

# 1. INTRODUCTION TO CNN

- **Convolutional Neural Networks (CNNs)** are a special type of neural network designed to work well with **image and spatial data** (data arranged in a grid, like 2D images or even 1D time series).
  - Instead of connecting every input pixel to every neuron (like a fully connected ANN), CNNs use **small filters (kernels)** that scan across the image to detect patterns.
  - This approach allows the network to **automatically learn useful visual features**:
    - Early layers → detect **edges** and simple shapes.
    - Middle layers → detect **textures or parts of objects**.
    - Deeper layers → detect **whole objects or complex patterns**.
  - CNNs are **parameter-efficient** compared to ANN: the same small filter is reused across the whole image (called **weight sharing**).
  - This makes them **faster to train** and better at understanding images than a standard fully connected network.
- 

## 1.1 Summary

CNNs are neural networks specialized for grid-like data (like images) that **learn visual features automatically** and are far more efficient than fully connected ANNs.

## 1.2 Problem with ANN on images

- An image of size **64×64×3 (RGB)** has **12,288 pixels**.
  - A single fully connected layer with just 100 neurons would need  **$12,288 \times 100 = 1.2M$  weights** → huge, slow, prone to overfitting.
  - CNN fixes this by using **small filters** that are reused across the image instead of learning a separate weight per pixel.
- 

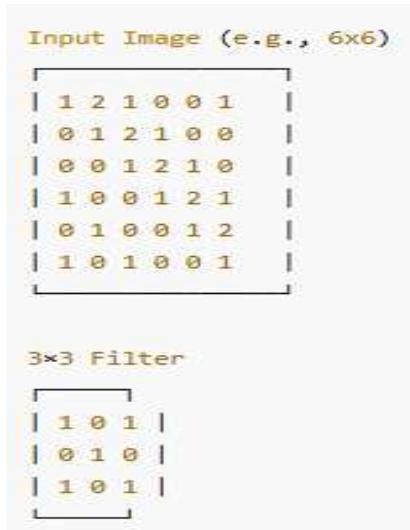
## 1.3 Why CNN (Convolutional Neural Networks)

- Uses **small filters/kernels** (e.g., 3×3, 5×5) that **slide** over the image.
  - The same small set of weights is **reused across the entire image (weight sharing)**.
  - Learns **local spatial patterns** first (edges → shapes → objects).
  - Far **fewer parameters**, trains faster, generalizes better.
-

## 2. HOW CNN WORKS — VISUAL INTUITION

Think of an image as a grid of numbers (pixel values).

A **filter/kernel** is a small grid of weights (e.g., 3x3) that slides (convolves) across the image:



- The filter **slides over** the image, multiplying and summing values → creates a **feature map**.
- One filter may detect **horizontal edges**, another **vertical edges**, etc.
- Stacking many filters helps the network learn **different features automatically**.

### 2.1 CNN Feature Learning Hierarchy

- **Layer 1:** Learns edges (horizontal, vertical, diagonal).
- **Layer 2:** Learns textures, corners.
- **Layer 3+:** Learns shapes and objects (faces, wheels, etc.).

### 2.2 One-liner for interviews:

“CNNs use small filters that slide over an image to detect patterns. Early layers find simple edges; deeper layers combine them into complex shapes and objects — all with far fewer parameters than a fully connected ANN.”

### 2.3 ANN v CNN

ANN	CNN
Fully connected → huge number of weights	Convolution filters → very few weights
Ignores spatial info	Exploits local spatial structure
Easily overfits on images	Generalizes well

### 2.4 One-liner for Interviews

“ANNs treat every pixel independently and explode in parameter count, while CNNs share small filters across the image, preserving spatial structure and using far fewer weights.”

### Step 2 — Move 1 step right

Next 3x3 patch:

```
[  
[2, 3, 0],  
[1, 2, 3],  
[2, 1, 0]  
]
```

Multiply with filter F:

$$(2*1) + (3*0) + (0*-1) + (1*1) + (2*0) + (3*-1) + (2*1) + (1*0) + (0*-1)$$

$$= (2+0+0) + (1+0-3) + (2+0+0)$$

$$= 2$$

So, **Output [0,1] = 2.**

You repeat this sliding process until the filter has scanned the whole 5x5 image.

Since we used:

- **Input = 5x5**
- **Filter = 3x3**
- **Stride = 1**
- **Padding = Valid (no padding)**

👉 The **Output feature map size = 3x3.**

$$\begin{bmatrix} -4 & 2 & 3 \\ 0 & 0 & -2 \\ 0 & 0 & -3 \end{bmatrix}$$

#### ◆ What do negative / zero / positive numbers in the feature map mean?

When a filter slides over the image:

- **Each output number** is the **dot product** (multiplication + sum) between the filter and the small patch of the image.
- It's basically a **score** for "how well this patch matches the pattern" the filter has learned.

Value	Meaning
<b>Positive</b> (large)	Patch matches the filter's pattern strongly.
<b>Near zero</b>	Patch doesn't match the pattern much.
<b>Negative</b>	Patch matches the <b>opposite</b> of the filter's pattern.

#### ◆ Example:

- If the filter learned to detect a **vertical edge** (bright on left, dark on right):
  - Positive → strong vertical edge found.
  - Zero → no vertical edge.
  - Negative → **opposite edge** (dark left, bright right).

#### ◆ Putting it together

- You **choose how many filters** and their size (e.g., 32 filters of size 3x3).
- The **network learns what each filter detects** during training (no need to hand-design).