ATME COLLEGE OF ENGINEERING

13th KM Stone, Bannur Road, Mysore - 560 028

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

LECTURE NOTES

COURSE: DEEP LEARNING
COURSE CODE: BCS714A
SEMESTER: VII

COURSE COORDINATOR: KAVITHA S

(ACADEMIC YEAR 2026-27)

MODULE-1

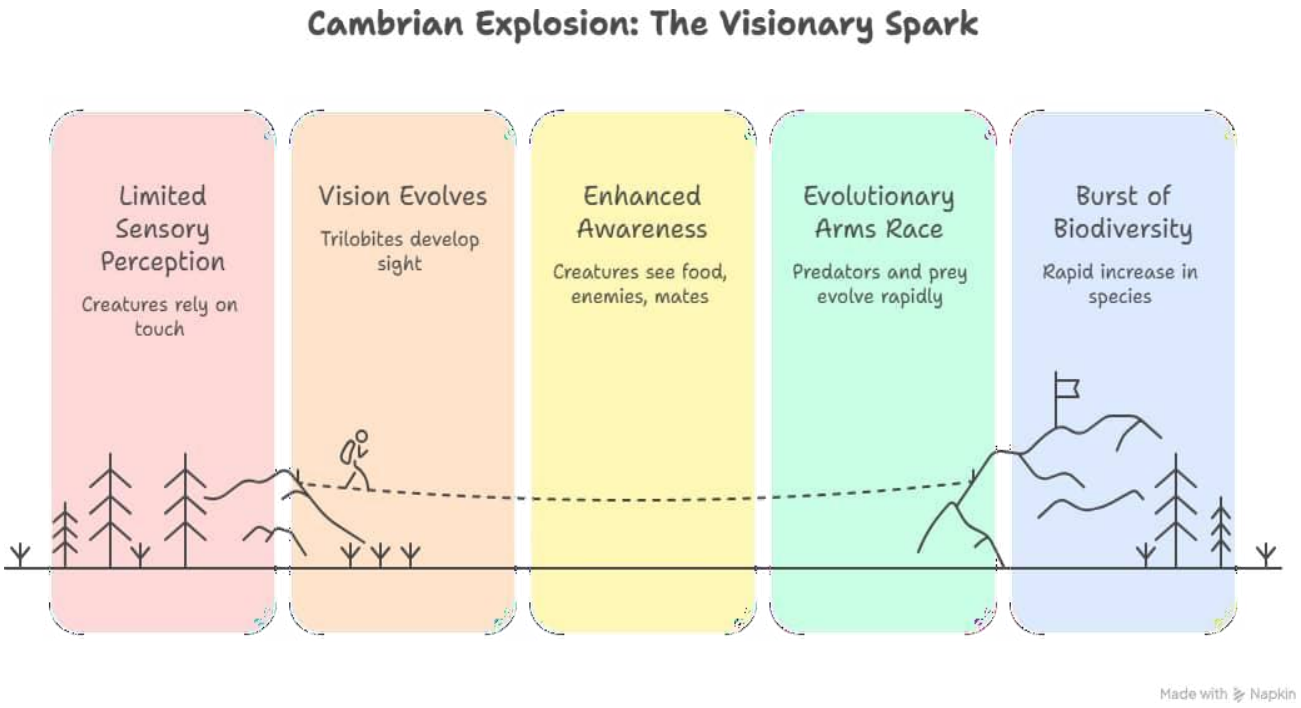
Introducing Deep Learning: Biological and Machine Vision: Biological Vision, Machine Vision: The Neocognitron, LeNet-5, The Traditional Machine Learning Approach, ImageNet and the ILSVRC, AlexNet, TensorFlow Playground.

Human and Machine Language: Deep Learning for Natural Language Processing: Deep Learning Networks Learn Representations Automatically, Natural Language Processing, A Brief History of Deep Learning for NLP, Computational Representations of Language: One-Hot Representations of Words, Word Vectors, Word-Vector rithmetic, word2viz, Localist Versus Distributed Representations, Elements of Natural Human Language.

Biological Vision (how it has influenced the field of machine vision):

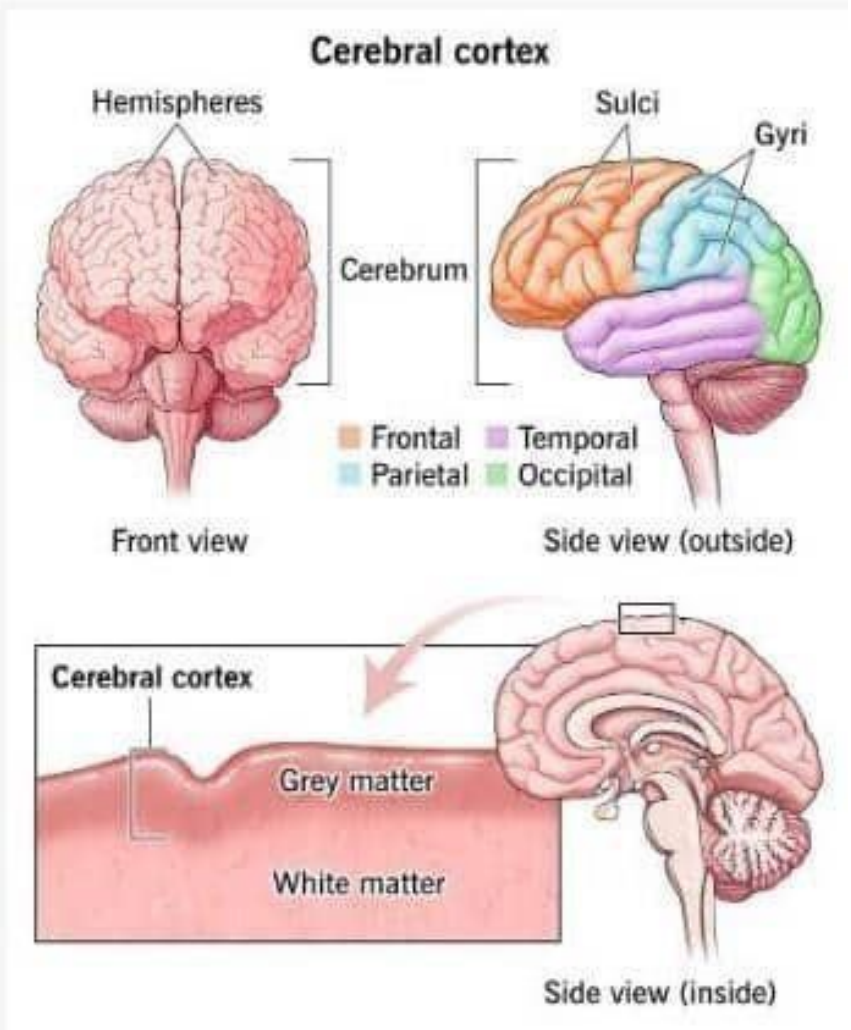
1. Origin of Vision and the Cambrian Explosion

- Around 550 million years ago, during the Cambrian period, there was a rapid increase in species.
- A key trigger was the evolution of vision in trilobites (ancient sea animals).
- Fossil evidence suggests the development of light detectors in trilobites (primitive marine animals) triggered an evolutionary "arms race."
- Vision provided major survival advantages—spotting food, avoiding predators, finding mates—faster and more accurately than smell.
- This led to an "arms race" in evolution: predators and prey had to evolve rapidly to survive, fueling a burst of biodiversity.



2. Visual Processing in the Brain

- In modern mammals, a large part of the cerebral cortex (the brain's outer layer) is used for vision.
- The cerebral cortex is gray matter (slow but highly interconnected, like city streets).
- Beneath it is the white matter (fast signal transmission, like highways).



Hubel & Wiesel's pioneering research (1950s):

- At Johns Hopkins, David Hubel and Torsten Wiesel studied how the mammalian brain processes visual information.
- They recorded neuron activity in the primary visual cortex of anesthetized cats while showing them images.
- Initially, showing simple dots yielded no results.
- They accidentally discovered that certain neurons responded strongly to edges (straight lines) at specific orientations.
- These were named simple neurons.

- Groups of simple neurons covered all 360° of edge orientations.

Figure 1.3 The Nobel Prize-winning neurophysiologists Torsten Wiesel (left) and David Hubel

- At Johns Hopkins, they studied how the brain processes vision using cats.
- They showed images (dots, lines) and measured neuron activity in the primary visual cortex (V1).

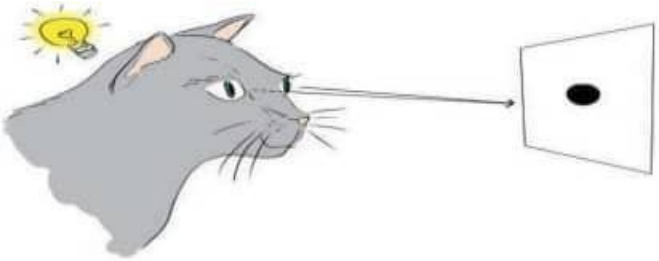


Figure 1.4 Hubel and Wiesel used a light projector to present slides to anesthetized cats while they recorded the activity of neurons in the cats' primary visual cortex. In the experiments, electrical recording equipment was implanted within the cat's skull. Instead of illustrating this, we suspected it would be a fair bit more palatable to use a lightbulb to represent neuron activation. Depicted in this figure is a primary visual cortex neuron being serendipitously activated by the straight edge of a slide.

- Scientists Hubel and Wiesel were showing pictures (like dots) to a cat while measuring brain activity from its visual cortex (part of the brain that sees).
- But nothing happened. The brain cells stayed quiet.
- Then, by accident, they moved a slide in front of the projector — and suddenly, one neuron started firing!

What Did They Discover?

- That brain cell wasn't interested in dots.
- It responded only when it saw a line or edge.
- Even more interesting: it only liked lines at a certain angle (like vertical or diagonal).

What Are "Simple Cells"?

- These are special neurons in the brain that:
- Detect straight lines or edges.
- Fire (activate) only when the line is at the right angle.

Example:

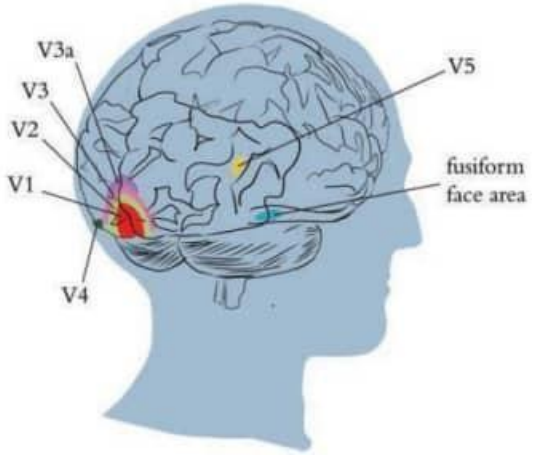
- One cell might fire only when it sees a vertical line.
- Another cell might prefer a diagonal line.

How the Brain Builds Vision Hierarchically:

- Layer 1: Simple Cells (in V1)
  - Detect edges or lines in specific orientations (e.g., vertical or horizontal).
  - Different cells prefer different angles (see Figure 1.5).
- Layer 2: Complex Cells
  - Combine information from multiple simple cells.
  - Detect shapes like corners, curves, or movement.
- Layer 3+: Higher Visual Areas (V2, V3, V4, IT)
  - More abstract processing (objects, faces, motion)

Different regions of the visual cortex are specialized for processing different visual features:

- V1: Edge orientation (basic feature maps)
- V2, V3, V3a: Combine features (contours, disparity, depth)
- V4: Color perception
- V5 (MT): Motion perception
- Fusiform Face Area (FFA): Face recognition



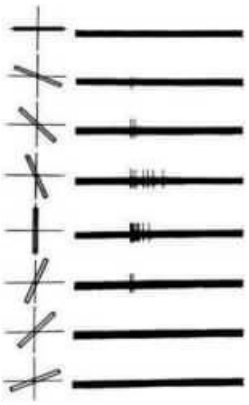
Simple Cells Detect Line Orientation:

What's Happening?

- This figure shows how a single brain cell in a cat's visual cortex responds to lines at different angles.
- When a vertical line is shown, the neuron fires a lot.
- When the line is slightly tilted, it fires less.
- When the line is almost horizontal, it doesn't fire at all.

Why?

- That neuron is tuned to vertical lines only.



How the Brain Builds Complex Images:

What's Happening?

- Figure shows how the brain builds up a full image using layers:
- **Eyes** see the image (e.g., a cat ✯ or mouse).
- **Simple cells** in the brain detect **straight lines** (|, —, , /).
- **Complex cells** combine lines to detect **curves or shapes** (like a circle or corner).
- **Higher brain layers** combine those shapes to recognize **whole objects** — a cat, mouse, dog, or bird.

Why?

- The brain works **step-by-step** to build up understanding:
  - From **edges**, to **shapes**, to **objects**.

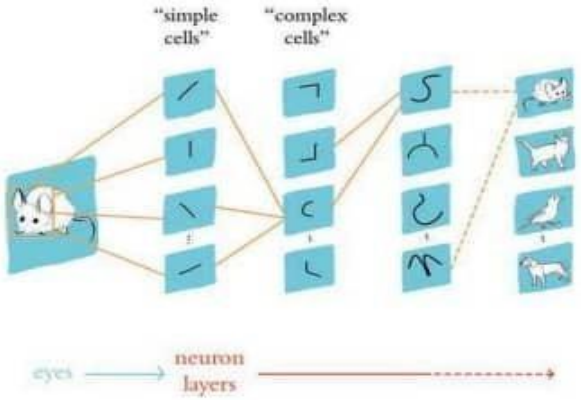


Figure 1.6 A caricature of how consecutive layers of biological neurons represent visual information in the brain of, for example, a cat or a human

- You find the **edge pieces** (simple cells),
- Then, you group by **colors and curves** (complex cells),
- Finally, the full **picture appears** (higher-level vision).

Parts Of The Human Brain That Handle Different Aspects Of Vision

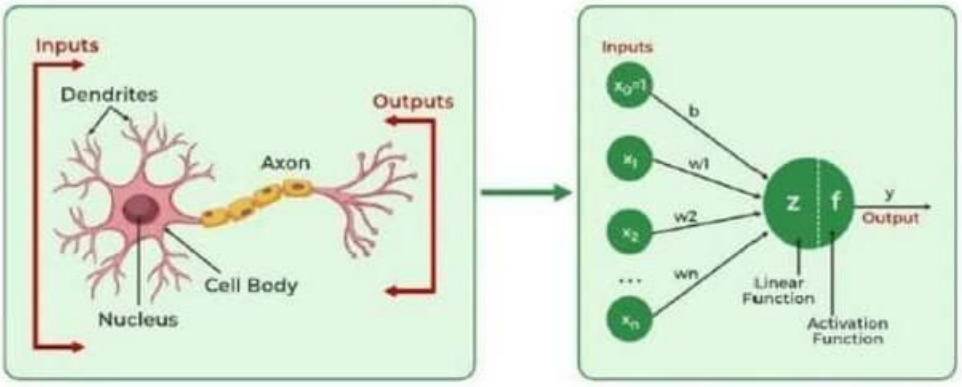
Brain Area	Function
V1	Detects edges and line orientations (simple cells)
V2, V3, V3a	Start combining visual data (shapes, depth, patterns)
V4	Specializes in **color** recognition
V5	Specializes in detecting **motion**
Fusiform Face Area	Recognizes **faces**

The visual system is organized in a **hierarchical layered architecture**:

1. **Eyes** capture raw pixel-level data (like a camera sensor).
2. **Simple cells** in V1 detect **oriented edges** (like applying Sobel filters).
3. **Complex cells** combine edge information to detect **curves, angles, corners.**
4. **Higher-level neurons** integrate patterns into **shapes, objects, and categories** (e.g., face, cat).

Signal Flow:

Input → V1 (edges) → V2/V3 (patterns, shapes) → IT Cortex (object identity)



- We studied biological vision **not just because it's interesting**, but because it **inspired deep learning** and modern **machine vision**.

What Is Machine Vision?:

- **Machine vision** is when computers are taught to "see" and understand images (like faces, objects, scenes).
- This idea has **two paths**:
  - **Traditional Machine Learning (ML)**
  - **Deep Learning (DL)**

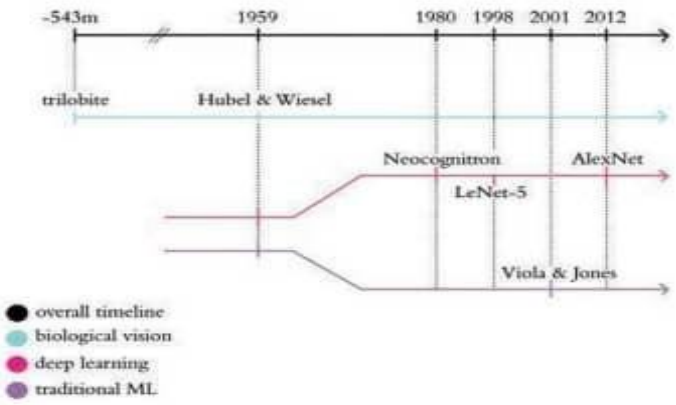


Figure 1.8 Abridged timeline of biological and machine vision, highlighting the key historical moments in the deep learning and traditional machine learning approaches to vision that are covered in this section

- Biological Vision (Blue Line)
  - Starts with **trilobites** (ancient animals with eyes) – shows how vision evolved.
  - **1959**: Big breakthrough — **Hubel & Wiesel** discovered how brain cells respond to visual patterns (like lines).
  - This inspired how we design artificial vision systems.

Machine Vision: Split into Two Tracks:

A. Deep Learning (Pink Line):

- Inspired by biological brain structure.
- Uses **neural networks** to learn from images.
- Key milestones:
  - **Neocognitron** (1980): Early neural net for vision.
  - **LeNet-5** (1998): For reading handwritten digits (like on checks).
  - **AlexNet** (2012): Big success in image recognition — started the deep learning boom.

B. Traditional ML (Purple Line):

- Uses **handcrafted features** (you manually tell the system what to look for in an image).
- Example:
    - **Viola & Jones (2001)**: Face detection using basic patterns.

NEOCOGNITRON:

- **Neocognitron** is a type of **artificial neural network** developed by **Kunihiko Fukushima** in **1980**.

  It was one of the **first models** inspired by the **human visual system**—particularly the way our brain recognizes shapes and patterns.

Why Was It Important?

- It could recognize **objects** (like letters or digits) even if they were:
    - **Shifted** (moved left/right/up/down)
    - **Distorted**
    - **Partially hidden**
- This was a **big deal** at the time, because earlier systems needed **perfect input**!
- Neocognitron has a **hierarchical layered structure**, just like the **visual cortex** in our brain.

  It has two main types of layers:

1. S-Cells (Simple Cells)

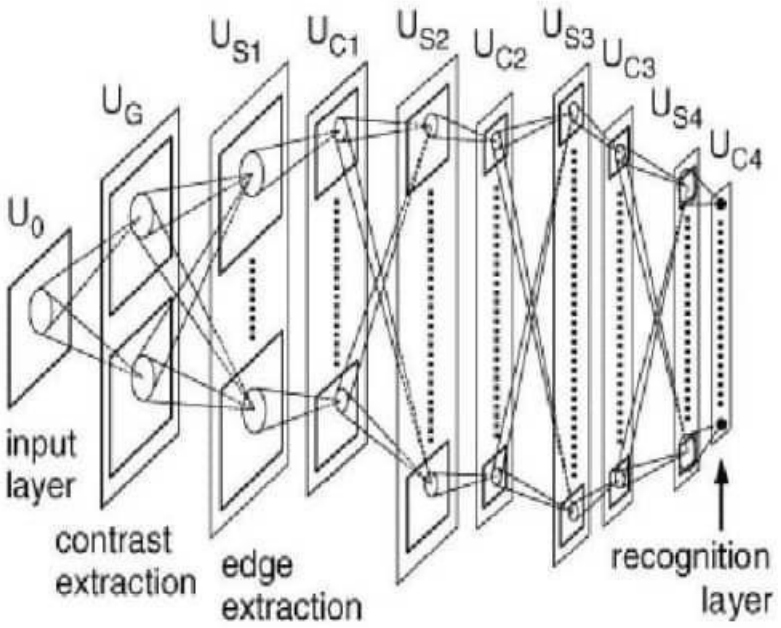
- Detect **basic features** like lines, edges, corners.
- Similar to what Hubel & Wiesel found in biological vision.

2. C-Cells (Complex Cells)

- Combine outputs from multiple S-cells.
- Help make the recognition **tolerant to shifts** (i.e., position changes).

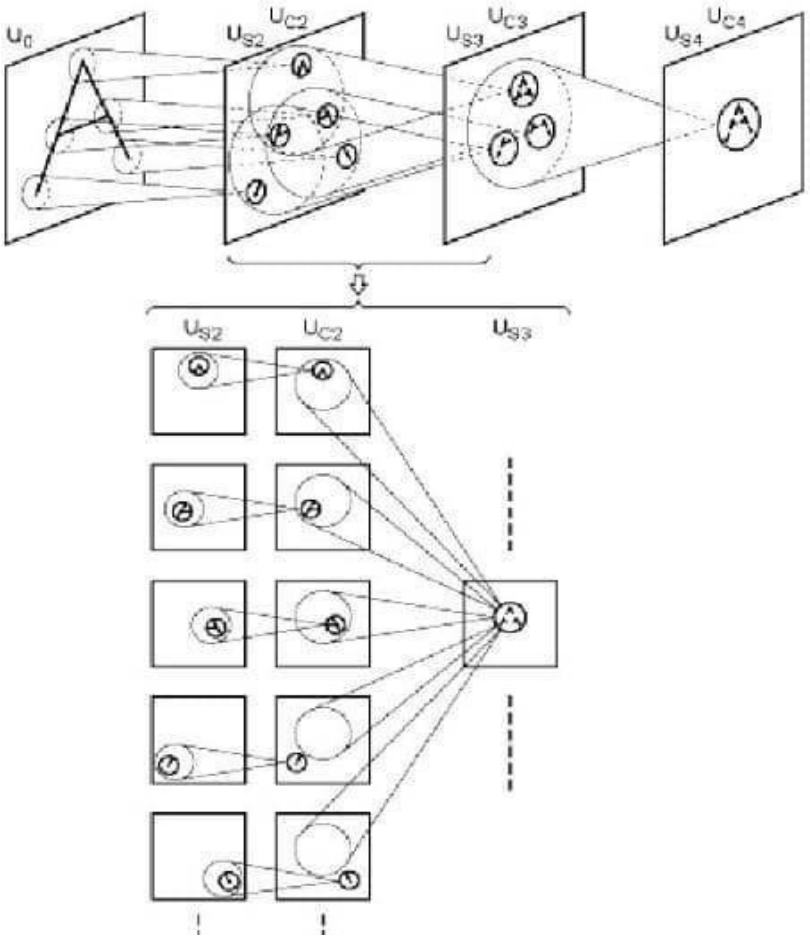
The Neocognitron is an early type of hierarchical neural network introduced by Kunihiko Fukushima in 1980, designed specifically for pattern recognition and machine vision. Its architecture consists of multiple layers of S-cells (simple cells) and C-cells (complex cells) arranged hierarchically to progressively extract increasingly abstract features from visual inputs. The S-cells perform convolution-like operations, detecting local features such as edges or orientations, while the C-cells provide spatial invariance by pooling over neighboring S-cells, making the network robust to shifts and distortions in the input. These alternating layers allow the Neocognitron to recognize patterns regardless of their exact position or slight variations, a key challenge in vision tasks. The network ends with higher-level layers that integrate the extracted features for final classification. The significance of the Neocognitron lies in its biologically inspired design and its role as a precursor to modern convolutional neural networks

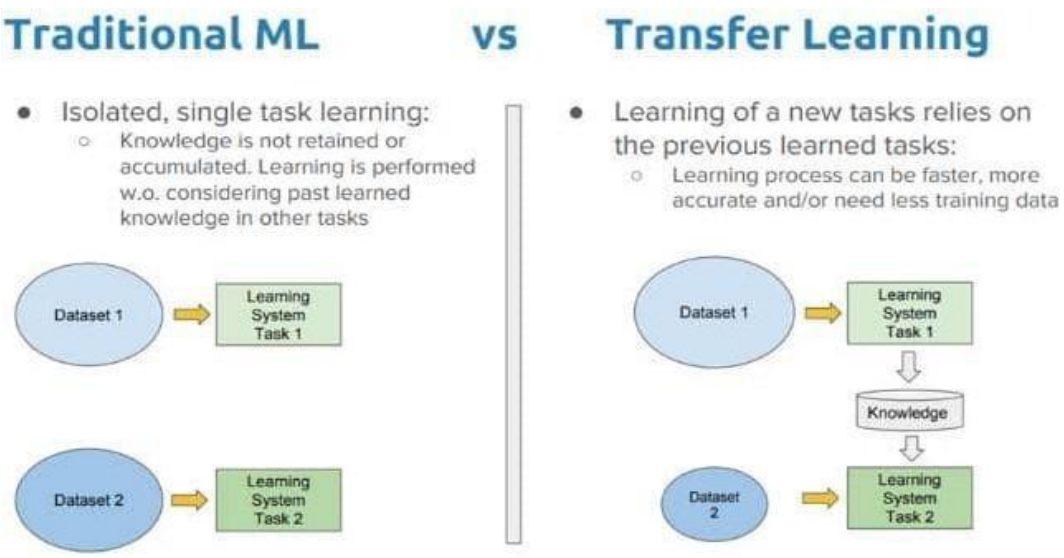
(CNNs), laying the groundwork for hierarchical feature extraction, translation invariance, and deep learning approaches widely used in image recognition today.

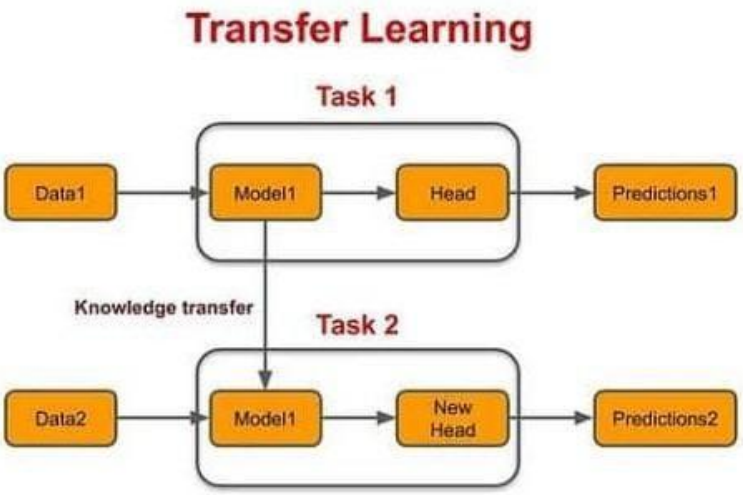


Neocognitron neural network

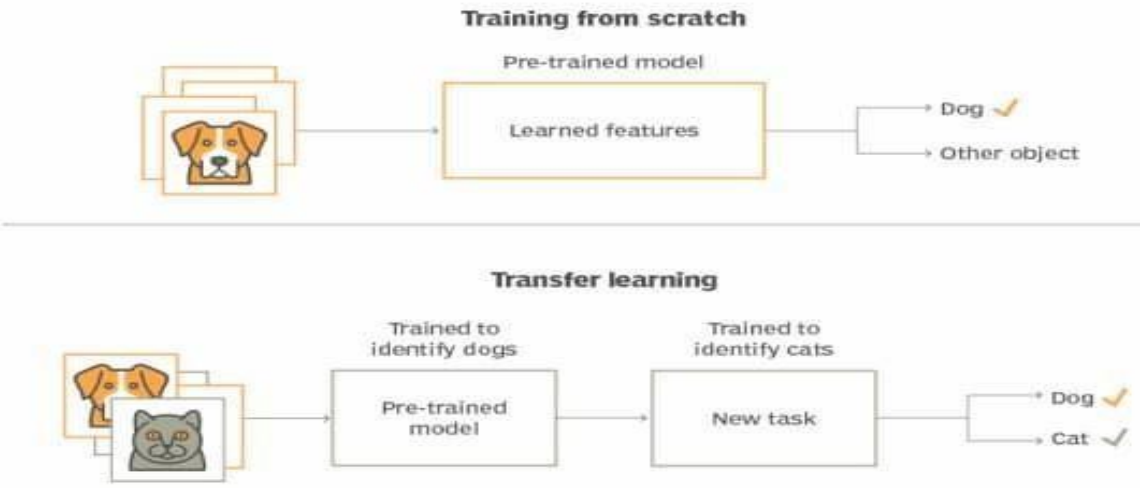
The process of pattern recognition in the neocognitron







For example, the knowledge gained from recognizing an image of a dog in a supervised ML system could be transferred to a new system to recognize images of cats. The new system filters out images it already recognizes as a dog.



Transfer Learning:

Build a model to detect defects in mechanical parts using images, but:

- You don't have a lot of images
- Training a deep CNN from scratch is slow and needs lots of data

Solution: Use Transfer Learning

- **Start with a pre-trained model (like ResNet, VGG, or MobileNet)**
  These are trained on ImageNet, a huge dataset with millions of images.
- **Freeze the earlier layers**
  These layers have learned to detect generic features (edges, shapes, colors)
- **Replace the final layer**
  With a new layer for your specific defect classes (e.g., crack, corrosion, missing screw)
- Fine-tune the model using your smaller dataset

LeNet-5:

- LeNet-5 is one of the first Convolutional Neural Networks (CNNs) developed by Yann LeCun in the 1990s. It was designed to recognize handwritten digits (like 0 to 9) — for example, to read ZIP codes on mail.
- LeNet-5 was inspired by biological vision systems.
- It was based on the earlier idea of the Neocognitron.

LeNet-5 is one of the earliest and most influential convolutional neural network (CNN) architectures, developed by Yann LeCun and colleagues in 1998 for handwritten digit recognition (such as the MNIST dataset). It follows a **hierarchical layered structure** that extracts increasingly abstract features from raw pixel data. The network begins with an **input layer** of 32×32 grayscale images, slightly larger than the original MNIST digits (28×28), to preserve boundary information during convolution. The first processing stage is **C1**, a convolutional layer with six filters of size 5×5, producing six 28×28 feature maps. This is followed by **S2**, a subsampling (average pooling) layer that reduces each feature map to 14×14, making the model more computationally efficient and translation-invariant. The next stage, **C3**, is another convolutional layer with sixteen 5×5 filters connected in a selective pattern to the previous six maps, generating 16 feature maps of size 10×10. This selective connectivity reduces computational complexity while encouraging feature diversity. Layer **S4** then performs subsampling again, reducing each feature map to 5×5. After this, **C5**, a convolutional layer with 120 filters of size 5×5, operates across the entire 5×5 region of each input, resulting in 120 output nodes, effectively acting like a fully connected layer. These outputs are then passed to **F6**, a fully

connected layer with 84 units that mimic biological neurons. Finally, the **output layer** consists of 10 neurons, one for each digit class (0–9), using a softmax function to produce classification probabilities. Overall, LeNet-5 demonstrates a clear hierarchical architecture where lower layers capture simple features like edges and curves, while deeper layers combine them into higher-level representations for digit recognition.

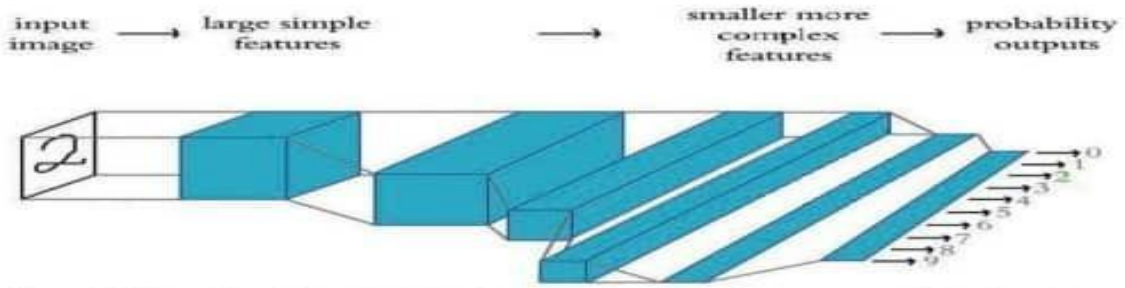
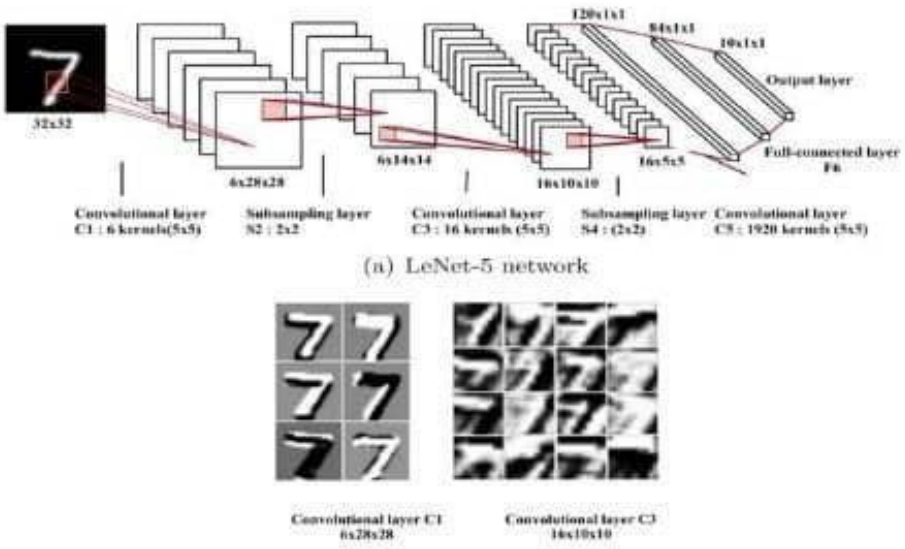


Figure 1.11 LeNet-5 retains the hierarchical architecture uncovered in the primary visual cortex by Hubel and Wiesel and leveraged by Fukushima in his neocognitron. As in those other systems, the leftmost layer represents simple edges, while successive layers represent increasingly complex features. By processing information in this way, a handwritten "2" should, for example, be correctly recognized as the number two (highlighted by the green output shown on the right).

Diagram shows a CNN pipeline:

- Starts with an input image (like a digit "2")
- Layers extract simple features (edges, curves)
- Then, deeper layers capture more complex patterns
- Finally, it gives probabilities for what digit it is (e.g., 0-9)

- This process lets the computer automatically learn features from images.
- Used by the U.S. Postal Service to read ZIP codes.
- It was one of the first real-world applications of deep learning.
- It showed that CNNs can learn from raw images without needing manual work



The Traditional Machine Learning Approach:

- In TML, you might write code to find shapes manually.
- In deep learning (like LeNet-5), the model learns the shapes on its own.

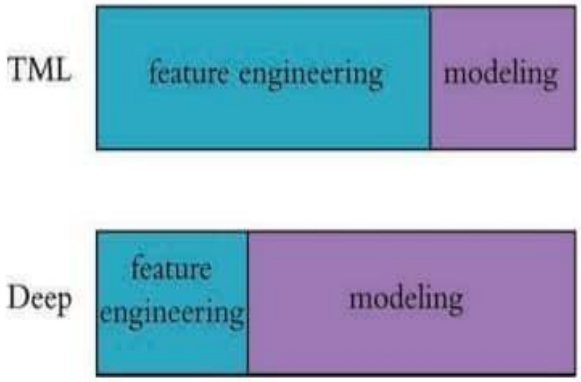


Figure 1.12 Feature engineering—the transformation of raw data into thoughtfully transformed input variables—often predominates the application of traditional machine learning algorithms. In contrast, the application of deep learning often involves little to no feature engineering, with the majority of time spent instead on the design and tuning of model architectures.

ImageNet:

What is ImageNet?

- It's a huge database of labeled photos (14 million+ images).
- Created by Fei-Fei Li, it's like a visual dictionary where each image is tagged with what it contains (like “dog,” “car,” “tree,” etc.).
- It includes 22,000+ categories — from everyday items to animals and machines.

Why was this a breakthrough?

- Earlier neural networks (like LeNet-5) used smaller datasets like handwritten digits (only tens of thousands of images).
- But ImageNet gave researchers access to millions of images — so they could train much deeper, smarter models.

ILSVRC (ImageNet Large Scale Visual Recognition Challenge):

- Started in 2010, this is a yearly competition.
- Teams from all over the world use a subset of ImageNet to:
  - Classify images (What is in this image?)
  - Detect objects (Where is the dog in the picture?)

The success of ImageNet + ILSVRC led to:

- Major advances in deep learning.

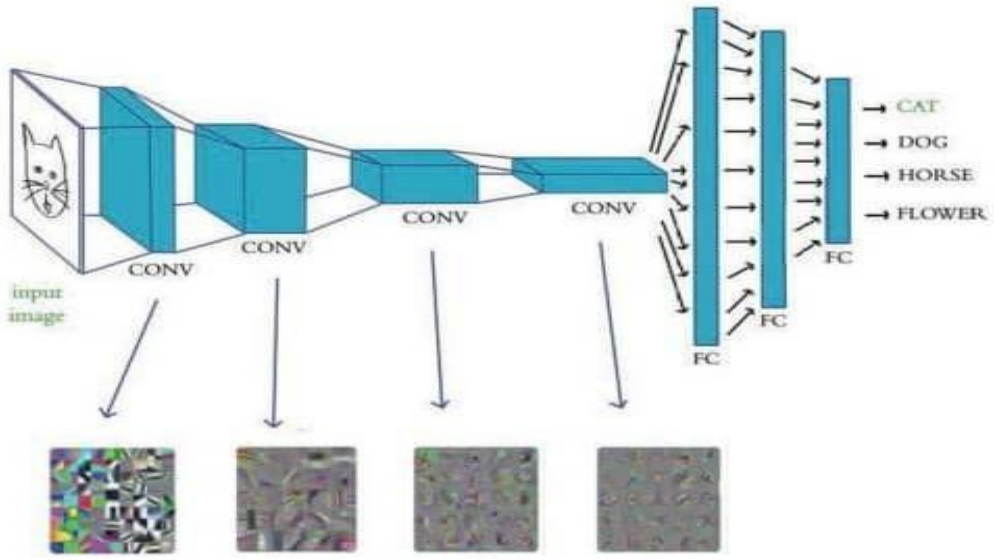
- The birth of famous models like AlexNet, VGG, ResNet.

Imagine you're training a robot to recognize tools.

- LeNet-5: It learns only from 1,000 pictures of spanners and hammers.
- ImageNet: Now the robot gets to learn from 14 million images of all types of tools, machines, animals, and more. It becomes far smarter and more accurate.

AlexNet:

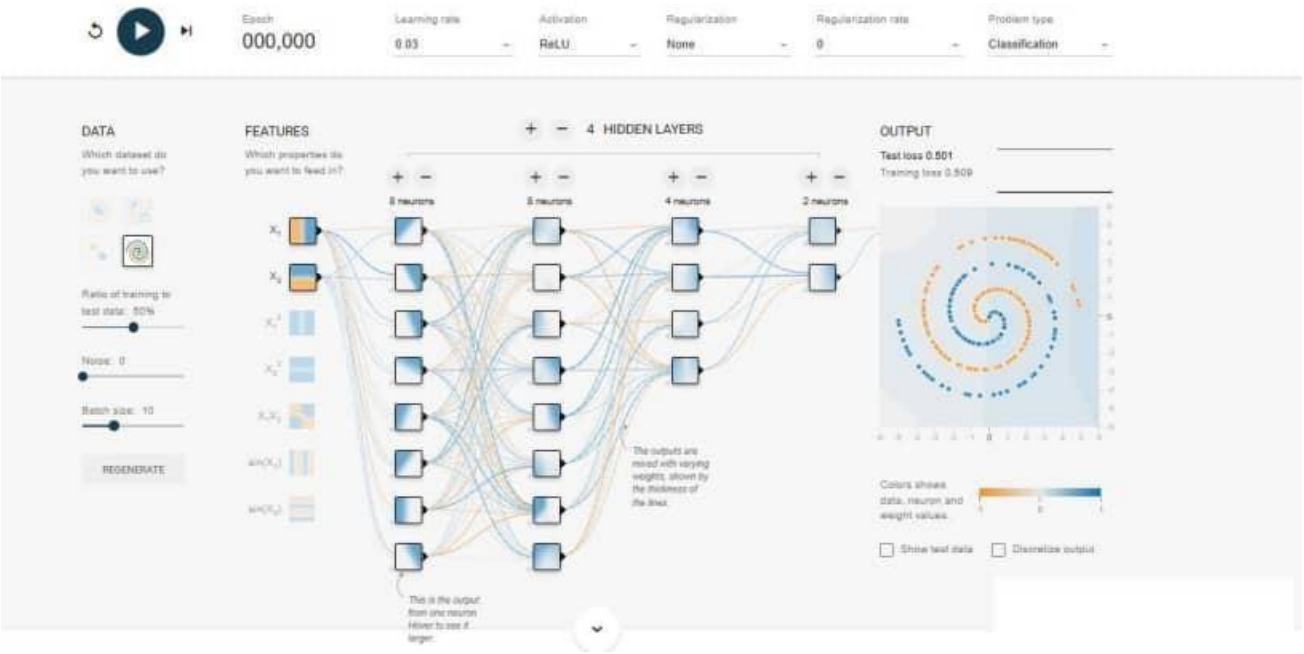
- AlexNet is a deep learning model developed by Alex Krizhevsky, Ilya Sutskever, and Geoffrey Hinton.
- Before AlexNet, traditional machine learning models were used for image classification. AlexNet showed that deep CNNs trained with GPUs can achieve much higher accuracy if trained on large datasets like ImageNet.
- **Architecture**: The diagram illustrates AlexNet's hierarchical architecture, which is an evolution of the earlier LeNet-5 model.
- It consists of multiple layers: **CONV (Convolutional Layers)**: These layers apply convolution operations to the input image, extracting features like edges and textures. The diagram shows several CONV layers processing the input image (e.g., a cat) step-by-step.
- **FC (Fully Connected Layers)**: These layers connect every neuron from the previous layer to every neuron in the next, used for final classification. The diagram shows FC layers outputting probabilities for categories like "CAT," "DOG," "HORSE," and "FLOWER."
- **CAT Operation**: The "CAT" label indicates concatenation, where outputs from different layers are combined.



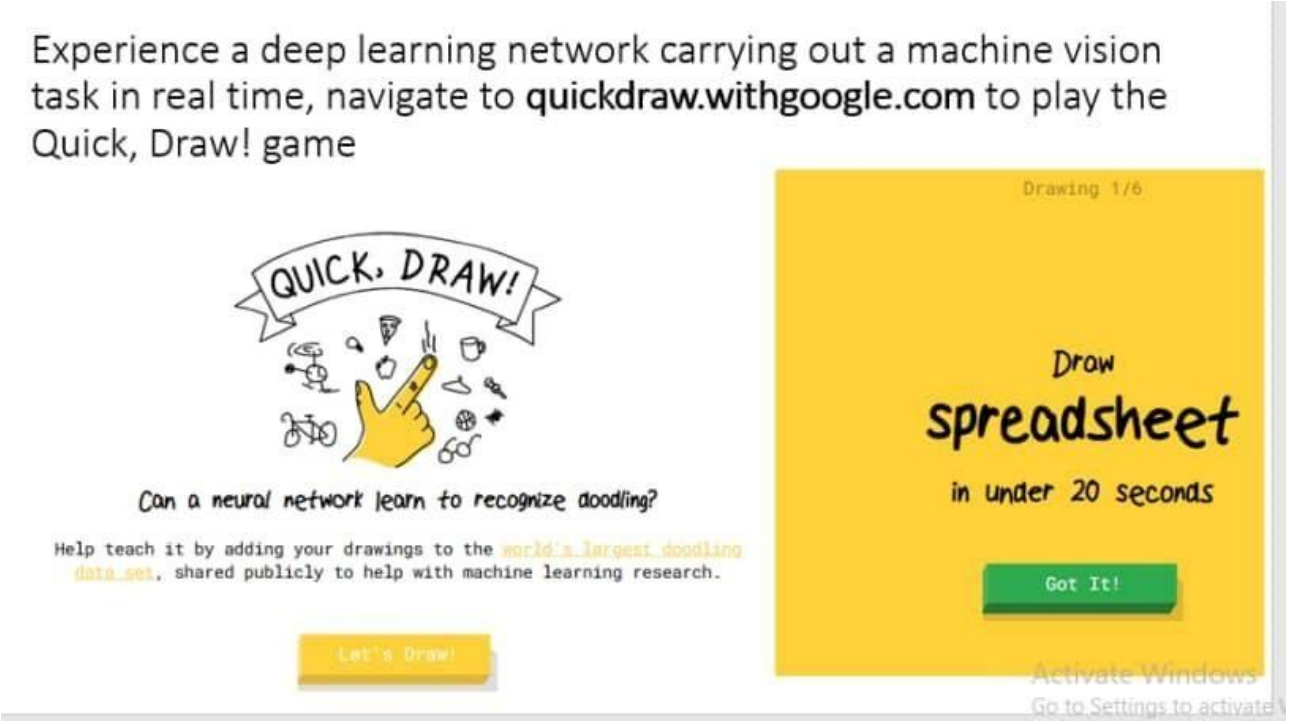
TENSORFLOW PLAYGROUND:

Build Own ANN With Tensor

- TensorFlow Playground is a visual, interactive web tool that helps you learn and experiment with neural networks — without writing any code!
- The network's goal is to learn how to distinguish orange dots (negative cases) from blue dots (positive cases) based solely on their location on the grid.
- In the input layer, we are only feeding in two pieces of information about each dot: its horizontal position (X1) and its vertical position (X2).
- http://bit.ly/TFplayground



Experience a deep learning network carrying out a machine vision task in real time, navigate to **quickdraw.withgoogle.com** to play the Quick, Draw! game



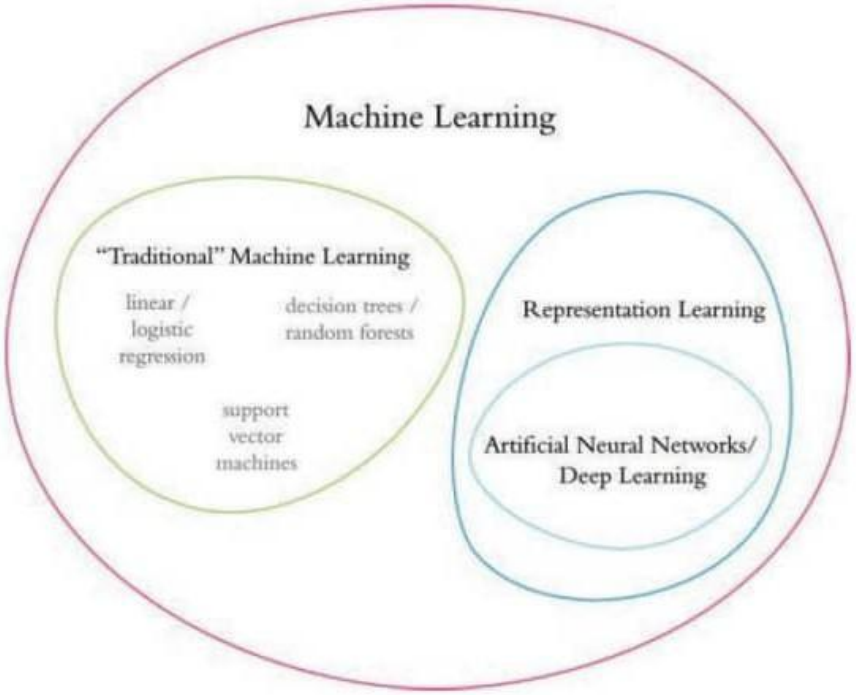
Human and Machine Language:

Deep Learning Networks Learn Representations Automatically:

- Deep Learning is a branch of machine learning where we stack many simple algorithms, called artificial neurons, into multi-layer networks.
- These layers learn to process data step by step, with each layer extracting more complex features from the previous one.

In the broader context:

- Representation Learning is a family of techniques that can automatically learn features (representations) from data, without manual feature engineering.
- Deep Learning is a subset of representation learning and currently the most widely used and powerful approach within it.
- In this context, the terms “feature” and “representation” mean the same thing.

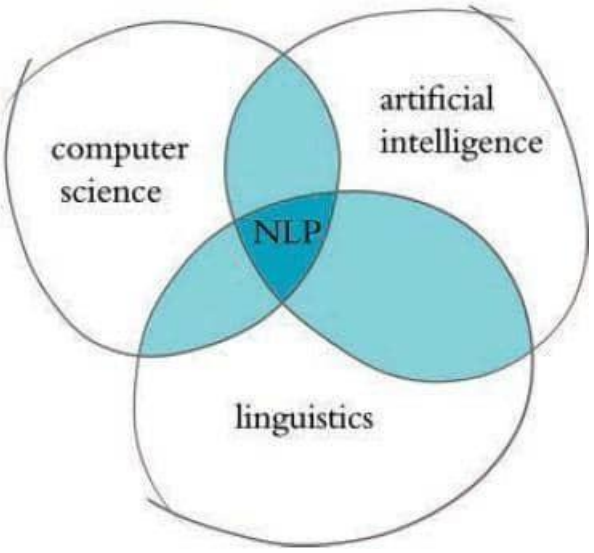


Comparison of Machine Learning and Deep Learning

Machine Learning	Deep Learning
Apply statistical algorithms to learn the hidden patterns and relationships in the dataset.	Uses artificial neural network architecture to learn the hidden patterns and relationships in the dataset.
Can work on the smaller amount of dataset	Requires the larger volume of dataset compared to machine learning
Better for the low-label task.	Better for complex task like image processing, natural language processing, etc.
Takes less time to train the model.	Takes more time to train the model.
A model is created by relevant features which are manually extracted from images to detect an object in the image.	Relevant features are automatically extracted from images. It is an end-to end learning process.
Less complex and easy to interpret the result.	More complex, it works like the black box Interpretations of the result are not easy.
It can work on the CPU or requires less Computing power as compared to deep learning.	It requires a high-performance computer with GPU.

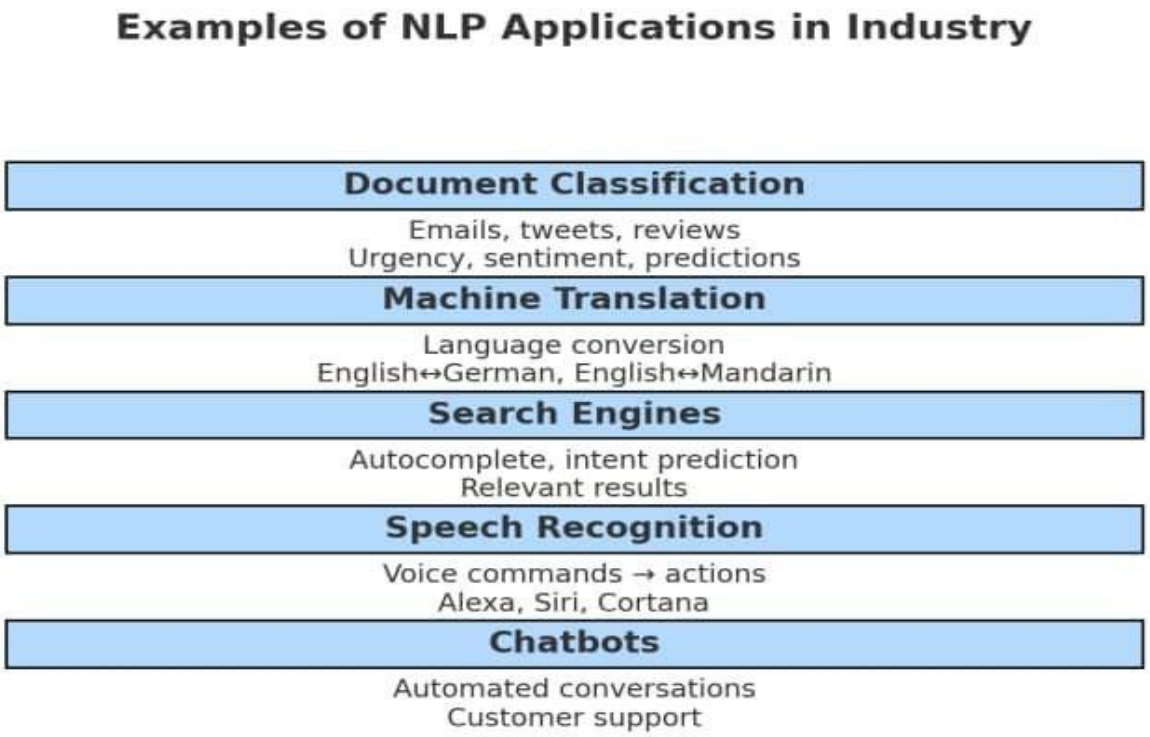
Natural Language Processing:

- **Natural Language Processing (NLP)** combines **computer science**, **linguistics**, and **AI** to enable machines to understand and process human language (spoken or written).
- Its goal is to enable machines to understand and process human languages—both spoken and written—for specific tasks.



Examples of NLP Applications in Industry:

- **Document Classification** – Categorizing text like emails, tweets, or reviews (e.g., urgency level, sentiment, market predictions).
- **Machine Translation** – Translating text between languages (e.g., English to German) with machine assistance or automation.
- **Search Engines** – Autocompleting queries, predicting user intent, and retrieving relevant results.
- **Speech Recognition** – Understanding spoken commands (e.g., Alexa, Siri, Cortana).
- **Chatbots** – Conducting conversations, often for customer service tasks, though usually limited in complexity.



Computational Representations of Language:

- In order for deep learning models to process language, we have to supply that language to the model in a way that it can digest(we must convert words into numbers)
- Two common ways to do this are:
    - **one-hot encoding** and
    - **word vectors(bag of words and skip-Gram)**

One-Hot Representations of Words:

- One-Hot Encoding is a traditional way to turn words into numbers for computers(MATRIX format).

- Imagine you have a list of all unique words in your dataset (called the *vocabulary*).
- Each word gets its own row in a matrix.
- Each column represents a word position in the sentence or document.

Example: If your vocabulary has 100 unique words, the matrix will have 100 rows. If you have 1,000 words, it will have 1,000 rows.

- Inside this matrix, each cell can only be 0 or 1.
- A 1 means "this word is present at this position."
- A 0 means "this word is not present at this position."
- Because most of the cells are 0, this matrix is called sparse.

The bat sat on the cat.

words						
the	1	0	0	0	1	0
bat	0	1	0	0	0	0
on	0	0	0	1	0	0
⋮						
n_{unique_words}						

The bat sat on the cat and dog sat on mat

	the	bat	sat	on	cat	and	dog	mat
the	1	0	0	0	0	0	0	0
bat	0	1	0	0	0	0	0	0
sat	0	0	1	0	0	0	0	0
on	0	0	0	1	0	0	0	0
the	1	0	0	0	0	0	0	0
cat	0	0	0	0	1	0	0	0
and	0	0	0	0	0	1	0	0
dog	0	0	0	0	0	0	1	0
sat	0	0	1	0	0	0	0	0
on	0	0	0	1	0	0	0	0
mat	0	0	0	0	0	0	0	1

Word Vectors:

- A word vector is an attempt to mathematically represent the meaning of a word.
- A computer goes through the text and calculates how often words show up next to each other.
- Older NLP methods like:
  - One-hot encoding – representing each word as a binary vector (1 for presence, 0 for absence).
  - Bag-of-words – counting how many times each word appears in a sentence/document.
- These methods only know if a word is there or not (or how often it appears).
- They do not understand what the word means or how it relates to other words.

Why is this a problem?

- Example:
  - "dog" and "puppy" are related in meaning.
  - In one-hot or bag-of-words, they are stored as completely different and unrelated entries.
  - The model cannot guess that "dog" and "puppy" are similar animals.
- Example:
  - "Apple" (fruit) vs "Apple" (company) — traditional methods treat them as the same word, ignoring context.

- One-hot encoding & Bag-of-words- Simple presence/absence indicators.
- Limitation - No understanding of meaning or context.
- This is why modern NLP uses word embeddings (word vectors or vector-space embeddings) — they capture meaning, similarity, and relationships between words.

How Word Vectors works?

Goal: Assign each word in a dataset (corpus) to a specific, meaningful location in a multidimensional space called vector space.

Initial Step: Words are placed randomly in the vector space at the start.

Learning Process: The system observes which words often appear near each other in text.

Adjustment: Words used in similar contexts are moved closer together in the space.

Outcome: Words with similar meanings are close to each other; words with different meanings are far apart.

Example:

Like arranging guests at a wedding — start random, then group by observed relationships ÿ

- Related words are close in vector space (e.g., wheel, engine near car).
- Unrelated words are far apart (e.g., banana far from car).
- Closeness is based on context in which words are used.
- Word vectors support math operations to capture meaning changes.

Example: king - man + woman = queen shows how meaning shifts can be computed.

- Step 1: Take the meaning of king and remove “maleness” (- man).
- Step 2: Add “femaleness” (+ woman).
- Step 3: The result points to queen in the vector space

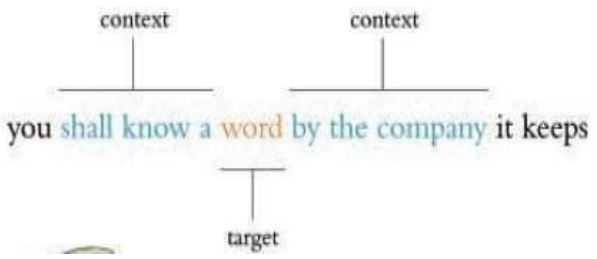
Where Do Word Vectors Come From?

There are two common ways through which word vectors are generated:

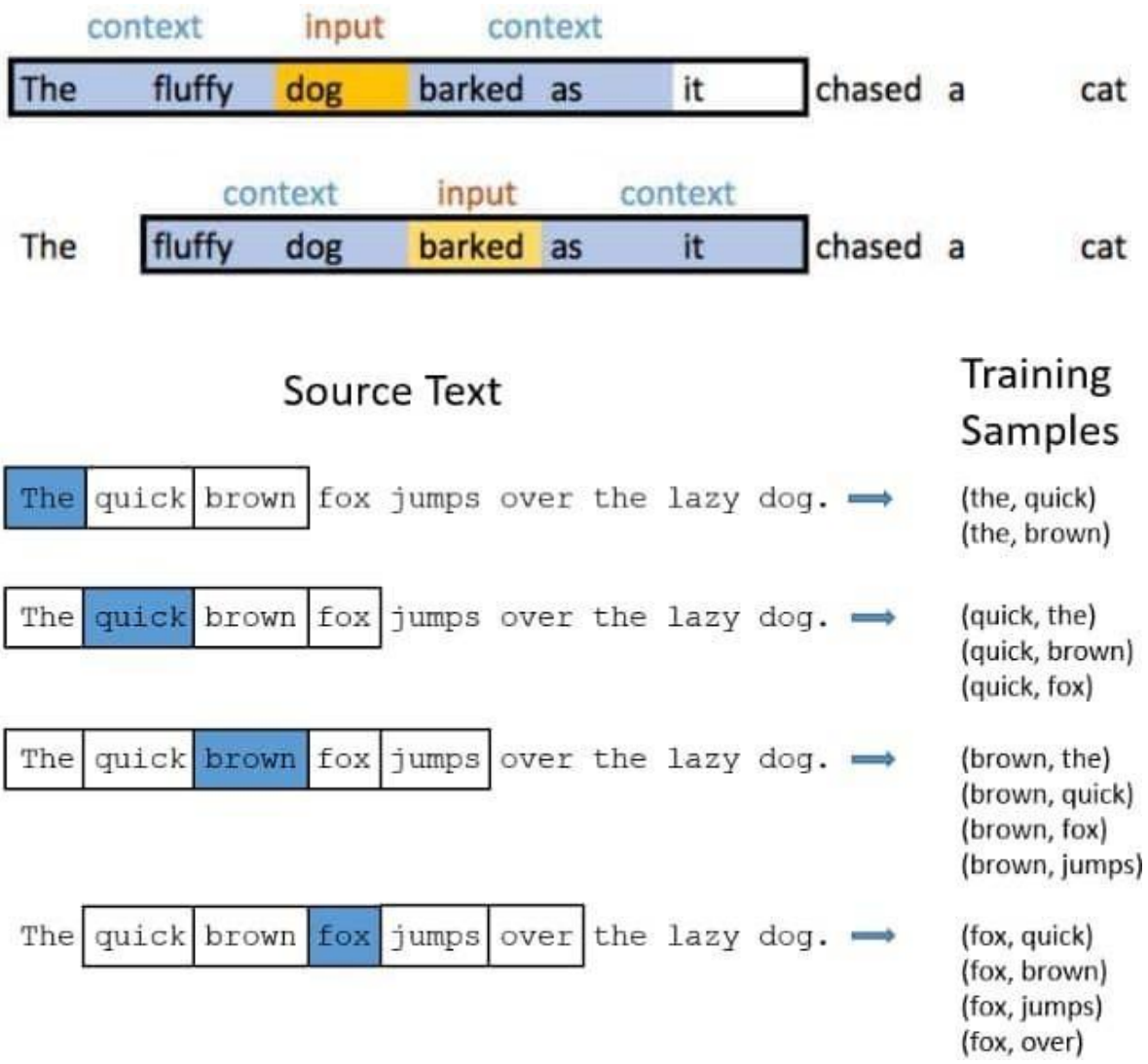
- Counts of word/context co-occurrences
- Predictions of context given word (skip-gram neural network models, i.e. word2vec)
- Both approaches to generating word vectors build on Firth's (1957) distributional hypothesis which states:

"You shall know a word by the company it keeps."

- For each target word in turn, we consider it relative to the words around it—its *context words*.
- In our example, we’re using a context-word window size of three words.



For example, given the input sequence, "The fluffy dog barked as it chased a cat," the two-window (two-words preceding and proceeding the focal word) context for the words "dog" and "barked" would look like:



The network is going to learn the statistics from the number of times each pairing shows up. So, for example, the network is probably going to get many more training samples of (“Soviet”, “Union”) than it is of (“Soviet”, “Sasquatch”). When the training is finished, if you give it the word

"Soviet" as input, then it will output a much higher probability for "Union" or "Russia" than it will for "Sasquatch".

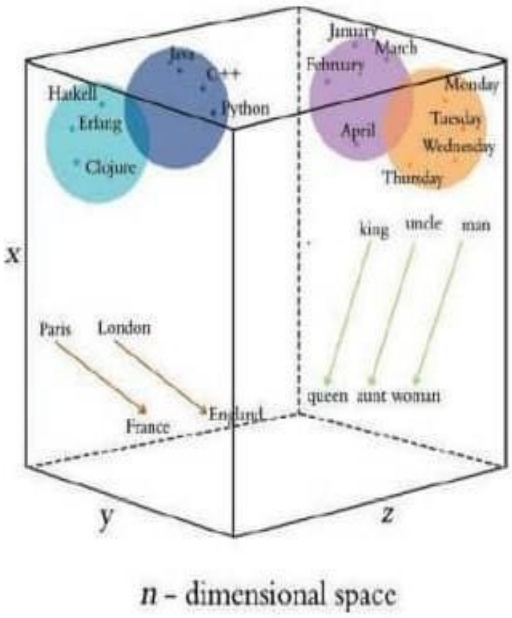


Figure 2.6 Diagram of word meaning as represented by a three-dimensional vector space

One-Hot Representation:

Each word in the vocabulary is represented by a binary vector.
The vector length = size of vocabulary (say 10,000 words).
Only one position is 1 (for that word), rest are 0.
Example
Vocabulary: {cat, dog, apple, run}
"cat" → [1, 0, 0, 0]
"dog" → [0, 1, 0, 0]
"apple" → [0, 0, 1, 0]
"run" → [0, 0, 0, 1]

Advantages:
Simple to create.
Unique IDs for words.

Limitations:
Very high dimensional for large vocabularies.
No meaning captured → "cat" and "dog" are as far apart as "cat" and "run".
Sparse vectors (lots of zeros).

Word Vectors (Embeddings):

Represent each word as a dense vector of real numbers (e.g., 50, 100, or 300 dimensions).
These vectors are learned from text, so they capture semantic meaning.

Example (in 3D for simplicity)
"cat" → [0.6, 0.2, 0.1]
"dog" → [0.7, 0.1, 0.2]
"apple" → [0.1, 0.8, 0.3]
"run" → [0.2, 0.1, 0.9]
Here:
"cat" and "dog" vectors are close in space → similar meaning.
"apple" is far → belongs to fruits.
"run" is closer to action verbs.

Advantages:
Compact & dense → much fewer dimensions than one-hot.
Semantic similarity → words with similar meanings have closer vectors.
Supports analogies → vector math works (e.g., king – man + woman ≈ queen).

Limitation:
Requires training on large text data.

Feature	One-Hot	Word Vectors
Size	Vocabulary size (huge, sparse)	Small (50–300 dims, dense)
Meaning	No semantic info	Captures meaning & context
Similarity	All words equally distant	Similar words closer in vector space
Memory use	Inefficient (lots of zeros)	Efficient, compact
Use case	Simple models, lookup	Modern NLP (translation, chatbots, search, etc.)

Localist Versus Distributed Representations

Attribute	One-Hot Representations	Word Vectors
Representation Style	Localist; stores word info discretely in a single row of a sparse matrix as binary flags.	Distributed; spreads word meaning across n-dimensional space, smeared gradually in a continuous, quantitative space.
Nuance(how similar/relationship b/w words)	Lacks nuance; limited to simple binary flags.	Highly nuanced; captures infinite possibilities for word relationships in high-dimensional space.
Resource Dependency	Requires labor-intensive, manually curated taxonomies (e.g., dictionaries, databases).	Fully automatic; derived from natural language data alone, no external references needed.
Handling New Words	Poor; requires a new matrix row, corpus reanalysis, code changes, and possibly external data, disrupting the model.	Seamless; incorporates new words via training on contextual data, assigning a new vector that refines over time without affecting existing vectors.
Meaning Interpretation	Subjective; relies on developer-designed rules or databases, introducing bias.	Data-driven; meaning derived from training corpus.
Word Similarity	Ignores similarity; treats "couch" and "sofa" the same as "couch" and "cat".	Innately handles similarity; closer vectors indicate more similar words (e.g., "couch" and "sofa").

Activate Windows
Go to Settings to acti

Elements of Natural Human Language:

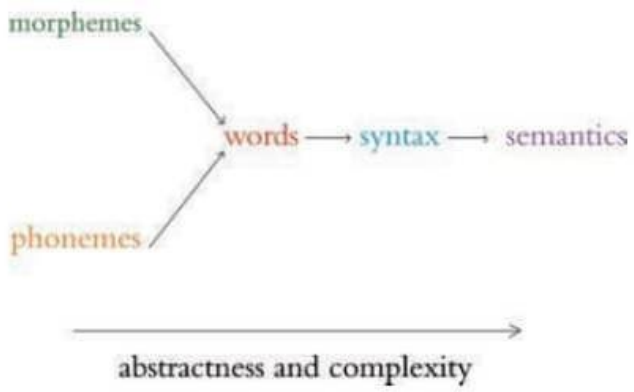


Figure 2.9 Relationships between the elements of natural human language. The leftmost elements are building blocks for further-right elements. As we move to the right, the more abstract the elements become, and therefore the more complex they are to model within an NLP application.

Phonology
Concerned with language sounds. Traditional ML encodes specific phonemes; deep learning represents phonemes in vector space.

Morphology
Concerned with word forms. Traditional ML identifies morphemes from a list; deep learning predicts morpheme occurrence with vectors.

Words
Traditional ML uses one-hot encoding; deep learning uses word vectors that capture meaning relationships.

Syntax
Traditional ML uses formal linguistic categories; deep learning represents phrases with vectors.

Semantics
The most abstract element, concerned with meaning. Deep learning uses vectors to represent logical expressions and semantic relationships.

Table 2.2 **Traditional machine learning and deep learning representations, by natural language element**

Representation	Traditional ML	Deep Learning	Audio-Only
Phonology	All phonemes	Vectors	True
Morphology	All morphemes	Vectors	False
Words	One-hot encoding	Vectors	False
Syntax	Phrase rules	Vectors	False
Semantics	Lambda calculus	Vectors	False

Language Element	Traditional ML Representation	Deep Learning Representation
Phonology (Sounds)	Encodes auditory input as discrete phonemes from a language-specific set, requiring manual identification.	Trains models to predict phonemes from automatically learned audio features, represented as vectors in space (adaptable to speech data).
Morphology (Word Forms)	Identifies morphemes (e.g., "out," "go," "ing" in "outgoing") from a predefined list, rigid and local.	Predicts morpheme occurrences, combining multiple vectors (e.g., for "out," "go," "ing") into a single word vector using neural layers.
Words	Relies on one-hot representations, local and inflexible, with limitations in nuance and similarity (as detailed previously).	Uses distributed word vectors, functional, efficient, and accurate, leveraging tokenization and extensive research (Chapter 11 focus).
Syntax (Sentence Structure)	Buckets phrases into discrete, formal linguistic categories, lacking fluidity.	Represents words and phrases as vectors, with neural layers combining them into phrase vectors in n-dimensional space.
Semantics (Meaning)	Limited by rigid representations, struggles with context and "fuzziness" (e.g., sarcasm, cultural nuances).	Uses vectors for words, phrases, and logical expressions, recombining them nonlinearly via neural layers to calculate semantic vectors based on context.
Flexibility	Rigid, local, and manually curated, requiring redefinition for new data.	Fluid, flexible, and data-driven, adapting to new data through training.

Activate Windov
Go to Settings to acti