CS131 Notes

Sean Wu

May 24, 2020

Contents

1 Introduction **5**
1.1 What is Computer Vision and why is it hard . . . 5
1.2 Definition of Vision and Comparisons to Human Vision . . . 5
1.3 Human Vision Strengths and Weaknesses . . . 5
1.4 Extracting info from images . . . 5
1.4.1 Measurement in Vision . . . 6
1.4.2 Obtaining Semantic Info from Vision . . . 6
1.5 Applications of Computer Vision . . . 6

2 Linear Algebra Review **6**
2.1 Vectors . . . 6
2.2 Matrix . . . 7
2.2.1 Images . . . 7
2.3 Basic Matrix Operations . . . 8
2.3.1 Addition . . . 8
2.3.2 Scaling . . . 8
2.3.3 Vector Norms . . . 8
2.3.4 Inner Product (Dot Product) . . . 9
2.4 Matrix Multiplication . . . 9
2.4.1 Properties of Matrix Multiplication . . . 9
2.5 Matrix Powers . . . 10
2.6 Matrix Transpose . . . 10
2.7 Determinant . . . 10
2.7.1 Properties of the determinant . . . 10
2.8 Trace . . . 10
2.8.1 Properties of trace . . . 11
2.9 Special Matrices . . . 11
2.9.1 Identity Matrix . . . 11
2.9.2 Diagonal Matrix . . . 11
2.9.3 Symmetric Matrix . . . 12
2.9.4 Skew-symmetric Matrix . . . 12
2.10 Transformation Matrices . . . 12

2.10.1 Scaling Transformation . . . 12
2.10.2 Converting to a rotated reference frame . . . 12
2.10.3 2D Rotation Matrix . . . 13
2.10.4 Normal Matrices . . . 13
2.10.5 Multiple Transformation Matrices . . . 13
2.11 Homogenous System . . . 14
2.11.1 Translation . . . 14
2.11.2 Division . . . 14
2.11.3 2D translation using Homogenous Coordinates . . . 15
2.11.4 Scaling Matrix in Homogenous Coordinates . . . 15
2.11.5 Scaling and Translating . . . 15
2.11.6 Scaling & Translating != Translating & Scaling . . . 15
2.11.7 Rotation Matrix in Homogenous Coordinates . . . 16
2.11.8 Scaling + Rotation + Translation . . . 16
2.12 Matrix Inverse . . . 16
2.12.1 Properties of the Matrix Inverse . . . 16
2.12.2 Pseudo Inverse . . . 17
2.13 Linear Independence . . . 17
2.14 Matrix Rank . . . 17
2.15 Eigenvector & Eigenvalues . . . 18
2.15.1 Properties of Eigenvectors and Eigenvalues . . . 18
2.15.2 Spectral Theory . . . 19
2.16 Diagonalization . . . 19
2.16.1 Symmetric Matrices . . . 20
2.16.2 Applications of Eigenvalues and Eigenvectors . . . 20
2.17 Matrix Calculus . . . 20
2.17.1 Gradient . . . 20
2.17.2 Properties of the Gradient . . . 21
2.18 Hessian Matrix . . . 22
2.19 Singular Value Decomposition . . . 23
2.19.1 SVD Applications . . . 23
2.19.2 Principal Component Analysis . . . 23
2.19.3 SVD Algorithm . . . 24

3 Colour Theory **24**
3.1 Colour . . . 24
3.2 Physics of Light . . . 24
3.3 Human Encoding of Colour . . . 25
3.3.1 Cones and Colours . . . 25
3.4 Colour Spaces . . . 25
3.4.1 Linear Colour Spaces . . . 25
3.4.2 RGB Space . . . 26
3.4.3 CIE XYZ Colour Space . . . 26
3.4.4 Nonlinear Colour Spaces - HSV . . . 27
3.5 White Balancing . . . 27

3.5.1 Importance of White Balancing . . . 28
3.5.2 Von Kries Method . . . 28
3.5.3 Other White Balancing Methods . . . 28
3.6 Colour in Computer Vision . . . 29

4 Pixels and Filters **29**
4.1 Images Types . . . 29
4.1.1 Binary Image Representation . . . 29
4.1.2 Grayscale . . . 29
4.1.3 Colour Image . . . 29
4.2 Sampling and Resolution . . . 30
4.3 Image Histograms . . . 30
4.4 Images as Functions . . . 31
4.5 Linear Systems (Filters) . . . 32
4.5.1 Filter 1: Moving Average (Smoothing) . . . 32
4.5.2 Filter 2: Image Segmentation . . . 33
4.6 Properties of Systems . . . 33
4.6.1 Amplitude Properties . . . 33
4.6.2 Spatial Properties . . . 33
4.7 Linear Systems . . . 34
4.7.1 Impulse Response . . . 34
4.7.2 Impulse Response Example: Moving Average Filter . . . 35
4.8 Convolution . . . 35
4.8.1 2D Convolution Example: Identity Filter . . . 36
4.8.2 2D Convolution Example: Shift Left Filter . . . 36
4.8.3 2D Convolution Example: Box Blur Filter . . . 37
4.8.4 2D Convolution Example: Sharpening Filter . . . 37
4.9 Implementation of Convolution: Image Support and Edge effects . . . 37
4.10 Cross Correlation ** . . . 38
4.10.1 Properties of Cross Correlation . . . 38
4.10.2 Convolution vs Cross Correlation . . . 38
4.11 Discrete Convolution * . . . 39

5 Edge Detection **39**
5.0.1 Why are Edges Useful . . . 39
5.0.2 Origins of Edges in Images . . . 39
5.1 Image Gradients . . . 40
5.1.1 1D Derivatives . . . 40
5.1.2 1D Discrete Derivatives and Filters . . . 40
5.2 2D Continuous Derivatives . . . 41
5.2.1 2D Discrete Gradients . . . 41
5.3 Simple Edge Detection . . . 42
5.3.1 Image Gradient . . . 43
5.4 Intensity Profile . . . 44
5.4.1 Effects of Noise . . . 44

5.4.2 1D Smoothing Filters . . . 44
5.4.3 Gaussian Blur . . . 45
5.5 Smooth First Edge Detection (Unoptimized) . . . 45
5.5.1 Derivative Theorem of Convolution . . . 46
5.5.2 Improved Smooth First Edge Detection . . . 46
5.6 Tradeoff between Smoothing at Different Scale σ . . . 46
5.7 Criteria for an Optimal Edge Detector . . . 46
5.8 Sobel Edge Detector . . . 47
5.8.1 Sobel Operator . . . 47
5.8.2 Sobel Operator Issues . . . 48
5.9 Canny Edge Detector . . . 48
5.9.1 Canny Edge Detector Features . . . 48
5.9.2 Non-Maximum Suppression . . . 48
5.9.3 Removing Unnecessary Gradients . . . 48
5.9.4 Hysteresis Thresholding . . . 49
5.9.5 Canny Edge Detector Algorithm . . . 49
5.10 Effect of Standard Deviation σ (aka scale) in Gaussian Filter . . . 49
5.11 Hough Transform . . . 49
5.11.1 Preparation before applying Hough Transform . . . 50
5.11.2 Detecting Lines using Hough Transform . . . 50
5.11.3 Hough Transform Algorithm . . . 51
5.11.4 Hough Transform in (ρ, θ) space) . . . 51
5.11.5 Hough Transform Advantages . . . 52
5.11.6 Hough Transform Disadvantages . . . 52

1 Introduction

1.1 What is Computer Vision and why is it hard

Computer Vision : extracting info from digital images OR developing algorithms to understand image content for other applications

- Computer Vision is a hard interdisciplinary problem that is still unsolved
  - Hard to convert data storing RGB values in many pixels to semantic info (ex. this blob of black pixels is a chair)
- Vision (extracting meaningful info) is harder than 3D modelling

1.2 Definition of Vision and Comparisons to Human Vision

sensing device : captures details from a scene

interpreting device : processes image from sensing device to extract meaning

- Humans use eyes as sensing devices while computers use cameras
- For sensing devices, computer vision is actually better than human vision because cameras can see infrared, have longer range, and capture greater detail
- For interpreting devices, the human brain is way more advanced than computer systems

1.3 Human Vision Strengths and Weaknesses

- Human vision evolved to quickly recognize danger for survival
- It is very fast $\longrightarrow \sim 150$ ms to recognize an animal
- For speed, humans *focus* only on "relevant" *areas of interest*
- Thus, small signals/changes in the background can be difficult to detect and segment
- Humans also use *context* to infer clues
  - Used to determine next area of focus, when to expect certain objects in certain positions, and colour compensation in shadows
  - However, context can be used to trick human vision
- Context is very hard to include in computer vision

1.4 Extracting info from images

- 2 types of info extracted in computer vision: **measurements** and **semantic info**

1.4.1 Measurement in Vision

- Robots scan surroundings to make a map of its environment
- Stereo vision gives depth information (like 2 eyes) using triangulation
  - Depth info represented as a depth map
- With multiple viewpoints of an object, a 3D surface can be created (or even a 3D model)

1.4.2 Obtaining Semantic Info from Vision

- Labelling objects (or scene)
- Recognizing people, actions, gestures, faces

1.5 Applications of Computer Vision

- Video special effects
- 3D object modelling
- Scene recognition
- Face detection
  - Note: face recognition is harder than face detection
- Optical Character Recognition (OCR)
- Reverse image search
- Vision based interaction (ex. Microsoft Kinect)
- Augmented reality
- Virtual reality

2 Linear Algebra Review

2.1 Vectors

- a *column vector* $\mathbf{v} \in \mathbb{R}^{n\times 1}$ where

$$\mathbf{v} = \begin{bmatrix} v_1 \\ v_2 \\ \vdots \\ v_n \end{bmatrix} \tag{1}$$

- a *row vector* $\mathbf{v}^T \in \mathbb{R}^{1\times n}$ where

$$\mathbf{v}^T = \begin{bmatrix} v_1 & v_2 & \dots & v_n \end{bmatrix} \tag{2}$$

- The transpose of a matrix/vector is denoted with a subscript T
- Note: with numpy in python, you can transpose a vector v with v.T

- In 2D and 3D, vectors have a geometric interpretation as points
- Can also use vectors to represent pixels, gradients at an image keypoint, etc
- In this use case, vectors do not have a geometric interpretation, but calculations like "distance" are still useful
  - The distance measures "similarity" between 2 vectors

2.2 Matrix

- A *matrix* $\mathbf{A} \in \mathbb{R}^{m \times n}$ is an array of numbers with size m by n
- i.e. m rows and n columns

$$\mathbf{A} = \begin{bmatrix} a_{11} & a_{12} & a_{13} & \dots & a_{1n} \\ a_{21} & a_{22} & a_{23} & \dots & a_{2n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ a_{d1} & a_{d2} & a_{d3} & \dots & a_{dn} \end{bmatrix} \tag{3}$$

- if $m = n$, we say that $\mathbf{A}$ is square

2.2.1 Images

- Python represents an *image* as a matrix of pixel brightnesses
- Note: the upper left corner has indices $\underbrace{[x, y]}_{\text{row, column}} = (0, 0)$
  - Python indices start at 0
  - MATLAB indices start at 1
- Images can be also be represented as a vector of pixels by stacking rows into a single tall column vector

grayscale image : 1 number per pixel; stored as a $m \times n$ matrix

color image : 3 numbers per pixel $\longrightarrow$ red, green, blue brightnesses (RGB); stored as a $m \times n \times 3$ matrix

2.3 Basic Matrix Operations

2.3.1 Addition

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

- Can only add matrices with matching dimensions or a scalar

2.3.2 Scaling

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix} * 3 = \begin{bmatrix} 3a & 3b \\ 3c & 3d \end{bmatrix} \tag{5}$$

2.3.3 Vector Norms

ℓ_1 **Norm - Manhattan Norm** $\|\mathbf{x}\|_1 = \sum_{i=1}^{n} |x_1|$

ℓ_2 **Norm - Euclidean Norm** $\|\mathbf{x}\|_2 = \sqrt{\sum_{i=1}^{n} x_i^2}$

ℓ_∞ **Norm - Max Norm** $\|\mathbf{x}\|_\infty = \max_i |x_i|$

ℓ_p **Norm** $\|\mathbf{x}\|_p = \left(\sum_{i=1}^{n} x_i^p \right)^{\frac{1}{p}}$

Matrix Norm $\|\mathbf{A}\|_F = \sqrt{\sum_{i=1}^{m} \sum_{j=1}^{n} A_{ij}^2} = \sqrt{\mathrm{tr}(\mathbf{A}^T \mathbf{A})}$

- Note: a matrix norm is a vector norm in a vector space whose elements (vectors) are matrices (of a given dimension)

- Formally, a **norm** is any $f : \mathbb{R}^n \to \mathbb{R}$ that satisfies these 4 properties

1. **Non-negativity**: $\forall \mathbf{x} \in \mathbb{R}^n, f(\mathbf{x}) \geq 0$
2. **Definiteness**: $f(\mathbf{x}) = 0 \iff \mathbf{x} = \begin{bmatrix} 0 & 0 & \dots & 0 \end{bmatrix}$

3. **Homogeneity**: $\forall \mathbf{x} \in \mathbb{R}^n, t \in \mathbb{R}, f(t\mathbf{x}) = |t| f(\mathbf{x})$
4. **Triangle Inequality**: $\forall \mathbf{x}, \mathbf{y} \in \mathbb{R}^n, f(\mathbf{x}+\mathbf{y}) \leq f(\mathbf{x}) + f(\mathbf{y})$

2.3.4 Inner Product (Dot Product)

- The **inner product (dot product)** $\mathbf{x} \cdot \mathbf{y}$ or $\mathbf{x}^T\mathbf{y}$ is calculated by multiplying the corresponding entries of 2 vectors and adding up the result
- Note: the inner product takes 2 vectors as input and outputs a single scalar

$$\mathbf{x} \cdot \mathbf{y} = |\mathbf{x}||\mathbf{y}| \cos(\theta) \tag{6}$$

$$\mathbf{x}^T\mathbf{y} = \mathbf{x} \cdot \mathbf{y} = \begin{bmatrix} x_1 & x_2 & \dots & x_n \end{bmatrix} \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix} = \sum_{i=1}^{n} x_i y_i \tag{7}$$

- if $\mathbf{y}$ is a unit vector, then $\mathbf{x} \cdot \mathbf{y} = |\mathbf{x}| \cos(\theta)$ gives the (signed) length of $\mathbf{x}$ which lies in the direction of $\mathbf{y}$

2.4 Matrix Multiplication

- Inner dimensions of matrices must match
- For $\mathbf{A} \in \mathbb{R}^{m \times n}$ and $\mathbf{B} \in \mathbb{R}^{n \times p}$, the product $\mathbf{C} = \mathbf{AB} \in \mathbb{R}^{m \times p}$ where $\mathbf{C}_{ij} = \sum_{k=1}^{n} A_{ik} B_{kj}$

$$\mathbf{C} = \mathbf{AB} = \begin{bmatrix} — & a_1^T & — \\ — & a_2^T & — \\ & \vdots & \\ — & a_m^T & — \end{bmatrix} \begin{bmatrix} | & | & & | \\ b_1 & b_2 & \dots & b_p \\ | & | & & | \end{bmatrix} = \begin{bmatrix} a_1^T b_1 & a_1^T b_2 & \dots & a_1^T b_p \\ a_2^T b_1 & a_2^T b_2 & \dots & a_2^T b_p \\ \vdots & \vdots & \ddots & \vdots \\ a_m^T b_1 & a_m^T b_2 & \dots & a_m^T b_p \end{bmatrix} \tag{8}$$

- i.e. matrix multiplication gives a matrix where the entries are the dot product of the rows of A and columns B

2.4.1 Properties of Matrix Multiplication

1. **Associative**: $(\mathbf{AB})\mathbf{C} = \mathbf{A}(\mathbf{BC})$
2. **Distributive**: $\mathbf{A}(\mathbf{B}+\mathbf{C}) = \mathbf{AB} + \mathbf{AC}$
3. **Not Commutative**: Generally, $\mathbf{AB} \neq \mathbf{BA}$

- ex. if $\mathbf{A} \in \mathbb{R}^{m \times n}$ and $\mathbf{B} \in \mathbb{R}^{n \times p}$, then the matrix product $\mathbf{BA}$ does not exist if $m \neq p$

2.5 Matrix Powers

matrix powers : repeated matrix multiplication of a matrix $\mathbf{A} \in \mathbb{R}^{n \times n}$ with itself

$$\mathbf{A}^2 = \mathbf{AA} \qquad \mathbf{A}^3 = \mathbf{AAA} \tag{9}$$

- Note: only *square* matrices can have powers because the dimensions must match

2.6 Matrix Transpose

matrix transpose : flip matrix across the main diagonal so that the rows become the columns, and vice versa

$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{bmatrix}^T = \begin{bmatrix} 1 & 3 & 5 \\ 2 & 4 & 6 \end{bmatrix} \tag{10}$$

- Identity: $(\mathbf{ABC})^T = \mathbf{C}^T\mathbf{B}^T\mathbf{A}^T$

2.7 Determinant

determinant : represents the area (or volume) of the parallelogram described by the vectors in the rows of the matrix

- Note: $\det(\mathbf{A})$ takes a matrix input and returns a scalar
- For $\mathbf{A} = \begin{bmatrix} a & b \\ c & d \end{bmatrix}$, $\det(\mathbf{A}) = ad - bc$

2.7.1 Properties of the determinant

1. $\det(AB) = \det(BA)$
2. $\det(A^{-1}) = \frac{1}{\det(1)}$
3. $\det(A^T) = \det(A)$
4. $\det(A) = 0 \iff A$ is singular

2.8 Trace

trace : sum of the main diagonal elements

$$\operatorname{tr}\left(\begin{bmatrix} 1 & 3 \\ 5 & 7 \end{bmatrix}\right) = 1 + 7 = 8 \tag{11}$$

- Note: the $\text{tr}(A)$ is only defined for square matrices
- $\text{tr}(A)$ is invariant to a lot of transformations so it is sometimes used in proofs

2.8.1 Properties of trace

1. $\text{tr}(AB) = \text{tr}(BA)$
2. $\text{tr}(A + B) = \text{tr}(A) + \text{tr}(B)$

2.9 Special Matrices

2.9.1 Identity Matrix

Identity Matrix : a square matrix $\mathbf{I} \in \mathbb{R}^{n \times n}$ with 1's along the main diagonal and 0's everywhere else

$$\mathbf{I} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{12}$$

- For any matrix $\mathbf{A}$ (with proper dimensions)
- $\mathbf{I} \cdot \mathbf{A} = \mathbf{A}$
- $\mathbf{A} \cdot \mathbf{I} = \mathbf{A}$
- i.e. matrix multiplication with $\mathbf{I}$ is commuative (special case)

2.9.2 Diagonal Matrix

Diagonal Matrix : a square matrix $\mathbf{D} \in \mathbb{R}^{n \times n}$ with scalars along the diagonal, 0's everywhere else

$$\mathbf{D} = \begin{bmatrix} 3 & 0 & 0 \\ 0 & 7 & 0 \\ 0 & 0 & 2.5 \end{bmatrix} \tag{13}$$

- For any matrix $\mathbf{B} \in \mathbb{R}^{n \times p}$, $\mathbf{DB}$ scales the rows of $\mathbf{B}$
- Note: the identity matrix $\mathbf{I}$ is a special diagonal matrix that scales all the rows by 1

2.9.3 Symmetric Matrix

Symmetric Matrix : $\mathbf{A}^T = \mathbf{A}$

$$\begin{bmatrix} 1 & 2 & 5 \\ 2 & 1 & 7 \\ 5 & 7 & 1 \end{bmatrix} \tag{14}$$

2.9.4 Skew-symmetric Matrix

Skew-symmetric Matrix : $\mathbf{A}^T = -\mathbf{A}$

$$\begin{bmatrix} 0 & -2 & -5 \\ 2 & 0 & -7 \\ 5 & 7 & 0 \end{bmatrix} \tag{15}$$

2.10 Transformation Matrices

Matrix transformation : transforms vectors by matrix multiplication: $\mathbf{Ax} = \mathbf{x}'$

2.10.1 Scaling Transformation

Scaling matrix : scales components of vector

$$\underbrace{\begin{bmatrix} S_x & 0 \\ 0 & S_y \end{bmatrix}}_{\text{Scaling Matrix}} * \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} S_x x \\ S_y y \end{bmatrix} \tag{16}$$

2.10.2 Converting to a rotated reference frame

Rotation Matrix : matrix that describes a rotation of a vector or equivalently changing to a rotated reference frame

- i.e. have the same data point but represent it in a new rotated frame
- Note: rotating a reference frame left == rotating a data point to the right
- Recall: a 2D vector stores a component in the x-direction and a component in the y-direction
- Thus the transformation for $\begin{bmatrix} x \\ y \end{bmatrix} \longrightarrow \begin{bmatrix} x' \\ y' \end{bmatrix}$ is found by computing the dot product of the original vector with the new unit vectors for the x'-direction and y'-direction
- Thus, the new coordinates $\begin{bmatrix} x' \\ y' \end{bmatrix}$ represent the length of the original vector lying in the direction of the new x-, y- axes

- Equivalently, can express the original x-, y- unit vectors in terms of the new x'-, y- unit vectors

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} \text{(new x'-axis)} \\ \text{(new y'-axis)} \end{bmatrix} * \begin{bmatrix} x \\ y \end{bmatrix} \tag{17}$$

$$= \begin{bmatrix} (\hat{x} \text{ in new x'-,y'- axes}) & (\hat{y} \text{ in new x'-,y'- axes}) \end{bmatrix} * \begin{bmatrix} x \\ y \end{bmatrix} \tag{18}$$

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \underbrace{\mathbf{R}}_{2 \times 2 \text{ Rotation Matrix}} * \begin{bmatrix} x \\ y \end{bmatrix} \tag{19}$$

$$\mathbf{P}' = \mathbf{RP} \tag{20}$$

2.10.3 2D Rotation Matrix

- For a CCW rotation of a point (aka a CW rotation of ref. frame)

$$x' = x\cos(\theta) - y\sin(\theta) \tag{21}$$

$$y' = x\sin(\theta) + y\cos(\theta) \tag{22}$$

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} \tag{23}$$

$$\mathbf{P}' = \mathbf{RP} \tag{24}$$

- Note: transpose of a rotation matrix produces a rotation in the opposite direction

2.10.4 Normal Matrices

- Note: $\mathbf{R}$ belongs to the category of **normal** matrices
- Properties of normal matrices

1. $\mathbf{RR}^T = \mathbf{R}^T\mathbf{R} = \mathbf{I}$
2. $\det \mathbf{R} = 1$

- Rows of a rotation matrix are always mutually perpendicular (aka orthogonal) unit vectors
- Same with columns

2.10.5 Multiple Transformation Matrices

- For multiple transformation matrices, the transformations are applied one by one from **right to left**

$$\mathbf{P}' = \mathbf{R}_2\mathbf{R}_1\mathbf{SP} \tag{25}$$

$$\mathbf{P}' = (\mathbf{R}_2(\mathbf{R}_1(\mathbf{SP}))) \tag{26}$$

- By associativity, the result is the same as multiplying the matrices first to form a single transformation matrix

$$\mathbf{P}' = (\mathbf{R}_2\mathbf{R}_1\mathbf{S})\mathbf{P} \tag{27}$$

- In general, matrix multiplication allows us to linearly combine components of a vector
- This is sufficient for scaling, rotating, skewing, but we <u>cannot</u> add a constant (not a linear operation)

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} ax + by \\ cd + dy \end{bmatrix} \tag{28}$$

2.11 Homogenous System

2.11.1 Translation

- Hacky Fix: can add translation by representing the problem in a higher $n + 1$ dimension and stick a 1 at the end of every vector

$$\begin{bmatrix} a & b & c \\ d & e & f \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} ax + by + c \\ dx + ey + f \\ 1 \end{bmatrix} \tag{29}$$

- Note: $\begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$ and $\begin{bmatrix} ax + by + c \\ dx + ey + f \\ 1 \end{bmatrix}$ are **homogenous coordinates**
- Now we can rotate, scale, skew, and translate
- Matrix multiplication with translation matrix results in adding the rightmost column of the translation vector to the original vector
- Generally, homogenous transformation matries have a bottom row of $\begin{bmatrix} 0 & 0 & 1 \end{bmatrix}$ so that the resulting vector $\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix}$ has a 1 at the bottom too

2.11.2 Division

- ex. want to divide a vector by a coordinate y_0 to make things scale down as they get farther away in a camera image
- Matrix multiplication can't actually divide so use this convention
- **Convention**: in homogenous coordinates, divide the resulting vector by its last coordinates after matrix multiplication

$$\begin{bmatrix} x \\ y \\ 7 \end{bmatrix} \longrightarrow \begin{bmatrix} \frac{x}{7} \\ \frac{y}{7} \\ 1 \end{bmatrix} \tag{30}$$

2.11.3 2D translation using Homogenous Coordinates

- $P = (x, y) \rightarrow (x, y, 1)$
- $T = (t_x, t_y) \rightarrow (t_x, t_y, 1)$

$$\mathbf{P}' = \begin{bmatrix} x + t_x \\ y + t_y \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} \mathbf{I} & \mathbf{t} \\ 0 & 1 \end{bmatrix} * \mathbf{P} \tag{31}$$

- Thus $\mathbf{P}' = \mathbf{T} \cdot \mathbf{P}$ where $\mathbf{T}$ is the translation matrix

2.11.4 Scaling Matrix in Homogenous Coordinates

- $P = (x, y) \rightarrow (s_x x, s_y y, 1)$
- $T = (t_x, t_y) \rightarrow (t_x, t_y, 1)$
- $P' = (x + t_x, y + t_y) \rightarrow (x + t_x, y + t_y, 1)$

$$\mathbf{P}' = \begin{bmatrix} s_x x \\ s_y y \\ 1 \end{bmatrix} = \underbrace{\begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{bmatrix}}_{\mathbf{S}} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} \mathbf{S}' & 0 \\ 0 & 1 \end{bmatrix} * \mathbf{P} \tag{32}$$

- Thus $\mathbf{P}' = \mathbf{S} \cdot \mathbf{P}$ where $\mathbf{S}$ is the scaling matrix

2.11.5 Scaling and Translating

- Recall: matrix transformations are applied right to left for $\mathbf{P}'' = \mathbf{TSP}$

$$\mathbf{P}'' = \mathbf{TSP} = \begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} s_x & 0 & t_x \\ 0 & s_y & t_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} \mathbf{S}' & \mathbf{t}' \\ 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} s_x x + t_x \\ s_y y + t_y \\ 1 \end{bmatrix} \tag{33}$$

2.11.6 Scaling & Translating != Translating & Scaling

- Recall: matrix multiplication is generally **not** commutative, so order matters
- If you scale after you translated, both the original vector and the translation will be scaled

2.11.7 Rotation Matrix in Homogenous Coordinates

- Rotation $\mathbf{P}' = \mathbf{R} \cdot \mathbf{P}$ in homogenous coordinates is the same as regular rotation, just with the extra 1 in the bottom row

$$\mathbf{P}' = \begin{bmatrix} x\cos(\theta) - y\sin(\theta) \\ x\sin(\theta) + y\cos(\theta) \\ 1 \end{bmatrix} = \underbrace{\begin{bmatrix} cos(\theta) & -\sin(\theta) & 0 \\ \sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 1 \end{bmatrix}}_{\mathbf{R}} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} \mathbf{R}' & 0 \\ 0 & 1 \end{bmatrix} * \mathbf{P} \tag{34}$$

2.11.8 Scaling + Rotation + Translation

$$\mathbf{P}' = (\mathbf{TRS})\mathbf{P} \tag{35}$$

$$= \begin{bmatrix} \mathbf{I} & \mathbf{t} \\ 0 & 1 \end{bmatrix} \begin{bmatrix} \mathbf{R} & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} \mathbf{S} & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \tag{36}$$

$$= \begin{bmatrix} \mathbf{RS} & \mathbf{t} \\ 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \tag{37}$$

- Therefore, the **general transformation matrix** is $\begin{bmatrix} \mathbf{RS} & \mathbf{t} \\ 0 & 1 \end{bmatrix}$

2.12 Matrix Inverse

- Given an invertible matrix $\mathbf{A}$, its inverse $\mathbf{A}^{-1}$ is a matrix such that $\mathbf{A}\mathbf{A}^{-1} = \mathbf{A}^{-1}\mathbf{A} = \mathbf{I}$
- ex. $\begin{bmatrix} 2 & 0 \\ 0 & 3 \end{bmatrix}^{-1} = \begin{bmatrix} \frac{1}{2} & 0 \\ 0 & \frac{1}{3} \end{bmatrix}$
- The inverse $\mathbf{A}^{-1}$ doesn't always exist
- If $\mathbf{A}^{-1}$ exists, $\mathbf{A}$ is **invertible** (aka **nonsingular**)
- Otherwise, it is **non-invertible/singular**

2.12.1 Properties of the Matrix Inverse

1. $(\mathbf{A}^{-1})^{-1} = \mathbf{A}$
2. $(\mathbf{AB})^{-1} = \mathbf{B}^{-1}\mathbf{A}^{-1}$
3. $\mathbf{A}^{-T} \triangleq (\mathbf{A}^T)^{-1} = (\mathbf{A}^{-1})^T$

2.12.2 Pseudo Inverse

- if inverse $\mathbf{A}^{-1}$ exists, we can solve $\mathbf{Ax} = \mathbf{b}$ with $\mathbf{x} = \mathbf{A}^{-1}\mathbf{b}$

```
np.linalg.inv(A) * b
```

- If inverse $\mathbf{A}^{-1}$ doesn't exist or the matrix is too large (too expensive to compute), we can use the pseudo-inverse to find $\mathbf{x}$

```
np.linalg.solve(A,b)
```

- Python will try several numerical methods (including pseudoinverse) and return solution for $\mathbf{x}$

– if no exact solution $\longrightarrow$ Python returns the closest value

– if many solutions $\longrightarrow$ Python returns the smallest one

2.13 Linear Independence

- For a set of vectors $\mathbf{v}_1, \ldots, \mathbf{v}_n$, if we can express $\mathbf{v}_1$ as a **linear combination** of other vectors $\mathbf{v}_2, \ldots, \mathbf{v}_n$, then $\mathbf{v}_1$ is **linearly dependent** on the other vectors
- ex. $\mathbf{v}_1 = 0.7\mathbf{v}_2 - 0.7\mathbf{v}_n$

Linearly independent set : no vector in a set is linearly dependent on the rest of the vectors

- ex. a set of vectors $\mathbf{v}_1, \ldots, \mathbf{v}_n$ is always linearly independent if each vector is perpendicular to every other vector (and nonzero)

2.14 Matrix Rank

Rank : the rank of a transformation matrix tells you how many dimensions it transforms a vector to; i.e. the dimensions of the output vecor

col-rank : number of linearly independent column vectors of $\mathbf{A}$

row-rank : number of linearly independent row vectors of $\mathbf{A}$

- Note: column rank always equals row rank

$$\text{rank}(\mathbf{A}) \triangleq \text{col-rank}(\mathbf{A}) = \text{row-rank}(\mathbf{A}) \tag{38}$$

- ex. if $\text{rank}(\mathbf{A}) = 1$, then the transformation $\mathbf{P}' = \mathbf{AP}$ maps points onto a line

$$\begin{bmatrix} 1 & 1 \\ 2 & 2 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} x + y \\ 2x + 2y \end{bmatrix} \tag{39}$$

- Here all the points are mapped to the line $y = 2x$

full rank : if an $m \times m$ matrix has rank m, we say it is full rank. It maps an $m \times 1$ vector uniquely to another $m \times 1$ vector. Also has an inverse matrix

singular : if an $m \times m$ matrix has rank $< m$, then at least one dimension is getting collapsed to zero. Thus there is no way to look at the output and find the input (not invertible)

- If an $m \times m$ matrix has **full rank** $\iff$ it is invertible

2.15 Eigenvector & Eigenvalues

Eigenvector : an eigenvector $\mathbf{x}$ of a linear transformation $\mathbf{A}$ is a nonzero vector that when $\mathbf{A}$ is applied to it, does not change direction

$$\mathbf{A}\mathbf{x} = \lambda\mathbf{x}, \qquad \mathbf{x} \neq 0 \tag{40}$$

- Applying $\mathbf{A}$ to an eigenvector only scales the eigenvector by the scalar value λ, called an **eigenvalue**
- An $m \times m$ matrix will have $\leq m$ eigenvectors where the eigenvalue λ is nonzero
- To find all eigenvalues of $\mathbf{A}$ solve this eqn for $\mathbf{x} \neq 0$

$$\mathbf{A}\mathbf{x} = \lambda\mathbf{x} \tag{41}$$
$$\mathbf{A}\mathbf{x} = (\lambda\mathbf{I})\mathbf{x} \tag{42}$$
$$(\mathbf{A} - \lambda\mathbf{I})\mathbf{x} = 0 \tag{43}$$

- Since $\mathbf{x} \neq 0$, $(\mathbf{A} - \lambda\mathbf{I})$ cannot be invertible/nonsingular and its determinant is zero (i.e. nonzero nullspace)

$$\det(\mathbf{A} - \lambda\mathbf{I}) = 0 \tag{44}$$

2.15.1 Properties of Eigenvectors and Eigenvalues

1. The trace of $\mathbf{A}$ is the sum of its eigenvalues

$$\mathrm{tr}(\mathbf{A}) = \sum_{i=1}^{n} \lambda_i \tag{45}$$

2. The determinant of $\mathbf{A}$ equal to product of its eigenvalues

$$\det(\mathbf{A}) = \prod_{i=1}^{n} \lambda_i \tag{46}$$

3. The rank of $\mathbf{A}$ is equal to the number of non-zero eigenvalues
4. Eigenvalues of a diagonal matrix $\mathbf{D} = \mathrm{diag}(d_1, \ldots, d_n)$ are just the diagonal entries $d_1, \ldots, d_n$

2.15.2 Spectral Theory

eigenpair : an eigenvalue λ and its associated eigenvector $\mathbf{x}$

eigenspace : the eigenspace associated with eigenvalue λ is the space of vectors where $\mathbf{A} - \lambda\mathbf{I} = 0$

spectrum of A : the set of all eigenvalues of a matrix $\mathbf{A}$

$$\sigma(\mathbf{A}) = \{\lambda \in \mathbb{C} \mid \det(\mathbf{A} - \lambda\mathbf{I}) = 0\} \tag{47}$$

spectral radius : magnitude of the largest eigenvalue

$$\rho(\mathbf{A}) = \max\{|\lambda_1|, \ldots, |\lambda_n|\} \tag{48}$$

Theorem 1 (Spectral radius bound)**.** *Spectral radius is bounded by the infinity norm of a matrix*

$$\rho(\mathbf{A}) = \lim_{k\to\infty} \left\|\mathbf{A}^k\right\|^{\frac{1}{k}} \tag{49}$$

Proof. let $|\lambda|^k \|\mathbf{v}\| = \left\||\lambda|^k\mathbf{v}\right\| = \left\|\mathbf{A}^k\mathbf{v}\right\|$

By the triangle rule,

$$|\lambda|^k \|\mathbf{v}\| \leq \left\|\mathbf{A}^k\right\| \cdot \|\mathbf{v}\| \tag{50}$$

and since $\mathbf{v} \neq 0$

$$|\lambda|^k \leq \left\|\mathbf{A}^k\right\| \tag{51}$$

which gives us

$$\rho(\mathbf{A}) = \lim_{k\to\infty} \left\|\mathbf{A}^k\right\|^{\frac{1}{k}} \tag{52}$$

■

2.16 Diagonalization

- A $n \times n$ matrix $\mathbf{A}$ is diagonalizable if it has n linearly independent eigenvectors
- Most square matrices are diagonalizable
  - Normal matrices are diagonalizable
  - Matrices w/ n distinct eigenvectors are diagonalizable

Lemma 1. *Eigenvectors associated with distinct eigenvalues are linearly independent*

- Eigenvalue equation can be written as $\mathbf{AV} = \mathbf{VD}$

- $\mathbf{D}$ is the matrix of eigenvalues and $\mathbf{V}$ is the matrix of corresponding eigenvectors

$$\mathbf{D} = \begin{bmatrix} \lambda_1 & & \\ & \ddots & \\ & & \lambda_n \end{bmatrix} \tag{53}$$

$$\mathbf{V} = \begin{bmatrix} \mathbf{v}_1 & \mathbf{v}_2 \dots \mathbf{v}_n \end{bmatrix} \tag{54}$$

- Assuming all λ_i's are unique, can diagonalize $\mathbf{A}$ by $\mathbf{A} = \mathbf{V}\mathbf{D}\mathbf{V}^{-1}$
- Recall: eigenvectors are independent so $\mathbf{V}$ is invertible
- if the eigenvectors are also all mutually orthogonal, then $\mathbf{V}$ is an orthogonal matrix and its inverse is its transpose so $\mathbf{A} = \mathbf{V}\mathbf{D}\mathbf{V}^T$

2.16.1 Symmetric Matrices

- if $\mathbf{A}$ is symmetric, then all of its eigenvalues are real and its eigenvectors are orthonormal
- So we can diagonalize $\mathbf{A}$ by $\mathbf{A} = \mathbf{V}\mathbf{D}\mathbf{V}^T$
- Given $y = \mathbf{V}^T x$

$$x^T \mathbf{A} x = x^T \mathbf{V}\mathbf{D}\mathbf{V}^T x = y^T \mathbf{D} y = \sum_{i=1}^{n} \lambda_i y_i^2 \tag{55}$$

- Thus, for the following maximization

$$\max_{x \in \mathbb{R}^n} x^T \mathbf{A} x \text{ subject to } \|x\|_2^2 = 1 \tag{56}$$

- Then the maximizing x can be found by finding the eigenvector that corresponds to the largest eigenvalue of $\mathbf{A}$

2.16.2 Applications of Eigenvalues and Eigenvectors

1. PageRank
2. Schrodinger equation
3. Principle Component Analysis (PCA)
4. Image compression

2.17 Matrix Calculus

2.17.1 Gradient

- Let a function $f : \mathbb{R}^{m \times n} \to \mathbb{R}$ take as input a matrix $A \in \mathbb{R}^{m \times n}$ and returns a real value

- Then the **gradient of f** is

$$\nabla_{\mathbf{A}} f(\mathbf{A}) = \begin{bmatrix} \frac{\partial f(\mathbf{A})}{\partial A_{11}} & \frac{\partial f(\mathbf{A})}{\partial A_{12}} & \dots & \frac{\partial f(\mathbf{A})}{\partial A_{1n}} \\ \frac{\partial f(\mathbf{A})}{\partial A_{21}} & \frac{\partial f(\mathbf{A})}{\partial A_{22}} & \dots & \frac{\partial f(\mathbf{A})}{\partial A_{2n}} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial f(\mathbf{A})}{\partial A_{m1}} & \frac{\partial f(\mathbf{A})}{\partial A_{m2}} & \dots & \frac{\partial f(\mathbf{A})}{\partial A_{mn}} \end{bmatrix} \tag{57}$$

- Every entry in the matrix is

$$\nabla_{\mathbf{A}} f(\mathbf{A})_{ij} = \frac{\partial f(\mathbf{A})}{\partial A_{ij}} \tag{58}$$

- The size of $\nabla_{\mathbf{A}} f(\mathbf{A})$ is always the same size as $\mathbf{A}$
- So if $\mathbf{A}$ is just a vector $\mathbf{x}$, then

$$\nabla_{\mathbf{x}} f(\mathbf{x}) = \begin{bmatrix} \frac{\partial f(\mathbf{x})}{\partial x_1} \\ \frac{\partial f(\mathbf{x})}{\partial x_2} \\ \vdots \\ \frac{\partial f(\mathbf{x})}{\partial x_n} \end{bmatrix} \tag{59}$$

- ex. for $\mathbf{x} \in \mathbb{R}^n$, let $f(\mathbf{x}) = \mathbf{b}^T\mathbf{x}$ for some known vector $\mathbf{b} \in \mathbb{R}^n$

$$f(\mathbf{x}) = \begin{bmatrix} b_1 & b_2 & \dots & b_n \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} = \sum_{i=1}^{n} b_i x_i \tag{60}$$

$$\frac{\partial f(\mathbf{x})}{\partial x_k} = \frac{\partial}{\partial x_k} \sum_{i=1}^{n} b_i x_i = b_k \tag{61}$$

$$\therefore \nabla_{\mathbf{x}} \mathbf{b}^T \mathbf{x} = \mathbf{b} \tag{62}$$

2.17.2 Properties of the Gradient

1. $\nabla_{\mathbf{x}}(f(\mathbf{x}) + g(\mathbf{x})) = \nabla_{\mathbf{x}} f(\mathbf{x}) + \nabla_{\mathbf{x}} g(\mathbf{x})$
2. For $t \in \mathbb{R}$, $\nabla_{\mathbf{x}}(t f(\mathbf{x})) = t \nabla_{\mathbf{x}} f(\mathbf{x})$

2.18 Hessian Matrix

- The **Hessian matrix** with respect to the vector $\mathbf{x} \in \mathbb{R}^n$ can be written as $\nabla^2_{\mathbf{x}} f(\mathbf{x})$ or as $\mathbf{H}$ and is an $n \times n$ matrix of partial derivatives

$$\nabla^2_{\mathbf{x}} f(\mathbf{x}) = \begin{bmatrix} \frac{\partial^2 f(\mathbf{x})}{\partial x_1^2} & \frac{\partial^2 f(\mathbf{x})}{\partial x_1 \partial x_2} & \cdots & \frac{\partial^2 f(\mathbf{x})}{\partial x_1 \partial x_n} \\ \frac{\partial^2 f(\mathbf{x})}{\partial x_2 \partial x_1} & \frac{\partial^2 f(\mathbf{x})}{\partial x_2^2} & \cdots & \frac{\partial^2 f(\mathbf{x})}{\partial x_2 \partial x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial^2 f(\mathbf{x})}{\partial x_n \partial x_1} & \frac{\partial^2 f(\mathbf{x})}{\partial x_n \partial x_2} & \cdots & \frac{\partial^2 f(\mathbf{x})}{\partial x_n^2} \end{bmatrix} \tag{63}$$

- Each entry is

$$\nabla^2_{\mathbf{x}} f(\mathbf{x})_{ij} = \frac{\partial^2 f(\mathbf{x})}{\partial x_i \partial x_j} \tag{64}$$

- Note: Hessian is the gradient **of every** entry of the gradient of the vector
- ex. 1st column of the Hessian is the gradient of $\frac{\partial f(\mathbf{x})}{\partial x_1}$
- Note: Hessian is always symmetric because of Schwarz's Theorem

Theorem 2 (Schwarz's Theorem)**.**

$$\frac{\partial^2 f(\mathbf{x})}{\partial x_i \partial x_j} = \frac{\partial^2 f(\mathbf{x})}{\partial x_j \partial x_i} \tag{65}$$

Order of partial derivatives doesn't matter as long as the 2nd derivative exists and is continuous

- ex. Consider quadratic function $f(\mathbf{x}) = \mathbf{x}^T \mathbf{A} \mathbf{x}$

$$f(\mathbf{x}) = \sum_{i=1}^{n} \sum_{j=1}^{n} A_{ij} x_i x_j \tag{66}$$

$$\frac{\partial f(\mathbf{x})}{\partial x_k} = \frac{\partial}{\partial x_k} \sum_{i=1}^{n} \sum_{j=1}^{n} A_{ij} x_i x_j \tag{67}$$

$$= \frac{\partial}{\partial x_k} \left[\sum_{i \neq k} \sum_{j \neq k} A_{ij} x_i x_j + \sum_{i \neq k} A_{ik} x_i x_k + \sum_{j \neq k} A_{kj} x_k x_j + A_{kk} x_k^2 \right] \tag{68}$$

$$= \sum_{i \neq k} A_{ik} x_i + \sum_{j \neq k} A_{kj} x_j + 2 A_{kk} x_k \tag{69}$$

$$= \sum_{i=1}^{n} A_{ik} x_i + \sum_{j=1}^{n} A_{kj} x_j = 2 \sum_{i=1}^{n} A_{ki} x_i \tag{70}$$

$$\frac{\partial^2 f(\mathbf{x})}{\partial x_k \partial x_l} = \frac{\partial}{\partial x_k} \left[\frac{\partial f(\mathbf{x})}{\partial x_l} \right] = \frac{\partial}{\partial x_k} \left[\sum_{i=1}^{n} 2 A_{li} x_i \right] \tag{71}$$

$$= 2 A_{lk} = 2 A_{kl} \tag{72}$$

- Thus $\nabla^2_{\mathbf{x}} f(\mathbf{x}) = 2\mathbf{A}$

2.19 Singular Value Decomposition

- Several computer algorithms can "factorize" a matrix into the product of other matrices
- Singular Value Decomposition is the most useful

  Singular Value Decomposition (SVD) : represent a matrix $\mathbf{A}$ as a product of 3 matrices $\mathbf{U}$, $\mathbf{S}$, $\mathbf{V}^T$, where $\mathbf{U}$ and $\mathbf{V}^T$ are rotation matrices and $\mathbf{S}$ is a scaling matrix
- MATLAB: [U,S,V] = **svd**(A)
- ex.

$$\underbrace{\begin{bmatrix} -0.40 & 0.916 \\ -0.916 & 0.40 \end{bmatrix}}_{\mathbf{U}} \underbrace{\begin{bmatrix} 5.39 & 0 \\ 0 & 3.154 \end{bmatrix}}_{\mathbf{S}} \underbrace{\begin{bmatrix} -0.05 & 0.999 \\ 0.999 & 0.05 \end{bmatrix}}_{\mathbf{V}^T} = \underbrace{\begin{bmatrix} 3 & -2 \\ 1 & 5 \end{bmatrix}}_{\mathbf{A}} \tag{73}$$

- In general, if $\mathbf{A}$ is $m \times n$, then $\mathbf{U}$ will be $m \times m$, $\mathbf{S}$ will be $m \times n$ and $\mathbf{V}^T$ will be $n \times n$
- ex.

$$\underbrace{\begin{bmatrix} -0.39 & -0.92 \\ -0.92 & 0.39 \end{bmatrix}}_{\mathbf{U}} \underbrace{\begin{bmatrix} 9.51 & 0 & 0 \\ 0 & 0.77 & 0 \end{bmatrix}}_{\mathbf{S}} \underbrace{\begin{bmatrix} -0.42 & -0.57 & -0.70 \\ 0.81 & 0.11 & -0.58 \\ 0.41 & -0.82 & 0.41 \end{bmatrix}}_{\mathbf{V}^T} = \underbrace{\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}}_{\mathbf{A}} \tag{74}$$

- Note: $\mathbf{U}$ and $\mathbf{V}$ are always rotation matrices
- also called "unitary" matrices because each column is a unit vector
- $\mathbf{S}$ is a diagonal matrix whose number of nonzero entries is the rank $\mathbf{A}$

2.19.1 SVD Applications

- Each product of (column i of $\mathbf{U}$) $\cdot$ (value i from $\mathbf{S}$) $\cdot$ (row i of $\mathbf{V}^T$) produces a component of the final $\mathbf{A}$
- We are building $\mathbf{A}$ as a linear combination of the columns of $\mathbf{U}$
- If we use all columns of $\mathbf{U}$, we can rebuild the original $\mathbf{A}$ perfectly
- But with real-world data, we can often just use the first few columns of $\mathbf{U}$ and get something close to $\mathbf{A}$
- Thus we call the first few columns of $\mathbf{U}$ the **Principal Components** of the data
- Principal components show the major patterns that can be added together to produce the columns of the original matrix
- Rows of $\mathbf{V}^T$ show how the principal components are mixed to produce the columns of $\mathbf{A}$
- For SVD with images, can use first few principal components to reproduce a recognizable picture

2.19.2 Principal Component Analysis

- Recall: columns of $\mathbf{U}$ are the Principal Components of the data

Principal Component Analysis (PCA) : construct a matrix $\mathbf{A}$ where each column is a separate data sample. Run SVD on $\mathbf{A}$ and look at the first few columns of $\mathbf{U}$ to see the common patterns

- Often raw data can have a lot of redundancy and patterns
- PCA allows you to represent data samples as weights on the principal components, rather than using the original raw form of the data
- This minimal PCA representation makes machine learning and other algorithms much more efficient

2.19.3 SVD Algorithm

- Computers can find eigenvectors $\mathbf{x}$ such that $\mathbf{Ax} = \lambda\mathbf{x}$ using this iterative algorithm

```
x = random unit vector
while (x not converged)
  x = Ax
  normalize x
```

- $\mathbf{x}$ will quickly converge to an eigenvector
- Some adjustments let this algorithm find all eigenvectors
- Note: eigenvectors are for square matrices, but SVD is for all matrices
- To do **svd**(A), computers do this

1. Take eigenvectors of $\mathbf{AA}^T$

– These eigenvectors are the columns of $\mathbf{U}$

– Square root of eigenvalues are the singular values (the entries of $\mathbf{S}$)

2. Take eigenvectors of $\mathbf{A}^T\mathbf{A}$

– These eigenvectors are columns of $\mathbf{V}$ (or rows of $\mathbf{V}^T$)

- SVD is fast (even for large matrices)

3 Colour Theory

3.1 Colour

colour : a psychological property caused by the interaction between physical light in the environment and our visual system; **not a physical property**

3.2 Physics of Light

- Visible light spectrum ranges from 400nm to 700nm and humans are most sensitive to light with wavelengths in the middle of this spectrum

- Any source of light can be completely described physically by its spectrum (i.e, the amount of energy emitted, per time unit, at each wavelength)
- Surfaces have reflectance spectra; reflected light is focused on a certain portion of the visible light spectrum (ex. green leaves and red tomatoes)
- Reflected colour is the result of interaction between the light source spectrum and the surface reflectance

3.3 Human Encoding of Colour

- Human eyes has 2 types of light-sensitive cells: rods and cones

  Rods : highly sensitive (good in low-light) but does not encode any colour information

  Cones less sensitive (good in high-light) and encodes colour information

- Rods are more numerous than cones

3.3.1 Cones and Colours

- Cones come in 3 types, each characterized by a unique response curve to different wavelengths of light
- Each response curve peaks at a unique wavelength: either 440nm (Blue), 530nm (Green), or 560nm (Red)
- Both cones and rods act as **filters**
  - Output is multiplication of each response curve by the spectrum, integrated over all wavelengths
  - Info encoded by resulting 3 numbers is usually good enough
  - But some info is lost in the compression from spectrum to electrical impulse in the retina
- Thus, some subset(s) of spectra will be erroneously perceived as identical; such spectra are called **metamers**

3.4 Colour Spaces

Colour space : describes range of colours as tuples of numbers, usually 3 or 4 values for colour components (ex. RGB)

- A colour space may be arbitrary or structured mathematically

3.4.1 Linear Colour Spaces

- Defined by a choice of 3 **primaries**

- i.e. 3 basis vectors in a 3D space

Figure 1: Mixing two lights produces colours that lie along a straight line in colour space. Mixing three lights produces colours that lie within the triangle they define in colour space.

3.4.2 RGB Space

- Primary colours are monochromatic lights (for monitors, they are the 3 types of phosphors used)
- Subtractive matching is required for certain wavelengths of light

Figure 2: Representation of RBG primaries and corresponding matching functions. The matching functions are the amounts of primaries needed to match the monochromatic test color at the wavelength shown on the horizontal scale. Source: `https://en.wikipedia.org/wiki/CIE_1931_color_space`

3.4.3 CIE XYZ Colour Space

- Primaries are imaginary, but matching functions are always positive
- The Y parameter corresponds to brightness or luminance of a colour
- Related to RGB space by linear transformation, upholding Grassman's Law

Figure 3: Source: `https://en.wikipedia.org/wiki/CIE_1931_color_space`

3.4.4 Nonlinear Colour Spaces - HSV

- Designed to reflect more traditional and intuitive colour mixing models (ex. mixing paint)
- Based on how colours are used in human vision
- Dimensions are: Hue, Saturation, Value (intensity)

Figure 4: General source: `https://en.wikipedia.org/wiki/HSL_and_HSV`

3.5 White Balancing

white balance : process of adjusting image data captured by sensors to properly render neutral colours (white, gray, etc)

- White balance adjustment performed automatically by digital cameras (custom settings for different lighting)
- Film cameras have different filters and film types for different conditions

3.5.1 Importance of White Balancing

- White balancing is important because unadjusted images have an unnatural colour

1. Sensors in cameras or film are different from human eyes
2. Different display media render images differently
3. Viewing conditions when image was taken are usually different from image viewing conditions

Figure 5: Example of two photos, one unbalanced, and one with incorrect white balancing. Source: `http://www.cambridgeincolour.com/tutorials/white-balance.htm`

3.5.2 Von Kries Method

Von Kries method : scale each colour channel by a "gain factor" to match appearnce of a gray neutral object. Accomplished by using the **gray card method**

Gray Card method : take photo of neutral (gray or white) card and determine values of each channel. If we find that the card has RGB values r_w, g_w, b_w, then we scale each channel by $\frac{1}{r_w}$, $\frac{1}{g_w}$, $\frac{1}{b_w}$

3.5.3 Other White Balancing Methods

- Without Gray Cards, we need to guess which pixels correspond with white objects

Gray World Assumption : assume that the average pixel value in the photo (r_{avg}, g_{avg}, b_{avg}) is gray and scale the image pixels by $\frac{1}{r_{avg}}$, $\frac{1}{g_{avg}}$, $\frac{1}{b_{avg}}$

Brightest Pixel Assumption : apply a weighting to each channel that is inversely proportional to the values of the brightest pixels. Works on non-saturated images and assumes that image highlights usually have the colour of the light source (which is usually white)

Gamut Mapping : apply a transformation to the image that maps the gamut of the image to the gamut of a "standard" image under white light

Gamut : set of all pixel colours displayed in an image. In mathematical terms, this is a "convex hull" and a subset of all possible colour combintations

3.6 Colour in Computer Vision

- Colour histograms for indexing and retrieval
- Skin detection
- Image segmentation and retrieval
- Buiilding appearance models for tracking

4 Pixels and Filters

4.1 Images Types

1. Binary (black and white only)
2. Greyscale
3. Colour

4.1.1 Binary Image Representation

- Pixel values have an integer 0 or 1 value
  - 0: black
  - 1: white
- Image is a 2D array of 0's and 1's

4.1.2 Grayscale

- Pixel values have integer values in [0, 255] to represent a wider range of intensity
  - 0: black
  - 1 - 254: some shade of grey
  - 255: white
- Image is 2D array of integers in [0, 255]

4.1.3 Colour Image

- 3 channels (RGB), each with integer values in [0, 255]
  - 0: absence of primary

- 1 - 254: some contribution of primary
- 255: full contribution of primary

- Image is a stack of 3 2D arrays of integers in [0, 255]
- i.e. 3 channels (RGB, LAB, HSV)
- Colour is represented with 3 numbers (3D coordinates in 3D colour space)
  - RGB: [R, G, B]
  - Lab: [L, a, b]
  - HSV: [H, S, V]

4.2 Sampling and Resolution

- Images are **samples**; they are not continuous and consist of discrete pixels of a certain size and density
- Pixels have a nonzero size $\longrightarrow$ images have limited resolution
- Fine details (smaller than pixel width) are approximated by pixels, which introduces <u>error</u>

Resolution : sampling parameter defined in dots per inch (DPI) or equivalent measures of spatial density. Standard screens have 72 DPI.

Pixels : quantized to an integer value within [0,255]. Pixel value is in "grayscale" (or "intensity")

4.3 Image Histograms

Histogram : provides frequency of brightness (intensity) values in the image (how many times a certain pixel value appears in the image). Captures the distribution of grey levels in the image (or color channel)

- Histograms help us detect particular image features
  - Sky: smooth coloration denotes consistency in image, consistent with the image of a sky
  - Grass: a jagged histogram shows a wide ranging variety in colouration, consistent with the shadows in a grass field
  - Faces: the colour composition of a face will be displayed in a histogram
- Histograms can be used to provide a quantifiable description of what things look like; this can be used as input to classifiers

```
def histogram(im):
  for row in im.shape[0]:
    for col in im.shape[1]:
      val = im[row, col]
      h[val] += 1
```

Figure 6: The image is sampled at two vertical positions, sampling a patch of sky and sampling a patch of grass. The corresponding histograms are shown to the right. Adapted from the accompanying lecture slide (Slide 23, slide credit Dr. Mubarak Shah

4.4 Images as Functions

- Images are usually digital (discrete representations of the photographed scene)
- They sample the 2D space onto a regular grid to produce a representation of the image as a matrix of integer values
- When dealing with images, we can imagine the image matrix as infinitely tall and wide

$$\begin{bmatrix} \ddots & & \vdots & & \iddots \\ & f[-1,1] & f[0,1] & f[1,1] & \\ \dots & f[-1,0] & f[0,0] & f[0,1] & \dots \\ & f[-1,-1] & f[0,-1] & f[1,-1] & \dots \\ \iddots & & \vdots & & \ddots \end{bmatrix} \tag{75}$$

- However, the displayed image is only a finite subset of this infinite matrix
- Thus, we can describe images as coordinates in a matrix

- An image can also be treated as a function $f : \mathbb{R}^2 \to \mathbb{R}^N$ where $f[m, n]$ is the intensity of a pixel at position (m, n)
- Note: we use square brackets, instead of usual parantheses, to denote discrete functions
- When we treat an image as a function, it is defined over a rectangle with finite range
- ex. the following function f returns the (grayscale) intensity of a single pixel in an image located in the region $[a, b] \times [c, d]$

$$f : [a, b] \times [c, d] \to [0, 255] \qquad \text{(Grayscale Pixel Intensity)}$$

domain support : the set of values $[a, b] \times [c, d]$ which contains all the values that are valid inputs to the function f

range : the set [0,255] which defines the set of possible outputs

- An image can also be treated a function mapping $\mathbb{R}^2 \to \mathbb{R}^3$
- ex. the RGB intensities of a given pixel can be written as the function g

$$g[x, y] = \begin{bmatrix} r[x, y] \\ g[x, y] \\ b[x, y] \end{bmatrix} \tag{Color Pixel Intensity}$$

- where $r, g, b : [a, b] \times [c, d] \to [0, 255]$

4.5 Linear Systems (Filters)

filtering : forming a new image whose pixel values are transformed from original pixel values

1. Denoising: removing salt and pepper noise
2. Super resolution: blurry $\longrightarrow$ super detailed

linear systems : converts an input function $f[m, n]$ to an output (or response) function $g[m, n]$ where m, n are the independent variables. For images, (m, n) correspond with spatial position in the image

system operator $\mathcal{S}$: maps a member of the set of possible output $g[m, n]$ to a member of the set of possible inputs $f[m, n]$

$$g = \mathcal{S}[f] \tag{76}$$

$$g[n, m] = \mathcal{S}\{f[n, m]\} \tag{77}$$

$$f[m, n] \xrightarrow{\mathcal{S}} g[m, n] \tag{78}$$

4.5.1 Filter 1: Moving Average (Smoothing)

Moving Average : sets the value of a pixel to be the average of its neigbouring pixels. (ex. the nine pixels in a 3×3 area, when applying a 3×3 filter)

$$g[m, n] = \frac{1}{9} \sum_{i=-1}^{1} \sum_{j=-1}^{1} f[m - i, n - j] \tag{Weighted Average}$$

- The weighted average filter smoothes out sharper edges/features in the image, creating a blurred or smoothed effect

$$h[m, n] = \frac{1}{9} \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix} \tag{79}$$

- Note: moving average filter is linear and shift-invariant

4.5.2 Filter 2: Image Segmentation

Image Segmentation : based on simple threshold where the filter sets a pixel either to 255 (white) or 0 (black) depending on whether or not it meets the threshold t

$$g[m,n] = \begin{cases} 255 & f[m,n] \geq t \\ 0 & \text{otherwise} \end{cases} \tag{Threshold}$$

- The basic image segmentation filter divides an image's pixels into binary black or white regions depending on whether $f[m,n] \geq t$

4.6 Properties of Systems

- Here are some useful properties that a system may or may not possess

4.6.1 Amplitude Properties

1. **Additivity**: A system is additive if it satisfies the equation

$$\mathcal{S}[f_i[m,n] + f_j[m,n]] = \mathcal{S}[f_i[m,n]] + \mathcal{S}[f_j[m,n]] \tag{80}$$

2. **Homogeneity**: A system is homogeneous if it satisfies the equation

$$\mathcal{S}[\alpha f_i[n,m]] = \alpha \mathcal{S}[f_i[n,m] \tag{81}$$

3. **Superposition**: A system has the property of superposition if it satisfies the equation

$$\mathcal{S}[\alpha f_i[n,m]] + \beta f_j[n,m]] = \alpha \mathcal{S}[f_i[n,m]] + \beta \mathcal{S} f_j[n,m]] \tag{82}$$

4. **Stability**: A system is stable if it satisfies the inequality

$$|f[n,m|] \leq k \implies |g[n,m]| \leq ck \qquad \text{for some } c \tag{83}$$

5. **Invertibility**: A system is invertible if it satisfies the equation

$$\mathcal{S}^{-1}[\mathcal{S}[f[n,m]]] = f[n,m] \tag{84}$$

4.6.2 Spatial Properties

1. **Causality**: A system is causal if for $m < m_0$ and $n < n_0$

$$f[m,n] = 0 \implies g[m,n] = 0 \tag{85}$$

2. **Shift Invariance**: A system is shift invariant if

$$f[m - m_0, n - n_0] \xrightarrow{\mathcal{S}} g[m - m_0, n - n_0] \tag{86}$$

4.7 Linear Systems

Linear system : system that satisfies the property of superposition

Linear shift-invariant (LSI) system : a linear system that is also shift-invariant

- When we use a linear system for filtering, the new image pixels are weighted sums of the original pixel values, where the same set of weights are used for each pixel
- Note: Linear shift-invariant systems are important because human vision is linear shift-invariant

4.7.1 Impulse Response

- To consider the impulse response of a system $\mathcal{S}$, pass the 2D delta function $\delta_2[m,n]$ into the system

$$\delta_2[m,n] = \begin{cases} 1 & m = 0 \text{ and } n = 0 \\ 0 & \text{otherwise} \end{cases} \tag{87}$$

impulse response h : by inputing the 2D delta function $\delta_2[m,n]$ into the system, we get a filter $h[m,n]$ that tells us what the system is actually doing at each pixel

$$h[m,n] = \mathcal{S}[\delta_2] \tag{88}$$

- Consider a 3×3 image $f[m,n]$

$$f[m,n] = \begin{bmatrix} f[0,0] & f[0,1] & f[0,2] \\ f[1,0] & f[1,1] & f[1,2] \\ f[2,0] & f[2,1] & f[2,2] \end{bmatrix} \tag{89}$$

- We can rewrite $f[m,n]$ as a sum of 2D delta functions

$$f[m,n] = f[0,0] \times \delta_2[m-0, n-0] + f[0,1] \times \delta_2[m-0, n-1] + \cdots + f[m,n] \times \delta_2[0,0] + \ldots \tag{90}$$

$$= \sum_{i=-\infty}^{\infty} \sum_{j=-\infty}^{\infty} f[i,j]\delta_2[m-i, n-j] \tag{91}$$

- Now if we apply the system operator $\mathcal{S}$ to our image $f[m,n]$, we get

$$\mathcal{S}[f[m,n]] = \mathcal{S}\Bigg[\sum_{i=-\infty}^{\infty} \sum_{j=-\infty}^{\infty} f[i,j]\delta_2[m-i, n-j]\Bigg] \tag{92}$$

$$= \sum_{i=-\infty}^{\infty} \sum_{j=-\infty}^{\infty} f[i,j]\mathcal{S}\Big[\delta_2[m-i, n-j]\Big] \tag{93}$$

- by applying superposition and shift invariance for $\mathcal{S}$

$$\mathcal{S}[\alpha f_i[n,m]] + \beta f_j[n,m]] = \alpha\mathcal{S}[f_i[n,m]] + \beta\mathcal{S}f_j[n,m]] \tag{94}$$

- Recall: the impulse response $h[m,n]$ is defined as $\mathcal{S}[\delta_2[m,n]]$, so

$$f[m,n] = \sum_{i=-\infty}^{\infty}\sum_{j=-\infty}^{\infty} f[i,j]h[m-i,n-j] \tag{95}$$

- Note: an LSI system is completely specified by its impulse response $h[m,n]$

$$f[n,m] \longrightarrow \underset{\delta_2[n,m] \xrightarrow{\mathcal{S}} h[m,n]}{\boxed{\mathcal{S}\text{ LSI}}} \longrightarrow \sum_{i=-\infty}^{\infty}\sum_{j=-\infty}^{\infty} f[i,j]h[m-i,n-j]$$

Figure 7: Graphical representation of the impulse response

- Thus the impulse response $h[m,n]$ can be used to find the output image $g[m,n]$ with discrete convolution

$$\mathcal{S}[f] = f[m,n] * h[m,n] \tag{96}$$

4.7.2 Impulse Response Example: Moving Average Filter

- The moving average filter is defined as

$$g[m,n] = \frac{1}{9}\sum_{i=-1}^{1}\sum_{j=-1}^{1} f[m-i,n-j] \tag{97}$$

- Using $h[m,n] = \mathcal{S}[\delta_2]$

$$h[m,n] = \frac{1}{9}\sum_{i=-1}^{1}\sum_{j=-1}^{1} \delta_2[m-i,n-j] \tag{98}$$

4.8 Convolution

kernel H : pattern of weights used for a linear filter

convolution : process of applying the filter. Given a filter kernel H, the convolution of the kernel with image F is an image G, where the i,jth component of G is

$$G = \sum_u \sum_v H_{i-u,j-v} F_{u,v} \tag{99}$$

- Note: for convolution the kernel H is flipped both horizontal and vertical direction (we have $-u$ and $-v$) before multiplying the overlapped input data F

- The easiest way to think of convolution is as a system that uses information from neighbouring pixels to filter the target pixel
- Convolution allows us to compute the output of passing any input signal through a system by simply considering the impulse response of the system
  - This works because any signal can be decomposed into a weighted sum of impulse functions (ex. pixels in an image are 2D delta functions)
  - This allows us to use the a weighted sum of the impulse responses for each part of the signal
- The impulse response of a system, $h[n]$ is defined as the output resulting from passing an impulse function into a system
- When a system is linear, scaling the impulse function results in scaling of the impulse response by the same amount
- When a system is shift-invariant, shifting the impulse function shifts the impulse response by the same amount
- Thus, in general for an arbitrary input signal $x[n] = \sum_{k=-\infty}^{\infty} x[k]\delta[n-k]$ passed into a linear, shift-invariant system, the output is $y[n] = \sum_{k=-\infty}^{\infty} x[h]h[n-k]$
  - i.e the convolution of the signal x with the impulse response h
- 2D convolution is similar to 1D, but we have to iterate over 2 axes instead of 1

$$f[m,n] * h[m,n] = \sum_{i=-\infty}^{\infty} \sum_{j=-\infty}^{\infty} f[i,j]h[m-i,n-j] \tag{100}$$

- $f[m,n] * h[m,n]$ represents a function being multiplied by a shifted impulse response
- f = image and h = kernel

4.8.1 2D Convolution Example: Identity Filter

$$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix} \qquad \text{(No change in output)}$$

4.8.2 2D Convolution Example: Shift Left Filter

$$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{bmatrix} \qquad \text{(Shifts image left by 1 pixel)}$$

4.8.3 2D Convolution Example: Box Blur Filter

$$\frac{1}{9}\begin{bmatrix}1 & 1 & 1\\ 1 & 1 & 1\\ 1 & 1 & 1\end{bmatrix} \qquad \text{(Blurs image)}$$

- The box blur filter takes the average of neighbouring pixels, which smoothes extreme features (less sharp)

4.8.4 2D Convolution Example: Sharpening Filter

$$\underbrace{\begin{bmatrix}0 & 0 & 0\\ 0 & 2 & 0\\ 0 & 0 & 0\end{bmatrix} - \frac{1}{9}\begin{bmatrix}1 & 1 & 1\\ 1 & 1 & 1\\ 1 & 1 & 1\end{bmatrix}}_{\text{filter sums to 1}} \tag{101}$$

$$= \underbrace{\begin{bmatrix}0 & 0 & 0\\ 0 & 1 & 0\\ 0 & 0 & 0\end{bmatrix}}_{\text{original image}} + \underbrace{\begin{bmatrix}0 & 0 & 0\\ 0 & 1 & 0\\ 0 & 0 & 0\end{bmatrix} - \frac{1}{9}\begin{bmatrix}1 & 1 & 1\\ 1 & 1 & 1\\ 1 & 1 & 1\end{bmatrix}}_{\text{details of the image}} \qquad \text{(Blurs image)}$$

- By subtracting the average from the original picture, only the extreme features remain
- Adding the extreme features to the original image then accentuates differences (sharpens image)

4.9 Implementation of Convolution: Image Support and Edge effects

- A computer will only compute finite support signals
- i.e. images that are zero for m, n outside of rectangular region (the domain support)
- So convolution software often zero pads the matrix
- Numpy's convolution performs 2D discrete system convolution of finite support signals
- At the edge, can either do
  - zero padding
  - edge value replication
  - mirror extension (wrap around to other side)

4.10 Cross Correlation **

- **Cross Correlation** of two 2D signals $f[m,n]$ and $g[m,n]$ is

$$r_{fg}[k,l] \triangleq \sum_{m=-\infty}^{\infty} \sum_{n=-\infty}^{\infty} f[m,n] * g[m-k, n-l] \tag{102}$$

$$= \sum_{m=-\infty}^{\infty} \sum_{n=-\infty}^{\infty} f[m-k, n-l] g^*[m,n], \qquad k,l \in \mathbb{Z} \tag{103}$$

- (k,l) is called the **lag**
- Cross correlation is equivalent to convolution without the flip of the filter kernel

$$r_{fg}[m,n] = f[m,n] * g^*[-m,-n] \tag{104}$$

- Note: g^* is defined as the complex conjugate of g. In this class, $g(n,m) \in \mathbb{R}$ so $g^* = g$
- Correlation can be used to find known features in images by using a kernel that contains target features

4.10.1 Properties of Cross Correlation

1. **Associative property**:

$$(f **h_1) **h_2 = f **(h_1 **h_2) \tag{105}$$

2. **Distributive property**:

$$f **(h_1 + h_2) = (f **h_1) + (f **h_2) \tag{106}$$

3. **Commutative property**:

$$h_1 **h_2 = h_2 **h_1 \tag{107}$$

4. **Shfit property**:

$$f[m,n] **\delta_2[m-m_0, n-n_0] = f[m-m_0, n-n_0] \tag{108}$$

5. **Shift invariance**:

$$g[m,n] = f[m,n] **h[m,n] \tag{109}$$

$$\implies f[m-\ell_1, n-\ell_l] **h[m-\ell_2, n-\ell_2] = g[m-\ell_1-\ell_2, n-\ell_1-\ell_2] \tag{110}$$

4.10.2 Convolution vs Cross Correlation

- Correlation is equivalent to convolution, i.e. $f **g = f * g$ **only** when the kernel/filter is **symmetric**

Convolution : integral that expresses the amount of overlap of one function as it is shifted over another function. Convolution is a filtering operation

Correlation : calculates a similarity measure for two input signals (ex. 2 image patches). The output of correlation reaches a maximum when the two signals match best. Correlation is a measure of relatedness of two signals

4.11 Discrete Convolution *

- Fold $h[k, l]$ about origin to form $h[-k, -l]$ (flip both horizontally and vertically)
- Shift the folded results by a number k to form $h[m-k, n-\ell]$
- Multiply the kernel $h[m-k, n-\ell]$ by the image pixel $f[k, l]$
- Sum over all k, ℓ
- Repeat for every number

5 Edge Detection

- **Goal**: identify sudden changes (discontinuities) in an image
- Intuitively, most semantic and shape info from the image can be encoded in the edges
- Edge representation is more compact than pixels
- **Ideal**: artist's line drawing (but artist is also using object-level knowledge)

5.0.1 Why are Edges Useful

- Extract info and recognize objects
- Recover geometry and viewpoints (ex. vanishing lines)

5.0.2 Origins of Edges in Images

1. Surface normal discontinuities
   - ex. corners or edges where different planes intersect
   - Each plane has a different normal and as you cross the corner/edge, the surface normal changes drastically resulting in an edge
2. Depth discontinuities
   - Differences in depth results in different lighting and thus an image edge
3. Surface colour discontinuities
   - ex. shadows, patterns on an object (ex. tiles)
4. Illumination discontinuity

Figure 8: Examples of discontinities that create edges

5.1 Image Gradients

5.1.1 1D Derivatives

Continuous :

$$\frac{df}{dx} = \lim_{\Delta x \to 0} \frac{f(x) - f(x - \Delta x)}{\Delta x} = f'(x) = f_x \tag{111}$$

Discrete estimate (finite differences) :

$$\frac{df}{dx} \approx f(x) - f(x-1) = f'(x) = f_x \tag{112}$$

5.1.2 1D Discrete Derivatives and Filters

- Note: can calculate discrete derivatives by convolving differentiation kernels (remember to flip filters for convolution)
- Differentiation is linear and shift invariant

Backward :

$$\frac{df}{dx} = f(x) - f(x-1) = f'(x) \tag{113}$$

$$\begin{bmatrix} 0 & 1 & -1 \end{bmatrix} \tag{114}$$

Forward :

$$\frac{df}{dx} = f(x) - f(x+1) = f'(x) \tag{115}$$

$$\begin{bmatrix} -1 & 1 & 0 \end{bmatrix} \tag{116}$$

Central :

$$\frac{df}{dx} = f(x+1) - f(x-1) = f'(x) \tag{117}$$

$$\begin{bmatrix} 1 & 0 & -1 \end{bmatrix} \tag{118}$$

5.2 2D Continuous Derivatives

2D function :

$$f(x, y) \tag{119}$$

Gradient vector :

$$\nabla f(x, y) = \begin{bmatrix} \dfrac{\partial f(x, y)}{\partial x} \\ \dfrac{\partial f(x, y)}{\partial y} \end{bmatrix} = \begin{bmatrix} f_x \\ f_y \end{bmatrix} \tag{120}$$

Gradient magnitude :

$$|\nabla f(x, y)| = \sqrt{f_x^2 + f_y^2} \tag{121}$$

Gradient direction :

$$\theta = \arctan\left(\frac{f_y}{f_x}\right) \tag{122}$$

- Note: gradient direction is largely independent of illumination intensity

5.2.1 2D Discrete Gradients

- Can approximate gradient using neighbouring pixels based on central discrete derivative equation extended to 2D
  - The gradient in the x-dir is then given by

$$\frac{1}{3}\begin{bmatrix} 1 & 0 & -1 \\ 1 & 0 & -1 \\ 1 & 0 & -1 \end{bmatrix} \tag{123}$$

  - The gradient in the y-dir is given by

$$\frac{1}{3}\begin{bmatrix} 1 & 1 & 1 \\ 0 & 0 & 0 \\ -1 & -1 & -1 \end{bmatrix} \tag{124}$$

- Note: the x-direction gradient detects vertical edges while the y-direction gradient detects horizontal edges
- ex. the x and y gradients for an image I can be found by convolution with the 3×3 kernels

$$I_y = \begin{bmatrix} 10 & 10 & 20 & 20 & 20 \\ 10 & 10 & 20 & 20 & 20 \\ 10 & 10 & 20 & 20 & 20 \\ 10 & 10 & 20 & 20 & 20 \\ 10 & 10 & 20 & 20 & 20 \end{bmatrix} * \frac{1}{3} \begin{bmatrix} 1 & 1 & 1 \\ 0 & 0 & 0 \\ -1 & -1 & -1 \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix} \tag{125}$$

$$I_x = \begin{bmatrix} 10 & 10 & 20 & 20 & 20 \\ 10 & 10 & 20 & 20 & 20 \\ 10 & 10 & 20 & 20 & 20 \\ 10 & 10 & 20 & 20 & 20 \\ 10 & 10 & 20 & 20 & 20 \end{bmatrix} * \frac{1}{3} \begin{bmatrix} 1 & 0 & -1 \\ 1 & 0 & -1 \\ 1 & 0 & -1 \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 10 & 10 & 0 & 0 \\ 0 & 10 & 10 & 0 & 0 \\ 0 & 10 & 10 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix} \tag{126}$$

5.3 Simple Edge Detection

edge : set of points with rapid change in the image intensity function. Edges correspond with the extrema of the derivative of the image intensity function.

Figure 9: An image with intensity function and first derivative

5.3.1 Image Gradient

Gradient of an Image : a vector that points in the direct of most rapid increase in intensity

$$\nabla f = \begin{bmatrix} \dfrac{\partial f}{\partial x} \\ \\ \dfrac{\partial f}{\partial y} \end{bmatrix} \tag{127}$$

Gradient direction :

$$\theta = \arctan\left(\frac{f_y}{f_x}\right) \tag{128}$$

Edge strength : magnitude of image gradient

$$\|\nabla f\| = \sqrt{\left(\frac{\partial f}{\partial x}\right)^2 + \left(\frac{\partial f}{\partial y}\right)^2} \tag{129}$$

Figure 10: The gradient vector directions

- The shadows (dark spots) in an image derivative show which direction derivative you have
  - vertical shadows $\longrightarrow$ x-dir derivative
  - horizontal shadows $\longrightarrow$ y-dir derivative

Figure 11: The gradients applied to an image of a tiger

5.4 Intensity Profile

5.4.1 Effects of Noise

- With noise, it is hard to find the edge from maxima in derivative (because derivative is sensitive to noise and makes a lot of erroneous peaks)
- Finite difference filters respond strongly to noise
  - Image noise results in pixels that look very different from their neighbours
  - Generally, the larger the noise, the stronger the response
- Solution: smoothing the image should help by forcing the pixels that look different to their neighbours (poteintally noise pixels) to look more like their neighbours

Figure 12: The derivative of an edge in a noisy image. Note it is extremely hard to find the edge with this noisy image

5.4.2 1D Smoothing Filters

Mean smoothing filter :

$$\begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} \qquad \begin{bmatrix} 1 & 1 & 1 \end{bmatrix} \tag{130}$$

Gaussian smoothing filter :

$$\begin{bmatrix} 1 \\ 2 \\ 1 \end{bmatrix} \qquad \begin{bmatrix} 1 & 2 & 1 \end{bmatrix} \tag{131}$$

5.4.3 Gaussian Blur

Gaussian blur : blurring image with a Gaussian function to reduce image noise. Usually involves convolving with a filter that roughly approximates the 2D Gaussian

- Note: Gaussian is a low-pass filter so it attenuates high frequency signals
- Gaussian blurring usually done as a preliminary step

1D Gaussian :

$$G(x) = \frac{1}{\sqrt{2\pi\sigma}} e^{-\frac{x^2}{2\sigma^2}} \tag{132}$$

2D Gaussian :

$$G(x, y) = \frac{1}{2\pi\sigma} e^{-\frac{x^2+y^2}{2\sigma^2}} \tag{133}$$

5.5 Smooth First Edge Detection (Unoptimized)

Figure 13: Smooth first edge detection

- To find edges, look for peaks in $\frac{d}{dx}(f * g)$
- Note: smoothing an image and then differentiating it is the same as convolving it with the derivative of a smoothing kernel

5.5.1 Derivative Theorem of Convolution

Theorem 3. *Derivative Theorem of Convolution*

$$\frac{d}{dx}(f * g) = f * \frac{d}{dx}g \tag{134}$$

- This allows us to save an extra step in edge detection

5.5.2 Improved Smooth First Edge Detection

Figure 14: Improved smooth first edge detection

5.6 Tradeoff between Smoothing at Different Scale σ

- Smoothed derivative reduces noise susceptibility but blurs edges

5.7 Criteria for an Optimal Edge Detector

1. Good detection
   - Minimizes probability of false positives (detecting fake edges caused by noise)
2. Good localization
   - Detected edges must be as close as possible ot the true edges
3. Single response
   - Detector must return one point only for each true edge point
   - i.e. minimize the number of local maxima around the true edge

Figure 15: Issues with bad edge detectors

5.8 Sobel Edge Detector

- Uses two 3×3 kernels which are convolved with the original image to calculate approximations to the image gradient
- One kernel for horizontal changes and one for vertical
- Based on smoothing and differentiation idea

5.8.1 Sobel Operator

- Note: Sobel kernels can be decomposed into a product of a smoothing and a differentiation kernel

Sobel operator in x-dir G_x :

$$G_x = \begin{bmatrix} +1 & 0 & -1 \\ 2 & 0 & -2 \\ +1 & 0 & -1 \end{bmatrix} = \underbrace{\begin{bmatrix} 1 \\ 2 \\ 1 \end{bmatrix}}_{\text{Gaussian smoothing}} \underbrace{\begin{bmatrix} +1 & 0 & -1 \end{bmatrix}}_{\text{Differentiation}} \tag{135}$$

Sopel Operator in y-dir G_y :

$$G_x = \begin{bmatrix} +1 & +2 & +1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix} = \underbrace{\begin{bmatrix} +1 \\ 0 \\ -1 \end{bmatrix}}_{\text{Differentiation}} \underbrace{\begin{bmatrix} 1 & 2 & 1 \end{bmatrix}}_{\text{Gaussian smoothing}} \tag{136}$$

Sobel magnitude G :

$$G = \sqrt{G_x^2 + G_y^2} \tag{137}$$

Angle/direction of gradient θ :

$$\theta = \arctan\left(\frac{G_y}{G_x}\right) \tag{138}$$

5.8.2 Sobel Operator Issues

1. Poor localization
- Sobel operator triggers response in multiple adjacent pixels
2. Thresholding value favours certain directions of otehrs
- Can miss oblique edges more than horizontal/vertical edges
- False negatives

5.9 Canny Edge Detector

- Most widely used edge detector in computer vision
- Thoretical model: step edges corrupted by additive Gaussian noise
- Canny shows that 1st derivative of Gaussian closely approximates the operator that optimizes the product of signal-to-noise ratio and lcoalization

5.9.1 Canny Edge Detector Features

1. Suppresses noise
2. Computes gradient magnitude and direction
3. Apply Non-Maximum suppression
- Assumes minimal response
4. Use hysteresis and connectivity analysis to detect edges

5.9.2 Non-Maximum Suppression

- This step makes sure that edges are specific
- Assume that edge occurs where gradient reaches a maximum
- Suppress non-maximizing gradient even if it passes threshold
- Round all gradient directions to the nearest 45°

5.9.3 Removing Unnecessary Gradients

- $\|\nabla G\|(x, y)$ is the gradient at each pixel

$$M(x, y) = \begin{cases} \|\nabla G\|(x, y) & \text{if } \|\nabla G\|(x, y) \geq \|\nabla G\|(x', y') \text{ and } \|\nabla G\|(x, y) \geq \|\nabla G\|(x'', y'') \\ 0 & \text{otherwise} \end{cases} \tag{139}$$

- Note: x' and x'' are the neighbours of x along normal direction to an edge

5.9.4 Hysteresis Thresholding

- Avoid streaks near threshold values
- Define 2 thresholds **low** and **high**
  - If less than low $\longrightarrow$ **not an edge**
  - If greater than high $\longrightarrow$ **strong edge**
  - If between low and high $\longrightarrow$ **weak edge**

- if gradient at a pixel is
  - Above high $\longrightarrow$ **strong edge pixel**
  - Below low $\longrightarrow$ **non-edge pixel**
  - Between low and high
    - if connected to a strong edge pixel or other pixels in between low and high $\longrightarrow$ **edge-pixel**
- Use breath-first or depth-first search to find all the edges

5.9.5 Canny Edge Detector Algorithm

1. Filter image with x, derivatives of Gaussian
2. Find magnitude and orientation of gradient
3. Non-maximum suppression
   - Thins multi-pixel wide "ridges" down to single pixel width
4. Thresholding and hysteresis
   - Define 2 thresholds: low and high
   - Use the high threshold to start edges and the low threshold to continue them

5.10 Effect of Standard Deviation σ (aka scale) in Gaussian Filter

- Large σ detects large scale edges
- Small σ detects fine features

5.11 Hough Transform

Hough Transform (HT) : detects location of lines in an image using parametrized equations

- Caveat: Hough transform can only detect lines, circles, and other structures **only** if their parametric equation is known
- Can give robust detection under noise and partial occlusion

5.11.1 Preparation before applying Hough Transform

- Perform some edge detection and a thresholding of the edge magnitude
- Thus we have some pixels that may partially describe the boundary of some objects

5.11.2 Detecting Lines using Hough Transform

- Goal: find sets of pixels that make up straight lines
- Consider a point of known coordinates (x_i, y_i)
- There are many lines passing through the point (x_i, y_i)
- Straight lines that pass through (x_i, y_i) have the equation $y = ax_i + b$ for some set of parameters (a, b)
- We can rewrite this equation as $b = -ax_1 + y_1$ and consider x, y as parameters and a, b as variables
- This gives a line in (a, b) space parametrixed in x only
- Thus a single point in (x_1, y_1) space gives us a line in (a, b) space
- 2 points (x_1, y_1) and (x_2, y_2) define a line in (x, y) plane and also give rise to 2 diferent liens in (a, b) space
- In (a, b) space, these lines will intersect at (a', b')
- Thus, all points on the line defined by (x_1, y_1) and (x_2, y_2) in (x, y) space will parametrize liens that interesect in (a', b') in (a, b) space

Figure 16: Transformation from original (x, y) space to Hough (a, b) space. Here we have two points $(x_1, y_1) = (1, 1)$ and $(x_2, y_2) = (2, 3)$. We transform these points into (a, b) space to get the lines $b = -a * 1 + 1$ and $b = -a * 2 + 3$. These two lines intersect at $a = 2$ and $b = -1$, which gives us the equation for the original line in (x, y) space as $y = 2x - 1$

5.11.3 Hough Transform Algorithm

1. Quantize parameter space (a, b) by dividing it into **accumulator cells**
2. Count the number of times a line intersects a given cell

- For each pair of points (x_1, y_1) and (x_2, y_2) detected as an edge, find the intersection (a', b') in (a, b) space
- Increase the value of a cell in the range $[[a[[a_{min}, a_{max}], [b_{min}, b_{max}]]$ that (a', b') belongs to

3. Cells receiving more than a certain number of counts (aka **votes**) are assumed to correspond with lines in (x, y) space

5.11.4 Hough Transform in (ρ, θ) space)

- Can represent lines as polar coordinates instead of $y = ax + b$ *allows us to represent vertical lines

– A vertical line will have $\theta = 0$ and ρ equal to the x-intercept
– A horizontal line will have $\theta = 90$ and ρ equal to the y-intercept

Polar coordinate representation : For a pixel (x_i, y_i), we transform it with the equation $x \cos\theta + y \sin\theta = \rho$

- Note: lines in (x, y) space are sine wave-like functions in (ρ, θ) space

Figure 17: Transformation from original (x, y) space to Hough (ρ, θ) space

- Still use regular Hough Transform algorithm by looking for intersections of these transformed functions in (ρ, θ) space to get the ρ, θ parameter values
- Again, the accumulator cells with the most votes are most likely the real lines in our image

5.11.5 Hough Transform Advantages

- Conceptually simple
- Easy implementation
- handles missing and occluded data very gracefully
- Can be adapted to many types of forms, not just lines

5.11.6 Hough Transform Disadvantages

- Computational complex for objects with many parameters
- Looks for only one single type of object
- Can be "fooled" by "apparent edges"
- Length and position of a line segment cannot be determined
- Co-linear segments cannot be separated