

Lecture 10:

# Data Modality and Feature Extraction

**Ho-min Park**

[homin.park@ghent.ac.kr](mailto:homin.park@ghent.ac.kr)

[powersimmani@gmail.com](mailto:powersimmani@gmail.com)

# Lecture Contents

**Part 1:** Understanding Data Modalities

**Part 2:** Traditional Feature Extraction

**Part 3:** Learning-based Representations

**Part 1/3:**

# **Understanding Data Modalities**

- 1.** Overview of Data Modalities
- 2.** Structured vs Unstructured Data
- 3.** Text Data Characteristics
- 4.** Image Data Characteristics
- 5.** Audio/Speech Data Characteristics
- 6.** Video Data Characteristics
- 7.** Graph/Network Data
- 8.** Multimodal Data

# Overview of Data Modalities

**Data modality** refers to the type or mode through which information is represented



Text



Images



Audio



Video



Structured Data



Sensor Data



- Unique characteristics per modality

- Specific processing methods needed

- Different storage requirements

- Affects ML algorithm selection

# Structured vs Unstructured Data

## Structured Data

- Organized in predefined format
- Easy to query and analyze
- Efficient storage
- Standard analytical tools

### Examples:

Tables, databases, spreadsheets

VS

## Unstructured Data

- No predefined structure
- Rich information content
- Complex processing required
- Advanced analytical methods

### Examples:

Text, images, audio, video



## Semi-Structured Data

Hybrid format with some organization (JSON, XML)

**80-90%**

of enterprise data is unstructured and growing rapidly

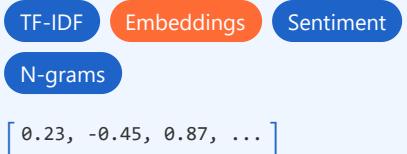
**Feature Extraction: Unstructured → Structured**

Numeric Features

Numeric Features



Text

→  
NLP

Image

→  
CNN

Audio

→  
DSP

Numeric Features

MFCC   Spectral   Pitch   Chroma

[ -0.82, 3.41, 0.15, ... ]



Video

→  
CV+DL

Numeric Features

3D-CNN   Optical Flow   I3D

[ 2.67, -1.23, 0.94, ... ]

## Structured Data Examples

### Classification Example

Age	Income	Credit Score	Education	Approved
25	45K	680	Bachelor	No
35	75K	720	Master	Yes
42	90K	780	PhD	Yes
28	52K	650	Bachelor	No
50	110K	800	Master	Yes

### Classification with PCA Features

PC1	PC2	PC3	Gender	Churn
2.4	-1.2	0.8	M	No
-0.5	3.1	-1.5	F	Yes
1.8	0.3	2.2	M	No
-2.1	2.7	-0.9	F	Yes
3.2	-0.8	1.4	M	No

### Regression Example

Size (sq ft)	Bedrooms	Age (years)	Distance (km)	Price (\$K)
1200	2	10	5.2	320
1800	3	5	3.1	485
2400	4	2	1.8	650

### Regression with PCA Features

PC1	PC2	PC3	PC4	Sales (\$M)
12.5	-3.2	1.8	0.5	2.4
8.7	4.1	-2.3	1.2	3.8
15.3	-1.5	3.4	-0.8	5.2
6.2	2.8	-1.1	0.9	1.9

Size (sq ft)	Bedrooms	Age (years)	Distance (km)	Price (\$K)
950	1	15	8.5	245
2100	3	8	4.3	560

PC1	PC2	PC3	PC4	Sales (\$M)
11.8	0.3	2.7	-0.4	4.5

# Text Data Characteristics



## Sequential Nature

Word order and context matter significantly



## High Dimensionality

Vocabulary size can reach tens of thousands



## Sparse Representation

Most words don't appear in any given document



## Ambiguity

Words have multiple meanings depending on context



## Language-Dependent

Different languages have unique structures and rules



## Rich Semantic Info

Information encoded in syntax and grammar



## Required Processing Steps

Tokenization

Normalization

Vocabulary Management



## Text Data Examples

Example 1: Product Review

"This laptop is AMAZING!!! Best purchase ever 😊"

### Example 2: Social Media Post

"Can't believe it's already 2024... time flies! #newyear"

### Example 3: Customer Inquiry

"Hi, I'm wondering if this product comes in blue?"

## 🔧 Processing Steps Example

### 1 Tokenization

Split text into individual tokens (words, subwords, or characters)

Input: "This laptop is AMAZING!!!"

Output: ["This", "laptop", "is", "AMAZING", "!", "!", "!" ]

### 2 Normalization

Convert to lowercase, remove punctuation, handle special characters

Input: ["This", "laptop", "is", "AMAZING", "!", "!", "!" ]

Output: ["this", "laptop", "is", "amazing"]

### 3 Vocabulary Management

Map tokens to unique IDs based on vocabulary dictionary

Input: ["this", "laptop", "is", "amazing"]

Vocabulary: {this: 45, laptop: 1203, is: 12, amazing: 789}

Output: [45, 1203, 12, 789]

# Image Data Characteristics



## Spatial Structure

Pixel relationships and local patterns are critical



## High Dimensionality

Small images contain thousands of features



## Translation Invariance

Objects recognized regardless of position



## Scale Variance

Same object can appear at different sizes



## Color Channels

RGB, grayscale, or other color spaces



## Resource Intensive

Significant storage and computation needed

## Hierarchical Features

Edges



Textures



Parts



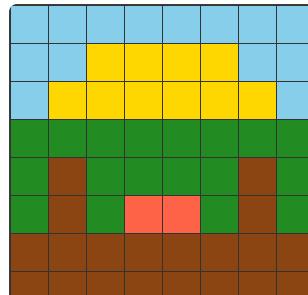
Objects



Requires significant storage and computational resources

Image as Numeric Data: Visual Representation

Original Image



8 × 8 pixels



Split RGB

Color Channels (0-255)

R [135 135 135 135 135 135 135 135]

G [206 206 206 206 206 206 206 206]

B [235 235 235 235 235 235 235 235]

3 channels × 8 × 8



To Tensor

Numeric Tensor

0.529 0.529 ...

0.808 0.808 ...

0.922 0.922 ...

:

Shape: (3, 8, 8)

3  
Channels (RGB)

×

8  
Height

×

8  
Width

=

192

Total Features

# Audio/Speech Data Characteristics



## Temporal Structure

Time-dependent sequential information



## Frequency Domain

Sound is combination of different frequencies



## Sampling Rate

Typically 16kHz-44.1kHz for speech and music



## Variable Length

Utterances and clips have different durations



## Background Noise

Acoustic environment affects quality



## Speaker Variability

Accent, pitch, speed vary across individuals



## Required Time-Frequency Analysis

Spectrograms

MFCCs

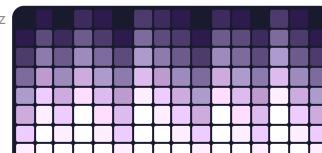
Mel-Frequency Analysis

## Audio as Numeric Data: Visual Representation

Raw Waveform



Mel-Spectrogram



MFCC Features



Numeric Tensor

-2.34	1.52	...
0.87	-1.23	...
3.14	0.45	...

1D Signal: 16,000 samples/sec

STFT

0Hz  
0s      Time      1s

DCT

C1

Flatten

2D: Frequency × Time

13 coefficients × T frames

Shape: (13, T)

**16,000**

Samples/sec

**128**

Mel Bands

**13**

MFCC Coeffs

**T**

Time Frames

**13 × T**

Features

# Video Data Characteristics

Spatial (Image)

Temporal (Sequence)

= Video



High Dimensionality

Multiple frames per second



Motion Patterns

Object & camera motion



Temporal Redundancy

Similar consecutive frames



Frame Rate

24-60 fps typical for video



Resource Requirements

Massive storage & processing power



Optimization Strategy

Can be decomposed into keyframes for efficient processing

## ⌚ How Video is Composed

Frame 1



Frame 2



Frame 3



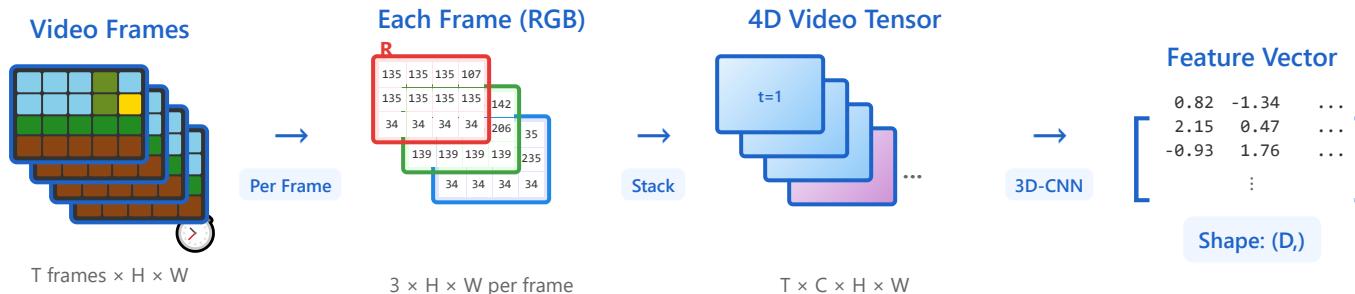
Frame 4



...

**Key Concept:** Video = Sequence of frames where keyframes store complete information, and intermediate frames only store changes (deltas) from previous frames

## Video as Numeric Data: Visual Representation



$$30 \text{ Frames (T)} \times 3 \text{ Channels (C)} \times 224 \text{ Height (H)} \times 224 \text{ Width (W)} = 4.5M \text{ Total Values}$$



### Data Size Comparison (same resolution)

Image: **150K** values

Video (1 sec): **4.5M** values

Ratio: **30x** more data

# Graph/Network Data

Represents relationships and connections between entities



## Nodes (Vertices)

Represent objects or entities



## Edges

Relationships between nodes

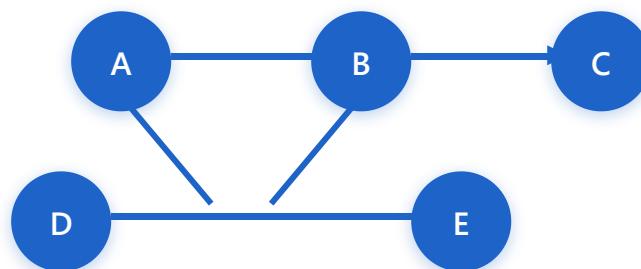
Directed

Undirected

Weighted

Unweighted

## Example Graph Structure



## Common Applications

Social Networks

Molecular Structures

Knowledge Graphs

Non-Euclidean Structure → Requires Specialized Algorithms:

**Graph Neural Networks (GNNs)**

# Graph Feature Extraction Process



## Step-by-Step Feature Extraction

1

### Graph Representation

Convert graph to mathematical structures (adjacency matrix, edge list)

2

### Feature Computation

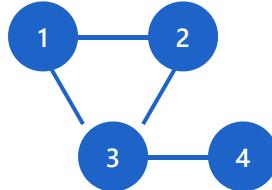
Calculate node-level and graph-level features

3

### Feature Vector

Aggregate features into numerical vectors for ML models

#### Sample Graph



#### Adjacency Matrix

	1	2	3	4
1	0	1	1	0
2	1	0	0	1
3	1	0	0	1
4	0	1	1	0

# Types of Graph Features

## ◆ Node-Level Features

**Degree:** Number of connections (노드 1: degree = 2)

**Centrality:** Importance in the network

**Clustering Coefficient:** How connected neighbors are

**PageRank:** Node influence score

## ◆ Edge-Level Features

**Weight:** Connection strength

**Distance:** Shortest path length

**Common Neighbors:** Shared connections

## ◆ Graph-Level Features

**Number of Nodes:** Total vertices (예시: 4개)

**Number of Edges:** Total connections (예시: 4개)

**Density:** Connectivity ratio

**Diameter:** Maximum distance between nodes

## ◆ Structural Features

**Triangles:** Number of 3-node cycles

**Connected Components:** Separate subgraphs

**Average Path Length:** Mean distance between nodes



**Key Insight:** These features transform non-Euclidean graph data into numerical vectors that machine learning models can process!

# Practical Example: Social Network Analysis



## Friend Network Feature Extraction (실제 수치 예시)

### Graph Structure

Node 1 ——— Node 2

| |

Node 3 ——— Node 4

Edges: (1,2), (1,3), (2,4), (3,4)

### Adjacency Matrix (A)

	1	2	3	4
1	0	1	1	0
2	1	0	0	1
3	1	0	0	1
4	0	1	1	0

### Node Feature Matrix (X)

Node	Degree	Betweenness Centrality	Clustering Coefficient	Feature Vector
Node 1	2	0.333	0.0	[2, 0.333, 0.0]
Node 2	2	0.333	0.0	[2, 0.333, 0.0]
Node 3	2	0.333	0.0	[2, 0.333, 0.0]
Node 4	2	0.333	0.0	[2, 0.333, 0.0]

Nodes ( $|V|$ )

4

Edges ( $|E|$ )

4

Density

0.667

Avg Degree

2.0

# GNN Message Passing: 실제 계산 예시

## 🧠 Node 1의 Feature Update 과정 (Step-by-Step)

### Step 1 Initial Node Features ( $h^0$ )

$h_1^0$   
[2, 0.33, 0]

$h_2^0$   
[2, 0.33, 0]

$h_3^0$   
[2, 0.33, 0]

$h_4^0$   
[2, 0.33, 0]

🟡 Node 1 (target) | 🔵 Neighbors (2,3) | ⚡ Non-neighbor (4)

### Step 2 Aggregate Neighbor Features (Mean Pooling)

Node 1의 이웃: {Node 2, Node 3}

Aggregation:

$$\begin{aligned}m_1 &= (h_2^0 + h_3^0) / 2 \\&= ([2, 0.33, 0] + [2, 0.33, 0]) / 2 \\&= [2, 0.33, 0]\end{aligned}$$

### Step 3 Update Node Feature (Linear + ReLU)

Weight Matrices (예시):

$W_{self} =$   
[0.5, 0.1, 0.0]  
[0.2, 0.4, 0.1]

$W_{neigh} =$   
[0.3, 0.2, 0.1]  
[0.1, 0.3, 0.2]

Update Rule:  $h_1^1 = \text{ReLU}(W_{self} \cdot h_1^0 + W_{neigh} \cdot m_1)$

$$\begin{aligned}W_{self} \cdot h_1^0 &= [0.5 \times 2 + 0.1 \times 0.33 + 0 \times 0, 0.2 \times 2 + 0.4 \times 0.33 + 0.1 \times 0] = [1.033, 0.532] \\W_{neigh} \cdot m_1 &= [0.3 \times 2 + 0.2 \times 0.33 + 0.1 \times 0, 0.1 \times 2 + 0.3 \times 0.33 + 0.2 \times 0] = [0.666, 0.299]\end{aligned}$$

$$h_1^1 = \text{ReLU}([1.033 + 0.666, 0.532 + 0.299])$$

$$= \text{ReLU}([1.699, 0.831])$$

$$= [1.70, 0.83]$$

✨ Result: Updated Feature Vectors ( $h^1$ )

$$h_1^1 = [1.70, 0.83]$$

$$h_2^1 = [1.70, 0.83]$$

$$h_3^1 = [1.70, 0.83]$$

$$h_4^1 = [1.70, 0.83]$$

💡 이 과정을 여러 층(layer)에 걸쳐 반복하면, 노드들은 점점 더 멀리 있는 이웃 정보까지 통합하게 됩니다!

# Feature Extraction Pipeline Summary



## Traditional ML Approach

### 1. Manual Feature Engineering

- Calculate statistical features
- Create handcrafted features
- Limited to known patterns

### 2. Fixed Representation

- Features don't adapt
- Same for all tasks



## GNN Approach

### 1. Learned Representations

- Neural networks learn features
- Captures complex patterns
- Discovers hidden structures

### 2. Task-Specific

- Adapts to specific problems
- End-to-end learning



## Key Takeaway

Graph feature extraction transforms complex relational data into numerical vectors  
that preserve structural information for machine learning tasks.

**GNNs automate and optimize this process through learned representations!**

# Multimodal Data

Combines information from multiple modalities simultaneously



Video with Audio



Image with Captions



Sensor with Location



## Benefits

- Complementary information beyond single modality
- Improved robustness: other modalities compensate for noise
- Richer understanding of data



## Challenges

- Alignment across modalities
- Synchronization of data streams
- Fusion of different modalities

## Requirements

Cross-Modal Learning

Joint Representations

## Applications

Video Understanding

Visual Question Answering

Robotics

## **Part 2/3:**

# **Traditional Feature Extraction**

- 9.** Feature Engineering Principles
- 10.** Text - BoW, TF-IDF
- 11.** Text - N-gram, POS
- 12.** Image - Edge, Corner Detection
- 13.** Image - SIFT, SURF, HOG
- 14.** Audio - FFT, Spectrogram
- 15.** Audio - MFCC, Chroma
- 16.** Time Series - Statistical Features

# Feature Engineering Principles

Features are measurable properties used to represent data for ML models

## Good Features Are

✓ Informative

✓ Discriminative

✓ Independent



Domain knowledge is crucial for effective feature engineering



Feature quality often more important than model complexity



Consider relevance, redundancy, and computational cost



Feature scaling and normalization improve model performance



Feature quality often more important than model complexity

## Iterative Process

Extract



Evaluate



Refine



# Text Feature Extraction: BoW & TF-IDF



## Bag of Words (BoW)

Represents text as word frequency counts

- Ignores word order and grammar
- Only considers word occurrence
- Simple frequency-based representation



## TF-IDF

Term Frequency-Inverse Document Frequency weighting

- Balances local and global importance
- Reduces weight of common words
- Increases weight of rare words

### TF-IDF Components

#### TF (Term Frequency)

How often word appears in document (local importance)



#### IDF (Inverse Document Frequency)

How rare word is across documents (global importance)

✓ **Advantages:** Simple and interpretable

✗ **Limitations:** Loses semantic and syntactic information

## Bag of Words (BoW) - Practical Example

### Example Documents:

**Doc 1:** "I love machine learning"

**Doc 2:** "Machine learning is amazing"

**Doc 3:** "I love deep learning"

### Step 1: Build Vocabulary

Unique words across all documents:

I love machine learning is amazing deep

### Step 2: Count Word Frequencies

Document	I	love	machine	learning	is	amazing	deep
Doc 1	1	1	1	1	0	0	0
Doc 2	0	0	1	1	1	1	0
Doc 3	1	1	0	1	0	0	1

 **Result Interpretation:** Each document is represented as a 7-dimensional vector. Example: Doc 1 = [1, 1, 1, 1, 0, 0, 0]  
→ Word order and grammar are ignored; only word occurrence counts are considered.

## TF-IDF - Practical Example

### Same Example Documents:

**Doc 1:** "I love machine learning"

**Doc 2:** "Machine learning is amazing"

**Doc 3:** "I love deep learning"

### TF-IDF Formula:

$$\text{TF-IDF}(\text{word}, \text{doc}) = \text{TF}(\text{word}, \text{doc}) \times \text{IDF}(\text{word})$$

$$\text{TF}(\text{word}, \text{doc}) = (\text{word count in doc}) / (\text{total words in doc})$$

$$\text{IDF}(\text{word}) = \log(\text{total documents} / \text{documents containing word})$$

**Example:** Calculate TF-IDF for "learning" in Doc 1

### Step 1: Calculate TF

$$TF("learning", Doc 1) = 1 / 4 = 0.25$$

(appears 1 time / total 4 words)

### Step 2: Calculate IDF

$$IDF("learning") = \log(3 / 3) = \log(1) = 0$$

(3 documents / 3 documents containing "learning")

### Step 3: Calculate TF-IDF

$$TF-IDF("learning", Doc 1) = 0.25 \times 0 = 0$$

→ "learning" appears in all documents, so it has low importance!

## Example: Calculate TF-IDF for "deep" in Doc 3

### Step 1: Calculate TF

$$TF("deep", Doc 3) = 1 / 4 = 0.25$$

### Step 2: Calculate IDF

$$IDF("deep") = \log(3 / 1) = \log(3) \approx 1.099$$

("deep" appears in only 1 document → rare word)

### Step 3: Calculate TF-IDF

$$TF-IDF("deep", Doc 3) = 0.25 \times 1.099 \approx 0.275$$

→ "deep" is a rare word, so it has high importance!

 **Key Point:** TF-IDF reduces the weight of common words (e.g., "learning") and increases the weight of rare words (e.g., "deep") to better represent document characteristics.

# Text Feature Extraction: N-grams & POS



## N-grams

Contiguous sequences of N words

### Examples:

Bigrams (2): "machine learning"  
Trigrams (3): "natural language processing"

- ✓ Capture local word order
- ✓ Identify common phrases
- ✓ Provide more context than single words



## POS Tagging

Identifies grammatical roles of words

### Tags:

Noun, Verb, Adjective, Adverb...

- ✓ Identify syntactic patterns
- ✓ Analyze sentence structure
- ✓ Distinguish word usage contexts



## Combined Benefits

Using N-grams and POS tags together improves text classification and information extraction



## Trade-off

Between context capture and computational complexity (N-grams increase feature space)



## Practical Processing Examples

## 1 Input Sentence:

"I love machine learning"



## 2 N-gram Extraction:

Unigrams (1):

["I", "love", "machine", "learning"]

Bigrams (2):

["I love", "love machine", "machine learning"]

Trigrams (3):

["I love machine", "love machine learning"]

## Features Generated:

Feature Vector:

[1, 1, 1, 1, 1, 1, 1, ...]

(Total: 4 unigrams + 3 bigrams + 2 trigrams)

## 1 Input Sentence:

"The quick brown fox jumps"



## 2 POS Tagging:

The/DET quick/ADJ brown/ADJ  
fox/NOUN jumps/VERB

Tags Extracted:

- DET: Determiner (1)
- ADJ: Adjective (2)
- NOUN: Noun (1)
- VERB: Verb (1)



## Features Generated:

POS Pattern:

"DET-ADJ-ADJ-NOUN-VERB"

POS Counts:

[DET:1, ADJ:2, NOUN:1, VERB:1]

# Image Feature Extraction: Edge & Corner Detection



## Edge Detection

Boundaries where intensity changes sharply

### Detection Operators:

Sobel

Prewitt

Canny



## Corner Detection

Points where edges intersect or change direction rapidly

### Detection Method:

**Harris Corner Detector:** Identifies distinctive points for matching



## Feature Hierarchy

Low-level features (edges, corners) are fundamental for higher-level understanding

Foundation for object detection and image matching



Robust to illumination changes



Resilient to minor transformations



## Visual Examples



### Edge Detection - Building Example

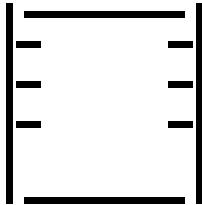
Original Image

Sobel/Canny Edge

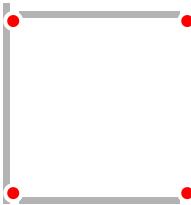
Harris Corner



Full color building with texture and details



Outlines extracted - object boundaries detected



Key points identified - useful for matching



## Use Cases

### Object Recognition



Edge detection helps identify object boundaries for classification and recognition

### Image Matching



Corner points serve as landmarks for matching images across different views

### Motion Tracking



Corners are stable features that can be tracked across video frames

# Image Descriptors: SIFT, SURF & HOG



## SIFT

Scale-Invariant Feature Transform

Detects and describes local features

- ✓ Scale invariant
- ✓ Rotation invariant
- ✓ Partial illumination invariance



## SURF

Speeded Up Robust Features

Faster approximation of SIFT

- ✓ Improved speed
- ✓ Similar robustness
- ✓ Efficient computation



## HOG

Histogram of Oriented Gradients

Captures edge orientation distribution

- ✓ Shape description
- ✓ Edge patterns
- ✓ Object detection

## Capabilities

Enable robust object recognition and image matching



**Historical Context:** Widely used before deep learning era for computer vision tasks



**Current Use:** Still valuable for lightweight applications and limited data scenarios

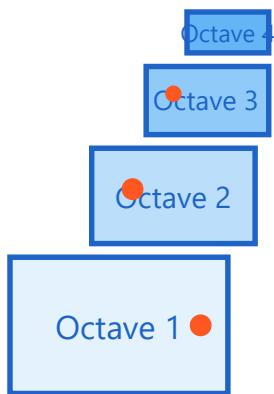


# SIFT: Scale-Invariant Feature Transform

## 💡 Core Principle

SIFT finds feature points in **scale space** and generates descriptors using **gradient orientation histograms** around each keypoint.

### DoG Pyramid (Difference of Gaussians)



### Multiple Scales

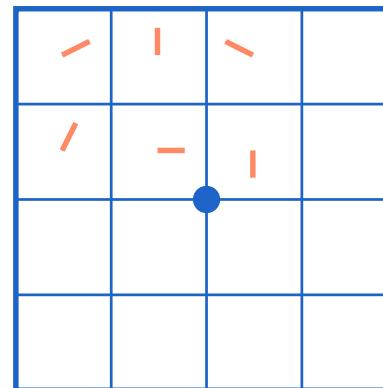
Apply Gaussian blur at each scale

#### ● Keypoints

Detected at extrema locations

Divide image into multiple scales and compute DoG to find scale-invariant features

### SIFT Descriptor (128-dimensional)



$4 \times 4 = 16$  blocks

Each block:  
8-bin histogram

$16 \times 8 = 128$   
dimensional vector

Divide 16x16 region into 4x4 blocks and compute 8-direction gradient histogram per block

1

### Scale-Space Extrema Detection

Find extrema in DoG pyramid → Detect scale-invariant keypoint candidates



2

### Keypoint Localization

Refine with Taylor expansion → Remove unstable points



**3**

### Orientation Assignment

Gradient orientation histogram → Assign dominant orientation (rotation invariance)

**4**

### Descriptor Generation

$4 \times 4 \text{ blocks} \times 8 \text{ orientations} = 128\text{-dimensional feature vector}$

# SURF: Speeded Up Robust Features

## Core Principle

SURF uses **Integral Image** and **Box Filters** to approximate SIFT faster. Detects keypoints based on Hessian matrix.

### Integral Image

Original Image  
3 5 2 ...  
1 7 4 ...  
6 2 8 ...

Trans

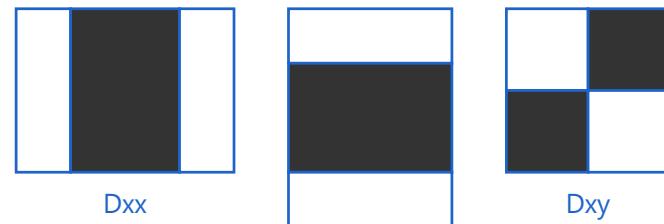
Integral Image  
3 8 10 ...  
4 16 22 ...  
10 24 38 ...

### Fast Region Sum

Calculate sum of any rectangle in 4 operations

With Integral Image, box filters can be computed in  $O(1)$  time

### Fast-Hessian Detector



### Approximate Hessian with Box Filters

$$\text{Det}(H) \approx D_{xx} \cdot D_{yy} - (0.9 \cdot D_{xy})^2$$

Max value location = Keypoint

Quickly approximate Hessian matrix with 3 box filters to detect blobs

1

### Generate Integral Image

Original image  $\rightarrow$  Integral Image (preprocessing)



2

### Fast-Hessian Detector

Calculate Hessian determinant with box filters  $\rightarrow$  Detect keypoints



### Orientation via Haar Wavelet

**3** Determine dominant orientation using Haar wavelet responses



**4** **64D Descriptor**

$4 \times 4$  sub-regions  $\times$  4D Haar response = 64-dimensional vector

⚡ **Speed Improvement:** 2-3x faster than SIFT, with Integral Image enabling same speed for all scales



# HOG: Histogram of Oriented Gradients

## 💡 Core Principle

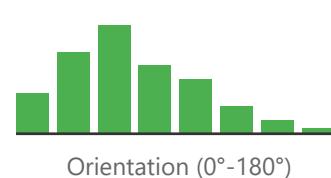
HOG divides images into small **cells** and computes **gradient orientation histograms** in each cell. Normalizes by blocks to create illumination-robust shape descriptors.

### Gradient Computation & Orientation



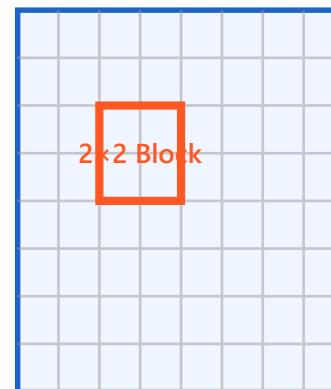
Aggregate gradient orientations in each cell ( $8 \times 8$ ) to 9 bins

### 9-bin Histogram



Calculate gradient magnitude and direction at each pixel and aggregate into histogram

### Cell & Block Structure



64×128 Window

Group 2 $\times$ 2 cells into blocks and normalize to make robust against illumination changes

#### Structure:

- Cell:  $8 \times 8$  pixels
- Block: 2 $\times$ 2 cells
- Stride: 8 pixels

#### Dimension:

7 blocks (H)  $\times$  15 blocks (V)  
 $\times$  4 cells  $\times$  9 orientations  
= 3,780 dimensions

1

### Gradient Computation

Calculate x, y direction gradients at each pixel (using [-1,0,1] filter)



2

### Orientation Binning

Divide into  $8 \times 8$  cells, generate 9-bin orientation histogram in each cell



**3****Block Normalization**

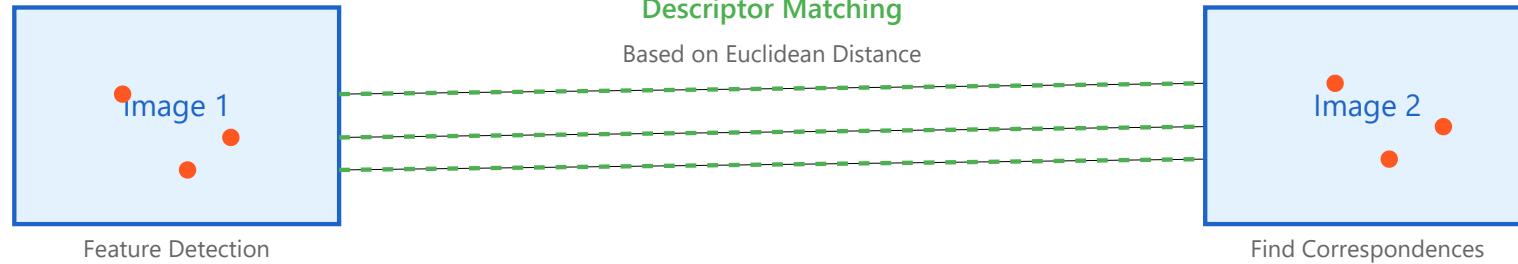
Group into  $2 \times 2$  cell blocks and apply L2 normalization (50% overlap)

**4****Feature Vector**

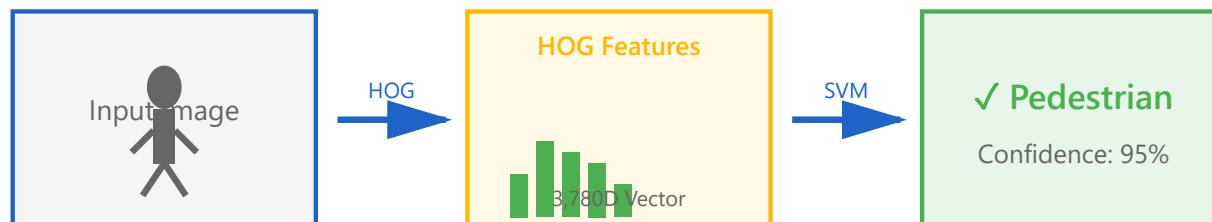
Concatenate all block histograms  $\rightarrow$  3,780-dimensional descriptor

# Real-world Applications

## Image Matching (SIFT/SURF)



## Object Detection (HOG + SVM)



Extract HOG features from sliding windows and detect pedestrians with pre-trained SVM

# Detailed Algorithm Comparison

Characteristic	SIFT	SURF	HOG
<b>Core Method</b>	DoG (Difference of Gaussians)	Fast-Hessian (Box Filter)	Gradient Orientation Histogram
<b>Feature Detection</b>	Scale-space extrema	Hessian determinant maxima	Dense sampling (entire window)
<b>Descriptor Dimension</b>	128-dimensional	64-D (standard) / 128-D	3,780-D (64x128 window)
<b>Computation Speed</b>	Slow (baseline)	Fast (2-3x faster than SIFT)	Medium (depends on window size)
<b>Scale Invariance</b>	Excellent	Excellent	Limited (fixed window)
<b>Rotation Invariance</b>	Excellent	Excellent	Limited
<b>Main Applications</b>	Image matching, panorama, 3D reconstruction	Real-time tracking, video processing	Pedestrian detection, object recognition
<b>Patent Status</b>	Expired (2020)	Patent protected	Free to use

## Selection Guide

**Need high accuracy** → SIFT (image stitching, 3D reconstruction, object recognition)

**Need real-time processing** → SURF (video tracking, AR applications, real-time matching)

**Object detection goal** → HOG + SVM (pedestrian, vehicle, face detection)



**Key Difference:** SIFT/SURF are keypoint-based, HOG is dense descriptor. Choose based on your application.



**Deep Learning Era:** These algorithms are still useful for limited data or lightweight systems.

# Audio Feature Extraction: FFT & Spectrogram



## Fast Fourier Transform (FFT)

Frequency Domain Transformation



Reveals frequency components present in audio signal



## Spectrogram

Visual representation of frequency spectrum over time

### Temporal Info

Time

### Spectral Info

Frequency

**STFT (Short-Time Fourier Transform):** Applies FFT to windowed segments



## Essential Applications



Audio Analysis



Speech Recognition



Music Processing



## Trade-off

Between time and frequency resolution (determined by window size)

# Audio Features: MFCC & Chroma



## MFCC

Mel-Frequency Cepstral Coefficients

Models human auditory perception

**Based on:** Mel-scale (perceptually motivated frequency scale)

- Speech recognition
- Speaker identification
- Audio classification



## Chroma Features

Pitch Content & Harmonic Structure

Represent pitch content and harmonic structure

**Focus on:** 12 pitch classes (chromatic scale)

- Music analysis
- Chord recognition
- Similarity detection



Both features capture complementary aspects of audio signals



Compact representations suitable for machine learning models



# Visual Process Flow



## MFCC Extraction Process

### 1 Audio Signal

Input original audio signal (time domain)

### 2 STFT

Frequency analysis with Short-Time Fourier Transform

### 3 Mel Filter Bank

Apply Mel-scale filters that reflect human auditory characteristics

### 4 DCT

Extract coefficients with Discrete Cosine Transform



### MFCC Coefficient Example (13 coefficients)



## Chroma Extraction Process

### 1 Audio Signal

Input original music signal

### 2 STFT

Analyze frequency spectrum

### 3 Pitch Mapping

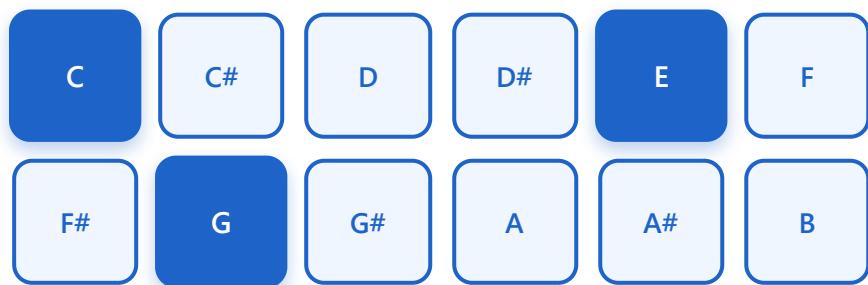
Map frequencies to 12 pitch classes (C, C#, D, ...)

### 4 Octave Folding

Fold all octaves into one (octave invariance)



### Chroma Vector Example (12 pitch classes)



*Each coefficient represents different frequency characteristics of the audio*

*Highlighted notes = C Major chord (C, E, G)*

*The intensity of each pitch class is represented as a vector value*

# Time Series Statistical Features



## Basic Statistics

Mean

Median

Std Dev

Basic statistical properties of the signal



## Amplitude Variations

Min

Max

Range

Capture the extent of signal variations



## Autocorrelation

Measures signal's correlation with itself at different time lags



## Spectral Features

Dominant Frequency

Spectral Entropy

Frequency domain characteristics



## Temporal Patterns

Trend

Seasonality

Cyclic Components



## Rolling Window Statistics

- Moving average
- Exponential smoothing



## Domain-Specific Features

- Peak detection
- Zero-crossing rate

## **Part 3/3:**

# **Learning-based Representations**

- 17.** Concept of Representation Learning
- 18.** Word Embeddings (Word2Vec, GloVe)
- 19.** CNN-based Image Features
- 20.** RNN-based Sequence Features
- 21.** Autoencoders and Latent Representations
- 22.** Transfer Learning Strategies
- 23.** Fine-tuning vs Feature Extraction
- 24.** Domain Adaptation
- 25.** Multimodal Fusion

# Concept of Representation Learning

Automatically learns features from raw data instead of manual engineering

## Hierarchical Representations

### Lower Layers

Capture simple patterns

### Higher Layers

Capture complex concepts



End-to-end learning: features optimized for specific task



Reduces need for domain expertise in feature design



Can discover non-obvious patterns humans might miss



Discovers multiple levels of abstraction automatically



Foundation of Modern Deep Learning Success

# Word Embeddings: Word2Vec & GloVe

Dense vector representations that capture semantic relationships



## Word2Vec

Learns embeddings from local context

CBOW

Skip-gram



## GloVe

Learns from global word co-occurrence statistics

### Key Properties



Similar words have similar vectors



Much lower dimensions than vocabulary



Transfer learning to downstream tasks



Captures semantic analogies



### Classic Analogy Example

king - man + woman ≈ queen



Typical Dimensions: **100-300** (vs. thousands in vocabulary size)

## Word2Vec Training Process (Window Size = 2)

### CBOW (Continuous Bag of Words)

"The **quick** brown **fox** jumps over "

#### Step 1: One-Hot Encoding ↓

##### Context Words (Input):

```
quick: [0,1,0,0,0,...] (vocab size)  
brown: [0,0,1,0,0,...]  
jumps: [0,0,0,1,0,...]  
over: [0,0,0,0,1,...]
```

Input Layer (One-Hot)



Hidden Layer (Embeddings)



Output Layer (Softmax)

##### Target (Output):

```
fox: [0,0,0,0,0,1,0,...] (one-hot)
```

 Predict center word from averaged context embeddings

### Skip-gram

"The quick brown **fox** jumps over"

#### Step 1: One-Hot Encoding ↓

##### Center Word (Input):

```
fox: [0,0,0,0,0,1,0,...] (vocab size)
```

Input Layer (One-Hot)



Hidden Layer (Embeddings)



Output Layer (Softmax) ×4

##### Target (4 Context Words):

```
quick: [0,1,0,0,0,...]  
brown: [0,0,1,0,0,...]  
jumps: [0,0,0,1,0,...]  
over: [0,0,0,0,1,...]
```

 Predict each context word from center word embedding

# CNN-based Image Features

Convolutional Neural Networks learn hierarchical visual features

## Early Layers

Edges  
Textures  
Colors

## Feature Hierarchy Across Layers

## Middle Layers

Object Parts  
Patterns

## Deep Layers

High-level Semantic  
Concepts & Objects



Convolutional filters automatically learned from data



Pooling operations provide translation invariance



## Pre-trained CNNs as Feature Extractors

VGG, ResNet, and more serve as powerful feature extractors

# Historical Context of CNNs

## The evolution of Convolutional Neural Networks in computer vision

1989

### LeNet (Yann LeCun)

First successful CNN, used for handwritten digit recognition. Applied to automatic postal code recognition systems

2012

### AlexNet (ImageNet Challenge)

Marked the resurgence of deep learning. Achieved overwhelming performance through large-scale GPU training

2014

### VGGNet & GoogLeNet

Explored deeper network architectures. VGG featured simple, uniform structure; GoogLeNet introduced Inception modules

2015

### ResNet (Residual Networks)

Enabled training of very deep networks (152 layers) with skip connections. Achieved human-level image recognition performance

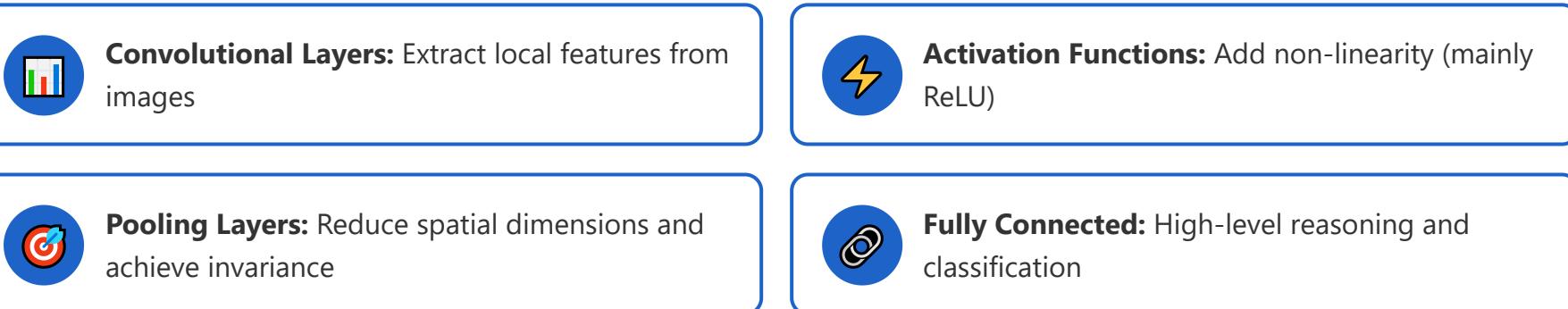
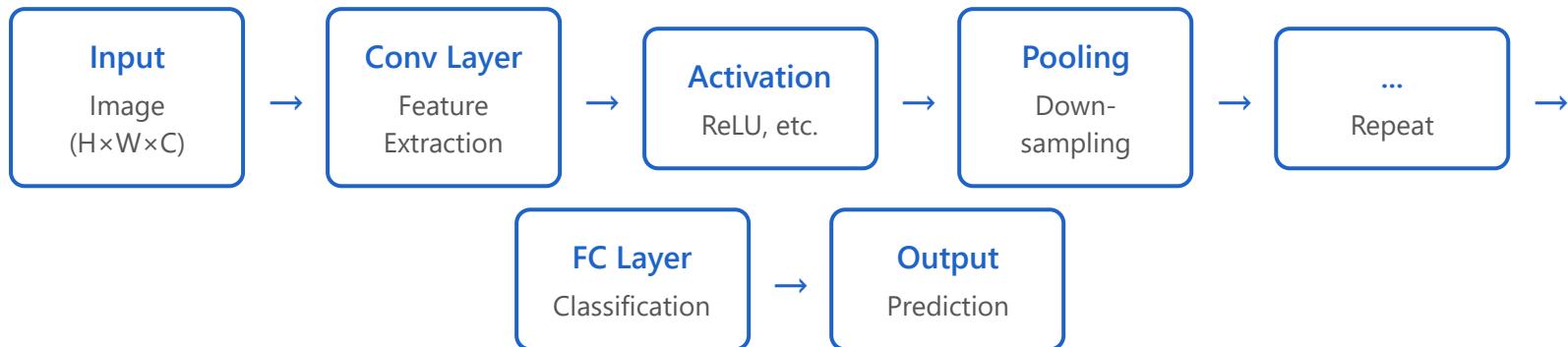
2017+

### Modern Architectures

Emergence of modern architectures like EfficientNet and Vision Transformer (ViT) pursuing both efficiency and performance

# CNN Layer Architecture

## Typical structure of a Convolutional Neural Network



**Key Principle:** Learn hierarchical features by repeating Conv-Activation-Pooling blocks multiple times

# Convolution & Pooling Operations



## Convolution Operation

### How it Works

A small filter (kernel) slides over the image, performing element-wise multiplication and summation

### Key Parameters

- **Filter size:**  $3 \times 3$ ,  $5 \times 5$ , etc.
- **Stride:** Step size
- **Padding:** Border handling

### Features

- Detects local patterns
- Parameter efficiency through weight sharing
- Preserves spatial relationships

### Learning

Filter weights are automatically learned from data through backpropagation



## Pooling Operation

### Purpose

Reduces spatial dimensions of feature maps, decreases computation, and achieves invariance

### Max Pooling

Selects maximum value within window. Preserves strongest features. Most widely used

### Average Pooling

Computes average value within window. Extracts smoother features

### Advantages

- Invariance to small translations/deformations
- Reduces overfitting
- Improves computational efficiency
- Expands receptive field



**Together:** Convolution extracts features, and Pooling compresses and makes them more robust

# Key Concepts in CNN Operations

## Receptive Field

- The region of input image that a single neuron "sees"
- Deeper layers have wider receptive fields
- Enables capturing larger contextual information

1  
2  
3  
4

## Output Size Calculation

$$\text{Output size} = \lfloor (\text{Input size} - \text{Filter size} + 2 \times \text{Padding}) / \text{Stride} \rfloor + 1$$

Example: Input 32×32, Filter 5×5, stride 1, padding 2

$$\rightarrow \text{Output: } \lfloor (32 - 5 + 4) / 1 \rfloor + 1 = 32 \text{ (size preserved)}$$

## Why CNNs Work Well

### **1. Translation Invariance:**

Can recognize objects even when their position changes

### **2. Hierarchical Learning:**

Simple features → Complex features

### **3. Parameter Sharing:**

Same filter applied across entire image

### **4. Sparse Connectivity:**

Efficient learning through local connections

## Feature Visualization

### **Early layers:**

Basic elements like edges, corners, color blobs

### **Middle layers:**

Textures, repetitive patterns, object parts

### **Deep layers:**

Complete objects, scenes, semantic concepts

Filters in each layer learn increasingly complex and abstract features

# Interactive Learning Resource

Experience how CNNs work and visually understand the impact of hyperparameters



## CNN Explainer

Interactive visualization of Convolutional Neural Networks



[Visit CNN Explainer](#)



**Interactive Demo:** Visualize what happens in each CNN layer in real-time



**Hyperparameters:** Directly see the effects of kernel size, stride, padding, etc.



**Feature Maps:** Observe outputs and activations of each layer in real-time



**Educational:** The best tool to intuitively understand how CNNs work

### Learning Tips

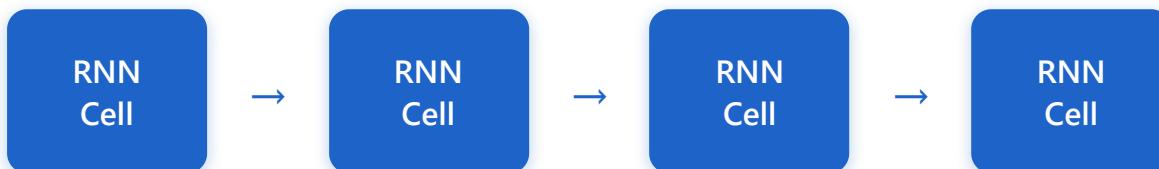
- Start with the "Understanding Hyperparameters" section to learn basic concepts
- Change various hyperparameter values and observe how the output changes

- Follow step-by-step how CNNs respond to real images

# RNN-based Sequence Features

Recurrent Neural Networks process sequential data with temporal dependencies

## Sequential Processing Architecture



🧠 Hidden states maintain memory of previous inputs

## RNN Variants

### LSTM

Long Short-Term Memory

### GRU

Gated Recurrent Unit

### Bidirectional

Past + Future Context

✓ Address vanishing gradient problem



Model variable-length sequences naturally

## Applications

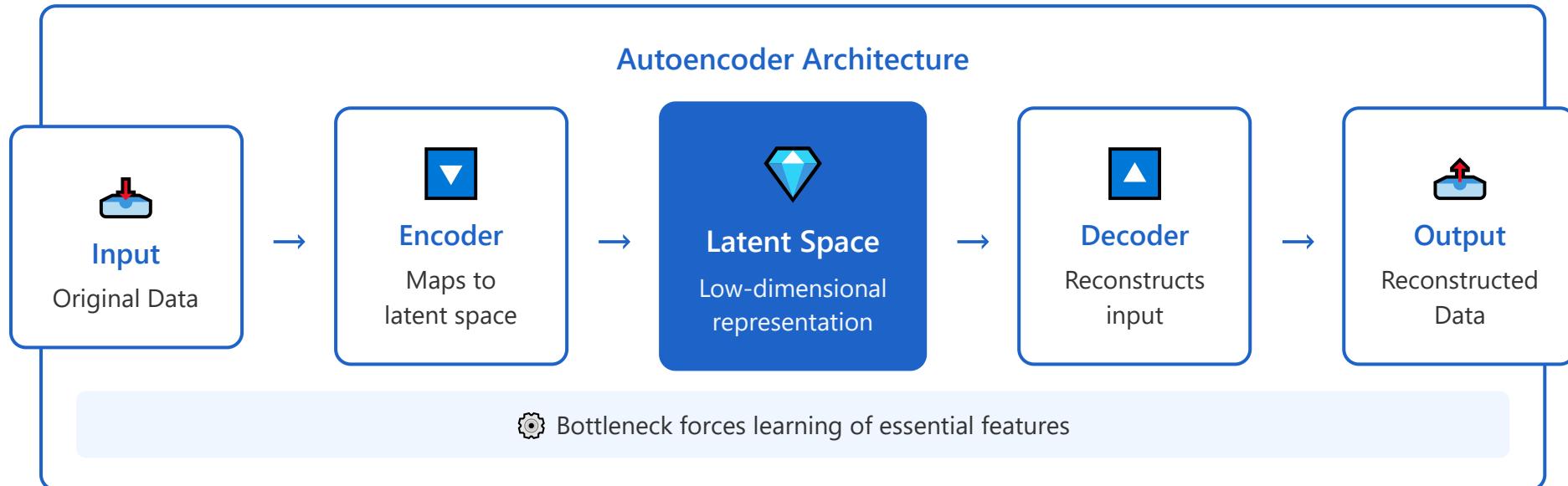
💬 Language Modeling

🗣 Speech Recognition

📈 Time Series

# Autoencoders and Latent Representations

Unsupervised learning of compressed data representations



## 📖 Historical Context

**1980s:** Hinton and Rumelhart pioneered dimensionality reduction using backpropagation

**2006:** Hinton's Deep Belief Networks marked the dawn of the deep learning era

**2013:** Kingma and Welling proposed the Variational Autoencoder (VAE)

**Present:** Evolved into a core technology for generative models, anomaly detection, and representation learning

## 🏗 Structural Details

**Encoder:**  $x \rightarrow z = f_{enc}(x; \theta_{enc})$

- Compresses input dimension  $n$  to latent dimension  $d$  ( $d \ll n$ )
- Composed of multiple neural network layers (e.g., FC layers, CNNs)

**Decoder:**  $z \rightarrow \hat{x} = f_{\text{dec}}(z; \theta_{\text{dec}})$

- Reconstructs original input from latent representation  $z$
- Can be symmetric or independent from encoder architecture

**Loss Function:**  $L = \|x - \hat{x}\|^2$  (reconstruction error)

## Vector Operation Example

**Simple Example:** 784-dimensional MNIST image  $\rightarrow$  2-dimensional latent space

Input vector:  $x \in \mathbb{R}^{784}$  (28×28 image flattened)

e.g.,  $x = [0.1, 0.0, 0.8, \dots, 0.3]$

Encoder operations:

$h_1 = \text{ReLU}(W_1 x + b_1)$  #  $W_1 \in \mathbb{R}^{(128 \times 784)}$

$h_2 = \text{ReLU}(W_2 h_1 + b_2)$  #  $W_2 \in \mathbb{R}^{(64 \times 128)}$

$z = W_3 h_2 + b_3$  #  $W_3 \in \mathbb{R}^{(2 \times 64)}$ ,  $z \in \mathbb{R}^2$

$\rightarrow z = [1.2, -0.5]$  # 2D latent vector

Decoder operations:

$h_3 = \text{ReLU}(W_4 z + b_4)$  #  $W_4 \in \mathbb{R}^{(64 \times 2)}$

$h_4 = \text{ReLU}(W_5 h_3 + b_5)$  #  $W_5 \in \mathbb{R}^{(128 \times 64)}$

$\hat{x} = \sigma(W_6 h_4 + b_6)$  #  $W_6 \in \mathbb{R}^{(784 \times 128)}$

$\rightarrow \hat{x} = [0.09, 0.02, 0.79, \dots, 0.31]$

Loss:  $\text{MSE} = (1/784) \sum (x_i - \hat{x}_i)^2 = 0.003$

## Variational Autoencoders (VAE)

Learn probabilistic latent distributions for generation and sampling

- Models latent space as probability distribution in the form  $z \sim N(\mu, \sigma^2)$
- Learns regularized latent space with additional KL divergence



Dimensionality  
Reduction



Denoising

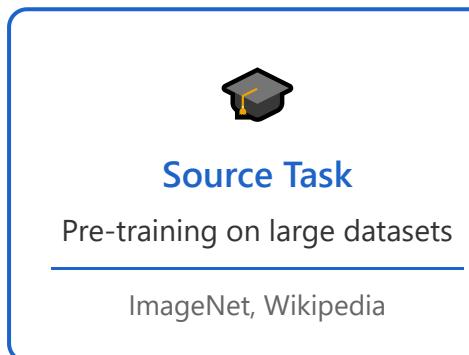


Generation

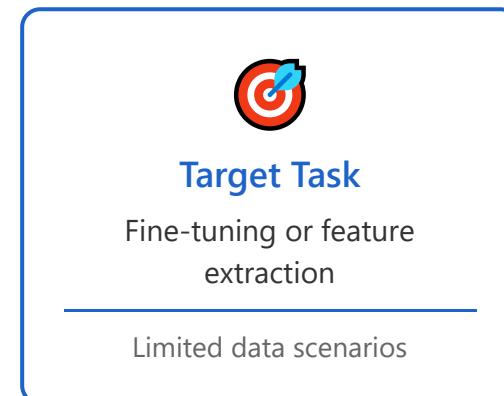
# Transfer Learning Strategies

Leverage knowledge from source task to improve target task performance

## Transfer Learning Process



Knowledge Transfer



Particularly effective when target task has limited data



Source and target tasks should be related for best results



Reduces training time significantly



Reduces computational requirements



Key Technique in Modern Deep Learning Applications



Transfer Learning Performance Comparison

### Training from Scratch

Small Dataset (500 images)

Training Time: 8 hours

Accuracy: 65%

Epochs: 200

**VS**

### Transfer Learning

Same Dataset (500 images)

Training Time: 1 hour

Accuracy: 92%

Epochs: 20



**8x faster training and 27% higher accuracy** with pre-trained ImageNet model

# Fine-tuning vs Feature Extraction



## Feature Extraction

Freeze pre-trained model, train only new classifier

- ✓ Faster training
- ✓ Requires less data
- ✓ Lower computational cost
- ✓ Fixed feature representation



## Fine-tuning

Update pre-trained weights with small learning rate

- ✓ More task-specific adaptation
- ✓ Better final performance
- ✓ Requires more resources
- ✓ Adapts feature representation



## Decision Factors

Dataset Size

Task Similarity

Available Resources



## Typical Strategy

1. Feature Extraction



2. Evaluate



3. Fine-tune if needed



## Advanced Technique

Layer-wise fine-tuning: Gradually unfreeze deeper layers for progressive adaptation

# Domain Adaptation

Addresses distribution shift between training (source) and test (target) domains

## Domain Distribution Shift



### Source Domain

Training data distribution

Labeled data available



### Distribution Shift



### Target Domain

Test data distribution

Limited/no labels

## Example Scenario

Train on synthetic data → Test on real data

## Adaptation Techniques

Domain Adversarial Training

Self-Training

Pseudo-Labeling



Learn domain-invariant representations



Important for real-world deployment

# Transfer Learning vs Domain Adaptation



## Transfer Learning

**Core:** Apply knowledge from one task to a **different task**

**Changes:** Task changes

**Scope:** Broader concept (parent)

Dog/Cat Classification



Cancer Diagnosis



## Domain Adaptation

**Core:** **Same task**, adapt to different data distribution

**Changes:** Distribution changes

**Scope:** Subset of Transfer Learning

Daytime Photos

Nighttime Photos

## Conceptual Hierarchy

### Transfer Learning



#### Domain Adaptation

Same task, different distribution

#### Fine-tuning

Adjust pretrained weights

#### Multi-task Learning

Multiple tasks together

## Key Difference Summary



**Transfer Learning:** Leverage existing knowledge to solve a different problem (Task) | **Domain Adaptation:** Adapt to different data environments (Distribution) while solving the same problem

# Multimodal Fusion

Combines representations from multiple modalities for unified understanding



## Early Fusion

Combine raw features before processing



## Intermediate Fusion

Combine learned representations at hidden layers



## Late Fusion

Combine decisions from separate modality models

## Advanced Techniques



**Attention Mechanisms:** Focus on relevant modalities



**Cross-modal Learning:** Use one modality to improve another



## Applications

Video Understanding

Visual QA

Multimodal Sentiment Analysis



Improves robustness and performance over single-modality approaches

# Thank you

Ho-min Park

[homin.park@ghent.ac.kr](mailto:homin.park@ghent.ac.kr)

[powersimmani@gmail.com](mailto:powersimmani@gmail.com)