} x' \\ y' \\ z' \end{bmatrix} = \begin{bmatrix} \cos 45 & -\sin 45 & 0 \\ \sin 45 & \cos 45 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 2 & 4 & 4 & 2 \\ 1 & 1 & 3 & 3 \\ 1 & 1 & 1 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} 0.707 & 2.121 & 0.707 & -0.707 \\ 2.121 & 3.535 & 4.949 & 3.535 \\ 1 & 1 & 1 & 1 \end{bmatrix}$$

Recovering Best Affine Transformation

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} a_1 & a_2 & a_3 \\ a_4 & a_5 & a_6 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

- Find interest points
- "optimize" ; minimize distances b/w feature vectors ; Find $[a_1\ a_2 \dots a_6]$

$$E(a) = \sum_{j=1}^{n} \left((a_1 x_j + a_2 y_j + a_3 - x_j)^2 + \dots \right.$$
$$\left. (a_4 x_j + a_5 y_j + a_6 - y_j)^2 \right)$$

- Pseudo Inverse

$$A^T A x = A^T B$$

$$x = (A^T A)^{-1} A^T B$$

17 / 10 / 23

Computer Vision

Warping

Given : X ; $A = [a_1\ a_2\ \dots\ a_6]^T$

Out : $X' = AX$

↳ place same color on each pixel of X' after applying A ; will create holes

image interpolation

- nearest neighbourhood : fill holes with near 'true' pixels
- for odd images (5×5, etc.) choose either furthest or nearest to give weightage to

bilinear interpolation

$$y_n = y_0 + \frac{y_1 - y_0}{x_1 - x_0}(x_n - x_0)$$; where n lise (hehe) (lies) in b/w 0 & 1

OR ; After simplification

$$y_x = \frac{y_1(x_2 - x_x) + y_2(x_x - x_1)}{x_2 - x_1}$$ Value of y at position x

where ;

x_2 : max position	y_2 : value at x_2
x_1 : initial position	y_1 : value at x_1

→ Example

$x = 2 \sim x_2$ } we start from index 1 — 6
not 0 - 5

$$y_x = \frac{10(6-2) + 40(2-1)}{6-1} = 16$$

x_5 → need x b/w 30 and 20

$$y' = \frac{30(6-3) + 20(3-1)}{6-1} = 26$$

Similarly, $y_{x_2} = \frac{10(6-3) + 40(3-1)}{6-1} = 22$

Now, column - wise ;

$$y_{x_6} = \frac{22(6-4) + 26(4-1)}{6-1} \approx 24$$

18/10/23

Perspective Projection | Computer Vision

↝ Homogeneous Coordinates

$$\left(\frac{x}{w}, \frac{y}{w}, w\right) \rightarrow (x, y)$$

- $\tilde{x} = (\tilde{x}, \tilde{y}, 1)^T \in \mathbb{P}^2$ → 2D Projective Space
  ↳ vectors only differing by scale are equivalent
  Eg. $(10, 20, 1)^T \equiv (30, 60, 3)^T$ → so on

- Ideal Points $\triangleq \tilde{w} = 0$ → $(\tilde{x}, \tilde{y}, 0)$
  ↳ $(x/0, y/0)$
  at infinity, from origin

- $\mathbb{P}^2 = \mathbb{R}^3 - (0,0,0)^T$ → does not define a direction
  ↳
  Similarly, for 3D points;
  $\mathbb{P}^3 = \mathbb{R}^4 - (0,0,0,0)^T$

- Lines in 2D

$$ax + by + c = 0 \; ; \; (a, b, c)^T = L$$

Notation

$$(a, b, c)^T = k(a, b, c)^T$$

↳ same lines for $k \neq 0$

- Point on a Line

2D point $x = (x, y)^T$

if, $ax + by + c = 0$; point will lie on the line

$$(x, y, 1)(a, b, c)^T = 0$$

$$\left\{ \begin{array}{c} (x, y, 1)\, L = 0 \\ x^T L = 0 \end{array} \right\}$$

- Intersection of Two Lines Slides

$L \times L' = 0 \gg x$

{Intersection of Two Lines}? Ignore

- Line Joining Two Points Slides

$x \times x' = 0 \gg l$ or L

- Intersection of Two Parallel Lines Slides

$L \times L' = (c' - c)(b, -a, 0)^T$

$\gg (b, -a, 0)^T \rightarrow$ Hence, parallel lines intersect at the ideal points

Example

$I = [-1, 0, 1]^T \quad I' = [-1, 0, 2]^T$

Intersection $\rightarrow$ $I \times I'$

$$\gg \begin{bmatrix} i & j & k \\ -1 & 0 & 1 \\ -1 & 0 & 2 \end{bmatrix} = i(0) - j(-2+1) + k(0)$$

$\Rightarrow$ $(0, 1, 0)$ $\rightarrow$ infinity in $+y$ direction

- Perspective Projection
  - orthographic Projection := Lines remain parallel
  - Perspective

- wider fow : focal length is small
- orthographic : focal length $\rightarrow \infty$

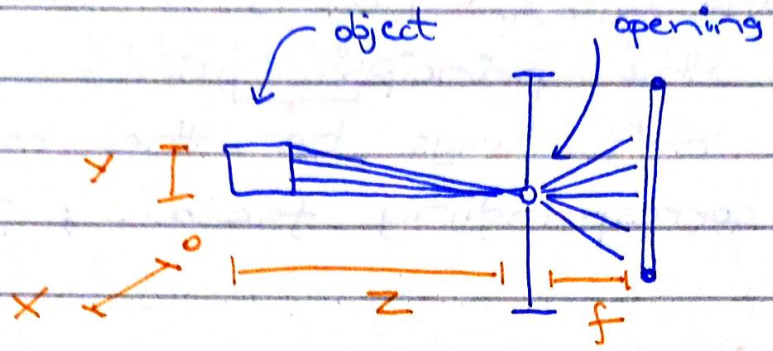
△ orthographic projection $\leftrightarrow$ parallel projection

△ where parallel lines meet (perspective) : is called the vanishing point

Lens : converges rays to points (removing blur)

Next lecture

Computer Vision



- in perspective projection, angles are lost ; but straight lines remain same

- World Point $[X, Y, Z]$ → Image $[x, y]$

» $\frac{f}{Z} = -y / Y$

↳ due to inversion / image formed is always upside down

→ $y = -\frac{fY}{Z}$, $x = -\frac{fX}{Z}$

$$\begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} -f/z & 0 & 0 \\ 0 & -f/z & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X \\ Y \\ 1 \end{bmatrix}$$

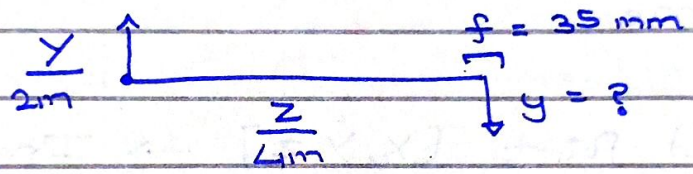
OR $$\begin{bmatrix} hx \\ hy \\ h \end{bmatrix} = \begin{bmatrix} f & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}$$

$hx = X$; $hy = Y$; $x = PX$

- Principal point : $x = Px$;
  $P = \text{diag}(f, f, 1)[I \mid 0]$ assumes origin to be the principal point
  - may not always be the case
  - add corresponding factors ; p_x and p_y

- Example :

$$y = -\frac{(f)(Y)}{Z} = -\frac{(35)(2000)}{(4000)} = -17.5 \text{ mm}$$



→ in pixel frame :
- suppose res : 640 x 480 pixels
  film : 8cm x 6cm

Principal point offset = 320, 240

→ Read from lectures

- 3D Transformations
  - Transformation of Lines

Under projective transformation;

$$l' = H^{-T} l$$

Theorem

Proof :

$$l^T x_i = 0$$

$$l^T \underbrace{H^{-1} \cdot H}_{I} x_i = 0 \quad \gg \quad l^T H^{-1} x_i' = 0$$

$$\hookrightarrow \; l' \triangleq l H^{-T}$$

Planes in 3D

$$\longrightarrow \qquad \pi^T X = 0$$

where $\pi = [\pi_1, \pi_2, \pi_3, \pi_4]^T$

- **Translation** (keep $\mathbb{P}^3 = \mathbb{R}^4 - (0,0,0,0)^T$ in mind)

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_x \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

- **Scaling**

$$\begin{bmatrix} S_x & 0 & 0 & 0 \\ 0 & S_x & 0 & 0 \\ 0 & 0 & S_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Computer Vision

3D shearing

└ From slides

Rotation in 3D

└ From slides

{ Properties: (of rotation matrices)

- orthonormal
- $R^{-1} = R^T \gg RR^T = I$
- $R_{z,0} = I$
- $R_{z\theta} R_{z\phi} = R_{z,\theta+\phi}$
- $R_{z,\theta}^{-1} = R_{z,-\theta}$

Rotation about Arbitrary Axis

$$\left.\begin{aligned} \theta &= \cos^{-1}\left[(Tr(R) - 1)/2\right] \\ k &= \frac{1}{2\sin\theta}\begin{bmatrix} r_{32} - r_{23} \\ r_{13} - r_{31} \\ r_{21} - r_{12} \end{bmatrix} \end{aligned}\right\} R_{k,\theta}$$

31/10/23

Computer Vision

Factorizing Transformations

- opposite of concatenations
- Any 2D Transformation can be written as RSR^T

» Given transformation matrix M

Decompose into M_1, M_2, M_3 etc.

- if M is symmetric :

  $M = M^T$

  Eigen-value decomposition
- if unsymmetric

  Single-value decomposition

Eigenvalues and Eigenvectors

L eigenvector u of matrix A satisfies :

$$Au = \lambda u$$

$$\sim (A - \lambda I)u = 0$$

Solving for Eigenvector

$$\begin{bmatrix} 6 & 3 \\ 2 & 1 \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

Perform row operations

$$\left[\begin{array}{cc|c} 6 & 3 & 0 \\ 2 & 1 & 0 \end{array}\right]$$

$$-3R_2 + R_1 \rightarrow R_1$$

x_1	x_2	
0	0	0
2	1	0

$$2x_1 + x_2 = 0$$

$$x_2 = -2x_1$$

Let $x_1 = t = 1$ → infinite solutions

$x_2 = -2$

Similarly; find eigen vector for each eigen value

$$\begin{bmatrix} 1 \\ -2 \end{bmatrix}$$

Single Value Decomposition

$$A_{m \times n} = U_{m \times m} \Sigma_{m \times n} V^T_{n \times n}$$

- U and V are orthonormal
- Σ is digonal ; $\sqrt{}$ of eigenvalues of AA^T or A^TA
- U and V are eigenvectors of AA^T and A^TA

Example:

$$A = \begin{bmatrix} 3 & 1 & 1 \\ -1 & 3 & 1 \end{bmatrix}$$

$A^T A \rightarrow V \quad ; \quad A A^T \rightarrow U$

$$\begin{bmatrix} 3 & 1 & 1 \\ -1 & 3 & 1 \end{bmatrix} \begin{bmatrix} 3 & -1 \\ 1 & 3 \\ 1 & 1 \end{bmatrix} = \begin{bmatrix} 11 & 1 \\ 1 & 11 \end{bmatrix} = U$$

└ Find eigenvalues of U'

$$\det\left(\begin{bmatrix} 11-\lambda & 1 \\ 1 & 11-\lambda \end{bmatrix}\right) = 0$$

$$\Rightarrow (11-\lambda)^2 - (1) = 0$$

$$\lambda = 12 \; ; \; \lambda = 10$$

└ Find eigenvectors of U'

$$\lambda = 12 : \quad \begin{bmatrix} -1 & 1 \\ 1 & -1 \end{bmatrix} \begin{bmatrix} U_{11} \\ U_{21} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

$$\lambda = 10 : \quad \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix} \begin{bmatrix} U_{12} \\ U_{22} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

$$U = \begin{bmatrix} U_{11} & U_{12} \\ U_{21} & U_{22} \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$$

Find (col-wise) magnitude of U and divide;

→ $U = \begin{bmatrix} 1/\sqrt{2} & 1/\sqrt{2} \\ 1/\sqrt{2} & -1/\sqrt{2} \end{bmatrix}$

→ $\Sigma = \begin{bmatrix} \sqrt{12} & 0 & 0 \\ 0 & \sqrt{10} & 0 \end{bmatrix}$ Same shape as A; Diagonal entries are that of either AA^T / A^TA's $\sqrt{\ }$'ed eigenvalues

Do the same for V as with U;

Then:

$A = U \Sigma V^T$

14/11/23

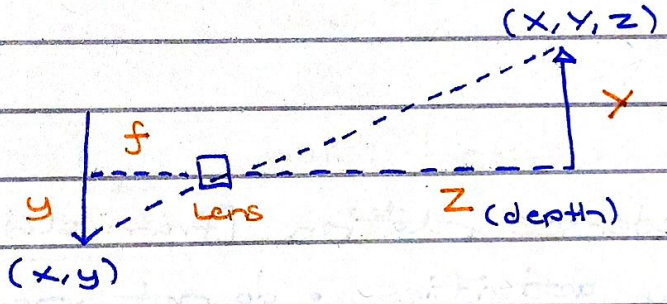
Computer Vision

Camera Model & Calibration

{ Intrinsic, Extrinsic } → Properties

Recall: $x = -\frac{fX}{Z}$; $y = -\frac{fY}{Z}$

∴ (X, Y, Z) World Point ∴ (x, y) Projected Point



→ $y = \frac{fY}{f-Z}$, $x = \frac{fX}{f-Z}$ → origin shift

→ $\begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} ? \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \end{bmatrix}$

↳ find !

→ $\begin{bmatrix} Ch_1 \\ Ch_2 \\ Ch_3 \\ Ch_4 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & -1/f & 1 \end{bmatrix} \begin{bmatrix} kX \\ kY \\ kZ \\ k \end{bmatrix} = \begin{bmatrix} kX \\ kY \\ kZ \\ k - kZ/f \end{bmatrix}$

- **Inverse homogeneous**

$$x = \frac{kX}{k - kZ/f}, \quad y = \frac{kY}{k - kZ/f}$$

Camera Model

P is projective matrix as made above

$$C_n = P C R_{-\phi}^{x} R_{-\theta}^{z} G \, w_n$$

$R_{-\phi}^{x} R_{-\theta}^{z} G$: any arbitrary matrices of transformation

w_n: world point

$CP = 0$ → (12×1)

C → $(2n \times 12)$

$P \sim [a_{11} \; a_{12} \; \dots \; a_{44}]$ — 12 params

15/11/23

Computer Vision

SIFT {Scale Invariant Feature Transform}

~ Pearson correlation coefficient $\{r \in [-1, 1]\}$
└ to find feature correlation

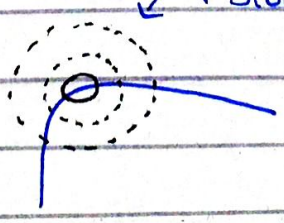
- We want invariance ito [scale, orientation, intensity]

- SIFT operates on blobs
  └ robust to occlusion & clutter

~ Detector ~ Descriptor ~ Correspondence
scene points | encode local window | match →

→ Harris corner detector fails to be robust against scale / scaling variations.

→ SIFT



↖ blobs ; problem : find optimal size of blob on the feature point

- SIFT works good upto a rotation of 60° world point

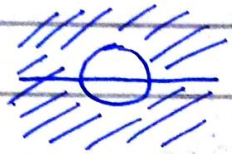
Steps

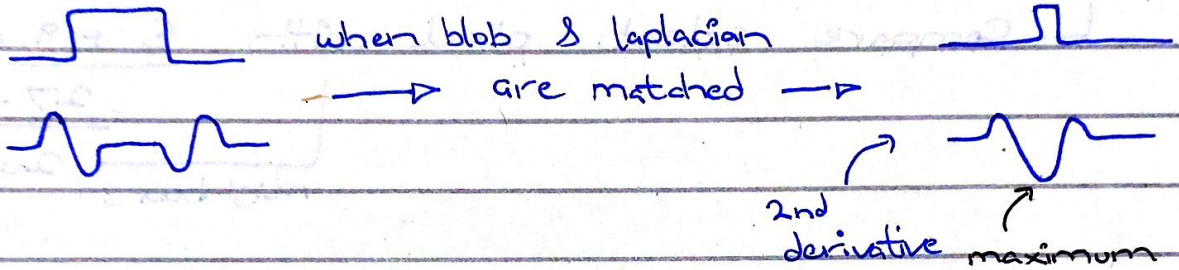
1. Detect the features
2. Remove the outliers — contrast & edge based ~
3. Adjust / undo the scaling + orientation
4. Design a descriptor
5. Matching based on the descriptor

21/11/23

Computer Vision

Edge Detection - Recap





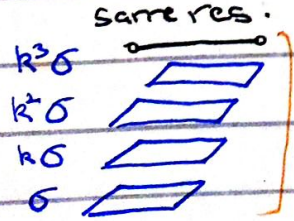
└ blob size depends on variance

→ Scale Space

image 0 → blur → resample (→ downscale) → subtract {difference of Gaussian}

└ varying the sigma / s

the stack of images of the same resolution



↳ similar for 2nd octave but downsized

→ We can approximate Laplacian through difference.

- DoG = $G(x,y,k\sigma) - G(x,y,\sigma)$
- Build the scale space
- Do the scale space peak detection
  - Compare select pixel with $\frac{8+9+9}{27-1}$ neighbours → 26 pixels
- outlier rejection ← check b/w two spaces (~continuous) ?
  - Let $x_0 = [x, y, \sigma]^T$, $z = [\delta x, \delta y, \delta\sigma]^T$

$$D(x_0 + x) = D(x_0) + \frac{\partial D}{\partial x}^T x + \frac{1}{2} x^T \left(\frac{\partial^2 D}{\partial x^2}\right) x$$

→ Minima / Maxima

$$\hat{x} = -\frac{\partial^2 D^{-1}}{\partial x^2} \frac{\partial D}{\partial x}$$

- Removing points on the edges [Further outlier rejection]

Hessian $H = \begin{bmatrix} D_{xx} & D_{xy} \\ D_{xy} & D_{yx} \end{bmatrix}$

$r = \frac{\alpha}{\beta}$ ← eigenvalues for eigenvectors of H

$Tr(H)^2 / \det(H) < (r+1)^2 / r$ for $r = 10$

L Descriptor

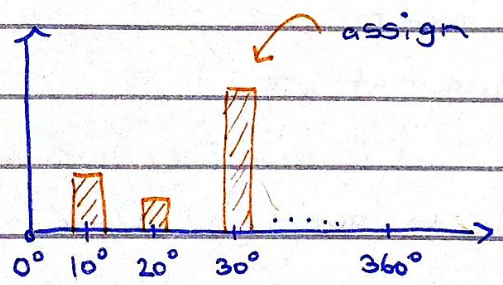
L descriptors for assigning signature to a feature & match the feature in the transformed image.

21/11/23

Computer Vision

→ Orientation Assignment & Descriptors

- Place a window on any feature at (x, y) and compute magnitude & direction near that pixel $x \pm 1$, $y \pm 1$
- after finding $m(x,y)$ and $\theta(x,y)$; plot the histogram



- find principal orientation of the whole image
- do inverse rotation by principal orientation on the pair image
- for descriptor, place a 16 x 16 window and make 4x4 weighted regions [8 bins]
- find matching % age of the descriptors in a 16 x 16 neighbourhood of each feature between pair images
- match using distance b/w two data arrays $H_1(k)$, $H_2(k)$ where H is the histogram

28/11/23

Computer Vision

Linear Regression

Gradient Descent

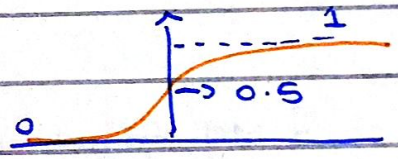
$$\theta_1 : \theta_1 - \alpha \frac{\partial J(\theta_1)}{\partial \theta_1}$$

where J is the cost func.
α is the learning rate

- $h_\theta(x) = \theta_0 + \theta_1 x$

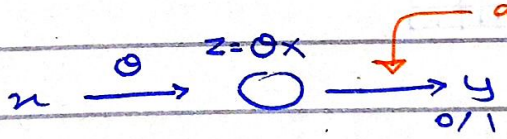
~ From slides

$$\text{Sigmoid}(z) = \frac{1}{1+e^{-z}}$$



↓ replace z with $h_\theta(x)$

Neurons

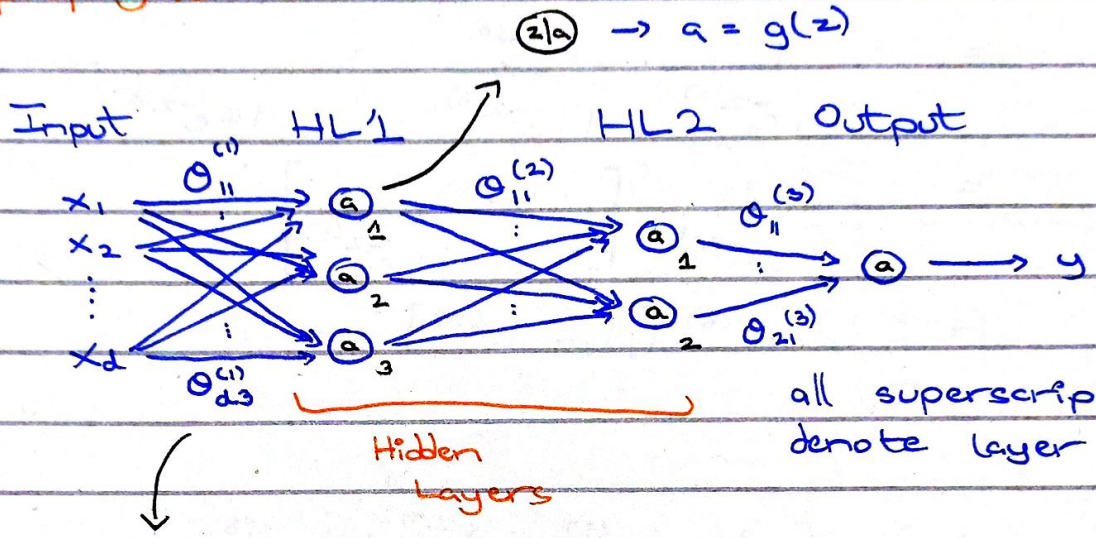


Next Slide

11/25/23

Computer Vision

Backpropagation



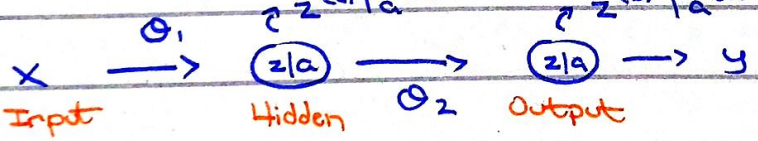
Let d = 3

all superscripts denote layer no.

Convention

$\theta^{(j)}_{mn}$ where j is layer no., m is where connection comes from and n is where it ends

└ can also be θ_{nm} [Its mostly θ_{nm}]



$g(z^{(3)}) = \theta_2 a^{(2)}$; $a^{(3)}$ $\delta \triangleq \frac{1}{2}(a^{(3)} - y)^2$

$$\frac{\partial \delta}{\partial \theta_2} = \frac{\partial \delta}{\partial a^{(3)}} \cdot \frac{\partial a^{(3)}}{\partial z^{(3)}} \cdot \frac{\partial z^{(3)}}{\partial \theta_2} \quad \text{— i}$$

$$\frac{\partial \delta}{\partial \theta_1} = \frac{\partial \delta}{\partial a^{(3)}} \cdot \frac{\partial a^{(3)}}{\partial z^{(3)}} \cdot \frac{\partial z^{(3)}}{\partial a^{(2)}} \cdot \frac{\partial a^{(2)}}{\partial z^{(2)}} \cdot \frac{\partial z^{(2)}}{\partial \theta_1} \quad \text{— ii}$$

→ From eq 1

$$\frac{\partial z^{(3)}}{\partial \theta_2} = \frac{\partial}{\partial \theta_2}\left[\theta_2 \cdot a^{(2)}\right] = a^{(2)}$$

$$\frac{\partial a^{(3)}}{\partial z^{(3)}} = \frac{\partial}{\partial z^{(3)}}\left[\frac{1}{1+e^{-z^{(3)}}}\right] = -(1+e^{-z^{(3)}})^{-2} e^{-z^{(3)}}(-1)$$

$$= (1+e^{-z^{(3)}})^{-2} e^{-z^{(3)}}$$

$$= \frac{1}{1+e^{-z^{(3)}}} \cdot \frac{e^{-z^{(3)}}}{1+e^{-z^{(3)}}} = \frac{1}{1+e^{-z^{(3)}}} \cdot \frac{-1+1+e^{-z^{(3)}}}{1+e^{-z^{(3)}}}$$

$$= \frac{1}{1+e^{-z^{(3)}}}\left[1 - \frac{1}{1+e^{-z^{(3)}}}\right]$$

$$= a^{(3)}\left[1 - a^{(3)}\right]$$

↳ $a^{(3)}$ is sigmoid

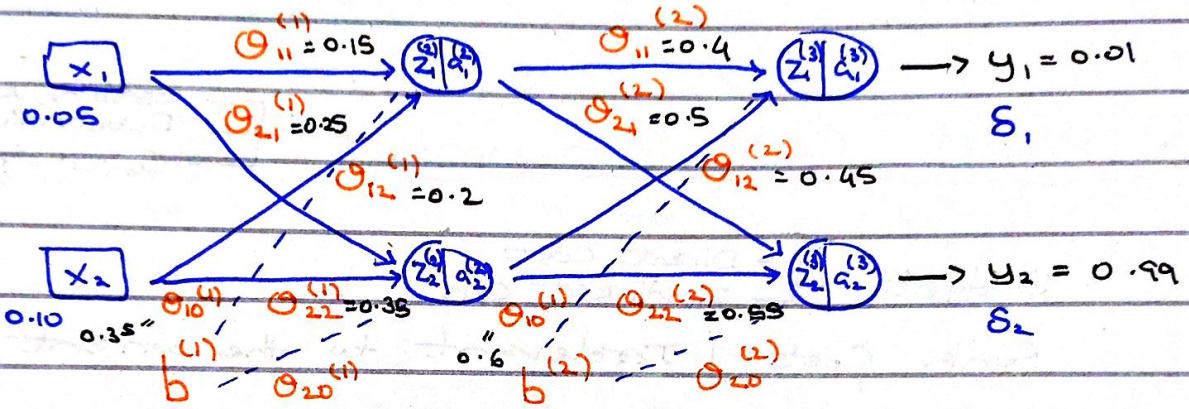
$$\frac{\partial \delta}{\partial a^{(3)}} = \frac{\partial}{\partial a^{(3)}}\left[1/2\,(a^{(3)} - y)^2\right] = a^{(3)} - y$$

Do same and substitute to update weights

4/11/23

Computer Vision

Backpropagation (Extended)



θ_{mn} : $m \triangleq$ destination index

$n \triangleq$ source index

$$\theta_i := \theta_i - \alpha \frac{\partial (J)}{\partial \theta_i}$$ ($J \hookrightarrow$ cost)

Gradient Descent

Total Loss $\delta = \delta_1 + \delta_2$

- Layer 2

$z_1^{(2)} = b^{(1)} + \theta_{11}^{(1)} x_1 + \theta_{12}^{(1)} x_2 = 0.377$

$a_1^{(2)} = 1/(1 + e^{-z_1^{(2)}}) = 0.59$

$z_2^{(2)} = b^{(1)} + \theta_{21}^{(1)} x_1 + \theta_{22}^{(1)} x_2 = 0.392$

$a_1^{(2)} = 1/(1 + e^{-z_2^{(2)}}) = 0.5968$

- Layer 3

$z_1^{(3)} = b^{(2)} + \theta_{11}^{(2)} a_1^{(2)} + \theta_{12}^{(2)} a_2^{(2)} = 1.105$

$a_1^{(3)} = 1/(1+e^{-z_1^{(3)}}) = 0.7513$

↳ Similarly ; $z_2^{(3)}$; $a_2^{(3)} = 0.7729$

→ $\delta_1 = \frac{1}{2}[y_1 - a_1^{(3)}]^2 = 0.274$

$\delta_2 = \frac{1}{2}[y_2 - a_2^{(3)}]^2 = 0.0234$

$\delta_T = \delta_1 + \delta_2 = 0.2985$ ⇝ objective is to make $\delta_T \approx 0$

Forward Prop. (Done)

Now, backprop :

$\theta_{11}^{(2)} := \theta_{11}^{(2)} - \alpha \frac{\partial}{\partial \theta_{11}^{(2)}} J$ where $J = \delta^T$

$\frac{\partial \delta^T}{\partial \theta_{11}^{(2)}} = \frac{\partial \delta^T}{\partial a_1^{(3)}} \frac{\partial a_1^{(3)}}{\partial z_1^{(3)}} \frac{\partial z_1^{(3)}}{\partial \theta_{11}^{(2)}}$ — i

$\frac{\partial \delta^T}{\partial a_1^{(3)}} = \frac{\partial}{\partial a_1^{(3)}}[\delta_1 + \delta_2] = a_1^{(3)} - y_1 = 0.7413$

$\frac{\partial a_1^{(3)}}{\partial z_1^{(3)}} = a_1^{(3)}[1 - a_1^{(3)}] = 0.1868$

$\frac{\partial z_1^{(3)}}{\partial \theta_{11}^{(2)}} = a_1^{(2)} = 0.5933$

→ i becomes $\frac{\partial \delta^T}{\partial \theta_{11}^{(2)}} = 0.08215$

$$\theta_{11}^{(2)} := \theta_{11}^{(2)} - \underbrace{\alpha}_{=0.6} \frac{\partial \delta^T}{\partial \theta_{11}^{(2)}} = 0.4 - 0.6\,(0.08215)$$

one update

→ ~ Updating $\theta_{11}^{(1)}$.

$$\theta_{11}^{(1)} := \theta_{11}^{(1)} - \alpha \frac{\partial \delta^T}{\partial \theta_{11}^{(1)}}$$

$$\frac{\partial \delta^T}{\partial \theta_{11}^{(1)}} = \frac{\partial \delta^T}{\partial a_1^{(2)}} \frac{\partial a_1^{(2)}}{\partial z_1^{(2)}} \frac{\partial z_1^{(2)}}{\partial \theta_{11}^{(1)}} \quad \rightarrow (1)$$

$\frac{\partial \delta^T}{\partial a_1^{(2)}}$ → (11)

$\frac{\partial a_1^{(2)}}{\partial z_1^{(2)}} = a_1^{(2)}(1-a_1^{(2)})$

$\frac{\partial z_1^{(2)}}{\partial \theta_{11}^{(1)}} = x_1$

$$\frac{\partial \delta^T}{\partial a_1^{(2)}} = \frac{\partial \delta_1}{\partial a_1^{(2)}} + \frac{\partial \delta_2}{\partial a_1^{(2)}} \quad \rightarrow (11)$$

$$\frac{\partial \delta_1}{\partial a_1^{(3)}} \frac{\partial a_1^{(3)}}{\partial z_1^{(3)}} \frac{\partial z_1^{(3)}}{\partial a_1^{(2)}} = (a_1^{(3)} - y_1)\, a_1^{(3)}(1-a_1^{(3)})\, \theta_{11}^{(2)} = 0.0554$$

$$\frac{\partial \delta_2}{\partial a_2^{(3)}} \frac{\partial a_2^{(3)}}{\partial z_2^{(3)}} \frac{\partial z_2^{(3)}}{\partial a_1^{(2)}} = (a_2^{(3)} - y_2)\, a_2^{(3)}(1-a_2^{(3)})\, \theta_{21}^{(2)} = -0.019$$

$\rightarrow$ (11) $\rightarrow \frac{\partial \delta^T}{\partial a_1^{(2)}} = 0.0364$

$\rightarrow$ (1) $\rightarrow \frac{\partial \delta^T}{\partial \theta_{11}^{(1)}} = (0.0364)\, a_1^{(2)}(1-a_1^{(2)})\, x_1$

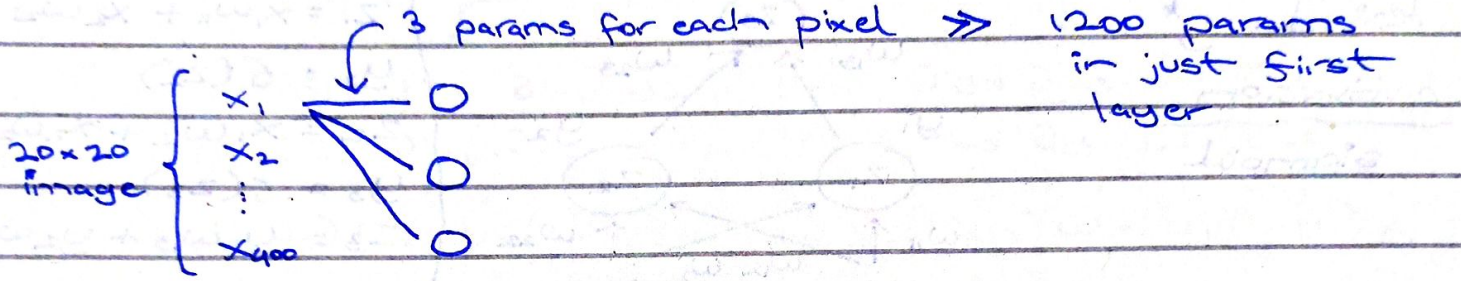
Do not trust the values, I did not do them myself BUHHHH :)

6/12/23

Computer Vision

CNNs

FNNs can't scale to images & video processing

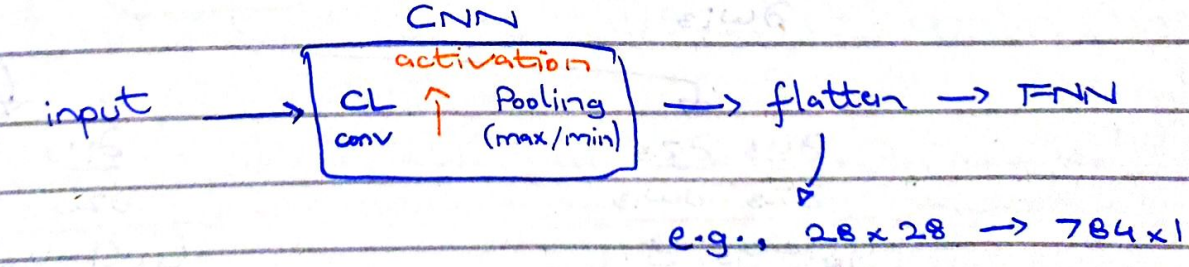


Typical CNN → Input, Convolutional, Pooling, FNN } Layers

Conv Filter

↳ kernel parameters are to be learned

Just do it from slides ^^



12/12/23

Computer Vision


CAM1


CAM2 ← Different orientation

$$A \times B = \begin{bmatrix} i & j & k \\ A_x & A_y & A_z \\ B_x & B_y & B_z \end{bmatrix}$$

$$= \begin{bmatrix} -A_z B_y + A_y B_z \; ; & -A_z B_x + A_x B_z \; ; & -A_y B_x + A_x B_y \end{bmatrix}$$

$$= \begin{bmatrix} 0 & -A_z & A_y \\ A_z & 0 & -A_x \\ -A_y & A_x & 0 \end{bmatrix} \begin{bmatrix} B_x \\ B_y \\ B_z \end{bmatrix}$$

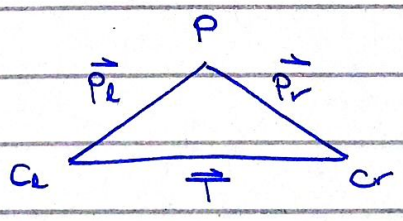
$$A \times B \equiv S \cdot B$$

↳ Matmul

Epipolar Geometry

$\vec{T}$ vector joining centers of two cameras

$\vec{P_l}$, $\vec{P_r}$ } vectors joining $\vec{P}$ and l & r cameras



→ Coplanarity Constraint

$$(\vec{P_l} - \vec{T})^T \vec{T} \times \vec{P_l} = 0 \quad | \quad \vec{P_r} = R(\vec{P_l} - \vec{T})$$

$\therefore R^{-1} = R^T$ » $\vec{R}^T \vec{P_r} = (\vec{P_l} - \vec{T})$ $\quad \vec{P_r}^T R = (\vec{P_l} - \vec{T})^T$

» $\vec{P_r}^T R \vec{T} \times \vec{P_l} = 0$

- $RS \triangleq E$ → essential matrix

  └→ $\vec{p}_r^{\,T} E \vec{p}_l = 0$

$$\begin{cases} x_l = M_l p_l \\ x_r = M_r p_r \end{cases} \Rightarrow \begin{matrix} M_l^{-1} x_l = p_l \\ M_r^{-1} x_r = p_r \end{matrix}$$

→ $x_r^T M_r^{-T} E M_l^{-1} x_l = 0$

→ $x_r^T \left[M_r^{-T} E M_l^{-1} \right] x_l$

→ $\boxed{x_r^T F x_l = 0}$

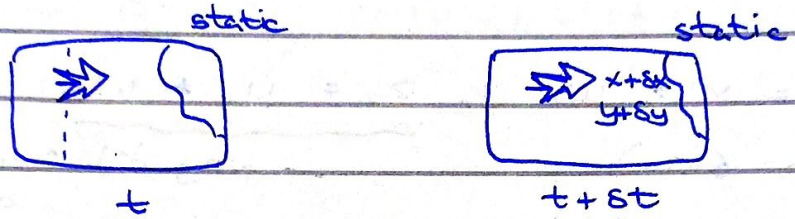
└→ fundamental matrix

19/12/23

Computer Vision

Optical Flow

└ to describe image motion



object tracking etc

- Brightness Constancy assumption A1

$$f(x,y,t) = f(x+dx, y+dy, t+dt)$$

- Displacements and Time step are small A2

$$f(x+dx, y+dy, t+dt) = f(x,y,t) + \cdots$$
$$\partial f/\partial x\, dx + \partial f/\partial y\, dy + \partial f/\partial t\, dt$$

Higher order → 0 as small displacements are considered

> Subtract A_1 from A_2

$$f_x dx + f_y dy + f_t dt = 0$$

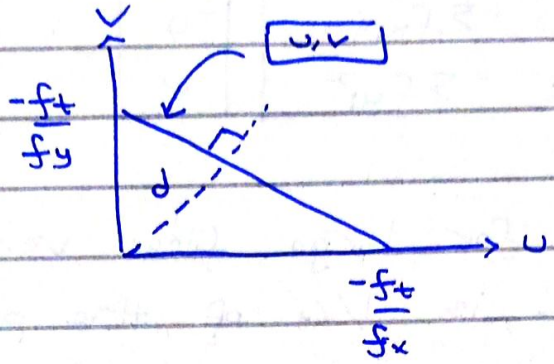
< Divide by dt >

$$f_x u + f_y v + f_t = 0 \quad \therefore f_x = \partial f/\partial x \quad f_y = \partial f/\partial y$$
$$f_t = \partial f/\partial t$$

$\frac{dx}{dt}$ $\frac{dy}{dt}$ → velocities in x & y

$$f_x u + f_y v = -f_t$$
$$v = -\frac{f_x}{f_y} u - \frac{f_t}{f_y}$$

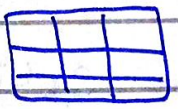
straight line



d := normal flow

- Lucas & Kanade
  - assumption: optical flow in neighbourhood is same

$\rightarrow f_x u + f_y v = -f_t$



Method 1

$$\begin{aligned} f_{x_1} u + f_{y_1} v &= -f_{t_1} \\ f_{x_2} u + f_{y_2} v &= -f_{t_2} \\ &\vdots \end{aligned}$$

$$\begin{bmatrix} f_{x_1} & f_{y_1} \\ f_{x_2} & f_{y_2} \\ \cdots & \cdots \\ \cdots & \cdots \\ f_{x_9} & f_{y_9} \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix} = \begin{bmatrix} -f_{t_1} \\ -f_{t_2} \\ \cdots \\ \cdots \\ -f_{t_9} \end{bmatrix}$$

9×2 — 9 as 3×3 window

9×1

$$U = (A^T A)^{-1} A^T f_t$$

Method 2 $\{ \min \sum_i (f_{x_i} u + f_{y_i} v + f_{t_i})^2 \}$ J

$$\partial J / \partial u = \Sigma (f_{x_i} u + f_{y_i} v + f_{t_i}) f_{x_i} = 0$$

$$\partial J / \partial v = \Sigma (f_{x_i} u + f_{y_i} v + f_{t_i}) f_{y_i} = 0$$

$$\begin{bmatrix} \Sigma fx_i^2 & \Sigma fx_i fy_i \\ \Sigma fx_i fy_i & \Sigma fy_i^2 \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix} = \begin{bmatrix} -\Sigma fx_i f_{t_i} \\ -\Sigma fy_i f_{t_i} \end{bmatrix}$$

Pyramids — for large flow vectors

└ each level is 1/4 of the previous level

- Level 4 : 8×8 → Level 3 : 4×4
- 8 + 8 = 16 / 16/4 = 4 → 4×4 =

Reduce (1D) $g_l(i) = \sum_{m=-2}^{2} w(m)\, g_{l-1}(2i+m)$

↳ previous level

idx | -2 | -1 | 0 | 1 | 2 |

vals. | | | | | |

Gaussian

Similarly,

Expand (1D) $g_{l,n}(i) = \sum_{p=-2}^{2} w(p)\, g_{l,n-1}\left(\frac{i-p}{2}\right)$

Convolution Mask

$a + 2b + 2c = 1$

[Sum should be equal to one

$a + 2c = 2b$

[all nodes should contribute equally

SUBTRACT

└ $-2b = 2b - 1 \gg b = 1/4$, $c = \frac{1}{4} - \frac{a}{2}$

Clustering

└ in images | segmentation

- K Means
  - ↳ Hyperparameter : K
    - Initialize centers
    - Assign cluster (→ to) each example / point based on distance
    - Update the centroid position using means

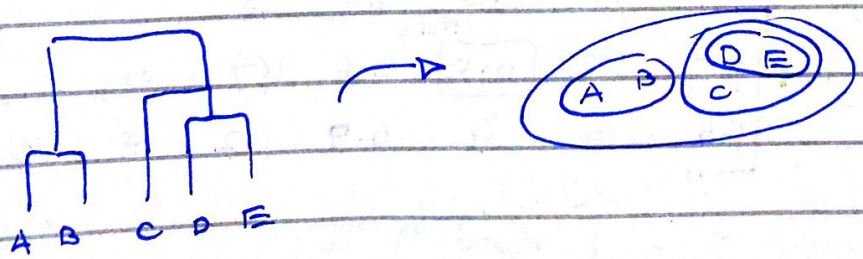
Distortion := $J(c^{(1)}, c^{(2)}, \dots \mu_1, \mu_2 \dots) = \frac{1}{N} \sum_i \| x_i - \mu_{c^{(i)}} \|^2$

minimize $J(\dots)$

- DBSCAN
  - ↳ Hyperparameters : Radius (Eps), MinPts
    - border point
    - core point

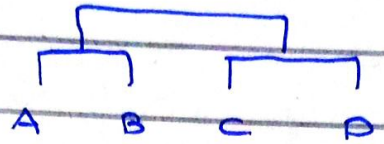
} // noise point

- Hierarchical Clustering



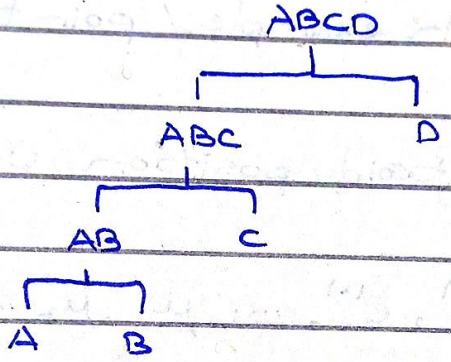
└ can cut it down

- Agglomerative (Bottom up)



Dendrogram

- Divisive (Top - down)



- Agglomerative
  - Single link [minimum distance]
  - Multi link / Complete link (maximum distance)

	x	x
p1	2	1
p2	4	3
p3	3	4
p4	7	1
p5	1	2
p6	5	2

Distance Matrix

	p1	p2	p3	p4	p5	p6
p1	0					
p2	3.1	0				
p3	2	1.2	0			
p4	1.8	3.5	4.1	0		
p5	1.2	0.8 (min)	2.6	1.7	0	
p6	3	4	0.9	1.2	3	0

↑ Random Values ☺

next ; $d_{se} = \min(d(P1, [P2, P5]))$

$= \min(d(P1, P2), d(P1, P5))$

$= \min(3.1, 1.2) = 1.2$

↳ Update

	P1	[P2,P5]	P3	P4
P1	0			
[P2,P5]	1.2 (min)	0		
P3	2	1.2 (min)	0	
P4	1.8	1.7	4.1	0

→ Take any

Sir ignored P6 & so will i

Hehe

→ Update

	(P1,[P2,P5])	P3	P4
(P1,[P2,P5])	0		
P3	1.2	0	
P4	1.7	4.1	0

$d_{se} = \min(d(P3, (P1, [P2, P5])))$

$= \min(d(P3, P1), d(P3, [P2, P5]))$

$= \min(d(P3, P1), d(P3, P2), d(P3, P5))$

(I may be down bad for my crush 

STOP.

2/01/24

Computer Vision

Face Recognition using Eigenvalues

1 : $N \times N$ image $\longrightarrow$ $[N^2 \times 1]$ vector (Γ_i)

2 : $\Psi = \frac{1}{M} \sum_{i}^{M} \Gamma_i$ $\rightsquigarrow$ image flattened vectors } Mean

$\Rightarrow [N^2 \times 1]$

3 : $\phi_i = \Gamma_i - \Psi$ } Mean centering

4 : $A = [N^2 \times k]$ where k is number of faces

{ Cov (A) } Let $C_1 = AA^T$

$\hookrightarrow$ $AA^T u_i = \lambda_i u_i$

[eigen vector (faces) ; eigen value

$\hookrightarrow$ $C u_i = \lambda_i u_i$

$u_i \Rightarrow [N^2 \times 1]$

Alternative

$C_2 = A^T A$

Then, $A^T A v_i = \lambda_i v_i$

$C v_i = \lambda_i v_i$

$v_i \Rightarrow [k \times 1]$ with k eigenvalues

5: $\Rightarrow$ $\underbrace{AA^T}_{C_1} \underbrace{A v_i}_{\text{eigen vector of } C_1} = \lambda_i \underbrace{A v_i}_{\text{eigen vector of } C_1}$ ($\lambda_i \rightarrow$ eigen value of C_1)

6: By comparison, $\underbrace{u_i = A v_i}_{\text{Found the eigen faces}} \gg [N^2 \times 1] = [N^2 \times k][k \times 1]$

$U \Rightarrow [N^2 \times k]$

7: { Compute scores and keep eigen faces that contribute 95 % to the overall score }

cumulative sum

$\Theta \Rightarrow [N^2 \times d]$ where $d \leq k$

8: Weights | $w_i = \Theta^T \phi_i \equiv [d \times N^2][N^2 \times 1]$

ϕ_i : mean centered

$\Omega = [w_1 \; w_2 \; \dots \; w_k]$

9: Reconstruction

$\hat{\phi}_i = \Theta w_i \equiv [N^2 \times d][d \times 1]$

$\equiv [N^2 \times 1]$

10 : Recognition

Given test image Γ

- Normalize $\phi = \Gamma - \Psi$
- Project ϕ on eigenface to calculate weight vector
  - $w = \Theta^T \phi \equiv [d \times N^2][N^2 \times 1] \equiv [d \times 1]$
  - Find minimum distance from training weights

??? Profit !

1 0 0 2) me org fr fr no cap

im retarded

Example

I_1 (img 1) $= \begin{bmatrix} 0 & 0 & 0 \\ 10 & 10 & 10 \\ 0 & 0 & 0 \end{bmatrix}$ [3x3]

I_2 (img 2) $= \begin{bmatrix} 0 & 10 & 0 \\ 0 & 10 & 0 \\ 0 & 10 & 0 \end{bmatrix}$ [3x3]

$\Gamma_1 = \begin{bmatrix} 0 \\ 10 \\ 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix}$ [9x1] $\quad \Gamma_2 = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 10 \\ \vdots \\ 0 \end{bmatrix}$ [9x1] $\quad \Psi = \frac{1}{2} \begin{bmatrix} 0 \\ 10 \\ 0 \\ 10 \\ \vdots \\ 0 \end{bmatrix}$ [9x1]

$\Gamma_1 - \Psi$ $\Gamma_2 - \Psi$

$$\Phi_1 = \begin{bmatrix} 0 \\ 5 \\ 0 \\ -5 \\ 0 \\ -5 \\ 0 \\ 5 \\ 0 \end{bmatrix}_{[9\times1]} \qquad \Phi_2 = \begin{bmatrix} 0 \\ -5 \\ 0 \\ 5 \\ 0 \\ 5 \\ 0 \\ -5 \\ 0 \end{bmatrix}_{[9\times1]}$$

} Mean Centering

$$A = [\Phi_1 \quad \Phi_2]_{[9\times2]}$$

$$A^TA = \begin{bmatrix} 100 & -100 \\ -100 & 100 \end{bmatrix}_{[2\times2]}$$

→ $A^TA\, v_i = \lambda_i v_i$

Find v_i → $\bar{A} = A^TA$

$\det(\bar{A} - \lambda I) = 0$ → Get λ_1, λ_2

Use λ_1, λ_2 → v_1, v_2

$$v_1 = \begin{bmatrix} 1 \\ 1 \end{bmatrix}_{[2\times1]} \qquad v_2 = \begin{bmatrix} -1 \\ 1 \end{bmatrix}_{[2\times1]}$$

→ $u_1 = Av_1$, $u_2 = Av_2$

Yo We get ;

$$U_1 = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}_{[9\times1]} \qquad U_2 = \begin{bmatrix} 0 \\ -10 \\ 0 \\ 10 \\ 0 \\ 10 \\ 0 \\ -10 \\ 0 \end{bmatrix}_{[9\times1]}$$

Eigenvectors of AA^T computed using A^TA

Normalize U_i's

mean U_1 is zero

mean $U_2 \rightarrow \sqrt{10^2+10^2+10^2+10^2} = 20$

$$\bar{U}_2 = \begin{bmatrix} 0 \\ -0.5 \\ 0 \\ 0.5 \\ 0 \\ 0.5 \\ 0 \\ -0.5 \\ 0 \end{bmatrix}_{[9\times1]}$$

Normalized eigenvectors

Weights → $[1\times9][9\times1] \equiv [1]$

$$w_{1i_1} = U_1^T \phi_1$$

$$w_{2i_1} = U_2^T \phi_1$$

$$w_{1i_2} = U_1^T \phi_2$$

$$w_{2i_2} = U_2^T \phi_2$$

$$\Omega_{i_1} = [w_{1i_1}^{0} \; w_{2i_1}^{0}] \quad [1\times2]$$

$$\Omega_{i_2} = [w_{1i_2}^{0} \; w_{2i_2}^{10}] \quad [1\times2]$$

Trained Weights

$\overline{\Omega}_{i_1} = 0 \qquad \overline{\Omega}_{i_2} = 5$ } mean

Test

$$I_{test} = \begin{bmatrix} 0 & 7 & 3 \\ 0 & 10 & 10 \\ 0 & 10 & 0 \end{bmatrix}$$

$$\Gamma_{Test} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 7 \\ 10 \\ 10 \\ 3 \\ 10 \\ 0 \end{bmatrix} \qquad \Phi_{Test} = \begin{bmatrix} 0 \\ -5 \\ 0 \\ 2 \\ 0 \\ 5 \\ 3 \\ 5 \\ 0 \end{bmatrix}$$

Mean Centering

$\Phi_{Test} = \Gamma_{Test} - \Psi$ ← of the training set

$$w_{1 i_{test}} = u_1^T \Phi_{Test} \; ; \; w_{2 i_{test}} = u_2^T \Phi_{test}$$

$$\Omega_{Test} = [0 \;\; 3.5]$$

↳ Find distance with all trained weights

choose the minimum distance for recognition