The Brain of AI: Exploring the World of Neural Networks

How artificial neurons power today's intelligent technologies.

Misbah Tasleem NS
5th Sem CSE(AI & ML)

Under the guidance of

Dr. Poorna Chandra (Mentor) Dr. Sachin Kumar (HOD)

Dept. of CSE(AI & ML)
Jain College of Engineering, Belagavi

December 12, 2025

Evolution of Neural Networks (Part 1)

1. What is a Neural Network?

A system that takes inputs (images, text) and produces useful outputs (e.g., recognition, translation).

2. First Wave (1940s–1960s): Early Neural Models

- McCulloch & Pitts (1943) created the first artificial neuron.
- Rosenblatt (1957) introduced the perceptron, the first learnable model.
- Perceptrons couldn't solve complex problems, leading to the first AI winter (Minsky & Papert, 1969).

Evolution of Neural Networks (Part 2) I

3. Second Wave (1980s–1990s): Backpropagation & CNNs

- Backpropagation (1986) enabled learning in multilayer networks.
- LeNet (1989) by LeCun et al. recognized handwritten ZIP codes.
- Neural networks lost popularity due to limited computing power and stronger traditional ML methods.

Evolution of Neural Networks (Part 2) II

4. Third Wave (2000s–Present): Deep Learning Era

- Improved algorithms, large datasets, and GPUs fueled renewed interest.
- AlexNet (2012) popularized deep learning by winning ImageNet.
- Neural networks now dominate across many applications and industries.

Biological Neuron

- A biological neuron has:
  - Dendrites – receive input signals.
  - Cell body – processes inputs.
  - Axon – sends outputs.
  - Synapses – connect neurons.

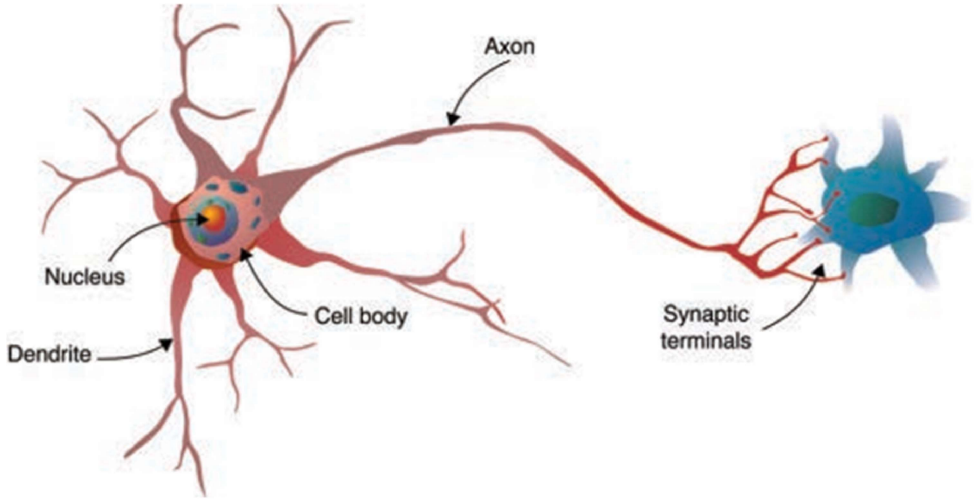


Figure: Structure of a Biological Neuron

Artificial Neuron (Perceptron)

- The perceptron is inspired by how a biological neuron works.
- It performs:
  - Input signals ($x_1, x_2, \ldots$) — features.
  - Weights ($w_1, w_2, \ldots$) — importance of inputs.
  - Summation — adds weighted inputs.
  - Bias (b) — adjusts sensitivity.
  - Activation function — decides output.

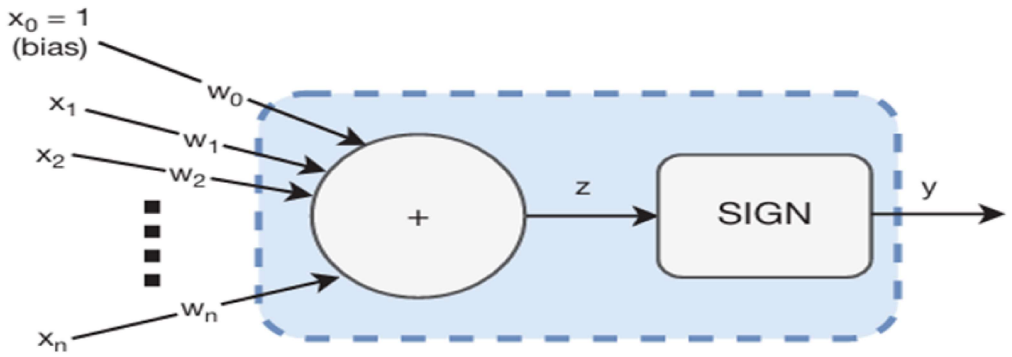


Figure: Structure of a Perceptron

How It Mimics the Biological Neuron

- Dendrites: Inputs ($x_1, x_2, \ldots$)
- Synaptic Strength: Weights ($w_1, w_2, \ldots$)
- Cell Body (Summing signals): Summation Function ($\sum x_i w_i$)
- Axon Hillock (Decision point): Activation Function (Sign/Step function)
- Axon (Signal path): Output (y)
- Threshold Potential: Bias (b)

Example: Perceptron with 2 inputs

For weights $w_0 = 0.9$, $w_1 = -0.6$, and $w_2 = -0.5$:

When tested with all combinations of inputs $(-1, 1)$, the perceptron behaves like a NAND gate. This means perceptrons can simulate basic Boolean logic.

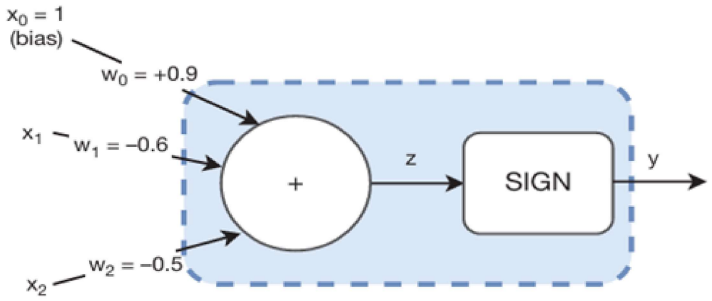


Figure 1-4 Perceptron with two inputs (in addition to the bias input) and defined weights

Figure: Structure of a 2-input Perceptron

Perceptron Learning Algorithm

The perceptron can learn correct weights automatically through supervised learning:

- Initialize weights randomly.
- Pick an input–output pair.
- Compute the perceptron's output.
- If it's wrong, update each weight using:

$$w_i = w_i + (\text{learning\_rate}) \times y \times x_i$$

- Repeat until all outputs are correct.

This process adjusts the weights gradually, allowing the perceptron to learn patterns.

Limitations

The perceptron can only classify linearly separable data (data that can be divided by a straight line). For example:
• Works for NAND or AND functions
• Fails for XOR, since XOR cannot be separated by a single straight line.

Overcoming the Limitation:

- • Combine multiple perceptrons in layers.
- • The first layer (hidden layer) separates parts of data.
- • The second layer (output neuron) combines them.

This forms a multi-layer neural network (basis of modern deep learning).

Journey: From Perceptron to Modern CNNs I

- **1. Single Neuron (Perceptron) — 1957**
  - Takes inputs, multiplies by weights, adds bias, passes through activation.
  - Can only solve linearly separable problems (simple yes/no).
- **2. Multi-Neuron (MLP) — 1980s**
  - Multiple neurons connected in layers; uses backpropagation to learn.
  - Not efficient for images; huge number of weights if image is large.
- **3. Local Connectivity — Late 1970s–1980s**
  - Each neuron connects only to a small patch of the input (like human vision).
  - Reduces computation and focuses on local patterns.
- **4. Neocognitron — 1980 (Fukushima)**

Journey: From Perceptron to Modern CNNs II

- Introduced convolution-like layers to detect patterns.
- Added pooling-like layers to reduce size and retain important info.

- **5. LeNet — 1989–1998**
  - First practical CNN combining convolution, pooling, and fully connected layers.
  - Used for handwritten digit recognition (ZIP codes, cheques).
- **6. Modern CNNs — 2012 Onwards**
  - Deep networks with GPUs and large datasets (AlexNet, VGG, ResNet).
  - Can learn complex features efficiently and power today's computer vision.

CNN in Image Recognition
1. Intuition & Overview

- A CNN (Convolutional Neural Network) processes images by learning features (like edges, shapes, textures) in successive layers.
- Early layers detect simple patterns (edges, corners); deeper layers combine those into more complex features (ears, face, tail) and finally the network decides what the object is.
- The advantage: CNNs learn where and what in the image is important automatically, using filters/kernels, pooling, and fully connected layers.

2. Example Setup

Let's say our dataset is small and we want to classify images into two classes: cat or dog.
Each image is 64 × 64 pixels, with 3 color channels (RGB). So the input shape is 64 × 64 × 3.

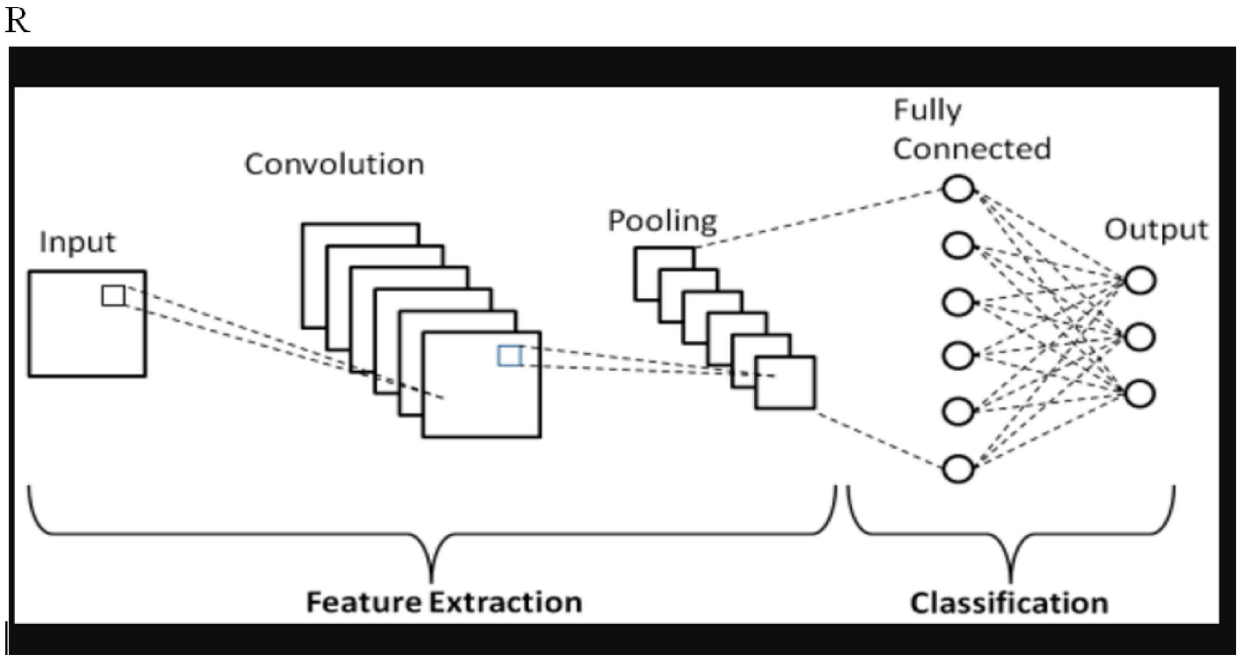
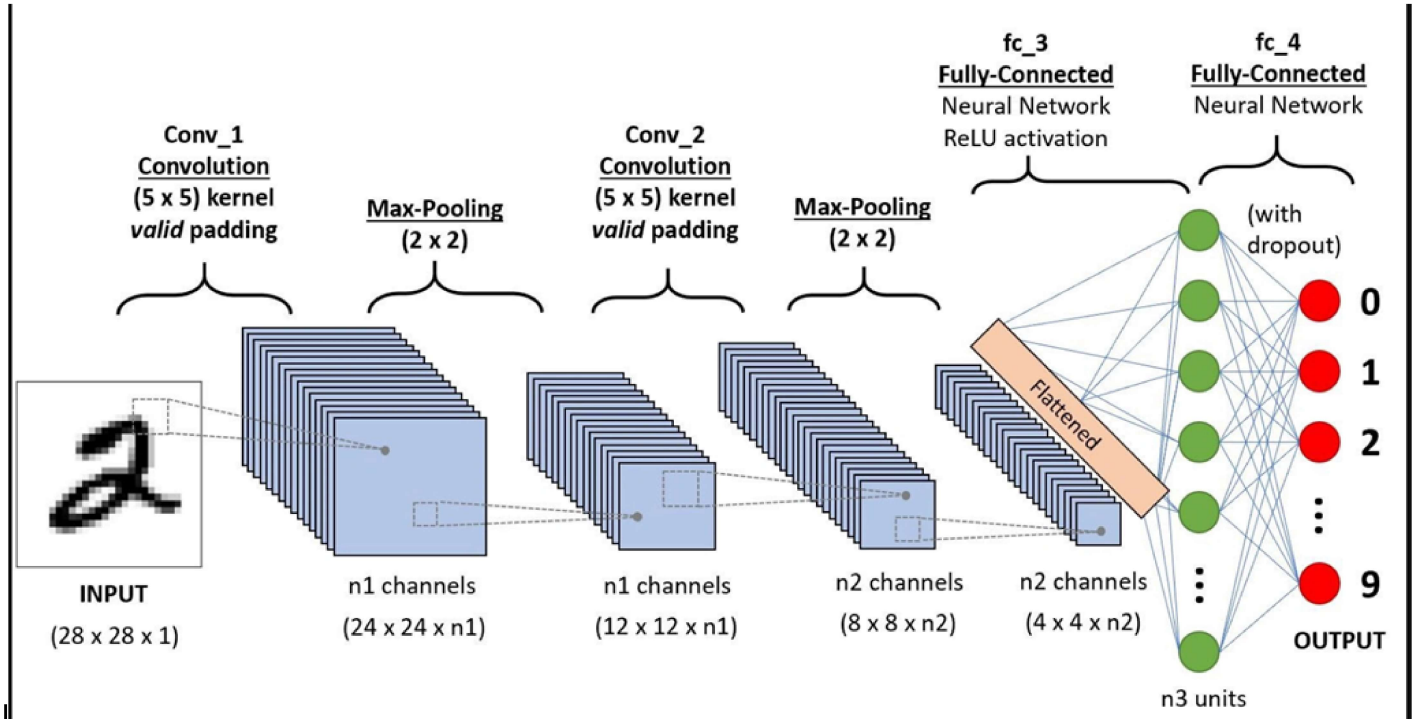


Figure: Overview of a Convolutional Neural Network

3. Layer-by-Layer Walkthrough

Here's a simple CNN architecture we use as our example:

- Input layer
- Convolutional layer + ReLU
- Max Pooling
- (Possibly more conv + pooling layers)
- Flatten layer
- Fully connected (Dense) layer(s)
- Output layer (softmax or sigmoid)



A. Input Layer

Receives the raw pixels: $64 \times 64 \times 3$
No weights here — it's just the data.

B. Convolutional Layer

- The convolution layer uses small filters (kernels) like 3×3 or 5×5.
- These filters slide over the image from left to right, top to bottom.
- At every position, the filter and the image region are multiplied and added $\rightarrow$ gives one value.
- Doing this for the whole image forms a feature map.
- These feature maps highlight important patterns such as:
  vertical edges,horizontal edges,corner,textures
- Convolution helps CNN understand the image locally (small region at a time).

C. Activation Function (ReLU)

- After convolution, you apply ReLU (Rectified Linear Unit):
- It keeps positive values as they are and makes all negative values = 0.

$$\text{ReLU}(x) = \max(0, x)$$

- This introduces non-linearity, making the model more expressive (can capture more complex patterns).

D. Pooling Layer (Max Pooling)

- Pooling reduces the size of the feature maps.
- Max Pooling takes the maximum value from each small region (e.g., 2×2 window).
- Helps keep the important features while removing unnecessary details.
- Makes the model faster and reduces computation.
- Provides spatial invariance (small changes in image don't affect output).

E. Additional Conv + Pooling Layers (optional)

- Adding more conv and pool layers makes the CNN deeper, so early layers learn simple things (edges), and later layers learn complex things (shapes and objects).
- More layers can improve accuracy, but they also take more time to train and more computation power .

F. Flatten Layer

- After convolution and pooling, the output is still in 2D or 3D form (like 7×7×64). Flatten layer converts this into a single long 1-dimensional vector.
- This step is needed because the next part of the model (fully connected layers) can only take 1D input.
- Example:
  If the feature map size is $7 \times 7 \times 64$, flattening it gives 3136 values in one row.

G. Fully Connected (Dense) Layer(s)

This is like a classic neural network layer.
Suppose we have one dense layer of 128 neurons. Each neuron connects to all flattened features.
Activation (e.g. ReLU) is applied again.
What it does: Combines all features to find higher-level decision boundaries.

H. Output Layer

- The output layer gives the final prediction of the CNN. We use Softmax or sigmoid function to convert the outputs into probabilities for each class.
- If we use Softmax
  We need 2 neurons (one for Cat, one for Dog).
  Softmax gives two probabilities that add up to 1.
- Since we classify Cat vs Dog, the model gives two probabilities, and the higher one tells whether the image is a Cat or a Dog.
- If we use Sigmoid
  We need 1 neuron.
  Sigmoid gives one probability (example: greater than 0.5 = Dog, less than 0.5 = Cat).

4. How It Works Together (Forward + Backprop)

Forward Pass:
Image $\rightarrow$ conv $\rightarrow$ ReLU $\rightarrow$ pooling $\rightarrow$ (repeat) $\rightarrow$ flatten $\rightarrow$ dense $\rightarrow$ output (probability).
Loss Computation:
Compare output with true label (cat or dog), compute loss (e.g. cross-entropy).
Backpropagation:
Compute gradients of loss w.r.t. all weights (in conv filters, dense layers).
Update weights (filters and dense weights) using gradient descent (or optimizer like Adam).
This trains the network to learn filters that detect features suited to distinguishing cats vs dogs.

5 . What Each Layer Learns

First conv layer: edges, simple textures
Second conv layer: combinations of edges $\rightarrow$ curves, small shapes
Dense layers: combine these shapes to detect parts like ears, face, fur
Output: final decision: cat or dog

What is LeNet?

- LeNet is a neural network (a type of AI model) created by Yann LeCun.
- Its job: look at a picture of a digit and tell which number it is.
- It was used for tasks like reading:
  - ZIP codes on letters
  - Numbers written on bank cheques

How LeNet Sees an Image

Think of recognizing a number the way you scan a page:

- First you look at tiny details (lines, small curves)
- Then you join these small parts to see bigger shapes
- Finally, your brain says: “This looks like a 3 or a 7.”

LeNet works in the same way.

How LeNet is Built (Simple Analogy)

LeNet has layers, like steps in a process. Each layer does something special.

- Step 1: Looking at small parts (Convolution Layer)
- Step 2: Keeping important things (Pooling Layer)
- Step 3: Understanding bigger shapes (More Convolution + Pooling)
- Step 4: Final decision-making (Fully Connected Layers)

Step 1: Convolution Layer

This layer looks at tiny pieces of the image:

- little lines
- edges
- corners

It works like zooming in to find small clues.

Step 2: Pooling Layer

From all the details, it keeps only the strongest or most important ones.

- Similar to taking notes – you only keep key points.
- Pooling reduces the size but keeps meaningful information.

Step 3: Understanding the Bigger Picture

The model now combines small clues to recognize bigger shapes:

- Curves of “3”
- Straight lines of “1”
- Round shapes of “0”

More convolution + pooling improves pattern recognition.

Step 4: Fully Connected Layers

This step works like the brain thinking:

> “Based on all the features, this looks like a 7.”

The final layer gives the answer:

- Which digit (0–9) the image represents

Why LeNet Was Special

- Before LeNet, computers struggled to read handwritten text.
- LeNet learned from examples, just like humans do.
- It didn't need someone to manually design rules.
- LeNet learned its own rules from data.

This idea became the foundation for modern AI systems like:

- Self-driving cars
- Face recognition
- Advanced image processing

LeNet in One Simple Line

Image $\rightarrow$ Find small patterns $\rightarrow$ Pick important patterns $\rightarrow$ Combine them $\rightarrow$ Decide the number

References I

Yann LeCun. *LeNet Convolutional Neural Network*. Available at: `http://yann.lecun.com/exdb/lenet/index.html`

Magnus Ekman. *Learning Deep Learning: Theory and Practice of Neural Networks, Computer Vision, Natural Language Processing, and Transformers Using TensorFlow*. NVIDIA Deep Learning Institute, 2021.