

Module-4

Deep Learning (BCA-701)

Module-4

Convolutional Neural Networks (CNN)

A **Convolutional Neural Network** is a deep learning model that uses **convolution**, **pooling**, and **fully connected layers** to extract hierarchical features from input data, especially images.

Why CNNs are Needed

Traditional neural networks:

- Treat all inputs equally
- Ignore spatial structure
- Have too many parameters for images

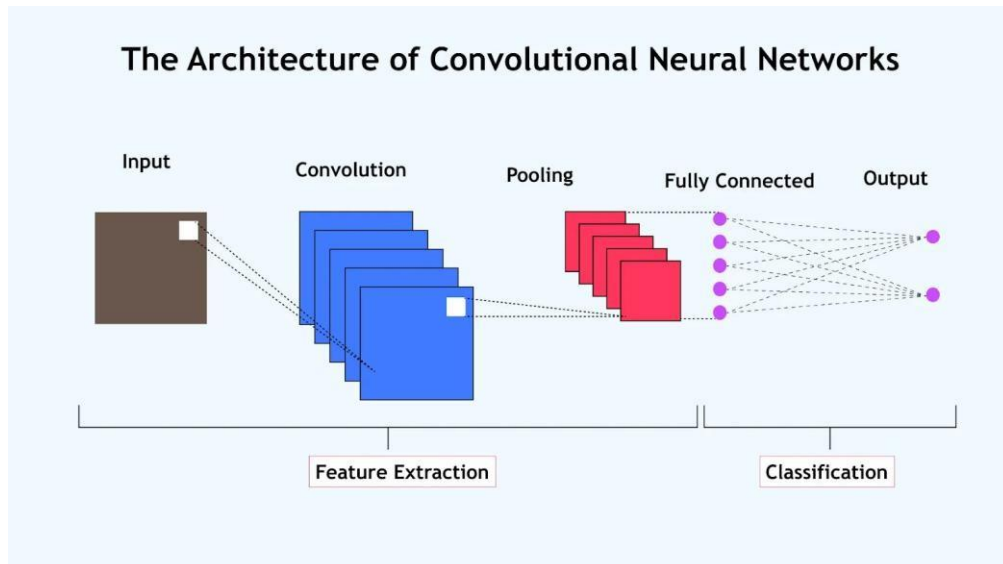
CNNs overcome these problems by:

- Using **local connections**
- Sharing weights
- Preserving spatial information

Basic Architecture of CNN

A typical CNN consists of:

1. **Input Layer**
2. **Convolution Layer**
3. **Activation Function (ReLU)**
4. **Pooling Layer**
5. **Fully Connected Layer**
6. **Output Layer**



1. Convolution Layer

- Core building block of CNN
- Uses **filters (kernels)** to extract features
- Produces **feature maps**

Feature Map

$$ap = Input * Filter$$

✓ Detects edges, corners, patterns

2. Activation Function (ReLU)

- Introduces non-linearity
- ReLU:

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

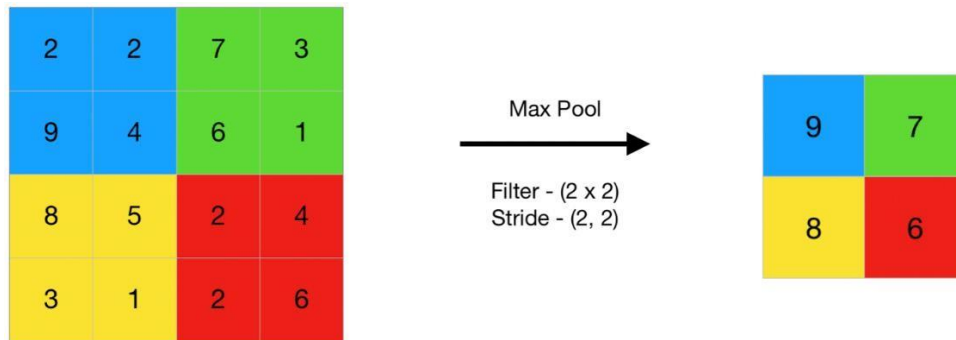
✓ Faster training

✓ Avoids vanishing gradients

3. Pooling Layer

- Reduces spatial size
- Common types:
 - Max Pooling
 - Average Pooling

- ✓ Reduces computation
- ✓ Provides translation invariance



4. Fully Connected Layer

- Acts like a traditional neural network
- Combines extracted features
- Performs classification

Key Properties of CNN

- Local receptive fields
- Weight sharing
- Translation invariance
- Fewer parameters than MLPs

Applications of CNN

- Image classification
- Face recognition
- Medical image analysis
- Object detection
- Handwritten digit recognition

Advantages

- Automatic feature extraction
- High accuracy in vision tasks
- Efficient parameter usage

Limitations

- Computationally expensive
- Requires large datasets
- Hard to interpret

1. The Convolution Operation

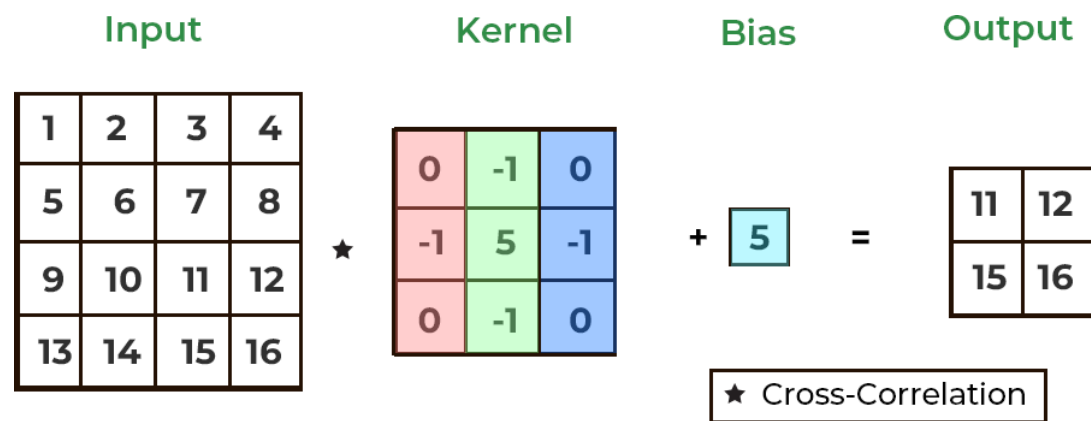
The **convolution operation** is the **core mathematical operation** in a Convolutional Neural Network (CNN). It is used to **extract local features** such as edges, corners, and textures from input data (usually images).

What is the Convolution Operation? (Simple Definition)

Convolution is a process where a **small matrix called a filter (kernel)** slides over the input image and computes **dot products** to generate a **feature map**.

Basic Idea

- Input image → large matrix
- Filter (kernel) → small matrix
- Slide the filter across the image
- Compute weighted sum at each position



★ Cross-Correlation

Mathematical Representation

For a 2D input image I and filter K :

$$S(i, j) = \sum_m \sum_n I(i + m, j + n) K(m, n)$$

Where:

- I = input image
- K = kernel (filter)
- S = output feature map

Step-by-Step Convolution

1. Place the filter on the top-left of the image
2. Multiply corresponding values
3. Add all multiplied values
4. Store result in feature map
5. Slide filter and repeat

Key Components of Convolution

1. Kernel (Filter)

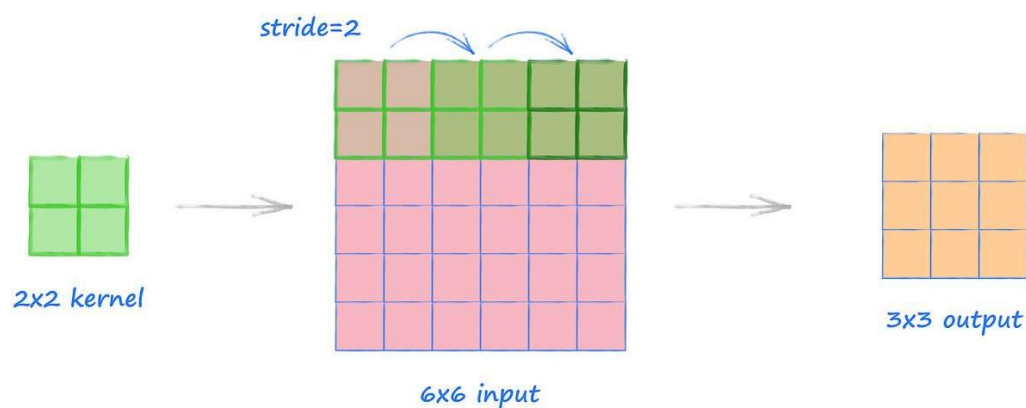
- Small matrix (e.g., 3×3 , 5×5)
- Learns specific features (edge, blur, texture)

2. Stride

- Number of pixels the filter moves each step
- Larger stride $\rightarrow$ smaller output size

3. Padding

- Adding extra pixels (usually zeros) around input
- Prevents shrinking of feature maps



Output Size Formula

For input size N , filter size F , padding P , stride S :

$$\text{Output} = \frac{N - F + 2P}{S} + 1$$

Why Convolution is Important

- Preserves spatial relationships
- Reduces number of parameters
- Enables automatic feature extraction

- Efficient for image data

Example (Exam-Oriented)

- Input image: 5×5
- Filter: 3×3
- Stride: 1
- Padding: 0

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

$$\Rightarrow 3$$

$$\times 3 f$$

eature m

ap

Key Exam Points

- Convolution uses filters to extract features
- Produces feature maps
- Uses stride and padding
- Reduces parameters via weight sharing

2. Motivation for Convolutional Neural Networks

The motivation for Convolutional Neural Networks (CNNs) arises from the limitations of traditional fully connected neural networks when handling image and spatial data. CNNs are designed to efficiently process data with grid-like structures, such as images.

Problems with Traditional Neural Networks

1. Large Number of Parameters

- Images have thousands to millions of pixels
- Fully connected networks require one weight per pixel per neuron
- Leads to huge memory and computation cost

2. Loss of Spatial Information

- Traditional networks flatten images into 1D vectors
- Spatial relationships (edges, shapes) are lost

3. Poor Scalability

- Small increase in image size → huge increase in parameters
- Training becomes impractical

4. Manual Feature Extraction

- Traditional ML needs hand-crafted features
- Not robust to variations in images

How CNNs Solve These Problems

1. Local Receptive Fields

- Neurons connect only to **local regions** of the image
- Capture local patterns like edges

2. Weight Sharing

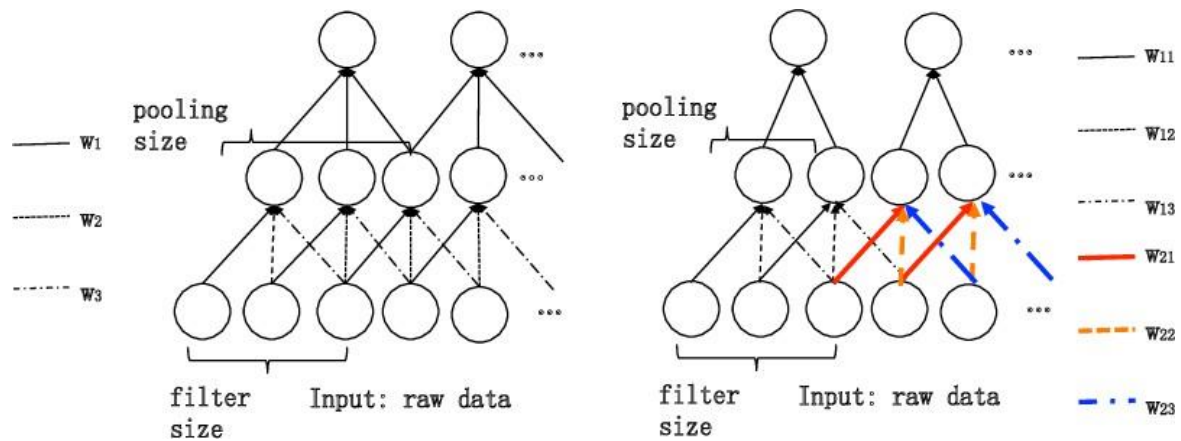
- Same filter used across the image
- Greatly reduces number of parameters

3. Translation Invariance

- Object recognition works even if object shifts slightly
- Achieved through convolution and pooling

4. Automatic Feature Learning

- CNNs learn features directly from data
- No manual feature engineering needed



Biological Inspiration

- Inspired by **visual cortex** of human brain
- Neurons respond to specific regions of visual field

Comparison: ANN vs CNN

Aspect	ANN	CNN
Connectivity	Fully connected	Local connectivity
Parameters	Very large	Much fewer
Spatial info	Lost	Preserved
Feature extraction	Manual	Automatic

Advantages Gained

- Efficient computation
- High accuracy in vision tasks
- Robust to noise and distortions

Key Points

- CNNs overcome ANN limitations
- Use local connectivity and weight sharing
- Preserve spatial relationships
- Inspired by human vision

3. Pooling

Pooling is an important operation in **Convolutional Neural Networks (CNNs)** used to **reduce the spatial size** of feature maps while retaining the most important information. It helps make the network **computationally efficient and robust**.

What is Pooling? (Simple Definition)

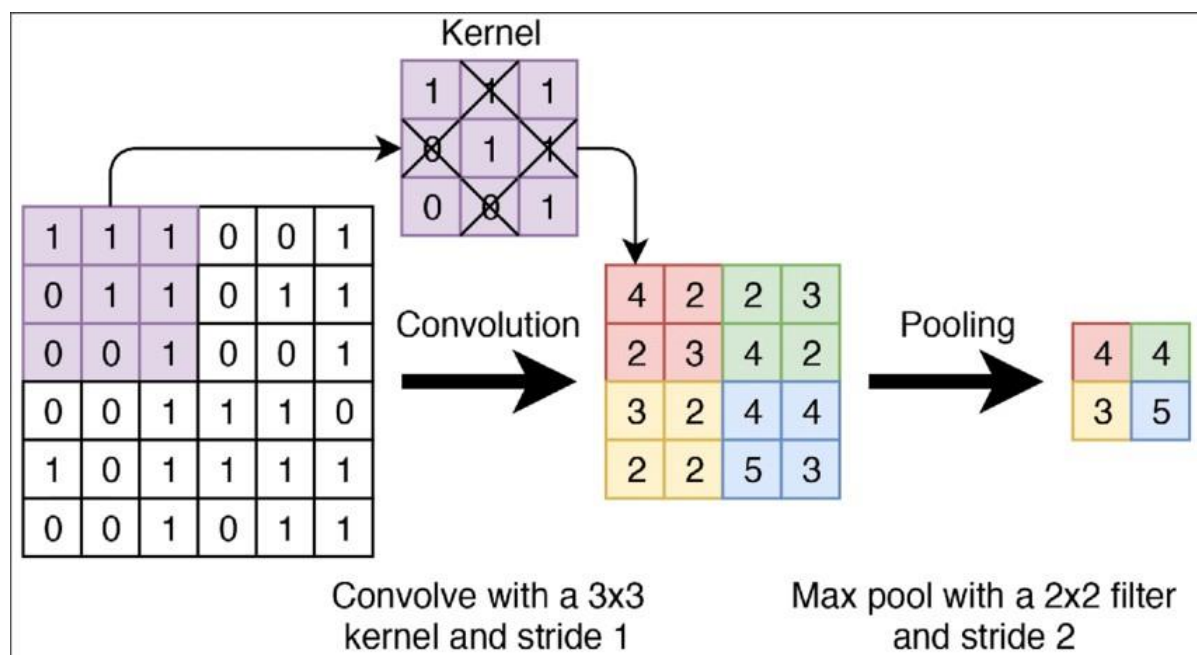
Pooling is a down-sampling technique that **summarizes a local region** of a feature map into a single value.

Why Pooling is Needed

- Reduces computation and memory
- Controls overfitting
- Provides **translation invariance**
- Simplifies feature maps

How Pooling Works

- A pooling window (e.g., 2×2) slides over the feature map
- A summary value is computed for each window
- No learnable parameters are involved



Types of Pooling

1. Max Pooling

- Takes the **maximum value** from each region
- Most commonly used

2. Average Pooling

- Takes the **average value** of each region

3. Global Pooling

- Pools over the entire feature map
- Commonly used before output layer

Key Parameters in Pooling

- Pooling size (e.g., 2×2)
- Stride
- Type of pooling

Advantages of Pooling

- Reduces dimensionality
- Prevents overfitting
- Improves generalization
- Makes CNN robust to small shifts

Limitations of Pooling

- Loss of precise spatial information
- May discard useful details

Key Points

- Pooling reduces feature map size
- No weights are learned
- Max pooling is most common
- Provides translation invariance

4. Convolution and Pooling as an Infinitely Strong Prior

In CNNs, **convolution and pooling** act as an **infinitely strong prior** because they **hard-code strong assumptions** about the structure of the data—especially images—*before* any learning happens.

What is a “Prior” in Machine Learning?

A **prior** is an assumption about the data **built into the model**.

- Weak prior → model must learn structure from data
- Strong prior → structure is assumed in advance

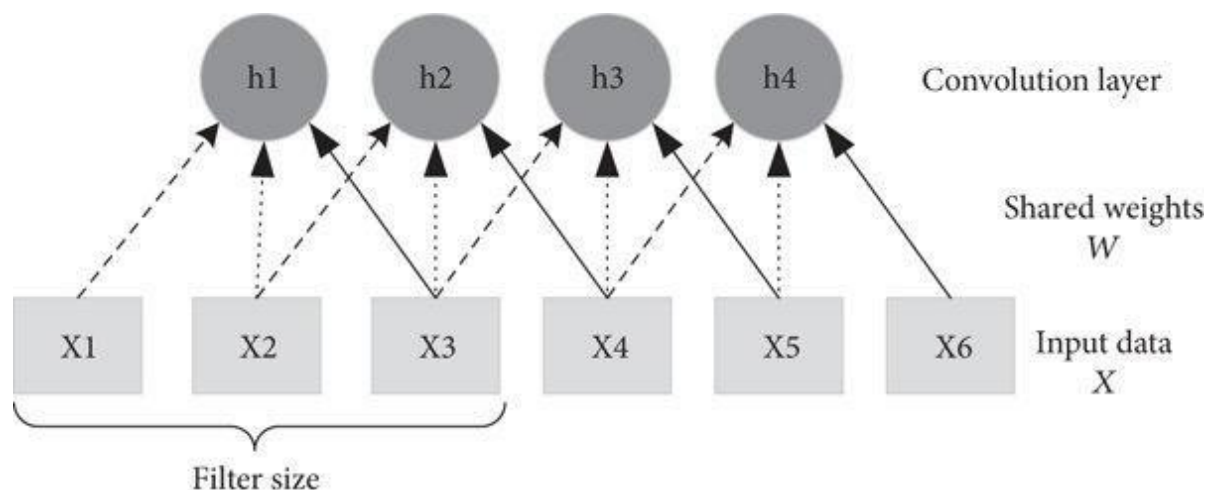
CNNs impose **very strong priors** about how images are formed.

Why Convolution is an Infinitely Strong Prior

Key Assumptions of Convolution

1. **Locality**
 - Nearby pixels are more related than distant ones
2. **Translation Equivariance**
 - Same feature (edge, corner) can appear anywhere
3. **Weight Sharing**
 - The same filter is applied across the entire image

These assumptions are **always enforced**, not learned.

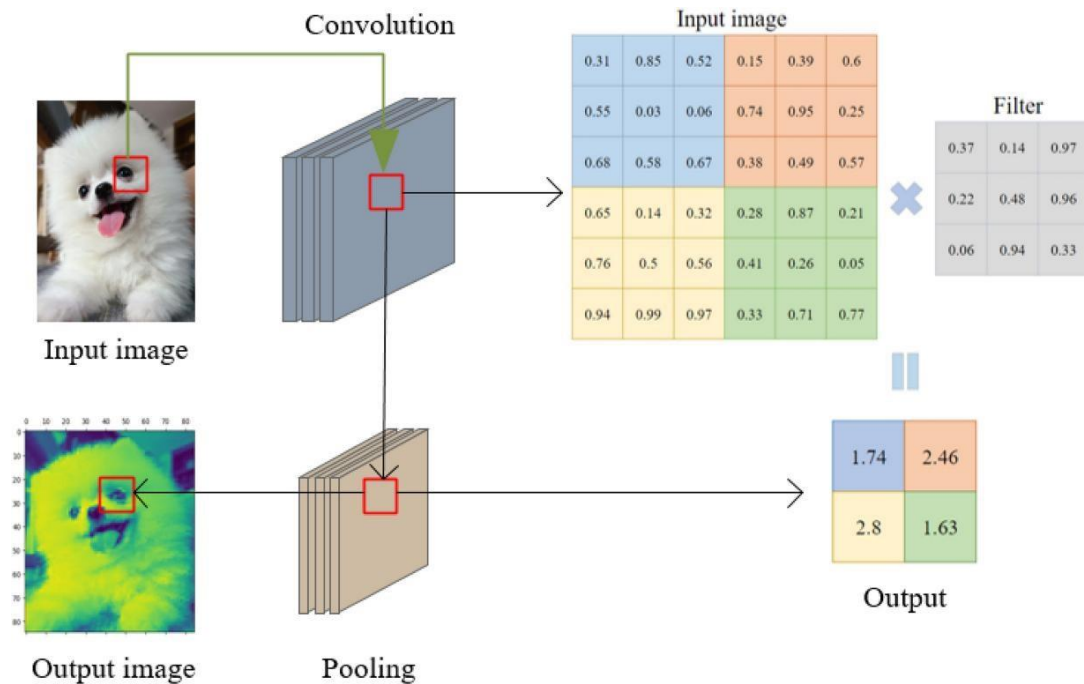


Why Pooling is an Infinitely Strong Prior

Key Assumptions of Pooling

1. Exact location is less important than presence
2. Small translations should not change output
3. Feature invariance is desirable

Pooling **forces invariance**, regardless of data.



Meaning of “Infinitely Strong”

- The model **cannot choose** to ignore these assumptions
- Unlike regularization (which can be tuned), convolution & pooling are **structural constraints**
- They permanently restrict the hypothesis space

Comparison with Fully Connected Networks

Aspect	Fully Connected NN	CNN
Prior strength	Weak	Very strong
Spatial assumptions	None	Built-in
Translation handling	Learned	Enforced

Aspect	Fully Connected NN	CNN
Parameters	Huge	Much fewer

Advantages of This Strong Prior

- Requires **less training data**
- Faster convergence
- Better generalization for images
- Reduced overfitting

Disadvantages

- Not suitable for data without spatial structure
- Less flexible than fully connected networks
- Assumptions may fail for non-image data

Key Points

- Convolution enforces locality & weight sharing
- Pooling enforces translation invariance
- Both impose hard constraints
- Hence, they act as infinitely strong priors

5. Variants of the Basic Convolution Function

In Convolutional Neural Networks (CNNs), the **basic convolution operation** can be modified in several ways to **improve efficiency, capture richer features, or reduce computation**. These modified forms are called **variants of the basic convolution function**.

Why Do We Need Variants of Convolution?

- To reduce computational cost

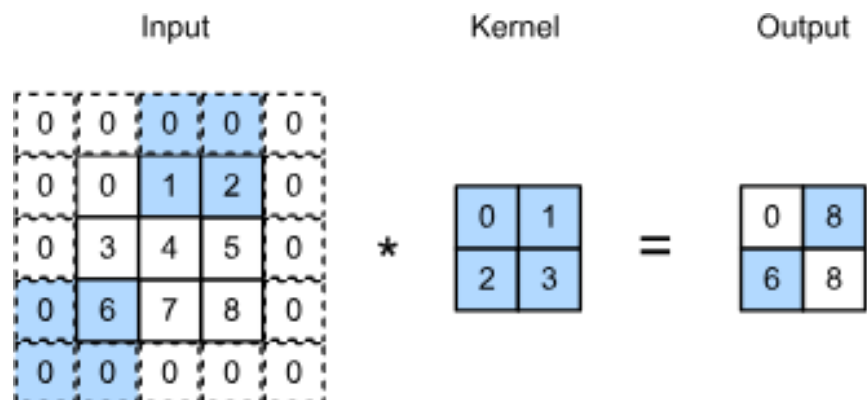
- To increase receptive field without increasing parameters
- To improve feature extraction
- To design deeper and more efficient CNNs

Major Variants of Convolution Functions

1. Strided Convolution

Description

- Filter moves with a **stride greater than 1**
- Reduces spatial dimensions directly (acts like pooling)



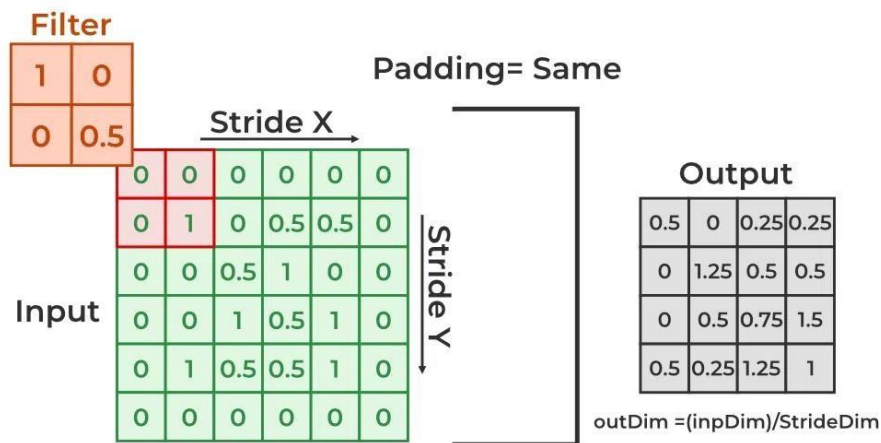
Key Points

- Down-sampling without pooling
- Fewer feature map values
- Faster computation

2. Padded Convolution

Description

- Extra pixels (usually zeros) are added around input
- Controls output size



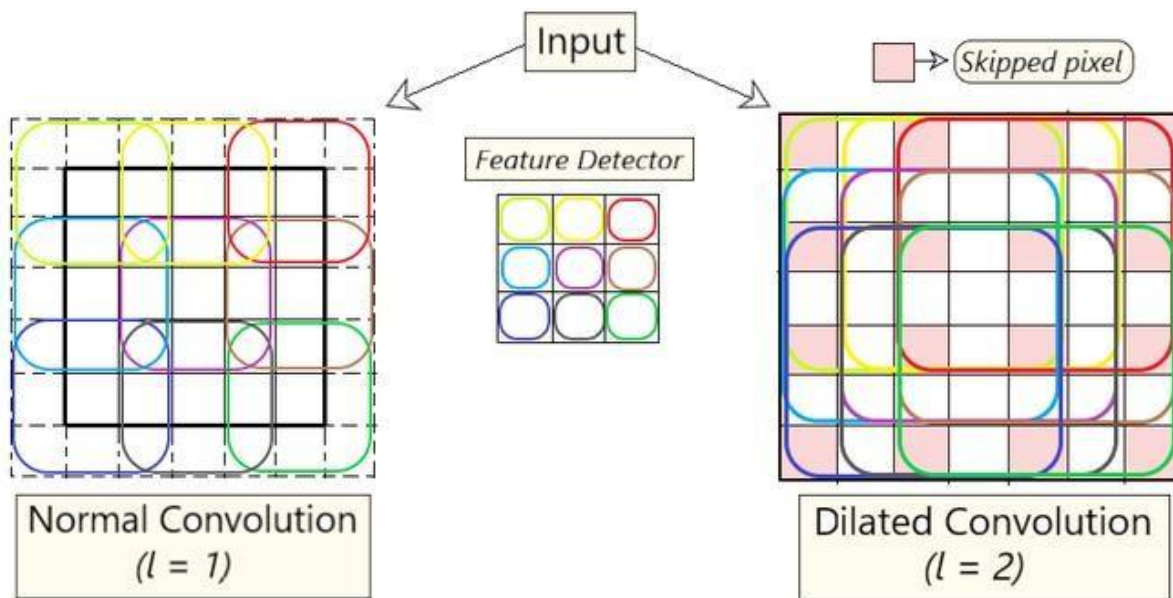
Types

- Valid convolution → No padding
- Same convolution → Output size = input size

3. Dilated (Atrous) Convolution

Description

- Inserts gaps between kernel elements
- Expands receptive field without increasing parameters



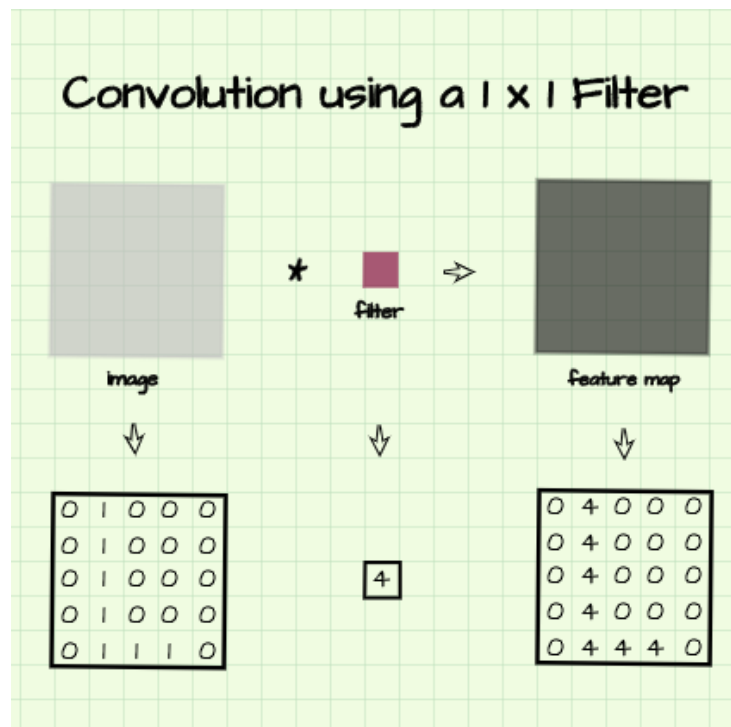
Key Points

- Captures global context
- Used in semantic segmentation

4. 1×1 Convolution (Pointwise Convolution)

Description

- Uses 1×1 filters
- Operates across channel dimension



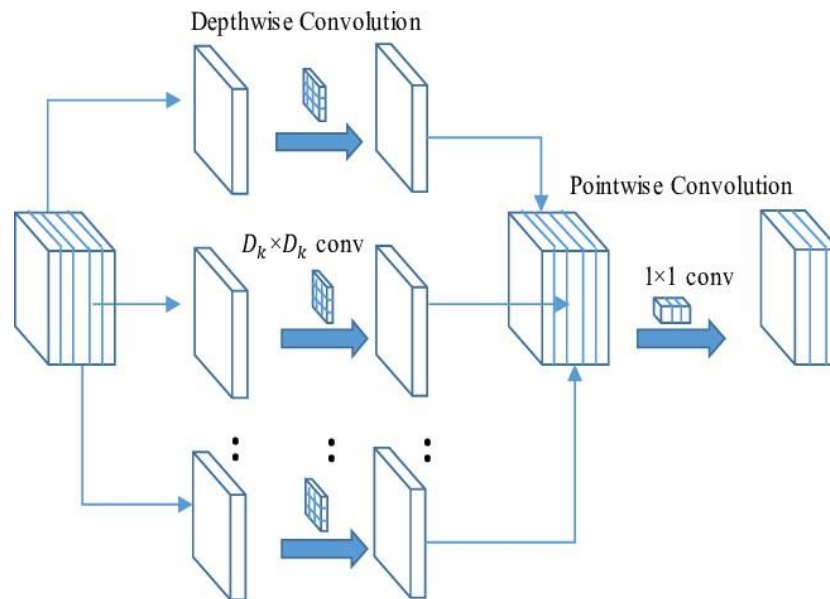
Key Points

- Reduces or increases number of channels
- Used in bottleneck layers
- Improves efficiency

5. Depthwise Separable Convolution

Description

- Splits convolution into:
 1. Depthwise convolution
 2. Pointwise (1×1) convolution



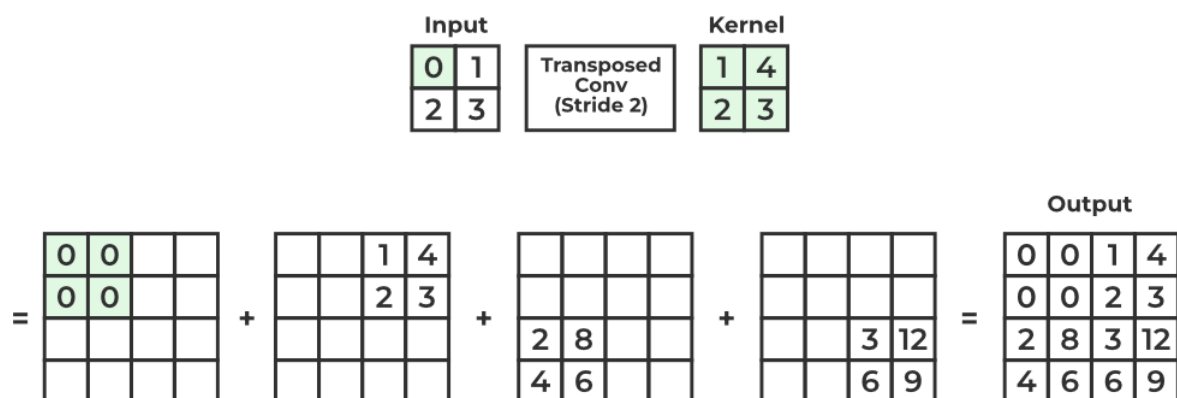
Key Points

- Drastically reduces computation
- Used in MobileNet
- Efficient for mobile devices

6. Transposed Convolution (Deconvolution)

Description

- Used to **increase spatial resolution**
- Often used in decoder networks



Key Points

- Used for up-sampling
- Common in segmentation and GANs

Comparison Summary

Variant	Purpose	Key Benefit
Strided	Down-sampling	Faster computation
Padded	Size control	Preserve borders
Dilated	Larger context	Bigger receptive field
1×1	Channel mixing	Parameter reduction
Depthwise	Efficiency	Low computation
Transposed	Up-sampling	Resolution increase

Key Points

- Variants improve efficiency and performance
- 1×1 convolution reduces parameters
- Dilated convolution increases receptive field
- Depthwise separable convolution is computationally efficient

7. Structured Outputs

In many real-world vision tasks, the output of a neural network is **not a single label or value**, but a **structured object** such as an image, a sequence, or a label for every pixel. Such outputs are called **structured outputs**.

What are Structured Outputs? (Simple Definition)

Structured outputs are outputs where **multiple predictions are interdependent** and have an internal structure, rather than being independent scalar values.

— Example: Predicting a **label for every pixel** in an image.

Why Structured Outputs are Important

Traditional CNN classifiers output:

- One label per image (e.g., "cat", "dog")

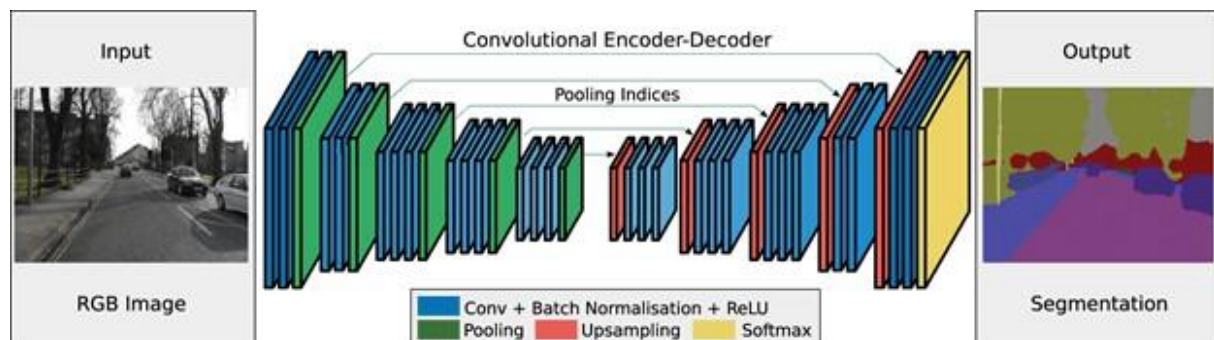
But many tasks require:

- Pixel-level predictions
- Spatial consistency
- Correlated outputs

Examples of Structured Output Problems

1. Semantic Segmentation

- Assign a class label to **each pixel**
- Output is an image-sized label map



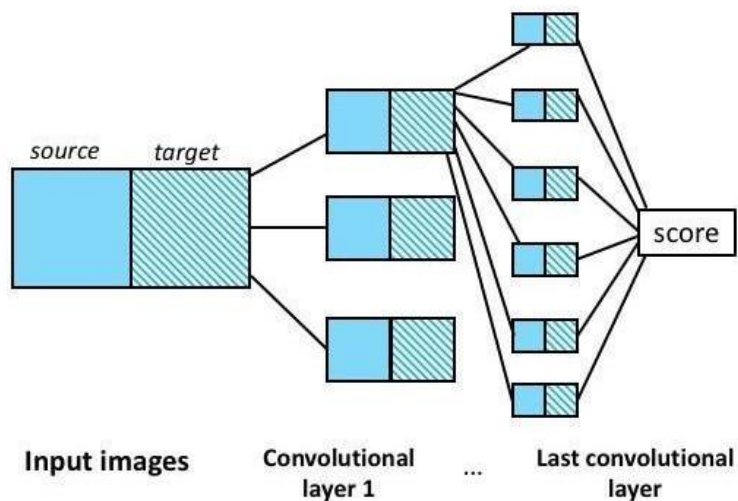
2. Object Detection

- Output includes:
 - Bounding box coordinates
 - Class labels
- Outputs are structured and related



3. Image-to-Image Mapping

- Input image $\rightarrow$ output image
- Examples:
 - Image denoising
 - Image super-resolution



4. Pose Estimation

- Output is a set of key points
- Each point depends on others

How CNNs Handle Structured Outputs

1. Fully Convolutional Networks (FCNs)

- Remove fully connected layers
- Output is a spatial map

2. Encoder-Decoder Architecture

- Encoder extracts features
- Decoder reconstructs structured output

3. Use of Transposed Convolution

- Upsamples feature maps
- Restores spatial resolution

Loss Functions for Structured Outputs

- Pixel-wise cross-entropy
- Dice loss
- IoU loss

These losses consider **spatial structure**.

Why CNNs Are Suitable for Structured Outputs

- Preserve spatial relationships
- Local connectivity ensures consistency
- Weight sharing improves generalization

Key Points

- Structured outputs contain dependent predictions
- Used in segmentation, detection, image generation
- CNNs preserve spatial structure
- FCNs and encoder–decoder models are common

8. Data Types (in Convolutional Neural Networks)

In CNNs, **data type** refers to the **structure and dimensionality** of the input data that the network processes. CNNs are especially designed for **grid-structured data**, where spatial or temporal relationships matter.

Why Data Types Matter in CNNs

- CNN operations (convolution & pooling) rely on **local neighborhoods**
- The data type determines:

- Kernel shape
- Dimensionality of convolution
- Network architecture

Major Data Types Used in CNNs

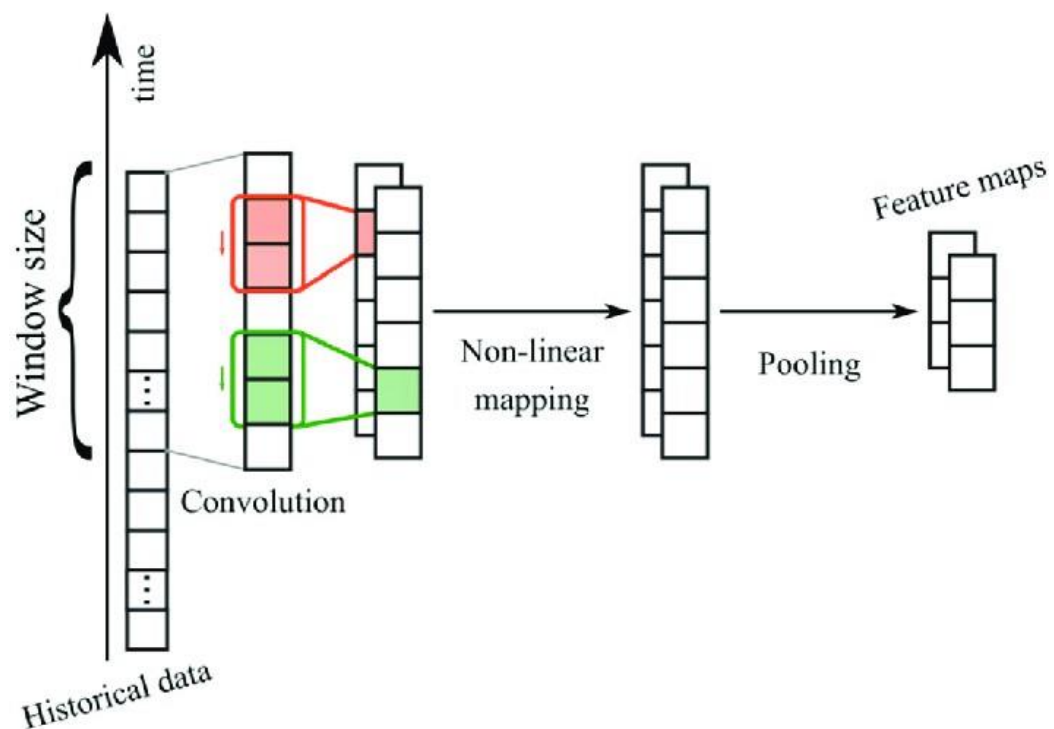
1. One-Dimensional (1D) Data

Description

- Data arranged along **one axis**
- Processed using **1D convolution**

Examples

- Time-series data
- Audio signals
- Sensor readings
- Text (after embedding)



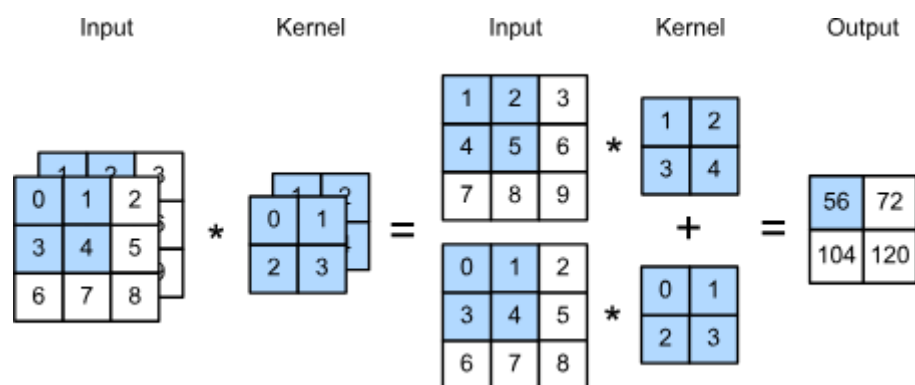
2. Two-Dimensional (2D) Data

Description

- Data arranged in **rows and columns**
- Most common CNN data type

Examples

- Grayscale images
- Color images
- Medical scans (X-rays)



Input Shape

- Grayscale: $H \times W \times 1$
- RGB image: $H \times W \times 3$

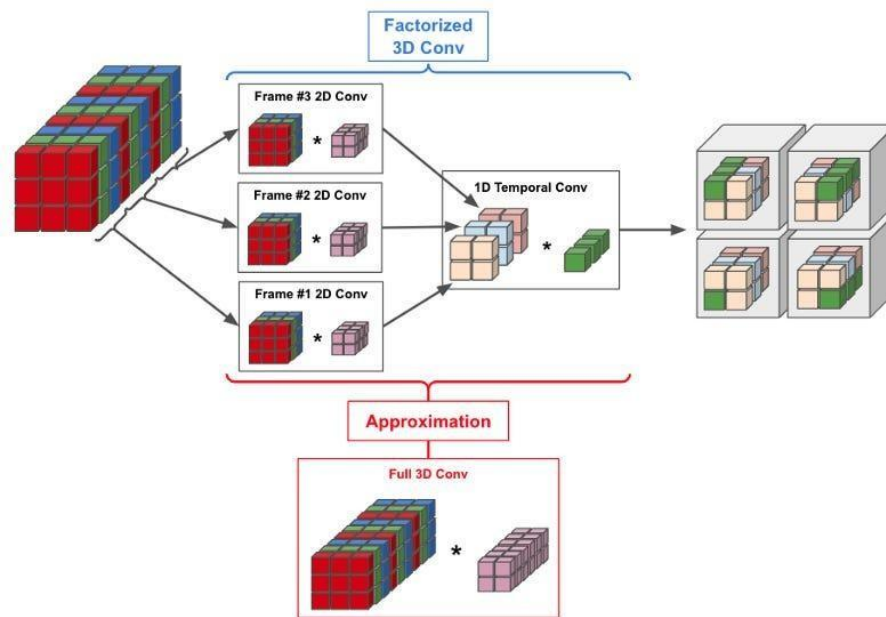
3. Three-Dimensional (3D) Data

Description

- Data has **three spatial dimensions**
- Uses **3D convolution**

Examples

- Video data (frames $\times$ height $\times$ width)
- CT / MRI medical volumes



Input Shape

$$Depth \times Height \times Width \times Channels$$

4. Multichannel Data

Description

- Multiple feature channels per data point
- CNN learns **cross-channel correlations**

Examples

- RGB images (3 channels)
- Hyperspectral images (many channels)

5. Structured Grid Data

Description

- Data lies on a **regular grid**
- CNN assumptions hold strongly

Examples

- Images

- Audio spectrograms

Data Types vs Convolution Type

Data Type	Convolution Used	Example
1D	1D Convolution	Audio
2D	2D Convolution	Images
3D	3D Convolution	Videos
Multi-channel	Channel-wise	RGB images

Key Points

- CNNs are best for grid-structured data
- 1D, 2D, and 3D convolutions match data dimensionality
- Channels represent different features
- Data type determines CNN architecture

G. Efficient Convolution Algorithms

Efficient convolution algorithms are techniques designed to **reduce the computational cost and memory usage** of convolution operations in CNNs, especially when dealing with **large images, deep networks, or real-time constraints**.

Why Efficiency is Important

- Convolution dominates CNN computation time
- Deep CNNs use many filters and layers
- Naïve convolution is computationally expensive
- Efficiency enables:
 - Faster training
 - Real-time inference
 - Deployment on mobile/embedded devices

Major Efficient Convolution Algorithms

1. Exploiting Weight Sharing (Basic CNN Advantage)

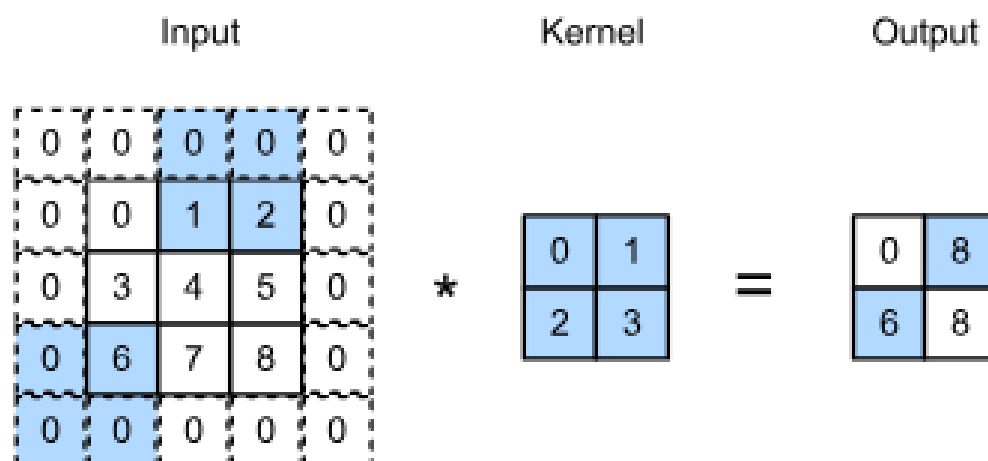
- Same kernel reused across entire input
- Already much more efficient than fully connected layers

✓ Reduces parameters drastically

✓ Built-in efficiency of CNNs

2. Strided Convolution

- Uses stride > 1 to **downsample** feature maps
- Reduces number of convolution operations



✓ Often replaces pooling

✓ Faster computation

3. FFT-Based Convolution

- Uses **Fast Fourier Transform (FFT)**
- Converts convolution into element-wise multiplication

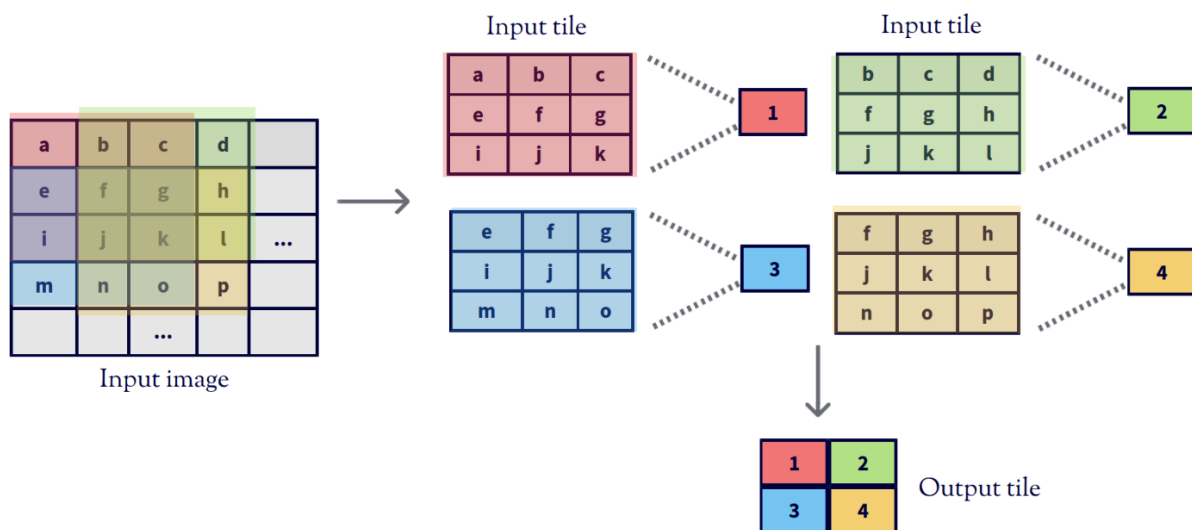
Idea

$$\text{Conv}(x, k) = \text{FFT}^{-1}(\text{FFT}(x) \cdot \text{FFT}(k))$$

- ✓ Very efficient for **large kernels**
- ✓ Used in some high-performance systems
- + Overhead for small kernels

4. Winograd Convolution Algorithm

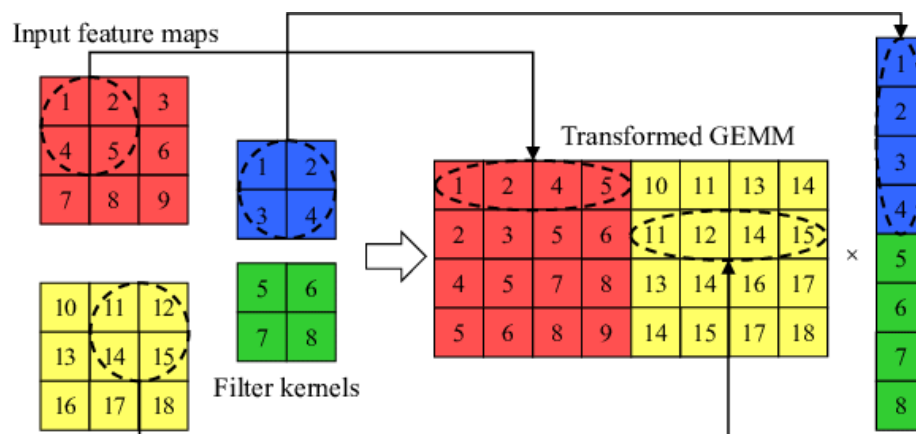
- Reduces number of multiplications
- Particularly effective for **small kernels (3×3)**



- ✓ Faster than standard convolution
- ✓ Widely used in optimized CNN libraries

5. Im2Col + Matrix Multiplication

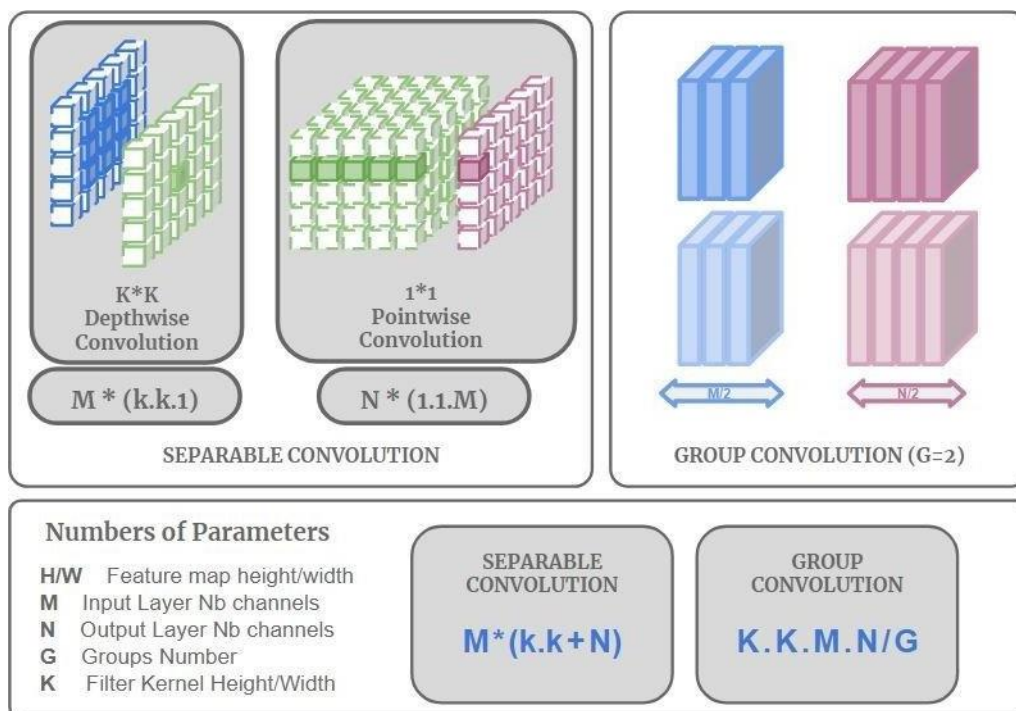
- Converts convolution into **matrix multiplication**
- Uses highly optimized BLAS routines



- ✓ Leverages GPU acceleration
- ✓ Common in deep learning frameworks

6. Depthwise Separable Convolution

- Splits convolution into:
 - Depthwise convolution
 - Pointwise (1×1) convolution



- ✓ Huge reduction in computation
- ✓ Used in MobileNet, EfficientNet

7. Grouped Convolution

- Channels divided into groups
 - Convolution applied separately per group
- ✓ Lower computation
 - ✓ Used in AlexNet, ResNeXt

Comparison of Efficiency Techniques

Method	Speed Gain	Best Use Case
FFT-based	High	Large kernels
Winograd	Very high	3×3 kernels
Im2Col	High	GPU acceleration
Depthwise	Very high	Mobile models
Grouped	Medium	Channel separation

Key Points

- Convolution is the most expensive CNN operation
- FFT and Winograd reduce multiplications
- Im2Col converts convolution to matrix multiplication
- Depthwise separable convolution is most efficient

10. Convolutional Networks and the History of Deep Learning.

The development of **Convolutional Neural Networks (CNNs)** is closely tied to the **overall history of deep learning**. CNNs played a crucial role in reviving neural networks and demonstrating their power in real-world applications, especially computer vision.

Early Inspiration (Biological Roots)

- CNNs were inspired by the **human visual cortex**
- Neurons in the visual cortex respond to **local regions** of the visual field
- This idea led to **local receptive fields** and **hierarchical feature learning**

Early Neural Networks (1G50s-1G80s)

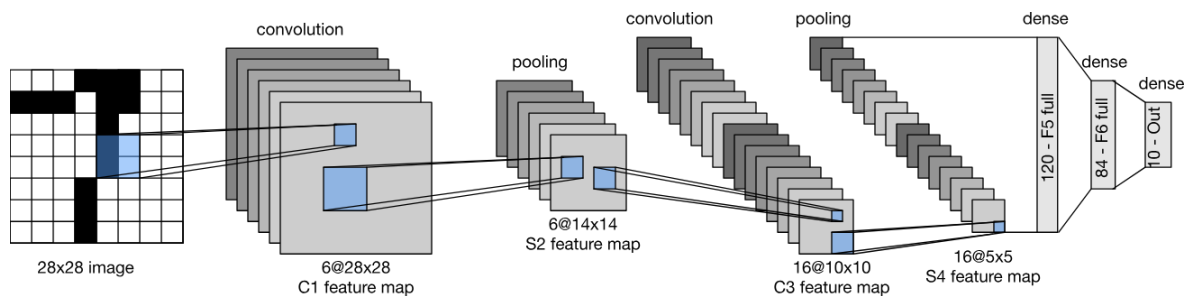
- 1957: **Perceptron** introduced
- Single-layer models were limited
- Could not solve non-linear problems (e.g., XOR)

- Led to reduced interest in neural networks

Birth of Convolutional Networks (1G80s-1GG0s)

Key Milestone: LeNet

- First practical CNN architecture
- Used for **handwritten digit recognition**
- Introduced:
 - Convolution layers
 - Pooling (subsampling)
 - Weight sharing



Despite success, CNNs were:

- Computationally expensive
- Limited by hardware and small datasets

The First AI Winter

- Lack of computing power
- Limited training data
- Slow training algorithms
- CNNs remained mostly academic

Revival of Deep Learning (2006-2010)

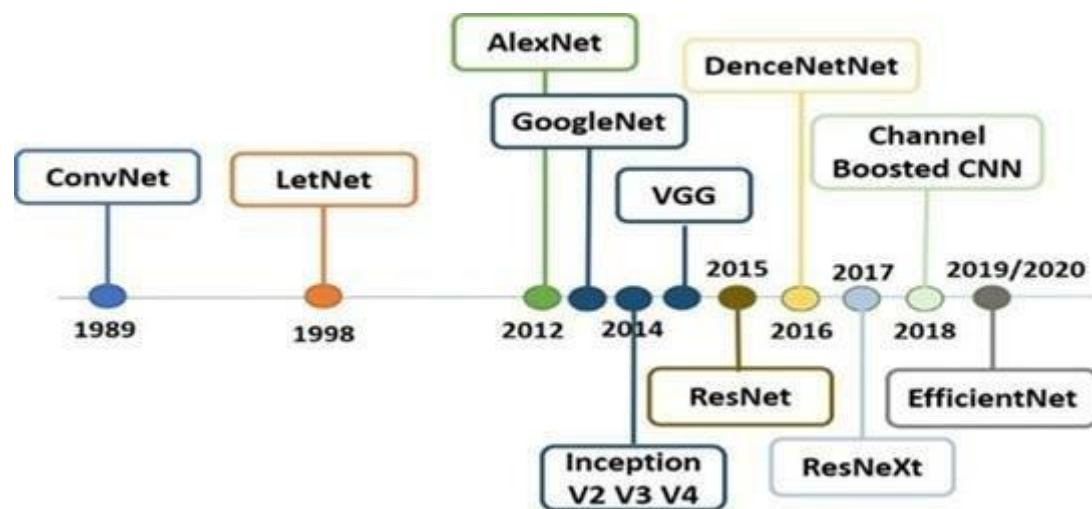
- Improvements in:
 - Back-propagation

- Weight initialization
- Regularization techniques
- Availability of **larger datasets**
- Rise of **GPUs** for parallel computation

Breakthrough Moment (2012) - CNN Revolution

ImageNet Success

- Deep CNNs achieved dramatic accuracy improvements
- Proved deep learning's superiority over traditional methods
- Marked the **rebirth of CNNs**



Rapid Evolution of CNN Architectures

After 2012, CNN research exploded:

- Deeper networks
- Better regularization
- Efficient convolution algorithms
- Specialized architectures for speed and accuracy

Impact on Deep Learning

CNNs:

- Made deep learning practical
- Enabled automatic feature extraction
- Replaced handcrafted features
- Became standard for vision tasks

Applications Enabled by CNNs

- Image classification
- Object detection
- Face recognition
- Medical imaging
- Autonomous driving

Why CNNs Changed Deep Learning

- Strong inductive priors (locality, weight sharing)
- Fewer parameters than fully connected networks
- Excellent generalization on image data
- Scalable with data and compute

Key Points

- CNNs revived interest in neural networks
- LeNet was the first successful CNN
- GPU computing enabled deep CNNs
- CNNs drove the deep learning revolution