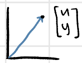
Vector

- column vector → $v \in R^{n \times 1}$ → $v = \begin{bmatrix} v_1 \\ v_2 \\ v_n \end{bmatrix}$
- row vector → $v^T \in R^{1 \times n}$ (transpose) → $v^T = [v_1 \; v_2 \dots v_n]$

Vector uses

- can represent offset in 2D/3D space
- data can be treated as a vector
  - these don't have geometric interpretations



Matrix

- an array of numbers → $A \in R^{m \times n}$
- size is m (rows) by n (columns)
- if m=n, then the matrix is a square

$$\begin{bmatrix} a_{11} & a_{12} \dots & a_{1n} \\ a_{21} & a_{22} \dots & a_{2n} \\ a_{31} & a_{32} \dots & a_{3n} \end{bmatrix} \xrightarrow{\text{flattening}} [a_{11}, a_{12}, a_{1n}, a_{21}, a_{22}, a_{2n}, a_{31}, a_{32}, a_{3n}]$$

Images

- In Python
  - an image → matrix of pixel brightness

Color Images

- Grayscale images
  - 1 number per pixel
  - stored as an m × n matrix
- Color images
  - 3 numbers per pixel → Red, Green, Blue
  - stored as m × n × 3 matrix

Norm → $\|u\|_2 = \sqrt{\sum_{i=1}^{n} u_i^2}$

- any function that satisfies 4 properties $f: R^n \to R$
  - Non Negatives → for all $u \in R^n$, $f(u) \geq 0$
  - Definiteness → $f(u) = 0$, only if $u = 0$
  - Homogenity → for all $u R^n$, $t \in R$, $f(tu) = |t| f(u)$
  - Triangle inequality → for all $u, y \in R^n$, $f(u+y) \leq f(u) + f(y)$

Dot Product → aka Inner product

$$[a \; b \; c] \begin{bmatrix} d \\ e \\ f \end{bmatrix} = [ad + be + cf]$$

- if B is a unit vector
  - the A.B = length of A
  - which lies in the direction of B

SPECIAL MATRICES

Identity Matrix I

$$I_{3\times3} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Diagonal matrix

$$\begin{bmatrix} 3 & 0 & 0 \\ 0 & 7 & 0 \\ 0 & 0 & 2.5 \end{bmatrix}$$

Symmetric matrix

$A^T = A$

$$\begin{bmatrix} 1 & 2 & 5 \\ 2 & 1 & 7 \\ 5 & 7 & 1 \end{bmatrix}$$

Skew symmetric matrix

$$\begin{bmatrix} 1 & -2 & -5 \\ 2 & 1 & -7 \\ 5 & 7 & 1 \end{bmatrix}$$

MATRIX OPERATIONS

Addition

- can only add a matrix with
  - matching dimensions

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix} + \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} = \begin{bmatrix} a+1 & b+2 \\ c+3 & d+4 \end{bmatrix}$$

- a scalar

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix} + 7 = \begin{bmatrix} a+7 & b+7 \\ c+7 & d+7 \end{bmatrix}$$

Scaling

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix} \times 3 = \begin{bmatrix} 3a & 3b \\ 3c & 3d \end{bmatrix}$$

Powers

- $A \cdot A = A^2$
- $A \cdot A \cdot A = A^3$

Product → aka Matrix Multiplication

A × B → order matters

$$\begin{bmatrix} 0 & 2 \\ 4 & 6 \end{bmatrix} \times \begin{bmatrix} 1 & 3 \\ 5 & 7 \end{bmatrix} = \begin{bmatrix} 0\times1 + 2\times5 & 0\times3 + 2\times7 \\ 4\times1 + 6\times5 & 4\times3 + 6\times7 \end{bmatrix}$$

- The product of 2 matrices is
  - Associative $(AB)C = A(BC)$
  - Distributive $A(B+C) = AB + AC$
  - Not Commutative $AB \neq BA$

Transpose

- flipped matrix
  - so row 1 becomes column 1
  - $(ABC)^T = C^T B^T A^T$ → useful identity

$$\begin{bmatrix} 0 & 1 \\ 2 & 3 \\ 4 & 5 \end{bmatrix}^T = \begin{bmatrix} 0 & 2 & 4 \\ 1 & 3 & 5 \end{bmatrix}$$

Trace

- trace (A) = sum of diagonal

$$tr\left(\begin{bmatrix} 1 & 3 \\ 5 & 7 \end{bmatrix}\right) = 1 + 7 = 8$$

- $tr(AB) = tr(BA)$
- $tr(A+B) = tr(A) + tr(B)$

Determinant

- det(A) returns a scalar
- represents area

$$det\left(\begin{bmatrix} a & b \\ c & d \end{bmatrix}\right) = ad - bc$$

- $det(AB) = det(A) \cdot det(B)$
- $det(AB) = det(BA)$
- $det(A^{-1}) = 1/det(A)$
- $det(A^T) = det(A)$
- $det(A) = 0$ → A is singular

TRANSFORMATION

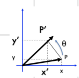
done using multiplication
$u' = Au$

Scaling

$$\underset{S}{\begin{bmatrix} S_x & 0 \\ 0 & S_y \end{bmatrix}} \times \underset{u}{\begin{bmatrix} u \\ y \end{bmatrix}} = \underset{u'}{\begin{bmatrix} S_x u \\ S_y y \end{bmatrix}}$$

Rotation (2D)

- counter clockwise rotation by an angle θ



$$\underset{P'}{\begin{bmatrix} u' \\ y' \end{bmatrix}} = \underset{R}{\begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix}} \underset{P}{\begin{bmatrix} u \\ y \end{bmatrix}} = \begin{bmatrix} \cos\theta u - \sin\theta y \\ \cos\theta y + \sin\theta u \end{bmatrix}$$

Multiple Transformations

$P' = R_2 R_1 S p$

- right to left → $R_2(R_1(Sp))$

Order Matters

$P' = T \cdot S \cdot P$ (translation, scale, rotation)

$$= \begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} S_x & 0 & 0 \\ 0 & S_y & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} u \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} S_x & 0 & t_x \\ 0 & S_y & t_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} u \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} S_x u + t_x \\ S_y y + t_y \\ 1 \end{bmatrix}$$

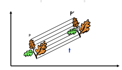
$P' = S \cdot T \cdot P$

$$= \begin{bmatrix} S_x & 0 & 0 \\ 0 & S_y & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} u \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} S_x & 0 & S_x t_x \\ 0 & S_y & S_y t_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} u \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} S_x u + S_x t_x \\ S_y y + S_y t_y \\ 1 \end{bmatrix}$$

→ Different ← $\neq$

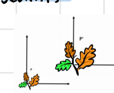
2D Transformation using Homogenous Coordinates

Translation



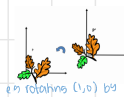
$$T = \begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix}$$

Scaling



$$S = \begin{bmatrix} S_x & 0 & 0 \\ 0 & S_y & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Rotation



$$R = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix}$$

- $R \cdot R^T = R^T \cdot R = 1$
- $det(R) = 1$
  - R produces rotation in the opposite direction

e.g. rotating (1,0) by 90°

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

Matrix Inverse

- A^{-1}

$$\begin{bmatrix} 2 & 0 \\ 0 & 3 \end{bmatrix}^{-1} = \begin{bmatrix} 1/2 & 0 \\ 0 & 1/3 \end{bmatrix}$$

- $A \cdot A^{-1} = I$ → if matrix has an inverse you can undo a transformation
- $(A^{-1})^{-1} = A$
- $(AB)^{-1} = B^{-1} A^{-1}$
- $A^{-T} = (A^T)^{-1} = (A^{-1})^T$

Matrix rank

- col rank(A) = max no. of linearly independent col vectors of A
- row rank(A) = max no. of linearly independent row vectors of A
- col rank = row rank
- for transformation matrices
  - it tells you dimmentions of output
  - e.g. if rank(A) = 1, then transformation $p' = Ap$
- matrix with rank 1

$$\begin{bmatrix} 1 & 1 \\ 2 & 2 \end{bmatrix} \times \begin{bmatrix} u \\ y \end{bmatrix} = \begin{bmatrix} u + y \\ 2u + 2y \end{bmatrix}$$

- full rank → no information lost
  - m × m matrix
  - has inverse matrix
    - Maps an $m \times 1$ vector uniquely to another $m \times 1$ vector
- singular
  - rank < m
  - inverse does not exist
    - At least one dimension is getting collapsed. No way to look at the result and tell what the input was
- for non-square matrice
  - inverse does not exist

e.g. rank 1 matrix transformation collapses a 2D image into a line → useful for detecting edges

Genaralized Matrix

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix} \times \begin{bmatrix} u \\ y \end{bmatrix} = \begin{bmatrix} au + by \\ cu + dy \end{bmatrix}$$

we can
- rotate
- scale
- skew
- can not translate

Homogenous Matrix

$$\begin{bmatrix} a & b & c \\ d & e & f \\ 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} u \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} au + by + c \\ du + ey + f \\ 0 + 0 + 1 \end{bmatrix}$$

(c, f: translation)

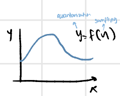
we can
- rotate
- scale
- skew
- Translate
- Divide → will divide the result by its last coordinate after doing matrix multiplication

$$\begin{bmatrix} u \\ y \\ 7 \end{bmatrix} \cdot \begin{bmatrix} u/7 \\ y/7 \\ 1 \end{bmatrix}$$

Digitization

- process of transforming continuous space into discrete space

Image → Sampling Quantization → Digital Image



1. Sampling

- Discretization of domain
  - width and height of image
- specifies resolution of image

$y = f(u)$ → domain

2. Quantization

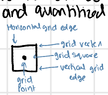
- Discretization of range
- specifies color space
  - what to store in each domain

$y = f(u)$ → range

Types of Digitization

- 1D function — discretization performed in
- 2D function → 2 axis
- 3D function → 3 axis
- an arc
- Grayscale image → 2 axis, intensity

Pictures are sample and quantitized

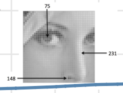


Definition

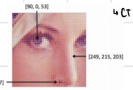
- an image P is a function defined on a finite rectangular subset G (→ 2D Grid) of a regular planar orthogonal array
- P assigns a value of P(p) to each P ∈ G
  - an element of G is called a pixel

Resolution
↳ a display parameter
↳ defined by dots per inch (DPI)
72 dPi → standard value for devices

Gray Scale image
↳ the image contains discrete no. of pixels
↳ Pixel value: grayscale
↳ [0, 255]



Color image
↳ the image contains discrete no. of pixels
↳ Pixel value: color
↳ RGB : [R,G,B]
↳ Lab : [L, a, b]
↳ HSV : [H, S, V]



Other images
↳ Infrared image
↳ ultrasound image
↳ CT scan image

Histograms
↳ Contrast
↳ Bright images have histograms to the right
↳ Dark images have histograms to the left
↳ Peak Distributions
↳ high contrast images have seperated peaks
↳ low contrast images have peaks closer
↳ identify uniformity
↳ a well exposed image will have a balanced histogram spread accross all brightness levels

intensity profiles
↳ dark → down
↳ bright → up

image noise
↳ Light variations
↳ Camera Electronics
↳ Surface Reflectance
↳ Lens → have something in your lens

noise
↳ It is random
↳ It occurs with some probability
↳ has a distribution

Types of Noise

1. Additive noise
↳ noise is added to original pixel intensity independently

$I_{new} = I_{og} + N$ (image, noise)

Solution: Gaussian Blur

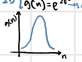
2. Multiplicative noise
↳ gives grainy texture
↳ noise scales with pixel intensity

$I_{new} = I_{og} \times N$ (image, noise)

3. Gaussian noise
mean = 0 (centered at true value)
variance: controls intensity spread
Solution: Gaussian Filter

1D $g(n) = e^{-\frac{n^2}{2\sigma^2}}$ → sd → intensity spread

$g(n) = e^{-\frac{(n^2+y^2)}{2\sigma^2}}$ 2D



4. Salt and Pepper noise
↳ random black and white dots
↳ each pixel is randomly made black or white with a uniform probability distribution
Solution: Median Filter

Convolution
→ True Filtering Operation
↳ The filter is flipped before applying

$f * h = \sum_k \sum_l f(k,l) h(-k,-l)$

f = Image
h = Kernel

h_1	h_2	h_3
h_4	h_5	h_6
h_7	h_8	h_9

X – flip

h_3	h_2	h_1
h_6	h_5	h_4
h_9	h_8	h_7

Y – flip

f_1	f_2	f_3
f_4	f_5	f_6
f_7	f_8	f_9

$*$

h_9	h_8	h_7
h_6	h_5	h_4
h_3	h_2	h_1

$f * h = f_1h_9 + f_2h_8 + f_3h_7 + f_4h_6 + f_5h_5 + f_6h_4 + f_7h_3 + f_8h_2 + f_9h_1$

↳ It is a filtering operation
↳ expresses the amount of overlap of one function as it is shifted over another function
↳ is associative
$F \cdot (G \cdot I) = (F \cdot G) \cdot I$
↳ used in
↳ edge detection
↳ sharpening
↳ smoothing

-1	-1	-1
-1	8	-1
-1	-1	-1

g
h:
this filter highlights edges by subtracting local averages

PROS
↳ Associative
↳ multiple filters can be combined into one
↳ used in CNN as efficient

Correlation
A similar alternate → Linear Relationship
↳ is not flipped

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

f = Image
h = Kernel

f_1	f_2	f_3
f_4	f_5	f_6
f_7	f_8	f_9

$\otimes$

h_1	h_2	h_3
h_4	h_5	h_6
h_7	h_8	h_9

$f \otimes h = f_1h_1 + f_2h_2 + f_3h_3 + f_4h_4 + f_5h_5 + f_6h_6 + f_7h_7 + f_8h_8 + f_9h_9$

↳ compares similarity of 2 sets of data
↳ relatedness of the signals
↳ used in template matching

Averages
↳ Mean

$I = \frac{I_1 + I_2 + \dots I_n}{n} = \frac{\sum_{i=1}^{n} I_i}{n}$

↳ weighted mean

$I = \frac{W_1 I_1 + W_2 I_2 + \dots W_n I_n}{n} = \frac{\sum_{i=1}^{n} W_i I_i}{n}$

Linear filter Properties
1. Linearity: $filter(f_1 + f_2) = filter(f_1) + filter(f_2)$
2. Shift invariance: $filter(shift(f)) = shift(filter(f))$
↳ same behaviour regardless of pixel location
3. More Properties

Boundary effects
What about near the edge?
↳ Pixels near the edges do not have full neighbours

Solutions
↳ Padding (replicate, zero, reflection)
↳ Cropping (ignore edges)
↳ clip filter (black)
↳ wrap around
↳ copy edge
↳ reflect across edge

How big should the filter be?
↳ values at edges should be near 0
↳ Gaussians have infinite extent...
↳ set filter half width to about 3σ (rule of thumb for Gaussian)

μ: center point (mean)
$\sigma\uparrow$: short curve
$\sigma\downarrow$: tall curve
(sd)

Image Filtering
↳ compute function of local neighbourhood at each position
↳ It is used to
↳ enhance images
↳ denoise, resize, increase contrast, etc
↳ extract information from images
↳ texture, edges, distinctive points, etc
↳ Detect Patterns
↳ template matching

Filtering → fixes noise
↳ modify pixels based on function of neighbornood
↳ output → linear combination of neighbourhood pixels

Output: $F \times I$ (filter, intensity; aka kernel)

Q)

$$\begin{bmatrix} 1 & 3 & 0 \\ 2 & 10 & 2 \\ 4 & 1 & 1 \end{bmatrix} \otimes \begin{bmatrix} 1 & 0 & 1 \\ 1 & 0.1 & -1 \\ 1 & 0 & -1 \end{bmatrix} = \begin{bmatrix} & & \\ & 5 & \\ & & \end{bmatrix}$$

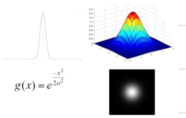
(I, K)

1x1 + 3x0 + 1x0 +
2x1 + 10x0.1 + 2x-1 +
4x1 + 1x0 + 1x-1

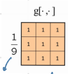
FILTERING EXAMPLES

A. Smoothing Filters
1. Gaussian Filter
2. Mean (Box) Filter

1. Gaussian Filter → KING OF SMOOTHING
↳ Provides natural blurring
PRO ↳ Preserves edge structure → better than mean filter



2. Box filter
↳ replaces each pixel with an avg of it's neighborhood
↳ It achives smoothness effect
↳ removes sharp features
CON ↳ reduces noise but blurs edges



normalized because

Filtering
↳ Sum of all pixels = 1
if not then normalize
$\frac{1}{N}$ → no of pixels

Gaussian Filter Properties
1. Most common natural model
2. Smooth function
↳ It has infinite no. of deratives
3. It is symmetric
4. Fourier Transform of Gaussian is Gaussian
5. Convolution of a Gaussian with itself is Gaussian
6. Gaussian is seperable
↳ 2D convolution can be performed by two 1D convolutions
7. There are cells in the eye that perform Gaussian filtering

B. Sharpening Filters

C. Edge Detection Filters
↳ Sobel Filtering

Sobel Filtering

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

↳ detects Gradients
↳ detects horizontal and vertical edges
↳ used in
↳ self driving cars
↳ object detection

D. Non Linear Filters → Handling Impulse Noise
↳ median filter

Median Filter → Saviour for Salt and Pepper noise
↳ replaces each pixel with median of it's neighbourhood
↳ operates over a window by selecting the median intensity in the window
↳ it is NOT the same as convolution
PRO ↳ retains edges better than mean box filtering

eg 1 1 2 (4) 5 6 6 → median

↳ used in fingerprint enhancement

Computational Efficiency
↳ convolutions are slow for large images
↓ Solution
↳ Fourier Transform
↳ converts convolution into multiplication
↳ used in MRI and astronomy

Filtering conclusion
1. Noise Removal
↳ Gaussian
↳ Mean
↳ Median
2. Sharpning
↳ Laplacian
↳ Unsharp Masking
3. Edge Detection
↳ Sobel
↳ Canny
4. Efficiency triks
↳ Fourier Transform

Q1) possible causes of artifctas such as jaggies

Possible Causes:
1.Low Resolution Rendering:
•The image might have been rendered at a low resolution, leading to visible pixelation and jagged edges.
2.Poor Anti-Aliasing:
•If the rendering system lacks proper anti-aliasing, edges will appear rough instead of smooth.
3.Incorrect Scaling:
•Improper resizing of the image can introduce artifacts.
4.Display or Hardware Issues:
•If displayed on a low-quality screen with poor pixel density or color depth, the image may appear jagged.
5.Compression Artifacts:
•If the image was heavily compressed, loss of detail could result in pixelation.

Solutions:
Hardware Solutions:
•Use a Higher Resolution Display:
A higher pixel density (DPI) display can help smooth out jagged edges.
•Upgrade Graphics Hardware:
A better GPU with proper anti-aliasing support can render smoother edges.
Software Solutions:
•Enable Anti-Aliasing:
Implement anti-aliasing techniques (SSAA) to smooth edges.
•Use Vector Graphics:
If possible, use scalable vector graphics (SVG) instead of raster images to avoid pixelation.
•Render at a Higher Resolution:
Generating the image at a higher resolution and then downscaling with proper interpolation can improve smoothness.
•Choose Proper Image Formats:
Use lossless formats (e.g., PNG, SVG) instead of lossy ones (e.g., JPEG) to avoid compression artifacts.

First order derivative
↳ diff b/w pixel intensity of neighbouring pixels → Masks

1. Backward diff → intensity $[-1, 1]$
I at current pixel − I at previous pixel
2. Forward diff $[1, -1]$
I at current pixel − I at next pixel
3. Central diff $[-1, 0, 1]$
I at next pixel − I at previous pixel

Edge Detection

↳ identify points in an image where intensity changes sharply

Characterizing Edges

↳ an edge is a point of rapid intensity change in an image

↳ first derivative → 1∂

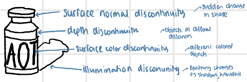
↳ measures the rate of intensity change

↳ second derivative → 2∂

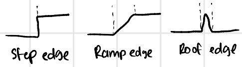
↳ measures the rate of change of gradient

Origin of Edges

↳ edges are caused by variety of factors



Types of Edges



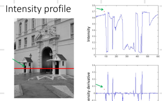
Why edge detection

↳ to extract useful info from images

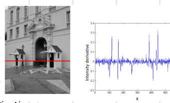
recognising objects

↳ Recover geometry

Intensity Profile



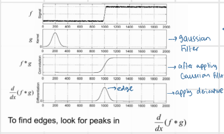
↳ with a little Gaussian Noise



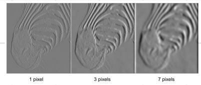
Smoothing

↳ removes noise

CON ↳ but blurs edges

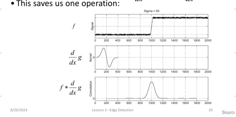


To find edges, look for peaks in $\frac{d}{dx}(f * g)$



Derevative theorem of convolution

- Convolution is differentiable: $\frac{d}{dx}(f*g) = f * \frac{d}{dx} g$
- This saves us one operation:



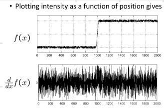
Effects of Noise in Edge Detection

↳ noise causes false edges as high frequency noise appears as sudden intensity variations

↓ solution

apply smoothing before differentiation

- Plotting intensity as a function of position gives a signal



Where is the edge?

Conclusion:
Noise in the graphs would be visible as random, irregular fluctuations in the intensity plot, making it harder to identify the true edges in the image

Edge Detectors

↳ Prewitt
↳ Sobel
↳ Marr-Hildreth
↳ Canny

Evaluate Edge Detection

$\text{Precision} = \frac{\text{Rel retrived}}{\text{Total retrived}}$ → TP

$\text{Recall} = \frac{\text{rel retrived}}{\text{Relevant}}$ → TP

Design Criteria for Edge Detection

↳ Good detection

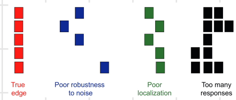
↳ find all real edges

↳ ignoring noise or other artifacts

↳ Good localization

↳ as close as possible to the true edges

↳ one point only for each true edge point



Prewitt and Sobel Edge Detector

↳ Compute Derivatives

↳ in x and y direction

↳ Find gradient magnitude

↳ Threshold gradient magnitude

1. Prewitt Edge Detector

↳ calculates gradient of image intensity which highlights regions where there are sharp changes in intensity → edges

Steps

1. Apply kernels $\begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{bmatrix}$ $\begin{bmatrix} 1 & 1 & 1 \\ 0 & 0 & 0 \\ -1 & -1 & -1 \end{bmatrix}$
2. Calculate gradient magnitude (edge strength) at each pixel $\Delta \sqrt{(\frac{\delta}{\delta x} I)^2 + (\frac{\delta}{\delta y} I)^2}$

↳ pixels with high gradient magnitudes are edges

CON

↳ sensitive to noise

2. Sobel Edge Detector

↳ similar to Prewitt

↳ adds higher weight to central pixel

Steps

1. Apply kernels $\begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}$ $\begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}$
2. Calculate gradient magnitude (edge strength) at each pixel $\Delta \sqrt{(\frac{\delta}{\delta x} I)^2 + (\frac{\delta}{\delta y} I)^2}$

↳ pixels with high gradient magnitudes are edges

PRO

↳ better noise supression

↳ more robust for edge detection

45 year boundary

1. Canny
2. Prewitt
3. Sobel

3. Marr Hildreth Edge Detector

(aka Lapacian of Gaussion (LOG))

↳ combines Gaussian smoothing + Lapacian operator to detect edges in an image

Steps

1. smooth image by Gaussian Filter → image is smoothed to remove noise
2. Apply Laplacian → highlights regions of rapid intensity change → edges
3. Find zero crossings → detects edges by finding points where Lapacian output crosses 0

↳ scan along each row

↳ record an edge point at the location of 0 crossing

↳ Repeat above step along each column

PROS

↳ detects both bright and dark edges

↳ can detect edges at different scales

CON

↳ sensitive to noise, may detect false edges

LOG Filter

$$\Delta^2 G_\sigma = \frac{-1}{\sqrt{2\pi}\,\sigma^3}\left[2 - \frac{x^2+y^2}{\sigma^2}\right] e^{-\frac{x^2+y^2}{2\sigma^2}}$$

1. $S I = G \times I$ (smoothed image; gaussian filter; og image)
2. $\Delta^2 S = \frac{\delta^2}{\delta x^2} S + \frac{\delta^2}{\delta y^2} S$ (Laplacian)
3. zero crossing
$[+,-]$ $[+,0,-]$
$[-,+]$ $[-,0,+]$

↳ slope $= |a+b|$

↳ apply threshold to slope

Laplacian of Gaussian (LoG):
- The combination of Gaussian smoothing and the Laplacian operator is called the Laplacian of Gaussian (LoG).
- LoG is defined as the second derivative of the Gaussian function

Second Derivative

↳ maxima minima of first derivative

↳ zero-crossings of 2nd derivative

Seperatability

- A filter is **separable** if it can be broken down into multiple, simpler filters that are applied sequentially.
- For example, a 2D filter can often be separated into two 1D filters (one for the horizontal direction and one for the vertical direction).

The 2D Gaussian function is separable, meaning it can be expressed as the product of two 1D Gaussian functions:

$G(x,y) = G(x)\, G(y)$

4. Canny Edge Detector

PROS ↳ detects edges accurately while minimizing noise and false edges

PROS

↳ finds optimal edges

↳ reduces false edges

↳ provides good localization → how close detected edges to true edges

CON

↳ more expensive

Steps

1. Smooth image with Gaussian Filter → to reduce noise that cause false edge detection
2. Compute derivative of filtered image
3. find magnitude and orientation of gradient → apply diff color to edges in same direction
4. Apply Non max Supression → removes weak edges by only keeping local maxima
5. Apply Thresholding (Hysteresis)

1. **Noise Reduction (Gaussian Smoothing)**:
   - The first step is to smooth the image using a **Gaussian filter** to reduce noise.
   - This helps prevent false edge detection caused by noise.
2. **Gradient Calculation**:
   - Compute the gradient of the image intensity using a gradient operator (e.g., Sobel or Prewitt).
   - This gives two gradient images:
     - **Gx**: Gradient in the horizontal direction (detects vertical edges).
     - **Gy**: Gradient in the vertical direction (detects horizontal edges).
4. **Non-Maximum Suppression**:
   - This step thins the edges by keeping only the local maxima in the gradient magnitude image.
   - For each pixel, the algorithm checks if it is the maximum along the gradient direction. If not, it is suppressed (set to zero).
5. **Double Thresholding**:
   - Two thresholds are applied to the gradient magnitude image:
     - **High threshold**: Strong edges are kept.
     - **Low threshold**: Weak edges are kept only if they are connected to strong edges.
   - This helps distinguish between strong edges, weak edges, and noise.

Edge Tracking by Hysteresis:
•Weak edges that are connected to strong edges are retained as part of the final edge map.
•Weak edges that are not connected to strong edges are discarded (considered noise).

Effect of Gaussian kernel (smoothing)

Smoothing with a Gaussian kernel **reduces noise** but also **blurs** the image.
The amount of smoothing depends on the **σ** value.

A smaller σ (standard deviation) value results in
-less smoothing.
-detects fine features and
-smaller edges

A larger σ value results in
-more smoothing
- detects large-scale edges and
-ignores smaller details.

- **Effect of σ (Standard Deviation):**
  - **Small σ**: Better for detecting fine details but more sensitive to noise.
  - **Large σ**: Better for detecting large-scale edges but may miss small details.
- **Application:**
  - The choice of σ depends on the desired behavior:
    - Use a **small σ** if you want to detect fine features.
    - Use a **large σ** if you want to detect large-scale edges and reduce noise.

1 **a. Original Image**:
- This is the starting image before any operations are applied.
- It shows the raw input image with no modifications.

2 **b. Laplacian Operator**:
- The Laplacian operator is used for **edge detection**.
- It highlights regions of rapid intensity change in the image, which typically correspond to edges.
- The result is an image where edges are emphasized, and smooth regions are suppressed.

3 **c. Horizontal Derivative**:
- This represents the result of applying a **horizontal derivative filter** (e.g., Sobel or Prewitt filter in the horizontal direction).
- It detects edges that are oriented **vertically** (i.e., horizontal intensity changes).

4 **d. Vertical Derivative**:
- This represents the result of applying a **vertical derivative filter** (e.g., Sobel or Prewitt filter in the vertical direction).
- It detects edges that are oriented **horizontally** (i.e., vertical intensity changes).

Question:
We have a signal with a roof edge (width of 5 pixels). If we use a smoothing kernel of size 11 with a **median filter** instead of a Gaussian filter, will we **not be able to detect any edges**? (True/False)?

Step-by-Step Reasoning:

1 **Roof Edge Definition**:
- A **roof edge** is a gradual transition in intensity (not a sharp step edge). It has a width of 5 pixels in this case.

2 **Median Filter**:
- A median filter is a **non-linear smoothing filter** that replaces each pixel's value with the median of the neighboring pixels (within the kernel size).
- It is effective at removing **noise** while preserving edges, especially in cases of salt-and-pepper noise.

3 **Kernel Size (11)**:
- The kernel size is **11 pixels**, which is significantly larger than the width of the roof edge (5 pixels).

4 **Effect of Median Filter on Roof Edge**:
- Since the kernel size (11) is larger than the roof edge width (5), the median filter will **blur the roof edge**.
- The gradual transition of the roof edge will be smoothed out, making it harder to detect.

5 **Conclusion**:
- With a median filter of size 11, the roof edge will be **blurred to the point where it may become undetectable**.

Answer:

True – With such a large median filter, we will not be able to detect the roof edge.

Interest Point

is a location in an image that is:
Highly distinctive: It stands out from its neighbors.
Stable under transformations: It remains recognizable despite changes in scale, rotation, or lighting.
Useful for matching across images.

Properties of Interest Point Detectors

A good interest point detector should:
✅ Detect **most** true interest points.
✅ Avoid **false** detections.
✅ Be **precise** (well localized).
✅ Be **robust** to noise, lighting, and transformations.
✅ Be **efficient** (fast enough for real-time applications).

Keypoint Matching

Detect Points that are
↳ Repeatable
↳ Distinctive

enable matching across images.
1. Detect Keypoints – Find corners or blobs.
2. Define a Region – Extract a neighborhood around the keypoint.
3. Normalize – Adjust for scale and rotation.
4. Compute Descriptor – Convert the patch into a unique vector.
5. Match Descriptors – Compare descriptors from different images.

Choosing Interest Points

Local Features

↳ the main components are

1) **Detection**: Identify the interest points
2) **Description** :Extract feature vector descriptor surrounding each interest point.
3) **Matching**: Determine correspondence between descriptors in two views

Goal

1. Interest Operator Repeatability
   ↳ Detecting the same points in both the images
2. Descriptor Distinctiveness
   ↳ Determine reliably which point goes to which

Symmetric & Real Interest Points

↳ Corners are indicated in red

Types of Interest Points

Corners: Where two edges meet.
Blobs: Distinct, uniform regions (e.g., circles in images).
Edges: Sudden intensity changes.

How to Detect Interest Points

There are two main approaches:
1. **Brightness-Based Approach**
Uses image derivatives (rate of intensity change).
Works well for high contrast regions.
2. **Boundary-Based Approach**
Detects points at edge intersections.
Requires edge detection + curvature analysis.

Trade Off

More Repeatable
Detecting many points
Robust to noise
Works with fewer textures

More Distinctive
Detecting fewer, but unique points
Robust to transformations
Works with occlusion

Invariant local Features

never changing

A good feature should be invariant to transformations:

Translation – Interest points should move consistently.
Rotation – Descriptors should stay the same even if the image rotates.
Scaling – Features should be detectable at multiple scales.
Lighting Changes – Interest points should be robust to brightness changes.

Harris Corner Detector

-Corner point can be recognized in a window
-If we shift a small window in any direction, a corner should produce a large intensity change in all directions.

Maths of Harris Detector

uses eigenvalues to find corners.

1. Auto-correlation function

$$E(u,v)=\sum_{x,y}\underbrace{[I(x+u,y+v)}_{\text{shifted intensity}}-\underbrace{I(x,y)]^2}_{\text{intensity}}$$

Measures the change in intensity when shifting the image.

2. Taylor series Expansion

Approximate E(u,v) using the Taylor Series Expansion:

3. Compute Eigenvalues λ_1, λ_2

Corner: λ1 λ 2 are large.
Edge: One eigenvalue is large, the other is small.
Flat Region: Both eigenvalues are small.

4. Corner Response Function

R=det(M)−k (trace(M))^2

R > 0: Corner detected!
R < 0: Edge detected.
R = 0: Flat region.

Taylor Series

$f(x)$ Can be represented at point a in terms of its derivatives

$$f(x)=f(a)+(x-a)f_x+\frac{(x-a)^2}{2!}f_{xx}+\frac{(x-a)^3}{3!}f_{xxx}+\dots$$

Express $I(x+u,y+v)$ at (x,y):

$$I(x+u,y+v)=I(x,y)+I_x(x+u-x)+I_y(y+v-y)$$

$$I(x+u,y+v)=I(x,y)+I_xu+I_yv$$

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

Taylor Series of right side

$$f(x,y,t)=f(x,y,t)+\frac{\partial f}{\partial x}(x+dx-x)+\frac{\partial f}{\partial y}(y+dy-y)+\frac{\partial f}{\partial t}(t+dt-t)$$

$$0=f_xdx+f_ydy+f_tdt$$

$$0=f_x\frac{dx}{dt}+f_y\frac{dy}{dt}+f_t\frac{dt}{dt}$$

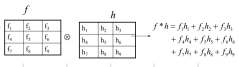
$$0=f_xu+f_yv+f_t \qquad f_xu+f_yv+f_t=0$$

Optical Flow Constrain Equation

Correlation → f x h

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

f = Image
h = Kernel



$$f*h=f_1h_1+f_2h_2+f_3h_3+f_4h_4+f_5h_5+f_6h_6+f_7h_7+f_8h_8+f_9h_9$$

$$f\otimes h=\sum_k\sum_l f(k,l)h(k,l)$$ Cross correlation

$$f\otimes f=\sum_k\sum_l f(k,l)f(k,l)$$ Auto correlation

Aperture Problem

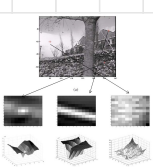
intrest Point Problem

arises when we observe motion through a small window (aperture).
If we can only see a small part of an image, we may misinterpret the true direction of motion.

The problem exists because edges have ambiguous motion perception.

Auto correlation → f x f

measures how similar a signal (or image) is to a shifted version of itself.



Cross correlation

measures the similarity between two signals (or images) as one is shifted over the other.

Measures how similar the template is to each part of the image. Used in template matching to find objects in an image.

Sum of Squared Differences (SSD)

↳ measures how similar 2 patches are

- SSD measures **how similar two patches** are:

$$SSD=\sum_{x,y}(I(x,y)-J(x,y))^2$$

- **Cross-Correlation** compares patterns:

$$R=\sum_{x,y}I(x,y)\cdot J(x,y)$$

- Used in **template matching.**

Correlation Vs SSD

$$\underset{\text{minimize}}{SSD}=\sum_k\sum_l(f(k,l)-h(k,l))^2$$ Sum of Squares Difference

$$\underset{\text{minimize}}{SSD}=\sum_k\sum_l\left(f(k,l)^2-2h(k,l)f(k,l)+h(k,l)^2\right)$$

$$\underset{\text{minimize}}{SSD}=\sum_k\sum_l(-2h(k,l)f(k,l))$$ These terms do not depend on correlation

$$\underset{\text{maximize}}{SSD}=\sum_k\sum_l(2h(k,l)f(k,l))$$

$$\underset{\text{maximize}}{Correlation}=\sum_k\sum_l(h(k,l)f(k,l))$$

$$f\otimes f=\sum_k\sum_l f(k,l)f(k,l)$$

Tradeoffs:
SSD is faster, but sensitive to intensity changes.
Correlation is more robust, but slower.

Real-World Examples:

✅ **Facial Recognition**: Detects keypoints like eyes, nose, and mouth corners.
✅ **Self-Driving Cars**: Identifies road markings, traffic signs, and objects.
✅ **3D Reconstruction**: Matches interest points across multiple views to infer depth.

1. Object Tracking
Track consistent features across video frames.
Example: Tracking a player in a sports video.
2. Stereo Vision (Depth Estimation)
Match points across left and right images to estimate depth.
Used in 3D reconstruction.
3. Image Stitching (Panoramas)
Detect interest points and match them across overlapping images.
Example: Creating a wide-angle panorama from multiple images.
4. Feature-Based Object Recognition
Detect key points and describe them with feature descriptors.
Example: Recognizing landmarks in images.

Maths of Harris Detector

- Change of intensity for the shift (u,v)

$$E(u,v)=\sum_{x,y}\underbrace{[I(x+u,y+v)}_{\text{shifted intensity}}-\underbrace{I(x,y)]^2}_{\text{intensity}}$$

Auto-correlation

$$E(u,v)=\sum_{x,y}\underbrace{[I(x+u,y+v)}_{\text{shifted intensity}}-\underbrace{I(x,y)]^2}_{\text{intensity}}$$

$$E(u,v)=\sum_{x,y}\underbrace{[I(x,y)+uI_x+vI_y}_{\text{shifted intensity}}-\underbrace{I(x,y)]^2}_{\text{intensity}}$$ Taylor Series

$$E(u,v)=\sum_{x,y}[uI_x+vI_y]^2$$

$$E(u,v)=\sum_{x,y}\left[\begin{pmatrix}u & v\end{pmatrix}\begin{pmatrix}I_x\\I_y\end{pmatrix}\right]^2$$

$$E(u,v)=\sum_{x,y}\begin{pmatrix}u & v\end{pmatrix}\begin{pmatrix}I_x\\I_y\end{pmatrix}\begin{pmatrix}I_x & I_y\end{pmatrix}\begin{pmatrix}u\\v\end{pmatrix}$$

$$M=\sum_{x,y}\begin{bmatrix}I_xI_x & I_xI_y\\I_xI_y & I_yI_y\end{bmatrix}$$

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

$$E(u,v)=\begin{pmatrix}u & v\end{pmatrix}M\begin{pmatrix}u\\v\end{pmatrix}$$

a) Given two points x=(2,4) and y=(3,7), find the line passing through the two points using homogenous coordinates and cross product

↳ Convert to homogenous

$u = (2,4,1)$

$y = (3,7,1)$

↳ cross product

$L = u \cdot y$

↳ find determinant

$$L = \begin{vmatrix} i & j & k \\ 2 & 4 & 1 \\ 3 & 7 & 1 \end{vmatrix}$$

$d = i((4\times1)-(7\times1)) - j(2-3) + k(14-12)$

$= -3i + j + 2k$

↳ form line eq

$L = (-3, 1, 2)$

$-3x + y + 2 = 0$

$3x - y - 2 = 0$

b) Derive the rotation matrix R such that a=Rb, where image a is the rotated version of image b in non-homogenous or conventional cartesian coordinates.

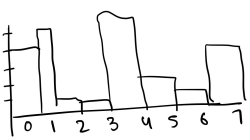
$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

↳ Generalise as $a = Rb$

$$R = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix}$$

Given the following grayscale image with each pixel represented by three bits and a 3x3 filter.

1	4	5	6	7
7	7	0	0	4
7	7	0	0	2
1	1	4	4	3
1	1	4	4	5



1. How would you add Gaussian noise to an image? Please use the 5×5 image shown below as an example and add Gaussian noise with a mean of 0 and a standard deviation (std) of 1.

image = p.array([
[100, 150, 200, 250, 100],
[50, 75, 125, 175, 225],
[0, 50, 100, 150, 200],
[25, 100, 175, 250, 50],
[100, 150, 200, 250, 150],
type=n.float32)

Gaussian Random noise

$N(\mu, \sigma^2) = N(0,1)$

$$\begin{bmatrix} 0.5 & -0.8 & 1.2 & -0.3 & 0.1 \\ -0.2 & 0.7 & -1.1 & 0.6 & -0.9 \\ 0.4 & -0.5 & 0.2 & -0.3 & 0.8 \\ \vdots & \vdots & \vdots & \vdots & \vdots \end{bmatrix} = \begin{bmatrix} \text{ANS} \end{bmatrix}$$

2. What would you do to increase the impact of noise so that the final noisy image is highly distorted?

To increase the impact of noise and make the final image highly distorted, you can:
- Increase the standard deviation (std) of the Gaussian noise, making the noise values larger and more disruptive.
- Scale the noise by multiplying it with a higher factor before adding it to the image.
- Apply the noise multiple times iteratively to amplify its effect.
- Reduce the image's original intensity range, making the noise more dominant.

These methods will result in a more distorted and less recognizable image.

B) Which of the images D and E has a higher contrast? Why? (4 points)
D has a higher contrast since it has a wider histogram, covering a wider range of intensities.
C) How does image B look like compared to image A? Explain. (4 points)
B is generally darker than A, as its histogram has been shifted to the left. But, both have about the same contrast.
D) What is the relation between images C and D? Explain your answer. (4 points) C is D with inverted intensities (negative of D). Mathematically C = 255 - D.

Filter		
-1	0	2
0	2	1
-3	1	2

Image				
0	0	1	3	2
2	3	6	0	3
5	4	2	5	7
1	2	0	4	4

↳ median

0 1 2 2 2 3 4 5 6

$$\begin{bmatrix} X & X & X & X & X \\ X & 2 & 3 & 3 & X \\ X & 2 & 3 & 4 & X \\ X & X & X & X & X \end{bmatrix}$$

↳ correlation

$(2\times-1)+(3\times0)+(6\times2)+(5\times0)+(4\times2)+(2\times1)+(1\times-3)+(2\times1)+(0\times2) = 19$

$$\begin{bmatrix} X & X & X & X & X \\ X & 7 & 18 & 19 & X \\ X & 19 & 8 & 29 & X \\ X & X & X & X & X \end{bmatrix}$$

↳ convolution

↳ flip filter horizontally then vertically

$$h = \begin{bmatrix} 2 & 0 & -1 \\ 1 & 2 & 3 \\ 2 & 1 & 3 \end{bmatrix} \rightarrow v = \begin{bmatrix} 2 & 1 & -3 \\ 1 & 2 & 0 \\ 2 & 0 & -1 \end{bmatrix}$$

= same steps =

$$\begin{bmatrix} X & X & X & X & X \\ X & 13 & 10 & 2 & X \\ X & 4 & 20 & 11 & X \\ X & X & X & X & X \end{bmatrix}$$

Sobel Filters (for correlation)		
-1	0	1
-2	0	2
-1	0	1

dx

-1	-2	-1
0	0	0
1	2	1

dy

Image				
0	0	1	3	2
2	3 (A)	6	0	3
5	4	2 (B)	5 (C)	7
1	2	0	4	4

For the image I above compute the gradient (dI/dx, dI/dy), magnitude of gradient and the angle of gradient for pixel locations A, B and C marked below. You have to use the Sobel filters (above) to compute the derivatives (no normalization is needed, that is you do not need to divide by 8 after applying the sobel filter). You need to write down how you derived each of these quantities. You may use arctan to represent the angles. The y axis is pointing downwards. (15 points)

A) For the image I above compute the **gradient** $(dI/dx, dI/dy)$, **magnitude of**

at Pixel A

$$\begin{bmatrix} 0 & 0 & 1 \\ 2 & 3 & 6 \\ 3 & 4 & 2 \end{bmatrix}$$

1. Compute Derivatives

$dI/dx = (-1\times0)+(0\times0)+(1\times1)+(-2\times2)+(0\times3)+(2\times6)+(-1\times5)+(0\times4)+(1\times2) = 6$

$dI/dy = (-1\times0)+(-2\times0)+(-1\times1)+(0\times2)+(0\times3)+(0\times6)+(1\times5)+(2\times4)+(1\times2) = 14$

2. Gradient Magnitude

$\|\nabla I\| = \sqrt{(dI/dx)^2 + (dI/dy)^2} = \sqrt{6^2+14^2} =$

3. Gradient Angle

$\theta = \tan^{-1}(dI/dy \,/\, dI/dx) = \tan^{-1}(14/6) =$

Repeat for B, C

$I_x = dI/dx$				
3	2	1	-1	-1
4	3	2	0	-1
4	3	4	2	1
1	1	3	2	2

$I_y = dI/dy$				
2	3	1	1	-1
2	3	2	-1	1
2	4	4	1	2
-1	0	3	2	3

A) Compute the Harris matrix for the 3 by 3 window highlighted in the above images. The Harris matrix is defined as (10 points)

$$H = \sum_{(x,y)\in W} \begin{bmatrix} I_x(x,y)^2 & I_x(x,y)I_y(x,y) \\ I_x(x,y)I_y(x,y) & I_y(x,y)^2 \end{bmatrix}$$

where W is the window highlighted in the gradient images.

$I_x^2 \rightarrow h_{11} = 3^2+2^2+0^2+3^2+4^2+2^2+1^2+3^2+2^2 = 56$

$I_y^2 \rightarrow h_{22} = 3^2+2^2+(-1)^2+4^2+4^2+1^2+0^2+3^2+2^2 = 60$

$I_xI_y \rightarrow h_{12} = h_{21} = (3\times3)+(2\times2)+(0\times(-1))+(3\times4)+(4\times4)+(2\times1)+(1\times0)+(3\times3)+(2\times2) = 56$

$$H = \begin{bmatrix} 56 & 56 \\ 56 & 60 \end{bmatrix}$$

B) Compute the cornerness score C = det(H) – k trace(H)2 for the above window, where det and trace represent the determinant and trace (sum of diagonal elements) of a matrix. Use k = 0.04. (3 points)

$C = (56\times60)-(56\times56) - \underbrace{0.04}_{k}(\underbrace{56+60}_{\text{trace}})^2 = -314.24$

MID2

USES OF FEATURES

1. Matching
2. Structure from Motion
3. Panorama Stitching
4. Detection
5. Recognition
6. Tracking
7. Stero Estimation

Features

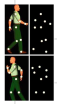
- info extracted from image/video
  - handcrafted → feature engineering (HOG, SIFT)
  - learned → automatically learned (CNN)

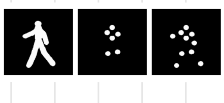
Types Of Features

1. Global Features
   - Extracted from the entire image
   - eg template, HOG
2. Region based Features (e.g. sliding window)
   - Extracted from a smaller window of image
   - apply global method to specific region
3. Local features
   - describe a Pixel and the area around the Pixel (e.g. SIFT, SURF)
   - always refer to specific Pixel locations

Finding Features in videos

- Complex actions can be recognized on the basis of 'point-light displays',
- Facial expressions,
- Sign Language,
- Arm movements,
- Various full-body actions.





Characteristics of Good Features

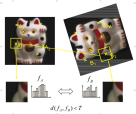
1. Distinctiveness
   - each feature can be uniquely identified
2. Repeatability → extract same features each time
   - The same feature can be detected in several images despite
     - Geometrically (viewpoint, scale)
     - Photometrically (brightness, exposure)
3. Compactness and efficiency → less memory
   - many fewer features than image pixels
   - run independently per image

- We want the representation to be as small and as fast as possible
  - Much smaller than a whole image
- We'd like to be able to run the detection procedure independently per image
  - Match just the compact descriptors for speed.
- Difficult! We don't get to see 'the other image' at match time, e.g., object detection.

Keypoint Matching

enable matching across images.

1. Detect Keypoints – Find corners or blobs.
2. Define a Region – Extract a neighborhood around the keypoint.
3. Normalize – Adjust for scale and rotation.
4. Compute Descriptor – Convert the patch into a unique vector.
5. Match Descriptors – Compare descriptors from different images.

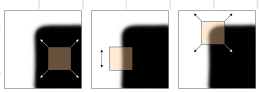


Keypoint Goal

- Detect points that are
  - repeatable
  - distinctive

Corner Detection

-Corner point can be recognized in a window
-If we shift a small window in any direction, a corner should produce a large intensity change in all directions.



"Flat" region: no change in all directions

"Edge": no change along the edge direction

"Corner": significant change in all directions

Template Matching

$$\begin{bmatrix} -4 & 5 & 5 \\ -4 & 5 & 5 \\ -4 & -4 & -4 \end{bmatrix} \quad \begin{bmatrix} 5 & 5 & 5 \\ -4 & 5 & -4 \\ -4 & -4 & -4 \end{bmatrix}$$

$$\begin{bmatrix} -4 & 5 & 5 \\ -4 & 5 & 5 \\ -4 & -4 & -4 \end{bmatrix} \quad \begin{bmatrix} 5 & 5 & 5 \\ -4 & 5 & -4 \\ -4 & -4 & -4 \end{bmatrix}$$

Complete set of eight templates can be generated by successive 90 degree of rotations.

Why the summation of filter is 0? → Insensitive to absolute change in intensity!

→ f x f

Correlation → f x h

f → image, h → kernel

$$f \otimes h = f_1h_1 + f_2h_2 + f_3h_3 + f_4h_4 + f_5h_5 + f_6h_6 + f_7h_7 + f_8h_8 + f_9h_9$$

Corner detection

Three different cases

$E(u,v)$

$E(u,v)$

s a surface

Corner Detection: strategy

Approximate $E(u,v)$ locally by a quadratic surface

≈

Corner Detection by Auto-correlation

Change in appearance of window $w(x,y)$ for shift $[u,v]$:

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

Window function | Shifted intensity | Intensity

Window function $w(x,y)$ = 1 in window, 0 outside / or Gaussian

$E(u,v)$ values
White → large shift
Black → small shift

$$E(u,v) \approx \begin{bmatrix} u & v \end{bmatrix} \left(\sum w(x,y) \begin{bmatrix} I_x^2 & I_xI_y \\ I_xI_y & I_y^2 \end{bmatrix} \right) \begin{bmatrix} u \\ v \end{bmatrix}$$

We want to find out how this function behaves for small shifts

$E(u,v)$

But this is very slow to compute naively.
O(window_width² * shift_range² * image_width²)

O(11² * 11² * 600²) = 5.2 billion of these 14.6k ops per image pixel

But we know the response in E that we are looking for – strong peak.

$$E(u,v) \approx \begin{bmatrix} u & v \end{bmatrix} M \begin{bmatrix} u \\ v \end{bmatrix}$$

where M is a *second moment matrix* computed from image derivatives:

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

$$M = \begin{bmatrix} \sum I_xI_x & \sum I_xI_y \\ \sum I_xI_y & \sum I_yI_y \end{bmatrix} = \sum \begin{bmatrix} I_x \\ I_y \end{bmatrix} \begin{bmatrix} I_x & I_y \end{bmatrix} = \sum \nabla I (\nabla I)^T$$

Recall: Taylor series expansion

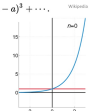
A function f can be represented by
- an infinite series of its derivatives at a single point a:

$$f(a) + \frac{f'(a)}{1!}(x-a) + \frac{f''(a)}{2!}(x-a)^2 + \frac{f'''(a)}{3!}(x-a)^3 + \cdots,$$

Wikipedia

As we care about window centered, we set $a = 0$ (MacLaurin series)

Approximation of f(x) = eˣ centered at f(0)



I_x and I_y are derivatives

Harris corner detection

$$E(u,v) \approx \begin{bmatrix} u & v \end{bmatrix} \left(\sum w(x,y) \begin{bmatrix} I_x^2 & I_xI_y \\ I_xI_y & I_y^2 \end{bmatrix} \right) \begin{bmatrix} u \\ v \end{bmatrix}$$

$$\begin{bmatrix} 2 & 1 & 2 \\ 1 & 2 & 0 \\ 0 & 1 & 1 \end{bmatrix} \begin{matrix} \text{Image} \\ \text{Gradients } I_x: \\ \text{Image} \\ \text{Gradients } I_y: \end{matrix} \begin{bmatrix} 1 & 2 & 1 \\ 0 & 2 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

$$I_x^2: \begin{bmatrix} 1^2 & 2^2 & 1^2 \\ 0^2 & 2^2 & 1^2 \\ 1^2 & 1^2 & 1^2 \end{bmatrix} = \begin{bmatrix} 1 & 4 & 1 \\ 0 & 4 & 1 \\ 1 & 1 & 1 \end{bmatrix} \Rightarrow \sum = 14$$

$$I_y^2: \begin{bmatrix} 2^2 & 1^2 & 2^2 \\ 1^2 & 2^2 & 0^2 \\ 0^2 & 1^2 & 1^2 \end{bmatrix} = \begin{bmatrix} 4 & 1 & 4 \\ 1 & 4 & 0 \\ 0 & 1 & 1 \end{bmatrix} \Rightarrow \sum = 16$$

$$I_x \cdot I_y: \begin{bmatrix} 1\cdot 2 & 2\cdot 1 & 1\cdot 2 \\ 0\cdot 1 & 2\cdot 2 & 1\cdot 0 \\ 1\cdot 0 & 1\cdot 1 & 1\cdot 1 \end{bmatrix} \Rightarrow \sum = 12$$

$$M = \begin{bmatrix} \sum I_x^2 & \sum I_xI_y \\ \sum I_xI_y & \sum I_y^2 \end{bmatrix} = \begin{bmatrix} 14 & 12 \\ 12 & 16 \end{bmatrix}$$

a shift of $(u,v) = (1,1)$:

$$E(1,1) = \begin{bmatrix} 1 & 1 \end{bmatrix} \begin{bmatrix} 14 & 12 \\ 12 & 16 \end{bmatrix} \begin{bmatrix} 1 \\ 1 \end{bmatrix}$$

$$\begin{bmatrix} 14*1 + 12*1 \\ 12*1 + 16*1 \end{bmatrix} = \begin{bmatrix} 26 \\ 28 \end{bmatrix}$$

$$1*26 + 1*28 = 54$$

$$E(1,1) = 54$$

→ Taylor Series

Corners as distinctive interest points

$$M = \sum w(x,y) \begin{bmatrix} I_xI_x & I_xI_y \\ I_xI_y & I_yI_y \end{bmatrix}$$

2 x 2 matrix of image derivatives (averaged in neighborhood of a point)

Notation: $I_x \Leftrightarrow \frac{\partial I}{\partial x}$ $I_y \Leftrightarrow \frac{\partial I}{\partial y}$ $I_xI_y \Leftrightarrow \frac{\partial I}{\partial x}\frac{\partial I}{\partial y}$

0. Input image
   We want to compute M at each pixel.
1. Compute image derivatives (optionally, blur first).
2. Compute M components as squares of derivatives.
3. Gaussian filter $g()$ with width s
   $= g(I_x^2),\ g(I_y^2),\ g(I_x \circ I_y)$
4. Compute cornerness
   $C = \det(M) - \alpha\ \text{trace}(M)^2 = C: \lambda_1\lambda_2 - \alpha(\lambda_1+\lambda_2)^2$
   $= g(I_x^2) \circ g(I_y^2) - g(I_x \circ I_y)^2 - \alpha[g(I_x^2) + g(I_y^2$
5. Threshold on C to pick high cornerness
6. Non-maximal suppression to pick peaks.

James Hays

CMPS415 - Lecture 9 47

C<0 edge $\lambda_1 \gg \lambda_2$ large, small

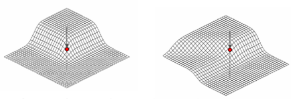
C>0 corner λ_1, λ_2 large, large

C≈0 flat λ_1, λ_2 small

Interest Point

is a location in an image that is:
Highly distinctive: It stands out from its neighbors.
Stable under transformations: It remains recognizable despite changes in scale, rotation, or lighting.
Useful for matching across images.

- Expressive texture
  - The point at which the direction of the boundary of object changes abruptly
  - Intersection point between two or more edge segments

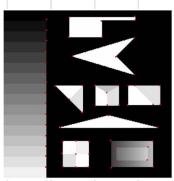


Types of Interest Points

Corners: Where two edges meet.
Blobs: Distinct, uniform regions (e.g., circles in images).
Edges: Sudden intensity changes.

Synthetic & Real Interest Points

- Corners are indicated in red



How to Detect Interest Points

There are two main approaches:

1. **Brightness-Based Approach**
   Uses image derivatives (rate of intensity change).
   Works well for high contrast regions.
2. **Boundary-Based Approach**
   Detects points at edge intersections.
   Requires edge detection + curvature analysis.

Properties of Interest Point Detectors

A good interest point detector should:

- ✅ Detect **most** true interest points.
- ✅ Avoid **false** detections.
- ✅ Be **precise** (well localized).
- ✅ Be **robust** to noise, lighting, and transformations.
- ✅ Be **efficient** (fast enough for real-time applications).

HISTOGRAM OF GRADIENTS (HOG)

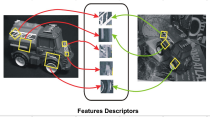
HOG
- algorithm that produces feature descriptor used for human detection

SCALE INVARIANT FEATURE TRANSFORM (SIFT)

SIFT
- Image content is transformed into local feature coordinates
- Invariant local features

Invariant local Features
A good feature should be invariant to transformations:

Translation – Interest points should move consistently.
Rotation – Descriptors should stay the same even if the image rotates.
Scaling – Features should be detectable at multiple scales.
Lighting Changes – Interest points should be robust to brightness changes.



Features Descriptors

Overall Procedure at a High Level

Scale-Space Extrema Detection	Search over multiple scales and image locations
KeyPoint Localization	Fit a model to determine location and scale. Select KeyPoints based on a measure of stability.
Orientation Assignment	Compute best orientation(s) for each keyPoint region.
KeyPoint Description	Use local image gradients at selected scale and rotation to describe each keyPoint region.

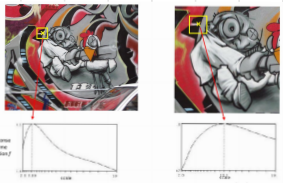
10/3/2024 CAPS415 - Lecture 9 81

Automatic Scale Selection

$$f(I_{i_1 \ldots i_m}(x,\sigma)) = f(I_{i_1 \ldots i_m}(x',\sigma'))$$

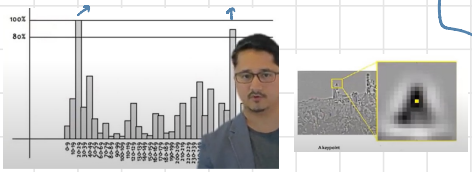
How to find patch sizes at which f response is equal?

What is a good f?



10/3/2024 CAPS415 - Lecture 9 88

Gradient of Orientation Histogram



HOG: Oriented Gradients
B) Given image I pixel location (i,j)
Compute the HOG feature for that pixel

steps
1. Extract a block of some size

2. Divide block into sq grid of sub blocks (cells)

2x2 grid so 4 cells

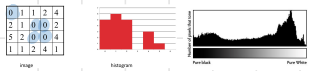
3. Compute orientation histogram of each cell
4. Concatenate the 4 histograms
V = vector of concatenation of 4 histograms
5. Normalize V
- There are 3 ways to normalize
  1. Divide V by its Euclidean norm
  2. Divide V by its L_1 norm
  3. Divide V by its Euclidean norm
     - then clip any value over 0.2 in the resulting vector
     - Then renormalize it by dividing again by its Euclidean

HOG: Human Detection

Navneet Dalal and Bill Triggs "Histograms of Oriented Gradients for Human Detection" CVPR05

HOG

Revist Histogram



CREATING Histogram of Gradients

Histogram of Oriented Gradients

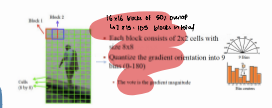
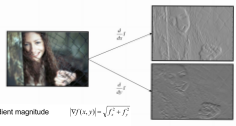


Image gradients



Gradient magnitude $|\nabla f(x,y)| = \sqrt{f_x^2 + f_y^2}$

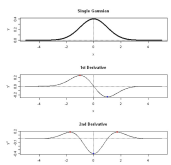
Gradient direction $\theta = \tan^{-1}\frac{f_y}{f_x}$

Histograms of Oriented Gradients

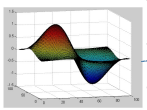
Parameters and design options:
- Angles range from 0 to 180 or from 0 to 360 degrees?
  - In the Dalal & Triggs paper, a range of 0 to 180 degrees is used,
  - and HOGs are used for detection of pedestrians.
- Number of orientation bins.
  - Usually 9 bins, each bin covering 20 degrees.
- Cell size.
  -Cells of size 8×8 pixels are often used.
- Block size.
  -Blocks of size 2x2 cells (16×16 pixels) are often used.

Usually a HOG feature has 36 dimensions.
- 4 cells * 9 orientation bins.

What Is A Useful Signature Function f?

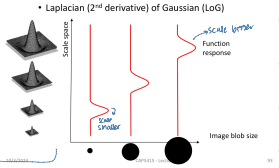


1st Derivative of Gaussian

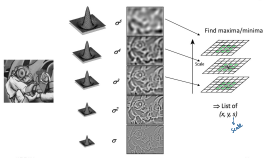


(Laplacian of Gaussian)

"Blob" detector is common for corners
- Laplacian (2nd derivative) of Gaussian (LoG)

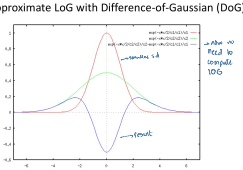


Find local maxima in position-scale space



Alternative kernel

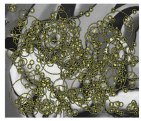
Approximate LoG with Difference-of-Gaussian (DoG).



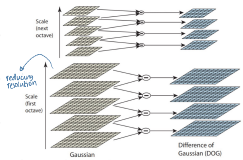
Approximate LoG with Difference-of-Gaussian (DoG).
1. Blur image with σ Gaussian kernel
2. Blur image with $k\sigma$ Gaussian kernel
3. Subtract 2. from 1.

Results: Difference-of-Gaussian

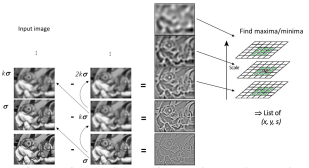
- Larger circles = larger scale
- Descriptors with maximal scale response



Scale-space



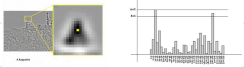
Find local maxima in position-scale space of DoG



MULT LAYER PERCEPTRON

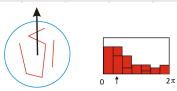
SIFT Orientation Estimation

- Compute gradient orientation histogram
- Select dominant orientation θ



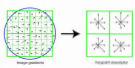
SIFT Orientation Normalization

- Compute gradient orientation histogram
- Select dominant orientation θ
- Normalize: rotate to fixed orientation



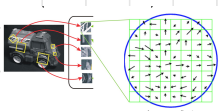
Review: Local Descriptors

- Most features can be thought of as
  - templates,
  - histograms (counts),
  - or combinations
- The ideal descriptor should be
  - Robust and Distinctive
  - Compact and Efficient
- Most available descriptors focus on edge/gradient information
  - Capture texture information
  - Color rarely used

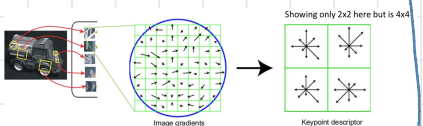


SIFT Descriptor formation

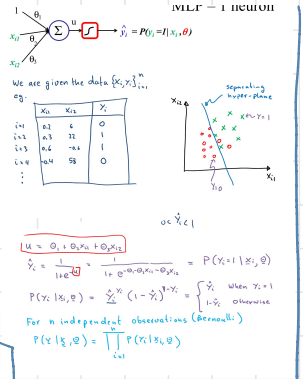
- Compute on local 16x16 window around detection
- Rotate and scale window according to discovered orientation θ and scale σ
- Compute gradients weighted by a Gaussian of variance half the window (for smooth falloff)



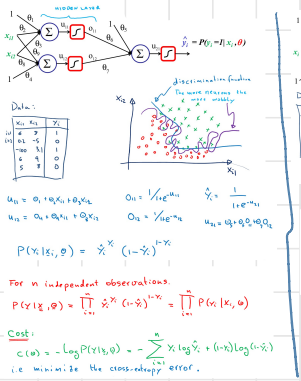
- 4x4 array of gradient orientation histogram weighted by gradient magnitude
- Bin into 8 orientations
  8 x 4x4 array = 128 dimensions



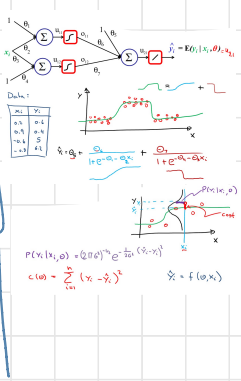
1. MLP - 1 Layer 1 neuron



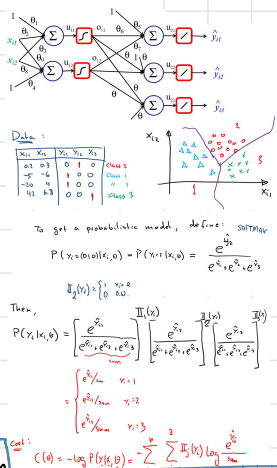
2. MLP - 2 layers 3 neurons



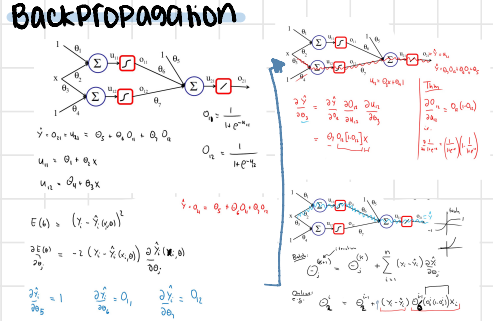
3. MLP - Regression



4. MLP - Multiclass

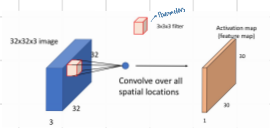


BackPropagation



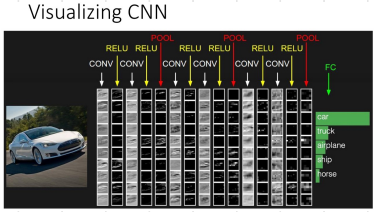
TRAINING NEURAL NETWORKS

Network Parameters - Recap

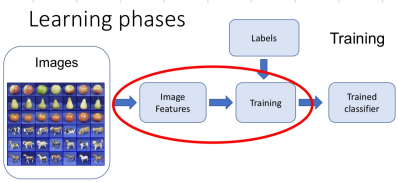


Convolution - Intuition

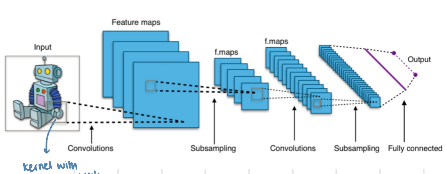
Visualizing CNN



Learning phases



General CNN architecture - recap



Loss Function

↳ tells how good the network performs in terms of prediction

↳ Network training (optimization)

↳ find the best network parameters → to minimize the loss

$$L(W) = \frac{1}{N}\sum_{i=1}^{N} L_i(f(x_i, W), y_i)$$

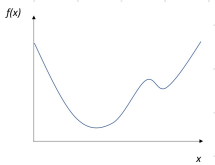
input, Ground truth, Loss function, Network parameters, network

Loss function measures how 'good' our network is at classifying the training examples wrt. the parameters of the model (the perceptron weights).

Network Training → Minimize cost

↳ Gradient Descent

↳ way to minimize a cost function



Gradient Descent

↳ Optimizer for functions

↳ Guaranteed to find optimum for convex functions

↳ non-convex ⟶ find local optimum → need some algorithms on the random weights

↳ most vision problems aren't convex

↳ Works for multi-variate functions

↳ need to compute matrix of partial derivatives

LOSS FUNCTION TYPES

1. Cross Entropy → for multiclass classification

↳ measures diff b/w 2 probability distributions

True Distribution (actual labels), Predicted Distribution

$$\mathcal{L} = -\frac{1}{N}\sum_{i=1}^{N}\left[y_i \log(\hat{y}_i) + (1-y_i)\log(1-\hat{y}_i)\right]$$

actual, predicted

2. Mean Squared Error (MSE)

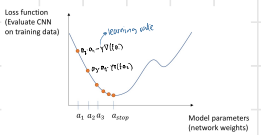
↳ used in regression tasks

↳ measures the avg of squared diff b/w actual and predicted values

$$\mathcal{L} = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2$$

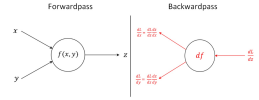
Train CNN with Gradient Descent

- x^i, y^i = n training examples
- $f(x)$ = feed forward network
- $L(x, y; \theta)$ = some *loss function*



Backpropagation - Chain Rule

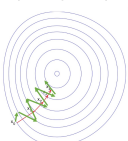
- Chain rule $\frac{\partial L}{\partial x} = \frac{\partial L}{\partial z}\frac{\partial z}{\partial x}$



- Add gate:
  - gradient distributor
- Max gate:
  - gradient router
- Mul gate:
  - gradient switcher

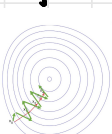
Gradient Descent Oscillations

Slow to converge to the (local) optimum



Lowering the learning rate

-Takes longer to get to the optimum



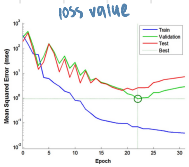
Problem of Fitting

↳ Too many parameters = overfitting

↳ Not enough parameters: underfitting

↳ more data: less chance to overfit

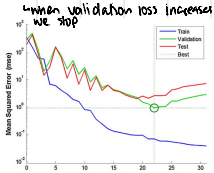
↳ How do we know what is required



Data Fitting Problem

1. Early Stopping

↳ when validation loss increases we stop



3. Dropout

↳ Stochastically switch neurons off

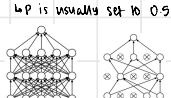
↳ each neuron is set to 0 → with probability p

↳ hidden units cannot co-adapt to each other

↳ Units are useful independently

↳ Hyperparameter

↳ p is usually set to 0.5



Stochastic Gradient Descent

↳ Dataset can be too large

↳ can not apply gradient descent wrt all data points

↳ Randomly sample a data point

↳ Perform gradient descent per sample and iterate

↳ Picking a subset of points: "mini batch" ⟶ batchsize

↳ Randomly initialize starting W and pick learning rate γ

While not at minimum:

↳ Shuffle training set

↳ for each data point i=1...n → maybe as mini-batch → Epoch

↳ Gradient Descent

↳ loss will not always decrease (locally)

↳ as training data point is random

↳ still converges over time

Learning rate

Momentum

↳ Adjust the gradient by a weighted sum of the prev amount plus the current amount.

↳ without momentum: $\theta_{t+1} = \theta_t - \gamma\frac{\partial L}{\partial \theta}$

↳ with momentum (new α parameter)

$\theta_{t+1} = \theta_t - \gamma\left(\alpha\left[\frac{\partial L}{\partial \theta}\right]_{t-1} + \left[\frac{\partial L}{\partial \theta}\right]_t\right)$

Regularization: Ensemble

↳ Networks start with random weights at every train = slightly different outcome

↳ Random weights are used cuz if weights are all equal, response across folders will be equivalent

↳ so network doesn't train

$w \cdot x = \sum_j w_j x_j$

2. Regularization

↳ Attempt to guide solution to not overfit

↳ But still give freedom with many parameters

↳ So basically

Penalise the use of parameters to prefer small weights

↳ Add a cost to having high weights

$$L(w) = \frac{1}{N}\sum_{i=1}^{N} L_i(f(x_i, W), y_i) + \lambda R(W)$$

Loss, Weight Decay, Regularization

- In common use,
  - L1 Norm $R(W) = \sum_i \sum_j |W_{ij}|$
  - L2 Norm $R(W) = \sum_i \sum_j W_{ij}^2$
  - Elastic net $R(W) = \sum_i \sum_j \beta W_{ij}^2 + |W_{ij}|$

Why not train 5 different networks with random starts and vote on their outcome?

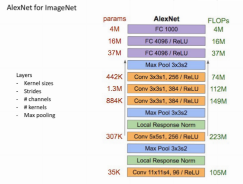
- Works fine!
- Helps generalization because error due to overfitting is averaged; reduces variance.

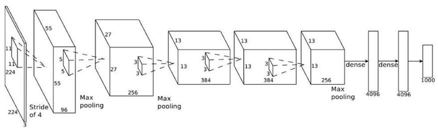
Training Steps

- Define network
- Loss function
- Initialize network parameters
- Get training data
  - Prepare batches
- Feedforward one batch
  - Compute loss
  - Backpropagate gradients
  - Update network parameters
  - Repeat

AlexNet - Training

Parameters:
- First use of ReLU
- Dropout 0.5
- Batch size 128
- Optimizer SGD
- Momentum 0.9
- Learning rate 1e-2
- Decay – lr reduced by 10 manually when val accuracy plateaus
- L2 weight decay 5e-4





Conv 1	Conv 2	Conv 3	Conv 4	Conv 4
11 x 11 x 3	5 x 5 x 48	3 x 3 x 256	3 x 3 x 192	3 x 3 x 192
Stride 4	Stride 1	Stride 1	Stride 1	Stride 1
96 filters	256 filters	384 filters	384 filters	256 filters

CATEGORY OF CNNs

1. Residual Networks

- Deep networks performs worse
  - As we add more layers
- Problem (solves this problem) as loss and gradient too small or too big
  - Vanishing gradients → gradients diminish/explode
- It models
  - H(x) = F(x) + x
- Skip connections
  - Help in backpropagation

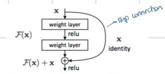
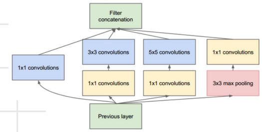


Figure 2. Residual learning: a building block.

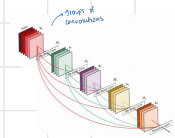
2. GoogleNet - Inception

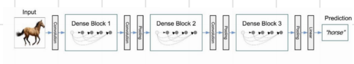
- ResNet is about going deeper
- Inception is about going wider
- Focused on computational efficiency
- The network learns
  - Which features are useful



3. DenseNet

- Densely Connected Convolutional Network
  - Similar to ResNet
  - Concat features instead of summation
  - A layer is passed all previous maps
  - Sequence of dense blocks





Huang, Gao, et al. "Densely connected convolutional networks." (2016).

CNN AND LEARNING RATE

Bias

error caused because the model lacks the ability to represent the (complex) concept

Variance

error caused because the learning algorithm overreacts to small changes (noise) in the training data

Total Loss = Bias + Variance (+ noise)

Address Overfitting for NN
- Use larger training data set
- Design better network architecture

Universality Theorem

↳ Any continuous function f

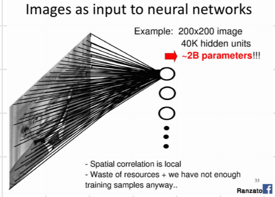
$f: R^N \rightarrow R^M$

can be realized by a network with one hidden layer → given enough hidden neurons

Universality is Not Enough
- Neural network has very high capacity (millions of parameters)
- By our basic knowledge of bias-variance tradeoff, so many parameters should imply very low bias, and very high variance. The test loss may not be small.
- Many efforts of deep learning are about mitigating overfitting!

Convolutional NN (CNN)

Images as input to neural networks



CNN = a multi-layer neural network with
- Local connectivity:
  - Neurons in a layer are only connected to a small region of the layer before it
- Share weight parameters across spatial positions:
  - Learning shift-invariant filter kernels

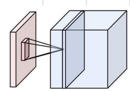
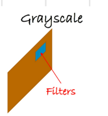


Image credit: A. Karpathy

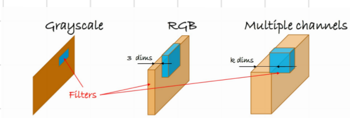
2D Spatial Features

- If images are 2-D, parameters should also be organized in 2-D
  - That way they can learn the local correlations between input variables
  - That way they can "exploit" the spatial nature of images

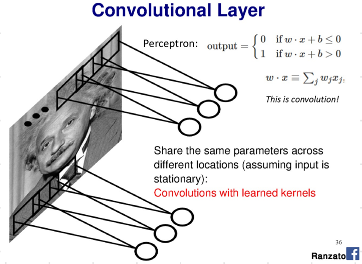


K-D Spatial Features

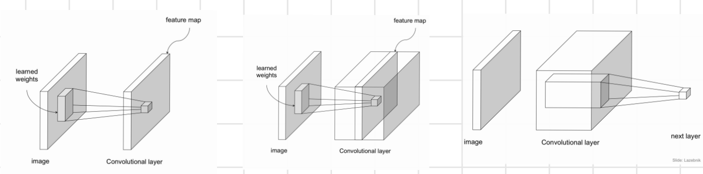
- Similarly, if images are k-D, parameters should also be k-D



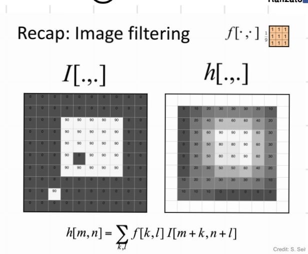
Convolutional Layer



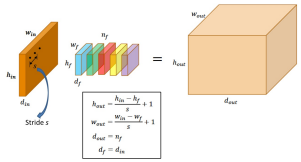
Dimensions of convolution



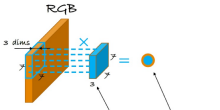
Recap: Image filtering

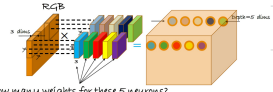


Dimensions of convolution



Number of weights





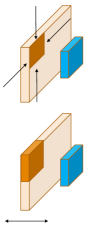
- Question: Spatial structure?
- Answer: Convolutional filters
- Question: Huge input dimensionalities?
- Answer: Parameters are shared between filters
- Question: Local variances?
- Answer: Pooling

Local connectivity

- The weight connections are surface-wise local!
  - Local connectivit

The weights connections are depth-wise global

- For standard neurons no local connectivity
  - Everything is connected to everything



Pooling operations

- Aggregate multiple values into a single value
- Invariance to small transformations
  - Keep only most important information for next layer
- Reduces the size of the next layer
  - Fewer parameters, faster computations
- Observe larger receptive field in next layer
  - Hierarchically extract more abstract features

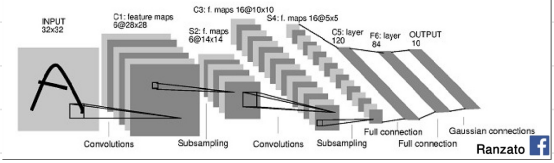
Single depth slice

1	1	2	4
5	6	7	8
3	2	1	0
1	2	3	4

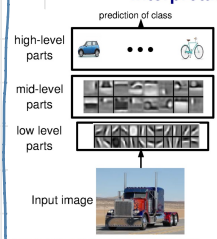
max pool with 2x2 filters and stride 2

6	8
3	4

Yann LeCun's MNIST CNN architecture



Interpretation



- distributed representations
- feature sharing
- compositionality

LEARNING NN

1. Setting Hyperparameters

Idea #1: Choose hyperparameters that work best on the data

BAD: big network always works perfectly on training data

Your Dataset

Idea #2: Split data into **train** and **test**, choose hyperparameters that work best on test data

BAD: No idea how algorithm will perform on new data

train | test

Idea #3: Split data into **train**, **val**, and **test**; choose hyperparameters on val and evaluate on test

Better!

train | validation | test

2. Select Optimizer

Stochastic gradient descent

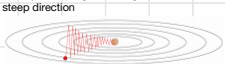
Gradient from entire training set:

$$\nabla C = \frac{1}{n}\sum_n \nabla C_n$$

- For large training data, gradient computation takes a long time
  - Leads to "slow learning"
- Instead, consider a mini-batch with m samples
- If sample size is large enough, properties approximate the dataset

$$\frac{\sum_{j=1}^{m} \nabla C_{X_j}}{m} \approx \frac{\sum_{n} \nabla C_n}{n} = \nabla C$$

Q) What if the loss function has a local minima or saddle point?
A) Zero gradient, gradient descent gets stuck Stochastic gradient descent
Q) What if loss changes quickly in one direction and slowly in another?
Q) What does gradient descent do?
A) Very slow progress along shallow dimension, jitter along steep direction



Loss function has high condition number: ratio of largest to smallest singular value of the Hessian matrix is large

Many variations of using momentum

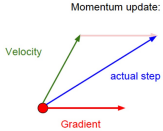
- In PyTorch, you can manually specify the momentum of SGD
- Or, you can use other optimization algorithms with "adaptive" momentum, e.g., ADAM
- ADAM: Adaptive Moment Estimation
- Empirically, ADAM usually converges faster, but SGD gives local minima with better generalizability

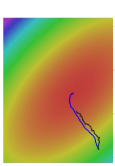
Our gradients come from minibatches so they can be noisy!

$$L(W) = \frac{1}{N}\sum_{i=1}^{N} L_i(x_i, y_i, W)$$

$$\nabla_W L(W) = \frac{1}{N}\sum_{i=1}^{N} \nabla_W L_i(x_i, y_i, W)$$

Momentum update:





SGD

$$x_{t+1} = x_t - \alpha \nabla f(x_t)$$

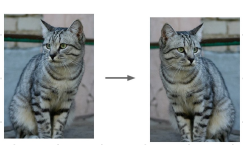
SGD+Momentum

$$v_{t+1} = \rho v_t + \nabla f(x_t)$$
$$x_{t+1} = x_t - \alpha v_{t+1}$$

Build up velocity as a running mean of grad

3. Data Augmentation

Horizontal flips



Random crops and scales

1. Pick random L in range [256, 480]
2. Resize training image, short side = L
3. Sample random 224 x 224 patch

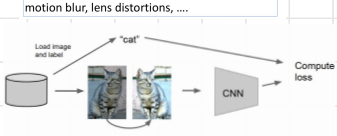
Color jitter

Simple: Randomize contrast and brightness

More Complex:

1. Apply PCA to all [R, G, B] pixels in training set
2. Sample a "color offset" along principal component directions
3. Add offset to all pixels of a training image

Can do a lot more: rotation, shear, non-rigid, motion blur, lens distortions, ...



Supervised vs unsupervised learning

Supervised Learning

Data: (x, y)
x is data, y is label

Goal: Learn function to map
$x \rightarrow y$

Examples: Classification, regression, object detection, semantic segmentation, etc.

Unsupervised Learning

Data: x
x is data, no labels!

Goal: Learn the *hidden* or *underlying structure* of the data

Examples: Clustering, feature or dimensionality reduction, etc.

Generative modeling

Goal: Take as input training samples from some distribution and learn a model that represents that distribution

How can we learn $P_{model}(x)$ similar to $P_{data}(x)$?

Why generative models? Debiasing

Capable of uncovering **underlying features** in a dataset

Homogeneous skin color, pose

Diverse skin color, pose, illumination

How can we use this information to create fair and representative datasets?

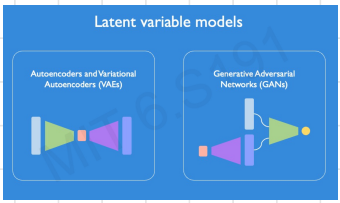
Software Lab!

- Problem: How can we detect when we encounter something new or rare?
- Strategy: Leverage generative models, detect outliers in the distribution
- Use outliers during training to improve even more!

Why generative models? Sample generation
- Generative models learn probability distributions
- Sampling from that distribution -> new data instances
- Backbone of Generative AI:

generate language, images, and more

Latent variable models

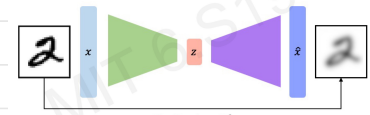


Autoencoders: background

Unsupervised approach for learning a **lower-dimensional** feature representation from unlabeled training data

"Encoder" learns mapping from the data, x, to a low-dimensional latent space, z

How can we learn this latent space?
Train the model to use these features to reconstruct the original data
"Decoder' learns mapping back from latent space, z, to a reconstructed observation, £

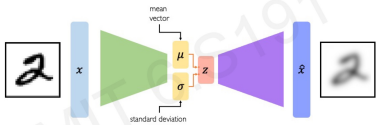


Dimensionality of latent space → reconstruction quality

Autoencoding is a form of compression!
Smaller latent space will force a larger training bottleneck

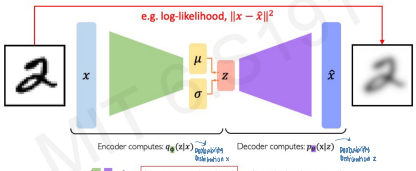
Autoencoders for representation learning
•Bottleneck hidden layer forces network to learn a compressed latent representation
•Reconstruction loss forces the latent representation to capture (or encode) as much "information" about the data as possible
• Autoencoding = Automatically encoding data; "Auto" = self-encoding

VAEs: key difference with traditional autoencoder

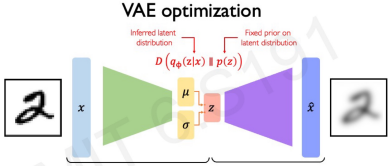


Variational autoencoders are a probabilistic twist on autoencoders!
Sample from the mean and standard deviation to compute latent sample

VAE optimization



VAE optimization



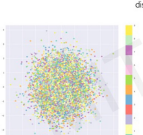
VAE summary
Compress representation of world to something we can use to learn
2. Reconstruction allows for unsupervised learning (no labels!)
3. Reparameterization trick to train end-to-end
4. Interpret hidden latent variables using perturbation
5. Generating new examples

Priors on the latent distribution

$$D\left(q_\phi(z|x) \,\|\, p(z)\right)$$

Inferred latent distribution

Fixed prior on latent distribution



Common choice of prior – Normal Gaussian:

$$p(z) = \mathcal{N}(\mu = 0, \sigma^2 = 1)$$

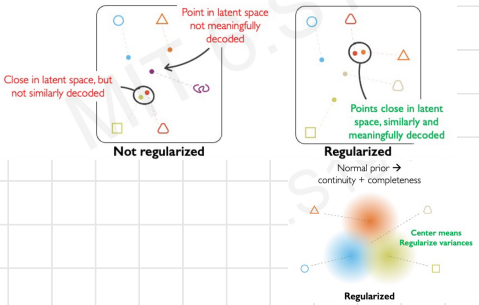
$$D\left(q_\phi(z|x) \,\|\, p(z)\right) = -\frac{1}{2}\sum_{j=1}^{k}(\sigma_j + \mu_j^2 - 1 - \log \sigma_j)$$

KL-divergence between the two distributions

- Encourages encodings to distribute encodings evenly around the center of the latent space
- Penalize the network when it tries to "cheat" by clustering points in specific regions (i.e., by memorizing the data)

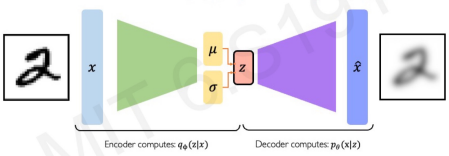
Intuition on regularization and the Normal prior

What properties do we want to achieve from regularization?
1. Continuity: points that are close in latent space -> similar content after decoding
2. Completeness: sampling from latent space - "meaningful" content after decoding



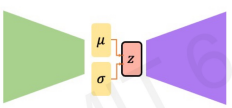
VAE computation graph

Problem: We cannot backpropagate gradients through sampling layers!



$\mathcal{L}(\phi, \theta, x)$ = (reconstruction loss) + (regularization term)

Reparametrizing the sampling layer



Key Idea:

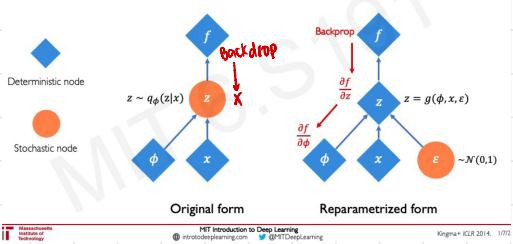
~~$z \sim \mathcal{N}(\mu, \sigma^2)$~~

Consider the sampled latent vector z as a sum of
- a fixed μ vector,
- and fixed σ vector, scaled by random constants drawn from the prior distribution

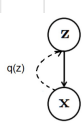
$$\Rightarrow z = \mu + \sigma \odot \varepsilon$$

where $\varepsilon \sim \mathcal{N}(0,1)$

Reparametrizing the sampling layer



- Approximate $p(z|x)$ by $q(z)$
- Minimize the KL Divergence:



$$D_{KL}\left[q(z)||p(z|x)\right] = -\int q(z) log \frac{p(z|x)}{q(z)} dz$$

VAEs: Latent perturbation

Slowly increase or decrease a single latent variable
Keep all other variables fixed

Head pose

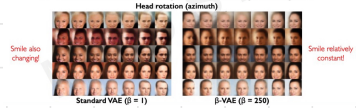
Different dimensions of z encodes **different interpretable latent features**

Latent space disentanglement with β-VAEs

~~β-VAE loss:~~

$$\mathcal{L}(\theta, \phi; \mathbf{x}, \mathbf{z}, \beta) = \underbrace{\mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})}[\log p_\theta(\mathbf{x}|\mathbf{z})]}_{\text{Reconstruction term}} - \beta \underbrace{D_{KL}(q_\phi(\mathbf{z}|\mathbf{x}) \parallel p(\mathbf{z}))}_{\text{Regularization term}}$$

β > 1: constrain latent bottleneck, encourage efficient latent encoding → disentanglement



Why latent variable models? Debiasing

Capable of uncovering **underlying latent variables** in a dataset

How can we use latent distributions to create fair and representative datasets?

Variational Inference

- Problem Definition
  - Observable Data: $x = \{x_1, x_2, ..., x_n\}$
  - Hidden Variable: $z = \{z_1, z_2, ..., z_n\}$

- Solutions
- Monte Carlo Sampling
  - Metropolis Hasting
  - Gibbs Sampling
- Variational Inference

$$p(z|x) = \frac{p(z,x)}{p(x)} = \frac{p(x|z)p(z)}{\int p(x|z)p(z)dz}$$

Variational Lower Bound

$$D_{KL}\left[q(z)||p(z|x)\right] = -\int q(z) log \frac{p(z|x)}{q(z)} dz$$

$$= -\int q(z) log \frac{p(z,x)}{q(z)p(x)} dz$$

$$= -\int q(z) log \frac{p(z,x)}{q(z)} dz + \int q(z) log\left(p(x)\right) dz$$

$$= -\int q(z)\left(log\left(p(z,x)\right) - log\left(q(z)\right)\right) dz + log\left(p(x)\right)$$

$$= -\underbrace{\left(E_{q(z)}\left[log\left(p(z,x)\right)\right] - E_{q(z)}\left[log\left(q(z)\right)\right]\right)}_{\text{Evidence Lower Bound (ELBO)}} + log\left(p(x)\right)$$

$$D_{KL}\left[q(z)||p(z|x)\right] = -L\left[q(z)\right] + log\left(p(x)\right)$$

$$log\left(p(x)\right) = D_{KL}\left[q(z)||p(z|x)\right] + L\left[q(z)\right]$$

Minimizing $D_{KL}\left[q(z)||p(z|x)\right]$
is equal to Maximizing $L\left[q(z)\right]$