Convolutional Neural Networks (CNNs)

Ankit Kumar

February 4, 2026

1 Introduction to CNNs

1.1 What is a CNN?

A **Convolutional Neural Network (CNN)** is a specialized deep learning architecture designed for processing **grid-like data**, particularly images. Unlike traditional neural networks that treat input as flat vectors, CNNs preserve spatial relationships.

1.2 Why Not Use Regular Neural Networks (ANNs) for Images?

- **Huge Number of Parameters**: For a 32x32 RGB image:

  $$32 \times 32 \times 3 = 3,072 \text{ input neurons}$$

  If connected to just 100 neurons in next layer:

  $$3,072 \times 100 = 307,200 \text{ weights!}$$

  This leads to computational explosion.

- **Loss of Spatial Information**: Flattening destroys local patterns (edges, textures).

- **Overfitting**: Too many parameters $\rightarrow$ poor generalization.

2 CNN Architecture Overview

A typical CNN has this structure:

Input Image $\rightarrow$ Convolution Layers $\rightarrow$ Pooling Layers $\rightarrow$ More Conv/Pool $\rightarrow$ Flatten $\rightarrow$ Fully Connected $\rightarrow$ Output

3 Convolution Layer: The Feature Detector

3.1 What is Convolution?

Imagine sliding a small magnifying glass (filter/kernel) over an image to detect patterns.

3.2 Mathematical Definition

For 2D convolution:

$$(I * K)[i,j] = \sum_m \sum_n I[i-m, j-n] \cdot K[m,n]$$

Where:

- I = Input image (matrix)
- K = Filter/kernel (small matrix)
- $*$ = Convolution operation

3.3 Example Calculation

Let's convolve a 4×4 image with a 3×3 filter:

Image I:

$$\begin{bmatrix} 1 & 0 & 1 & 2 \\ 0 & 2 & 1 & 0 \\ 1 & 1 & 0 & 1 \\ 2 & 0 & 1 & 1 \end{bmatrix}$$

Filter K (edge detector):

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

Step 1: Position filter at top-left:

$$\begin{bmatrix} 1 & 0 & 1 \\ 0 & 2 & 1 \\ 1 & 1 & 0 \end{bmatrix} \circ \begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{bmatrix}$$

$$= (1 \times -1) + (0 \times 0) + (1 \times 1) + (0 \times -1) + (2 \times 0) + (1 \times 1) + (1 \times -1) + (1 \times 0) + (0 \times 1)$$

$$= -1 + 0 + 1 + 0 + 0 + 1 - 1 + 0 + 0 = 0$$

Step 2: Slide right and continue. Final output (2×2):

$$\begin{bmatrix} 0 & 3 \\ 2 & -1 \end{bmatrix}$$

3.4 Output Size Formula

Given:

- Input size: $W \times H$
- Filter size: $F \times F$
- Padding: P (extra pixels around image)
- Stride: S (how many pixels to jump)

$$\text{Output Width} = \frac{W - F + 2P}{S} + 1$$
$$\text{Output Height} = \frac{H - F + 2P}{S} + 1$$

Example: Input 6×6, Filter 3×3, Padding 0, Stride 1:

$$\frac{6-3+0}{1} + 1 = 4 \quad \Rightarrow \quad 4 \times 4 \text{ output}$$

Example: Input 6×6, Filter 3×3, Padding 0, Stride 2:

$$\frac{6-3+0}{2} + 1 = 2.5 \to 2 \quad \Rightarrow \quad 2 \times 2 \text{ output}$$

4 Pooling Layer: The Downsampler

4.1 Why Pooling?

1. **Reduce size** (faster computation)
2. **Make features translation invariant**
3. **Prevent overfitting**

4.2 Types of Pooling

4.2.1 Max Pooling (Most Common)

Takes maximum value in each window.

Example: 4×4 input, 2×2 window, stride 2:

$$\begin{bmatrix} 1 & 3 & 2 & 9 \\ 4 & 6 & 1 & 5 \\ 7 & 2 & 8 & 3 \\ 0 & 4 & 3 & 6 \end{bmatrix} \to \begin{bmatrix} 6 & 9 \\ 7 & 8 \end{bmatrix}$$

Explanation:

- Window 1 (top-left): max(1,3,4,6) = 6
- Window 2 (top-right): max(2,9,1,5) = 9
- Window 3 (bottom-left): max(7,2,0,4) = 7
- Window 4 (bottom-right): max(8,3,3,6) = 8

4.2.2 Average Pooling

Takes average value in each window.

Same example:

$$\begin{bmatrix} 1 & 3 & 2 & 9 \\ 4 & 6 & 1 & 5 \\ 7 & 2 & 8 & 3 \\ 0 & 4 & 3 & 6 \end{bmatrix} \to \begin{bmatrix} (1+3+4+6)/4 & (2+9+1+5)/4 \\ (7+2+0+4)/4 & (8+3+3+6)/4 \end{bmatrix} = \begin{bmatrix} 3.5 & 4.25 \\ 3.25 & 5.0 \end{bmatrix}$$

4.2.3 Min Pooling (Rare)

Takes minimum value in each window.

5 Complete CNN Example: Step-by-Step

5.1 Input Image

RGB image: 32×32×3 (32 pixels tall, 32 wide, 3 color channels)

5.2 Layer 1: Convolution

- 32 filters of size 3×3×3
- Stride = 1, Padding = 1
- Output size calculation:

$$\frac{32 - 3 + 2 \times 1}{1} + 1 = 32 \quad \Rightarrow \quad 32 \times 32 \times 32$$

  (32 feature maps, each 32×32)

5.3 Layer 2: Max Pooling

- Window: 2×2
- Stride = 2
- Output size:

$$\frac{32}{2} = 16 \quad \Rightarrow \quad 16 \times 16 \times 32$$

5.4 Layer 3: Convolution

- 64 filters of size 3×3×32
- Stride = 1, Padding = 1
- Output: 16×16×64

5.5 Layer 4: Max Pooling

- Window: 2×2, Stride = 2
- Output: 8×8×64

5.6 Layer 5: Flatten

Convert 3D tensor to 1D vector:

$$8 \times 8 \times 64 = 4,096 \text{ neurons}$$

5.7 Layer 6: Fully Connected

- 4,096 $\rightarrow$ 128 neurons
- Uses ReLU activation

5.8 Layer 7: Output Layer

- 128 $\rightarrow$ 10 neurons (for 10 classes)
- Uses Softmax activation

6 Activation Functions

6.1 ReLU (Rectified Linear Unit)

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

- Pros: Simple, fast, prevents vanishing gradient
- Cons: "Dying ReLU" problem

6.2 Softmax (for output layer)

Converts scores to probabilities:

$$P(\text{class } i) = \frac{e^{z_i}}{\sum_{j=1}^{C} e^{z_j}}$$

Where C = number of classes.

Example: Scores = [2.0, 1.0, 0.1]

$$\text{Sum} = e^{2.0} + e^{1.0} + e^{0.1} = 7.39 + 2.72 + 1.11 = 11.22$$

$$P(\text{class } 1) = \frac{7.39}{11.22} = 0.66$$

$$P(\text{class } 2) = \frac{2.72}{11.22} = 0.24$$

$$P(\text{class } 3) = \frac{1.11}{11.22} = 0.10$$

7 Training a CNN

7.1 Loss Function: Cross-Entropy

For classification:

$$L = -\sum_{i=1}^{C} y_i \log(\hat{y}_i)$$

Where:

- y_i = true label (1 for correct class, 0 others)
- $\hat{y}_i$ = predicted probability

7.2 Backpropagation in CNNs

The chain rule applied through:

1. Output layer gradients
2. Fully connected layers
3. Pooling layers (pass gradients to max position)
4. Convolution layers (update filter weights)

7.3 Weight Update

Using gradient descent:

$$W_{\text{new}} = W_{\text{old}} - \eta \frac{\partial L}{\partial W}$$

Where η = learning rate.

8 Advanced CNN Concepts

8.1 1×1 Convolutions

- Used for dimensionality reduction
- Example: 32×32×256 → 32×32×64 using 64 filters of size 1×1×256
- Reduces computation cost

8.2 Dilated Convolutions

- Increases receptive field without pooling
- Has ”holes” in filter
- Used in segmentation tasks

8.3 Batch Normalization

Normalizes layer inputs:

$$\hat{x} = \frac{x - \mu}{\sqrt{\sigma^2 + \epsilon}}$$

Reduces internal covariate shift, speeds training.

9 Practical Example: Classifying Cats vs Dogs

1. **Input**: $128\times128\times3$ (RGB image)
2. **Conv1**: 32 filters, $3\times3 \rightarrow 128\times128\times32$
3. **Pool1**: 2×2 max pool $\rightarrow 64\times64\times32$
4. **Conv2**: 64 filters, $3\times3 \rightarrow 64\times64\times64$
5. **Pool2**: 2×2 max pool $\rightarrow 32\times32\times64$
6. **Conv3**: 128 filters, $3\times3 \rightarrow 32\times32\times128$
7. **Pool3**: 2×2 max pool $\rightarrow 16\times16\times128$
8. **Flatten**: $16\times16\times128 = 32{,}768$ neurons
9. **FC1**: $32{,}768 \rightarrow 512$ (ReLU)
10. **FC2**: $512 \rightarrow 2$ (Softmax)

10 Common CNN Architectures

10.1 LeNet-5 (1998)

- First successful CNN
- For digit recognition
- 7 layers total

10.2 AlexNet (2012)

- Won ImageNet competition
- 8 layers, ReLU activation
- Dropout for regularization

10.3 VGG-16 (2014)

- Simple: only 3×3 convolutions
- 16 layers deep
- 138 million parameters

10.4 ResNet (2015)

- Residual connections
- Solves vanishing gradient in deep networks
- Up to 152 layers

11 Mathematics Deep Dive

11.1 Convolution as Matrix Multiplication

A convolution can be expressed as matrix multiplication using the **im2col** operation:
Example: 3×3 input, 2×2 filter:

$$\begin{bmatrix} x_{11} & x_{12} & x_{13} \\ x_{21} & x_{22} & x_{23} \\ x_{31} & x_{32} & x_{33} \end{bmatrix} \rightarrow \begin{bmatrix} x_{11} & x_{12} & x_{21} & x_{22} \\ x_{12} & x_{13} & x_{22} & x_{23} \\ x_{21} & x_{22} & x_{31} & x_{32} \\ x_{22} & x_{23} & x_{32} & x_{33} \end{bmatrix}$$

Filter weights as column vector:

$$\begin{bmatrix} w_{11} & w_{12} \\ w_{21} & w_{22} \end{bmatrix} \rightarrow \begin{bmatrix} w_{11} \\ w_{12} \\ w_{21} \\ w_{22} \end{bmatrix}$$

Convolution = Matrix multiplication:

$$\text{Output} = \text{im2col}(X) \times W$$

11.2 Gradient Calculation for Convolution

For a convolution layer:

$$\frac{\partial L}{\partial K} = \text{convolution}\left(\frac{\partial L}{\partial O}, I\right)$$

Where:

- K = filter weights
- O = output
- I = input

12 Implementation Tips

1. **Always normalize inputs**: Scale pixels to [0,1] or [-1,1]
2. **Use data augmentation**: Rotation, flipping, zooming
3. **Start with small filters**: 3×3 is standard
4. **Increase filters as network deepens**: 32 → 64 → 128 → 256
5. **Use dropout**: Prevents overfitting
6. **Monitor training/validation loss**: Stop when validation loss increases

13 Summary

- **CNN** = Feature extraction + Classification
- **Convolution** = Pattern detection with shared weights
- **Pooling** = Downsampling for translation invariance
- **Training** = Minimize loss via backpropagation
- **Key advantage**: Parameter sharing → efficient for images

14 Exercises

14.1 Exercise 1: Size Calculation

Calculate output sizes:

1. Input: 28×28×1, Filter: 5×5, Stride: 1, Padding: 0
2. Input: 100×100×3, Filter: 3×3, Stride: 2, Padding: 1
3. Input: 14×14×64 after 2×2 max pooling with stride 2

14.2 Exercise 2: Convolution Calculation

Perform convolution:

$$I = \begin{bmatrix} 2 & 4 & 1 \\ 3 & 1 & 5 \\ 0 & 2 & 1 \end{bmatrix}, \quad K = \begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix}$$

Stride = 1, no padding.

14.3 Exercise 3: Pooling

Apply max pooling (2×2, stride 2):

$$\begin{bmatrix} 3 & 8 & 2 & 4 \\ 1 & 6 & 5 & 3 \\ 7 & 2 & 9 & 1 \\ 4 & 5 & 3 & 6 \end{bmatrix}$$

Answers

Exercise 1

1. $\frac{28-5}{1} + 1 = 24 \rightarrow 24 \times 24$
2. $\frac{100-3+2}{2} + 1 = 50 \rightarrow 50 \times 50 \times ?$ (depends on filters)
3. $\frac{14}{2} = 7 \rightarrow 7 \times 7 \times 64$

Exercise 2

$$\begin{bmatrix} (2\times 0)+(4\times 1)+(3\times -1)+(1\times 0) & (4\times 0)+(1\times 1)+(1\times -1)+(5\times 0) \\ (3\times 0)+(1\times 1)+(0\times -1)+(2\times 0) & (1\times 0)+(5\times 1)+(2\times -1)+(1\times 0) \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 1 & 3 \end{bmatrix}$$

Exercise 3

$$\begin{bmatrix} \max(3,8,1,6) & \max(2,4,5,3) \\ \max(7,2,4,5) & \max(9,1,3,6) \end{bmatrix} = \begin{bmatrix} 8 & 5 \\ 7 & 9 \end{bmatrix}$$