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.

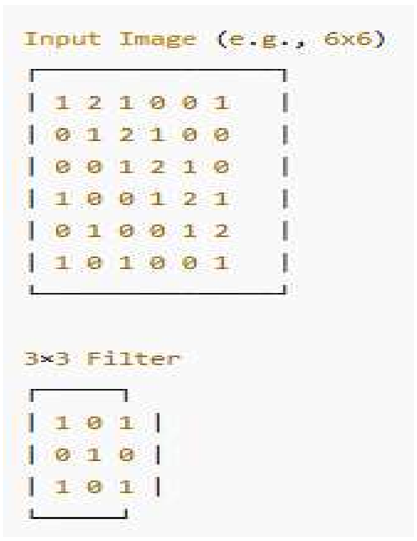
1.2 Problem with ANN on images
- An image of size **64×64×3 (RGB)** has **12,288 pixels**.
- A single fully connected layer with just 100 neurons would need **12,288 × 100 = 1.2M weights** → huge, slow, prone to overfitting.
- CNN fixes this by using **small filters** that are reused across the image 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., 3×3) that slides (convolves) across the image:



```
Input Image (e.g., 6x6)
┌───────────────┐
| 1 2 1 0 0 1   |
| 0 1 2 1 0 0   |
| 0 0 1 2 1 0   |
| 1 0 0 1 2 1   |
| 0 1 0 0 1 2   |
| 1 0 1 0 0 1   |
└───────────────┘

3×3 Filter
┌─────┐
| 1 0 1 |
| 0 1 0 |
| 1 0 1 |
└─────┘
```

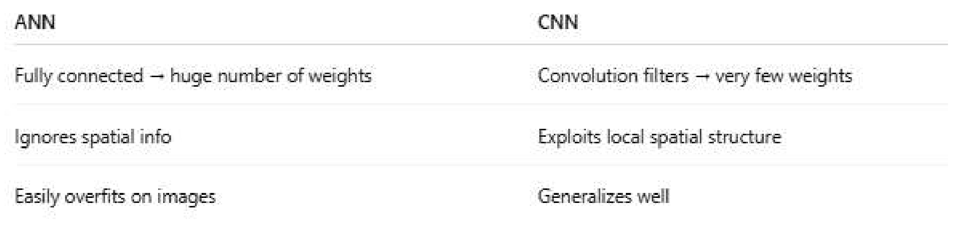
- The filter **slides over** the image, multiplying and summing values → creates a **feature map**.
- One filter may detect **horizontal edges**, another **vertical edges**, etc.
- Stacking many filters helps the network learn **different features automatically**.

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."

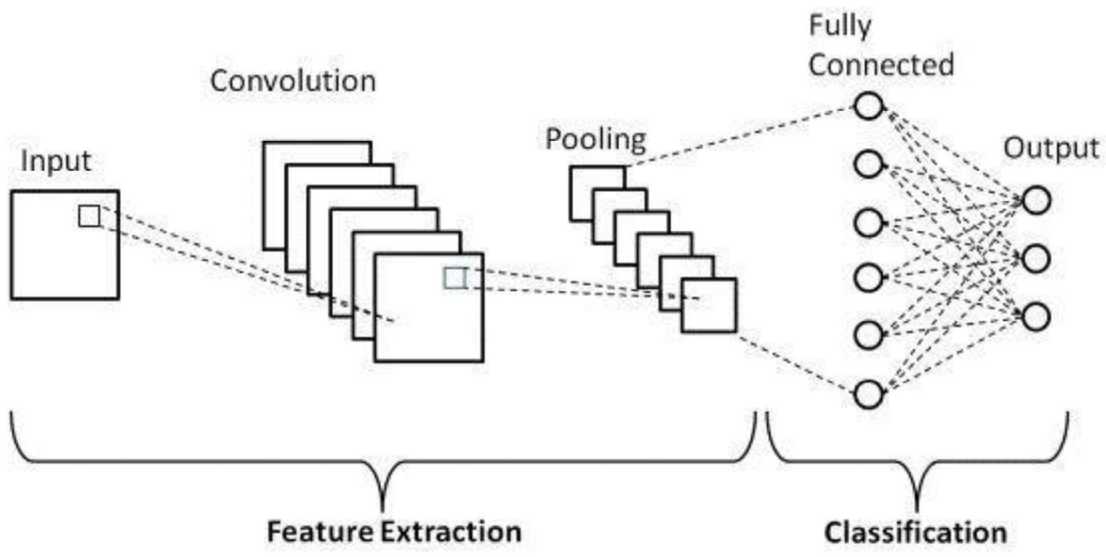
2.3 ANN v CNN



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."

3. CORE BUILDING BLOCKS OF CNN



1. **Input Layer**
   - Accepts the raw image (e.g., 224×224×3 for RGB).
   - Can also take other grid-like data (time series, video frames).
2. **Convolution Layer**
   - Applies **filters/kernels** to extract features.
   - **Filters/Kernels:** small matrices (e.g., 3×3, 5×5).
   - **Stride:** how far the filter moves each step.
   - **Padding:**
     - **Same:** keeps output size same as input.
     - **Valid:** no padding → smaller output.
3. **Activation Layer (ReLU)**
   - Adds **non-linearity** so network can learn complex features.
   - Usually applied after each convolution.
4. **Pooling Layer**
   - Reduces spatial size → fewer computations & parameters.
   - **Max Pooling:** keeps strongest value in each region.
   - **Average Pooling:** keeps average value.
5. **Batch Normalization (often used)**
   - Normalizes activations → stabilizes and speeds up training.
6. **Dropout Layer (optional)**
   - Randomly disables neurons during training → prevents overfitting.
7. **Flatten Layer**
   - Converts 2D feature maps into 1D vector to feed Dense layers.
8. **Fully Connected (Dense) Layer**
   - Combines learned features to make predictions.
   - Often used near the end for classification.
9. **Output Layer**
   - Produces final prediction:
     - **Softmax** for multi-class classification.
     - **Sigmoid** for binary classification.

```
┌──────────────────────────────────────────────┐
│                INPUT LAYER                   │
│         (Image: H × W × Channels, e.g., 224×224×3)│
└──────────────────────────────────────────────┘
                      │
                      ▼
┌───────────────────────────────────────────────┐
│             CONVOLUTION LAYER(S)              │
│ - Filters/Kernels slide over the image        │
│ - Stride = step size                          │
│ - Padding = SAME/VALID                        │
└───────────────────────────────────────────────┘
                      │
                      ▼
┌───────────────────────────────────────────────┐
│             ACTIVATION (ReLU)                 │
│   - Adds non-linearity so complex patterns can be learned │
└───────────────────────────────────────────────┘
                      │
                      ▼
┌───────────────────────────────────────────────┐
│               POOLING LAYER                   │
│   - Max Pooling: keep strongest feature       │
│   - Average Pooling: average values           │
│   - Reduces spatial size & computation        │
└───────────────────────────────────────────────┘
                      │
                      ▼
┌───────────────────────────────────────────────┐
│        (Optional) DROPOUT                     │
│  - Randomly disables neurons to reduce overfitting │
└───────────────────────────────────────────────┘
                      │
                      ▼
┌───────────────────────────────────────────────┐
│               FLATTEN                         │
│  - Converts 2D feature maps → 1D vector       │
└───────────────────────────────────────────────┘
                      │
                      ▼
┌───────────────────────────────────────────────┐
│        FULLY CONNECTED (DENSE) LAYER(S)       │
│   - Combines extracted features for classification │
└───────────────────────────────────────────────┘
                      │
                      ▼
┌───────────────────────────────────────────────┐
│               OUTPUT LAYER                    │
│   - Sigmoid → Binary classification           │
│   - Softmax → Multi-class classification      │
└───────────────────────────────────────────────┘
```