Digital Image Processing

Dr. Muhammad Sajjad
R.A: Asad Ullah

Lecture 08

Features Extraction and Description

Overview

7 Handcrafted Features vs CNN comparison

6 CNN Architecture Overview

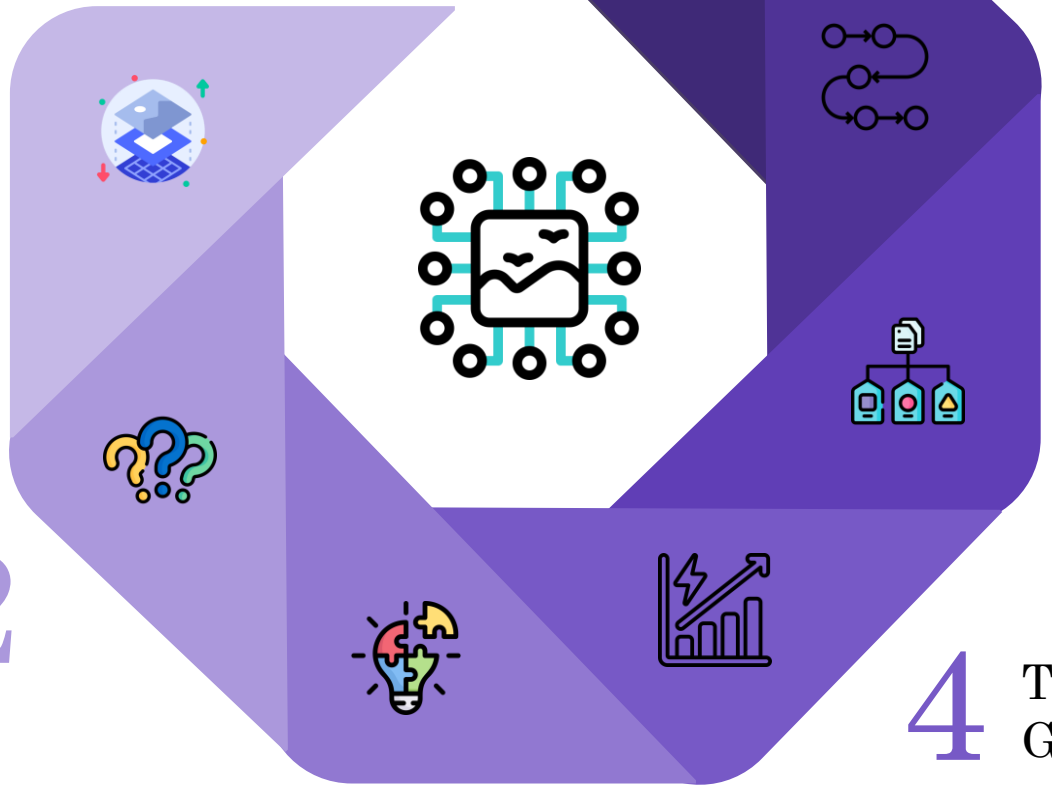
5 Bridging the Semantic Gap (CNN)

4 The Semantic Gap

3 Feature Descriptors Performance Comparison

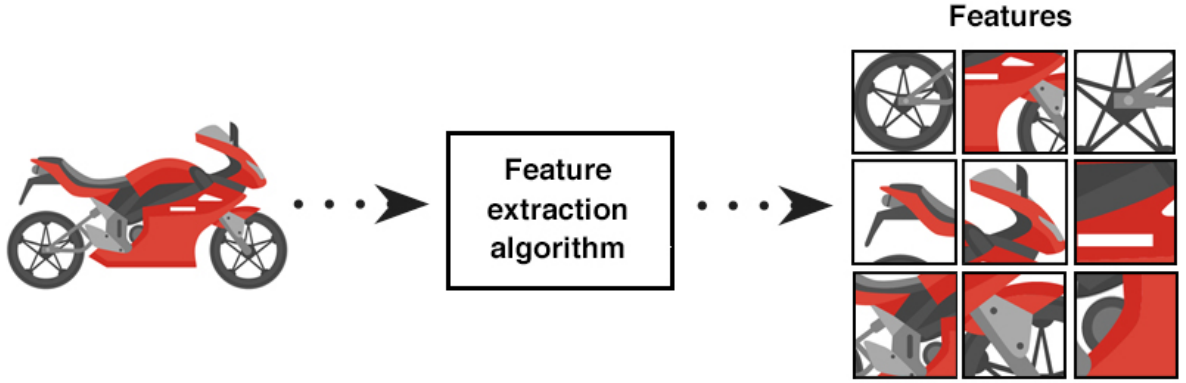
2 Feature Extraction & Feature Descriptors

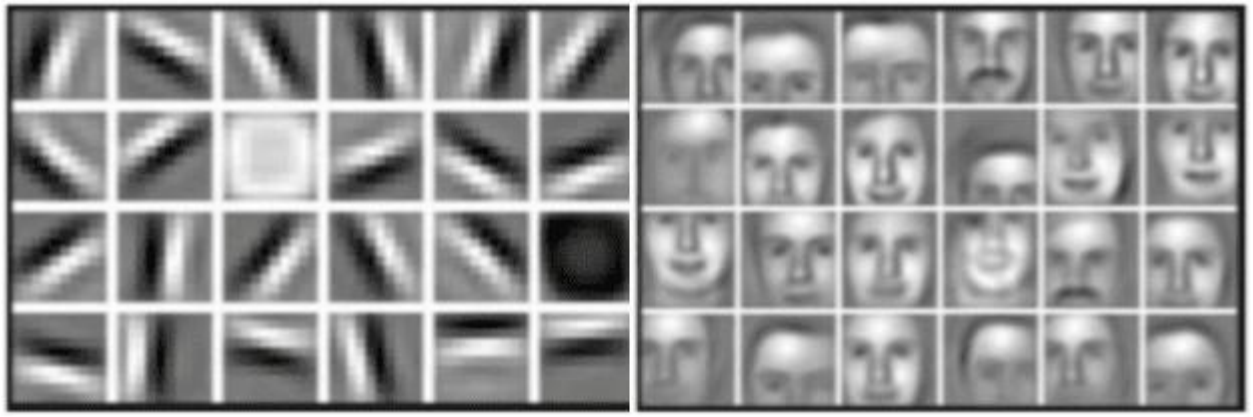
1 What are Features and Why we need them?



What Are Features in Images?

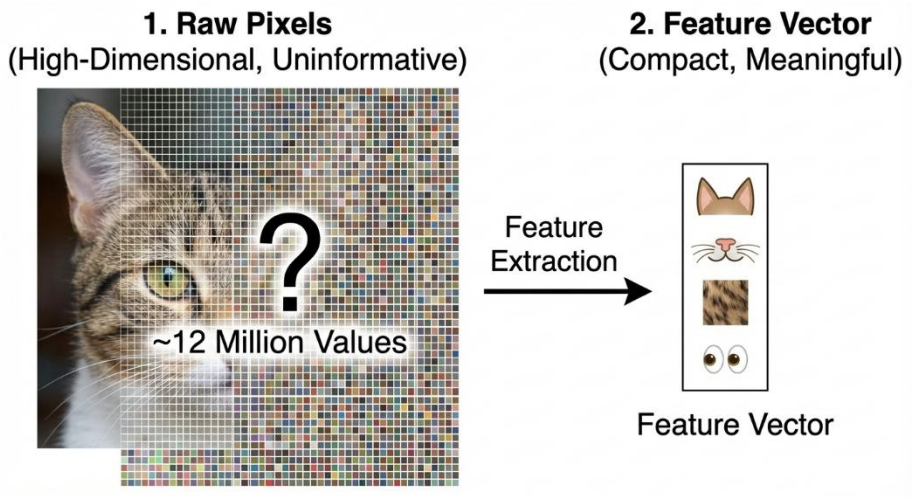
- What makes this image recognizable?
- Features are distinctive patterns or characteristics
- Examples: edges, corners, textures, shapes, colors that help identify objects

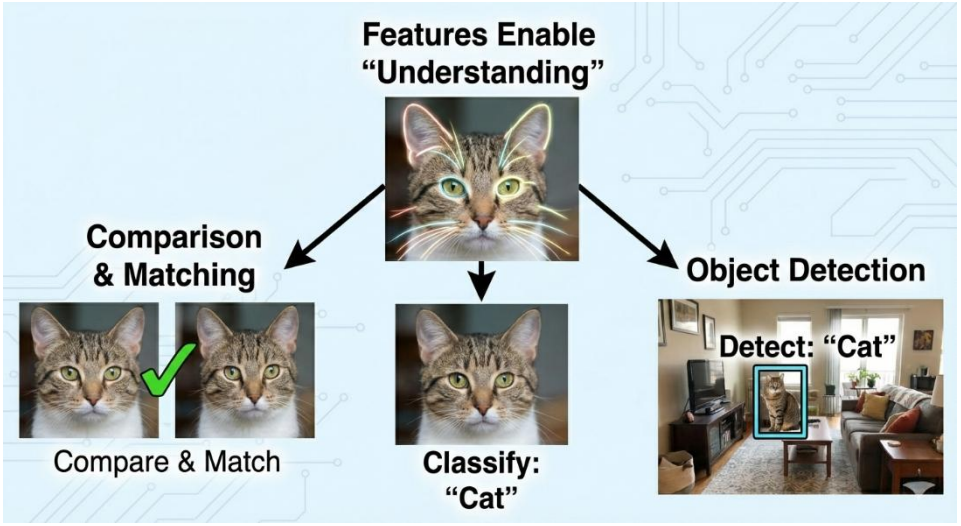




Why Do We Need Features?

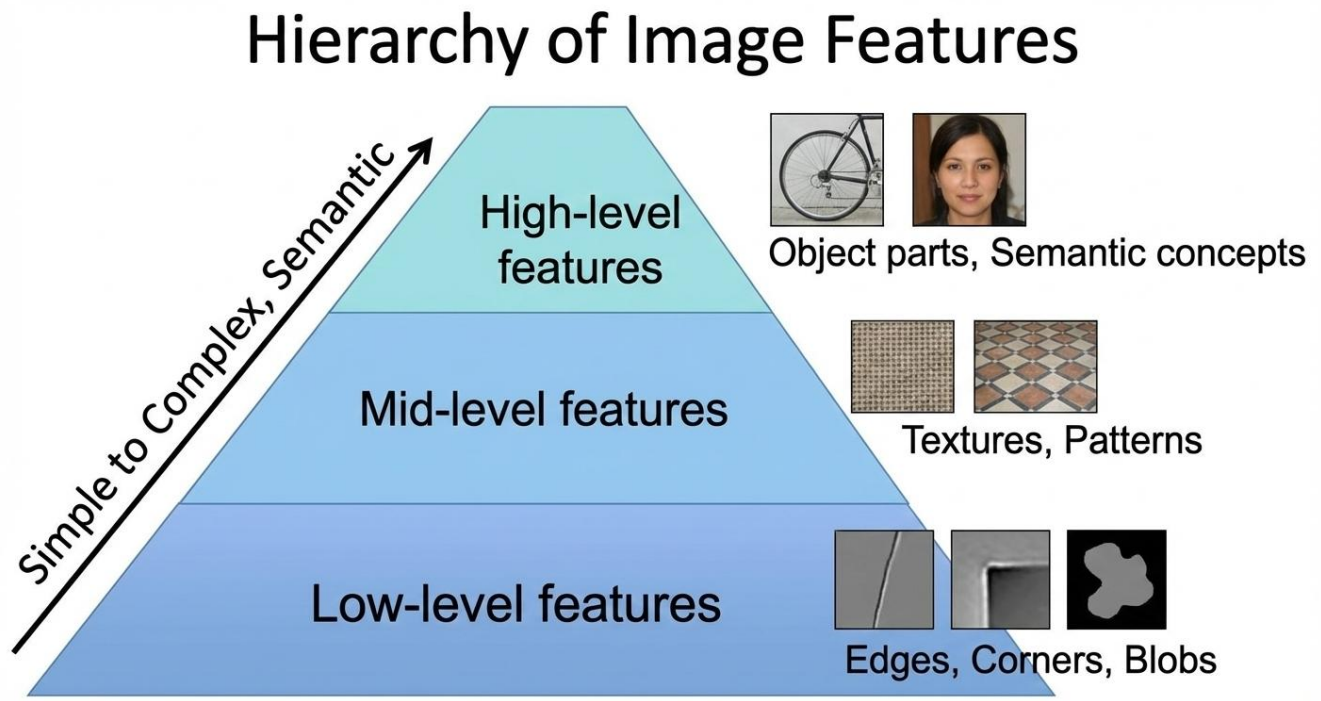
- Raw pixels are too high-dimensional and not informative
- Features provide compact, meaningful representations
- Enable computers to "understand" and compare images
- Foundation for image classification, object detection, matching





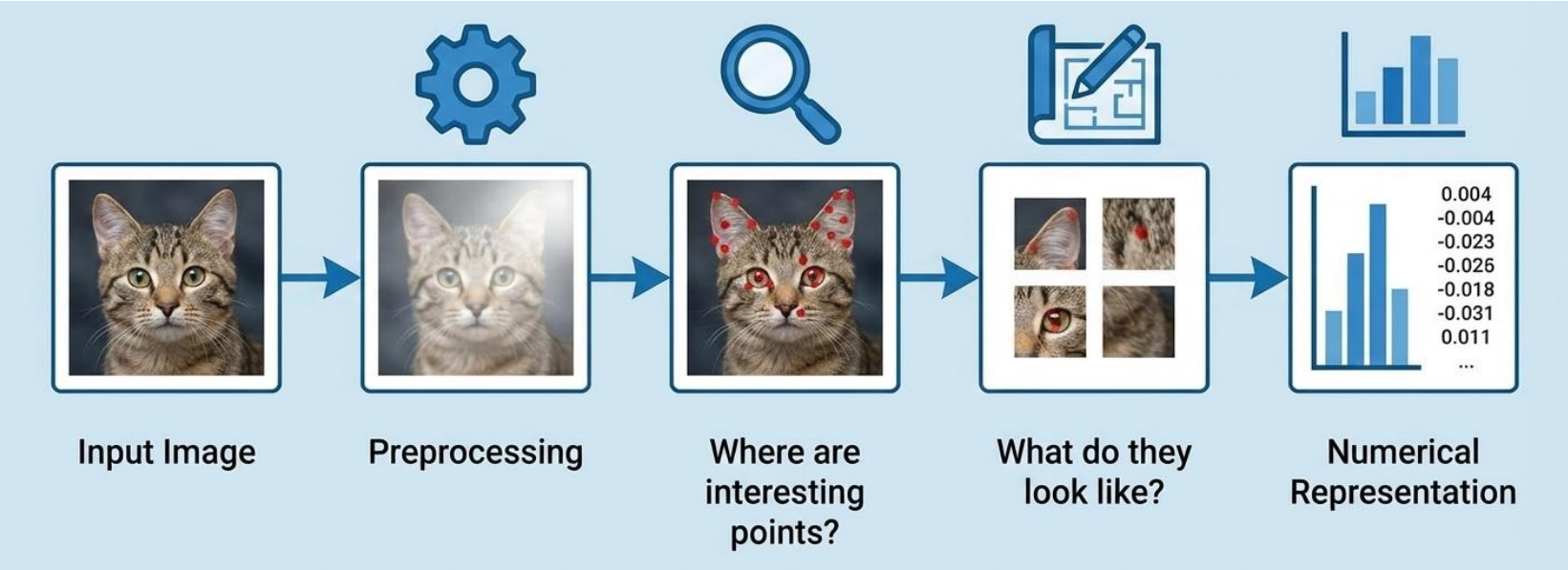
Types of Features

- **Low-level features:** edges, corners, blobs
- **Mid-level features:** textures, patterns
- **High-level features:** object parts, semantic concepts



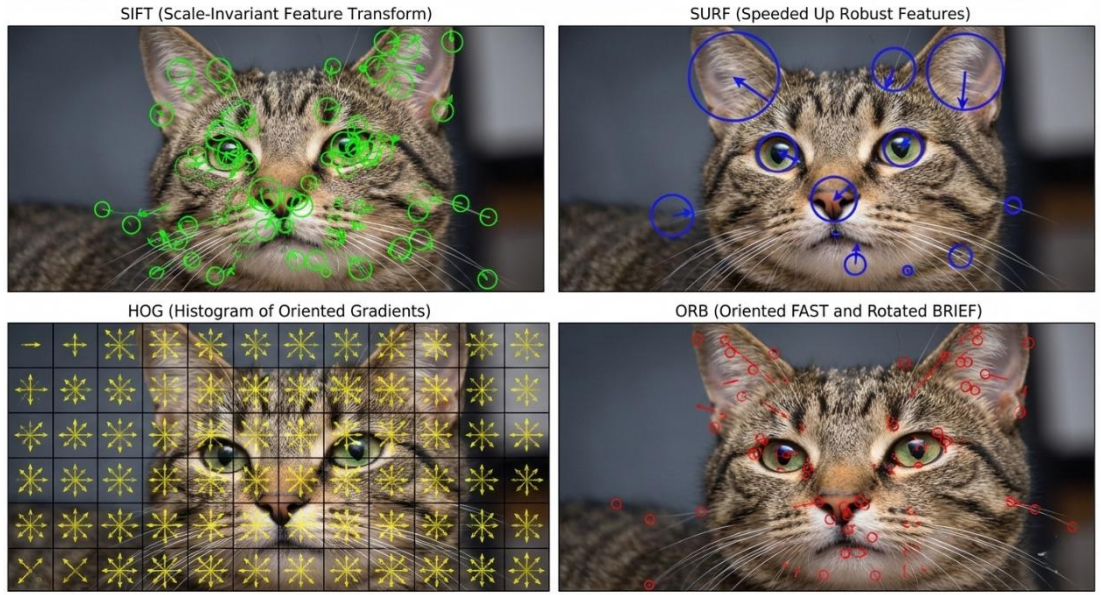
Feature Extraction Pipeline

1. Input Image
2. Preprocessing
3. Feature Detection (where are interesting points?)
4. Feature Description (what do they look like?)
5. Feature Vector (numerical representation)



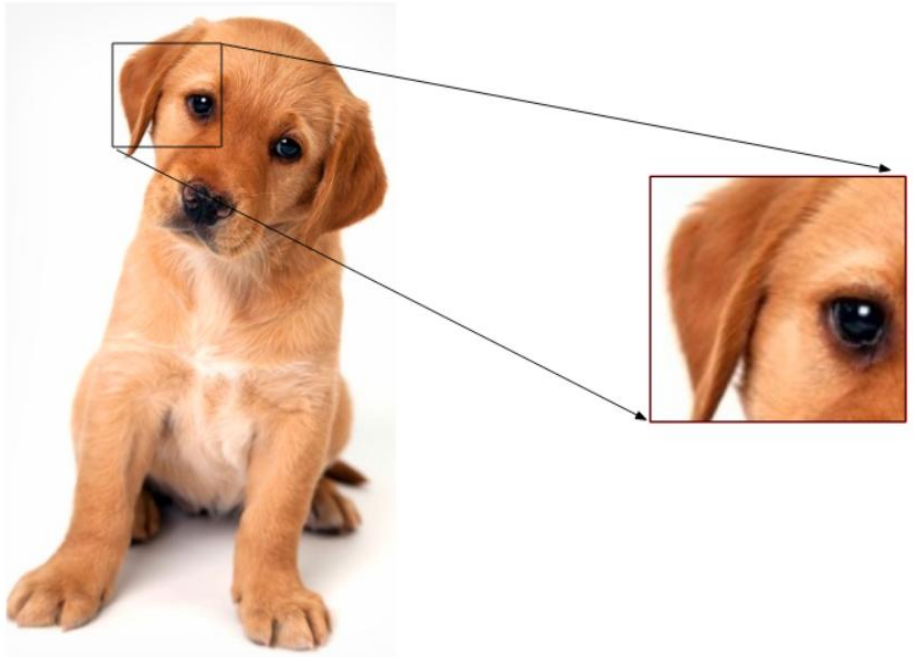
Traditional Feature Descriptors - Overview

- Hand-crafted descriptors designed by experts
- Based on mathematical transformations and statistical analysis
- Types we'll explore:
  - SIFT (Scale-Invariant Feature Transform)
  - SURF (Speeded Up Robust Features)
  - HOG (Histogram of Oriented Gradients)
  - ORB (Oriented FAST and Rotated BRIEF)



Histogram of Oriented Gradient (HOG)

- Counts occurrences of gradient orientation in the localized portion of an image.
- HOG method involves computing the gradient magnitude and orientation for and then dividing the image into small cells
- generate a **Histogram** for each of these regions separately. The histograms are created using the gradients and orientations of the pixel values each pixel in an image.



121	10	78	96	125
48	152	68	125	111
145	78	85	89	65
154	214	56	200	66
214	87	45	102	45

Gradient Calculation in HOG Cont.

- Determine the gradient (or change) in the x-direction.

Change in X direction(G_x) = 89 – 78 = 11

- Calculate the gradient in the

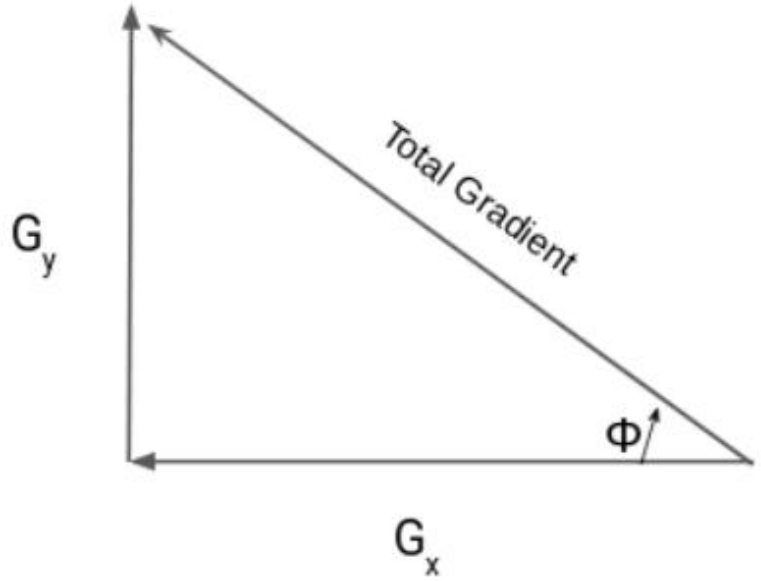
y-direction.

Change in Y direction(G_y) = 68 – 56 = 8

121	10	78	96	125
48	152	68	125	111
145	78	85	89	65
154	214	56	200	66
214	87	45	102	45

Calculate Magnitude and Orientation

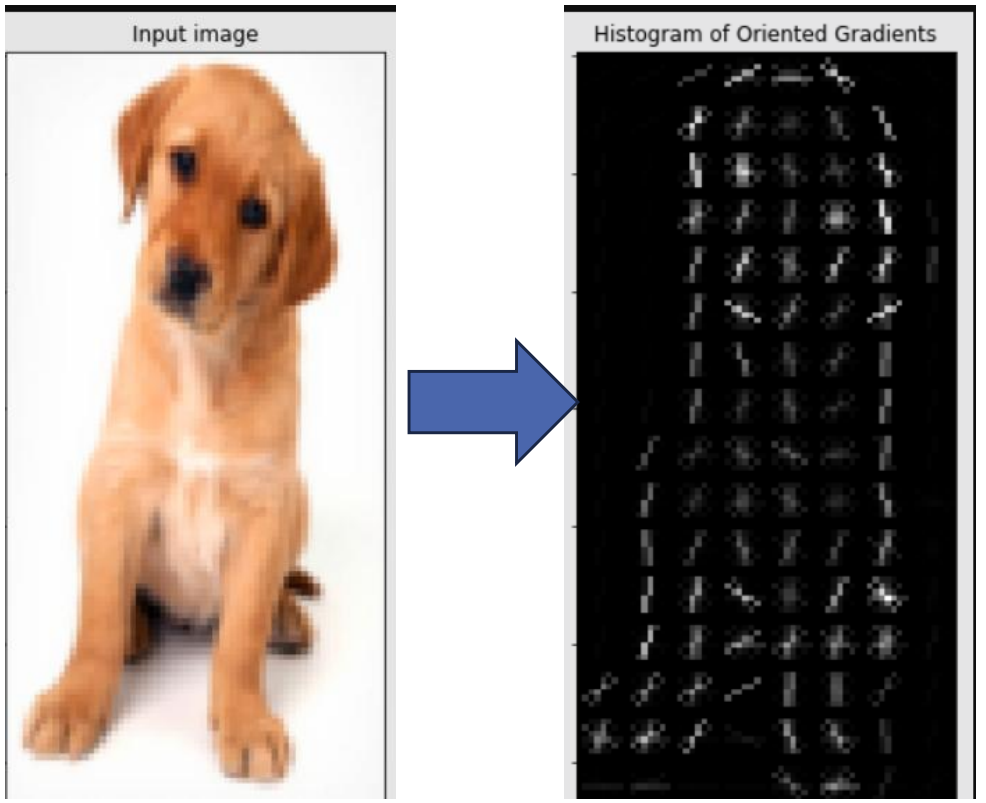
- Using the calculated gradients, now determine the magnitude and direction for each pixel value.
- Total Gradient Magnitude = $\sqrt{(G_x)^2 + (G_y)^2}$
- Total Gradient Magnitude = $\sqrt{11^2 + 8^2} = 13.6$



Create Histogram Using Gradient and Orientation

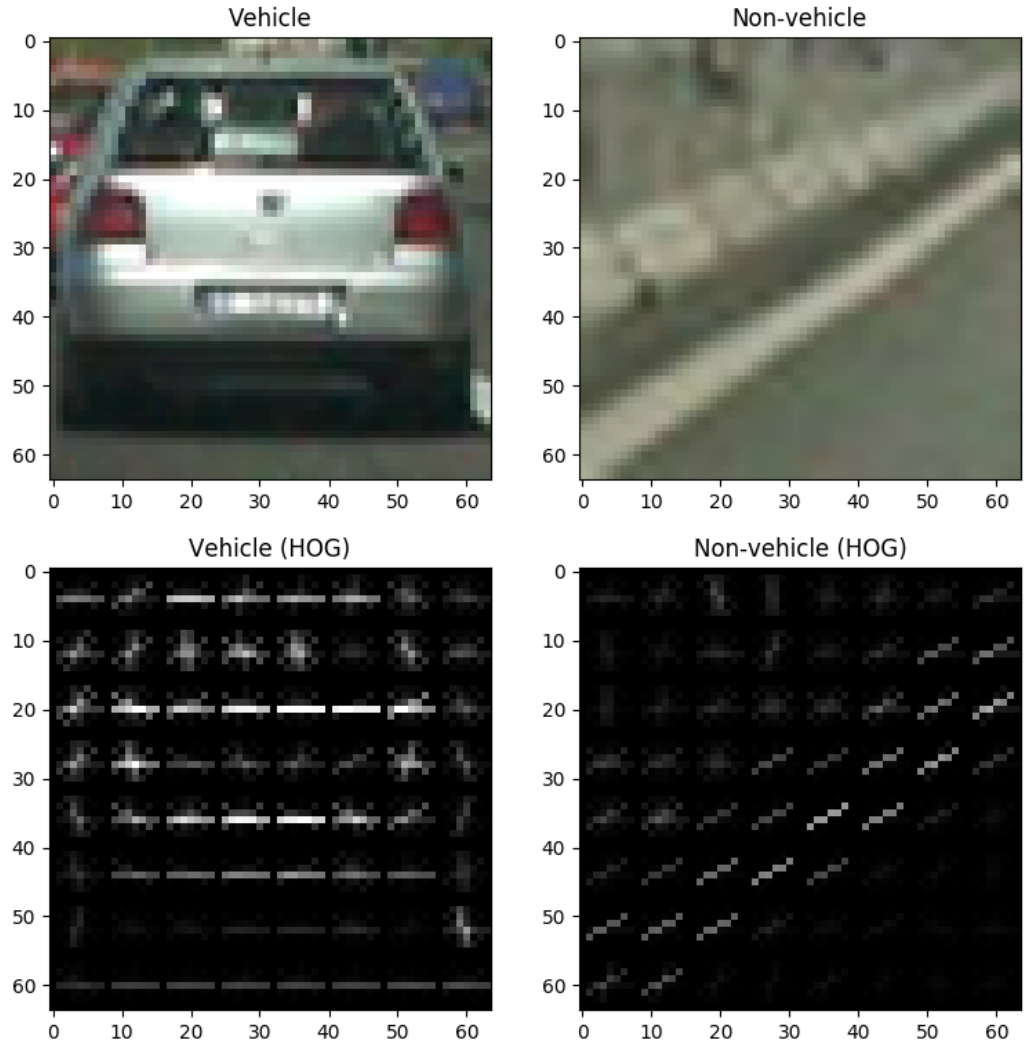
- calculate the orientation (or direction) for the same pixel.
- $tan\theta = \frac{\mathrm{Gy}}{\mathrm{Gx}}$
- $\theta = \arctan\left(\frac{\mathrm{Gy}}{\mathrm{Gx}}\right)$

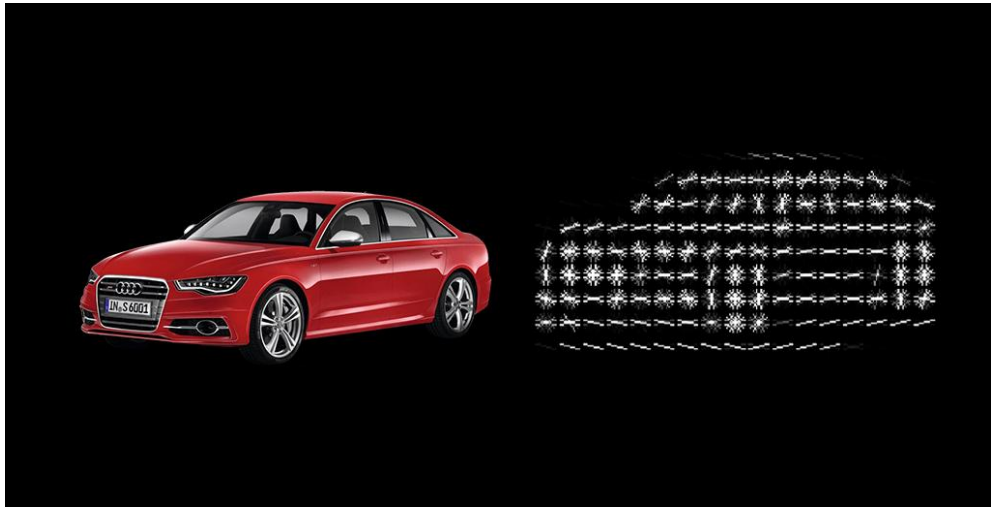
121	10	78	96	125
48	152	68	125	111
145	78	85	89	65
154	214	56	200	66
214	87	45	102	45



Frequency									
Angle	0	1	2	.	.	..	..	179	180

Some Example Of HOG Descriptor





Local Binary Pattern (LBP)

- component of an image is pixel.

- Local Binary Pattern (LBP) is a texture descriptor used in image analysis and computer vision.

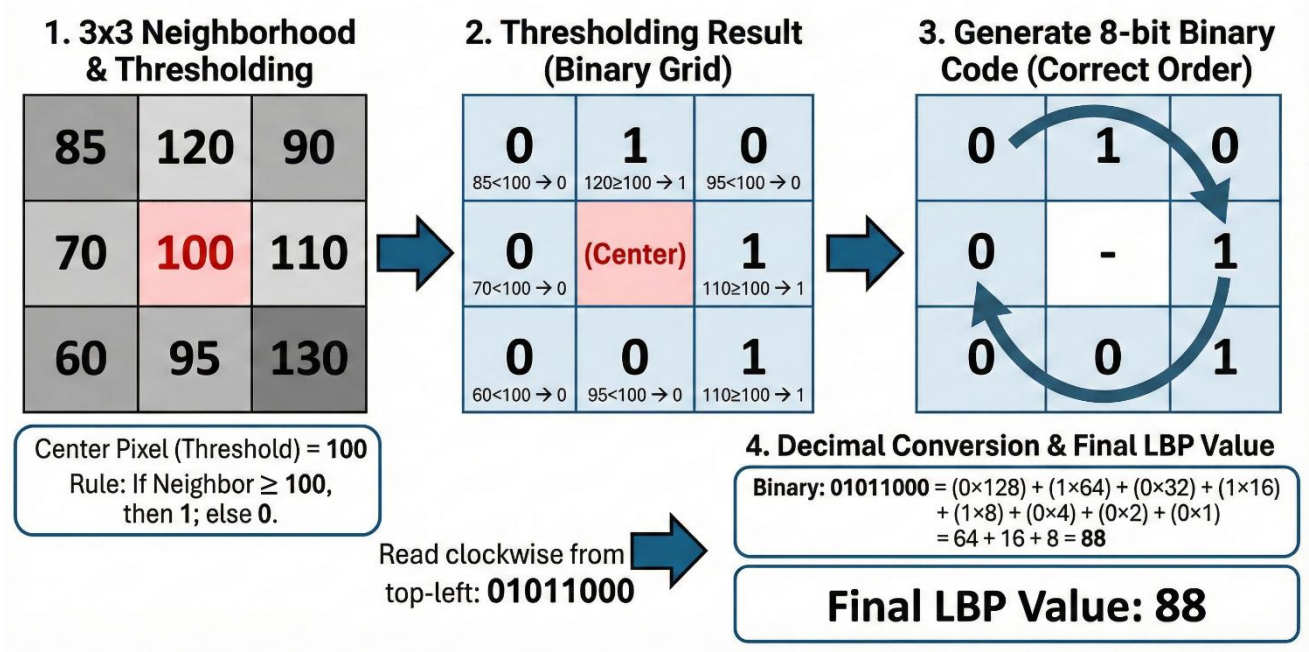
- LBP operates on grayscale images, capturing local patterns by comparing each pixel with its surrounding neighbors.

12	15	18
5	8	3
8	1	2

1	1	1
0	8	0
1	0	0

Local Binary Pattern (LBP) Cont.

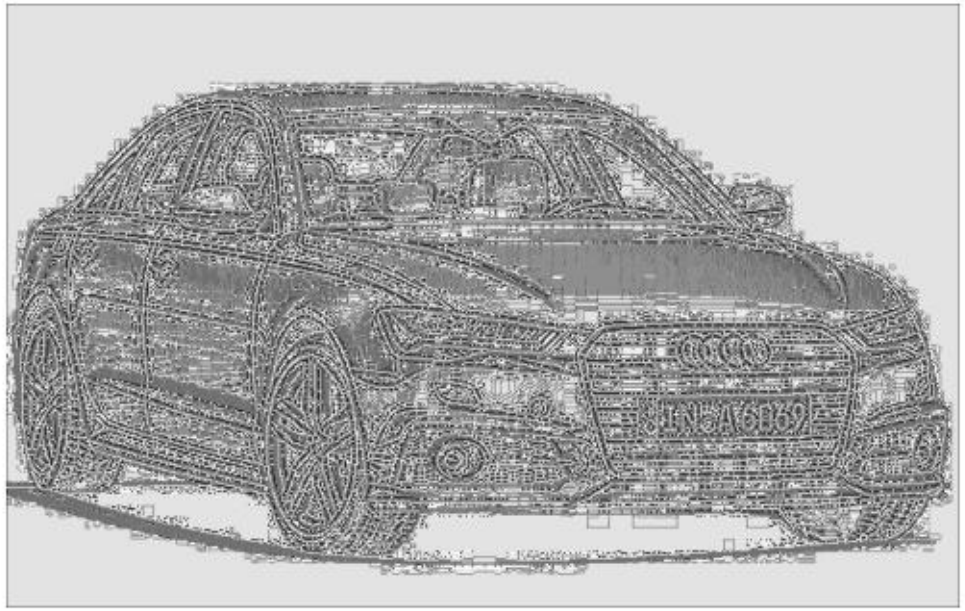
- For each pixel, a binary code is generated by thresholding the intensity values of its neighbors. The center pixel value is used as a threshold.
- Neighbors with values greater than or equal to the center pixel contribute a '1' to the binary code; otherwise, a '0' is added.
- This process results in an 8-bit binary code for each pixel, representing the local texture pattern.



Local Binary Pattern Features

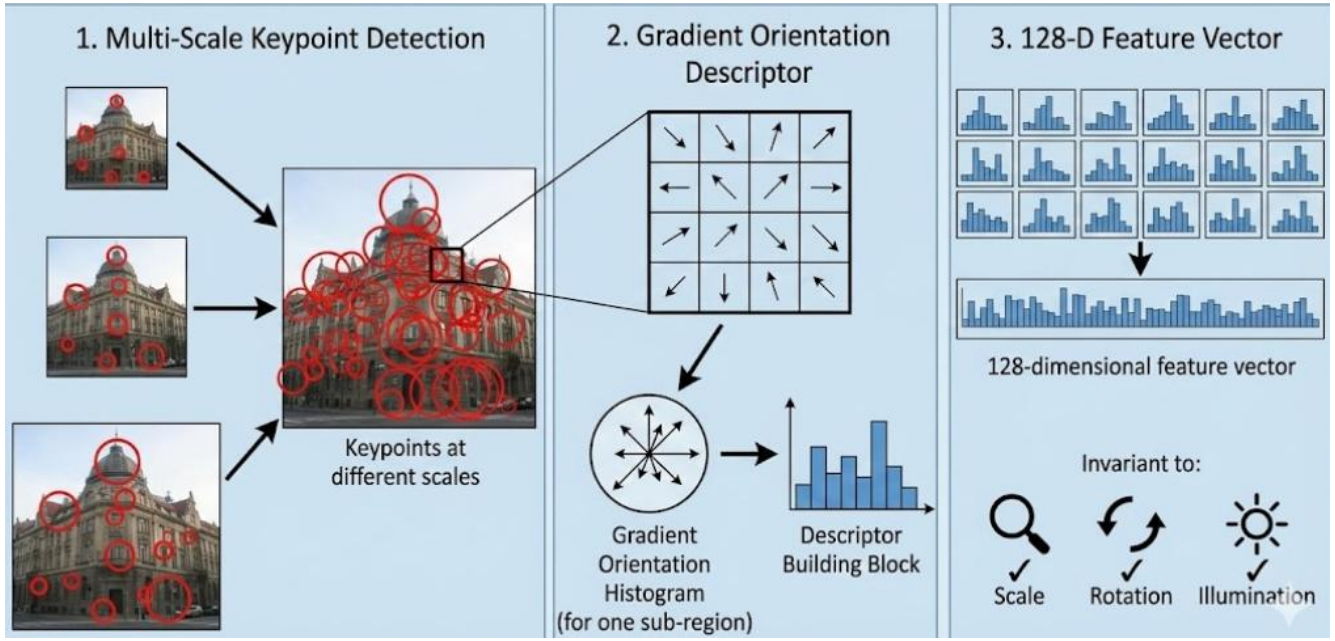
InputImage

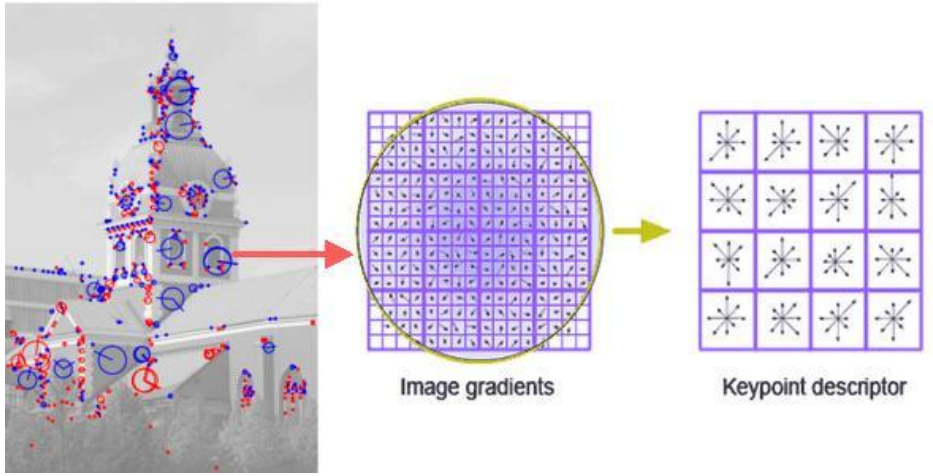
LBPFeatures



SIFT - Scale-Invariant Feature Transform

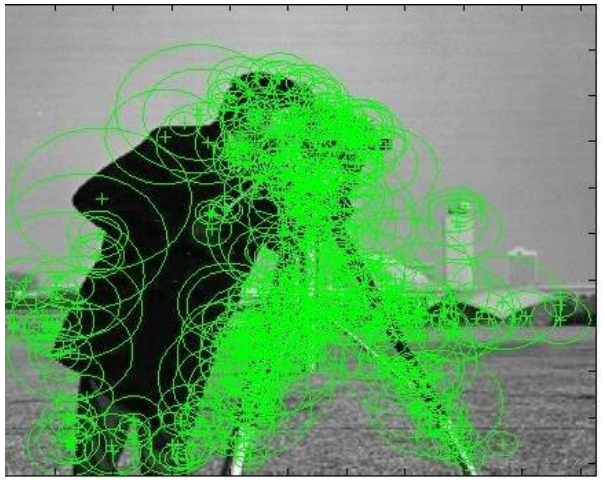
- Detects keypoints at different scales
- Builds descriptors based on gradient orientations
- Invariant to scale, rotation, illumination
- 128-dimensional feature vector per keypoint

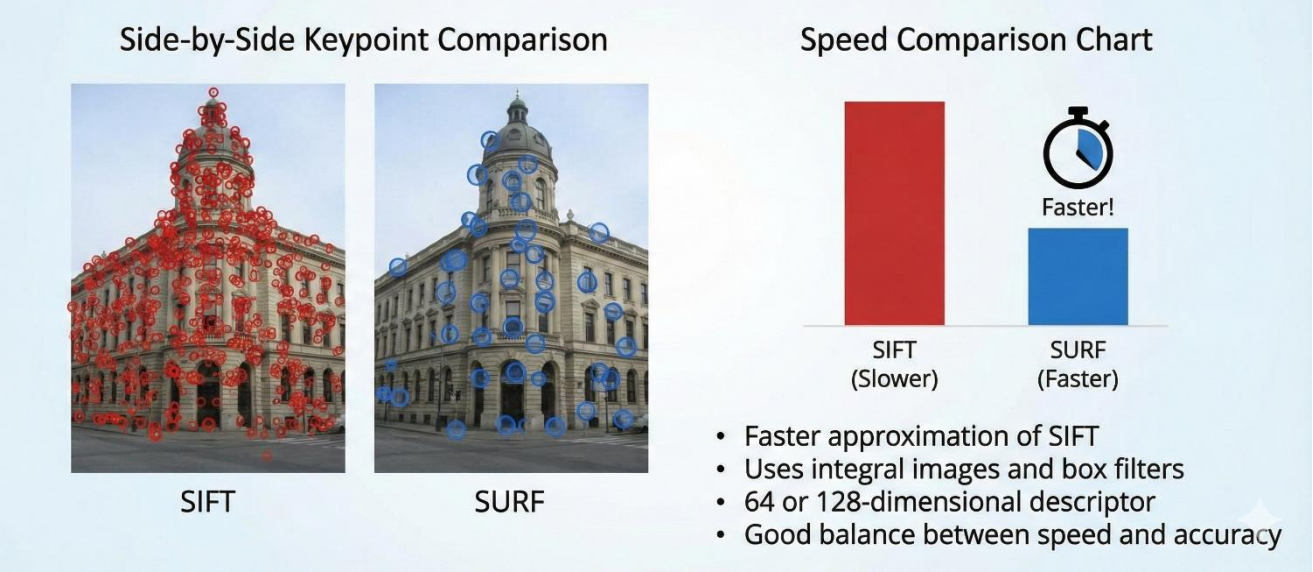


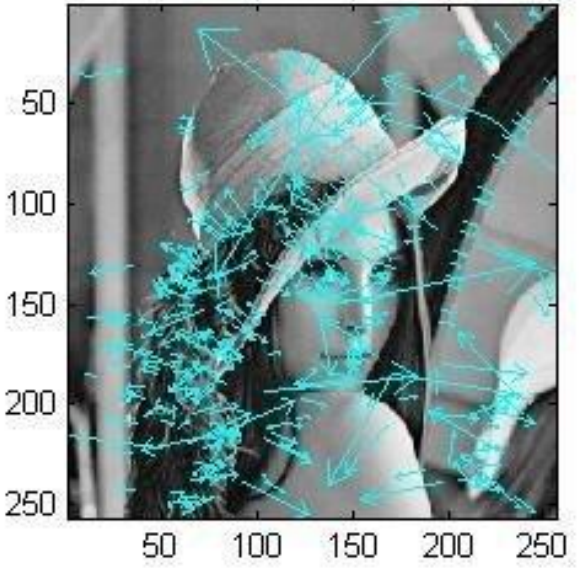


SURF - Speeded Up Robust Features

- Detects Faster approximation of SIFT
- Uses integral images and box filters
- 64 or 128-dimensional descriptor
- Good balance between speed and accuracy

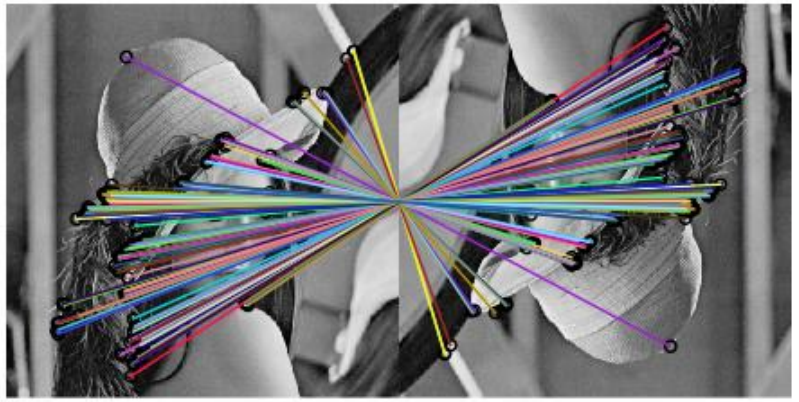


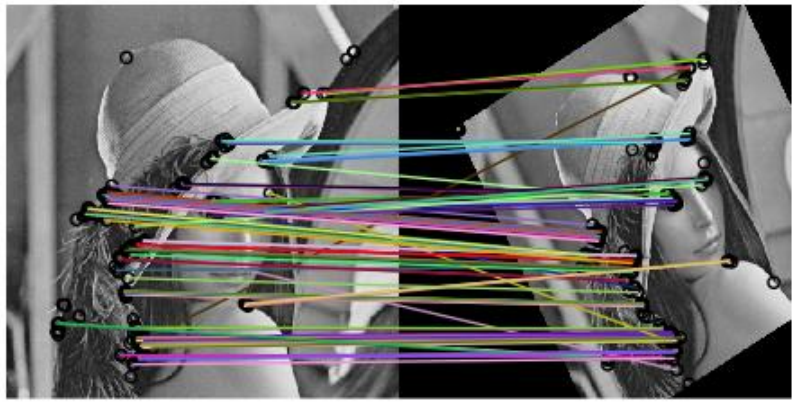


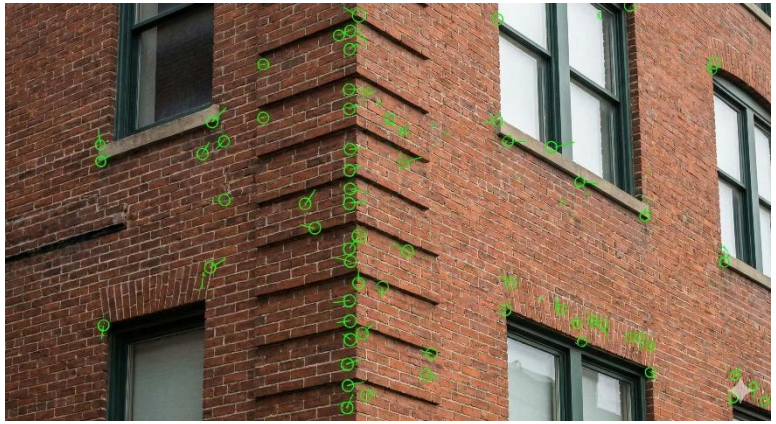


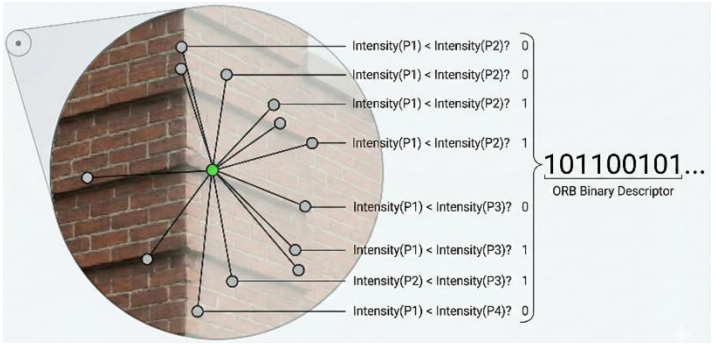
Oriented FAST and Rotated BRIEF

- Fast and efficient (good for real-time applications)
- Combines FAST keypoint detector and BRIEF descriptor
- Rotation invariant
- Binary descriptor (fast matching)

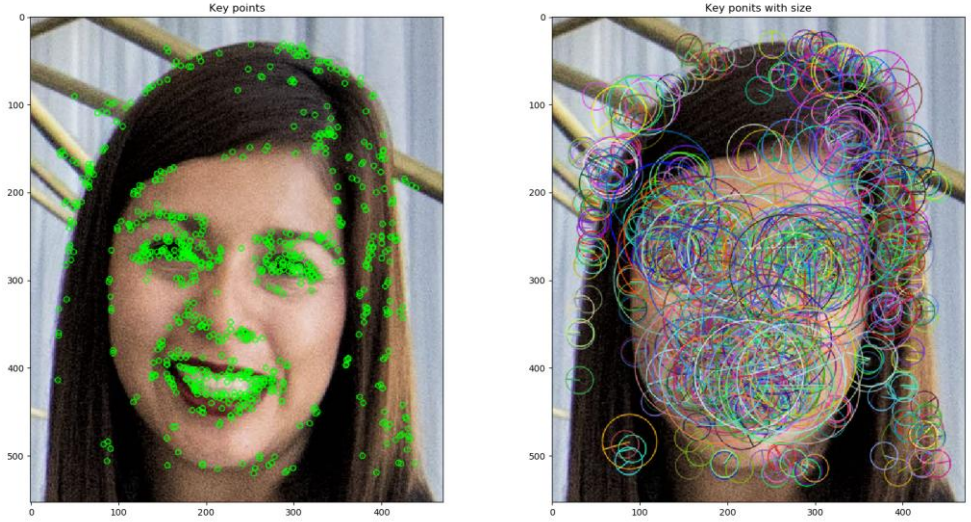






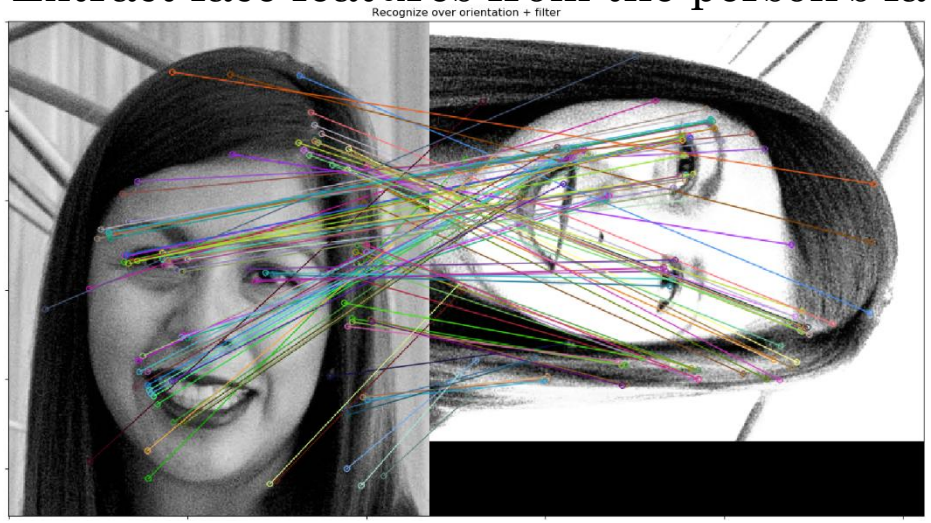


Face Identification using ORB

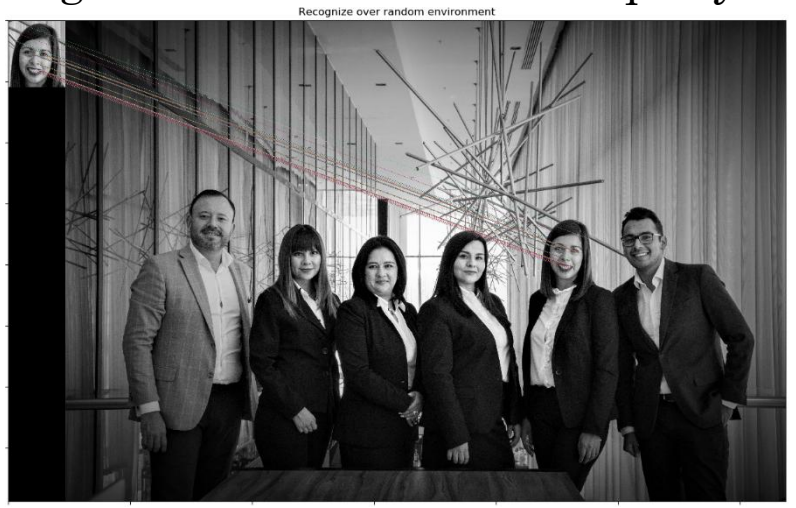


Extract face features from the person's face

Matching the face feature with a query version



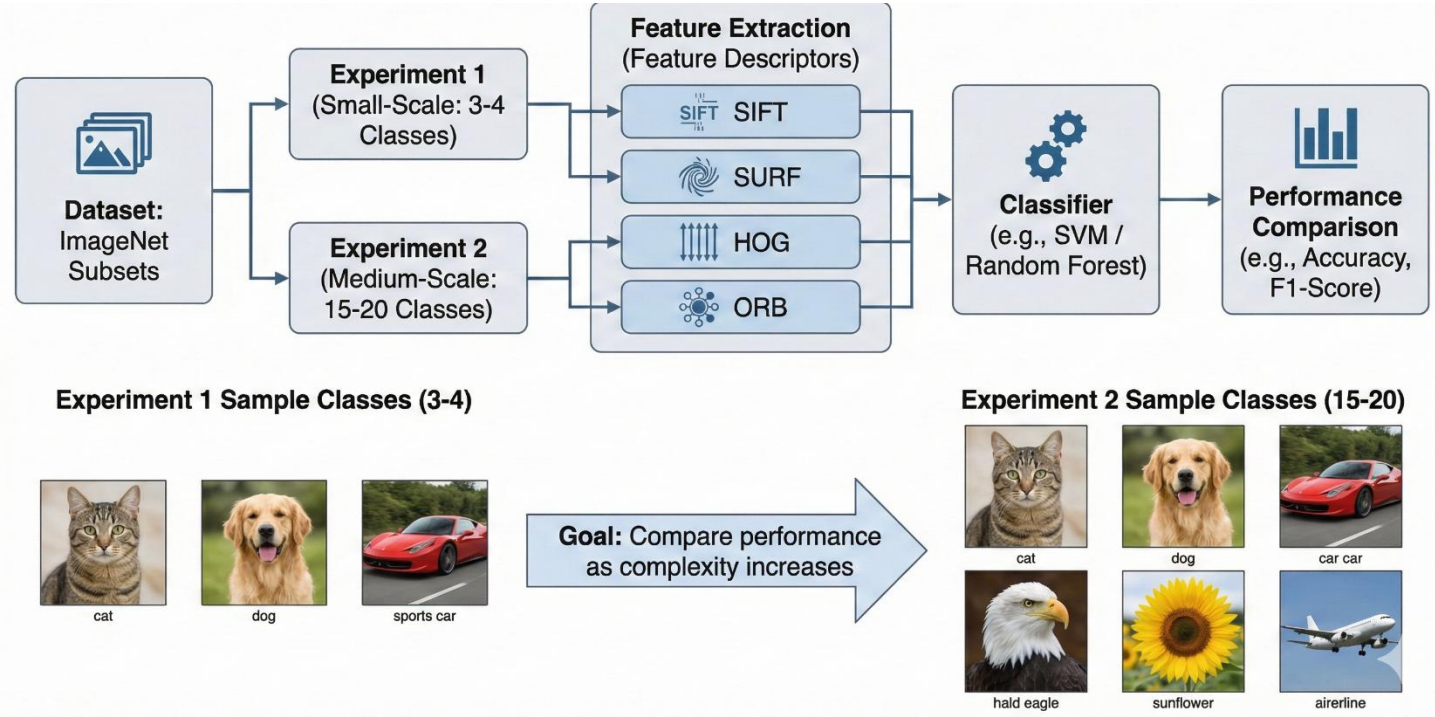
Recognize over filtered and rotated query version



Recognize over full environment

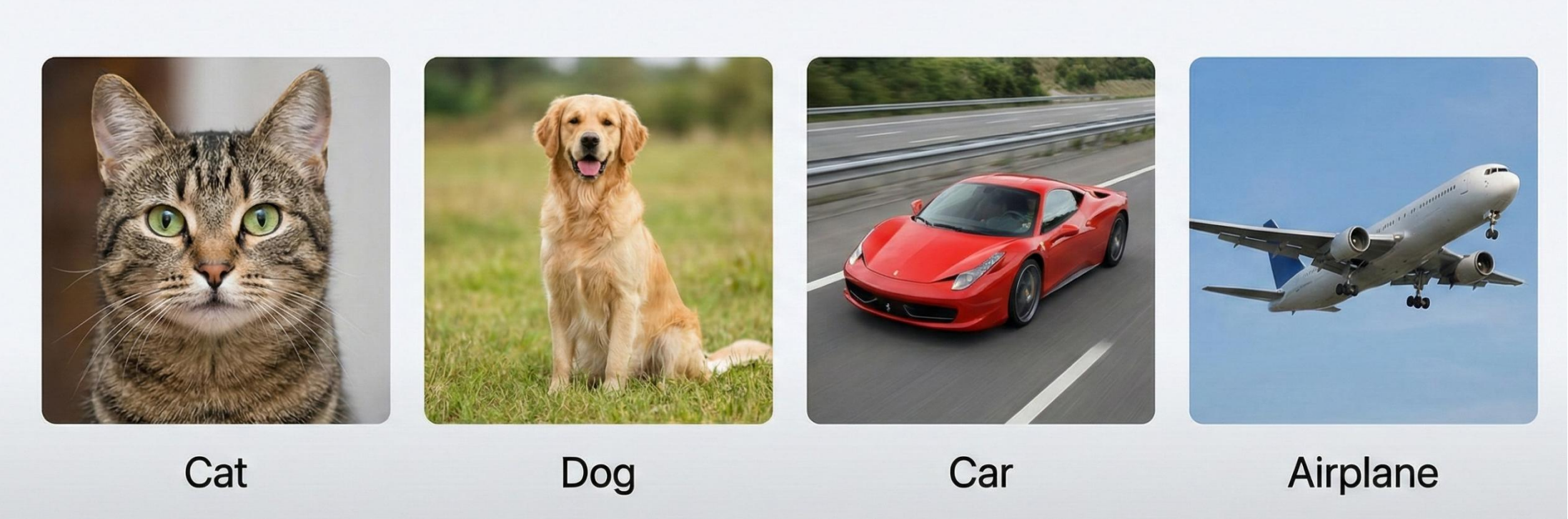
Our Experimental Setup

- **Dataset:** ImageNet subsets
- **Experiment 1:** 3-4 classes (small-scale)
- **Experiment 2:** 15-20 classes (medium-scale)
- **Feature Descriptors:** SIFT, SURF, HOG, ORB
- **Classifier:** SVM (or Random Forest)
- **Goal:** Compare performance as complexity increases



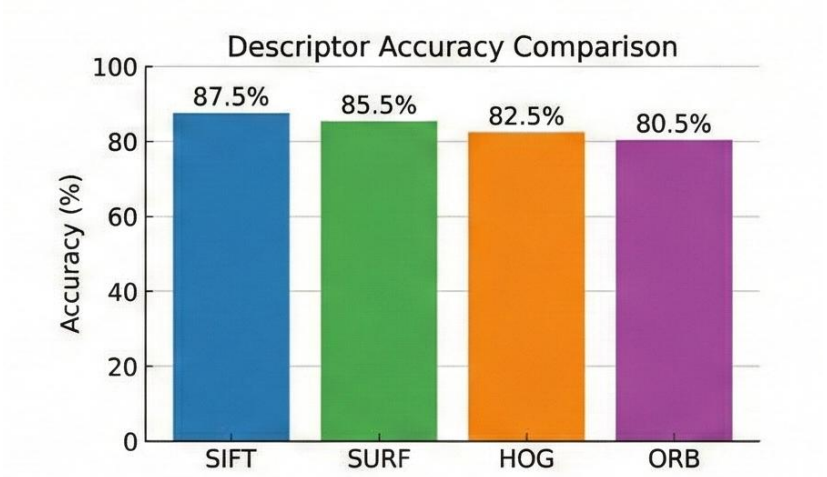
Experiment 1 - Small Scale (3-4 Classes)

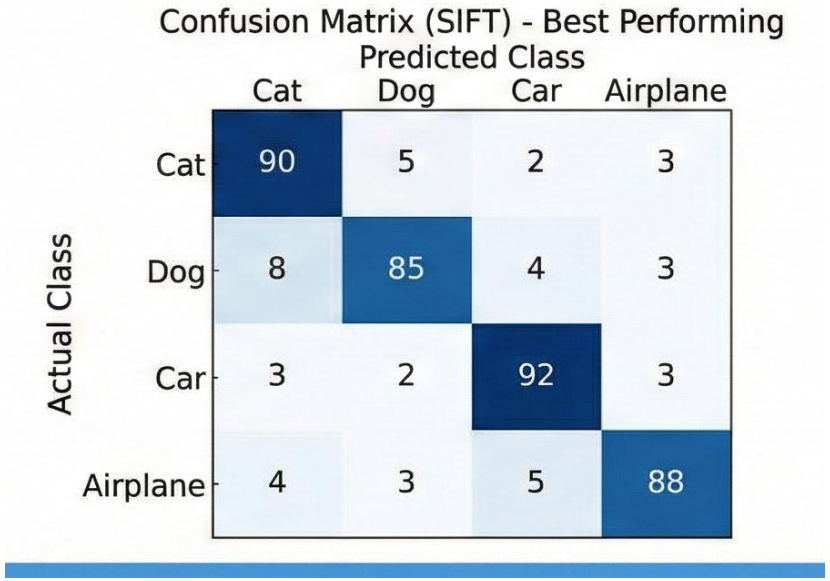
- Selected classes: Cat, Dog, Car, Airplane
- Relatively distinct visual characteristics
- Expected: Good performance with traditional descriptors



Results - Small Scale Performance

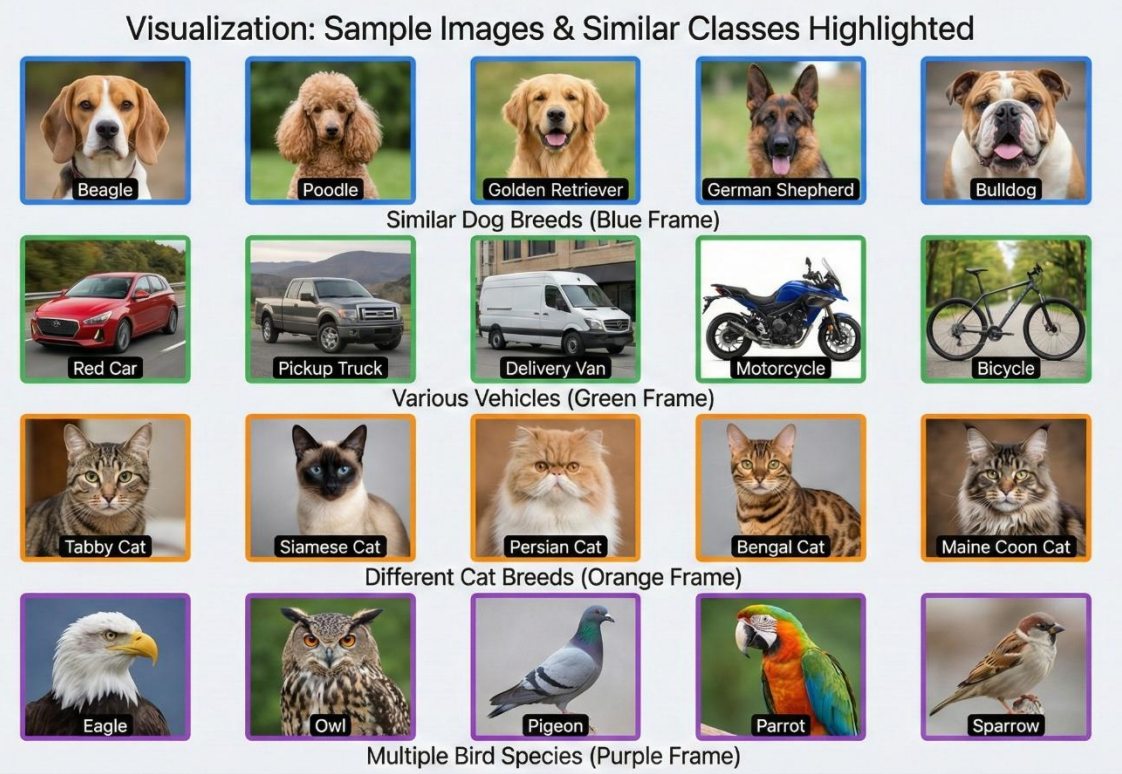
- Performance metrics table:
  - SIFT: ~85-90% accuracy
  - SURF: ~83-88% accuracy
  - HOG: ~80-85% accuracy
  - ORB: ~78-83% accuracy
- Traditional descriptors work reasonably well
- Classes are visually distinct





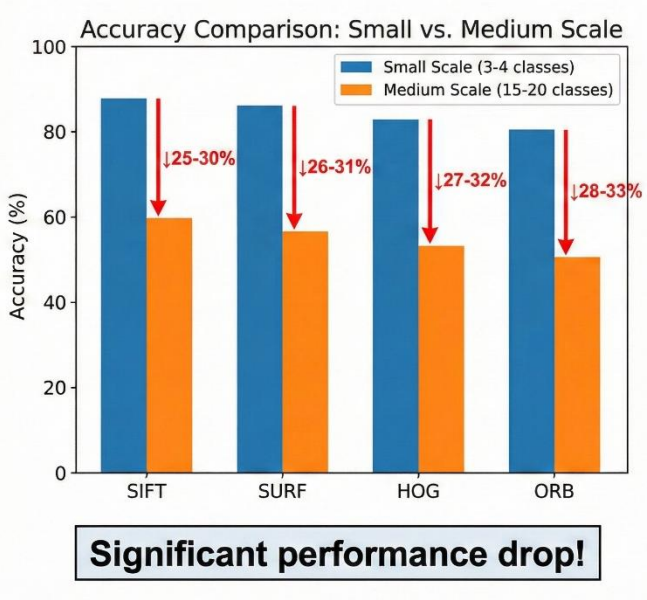
Experiment 2 - Medium Scale (15-20 Classes)

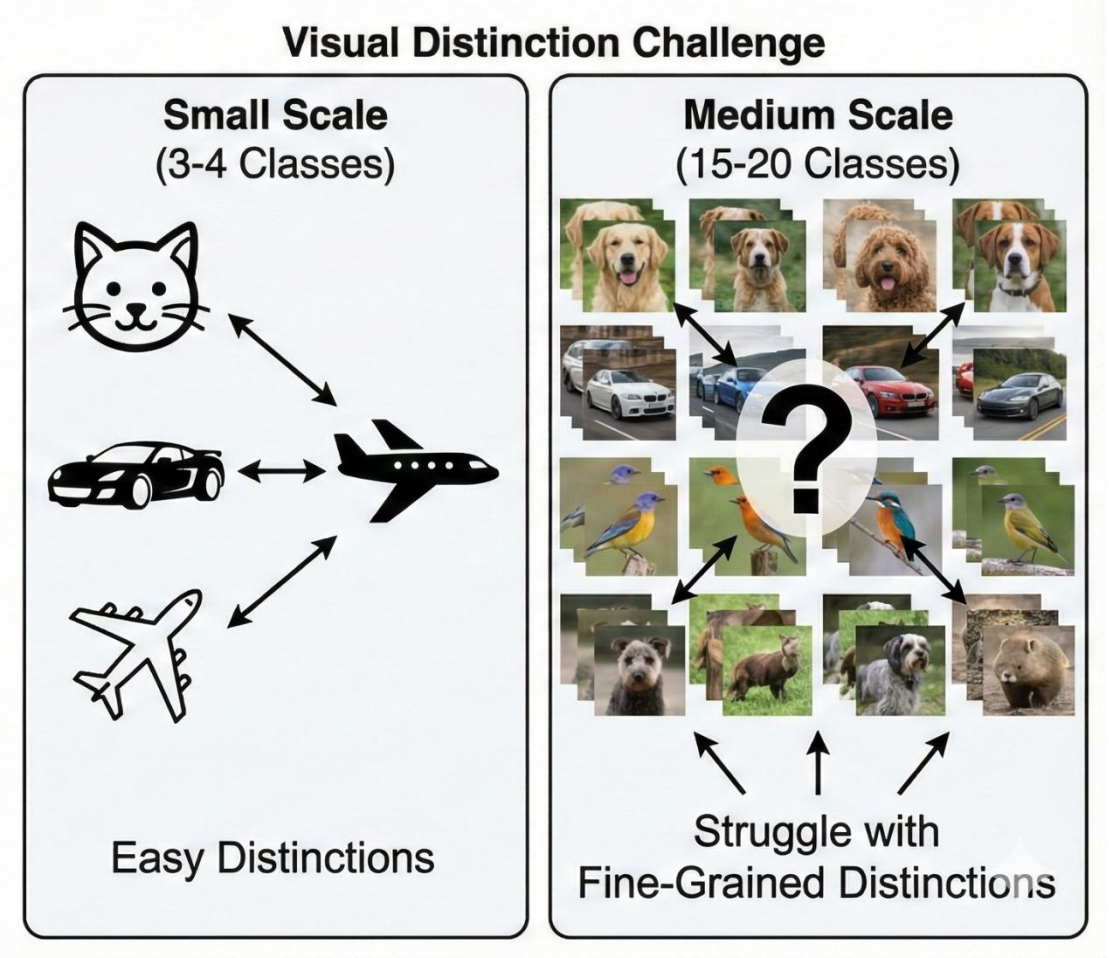
- Expanded to 15-20 classes
- Include visually similar classes
- Examples: different dog breeds, various vehicles, multiple animals



Results - Medium Scale Performance

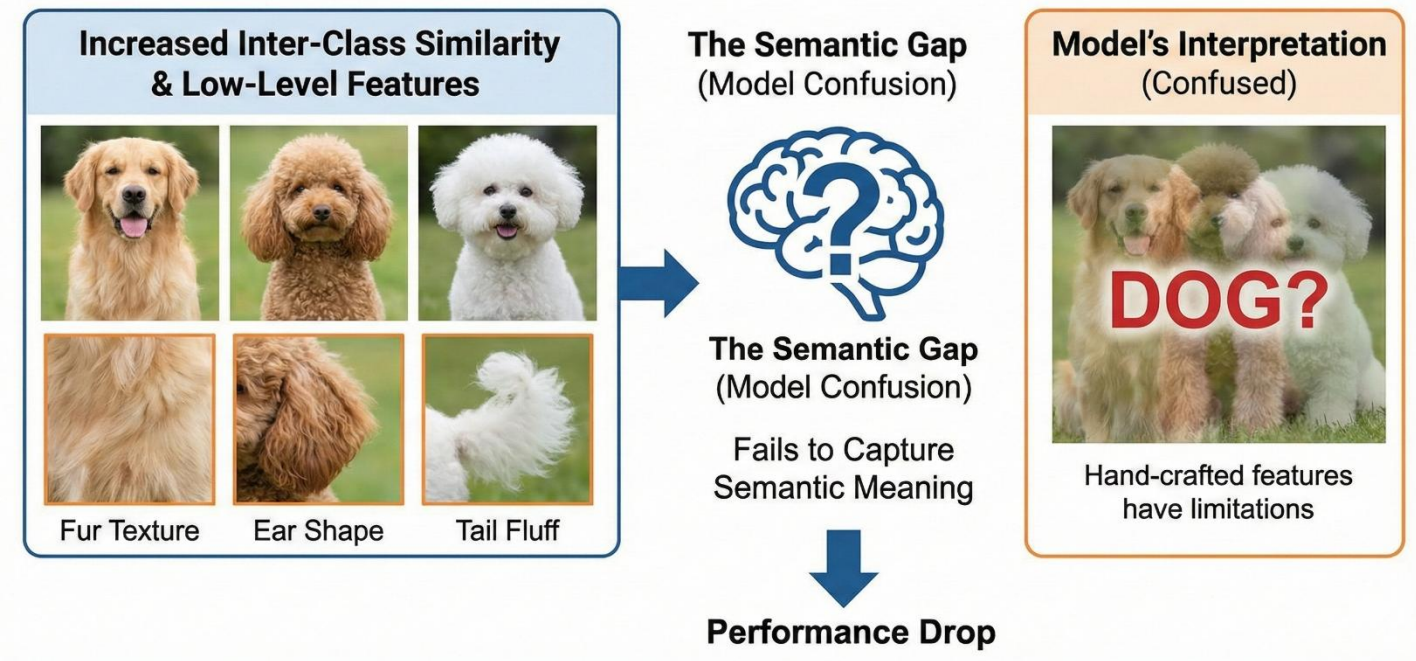
- Performance metrics table:
  - SIFT: ~55-65% accuracy (↓25-30%)
  - SURF: ~52-62% accuracy (↓26-31%)
  - HOG: ~48-58% accuracy (↓27-32%)
  - ORB: ~45-55% accuracy (↓28-33%)
- Significant performance drop!
- Struggle with fine-grained distinctions





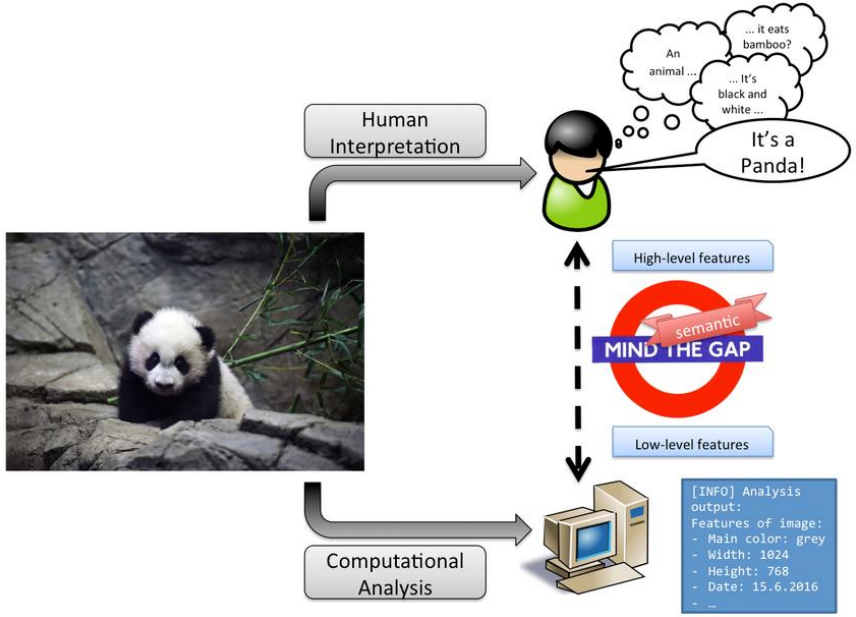
Why Did Performance Drop?

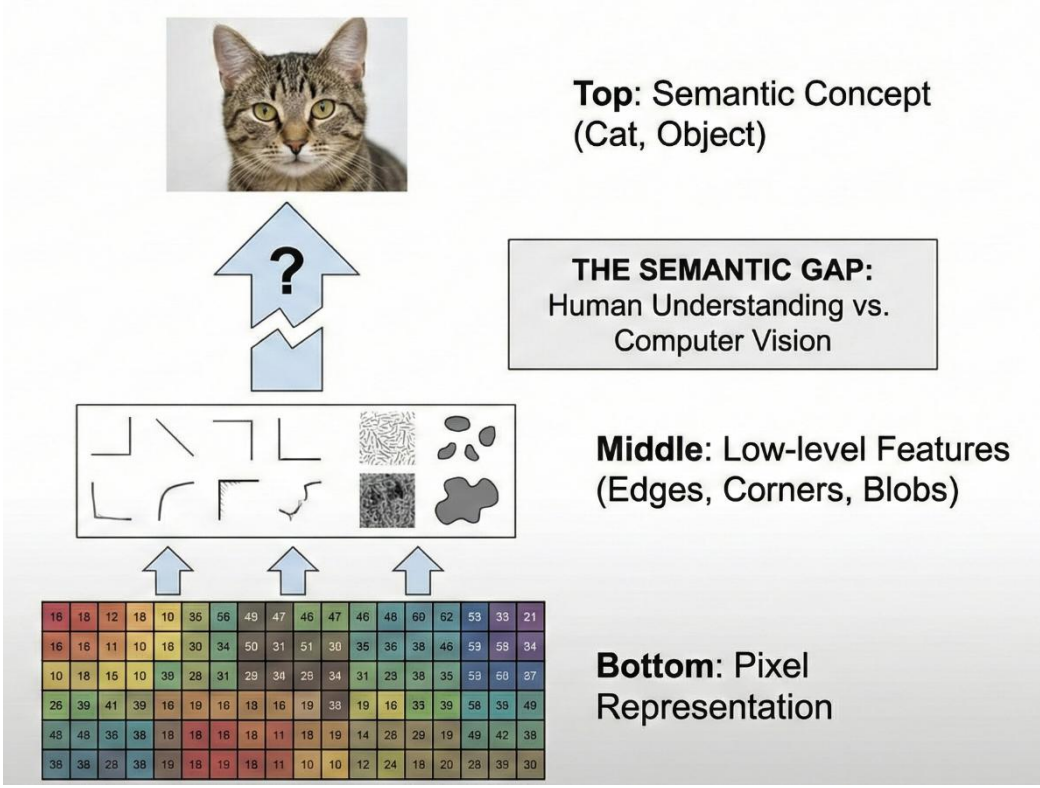
- Increased inter-class similarity
- Traditional descriptors capture low-level features
- Fail to capture semantic meaning
- Hand-crafted features have limitations
- → This leads us to the **Semantic Gap**



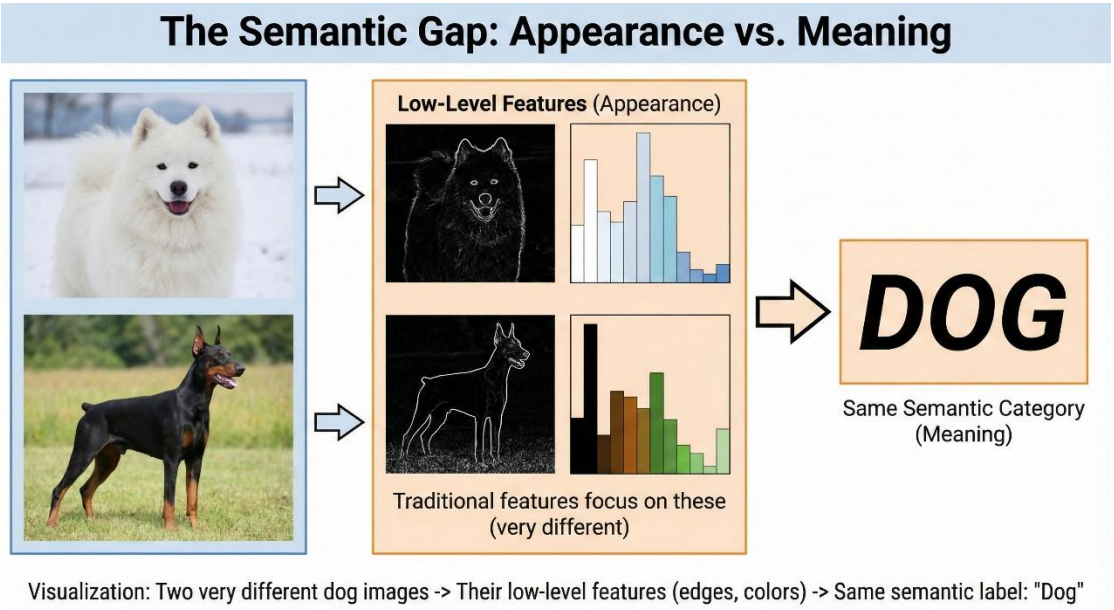
The Semantic Gap

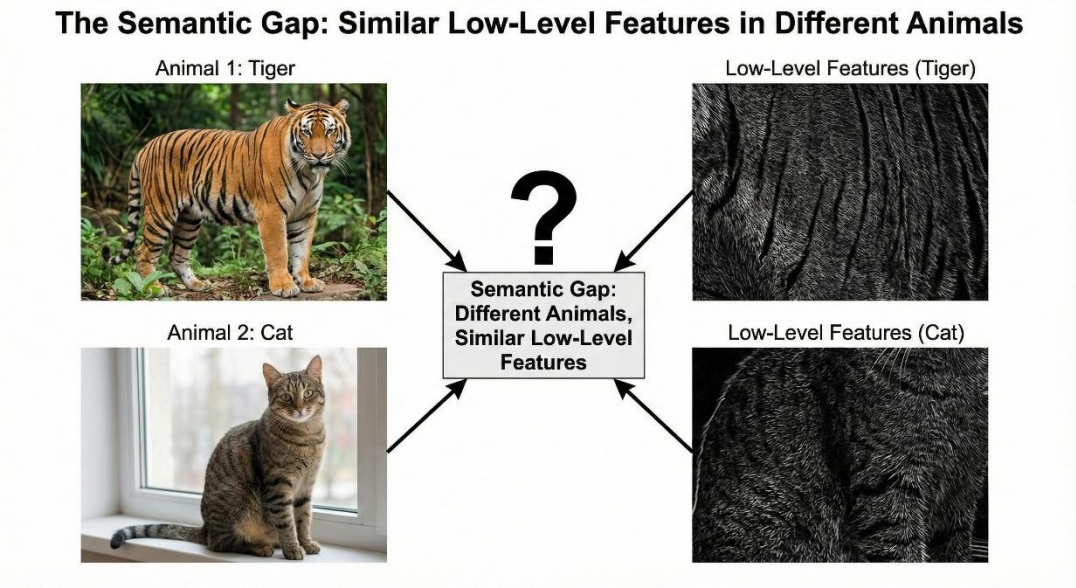
- **Definition:** The gap between low-level features (pixels, edges) and high-level semantic concepts (objects, categories)
- Humans understand "dog" as a concept
- Computer sees patterns of pixels and edges
- Traditional descriptors stay at low/mid-level

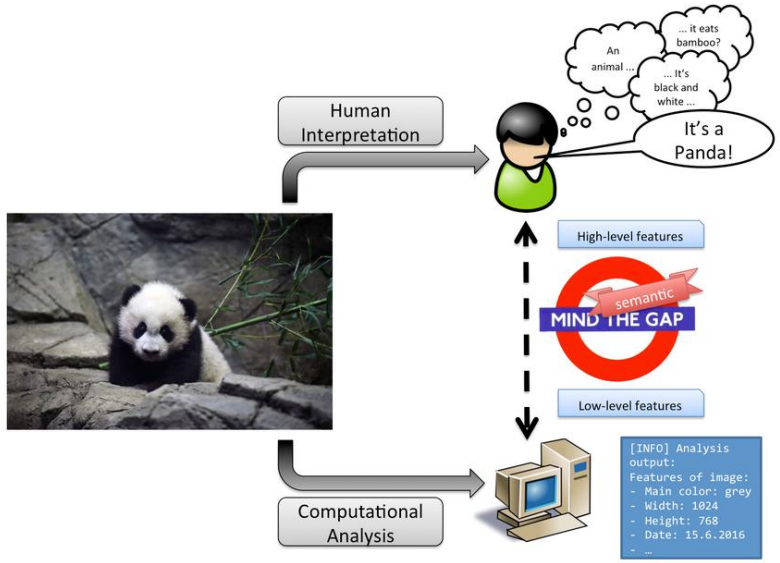




Semantic Gap Example

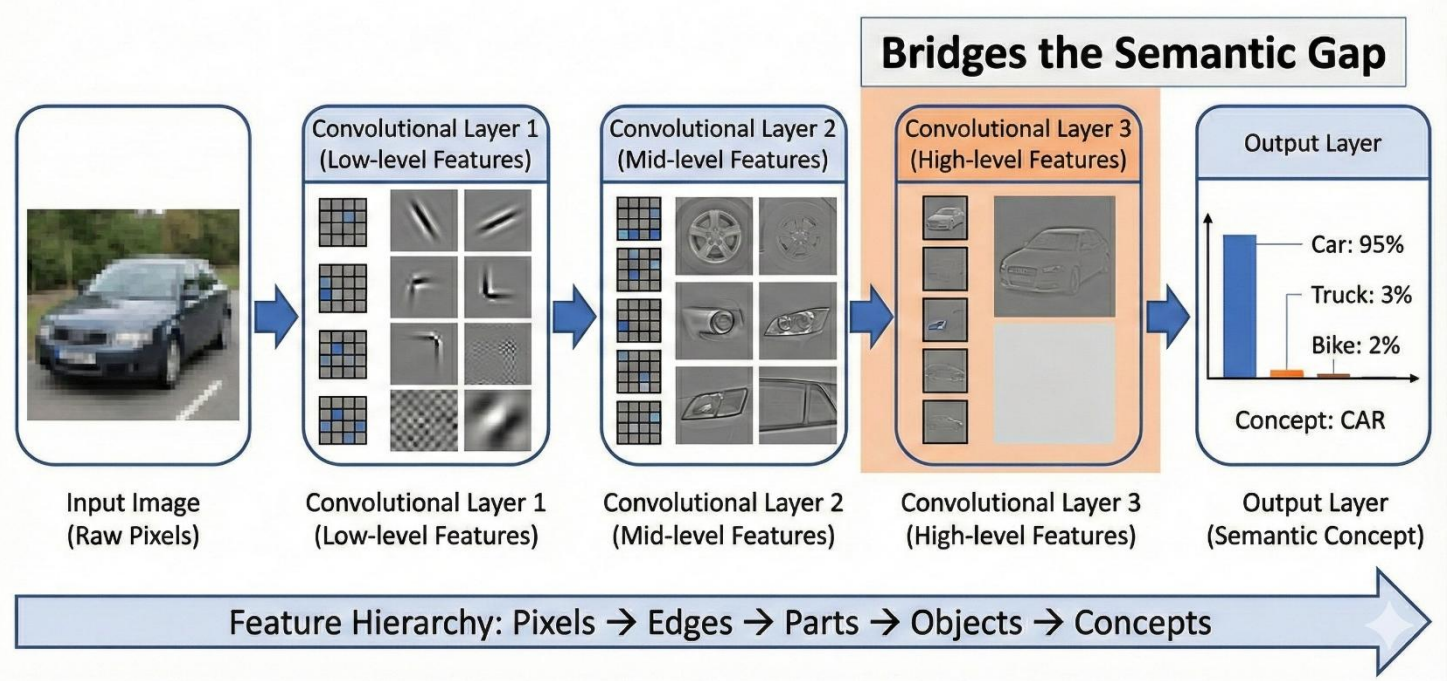


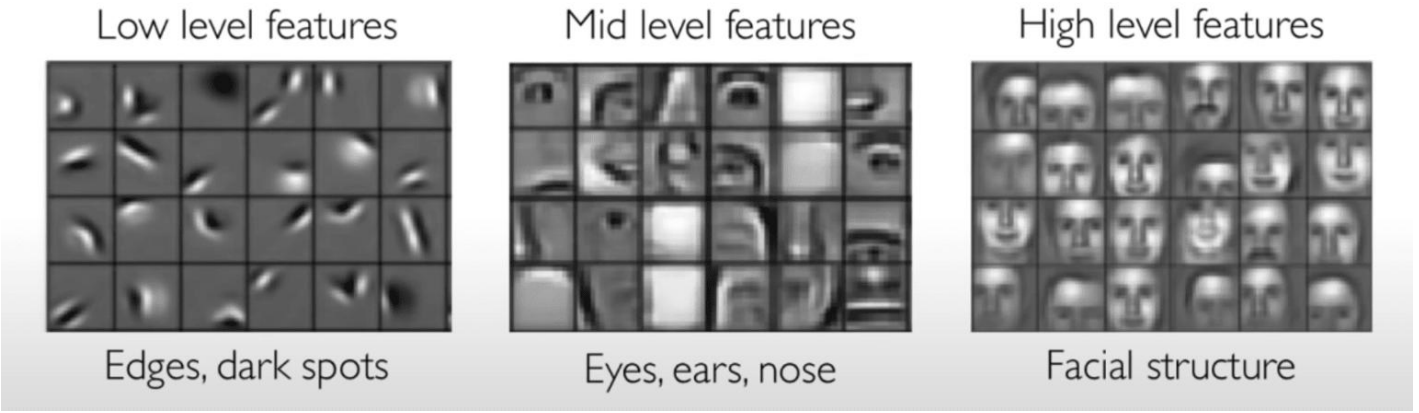




Enter Deep Learning - CNNs

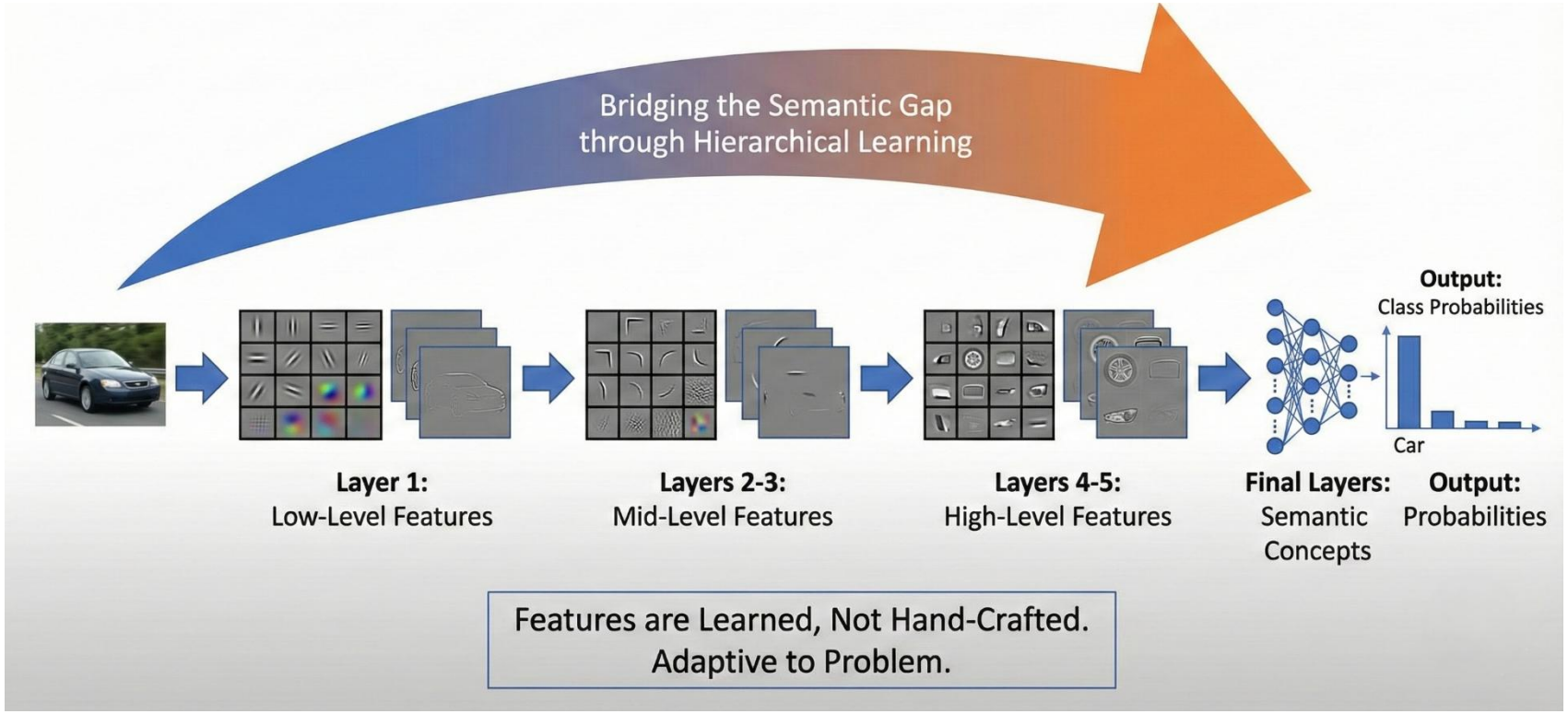
- Convolutional Neural Networks (CNNs)
- Learn features automatically from data
- Build hierarchical representations
- Bridge the semantic gap
- From pixels → edges → parts → objects → concepts





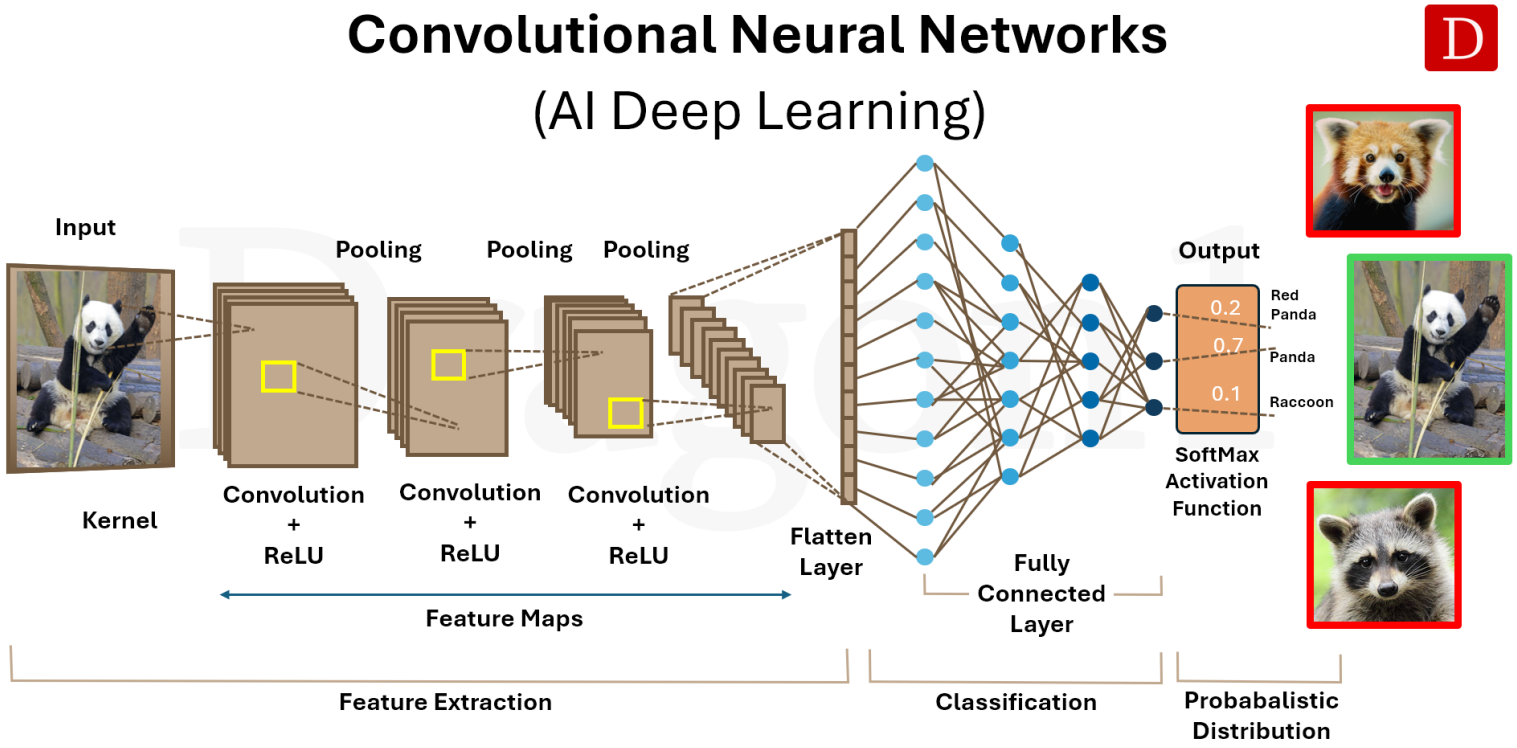
How CNNs Bridge the Semantic Gap

- **Layer 1:** Low-level features (edges, colors)
- **Layer 2-3:** Mid-level features (textures, patterns)
- **Layer 4-5:** High-level features (object parts)
- **Final layers:** Semantic concepts



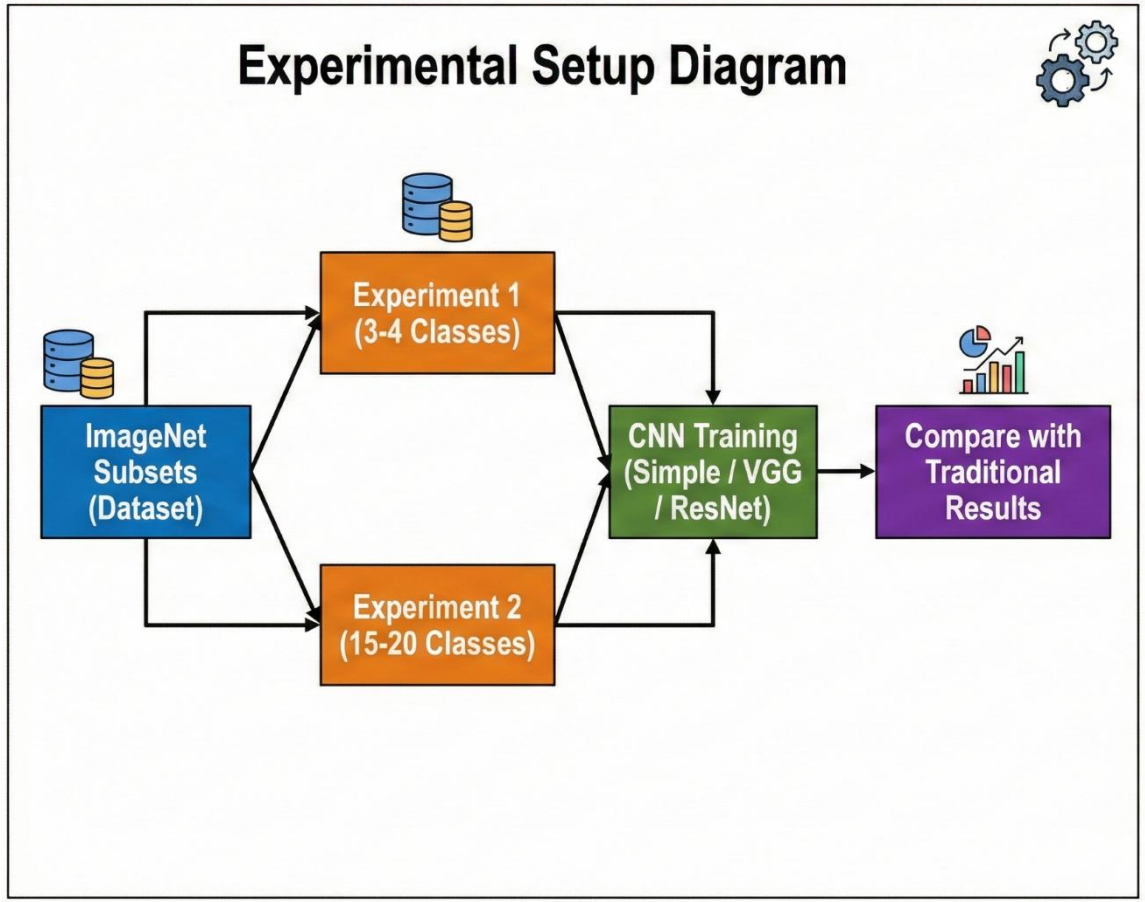
CNN Architecture Overview

- **Convolutional Layers:** Learn spatial features
- **Pooling Layers:** Reduce dimensionality
- **Activation Functions:** Introduce non-linearity (ReLU)
- **Fully Connected Layers:** Classification
- **Softmax:** Output probabilities



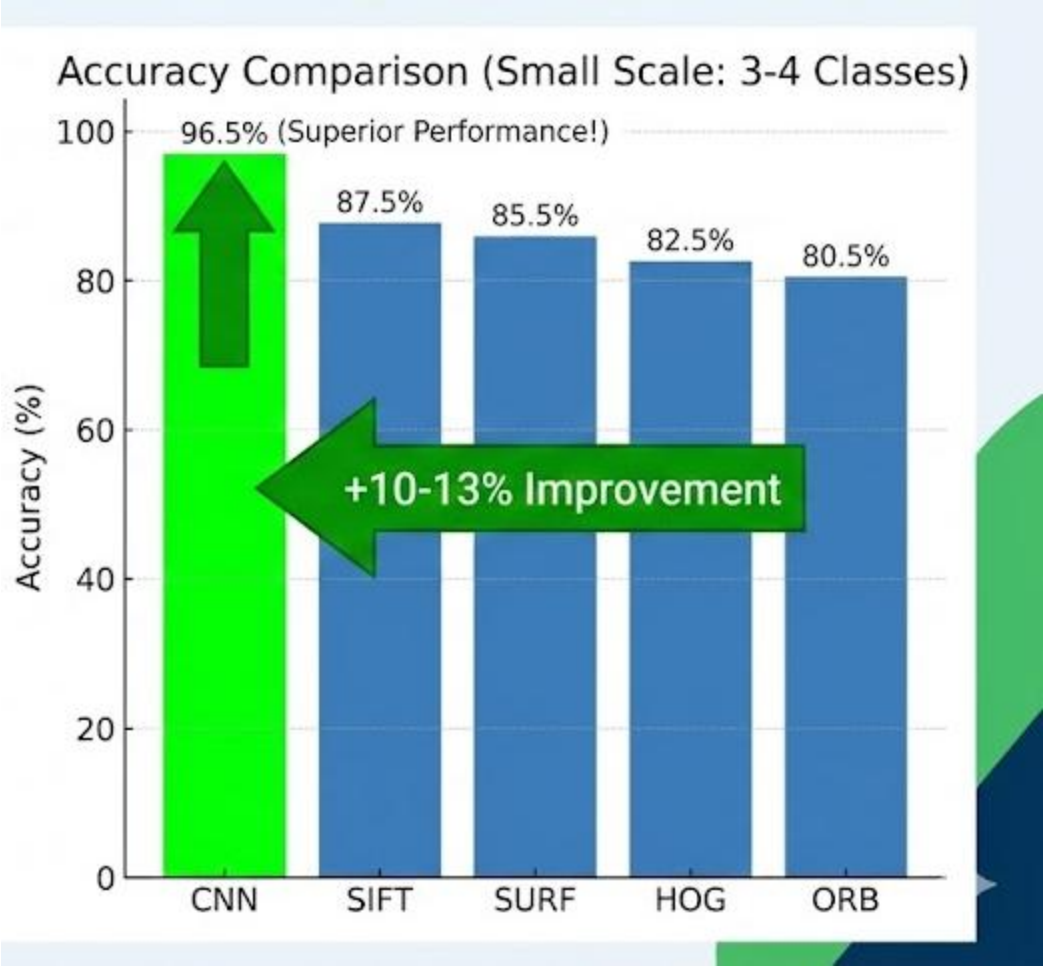
CNN Experiment - Same Setup

- Train CNN on same ImageNet subsets
- **Experiment 1:** 3-4 classes
- **Experiment 2:** 15-20 classes
- Architecture: Simple CNN (or VGG/ResNet)
- Compare with traditional descriptor results

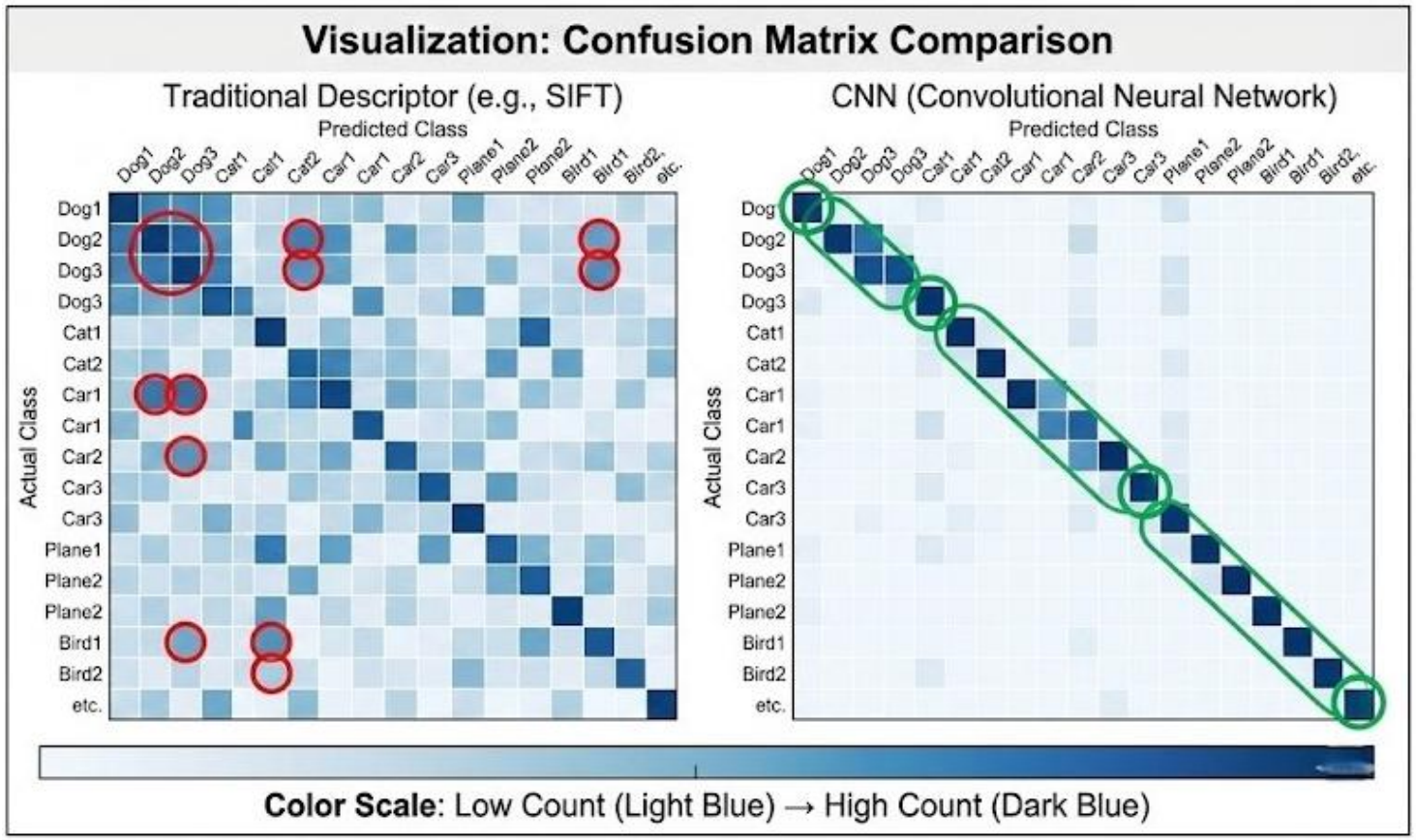


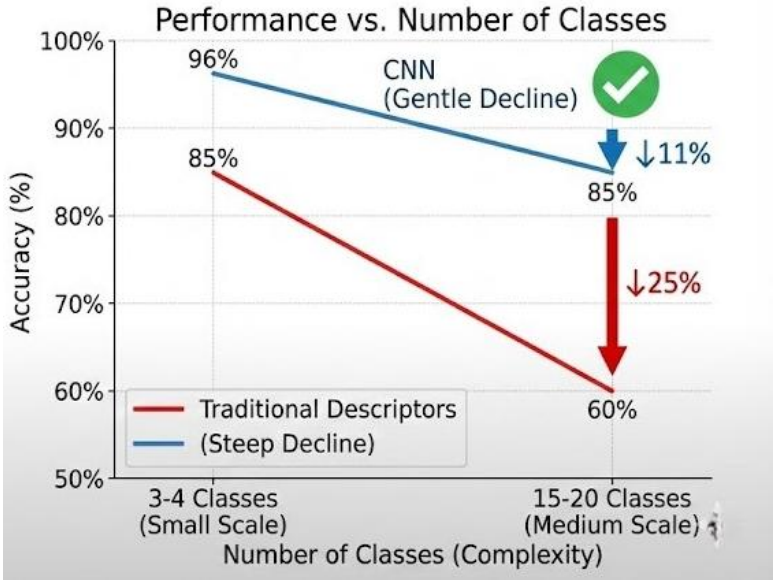
CNN Results - Small Scale (3-4 Classes)

- CNN Performance: ~95-98% accuracy
- Traditional best (SIFT): ~85-90% accuracy
- Improvement: +10-13%
- Even on simple tasks, CNNs excel



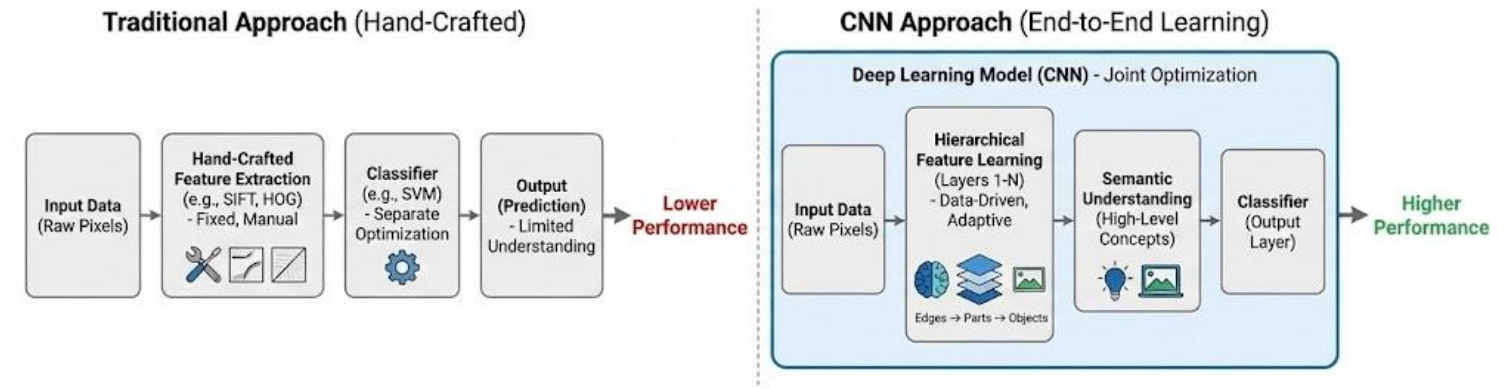
Performance Comparison

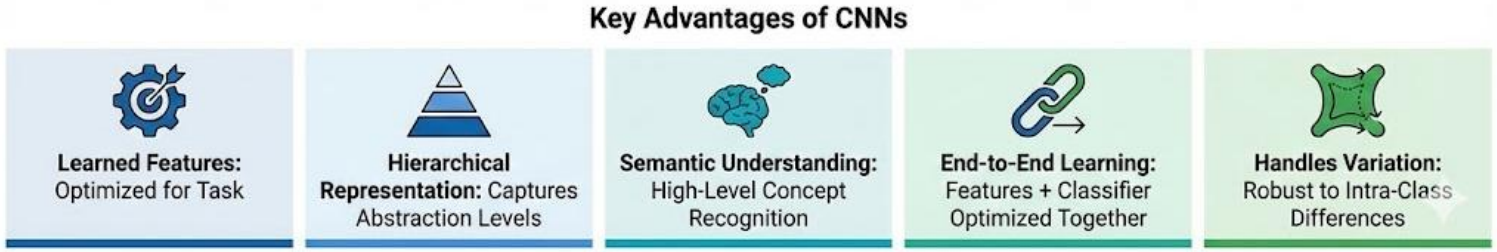




Why CNN Performs Better

- **Learned Features:** Optimized for specific task
- **Hierarchical Representation:** Captures abstraction levels
- **Semantic Understanding:** High-level concept recognition
- **End-to-End Learning:** Features + classifier optimized together
- **Handles Variation:** Robust to intra-class differences





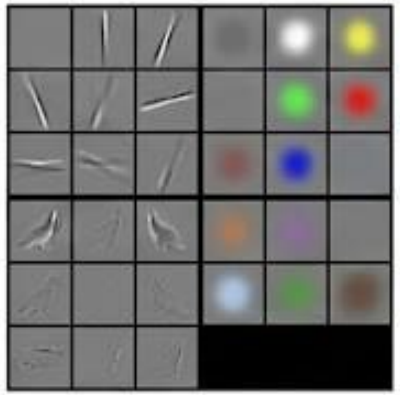
Visualizing CNN Feature Learning on Our Dataset

Content: What Our CNN Learned

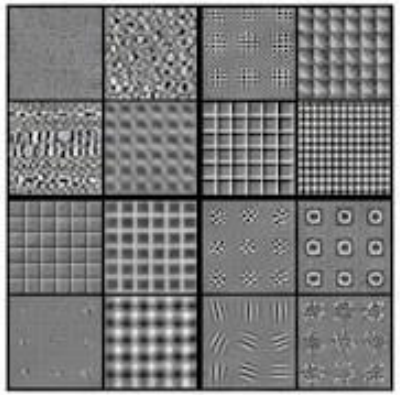
- Early layers: edges, colors (similar to traditional)
- Middle layers: textures, patterns
- Late layers: object parts and semantic features
- This is how semantic gap is bridged

Visualization: Bridging the Gap

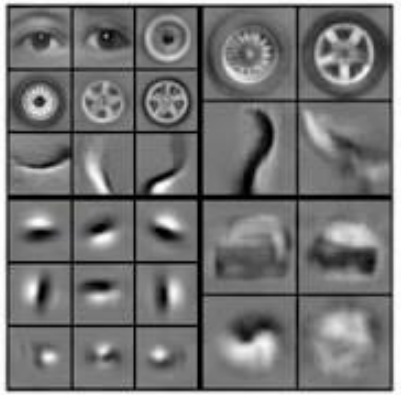
Feature Map Visualizations (Trained CNN)



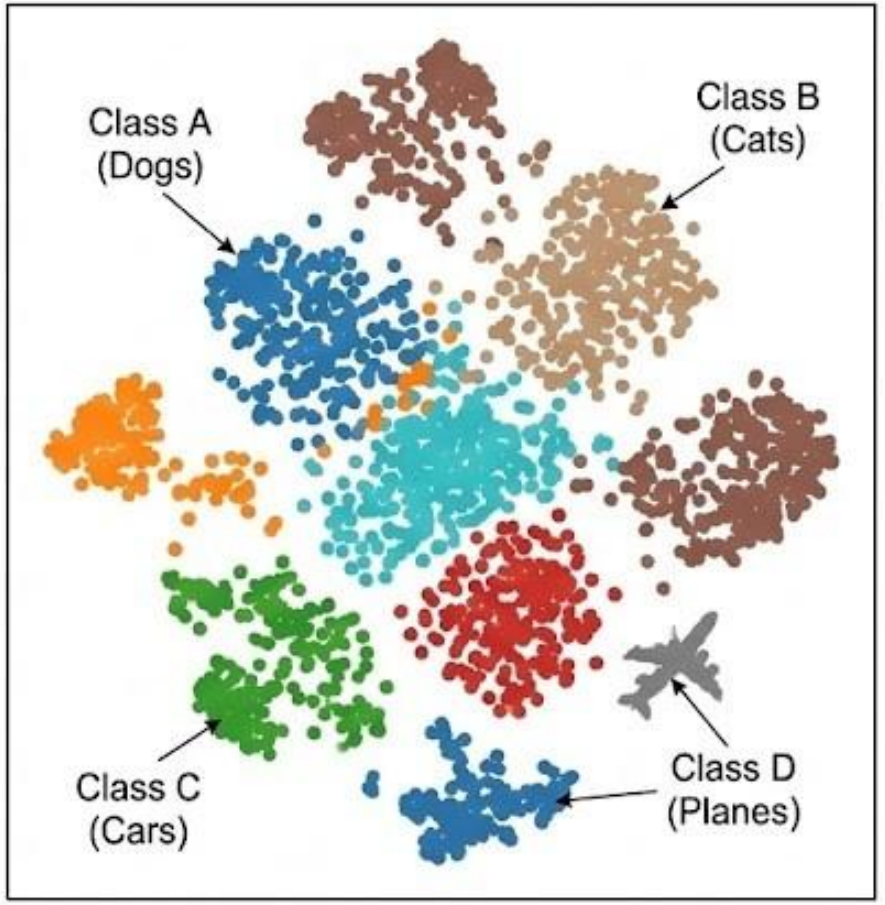
Early Layers
(Edges/Colors)



Middle Layers
(Textures/Patterns)



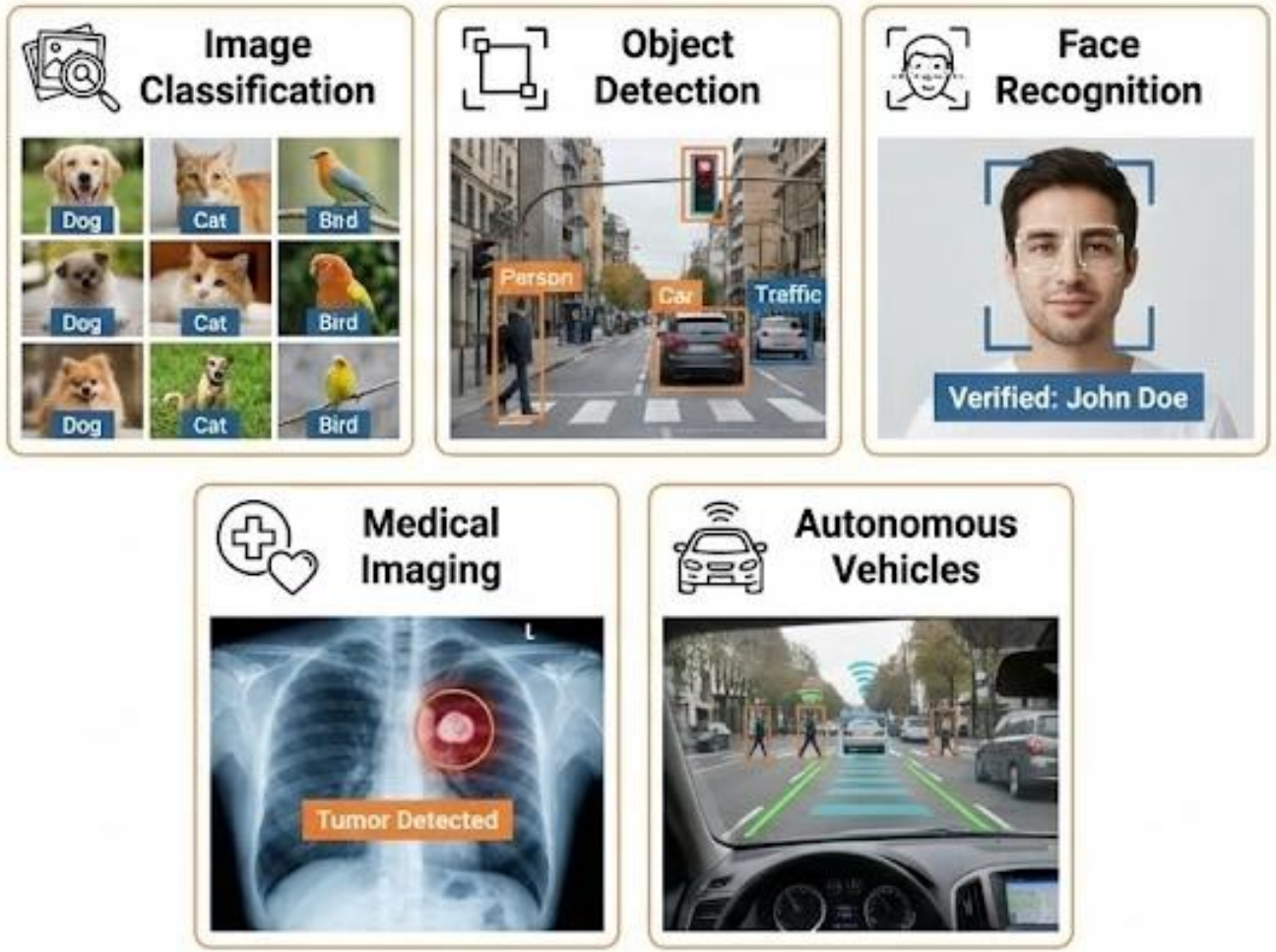
Late Layers
(Parts/Semantic)



t-SNE Plot
(Feature Space, Classes Well-Separated)

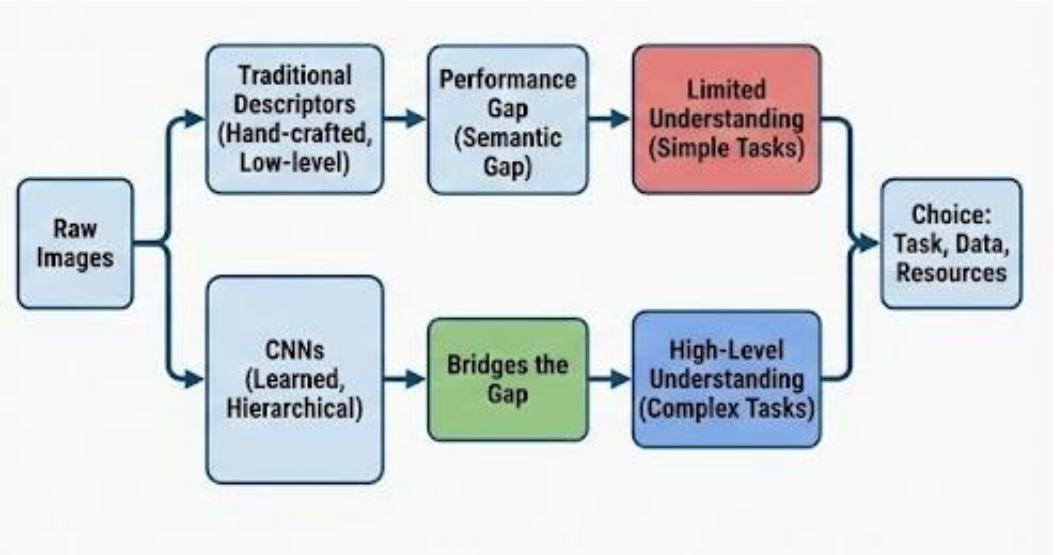
Real-World Impact

- **Traditional descriptors:** Still useful for simple tasks, limited data
- **CNNs:** State-of-the-art for complex vision tasks
- **Applications:**
  - Image classification
  - Object detection
  - Face recognition
  - Medical imaging
  - Autonomous vehicles



Key Takeaways

- Features are essential for image understanding
- Traditional descriptors use hand-crafted, low-level features
- Performance degrades significantly with increased complexity
- **Semantic gap** explains this limitation
- CNNs learn hierarchical features automatically
- CNNs bridge the semantic gap effectively
- Choice depends on task, data, and resources



Thank You

For Your Attention