Understanding Convolutional Neural Networks (CNNs)

Architecture, Operations, and Applications

Sayan CHAKI

April 9, 2025

Outline

What are Convolutional Neural Networks?

Definition

Convolutional Neural Networks (CNNs) are specialized deep neural networks designed to automatically and adaptively learn spatial hierarchies of features from data with grid-like topology.

- Inspired by the organization of the animal visual cortex
- Specialized for processing data with a known grid-like topology
- Employ a mathematical operation called convolution
- Learn features directly from data with minimal preprocessing

Input $\longrightarrow$ Conv $\longrightarrow$ Pool $\longrightarrow$ FC $\longrightarrow$ Output

Key Advantages of CNNs

Parameter Efficiency

- Weight sharing reduces parameters
- Local connectivity limits connections
- Scales well to high-dimensional inputs

Translation Invariance

- Detect features regardless of position
- Hierarchical feature learning
- Robust to small shifts/distortions

Hierarchical Representation

- Early layers: simple features
- Middle layers: parts of objects
- Deep layers: complete objects

Performance

- State-of-the-art in many visual tasks
- Highly optimized implementations
- Parallelizable operations (GPU-friendly)

Why CNNs Excel in Image Processing

Images Have Strong Spatial Properties:

- Local patterns (textures, edges, shapes) are important
- Neighboring pixels are highly correlated
- Same features appear at different locations
- Hierarchical composition (edges $\rightarrow$ shapes $\rightarrow$ objects)

Applications Beyond Image Processing

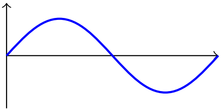
CNNs have been successfully applied to:

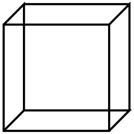
1D CNNs

- Time series analysis
- Audio processing
- Natural language processing
- Genomic sequence analysis

3D CNNs

- Video analysis
- Medical imaging (MRI, CT scans)
- Volumetric data processing
- Point cloud classification





CNNs vs. Traditional MLPs

Feature	**CNNs**	**MLPs**
Connectivity	Local connectivity in convolutional layers	Fully connected between layers
Parameter sharing	Weights shared across input space	Each connection has unique weight
Parameter count	Lower (more efficient)	Higher (grows quickly with input size)
Spatial awareness	Preserves spatial relationships	Flattens spatial information
Hierarchical features	Built-in by design	Must be learned implicitly
Translation invariance	Naturally handles spatial shifts	Must learn from many examples
Input dimensions	Handles high-dimensional data well	Struggles with high-dimensional data

Why CNNs Outperform MLPs for Structured Data

MLP Problem with Images

For a 256×256 RGB image:
Input neurons: $256 \times 256 \times 3 = 196{,}608$
Hidden layer (1000 neurons): $196{,}608 \times 1{,}000 = 196{,}608{,}000$ parameters!

CNN Solution

For the same image with 10 filters of size 5×5:
Parameters per filter: $5 \times 5 \times 3 = 75$
Total parameters: $75 \times 10 = 750$

Result: CNNs require orders of magnitude fewer parameters while better preserving spatial relationships in the data.

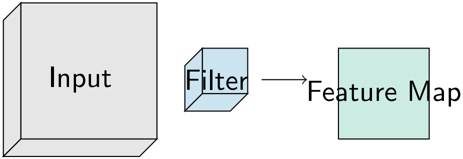
What is a Convolutional Layer?

Definition

A convolutional layer applies a set of learned filters to input data, creating feature maps that represent detected patterns at different locations.

Key Components:

- **Input:** Typically a 3D volume (width $\times$ height $\times$ channels)
- **Filters/Kernels:** Small matrices of learnable weights
- **Feature Maps:** Outputs produced by sliding filters over input
- **Hyperparameters:** Stride, padding, filter size, filter count



Convolutional Layer: Deep Dive

Mathematical Operation:

The convolution operation can be expressed as:

$$(I * K)(i,j) = \sum_m \sum_n I(i+m, j+n) \cdot K(m,n) \tag{1}$$

Where:

- I is the input
- K is the kernel/filter
- $*$ denotes convolution
- (i,j) are coordinates in the output feature map
- (m,n) are coordinates in the kernel

In a CNN Layer:

- Multiple kernels produce multiple feature maps
- Each kernel detects different patterns
- Biases are added to each feature map
- Activation functions introduce non-linearity

Kernels/Filters: The Feature Detectors

Properties of Kernels/Filters:

- Typically small (e.g., 3×3, 5×5, 7×7)
- Depth equals input channel depth
- Weights are learned during training
- Each produces one feature map

Example Filters:

Common Learned Patterns:

- Edges (horizontal, vertical, diagonal)
- Corners and junctions
- Simple textures and blobs
- Color transitions

Horizontal Edge Detector

$$\begin{bmatrix} 1 & 1 & 1 \\ 0 & 0 & 0 \\ -1 & -1 & -1 \end{bmatrix}$$

Vertical Edge Detector

$$\begin{bmatrix} 1 & 0 & -1 \\ 1 & 0 & -1 \\ 1 & 0 & -1 \end{bmatrix}$$

Feature Maps: Capturing Spatial Patterns

Feature Maps:

- Represent the presence of specific features
- Higher values indicate stronger pattern matches
- Preserve spatial relationships from input
- Dimensions depend on input size, filter size, stride, and padding

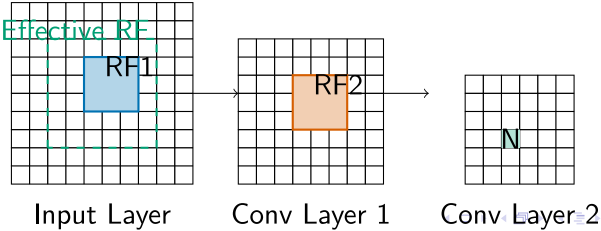
Receptive Fields: What Each Neuron "Sees"

Definition

The receptive field is the region in the input space that affects a particular neuron in a feature map.

Properties:

- Initial receptive field = filter size
- Deeper in the network, receptive fields grow
- Enables hierarchical feature learning
- Early layers: local features; Deep layers: global patterns



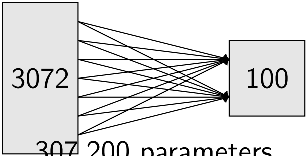
Convolutional vs. Fully Connected Layers

Property	Convolutional Layer	Fully Connected Layer
Weight sharing	Same weights applied throughout input	Each connection has unique weight
Connections	Local (defined by filter size)	Global (connects to all inputs)
Spatial awareness	Preserves spatial structure	No spatial information preserved
Parameters	Scales with filter count and size	Scales with input and output dimensions
Feature learning	Specialized for pattern detection	General feature learning
Position invariance	Built-in for detected features	Must be explicitly learned
Output structure	2D feature maps (multi-channel)	1D vectors

Parameter Efficiency in Convolutional Layers

Fully Connected Layer:

- Input: $32 \times 32 \times 3$ image
- Output: 100 neurons
- Parameters: $32 \times 32 \times 3 \times 100 = 307{,}200$



307,200 parameters

Convolutional Layer:

- Input: $32 \times 32 \times 3$ image
- 10 filters of size $5 \times 5 \times 3$
- Parameters: $5 \times 5 \times 3 \times 10 + 10$ (biases) $= 760$

403× fewer parameters!

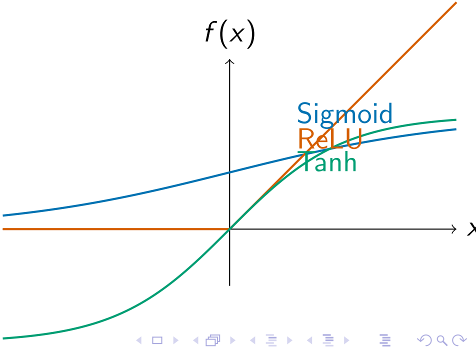
Common Activation Functions in Neural Networks

Purpose of Activation Functions

Activation functions introduce non-linearity into neural networks, allowing them to learn complex patterns beyond simple linear combinations.

Common Activation Functions:

- **Sigmoid:** $\sigma(x) = \frac{1}{1+e^{-x}}$
- **Tanh:** $\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$
- **ReLU:** $f(x) = \max(0, x)$
- **Leaky ReLU:** $f(x) = \max(\alpha x, x)$
- **PReLU:** Parametric ReLU
- **ELU:** Exponential Linear Unit



ReLU: The Preferred Activation in CNNs

ReLU (Rectified Linear Unit)

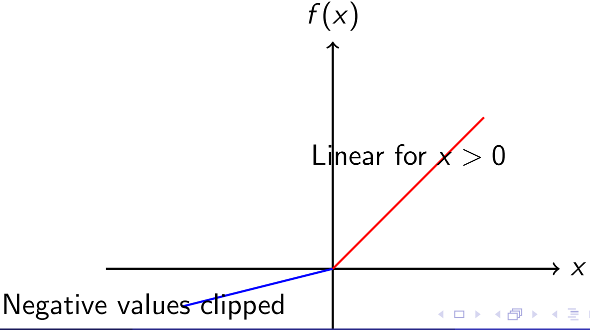
$$f(x) = \max(0, x)$$

ReLU: The Preferred Activation in CNNs

ReLU (Rectified Linear Unit)

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

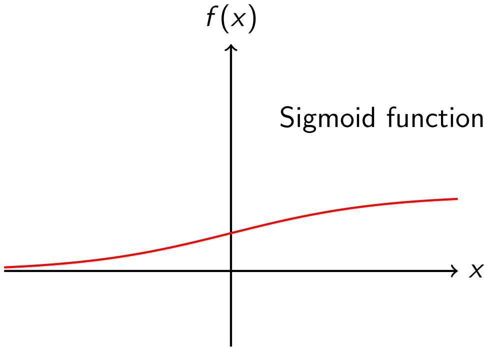
- Common activation functions in CNNs: ReLU, LeakyReLU, ELU.
- ReLU prevents vanishing gradients unlike sigmoid/tanh.
- Computationally efficient compared to non-linear functions.



Vanishing Gradient Problem in CNNs

- Occurs when gradients become too small during backpropagation.
- Sigmoid and tanh activation functions exacerbate this issue.
- Gradients diminish as they propagate through deep layers.

$$\sigma(x) = \frac{1}{1+e^{-x}}, \quad \sigma'(x) = \sigma(x)(1-\sigma(x)) \tag{2}$$



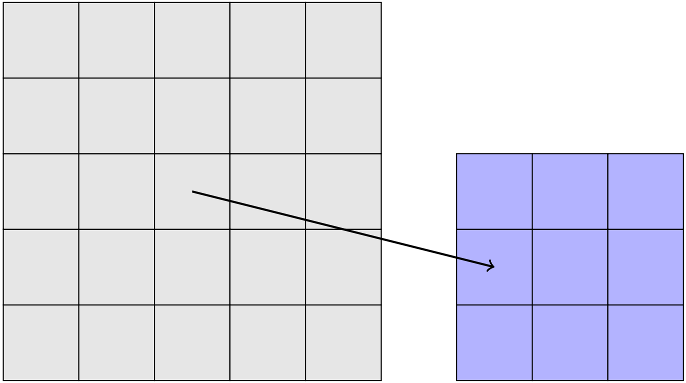
Frame Title

$$\frac{\partial L}{\partial w} = \prod_{i=1}^{n} \sigma'(x_i) \tag{3}$$

- **Why it happens:** When using activation functions like sigmoid or tanh, their derivatives are small (especially for large or small inputs), causing gradient magnitudes to shrink exponentially.
- **Effect:** Deep layers learn much slower, affecting training convergence.
- **Solutions:**
  - **ReLU:** Helps maintain gradient magnitudes for positive values.
  - **Batch Normalization:** Normalizes activations to stabilize learning.
  - **Residual Connections:** Skip connections allow gradients to flow more easily.
  - **Careful Weight Initialization:** Xavier/Glorot and He initialization help prevent gradient shrinkage.

Forward Pass in CNNs

- Input image undergoes convolution with a filter.
- Example: 3x3 kernel applied to a 5x5 input.
- Activation function is applied to the feature map.
- Repeated for multiple layers, producing hierarchical features.



Input Image (5x5)

Feature Map (3x3)

Introduction

- The vanishing gradient problem occurs in deep networks when gradients shrink exponentially during backpropagation.
- This leads to slow learning, poor feature extraction, and ineffective weight updates.

Backpropagation and Gradients

The weight update rule is given by:

$$W^{(l)} \leftarrow W^{(l)} - \eta \frac{\partial \mathcal{L}}{\partial W^{(l)}} \tag{5}$$

The gradient of the loss function propagates backward as:

$$\frac{\partial \mathcal{L}}{\partial W^{(l)}} = \delta^{(l)} h^{(l-1)} \tag{6}$$

where:

$$\delta^{(l)} = (W^{(l+1)})^T \delta^{(l+1)} \odot f'(z^{(l)}) \tag{7}$$

Small derivatives in deep networks cause the gradients to vanish.

Why Do Gradients Vanish?

Sigmoid Activation Function:

$$\sigma(x) = \frac{1}{1 + e^{-x}} \tag{8}$$

$$\sigma'(x) = \sigma(x)(1 - \sigma(x)) \tag{9}$$

For large $|x|$, $\sigma'(x) \approx 0$.

Impact on CNNs

- Shallow layers receive minimal updates, leading to ineffective feature extraction.
- Training deep networks (e.g., VGG, AlexNet) becomes extremely slow.

Better Activation Functions

- **ReLU:** $f(x) = \max(0, x)$
- **Leaky ReLU:** $f(x) = \max(0.01x, x)$
- **ELU:** $f(x) = \begin{cases} x, & x > 0 \\ \alpha(e^x - 1), & x \leq 0 \end{cases}$

Weight Initialization Techniques

- **Xavier Initialization:** $W \sim \mathcal{N}(0, \frac{1}{n})$
- **He Initialization:** $W \sim \mathcal{N}(0, \frac{2}{n})$

Batch Normalization and ResNets

Batch Normalization:

$$\hat{x} = \frac{x - \mu}{\sigma}, \quad y = \gamma \hat{x} + \beta \tag{10}$$

Residual Networks (ResNets):

$$y^{(l+1)} = f(W^{(l)} y^{(l)}) + y^{(l)} \tag{11}$$

Skip connections allow gradients to flow directly, solving vanishing gradient issues.

Conclusion

- Vanishing gradients hinder training in deep networks.
- Solutions include better activations, weight initialization, batch normalization, and ResNets.

Outline

What are Convolutional Neural Networks?

- Deep learning architecture specialized for processing grid-like data
- Inspired by the visual cortex of animals
- Key innovation: local connectivity patterns through convolutional layers
- Significantly reduces parameters compared to fully-connected networks
- Excellent for image recognition, video analysis, and other vision tasks

Key Components of CNNs

- Convolutional layers
- Pooling layers
- Activation functions (ReLU, etc.)
- Fully connected layers
- Batch normalization
- Dropout for regularization

2D Convolution Operation

The 2D convolution operation is defined as:

$$(I * K)(i,j) = \sum_m \sum_n I(i+m, j+n) \cdot K(m,n) \tag{12}$$

Where:

- I is the input (e.g., image)
- K is the kernel/filter
- (i,j) are the coordinates in the output feature map
- (m,n) are the coordinates in the kernel

Feature Maps

- Each convolution with a different filter produces a feature map
- Feature maps capture different aspects of the input (edges, textures, etc.)
- Output dimensions:

$$W_{out} = \frac{W_{in} - F + 2P}{S} + 1 \tag{13}$$

$$H_{out} = \frac{H_{in} - F + 2P}{S} + 1 \tag{14}$$

- Where W and H are width and height, F is filter size, P is padding, and S is stride

Basic CNN Architecture

1. Input Layer: Raw image pixels
2. Convolutional Layers: Apply filters to detect features
3. Activation Layers: Apply non-linearity (typically ReLU)
4. Pooling Layers: Reduce spatial dimensions
5. Fully Connected Layers: Final classification
6. Output Layer: Probabilities for each class

Popular CNN Architectures

- LeNet-5 (1998)
- AlexNet (2012)
- VGGNet (2014)
- GoogLeNet/Inception (2014)
- ResNet (2015)
- MobileNet (2017)
- EfficientNet (2019)

Architecture of AlexNet

- Consists of 8 layers: 5 convolutional layers and 3 fully connected layers.
- Uses ReLU activation instead of sigmoid/tanh to mitigate vanishing gradients.
- Includes overlapping max pooling for better feature extraction.
- Implements dropout in fully connected layers to prevent overfitting.

Impact of AlexNet on Deep Learning

- Showed that large datasets and computational power could enable deep networks.
- Inspired further architectures like VGG, ResNet, and GoogLeNet.

Conclusion

- Vanishing gradients hinder training in deep networks.
- Solutions include better activations, weight initialization, batch normalization, and ResNets.
- AlexNet played a crucial role in advancing deep learning research.

Loss Functions in CNNs: Guiding the Learning Process

Common Loss Functions:

- **Cross-Entropy Loss**
  - $L = -\sum_i y_i \log(\hat{y}_i)$
  - Ideal for multi-class classification tasks
- **Binary Cross-Entropy (BCE)**
  - $L = -\sum_i [y_i \log(\hat{y}_i) + (1 - y_i) \log(1 - \hat{y}_i)]$
  - Used for binary classification
  - Penalizes incorrect confidence
- **Focal Loss**
  - $L = -\sum_i (1 - \hat{y}_i)^\gamma y_i \log(\hat{y}_i)$
  - Addresses class imbalance

Key aspects:

- Cross-Entropy generalizes to multiple classes
- Binary Cross-Entropy is tailored for two-class problems
- Both are differentiable and optimize classification performance

"The loss function defines what your network is ultimately trying to achieve"

Batch Normalization: Stabilizing Training

Why BatchNorm?

- Reduces internal covariate shift
- Speeds up training convergence
- Helps gradient flow in deep networks

Computation:

$$\hat{x}_i = \frac{x_i - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}}, \quad y_i = \gamma \hat{x}_i + \beta \tag{15}$$

Key Benefits:

- Normalizes intermediate activations
- Adds learnable scale and shift parameters (γ, β)
- Reduces sensitivity to initialization

Group Normalization: Alternative to BatchNorm

Why GroupNorm?

- Works well with small batch sizes
- Normalizes within groups of channels instead of across batches

Computation:

$$\hat{x}_i = \frac{x_i - \mu_G}{\sqrt{\sigma_G^2 + \epsilon}}, \quad y_i = \gamma \hat{x}_i + \beta \tag{16}$$

Key Differences from BatchNorm:

- Independent of batch size
- Useful in memory-constrained environments

Pooling Methods in CNNs

Common Pooling Techniques:

- **Max Pooling:** Retains the highest activation in a region
- **Average Pooling:** Computes the mean activation in a region
- **Global Pooling:** Reduces entire feature map to a single value

Why Pooling?

- Reduces spatial dimensions
- Provides translational invariance
- Helps control overfitting