Convolutional Neural Network(CNN)

1. What is the need of CNN?

Problem:

Images are **not normal data** like marks, salary, or age.
Images have:

- **Pixels**
- **Spatial structure** (left, right, top, bottom)
- **Patterns** (edges, shapes, textures)

👉 CNN is designed specially to understand images, not just numbers.

2. What if we use ANN for images? (32 × 32 image)

Step 1: Image as numbers

A 32 × 32 grayscale image = 1024 pixels

If RGB:

= 32 × 32 × 3 □ **3072 values**

So ANN input = a **long vector** # [p1, p2, p3, p4, ... p1024]

Step 2: Problem with ANN

❌ **Problem 1:** Too many weights

Suppose:

- Input = 1024 neurons
- Hidden layer = 100 neurons

Weights = 1024 × 100 = **102,400 weights**

👉 Huge memory
👉 Slow training
👉 Overfitting

❌ **Problem 2:** No spatial understanding

ANN **does not know**:

- Which pixel is next to which
- Which pixels form an edge or shape

Example:

Eye pixels + Nose pixels + Background pixels

ANN treats all equally

👉 ANN loses image structure

❌ **Conclusion for ANN:-**

ANN sees image as: Just a long list of numbers

But image is: Patterns + shapes + spatial relations

3. Why not RNN for images?

What RNN is good at:

- Sequences
- Order matters

Examples:

- Text
- Speech
- Time series

Image is NOT a sequence

4. Architecture of CNN

cat.png

1. **Convert image to matrix:**

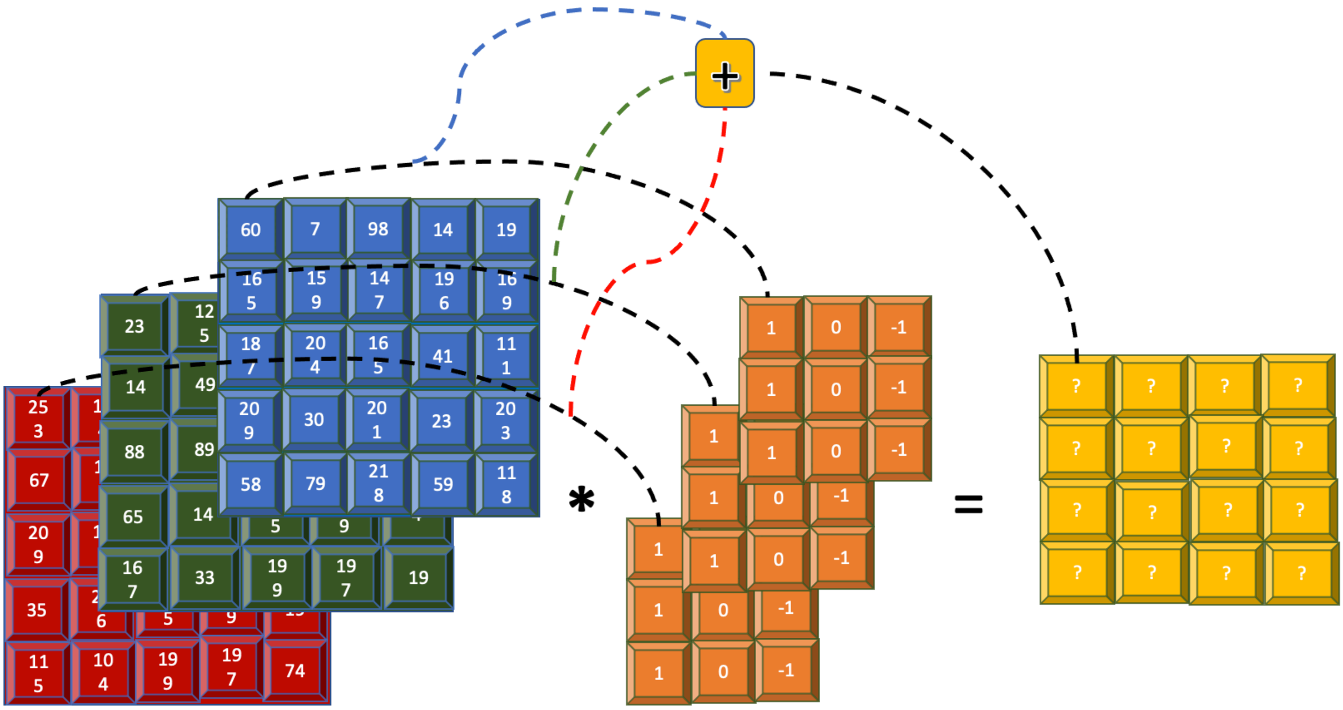
```
import numpy as np
from PIL import Image
# Load image and convert to grayscale
img = Image.open("cat.png").convert("L")   # L = grayscale, RGB = Red, Green, Blue
# Convert image to matrix (0–255)
image_255 = np.array(img)
print("Image Matrix (0-255 range):")
print(image_255)
```

```
Image Matrix (0-255 range):

[[138 134  76 147  76  79 241 149 150 177 127  76 167  76 130 130]
 [163 161 147 140 188   1  60 207 214  76  80 213 143 144 175 201]
 [138 123 160 174 130 255  72 149 151 121 179 152 186 158  96 117]
 [147 137 157 155 165 121 149 150 150 147 157 179 161 156 102 120]
 [141 136 146 127 148 157 158 162 162 158 155 148 130 140  89 103]
 [122 106 137 147 160 170 175 181 181 175 170 159 149 131  49  75]
 [ 70  27 137 166 181 188 185 189 189 185 187 181 166 134   0  31]
 [ 30 148 158 179 201 207 201 192 192 200 207 200 179 158 150   0]
 [148 149 163 180 141 121 145 191 190 154 119 139 180 163 149 148]
 [151 150 164 189 154 106  72 190 184  65 107 153 188 163 150 151]
 [146 149 161 185 198 165 146 202 202 141 166 198 184 160 149 148]
 [148 124 151 179 183 212 170  58  57 170 211 184 179 151 124 148]
 [177  10 145 155 175 233 236 144 144 235 231 175 155 145  11 177]
 [224 117  25 101 148 209 222 238 238 223 209 148 101  27 117 239]
 [109 104 117  29 192 106 192 213 212 192 105 191 226 117 104 109]
[108 131 168 108 123 131 191 191 191 191 130 133  29 167 129 108]]
```

2. Convert matrix to image:

```
# Create image from matrix
img = Image.fromarray(image_255, mode="L")
# Save image
img.save("generated_rgb_image.png")
# Display image
img.show()
```



3. **Element wise Matrix Multiplication:-**

$$\mathbf{Mat(A)} = \begin{bmatrix} 1, & 2, & 3 \\ 4, & 5, & 6 \\ 7, & 8, & 9 \end{bmatrix} \qquad \mathbf{Mat(B)} = \begin{bmatrix} 0, & 1, & 0 \\ 1, & 0, & 1 \\ 0, & 1, & 0 \end{bmatrix}$$

Multiplication:-

$$\begin{matrix} (1\times0) & (2\times1) & (3\times0) \\ (4\times1) & (5\times0) & (6\times1) \\ (7\times0) & (8\times1) & (9\times0) \end{matrix} = \begin{bmatrix} 0 & 2 & 0 \\ 4 & 0 & 6 \\ 0 & 8 & 0 \end{bmatrix}$$

4. **Working flow:-**

Image → (Padding + Filter) → Convolution → Feature Map → Pooling → Flatten → Dense → Softmax

Padding → Filter/Kernel → Stride

First we decide **what to detect** (filter), then **prepare the image** (padding), then **how to move** (stride), and finally perform convolution.

In CNN, **filter values are not given manually**; they are **randomly initialized** and learned automatically during training using backpropagation.

Kernel:

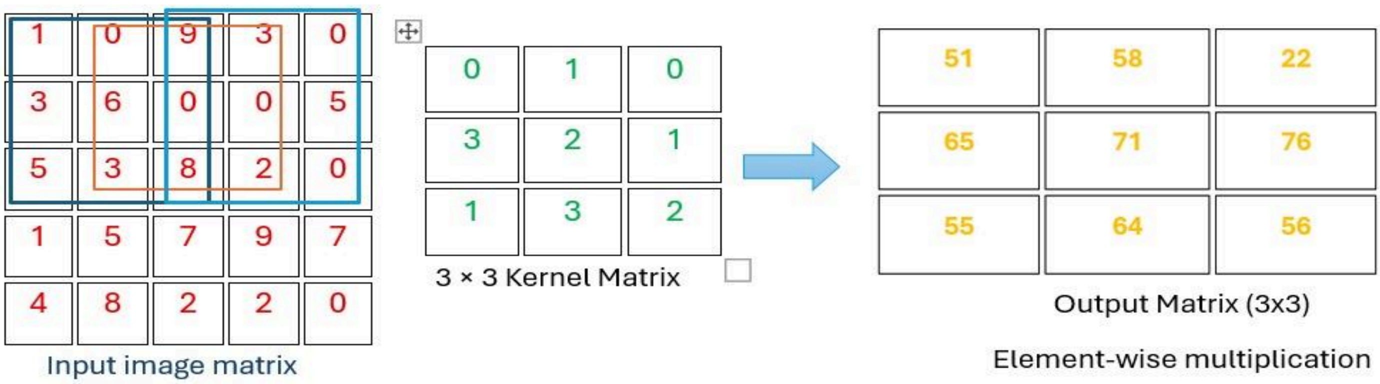
- It is a small 2D matrix of weights that slides over one input channel to extract features.
- They help detect **patterns** like edges, textures, corners, and shapes.
- And these values are also randomly initialized.
- Without kernels, CNNs wouldn't be able to automatically learn these features.

Example:-

$$\mathbf{Kernel} = \begin{bmatrix} 1 & 0 & -1 \\ 1 & 0 & -1 \\ 1 & 0 & -1 \end{bmatrix}$$

Filter:

A filter is a collection (stack) of kernels, one for each input channel, used to generate one feature map.



Output matrix size = (Image mat. size – Kernel mat. size) + 1

Stride: (Stride = Step Size)

Stride defines how many pixels the kernel moves during convolution and is used to control feature map size, computation cost, and downsampling.

Types of Stride:-

A. Stride = 1 (Default)

- Kernel moves 1 pixel
- Maximum feature extraction
- Large feature map

B. Stride = 2

- Kernel jumps **2 pixels**
- Smaller output
- Faster computation

Output = $\lfloor (N - F) / S \rfloor + 1$

Where:

N = input size

F = kernel size

S = stride

Output = $\lfloor (5 - 3)/2 \rfloor + 1 = 2$

C. Stride > 2

- Heavy downsampling
- Risk of losing information

Issue without padding?

Given:

Input: **5 × 5**

Kernel: **3 × 3**

Stride: **1**

Padding: **0 (VALID)**

Output: **(5 − 3) + 1 = 3 → 3 × 3 feature map**

❌ **Problems:**

- **Loss of edge information**
  a) Corner & border pixels are used only once
  b) Center pixels are used many times
- **Deep CNN issue**

  After many layers → image becomes too small

👉 This is why padding is introduced.

Padding: It adds extra pixels (usually zeros) around the image border before convolution.

Types of Padding:-

1. Valid Padding (No padding)

- Padding = 0
- Output shrinks
- Used when: Spatial reduction is desired

2. Same Padding (Most common)

- Padding chosen so **output size = input size**
- For 3×3 kernel → padding = 1
- Used in: Almost all deep CNNs (VGG, ResNet, CNNs)

3. Zero Padding

- Pad with zeros
- Most commonly used

Example with Padding = 1

Original Input (5 × 5)	After Zero Padding = 1 → 7 × 7 Input	Filter	5 × 5 Feature Map (With Padding = 1)
1 0 9 3 0 3 6 0 0 5 5 3 8 2 0 1 5 7 9 7 4 8 2 2 0	0000000 0109300 0360050 0538200 0157970 0482200 0000000	0 1 0 3 2 1 1 3 2	23 33 27 43 24 34 51 58 22 12 29 65 71 76 41 40 55 64 56 43 17 35 37 10 0

Convolution:

It is the process of **sliding a kernel over the input image**, performing **element-wise multiplication**, and **summing the results** to extract features.

Convolution Output = Σ (Input × Kernel)

Feature Map:

A **feature map** is the **output matrix formed by applying one filter across the entire input image**.

Pooling:

Pooling is a downsampling operation that **reduces the spatial size** of feature maps while keeping important information.

What Pooling Does?

- Takes a **small window** (e.g., 2×2)
- Moves with a stride
- Keeps **important value**

Types of Pooling:-

1. Max Pooling (Most Common)

Takes the **maximum value** from the window

Example:

Feature Map:

8 2

3 6

Max Pool → 8

Used when: Feature presence is important

2. Average Pooling

Takes the **average value**

Example: (8 + 2 + 3 + 6) / 4 = 4.75

Used when: Smooth representation needed

3. Global Pooling

- Global Max Pooling
- Global Average Pooling

Reduces entire feature map to **one value per channel**

Used in: Modern CNNs (ResNet, MobileNet)

Flatten:

Flatten converts a multi-dimensional feature map into a 1-D vector so it can be given to a Dense (Fully Connected) layer.

Dense Layer:

A Dense (Fully Connected) layer is a neural network layer where every neuron is connected to every neuron of the previous layer.

Why Do We Use Dense Layer in CNN? □ **Decision making & classification**

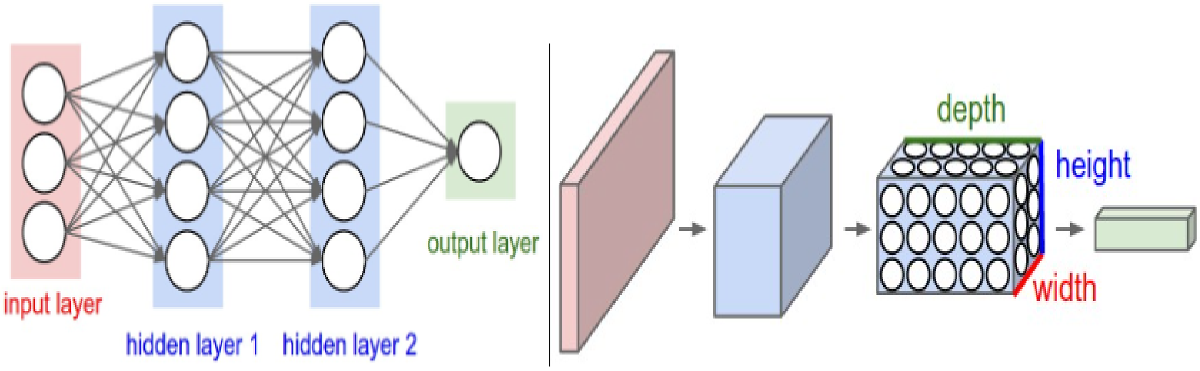
Output Dense Layer:-

Task	Activation
Binary classification	Sigmoid
Multi-class	Softmax
Regression	Linear

CNN layers = Feature extractor

Dense layer = Decision maker

Architecture of CNN



How a CNN model works