Lecture 10:

Data Modality and Feature Extraction

Ho-min Park

homin.park@ghent.ac.kr

powersimmani@gmail.com

Lecture Contents

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
- Different storage requirements
- Specific processing methods needed
- 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

Structured Data Examples

Classification Example

Classification with PCA Features

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

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**
950	1	15	8.5	**245**
2100	3	8	4.3	**560**

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**
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

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

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 Keyframe → Frame 2 Delta → Frame 3 Delta → Frame 4 Keyframe → ...

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

Graph/Network Data

Represents relationships and connections between entities

Nodes (Vertices)

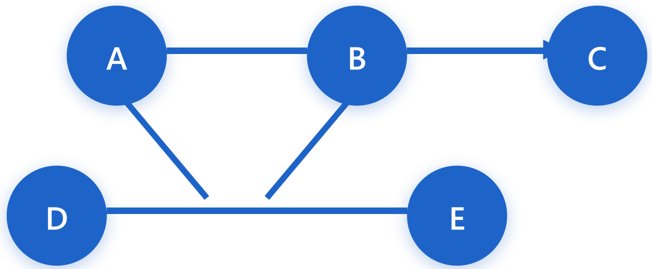
Represent objects or entities

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

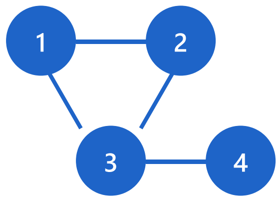
Calculate node-level and graph-level features

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

```
# Python example using NetworkX import networkx as nx import numpy as np # 1. Create graph G = nx.Graph() G.add_edges_from([(1,2), (1,3), (2,4), (3,4)]) # 2. Extract node features degrees = dict(G.degree()) # {1:2, 2:2, 3:2, 4:2} centrality = nx.betweenness_centrality(G) clustering = nx.clustering(G) # 3. Extract graph features num_nodes = G.number_of_nodes() # 4 num_edges = G.number_of_edges() # 4 density = nx.density(G) # 0.667 avg_degree = np.mean(list(degrees.values())) # 2.0 # 4. Create feature vector for Node 1 node_1_features = [ degrees[1], # degree: 2 centrality[1], # centrality: 0.0 clustering[1] # clustering: 1.0 ] # Result: [2, 0.0, 1.0]
```

🧠 Graph Neural Network (GNN) Process

1

Initial Features

Each node starts with feature vector (e.g., [2, 0.0, 1.0])

2

Message Passing

Nodes aggregate features from neighbors

3

Feature Update

Neural network combines local + neighbor features

```
# GNN-style feature aggregation (simplified) def aggregate_features(node, neighbors, features): # Gather neighbor features neighbor_features = [features[n] for n in neighbors] # Aggregate (e.g., mean pooling) aggregated = np.mean(neighbor_features, axis=0) # Combine with node's own features updated = neural_network([features[node], aggregated]) return updated # For Node 1 with neighbors [2, 3]: # Input: own features + aggregated neighbor features # Output: enriched representation capturing local graph structure
```

Feature Extraction Pipeline Summary

Raw Graph
Nodes + Edges

→

Mathematical Representation
Adjacency Matrix / Edge List

→

Feature Extraction
Degree, Centrality, etc.

→

ML-Ready Vectors
[2, 0.0, 1.0, ...]

🎯 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:

```
TF-IDF(word, doc) = TF(word, doc) × IDF(word)
```

```
TF(word, doc) = (word count in doc) / (total words in doc)
```

```
IDF(word) = log(total documents / 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 × 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) ≈ **1.099**
("deep" appears in only 1 document → rare word)

Step 3: Calculate TF-IDF
TF-IDF("deep", Doc 3) = 0.25 × 1.099 ≈ **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

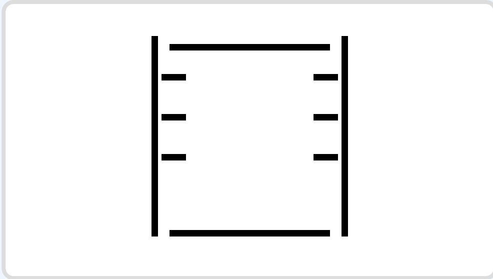
🔄 Resilient to minor transformations

🖼️ Visual Examples

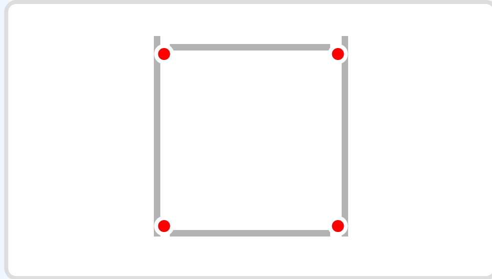
🏢 Edge Detection - Building Example

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

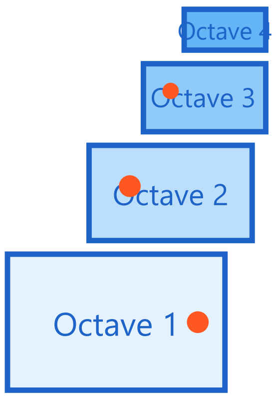
Current Use: Still valuable for lightweight applications and limited data scenarios

🔍 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

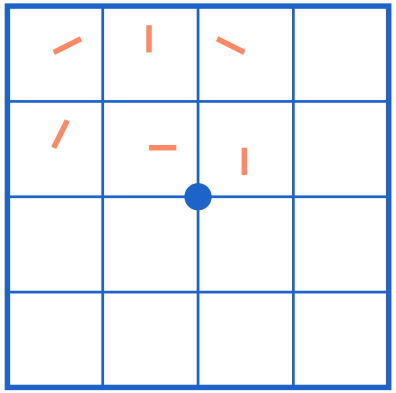
Apply Gaussian blur at each scale

● Keypoints

Detected at extrema locations

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

SIFT Descriptor (128-dimensional)



4×4 = 16 blocks

Each block: 8-bin histogram

16 × 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

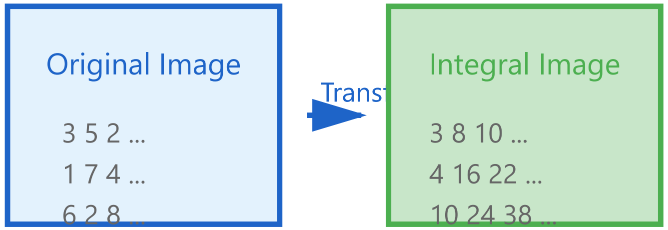
4x4 blocks × 8 orientations = 128-dimensional feature vector

⚡ 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

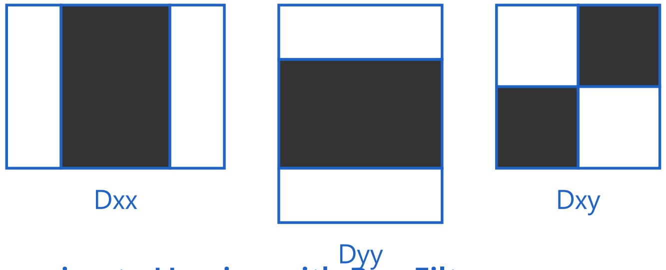


Fast Region Sum

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

$\text{Det}(H) \approx \text{Dxx} \cdot \text{Dyy} - (0.9 \cdot \text{Dxy})^2$

Max value location = Keypoint

Quickly approximate Hessian matrix with 3 box filters to detect blobs

1 **Generate Integral Image**

Original image → Integral Image (preprocessing)

↓

2 **Fast-Hessian Detector**

Calculate Hessian determinant with box filters → Detect keypoints

↓

Orientation via Haar Wavelet

3 Determine dominant orientation using Haar wavelet responses

↓

4

64D Descriptor

4x4 sub-regions × 4D Haar response = 64-dimensional vector

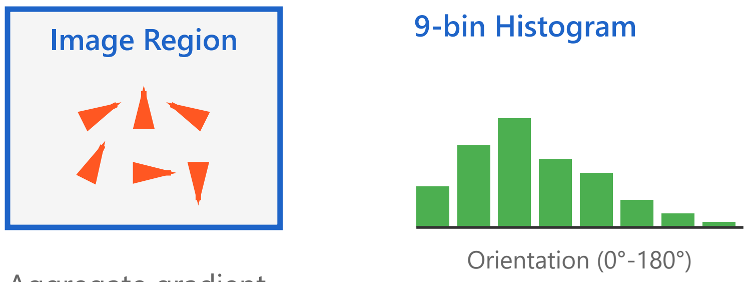
⚡ **Speed Improvement:** 2-3x faster than SIFT, with Integral Image enabling same speed for all scales

📊 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.

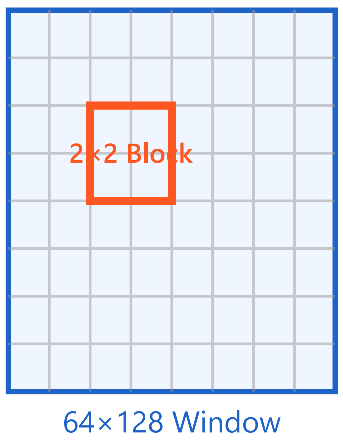
Gradient Computation & Orientation



Aggregate gradient orientations in each cell (8x8) to 9 bins

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

Cell & Block Structure



Structure:

- Cell: 8×8 pixels
- Block: 2×2 cells
- Stride: 8 pixels

Dimension:

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

Group 2×2 cells into blocks and normalize to make robust against illumination changes

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

↓

2 Orientation Binning
Divide into 8×8 cells, generate 9-bin orientation histogram in each cell

↓

3

Block Normalization

Group into 2×2 cell blocks and apply L2 normalization (50% overlap)

↓

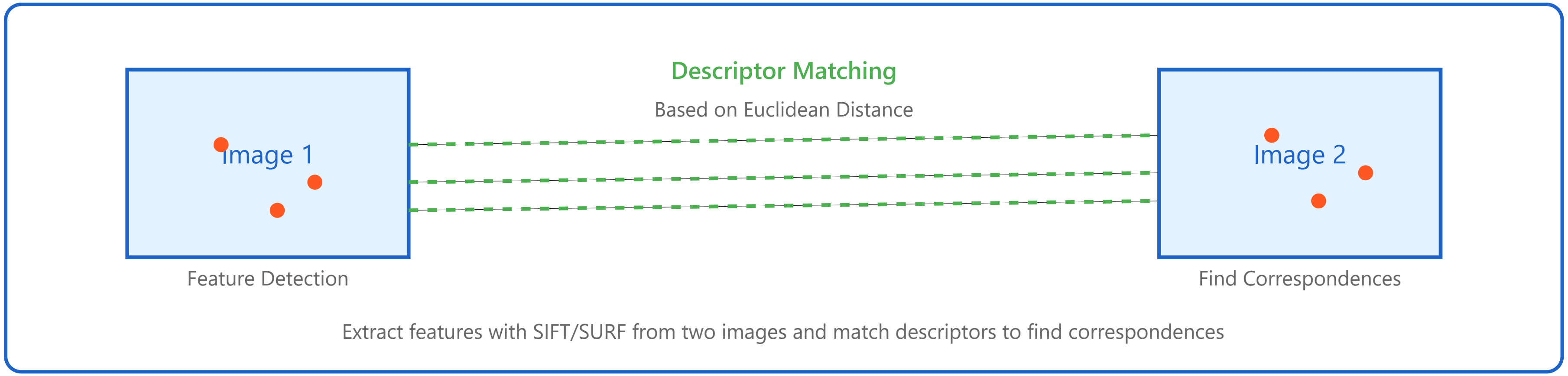
4

Feature Vector

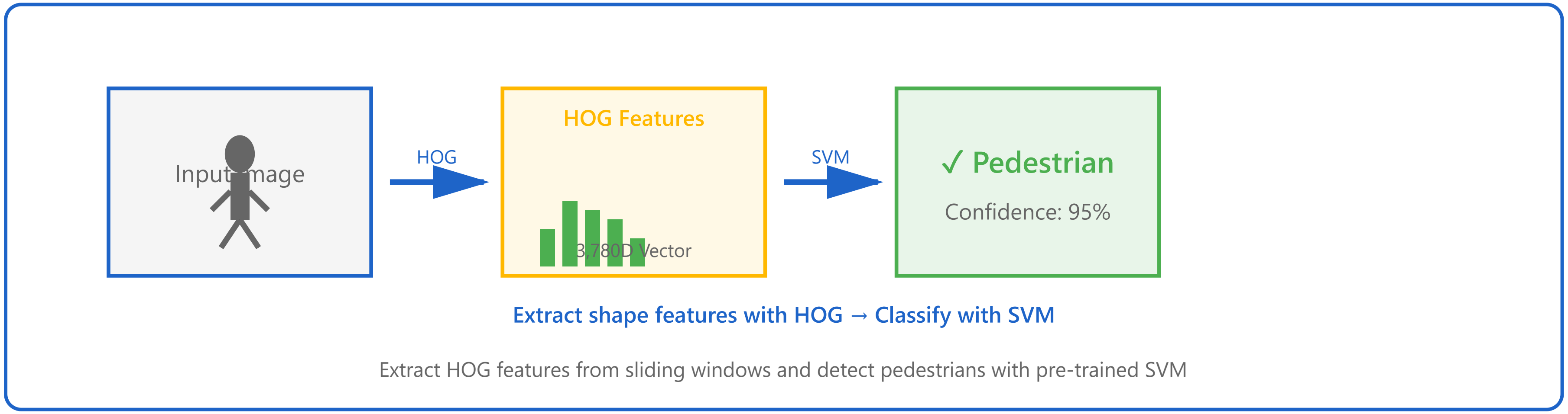
Concatenate all block histograms → 3,780-dimensional descriptor

🎯 Real-world Applications

Image Matching (SIFT/SURF)



Object Detection (HOG + SVM)



📋 Detailed Algorithm Comparison

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

Time Domain Signal → Frequency Domain Components

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

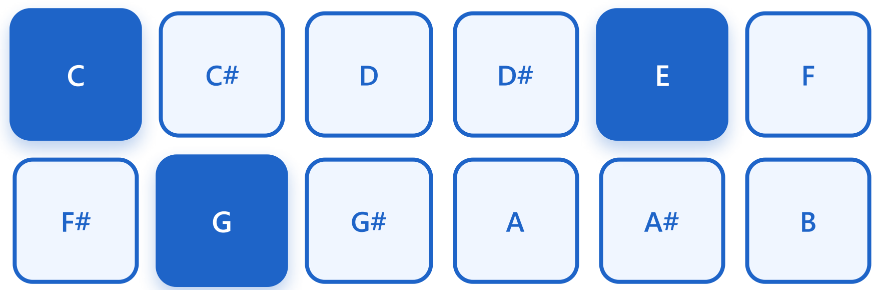
3 Pitch Mapping

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

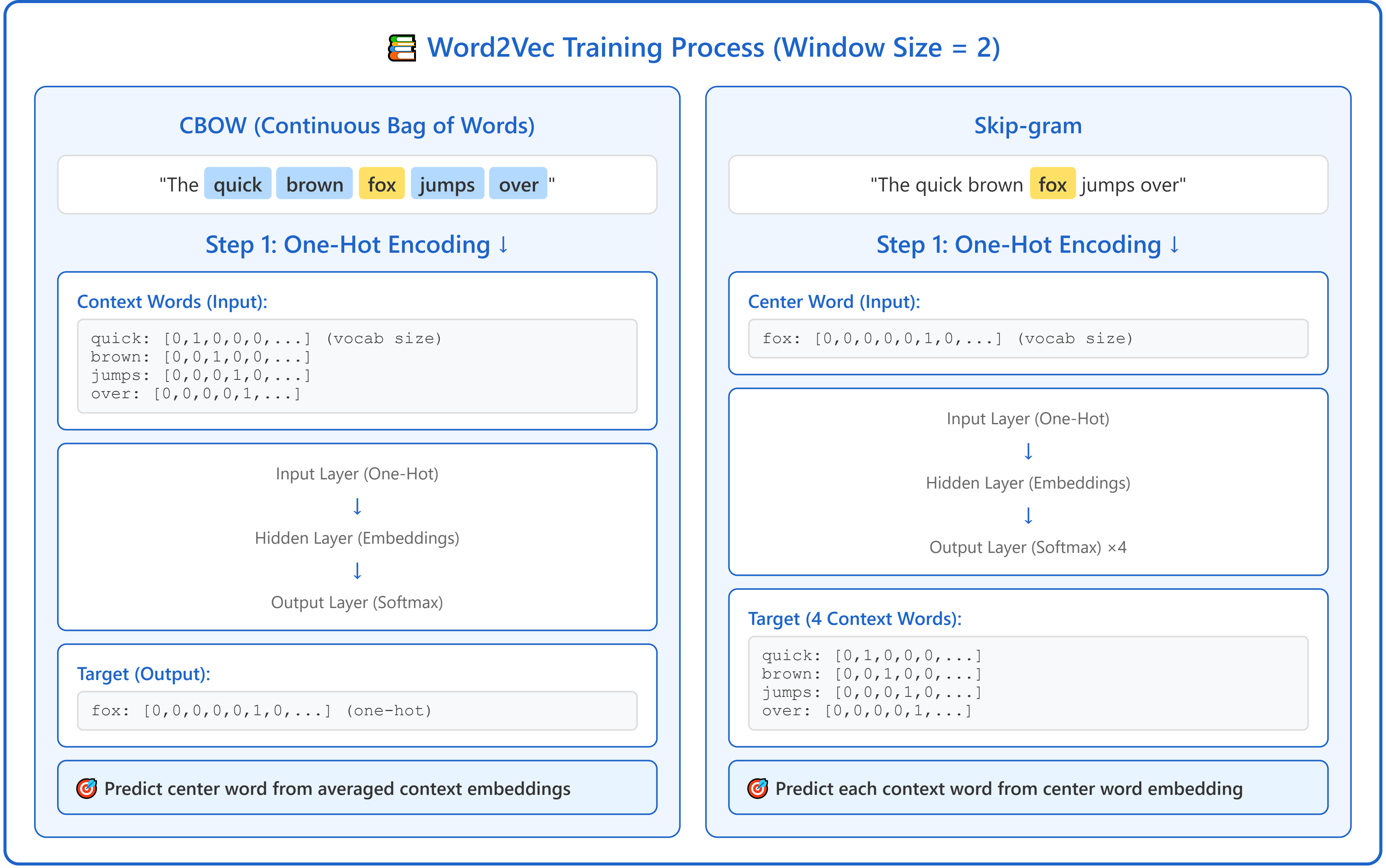
Key Properties

- 🎯 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

Feature Hierarchy Across Layers

Early Layers
Edges
Textures
Colors

→

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

Input Image (H×W×C) → **Conv Layer** Feature Extraction → **Activation** ReLU, etc. → **Pooling** Down-sampling → **...** Repeat → **FC Layer** Classification → **Output** Prediction

Convolutional Layers: Extract local features from images

Activation Functions: Add non-linearity (mainly ReLU)

Pooling Layers: Reduce spatial dimensions and achieve invariance

Fully Connected: High-level reasoning and classification

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:** 3×3, 5×5, 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

🔢 Output Size Calculation

Output size = ⌊(Input size - Filter size + 2×Padding) / Stride⌋ + 1

Example: Input 32×32, Filter 5×5, stride 1, padding 2
→ Output: ⌊(32 - 5 + 4) / 1⌋ + 1 = 32 (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

RNN Cell → RNN Cell → RNN Cell → RNN Cell

🧠 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

Autoencoder Architecture

Input
Original Data

→

Encoder
Maps to latent space

→

Latent Space
Low-dimensional representation

→

Decoder
Reconstructs input

→

Output
Reconstructed Data

Bottleneck forces learning of essential features

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: $x \to z = f\_enc(x; \theta\_enc)$

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

Decoder: $z \rightarrow \hat{x} = f\_dec(z; \theta\_dec)$
• 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 → 2-dimensional latent space

```
Input vector: x ∈ ℝ^784 (28×28 image flattened)
e.g., x = [0.1, 0.0, 0.8, ..., 0.3]

Encoder operations:
h_1 = ReLU(W_1 x + b_1 ) # W_1 ∈ ℝ^(128×784)
h_2 = ReLU(W_2 h_1 + b_2 ) # W_2 ∈ ℝ^(64×128)
z = W_3 h_2 + b_3 # W_3 ∈ ℝ^(2×64), z ∈ ℝ^2
→ z = [1.2, -0.5] # 2D latent vector

Decoder operations:
h_3 = ReLU(W_4 z + b_4 ) # W_4 ∈ ℝ^(64×2)
h_4 = ReLU(W_5h_3 + b_5) # W_5 ∈ ℝ^(128×64)
x^= σ(W_6h_4 + b_6) # W_6 ∈ ℝ^(784×128)
→ x^= [0.09, 0.02, 0.79, ..., 0.31]

Loss: MSE = (1/784) Σ(x_i - xî)^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

Source Task

Pre-training on large datasets

ImageNet, Wikipedia

Knowledge Transfer →

Target Task

Fine-tuning or feature extraction

Limited data scenarios

- 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

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

powersimmani@gmail.com