

# Convolutional Neural Networks

**Tianxiang (Adam) Gao**

October 10, 2024

# Outline

1 Computer Vision Problems

2 Convolutional Neural Networks (CNNs)

3 Stabilize CNNs Training

4 Classic CNNs

## Recap: Neural Networks and Training

- MLPs are **parameterized** function  $f_{\theta}$ , where  $\theta = \{W^{\ell}, b^{\ell}\}$ :

- Forward Propagation (biases omitted): Start with  $x^0 = x$

$$\begin{aligned} z^{\ell} &= W^{\ell}x^{\ell-1}, \quad \forall \ell \in \{0, 1, 2, \dots, L\} \\ x^{\ell} &= \phi(z^{\ell}), \end{aligned}$$

- Backward Propagation (biases omitted): Start with  $dz^L = (x^L - y) \odot \phi'(z^L)$

$$\begin{aligned} dz^{\ell} &= [(W^{\ell+1})^T dz^{\ell+1}] \odot \phi'(z^{\ell}), \quad \forall \ell \in \{1, 2, \dots, L-1\} \\ dW^{\ell} &= dz^{\ell} x^{(\ell-1)\top} \end{aligned}$$

- The training involves solving an **optimization** problem to iteratively update the  $\theta$

$$\min_{\theta} \quad \mathcal{L}(\theta) = \frac{1}{n} \sum_{i=1}^n \ell(f_{\theta}(x_i), y_i) := R_S(f_{\theta}),$$

where  $\ell$  is a **loss** function and  $\mathcal{S} := \{x_i, y_i\}_{i=1}^{\ell}$  is a **training set**.

- This optimization problem can be solved using **gradient**-based methods such as (*stochastic*) *gradient descent (SGD)*, *gradient descent with momentum*, *RMSProp*, *Adam*, etc:

$$\theta^+ = \theta - \eta \cdot v^+,$$

where  $\eta > 0$  is a **learning rate** and  $v$  is a **search direction**.

## Recap: Generalization and Regularization

- **Model complexity trade-off:** The expected risk  $R(f_S) = \mathbb{E}_{(\mathbf{x}, \mathbf{y}) \sim \mathcal{D}} \ell(f_S(\mathbf{x}), \mathbf{y})$  is upper bounded:

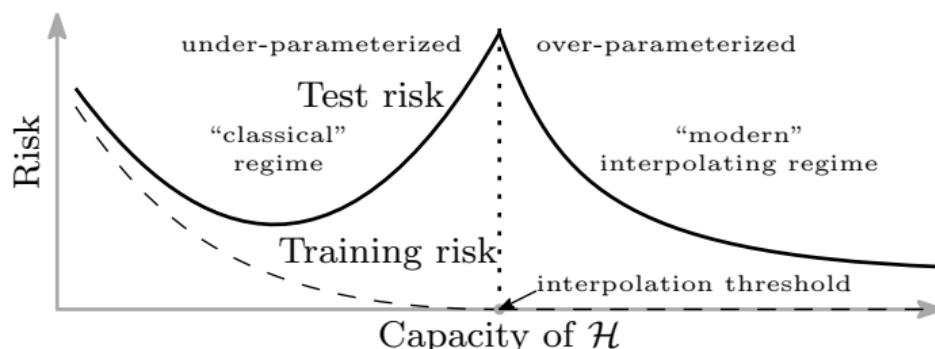
$$R(f_S) \leq R_S(f_S) + \mathfrak{R}_S(\mathcal{H}) + \tilde{\mathcal{O}}(n^{-1}).$$

- **Bias-Variance trade-off:** The expectation of  $R(f_S)$  over random sample  $S$  is decomposed as:

$$\mathbb{E}_S[R(f_S)] = \underbrace{\mathbb{E}_S[(f_S - \bar{f})^2]}_{\text{Variance term}} + \underbrace{\mathbb{E}_{\mathcal{D}}(\bar{f} - f^*)^2}_{\text{Bias term}} + \underbrace{R(f^*)}_{\text{irreducible}}$$

where  $\bar{f} := \mathbb{E}_S[f_S]$  and  $f^*$  is the optimal hypothesis.

- **Regularization:** Weight decay and dropout regularization
- **Hyperparameter tune:** Validation set, random search, log scale
- **Overparameterization:** Double descent, flat minimum, implicit regularization



# Outline

1 Computer Vision Problems

2 Convolutional Neural Networks (CNNs)

3 Stabilize CNNs Training

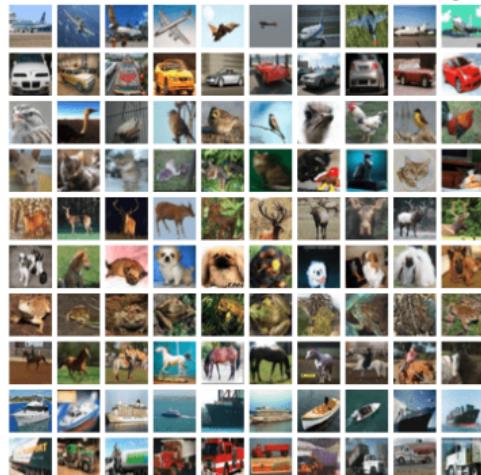
4 Classic CNNs

# Image Classification



- **Input:** An image
- **Output:** Cat? Binary classification (0 or 1).

## Multiple Classification: Softmax

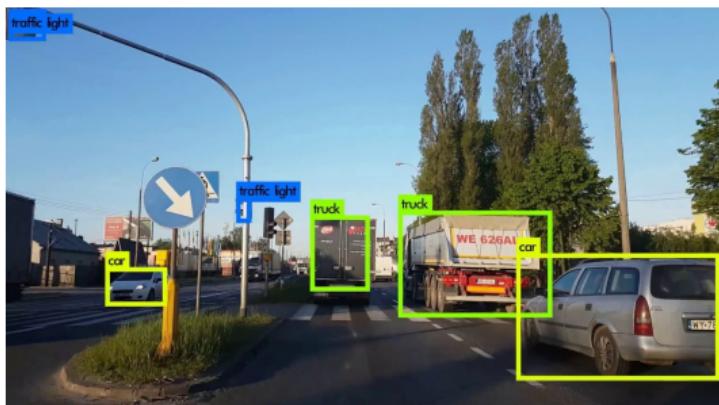


- **Input:** An image
  - **Output:** Class label  $\{0, 1, 2, \dots, 9\}$ .
  - **Softmax:** Converts a vector  $z$  of **logits** into probability distribution across classes

$$\text{Softmax}(\mathbf{z}_i) = \frac{e^{\mathbf{z}_i}}{\sum_{j=1}^C e^{\mathbf{z}_j}},$$

where  $C$  is the number of classes.

# Object Detection



- **Input:** An image

- **Outputs:**

- Class label
- Bounding box:  $[x_{\min}, y_{\min}, x_{\max}, y_{\max}]$
- Confidence scores: A probability or confidence score between 0 and 1.

# Pose Estimation



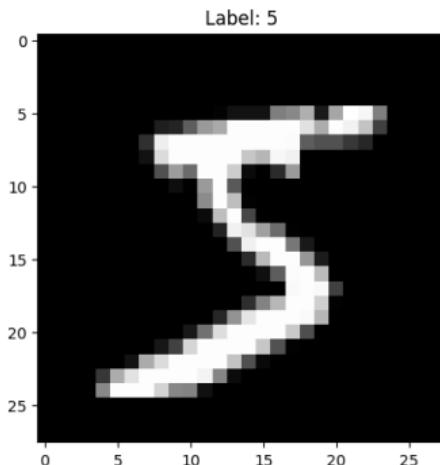
- **Input:** An image
- **Outputs:**
  - Keypoint coordinates:  $(x, y)$
  - Confidence scores: A probability or confidence score between 0 and 1.
- **Key points:** head, neck, shoulders, elbows, wrists, hips, knees, ankles

# Neural Style Transfer



- **Input:** Content image, style image
- **Output:** Generated image

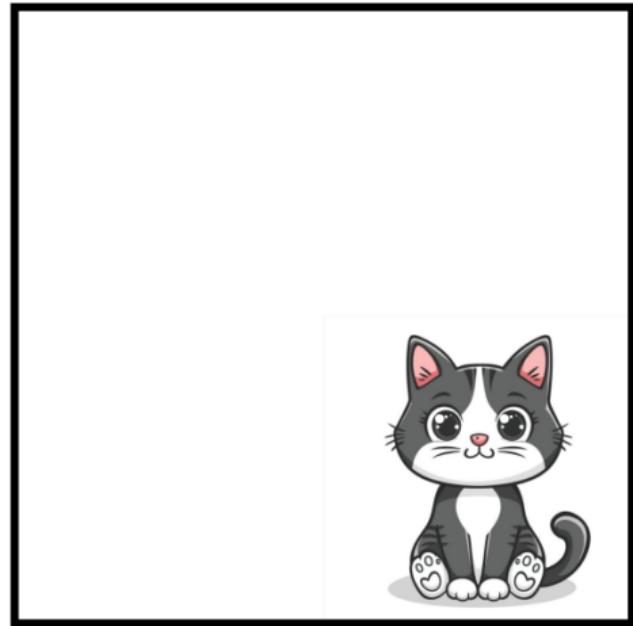
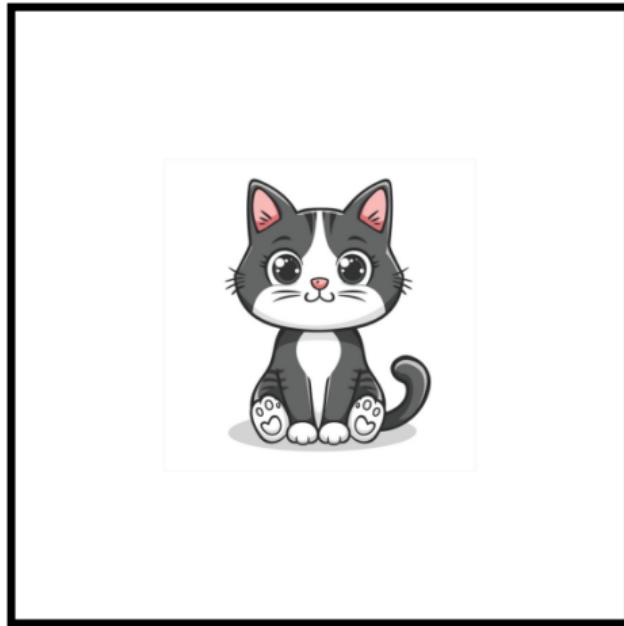
# Challenges in Image Data: High Dimensionality


$$28 \times 28 \times 1 = 784$$

$$1099 \times 733 \times 3 \approx 2.5 \text{ million pixels}$$

- A **two-layer** neural network with width 1000 leads to **3 billion parameters** to train.
- Despite having large datasets, the limited *computational cost* makes training challenging.

## Translation Invariance in Images



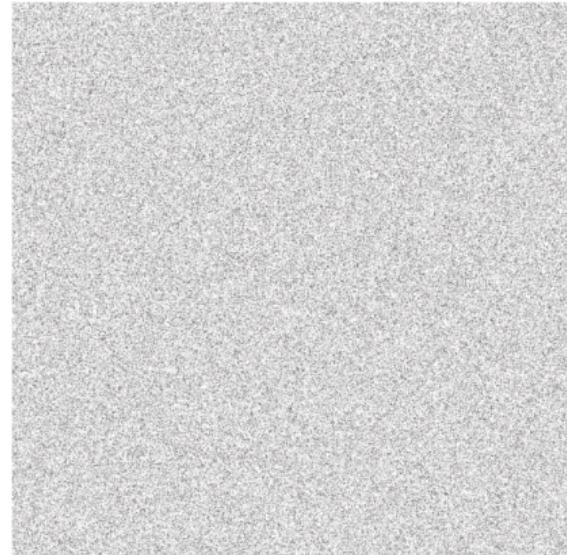
### Key Insight

Image features (edges, textures, or objects) can appear **anywhere** in the image, but they retain the same meaning regardless of their position. This is known as **translation invariance**.

# Importance of Spatial Structure in Images



Original Image



Randomly Shuffled Pixels

## Key Insight

The **spatial structure** and local connectivity of pixels define an image's recognizable features. When the spatial arrangement is disrupted, the image loses its recognizable form.

# Outline

1 Computer Vision Problems

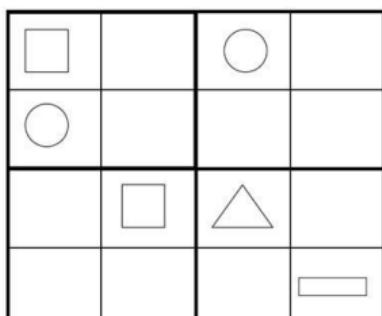
2 Convolutional Neural Networks (CNNs)

3 Stabilize CNNs Training

4 Classic CNNs

# Filters and Edge Detection in Image Processing

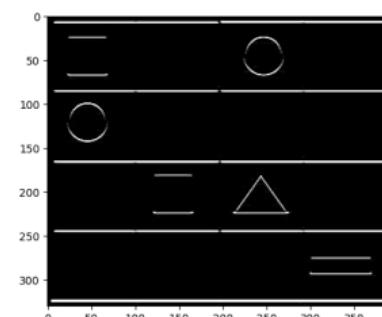
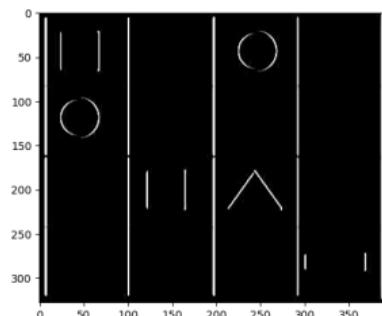
**Filter:** Filters are small matrices that are used to **detect** certain patterns, such as edges, textures, or other important features from the input data.



Original Image

$$\underbrace{\begin{bmatrix} 1 & 0 & -1 \\ 1 & 0 & -1 \\ 1 & 0 & -1 \end{bmatrix}}_{\text{Vertical Filter}}$$

$$\underbrace{\begin{bmatrix} -1 & -1 & -1 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \end{bmatrix}}_{\text{Horizontal Filter}}$$



# Convolution Operation

**Define:** In image processing, the convolution operation slides a small **filter** over the input image, performing a **locally linear transformation** (*i.e.*, element-wise multiplication and summing the results) to produce a feature map that detects patterns.

$$\underbrace{\begin{bmatrix} 3 & 0 & 1 & 2 & 7 & 4 \\ 1 & 5 & 8 & 9 & 3 & 1 \\ 2 & 7 & 2 & 5 & 1 & 3 \\ 0 & 1 & 3 & 1 & 7 & 8 \\ 4 & 2 & 1 & 6 & 2 & 8 \\ 2 & 4 & 5 & 2 & 3 & 9 \end{bmatrix}}_{\text{input image } 6 \times 6} * \underbrace{\begin{bmatrix} 1 & 0 & -1 \\ 1 & 0 & -1 \\ 1 & 0 & -1 \end{bmatrix}}_{\text{filter } 3 \times 3} = \underbrace{\quad\quad\quad}_{\text{feature map } 4 \times 4}$$

## Convolution Operation

**Define:** In image processing, the convolution operation slides a small **filter** over the input image, performing a **locally linear transformation** (i.e., element-wise multiplication and summing the results) to produce a feature map that detects patterns.

$$\underbrace{\begin{bmatrix} 3 & 0 & 1 & 2 & 7 & 4 \\ 1 & 5 & 8 & 9 & 3 & 1 \\ 2 & 7 & 2 & 5 & 1 & 3 \\ 0 & 1 & 3 & 1 & 7 & 8 \\ 4 & 2 & 1 & 6 & 2 & 8 \\ 2 & 4 & 5 & 2 & 3 & 9 \end{bmatrix}}_{\text{input image } 6 \times 6} * \underbrace{\begin{bmatrix} 1 & 0 & -1 \\ 1 & 0 & -1 \\ 1 & 0 & -1 \end{bmatrix}}_{\text{filter } 3 \times 3} = \underbrace{\begin{bmatrix} -5 & -4 & 0 & 8 \\ 10 & -2 & 2 & 3 \\ 0 & -2 & -4 & -7 \\ -3 & -2 & -3 & -16 \end{bmatrix}}_{\text{feature map } 4 \times 4}$$

- If the input image is  $n \times n$  and the filter size is  $f \times f$ , then the output feature map has size  $(n - f + 1) \times (n - f + 1)$ .

# Padding

**Define:** Padding refers to adding extra pixels (usually zeros) around the input data to control the size of the output feature map.

0	0	0	0	0	0	0	0
0	3	0	1	2	7	4	0
0	1	5	8	9	3	1	0
0	2	7	2	5	1	3	0
0	0	1	3	1	7	8	0
0	4	2	1	6	2	8	0
0	2	4	5	2	3	9	0
0	0	0	0	0	0	0	0

- **Preserving Spatial Dimensions:** Padding maintains the spatial dimensions in deeper neural networks.
- **Capture Edge information:** Padding prevents the loss of boundary information during convolution.
- **Controlling Output Size:** Padding helps ensure feature maps retain the required size for subsequent layers.
- The shape of feature map:  $(n + 2p - f + 1) \times (n + 2p - f + 1)$ .

## “Valid” and “Same” Convolution

**Define:** Padding refers to adding extra pixels (usually zeros) around the input data to control the size of the output feature map.

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

- **Valid:** No padding, output size is  $(n - f + 1) \times (n - f + 1)$ .
- **Same:** Padding ensures the output has the same shape as the input, with output size  $(n + 2p - f + 1) \times (n + 2p - f + 1)$ .

$$n + 2p - f + 1 = n \implies p = \frac{f - 1}{2}$$

Hence, generally, filters have **odd**, e.g.,  $3 \times 3$  or  $5 \times 5$ .

## Stride

**Define:** Stride in CNNs refers to the number of pixels by which the filter moves across the input during convolution, affecting the output size by skipping certain positions.

$$\underbrace{\begin{bmatrix} 3 & 0 & 1 & 2 & 7 & 4 \\ 1 & 5 & 8 & 9 & 3 & 1 \\ 2 & 7 & 2 & 5 & 1 & 3 \\ 0 & 1 & 3 & 1 & 7 & 8 \\ 4 & 2 & 1 & 6 & 2 & 8 \\ 2 & 4 & 5 & 2 & 3 & 9 \end{bmatrix}}_{\text{input image } 6 \times 6} * \underbrace{\begin{bmatrix} 1 & 0 & -1 \\ 1 & 0 & -1 \\ 1 & 0 & -1 \end{bmatrix}}_{\text{filter } 3 \times 3} = \underbrace{\begin{bmatrix} -5 & 0 \\ 0 & -4 \end{bmatrix}}_{\text{feature map } 2 \times 2 \text{ with stride 2}}$$

- **Control Output Size:** Larger stride results in a smaller feature map.
- **Computational Efficiency:** Larger strides require fewer convolution operations.
- **Output Feature Map Shape:**

$$\left\lfloor \frac{n + 2p - f}{s} + 1 \right\rfloor \times \left\lfloor \frac{n + 2p - f}{s} + 1 \right\rfloor,$$

where  $\lfloor x \rfloor$  is the floor function, returning the largest integer less than or equal to  $x$ .

## Simplified Convolutional Layer

- Let  $\mathbf{X} \in \mathbb{R}^{n \times n}$  be the input image, and  $\mathbf{F} \in \mathbb{R}^{f \times f}$  be the **trainable** filter.
- The convolutional layer is defined as:

$$\mathbf{Z} = \mathbf{X} * \mathbf{F} + b, \quad \mathbf{A} = \text{ReLU}(\mathbf{Z})$$

where:

- $b \in \mathbb{R}$  is the **bias** term added to each element in  $\mathbf{Z}$ .
- $\mathbf{Z} \in \mathbb{R}^{(n-f+1) \times (n-f+1)}$  represents the **pre-activation values**, assuming no padding and a stride of 1.

$$\underbrace{\begin{bmatrix} 3 & 0 & 1 & 2 & 7 & 4 \\ 1 & 5 & 8 & 9 & 3 & 1 \\ 2 & 7 & 2 & 5 & 1 & 3 \\ 0 & 1 & 3 & 1 & 7 & 8 \\ 4 & 2 & 1 & 6 & 2 & 8 \\ 2 & 4 & 5 & 2 & 3 & 9 \end{bmatrix}}_{\mathbf{X}} * \underbrace{\begin{bmatrix} 1 & 0 & -1 \\ 1 & 0 & -1 \\ 1 & 0 & -1 \end{bmatrix}}_{\mathbf{F}} = \underbrace{\begin{bmatrix} -5 & -4 & 0 & 8 \\ 10 & -2 & 2 & 3 \\ 0 & -2 & -4 & -7 \\ -3 & -2 & -3 & -16 \end{bmatrix}}_{\mathbf{Z}}$$

### Key Observation

While MLPs use explicit weight matrices, CNNs use **filters** that serve the role of weight matrices, learning specific features directly from the data.

## Neurons in CNNs

- We can represent the input image  $\mathbf{X}$  and filter  $\mathbf{F}$  as vector forms,  $\mathbf{x} \in \mathbb{R}^{n^2 \times 1}$  and  $\mathbf{w} \in \mathbb{R}^{f^2 \times 1}$ , by stacking their entries:

$$\mathbf{X} = [\mathbf{x}_1 \quad \cdots \quad \mathbf{x}_n] \implies \mathbf{x} = \begin{bmatrix} \mathbf{x}_1 \\ \vdots \\ \mathbf{x}_n \end{bmatrix},$$

$$\mathbf{F} = [\mathbf{f}_1 \quad \cdots \quad \mathbf{f}_f] \implies \mathbf{w} = \begin{bmatrix} \mathbf{f}_1 \\ \vdots \\ \mathbf{f}_f \end{bmatrix}.$$

- Thus, each convolution can be viewed as applying a permutation matrix  $\Pi_i \in \mathbb{R}^{f^2 \times n^2}$  to  $\mathbf{x}$  to obtain  $\hat{\mathbf{x}}_i$ , followed by an inner product with  $\mathbf{w}$ :

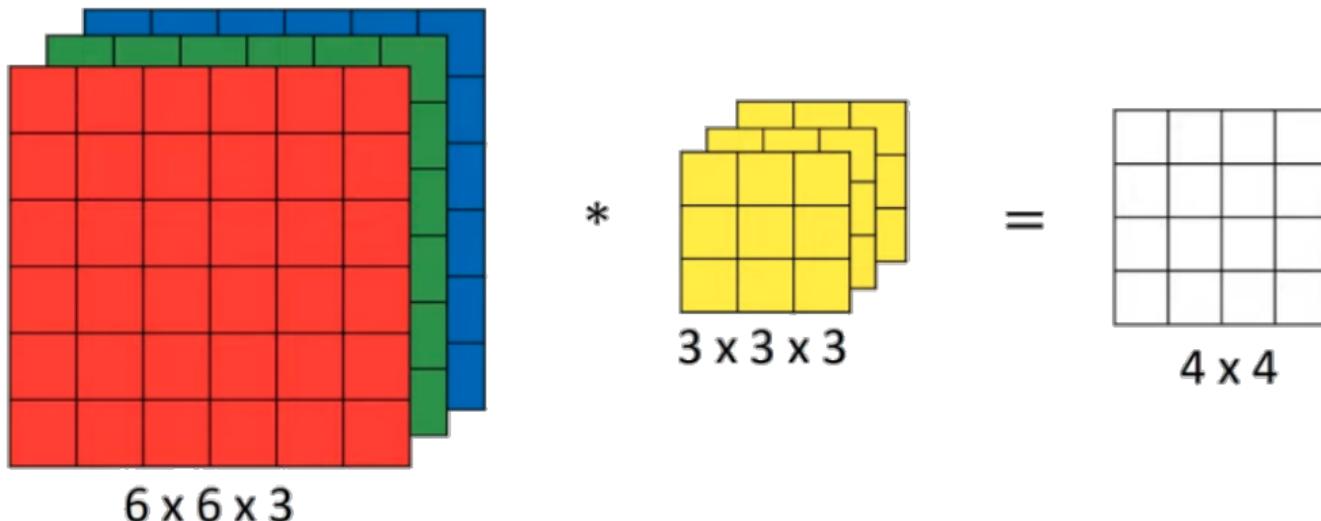
$$\hat{\mathbf{x}}_i = \Pi_i \mathbf{x}, \quad \mathbf{z}_i = \mathbf{w}^\top \hat{\mathbf{x}}_i + b, \quad \mathbf{a}_i = \text{ReLU}(\mathbf{z}_i), \quad \forall i \in \{1, 2, \dots, (\mathbf{n} - f + 1)^2\}$$

### Key Insights

- Sharing:** Each neuron in convolutional layer **shares** the same weights and bias across spatial locations, reducing the impact of high dimensionality.
- Sparsity:** Each output depends only on a **locally** small portion of the input.

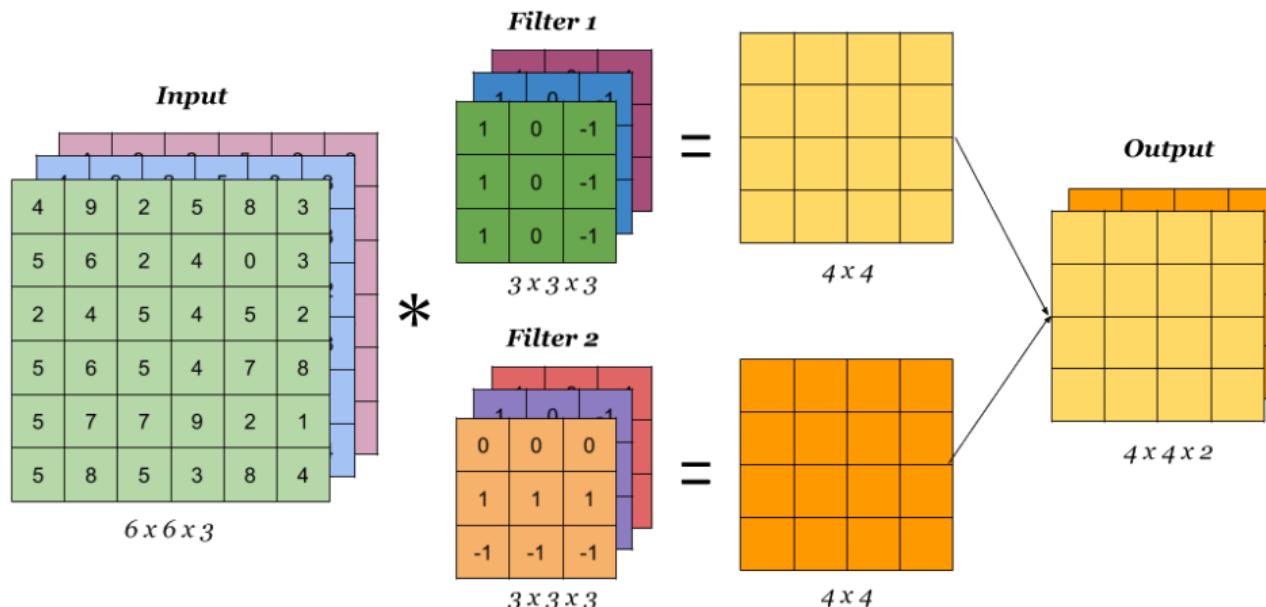
# Convolution Over Volumes

- The input can have **multiple** channels (e.g., an RGB image), and the filter should have a **matching** number of channels.
- The filter has size  $n_H \times n_W \times n_C$

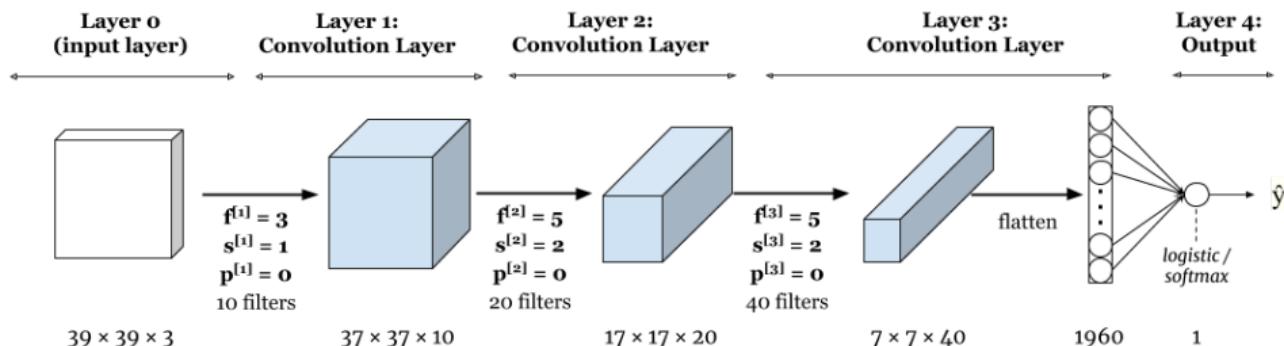


# Convolution with Multiple Filters

- Multiple filters can be used in a convolution layer to detect multiple features.



# Simple CNN Example

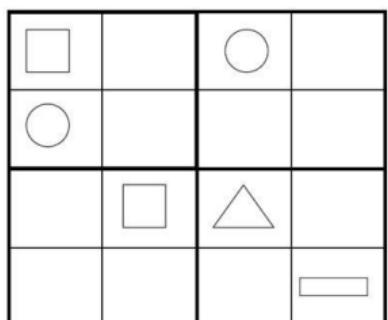


<https://indoml.com>

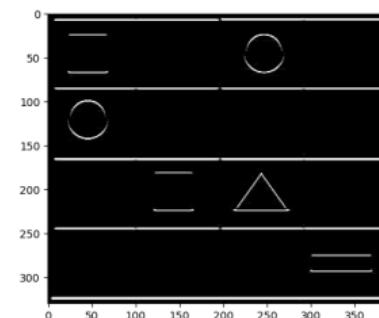
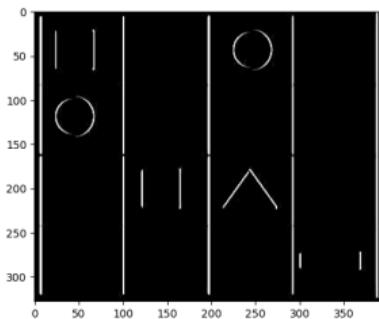
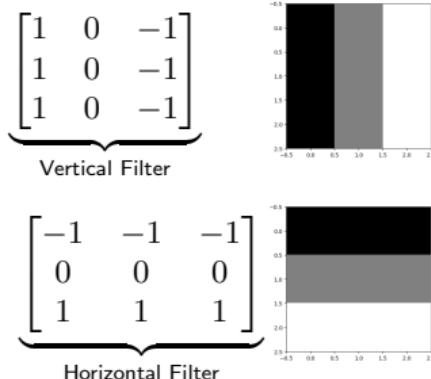
- The output of CNN is **flattened** into a vector
- The flattened vector serves as the input to a **fully connected** layer
- In CNN design, feature maps typically shrink in spatial size while channels increase as depth grows.

## Feature Map as an Indicator

**Feature Map as an Indicator:** The output feature map highlights detected patterns, with higher values indicating matched regions.

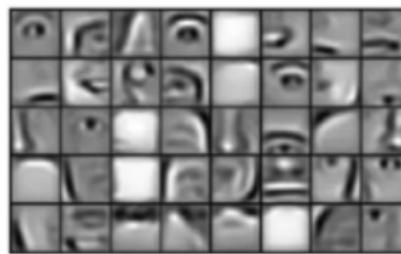
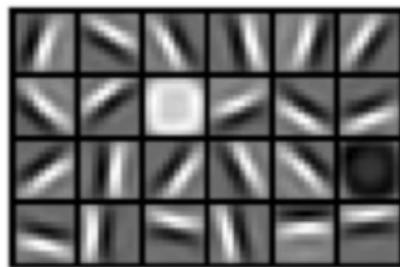


Original Image



## Hierarchical Feature Detection

**Feature Map as an Indicator:** The output feature map highlights detected patterns, with higher values indicating matched regions.



- **Early Layers:** Detect basic elements like edges and textures, forming the foundation for more complex patterns.
- **Middle Layers:** Combine edges into shapes (e.g., circles, squares) by recognizing the arrangement of basic features.
- **Deeper Layers:** Recognize object parts by detecting combinations of shapes and features.
- **Final Layers:** Detect entire objects by assembling recognized parts, outputting a classification or region of interest.

# Summary of Convolutional Neural Networks

- **Filters:** Small, **trainable** matrices that detect features in the input data.
- **Convolution Operation:** A **locally linear transformation** that creates a feature map, emphasizing regions where the filter matches the pattern.
- **Padding and Stride:** Methods for controlling feature map size, preserving spatial dimensions, and improving *computational efficiency*.
- **Convolution Over Volumes:** Designed to process multi-channel inputs like RGB images with filters that **match** each channel.
- **Multiple Filters:** A single convolutional layer can use **multiple** filters to detect various features simultaneously.
- **Weight Sharing and Sparsity:** Neurons in CNNs **share** weights across locations, with each output relying on a **small, localized** input region.
- **Hierarchical Feature Detection:** Early layers capture basic features (like edges), which later layers combine into higher-level features.

# Outline

1 Computer Vision Problems

2 Convolutional Neural Networks (CNNs)

3 Stabilize CNNs Training

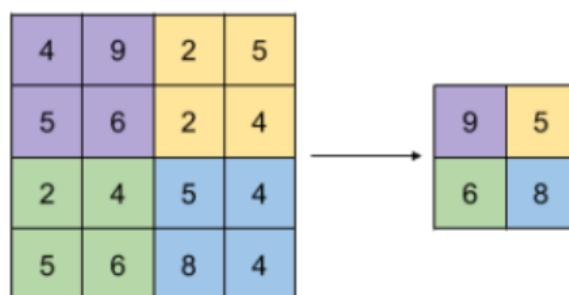
4 Classic CNNs

# Pooling

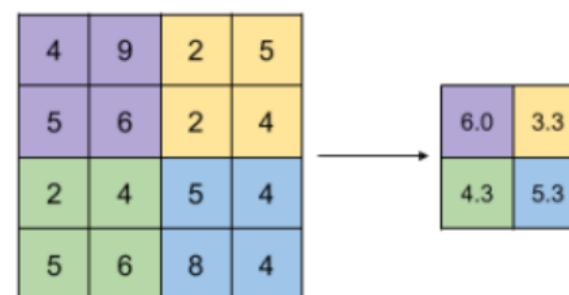
**Define:** The pooling layer in CNNs reduces spatial dimensions of feature maps through downsampling, commonly using max or average pooling operations.

- Pooling helps reduce the computational load
- It also enhances **robustness** by making the network less sensitive to small spatial variations.

*Max Pooling*



*Avg Pooling*



- Common hyperparameters: pool size  $f$  and stride  $s$ , typically  $f = s = 2$ .

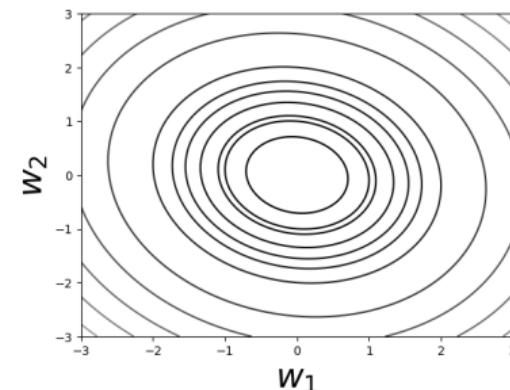
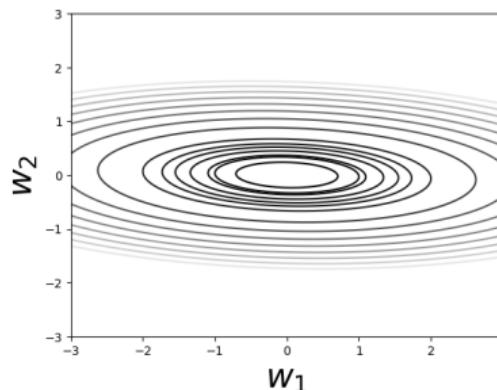
## Recap: Input Normalization

- Normalize the inputs using **training set**:

$$\boldsymbol{\mu} = \frac{1}{n} \sum_{i=1}^n \mathbf{x}_i, \quad \sigma^2 = \frac{1}{n} \sum_{i=1}^n (\mathbf{x}_i - \boldsymbol{\mu})^2, \quad \bar{\mathbf{x}}_i = (\mathbf{x}_i - \boldsymbol{\mu})/\sigma,$$

where all operations are taken element-wise.

- Consider a binary classification problem using linear model:  $f_{\theta}(x) = \mathbf{w}^\top \mathbf{x} = w_1 x_1 + w_2 x_2$ 
  - if  $x_1 = \mathcal{O}(100)$  and  $x_2 = \mathcal{O}(1)$ , to have output  $f_{\theta} = \mathcal{O}(1)$ , we must have  $w_1 = \mathcal{O}(\frac{1}{100})$  and  $w_2 = \mathcal{O}(1)$ .
  - After normalization,  $\bar{x}_1 = \mathcal{O}(1)$  and  $\bar{x}_2 = \mathcal{O}(1)$ , so we have  $w_1 = \mathcal{O}(1)$  and  $w_2 = \mathcal{O}(1)$ .



- At test time, apply  $\mu$  and  $\sigma$  from **training** to test set.

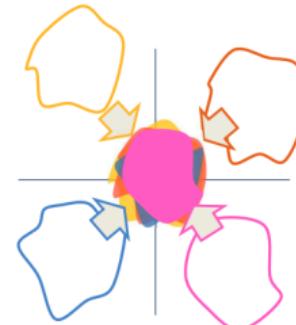
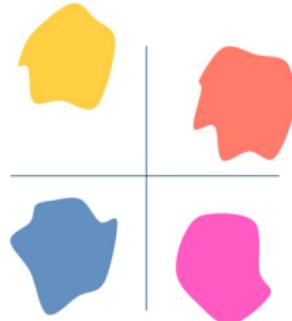
## Batch Normalization

- The forward propagation in DNNs:

$$\mathbf{z}_i^\ell = \mathbf{w}_i^\top \mathbf{x}^{\ell-1}, \quad \mathbf{a}_i^\ell = \phi(\mathbf{z}_i^\ell), \quad \forall i \in [n],$$

where  $n$  is the width of hidden layers.

- We can apply the same normalization from the input layer to each hidden layer to speed up the training for  $\mathbf{w}_i^\ell$  for all  $i \in [n]$ .
- Additionally, as we train DNNs using mini-batch rather than full batch, **internal covariance shift** in mini-batches. Hence, the normalization is applied on the mini-batch rather the full batches.



## Batch Normalization During Training

Given a mini-batch  $\{z^1, \dots, z^b\}$  for a hidden layer:

- Normalize the **pre-activation** to mean zero and variance one:

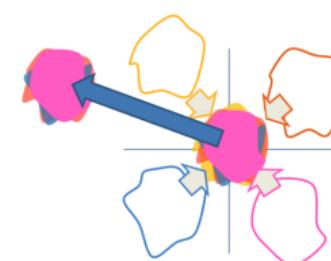
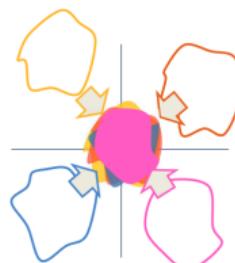
$$\mu = \frac{1}{b} \sum_{i=1}^b z^i, \quad \sigma^2 = \frac{1}{b} \sum_{i=1}^b (z^i - \mu)^2, \quad z_{\text{norm}}^i = \frac{z^i - \mu}{\sqrt{\sigma^2 + \varepsilon}}$$

where  $\varepsilon > 0$  ensures numerical stability.

- Re-scale and shift using learnable parameters:

$$\hat{z}^i = \gamma z_{\text{norm}}^i + \beta$$

where  $\gamma$  and  $\beta$  are **learnable** parameters.



- $\gamma$  and  $\beta$  enable identity transformation, allowing flexibility:

$$\gamma = \sqrt{\sigma^2 + \varepsilon}, \quad \beta = \mu$$

## Batch Normalization at Test Time

- **Training Phase:** For each mini-batch, compute batch statistics and update the normalized output:

$$\boldsymbol{\mu}_{\text{batch}} = \frac{1}{b} \sum_{i=1}^b \mathbf{z}^i, \quad \sigma_{\text{batch}}^2 = \frac{1}{b} \sum_{i=1}^b (\mathbf{z}^i - \boldsymbol{\mu}_{\text{batch}})^2, \quad \mathbf{z}_{\text{norm}}^i = \frac{\mathbf{z}^i - \boldsymbol{\mu}_{\text{batch}}}{\sqrt{\sigma_{\text{batch}}^2 + \varepsilon}}, \quad \hat{\mathbf{z}}^i = \gamma \mathbf{z}_{\text{norm}}^i + \beta$$

- **Running Statistics:** After each mini-batch, update the running mean and variance:

$$\boldsymbol{\mu}_{\text{run}} = (1 - \alpha) \boldsymbol{\mu}_{\text{run}} + \alpha \boldsymbol{\mu}_{\text{batch}}$$

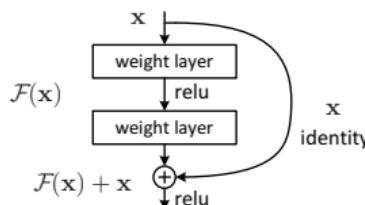
$$\sigma_{\text{run}}^2 = (1 - \alpha) \sigma_{\text{run}}^2 + \alpha \sigma_{\text{batch}}^2$$

- **Test Phase:** Normalize using running statistics and apply scale and shift:

$$\mathbf{z}_{\text{test}} \leftarrow \frac{\mathbf{z}_{\text{test}} - \boldsymbol{\mu}_{\text{run}}}{\sqrt{\sigma_{\text{run}}^2 + \varepsilon}}, \quad \hat{\mathbf{z}}_{\text{test}} \leftarrow \gamma \mathbf{z}_{\text{test}} + \beta$$

# Skip Connections

**Define:** A skip connection is a shortcut in a DNN that adds the input directly to the output.



- In MLPs, forward propagation without skip connections (omitting biases):

$$\mathbf{z}^\ell = \mathbf{W}^\ell \phi(\mathbf{z}^{\ell-1}) \approx \mathbf{W}^\ell \mathbf{z}^{\ell-1} \approx \mathbf{W}^\ell \cdots \mathbf{W}^1 \mathbf{z}^0 = \mathcal{O}(a^\ell)$$

This results in an **exponential growth or decay** of information.

- With skip connections, the propagation becomes:

$$\mathbf{z}^\ell = \mathbf{W}^\ell \phi(\mathbf{z}^{\ell-1}) + \mathbf{z}^{\ell-1} = \sum_{i=0}^{\ell} \mathbf{W}^i \phi(\mathbf{z}^i) = \mathcal{O}(\ell)$$

Here, **linear growth** of information is achieved, stabilizing the learning process.

# Outline

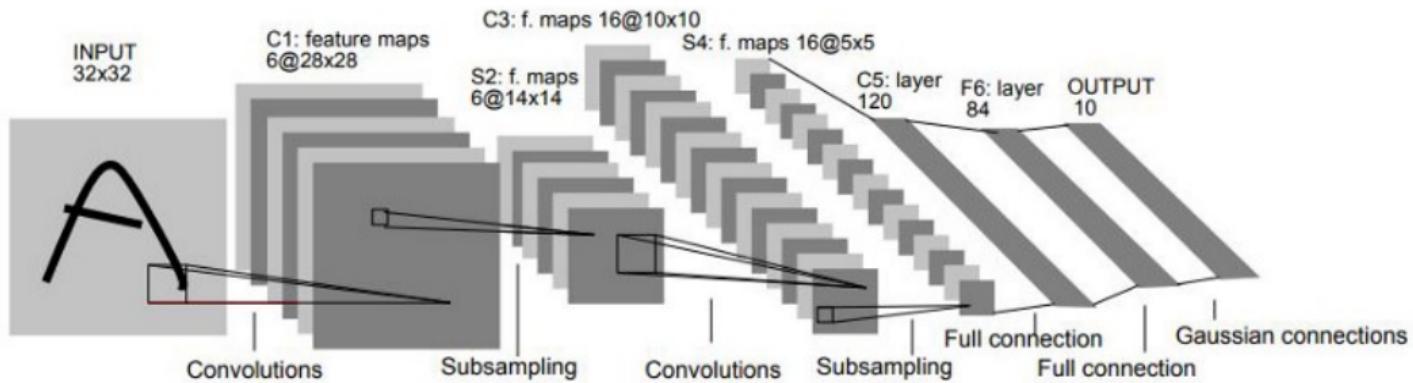
1 Computer Vision Problems

2 Convolutional Neural Networks (CNNs)

3 Stabilize CNNs Training

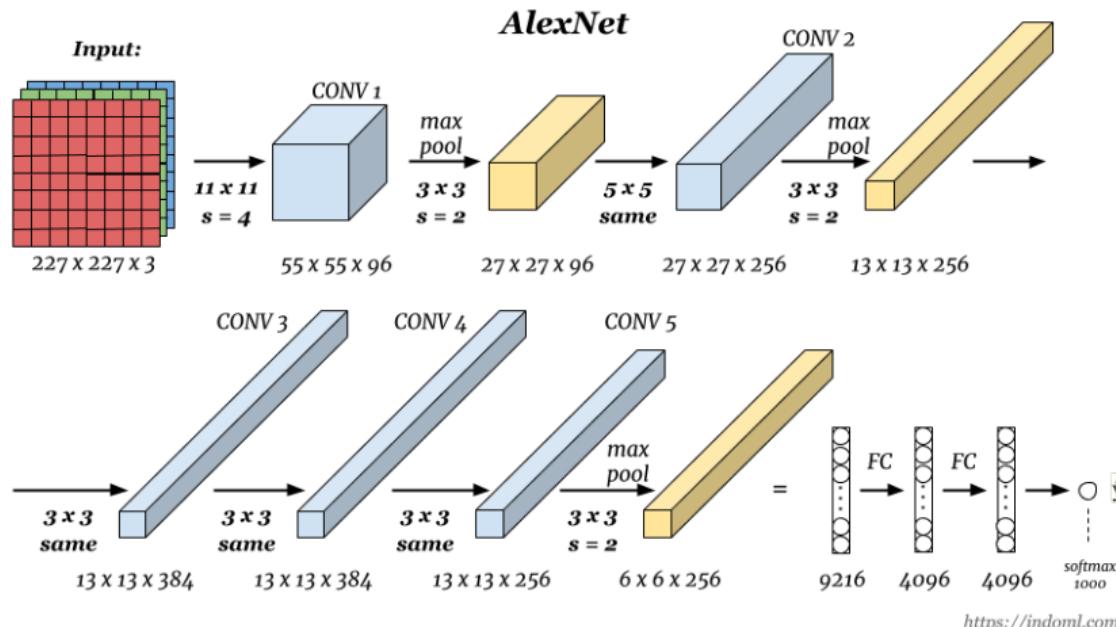
4 Classic CNNs

## LeNet-5



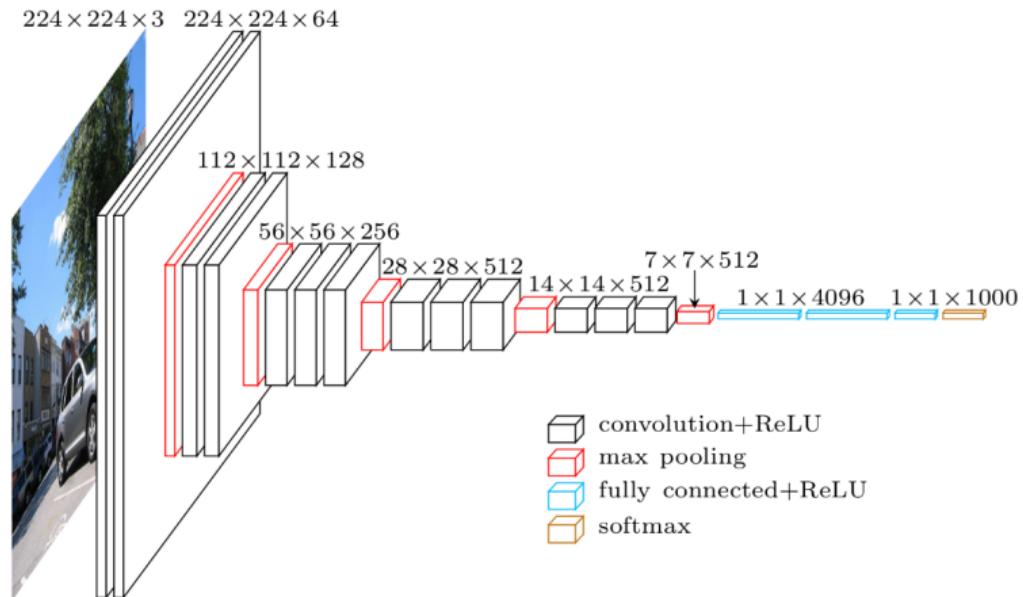
- It has totally 5 weight layers: 2 Convolutional layers and 3 fully connected (FC) layers
- Sigmoid and tanh activations
- Average pooling
- Number of parameters:  $\sim 60$  thousands.
- MNIST dataset:  $\sim 60$  thousands.

## AlexNet



- 5 convolutional and 3 FC.
- ReLU activation and max pooling layers.
- Dropout regularization in FC layers.
- Number of parameters:  $\sim 63$  million.
- ImageNet dataset:  $\sim 1.2$  million images.

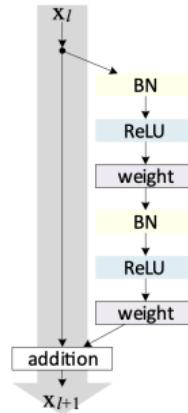
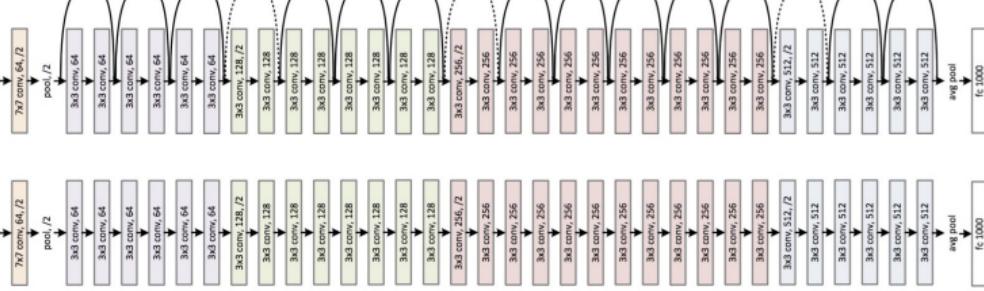
## VGG-16



- 13 convolutional and 3 FC.
- *Unified convolution and max-pooling setup:  $f = 3, s = 1$ , and “same”;  $f = 2$  and  $s = 2$*
- $\sim 138$  million parameters trained on ImageNet

## ResNet-34

34-layer residual image



- Batch normalization and skip connections applied to pre-activation.
- $\sim 11.7$  million parameters trained on ImageNet.