# Neural Networks Class 5

## Introduction to Convolutional Neural Networks

**Learning Objectives:**

- Understand CNN motivation and limitations of MLPs for images
- Master convolution operation mathematics
- Learn CNN building blocks and parameter calculations

**Duration:** 50 minutes

# Motivation: Why CNNs?

## Limitations of MLPs for Images

**Parameter Explosion Problem:**

- **32×32 RGB image:** 3,072 input features
- **Hidden layer (1000 neurons):** 3,072,000 parameters
- **224×224 RGB image:** 150,528 input features
- **Hidden layer (1000 neurons):** 150,528,000 parameters!

**Mathematical Challenge:**

For image of size $H \times W \times C$ with $N$ hidden neurons:

**Parameters =** $H \times W \times C \times N$

**Real Example:** ImageNet images (224×224×3) with 4096 hidden units

# Motivation: Key Problems with MLPs

**Translation Invariance Problem:**

- Cat in top-left corner vs bottom-right corner
- MLP treats these as completely different patterns
- No weight sharing across spatial locations

**Local Feature Detection:**

- **Edges, corners, textures** are local patterns, MLP uses global connections
- Cannot efficiently detect local spatial patterns

**Spatial Structure Ignored:**

- MLP flattens 2D image to 1D vector , **Loses spatial relationships** between pixels
- Adjacent pixels treated same as distant pixels

# Biological Inspiration

**Hubel & Wiesel (1962) Nobel Prize:**

- **Simple cells:** Respond to edges/lines in specific orientations
- **Complex cells:** Respond to patterns regardless of exact position
- **Hierarchical processing:** Simple → Complex → Higher-level features

**Key Insights:**

1. **Local receptive fields:** Neurons respond to small image regions
2. **Feature detectors:** Specialized neurons for specific patterns
3. **Hierarchical representation:** Low-level → High-level features

# Biological Inspiration

**CNN Design Principles:**

- **Local connectivity:** Mimic receptive fields
- **Parameter sharing:** Same detector at all locations
- **Hierarchical learning:** Build complex features from simple ones

# Convolution Operation

### 1D Continuous Convolution:

$$(f * g)(t) = \int_{-\infty}^{\infty} f(\tau)g(t-\tau)d\tau$$

### 2D Continuous Convolution:

$$(f * g)(x, y) = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} f(u, v)g(x-u, y-v)dudv$$

### Physical Interpretation:

- $f$: Input signal/image
- $g$: Filter/kernel (impulse response), - $f * g$: Filtered output

**Key Property:** Convolution is **commutative** $f * g = g * f$

# Discrete Convolution for Images

## The CNN Operation

### 2D Discrete Convolution:

$$(I * K)(i,j) = \sum_{u=-a}^{a} \sum_{v=-b}^{b} I(i+u, j+v) K(u,v)$$

### More Common Form (Cross-correlation):

$$(I * K)(i,j) = \sum_{u=-a}^{a} \sum_{v=-b}^{b} I(i-u, j-v) K(u,v)$$

# Discrete Convolution for Images

## The CNN Operation

### Practical Implementation:

$$(I * K)(i,j) = \sum_{u=0}^{k-1} \sum_{v=0}^{k-1} I(i+u, j+v) K(u,v)$$

### Notation:

- $I$: Input image (or feature map)
- $K$: Kernel/filter
- $(i, j)$: Output position

# Convolution Example

## Step-by-Step Calculation

**Input Image $I$ (4×4):**

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

**Kernel $K$ (3×3):**

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

# Convolution Example

## Step-by-Step Calculation

### Output at position (0,0):

$$(I * K)(0,0) = \sum_{u=0}^{2} \sum_{v=0}^{2} I(u,v) K(u,v)$$

$$= 1\times1 + 0\times0 + 1\times(-1) + 2\times1 + 1\times0 + 0\times(-1) + 1\times1 + 2\times0 + 1\times(-1) = 2$$

# Kernel Types and Effects

## Edge Detection and Feature Extraction

### Vertical Edge Detection:

$$K_{\text{vertical}} = \begin{bmatrix} 1 & 0 & -1 \\ 1 & 0 & -1 \\ 1 & 0 & -1 \end{bmatrix}$$

### Horizontal Edge Detection:

$$K_{\text{horizontal}} = \begin{bmatrix} 1 & 1 & 1 \\ 0 & 0 & 0 \\ -1 & -1 & -1 \end{bmatrix}$$

# Kernel Types and Effects

## Edge Detection and Feature Extraction

### Gaussian Blur:

$$K_{\text{blur}} = \frac{1}{16}\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$$

### Sharpening:

$$K_{\text{sharpen}} = \begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$$

# Stride and Padding

## Controlling Output Size

**Stride $s$:** Step size for kernel movement

- **Stride = 1:** Kernel moves 1 pixel at a time
- **Stride = 2:** Kernel moves 2 pixels, reducing output size

**Padding $p$:** Adding zeros around input border

- **Valid padding:** No padding ($p = 0$)
- **Same padding:** Padding to keep same size
- **Full padding:** Maximum padding

# Stride and Padding

## Controlling Output Size

### Mathematical Effect:

- **Without padding:** Output size decreases
- **With padding:** Can maintain or control output size

# Output Size Formula

## The Fundamental CNN Equation

**For square input and kernel:**

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

Where:

- $W$: Input width/height
- $F$: Filter/kernel size
- $P$: Padding
- $S$: Stride

# Output Size Formula

## The Fundamental CNN Equation

### Example Calculations:

- **Input:** 32×32, **Filter:** 5×5, **Padding:** 0, **Stride:** 1
  - Output: $\frac{32-5+0}{1} + 1 = 28 \times 28$
- **Input:** 32×32, **Filter:** 5×5, **Padding:** 2, **Stride:** 1
  - Output: $\frac{32-5+4}{1} + 1 = 32 \times 32$

# Multi-Channel Convolution

## RGB Images and Feature Maps

### 3-Channel Input (RGB):

Input: $I \in \mathbb{R}^{H \times W \times 3}$

Kernel: $K \in \mathbb{R}^{F \times F \times 3}$

### Convolution Across Channels:

$$(I * K)(i, j) = \sum_{c=0}^{2} \sum_{u=0}^{F-1} \sum_{v=0}^{F-1} I(i+u, j+v, c) \cdot K(u, v, c)$$

# Multi-Channel Convolution

## RGB Images and Feature Maps

### Multiple Filters:

- $N$ filters: $K_1, K_2, \ldots, K_N$
- Output: $N$ feature maps
- Each filter learns different features

### Output Dimension:

$$\text{Output} \in \mathbb{R}^{H' \times W' \times N}$$

# Convolutional Layer

**Complete Convolutional Layer:**

$$\mathbf{Y}_{i,j,k} = \sigma\left(\sum_{c=1}^{C}\sum_{u=1}^{F}\sum_{v=1}^{F}\mathbf{W}_{u,v,c,k}\mathbf{X}_{i+u-1,j+v-1,c} + b_k\right)$$

**Notation:**

- $\mathbf{X} \in \mathbb{R}^{H\times W\times C}$: Input volume, $\mathbf{W} \in \mathbb{R}^{F\times F\times C\times K}$: Filter weights
- $\mathbf{Y} \in \mathbb{R}^{H'\times W'\times K}$: Output volume
- $b_k$: Bias for filter $k$, $\sigma(\cdot)$: Non-linear activation (usually ReLU)

**Parameter Count:**

$$\text{Parameters} = F \times F \times C \times K + K$$

# Parameter Sharing

## CNN vs MLP Comparison

### MLP (Fully Connected):

- **Unique weight** for each input-output connection
- **No spatial structure** exploitation
- **Parameter count:** Input_size × Hidden_size

### CNN (Convolutional):

- **Shared weights** across spatial locations
- **Translation equivariance:** $f(T(x)) = T(f(x))$
- **Parameter count:** Filter_size² × Channels × Num_filters

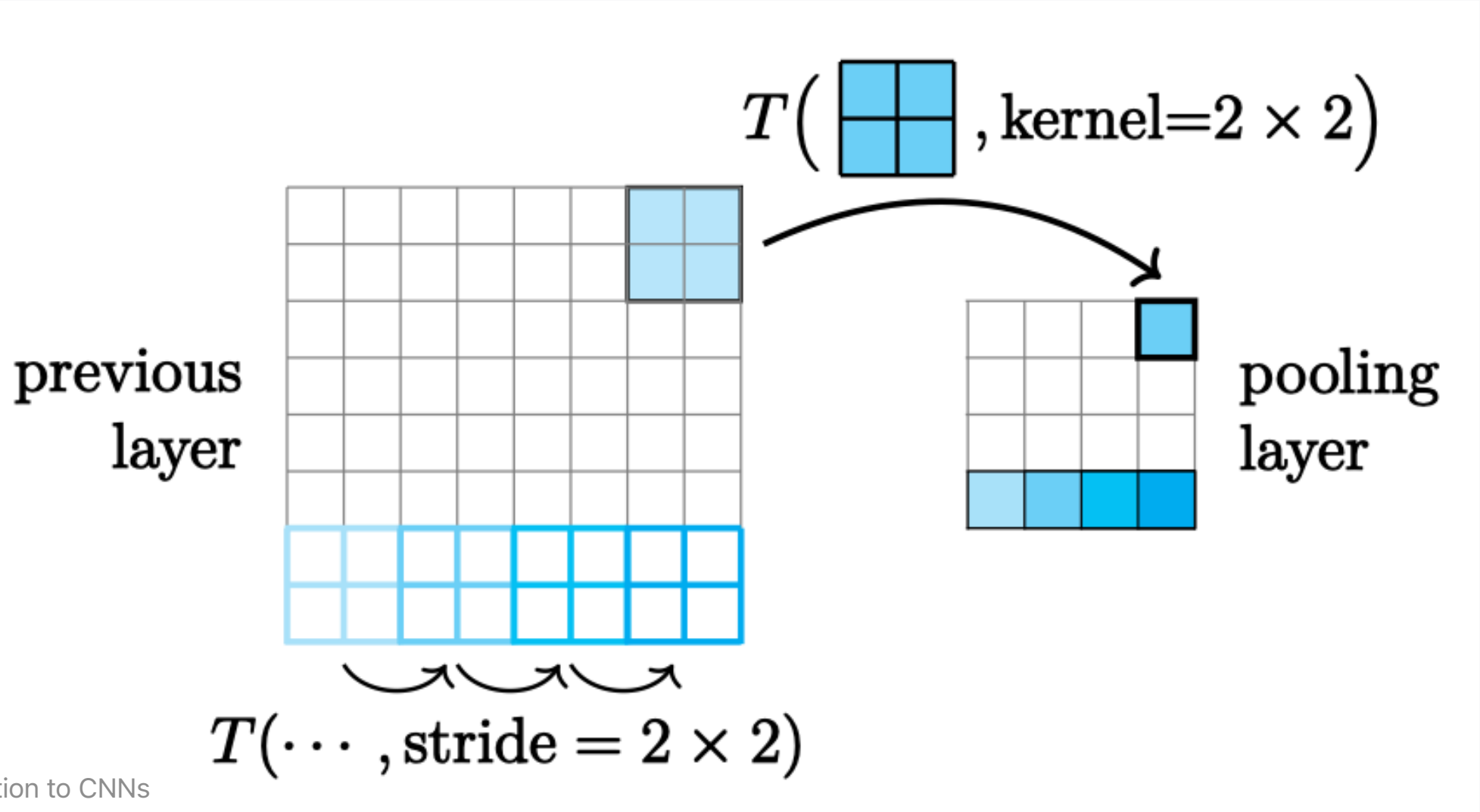
# Parameter Sharing

## CNN vs MLP Comparison

### Example Comparison:

- **28×28 grayscale image, 32 filters of size 5×5**
- **MLP:** 784 × 32 = 25,088 parameters
- **CNN:** 5 × 5 × 1 × 32 + 32 = 832 parameters
- **Reduction factor:** 30×!

# Pooling Operations



# Pooling Operations

## Spatial Downsampling

### Max Pooling:

$$\text{MaxPool}(\mathbf{X})(i,j) = \max_{u,v \in \text{window}} \mathbf{X}(i \cdot s + u, j \cdot s + v)$$

### Average Pooling:

$$\text{AvgPool}(\mathbf{X})(i,j) = \frac{1}{k^2} \sum_{u,v \in \text{window}} \mathbf{X}(i \cdot s + u, j \cdot s + v)$$

# Pooling Operations

## Spatial Downsampling

**Properties:**

- **Translation invariance:** Small shifts don't affect output
- **Dimensionality reduction:** Reduces spatial size
- **No learnable parameters**
- **Preserve depth:** Channel dimension unchanged

**Common Configuration:** 2×2 pooling with stride 2 → 50% size reduction

# Pooling Mathematical Effects

## Information Preservation and Loss

### Max Pooling Properties:

- **Preserves** strongest activations
- **Provides translation invariance**
- **Introduces** non-linearity
- **Loses** spatial precision

# Pooling Mathematical Effects

## Information Preservation and Loss

### Mathematical Analysis:

For 2×2 max pooling with stride 2:

- **Input:** $\mathbb{R}^{H \times W \times C}$
- **Output:** $\mathbb{R}^{H/2 \times W/2 \times C}$
- **Information loss:** 75% of spatial information discarded

### Trade-off:

- **Robustness** vs **Spatial precision**
- **Translation invariance** vs **Fine-grained localization**

# CNN Building Block Assembly

## Typical Layer Combinations

### Basic CNN Block:

$$\text{Input} \rightarrow \text{Conv} \rightarrow \text{ReLU} \rightarrow \text{Pool} \rightarrow \text{Output}$$

### Mathematical Flow:

1. **Convolution:** $\mathbf{Z} = \mathbf{W} * \mathbf{X} + \mathbf{b}$
2. **Activation:** $\mathbf{A} = \text{ReLU}(\mathbf{Z})$
3. **Pooling:** $\mathbf{P} = \text{MaxPool}(\mathbf{A})$

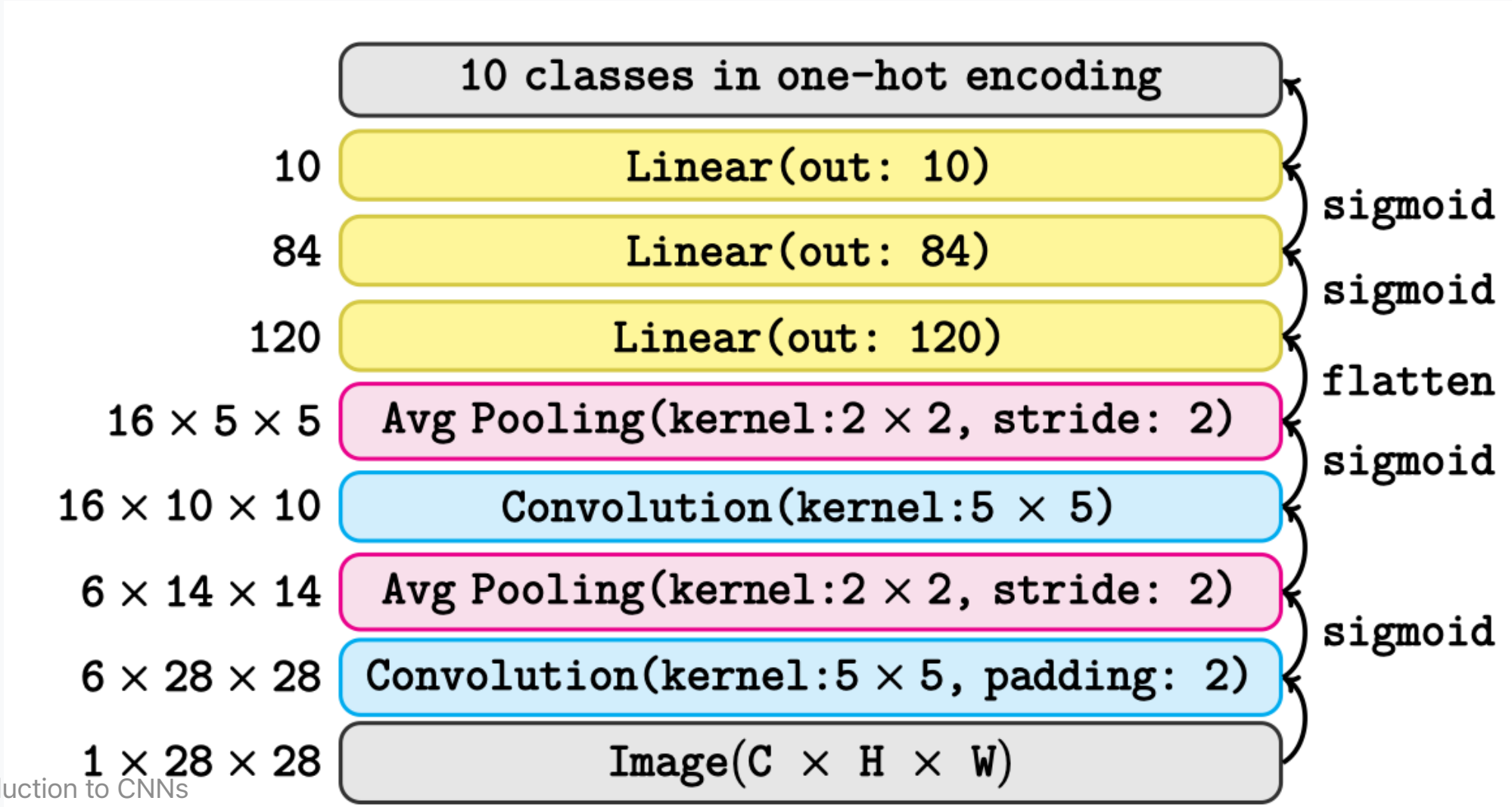
# CNN Building Block Assembly

## Typical Layer Combinations

### Modern Variations:

- **Batch Normalization:** Conv → BN → ReLU → Pool
- **Residual Connections:** Skip connections around blocks
- **Depthwise Separable:** Separate spatial and channel-wise convolutions

# CNN Building Block Assembly



# Receptive Field Calculation

## How Much Input Affects Output

**Definition:** Region of input that affects a particular output neuron

**Recursive Formula:**

$$RF_{l+1} = RF_l + (K_l - 1) \times \prod_{i=1}^{l} S_i$$

Where:

- $RF_l$: Receptive field at layer $l$
- $K_l$: Kernel size at layer $l$
- $S_i$: Stride at layer $i$

# Receptive Field Calculation

## How Much Input Affects Output

**Example Calculation:**

- Layer 0: $RF_0 = 1$ (input pixel)
- Layer 1: Conv 3×3, stride 1 → $RF_1 = 1 + (3 - 1) \times 1 = 3$
- Layer 2: Pool 2×2, stride 2 → $RF_2 = 3 + (2 - 1) \times 1 = 4$
- Layer 3: Conv 3×3, stride 1 → $RF_3 = 4 + (3 - 1) \times 2 = 8$

# Parameter Count Examples

## Practical CNN Sizing

**Example Network:**

- **Layer 1:** 32×32×3 input, 32 filters 5×5, stride 1, pad 2
- **Layer 2:** Max pool 2×2, stride 2
- **Layer 3:** 64 filters 5×5, stride 1, pad 2
- **Layer 4:** Max pool 2×2, stride 2
- **Layer 5:** 128 filters 3×3, stride 1, pad 1

# Parameter Count Examples

## Practical CNN Sizing

### Parameter Calculations:

- **Conv1:** $5 \times 5 \times 3 \times 32 + 32 = 2,432$
- **Conv3:** $5 \times 5 \times 32 \times 64 + 64 = 51,264$
- **Conv5:** $3 \times 3 \times 64 \times 128 + 128 = 73,856$
- **Total:** 127,552 parameters

**Memory Calculations:** Include activation maps, gradients, etc.

# Translation Equivariance

## Mathematical Property

**Definition:**

A function $f$ is translation equivariant if:

$$f(T_\delta(\mathbf{x})) = T_\delta(f(\mathbf{x}))$$

Where $T_\delta$ is translation by $\delta$.

**CNN Property:**

Convolutional layers are translation equivariant:

- **Shift input → Correspondingly shifted output**
- **Not translation invariant** (that requires pooling)

# Translation Equivariance

## Mathematical Property

### Mathematical Proof Sketch:

$$(I * K)(i+\delta, j) = \sum_{u,v} I(i+\delta+u, j+v)K(u,v)$$

$$= \sum_{u,v} (T_\delta I)(i+u, j+v)K(u,v) = (T_\delta I * K)(i,j)$$

# CNN vs Traditional Computer Vision

## Learning vs Hand-crafted Features

### Traditional Computer Vision Pipeline:

1. **Hand-crafted features:** SIFT, HOG, LBP
2. **Feature extraction:** Manual engineering
3. **Classifier:** SVM, Random Forest

# CNN vs Traditional Computer Vision

## Learning vs Hand-crafted Features

**CNN Pipeline:**

1. **Raw pixels → Learned features**
2. **End-to-end learning:** Features + Classifier
3. **Hierarchical representation:** Low → High level

**Mathematical Advantage:**

$$\text{Traditional: } \mathbf{y} = f_{\text{classifier}}(\phi(\mathbf{x}))$$

$$\text{CNN: } \mathbf{y} = f_{\text{CNN}}(\mathbf{x}; \boldsymbol{\theta})$$

Where $\phi(\cdot)$ is fixed feature extraction, but CNN learns optimal $\boldsymbol{\theta}$

# Summary: Class 5 Key Concepts

## CNN Motivation:

- **Parameter explosion** in MLPs for images
- **Need for translation invariance** and local feature detection
- **Biological inspiration** from visual cortex

## Convolution Mathematics:

- **2D discrete convolution:** $(I * K)(i, j) = \sum_{u,v} I(i + u, j + v)K(u, v)$
- **Output size formula:** $\frac{W-F+2P}{S} + 1$
- **Multi-channel convolution** and parameter sharing

## CNN Building Blocks:

- **Convolutional layers:** Feature detection with learned filters