Name: Prathamesh Arvind Jadhav

Unit-3-Convolution Neural Network(CNN)

Introduction to Convolutional Neural Networks (CNN) in Deep Learning

A **Convolutional Neural Network (CNN)** is a specialized deep learning architecture primarily used for processing structured grid-like data, such as images and videos. CNNs are particularly powerful in tasks related to computer vision, including image classification, object detection, and segmentation.

---

1. Why CNN?

Traditional neural networks (fully connected networks) struggle with image data due to the large number of parameters required, leading to high computational costs and overfitting. CNNs solve this problem by using **convolutional layers** that learn spatial hierarchies of features.

---

2. Key Components of CNN

CNNs are composed of several important layers:

a) Convolutional Layer

- The **convolutional layer** is the core building block of CNN.
- It applies filters (kernels) to the input image to detect features like edges, textures, and patterns.
- A filter slides (performs a convolution operation) across the image, producing a feature map.

- **Mathematically:**

$$Feature\ Map = Input\ Image * Filter$$

(where * represents convolution operation)

b) Activation Function (ReLU)

- After convolution, the **Rectified Linear Unit (ReLU)** is applied to introduce non-linearity.
- It replaces negative values with zero, which helps the network learn complex patterns.

c) Pooling Layer

- **Pooling layers** reduce the spatial dimensions of the feature maps while retaining important information.
- The most common type is **Max Pooling**, which selects the maximum value from a region.
- Pooling helps in reducing computation and prevents overfitting.

d) Fully Connected Layer (FC)

- After several convolutional and pooling layers, the extracted features are **flattened** and passed to a fully connected layer.
- The FC layer is responsible for classification by assigning probabilities to different classes using **Softmax or Sigmoid activation**.

e) Output Layer

- The final output layer provides classification results, usually with **Softmax (for multi-class) or Sigmoid (for binary classification)**.

---

3. Working of CNN

A CNN follows a hierarchical approach to extract features:

1. **Lower layers** detect simple patterns (edges, textures).
2. **Middle layers** detect more complex patterns (shapes, objects).
3. **Higher layers** classify the image based on learned features.

---

4. Advantages of CNN

 Reduces Parameters: Uses shared weights in convolution layers, making it computationally efficient.

 Spatial Hierarchy Learning: Detects patterns at different levels, making it effective for vision tasks.
 Handles Variability: Works well with different image variations like rotation, translation, and scaling.

---

5. Applications of CNN

 Image Classification: Identifying objects in images (e.g., Cats vs. Dogs).
 Object Detection: Locating objects in an image (e.g., YOLO, Faster R-CNN).
 Medical Image Analysis: Detecting diseases in X-rays, MRIs, etc.
 Facial Recognition: Used in authentication systems (e.g., Face ID).
 Self-Driving Cars: Identifying pedestrians, traffic signs, and roads.

CNN Architecture Overview in Deep Learning

A **Convolutional Neural Network (CNN)** is a deep learning architecture specifically designed for processing image and video data. The CNN architecture is inspired by the **visual cortex of the human brain**, allowing it to detect patterns, edges, and objects efficiently.

CNN Architecture Layers

A CNN consists of multiple layers that extract and learn features progressively from an input image. The key layers are:

1.Input Layer

- The input to a CNN is usually an image represented as a 3D matrix with dimensions **(Height × Width × Channels)**.
- Example: A **RGB image of size 128×128** has dimensions **(128, 128, 3)** (where 3 represents Red, Green, and Blue color channels).

2.Convolutional Layer (Conv Layer)

- The **convolutional layer** applies **filters (kernels)** to the input image.

- These filters detect **features** such as edges, textures, and shapes.
- The output is a **feature map**, which represents the detected patterns.

Mathematically:

$$Feature\ Map = Input\ Image * Filter + Bias$$

(where * represents convolution operation)

- Filters slide across the image (stride) and perform element-wise multiplication.
- A **ReLU (Rectified Linear Unit)** activation function is applied to introduce non-linearity.

Example:

- **3×3 filter** applied to an image highlights specific features.
- **Stride:** Defines how much the filter moves (stride=1 means moving pixel by pixel).
- **Padding:** Adds extra pixels to preserve spatial dimensions.

3.Pooling Layer

- The **pooling layer** reduces the spatial size of feature maps, making computations efficient and reducing overfitting.
- It **retains important features** while discarding less significant details.

Types of Pooling:

 Max Pooling: Takes the **maximum** value in a given window (e.g., 2×2).
 Average Pooling: Takes the **average** of values in a window.

 Example: A **2×2 max pooling** reduces a 4×4 feature map to **2×2** while keeping the strongest feature.

4.Flatten Layer

- Converts the **2D feature maps** into a **1D vector**.

- This step prepares the data for the **fully connected layer**.

5.Fully Connected Layer (FC)

- The **fully connected (dense) layer** connects every neuron to every neuron in the next layer.
- It **combines extracted features** and learns complex patterns.
- Uses an **activation function (e.g., ReLU, Softmax, Sigmoid)** for classification.

 Example:

- If detecting cats vs. dogs, the final FC layer might output **[0.9, 0.1]**, meaning 90% confidence for "cat" and 10% for "dog".

6.Output Layer

- The **final layer** produces the classification result.

Activation Functions:

 Softmax: Used for **multi-class classification** (e.g., "Dog", "Cat", "Bird").
 Sigmoid: Used for **binary classification** (e.g., "Spam" or "Not Spam").

 Summary of CNN Architecture

A CNN follows a **hierarchical structure** where each layer learns different features:

1.**Input Layer** → Raw Image Data
2.**Convolutional Layer** → Extracts Features
3.**ReLU Activation** → Adds Non-linearity
4.**Pooling Layer** → Reduces Dimensionality
5.**Flatten Layer** → Converts Feature Maps into a 1D Vector
6.**Fully Connected Layer** → Combines Features for Classification
7.**Output Layer** → Provides Final Prediction

Name: Prathamesh Arvind Jadhav

Advantages of CNN

 Automatically learns features (no manual feature extraction).
 Efficient and scalable for large image datasets.
 Handles spatial relationships in images effectively.

Applications

 Image Classification (e.g., Face Recognition, Object Detection)
 Medical Imaging (e.g., Cancer Detection, X-ray Analysis)
 Self-Driving Cars (e.g., Traffic Sign Recognition)
 NLP (e.g., Sentiment Analysis using CNN for text)

The Basic Structure of a Convolutional Neural Network (CNN) in Deep Learning

A **Convolutional Neural Network (CNN)** consists of several specialized layers that process and learn from image data. Each layer serves a specific purpose, such as extracting features, reducing dimensionality, or classifying objects.

Key Components of CNN Architecture

1.Padding

Padding is the process of adding extra pixels (usually zeros) around the input image to control the spatial size of the output feature maps.

Why Use Padding?

- Ensures the **size of the output** remains the same as the input.
- Helps **preserve edge features** of the image.
- Prevents the feature map from **shrinking** too much after multiple convolutions.

 Types of Padding:
 Same Padding (Zero Padding): Keeps the spatial dimensions the same.
 Valid Padding: No padding is added, reducing feature map size.

📌 **Example:**

If a **5×5 image** is convolved with a **3×3 filter** and no padding:

$$\text{Output size} = (5 - 3 + 1) = 3 \times 3$$

If padding of 1 is used, the output remains **5×5**.

2.Strides

Stride determines **how much the filter moves** at each step during convolution.

 Typical Settings:
 Stride = 1 (Default): Moves the filter **one pixel** at a time (detailed feature extraction).
 Stride = 2 or more: Moves the filter **two or more pixels** (reduces computational load but may lose details).

 Example:

- A **stride of 1** preserves details, while
- A **stride of 2** reduces feature map size faster.

3.ReLU (Rectified Linear Unit) Layer

After convolution, we apply an **activation function** to introduce **non-linearity** (since images are naturally non-linear).

 Why Use ReLU?

- **Removes negative values** (replaces them with 0).
- Prevents the **vanishing gradient problem** (compared to sigmoid/tanh).
- Accelerates training.

📌 Mathematical Representation:

$$f(x) = \max(0, x)$$

🔹 Typical Setting: Used after every convolutional layer.

4.Pooling Layer (Downsampling)

Pooling is used to **reduce the spatial dimensions** while preserving the important features.

 Why Use Pooling?

- **Reduces computation** and memory usage.
- Helps in **preventing overfitting**.
- Extracts **dominant features** (important edges, textures).

 Types of Pooling:
 Max Pooling: Takes the **maximum** value from each region.
 Average Pooling: Takes the **average** of values in the region.

 Typical Setting:

- A **2×2 max pooling with stride 2** is commonly used.
- Reduces the feature map size by **half**.

5.Fully Connected (FC) Layers

After extracting features through convolution and pooling, we **flatten** the feature maps into a **1D vector** and pass them to a fully connected layer for classification.

 Why Use FC Layers?

- Combines learned **spatial features** to make predictions.
- Uses **Softmax (multi-class)** or **Sigmoid (binary classification)** for final output.

📌 **Example:**

If detecting **Cats vs. Dogs**, the **FC layer output** might be:

$$\text{Softmax}([0.9, 0.1]) \Rightarrow 90\% \text{ probability for "Cat"}$$

6.Interleaving between Layers

CNNs are built using a **repetitive structure** of different layers.

 Typical Order of Layers in CNN:
1.**Input Layer** (e.g., 128×128×3 image)
2.**Convolution Layer** (Detects edges, textures)
3.**ReLU Activation** (Adds non-linearity)
4.**Pooling Layer** (Reduces size)
5.**Convolution Layer** (Extracts high-level features)
6.**ReLU Activation**
7.**Pooling Layer**
8.**Flatten Layer**
9.**Fully Connected Layer**
 Output Layer (Softmax or Sigmoid)

 Example: LeNet-5, AlexNet, VGG, ResNet follow this pattern.

7.Local Receptive Fields (Local Connectivity)

Instead of connecting every neuron to every pixel (like in traditional neural networks), CNNs **use local connectivity**.

 Why Use Local Receptive Fields?

- **Preserves spatial relationships** in images.
- **Reduces the number of parameters**, making training faster.

 Example:
A **3×3 filter** in the first layer only **looks at 9 pixels** instead of the whole image.

Name: Prathamesh Arvind Jadhav

□ Summary

Component	Purpose	Typical Setting
Padding	Keeps feature map size stable	**Same Padding** (default)
Strides	Controls movement of filters	**Stride = 1 or 2**
ReLU	Adds non-linearity	Used after **every Conv Layer**
Pooling	Reduces feature map size	**2×2 Max Pooling, Stride 2**
Fully Connected Layers	Combines extracted features	Last layers before output
Interleaving Layers	Maintains feature learning hierarchy	Conv → ReLU → Pooling → FC
Local Connectivity	Reduces parameters, learns spatial patterns	**3×3 filters are common**

Local Response Normalization (LRN) in Deep Learning

What is Local Response Normalization (LRN)?

Local Response Normalization (LRN) is a normalization technique used in deep learning, specifically in **Convolutional Neural Networks (CNNs)**. It helps improve the generalization of the model by **enhancing the activation of prominent features while suppressing weaker activations** in a local neighborhood.

It was first introduced in the **AlexNet architecture**, where it was applied after the **ReLU activation function**.

□ Why Use LRN?

- Helps **sharpen feature maps** by emphasizing high-activated neurons.
- Encourages **competition between neurons** that detect similar patterns.
- Improves **generalization** by reducing overfitting.
- Particularly useful in early CNN architectures for **image classification**.

Name: Prathamesh Arvind Jadhav

□ How Does LRN Work?

LRN applies a **normalization across nearby neurons** within the same feature map or across different feature maps. This means that each neuron's activation is normalized based on the activations of its neighboring neurons.

Mathematically, LRN is defined as:

$$b_{x,y}^{i} = \frac{a_{x,y}^{i}}{\left(k + \alpha \sum_{j=\max(0,i-n/2)}^{\min(N-1,i+n/2)} (a_{x,y}^{j})^2\right)^{\beta}}$$

where:

- $b_{x,y}^{i}$ = Normalized output at position (x, y) in the i^{th} channel.
- $a_{x,y}^{i}$ = Activation before normalization.
- k = Small positive constant (typically 1), avoids division by zero.
- α = Scaling factor (typically **1e-4**).
- n = Size of the local neighborhood (typically **5**).
- β = Exponent (typically **0.75**).
- N = Total number of feature maps.

Intuition:

- If a neuron is strongly activated, LRN **suppresses nearby activations**.
- This helps create **better contrast** between important and less important features.

□ Types of LRN

There are two main types of LRN:

1.**Across Channel Normalization (Depth LRN)**

- Normalizes over **neighboring channels** (feature maps) at the same spatial location.
- Used in **AlexNet** and similar models.

- Formula:

$$b_{x,y}^{i} = \frac{a_{x,y}^{i}}{\left(k + \alpha \sum_{j=\max(0,i-n/2)}^{\min(N-1,i+n/2)} (a_{x,y}^{j})^2\right)^{\beta}}$$

2.Within Channel Normalization (Spatial LRN)

- Normalizes **within a single feature map**, considering nearby spatial locations.
- Less commonly used.

□ LRN in AlexNet

- AlexNet applies LRN **after the first and second convolutional layers**.
- Helps in **reducing redundant activations** and improves feature discrimination.

□ LRN vs Batch Normalization

Feature	LRN	Batch Normalization
Purpose	Enhances contrast between activations	Normalizes activations for stable learning
Normalization Scope	Local (neighboring neurons)	Across entire batch
When Applied	After ReLU in CNNs (older models)	Before/after activation in any layer
Common in	AlexNet, early CNNs	Almost all modern deep networks

 Batch Normalization is preferred in modern networks because it improves training stability and reduces internal covariate shift.

When to Use LRN?

 When training **older architectures like AlexNet**.
 When working with **image data where contrast enhancement is needed**.
 When wanting **to create competition between neurons** in early layers.

 Avoid using LRN in modern CNNs since **Batch Normalization (BN)** is more effective.

Training a Convolutional Neural Network (CNN) in Deep Learning

Training a **Convolutional Neural Network (CNN)** involves multiple steps, including **data preprocessing, forward propagation, loss calculation, backpropagation, and optimization**. The goal is to adjust the network's weights to minimize the prediction error.

 Steps to Train a CNN

1.Data Preparation and Preprocessing

Before training, the dataset must be **preprocessed** to improve the model's performance.

 Common Preprocessing Techniques: **Normalization**: Scale pixel values to a range (e.g., [0,1] or [-1,1]).
 Data Augmentation: Apply transformations (flipping, rotation, scaling) to increase data variability.
 Resizing: Ensure all images have the same dimensions.
 Splitting Dataset: Divide into **training, validation, and test** sets.

2.Forward Propagation

In **forward propagation**, the input data is passed through the CNN layers, and the **feature maps** are generated.

 CNN Layers in Forward Propagation:

1.**Convolution Layer:** Extracts **spatial features** from images.
2.**ReLU Activation:** Introduces **non-linearity**.
3.**Pooling Layer:** Reduces **feature map size**.
4.**Flatten Layer:** Converts feature maps to a **1D vector**.
5.**Fully Connected (FC) Layer:** Learns **high-level patterns**.
6.**Output Layer (Softmax or Sigmoid):** Produces final **class probabilities**.

3.Loss Function Calculation

The **loss function** measures how well the CNN is performing.

 Common Loss Functions: **Categorical Crossentropy** → For multi-class classification.
 Binary Crossentropy → For binary classification.
 Mean Squared Error (MSE) → For regression problems.

4.Backpropagation (Gradient Calculation)

During **backpropagation**, the CNN calculates the **gradient of the loss function** with respect to each weight using **partial derivatives**.

 Steps in Backpropagation: 1.Compute the **error** (difference between predicted and actual values).
2.Compute gradients using **chain rule** (partial derivatives).
3.Update weights **in the opposite direction of gradients**.

📌 Mathematical Representation:

$$w = w - \eta \cdot \frac{\partial L}{\partial w}$$

where:

- w = weight parameter,
- η = learning rate,
- $\frac{\partial L}{\partial w}$ = gradient of the loss function.

5.Optimization (Weight Updates)

An **optimizer** updates the CNN's weights based on the computed gradients.

 Popular Optimizers: **SGD (Stochastic Gradient Descent):** Simple but slow convergence.
 Adam (Adaptive Moment Estimation): Adaptive learning rates, widely used.
 RMSprop: Works well with non-stationary objectives.

6.Model Training (Epochs and Batches)

CNN training involves feeding the dataset in small **batches** and running for multiple **epochs**.

 Key Training Terms: **Batch Size** → Number of samples processed at once.
 Epoch → One full pass over the entire dataset.
 Iteration → Number of batch updates per epoch.

7.Evaluating the Model

After training, we test the CNN's performance on unseen data.

Metrics for Evaluation: **Accuracy**: Measures the proportion of correct predictions.
 Precision, Recall, F1-score: Used for imbalanced datasets.
 Confusion Matrix: Visualizes correct and incorrect classifications.

8.Fine-Tuning and Regularization

To improve CNN performance, we apply techniques like:

 Dropout: Randomly drops neurons to prevent overfitting.
 L2 Regularization: Adds penalty to large weights.
 Learning Rate Scheduling: Adjusts learning rate dynamically.
 Transfer Learning: Uses pre-trained models like **VGG, ResNet**.

Summary of CNN Training Pipeline

Step	Description

Name: Prathamesh Arvind Jadhav

Step	Description
1. Data Preparation	Normalize, augment, and split dataset.
2. Forward Propagation	Pass images through CNN layers.
3. Compute Loss	Compare predictions with ground truth.
4. Backpropagation	Compute gradients and update weights.
5. Optimization	Use Adam, SGD, or RMSprop for learning.
6. Model Training	Run multiple epochs with batch processing.
7. Evaluation	Test accuracy, confusion matrix, and metrics.
8. Fine-Tuning	Use dropout, regularization, or transfer learning.