# Image Processing & Vision

## Lecture 12: Convolutional Neural Networks

**Hak Gu Kim**

hakgukim@cau.ac.kr

**Immersive Reality & Intelligent Systems Lab (IRIS LAB)**

**Graduate School of Advanced Imaging Science, Multimedia & Film (GSAIM)**

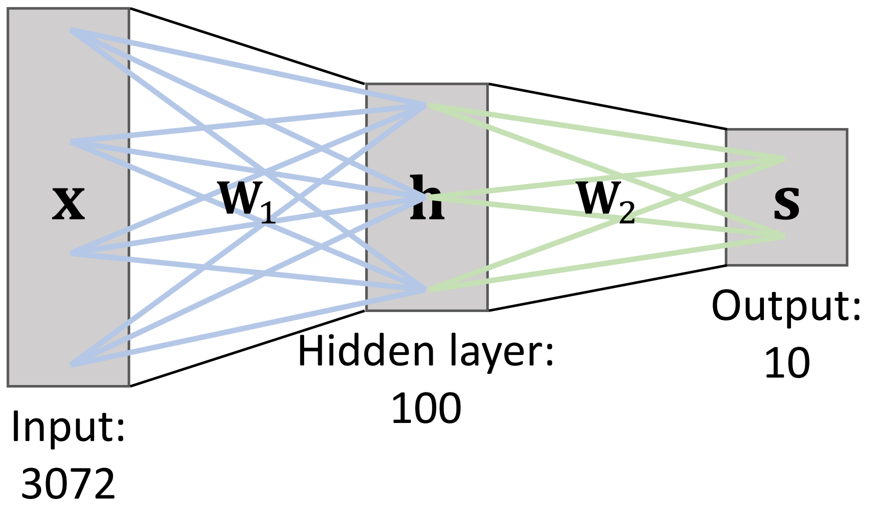
**Chung-Ang University (CAU)**

**5 June 2023**

# **Recap:** Neural Networks

- Input: $\mathbf{x} \in \mathbb{R}^{D\times 1}$ $where\ D = 3072$
- Output: $f(\mathbf{x}) \in \mathbb{R}^{C\times 1}$ $where\ C = 10$
- **Linear Classifier:** $f(\mathbf{x}) = \mathbf{W}\mathbf{x} + \mathbf{b}$
- **2-layer Neural Network:** $f(\mathbf{x}) = \mathbf{W}_2 \max(0, \mathbf{W}_1\mathbf{x} + \mathbf{b}_1) + \mathbf{b}_2$

All elements of $\mathbf{x}$ affect all elements of $\mathbf{h}$




All elements of $\mathbf{h}$ affect all elements of $\mathbf{s}$

**Fully-connected neural network (MLP)**

# Topics

- Convolutional Neural Networks (CNNs)
  - Convolution Layers
  - Pooling Layers
  - Normalization

- Evolution of CNNs

**Note:** Many of these slides in this course were adapted from Convolutional Neural Networks for Visual Recognition (Stanford Univ.) and Deep Learning for Computer Vision (Univ. of Michigan)

# Topics

- Convolutional Neural Networks (CNNs)
  — Convolution Layers
  — Pooling Layers
  — Normalization

- Evolution of CNNs

***Note:** Many of these slides in this course were adapted from Convolutional Neural Networks for Visual Recognition (Stanford Univ.) and Deep Learning for Computer Vision (Univ. of Michigan)

# **Limitations** of Linear Classifiers and MLP

- **Problem:**

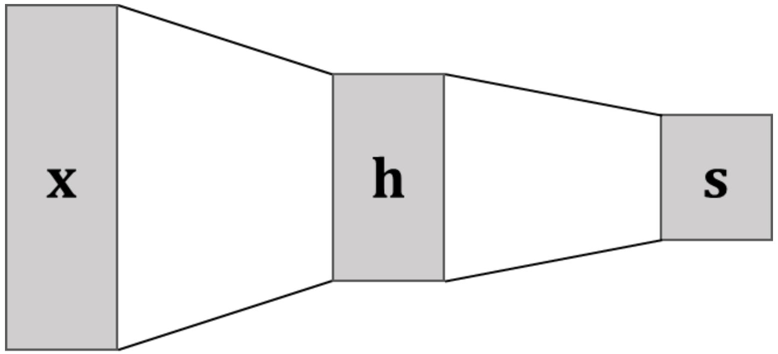
— The classifiers we learned such as linear classifiers and neural networks (i.e., multi-layer perceptron) do not respect the spatial structure of images

- **Solution:**

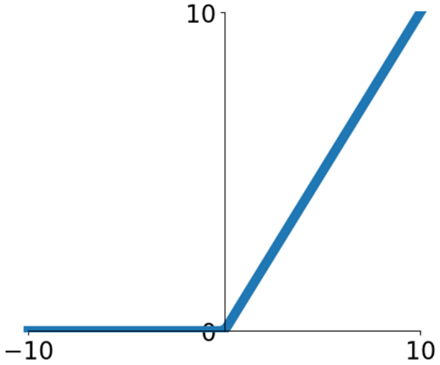
— Design new computational nodes on images for spatial operation

# Components of **Neural Networks (MLP)**

Fully-Connected Layers (FC layers)

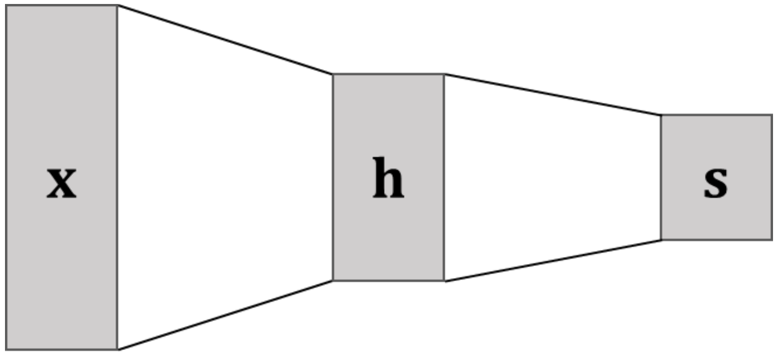



Activation Function

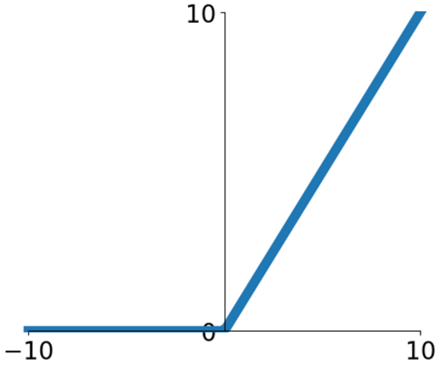


# Components of **Convolutional Neural Networks** (CNNs)

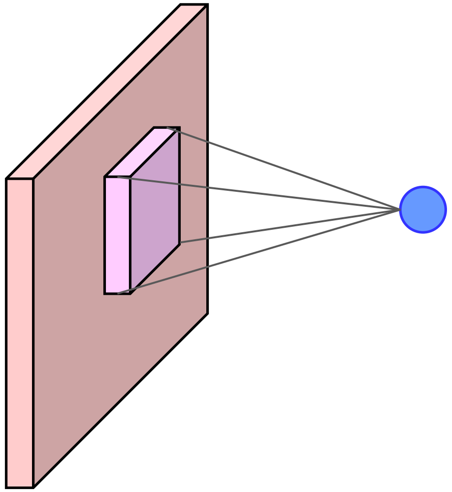
## Fully-Connected Layers (FC layers)



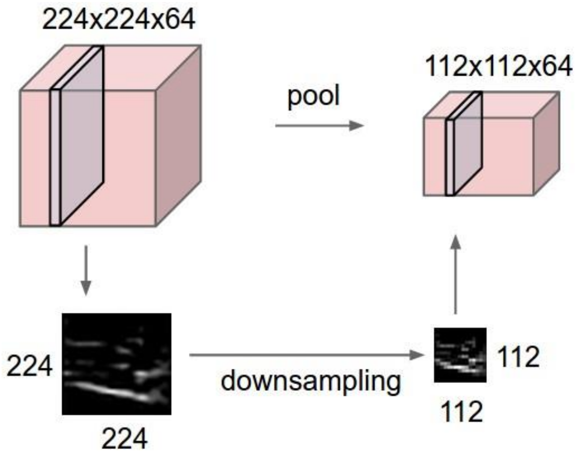

## Activation Function



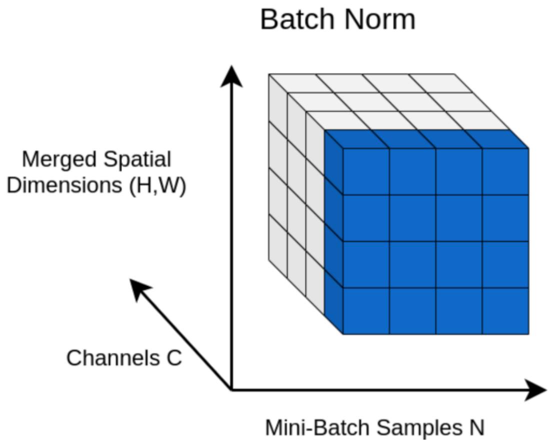

## Convolution Layers



## Pooling Layers

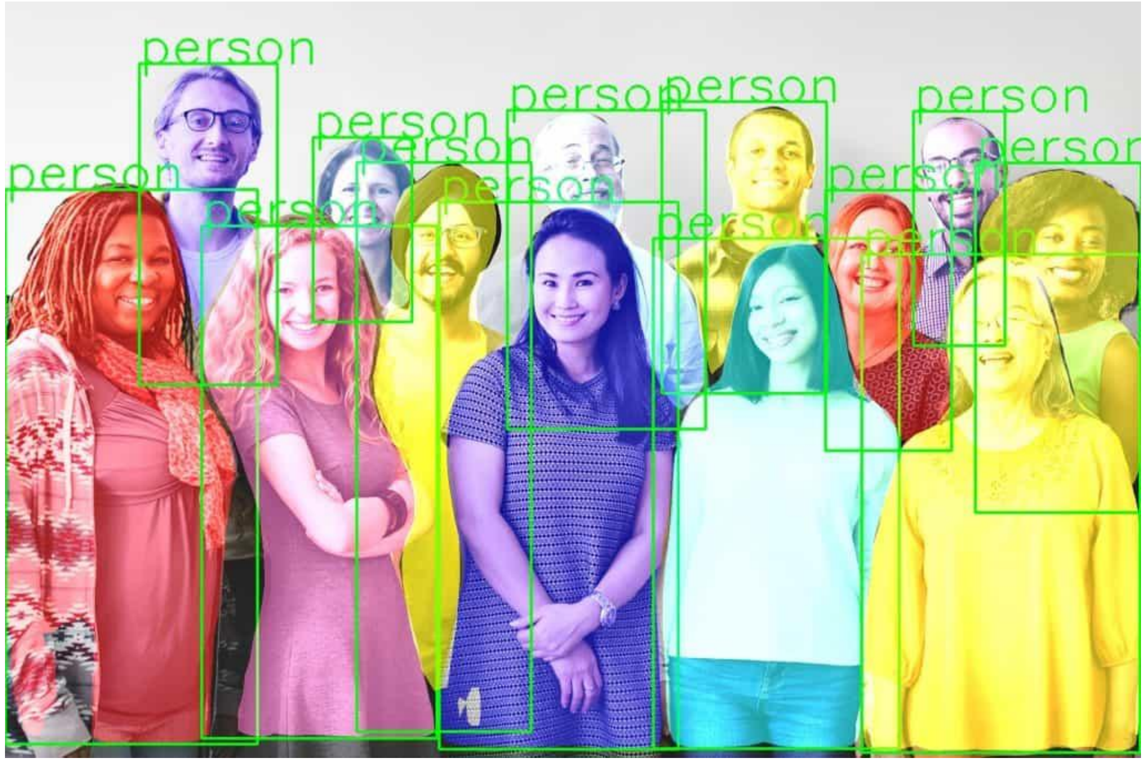


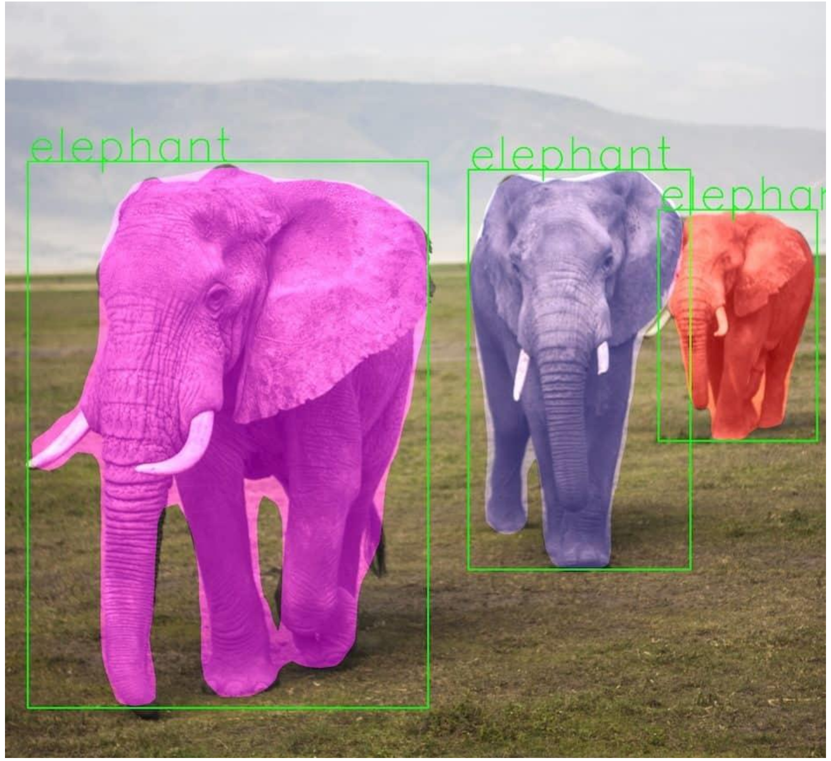

## Normalization



# CNNs

- **Convolutional neural networks (CNNs)** have been tremendously successful in practical computer vision applications such as object detection, classification, segmentation, *etc*.

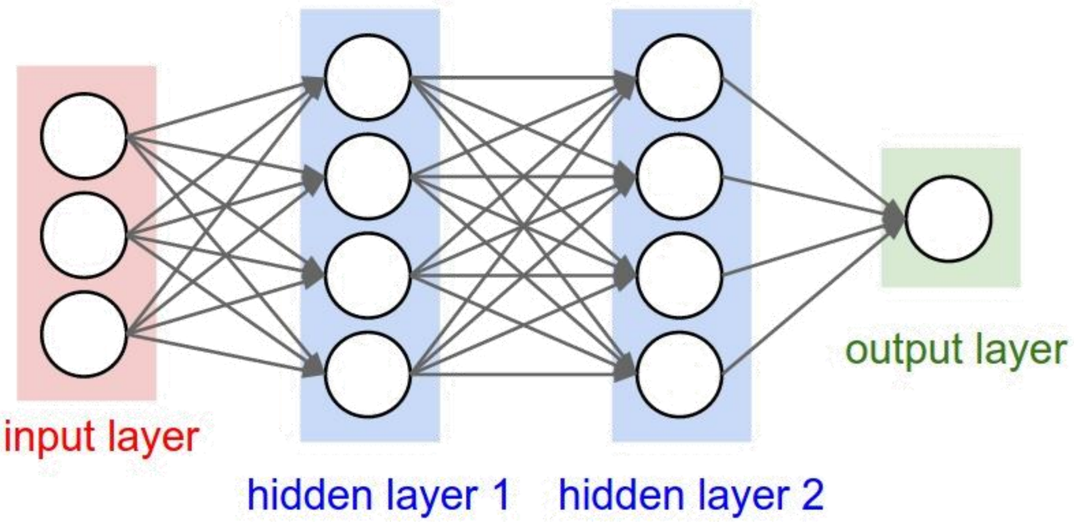



https://learnopencv.com/mask-r-cnn-instance-segmentation-with-pytorch/

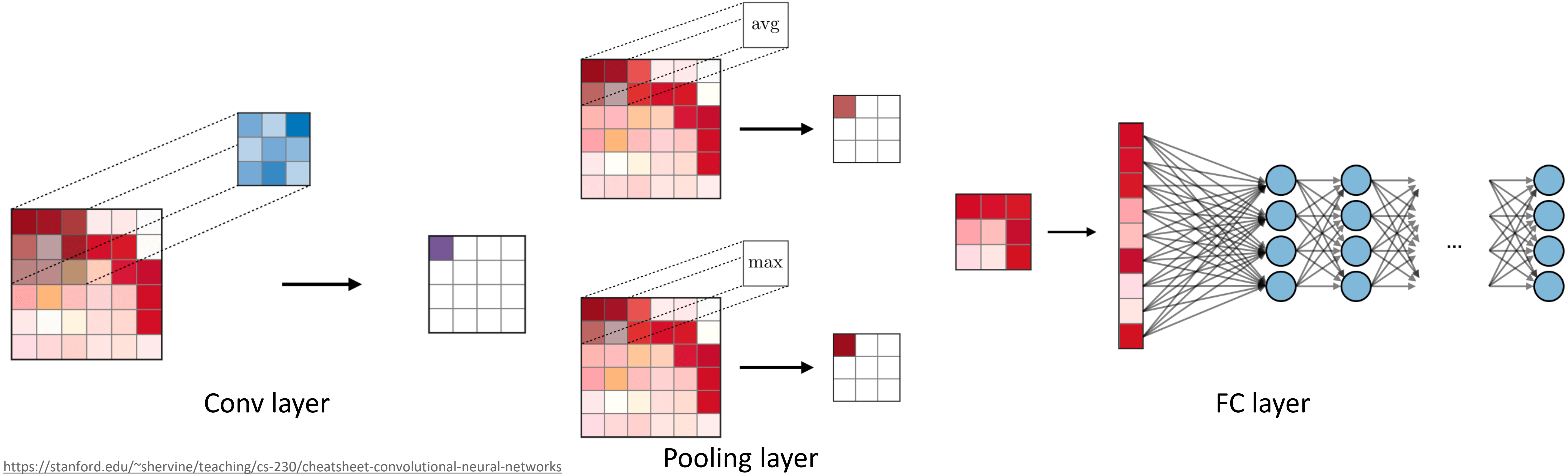
# Architecture of **Neural Networks (MLP)**

- **Neural Networks (MLP):**
  - An input (a single vector) is transformed it through a series of hidden layers
  - Each hidden layer is made up of a set of neurons, where each neuron is fully connected to all neurons in the previous layer, and where neurons in a single layer function completely independently and do not share any connections
  - The last fully-connected layer is called the output layer and in classification settings it represents the class scores



# Architecture of **CNNs**

- **CNNs** are a specific type of neural networks that are generally composed of the convolution layer, pooling layer, and fully connected layer
  - Sharing parameters across multiple image locations
  - Translation equivariant operation (in convolution layer)




https://stanford.edu/~shervine/teaching/cs-230/cheatsheet-convolutional-neural-networks

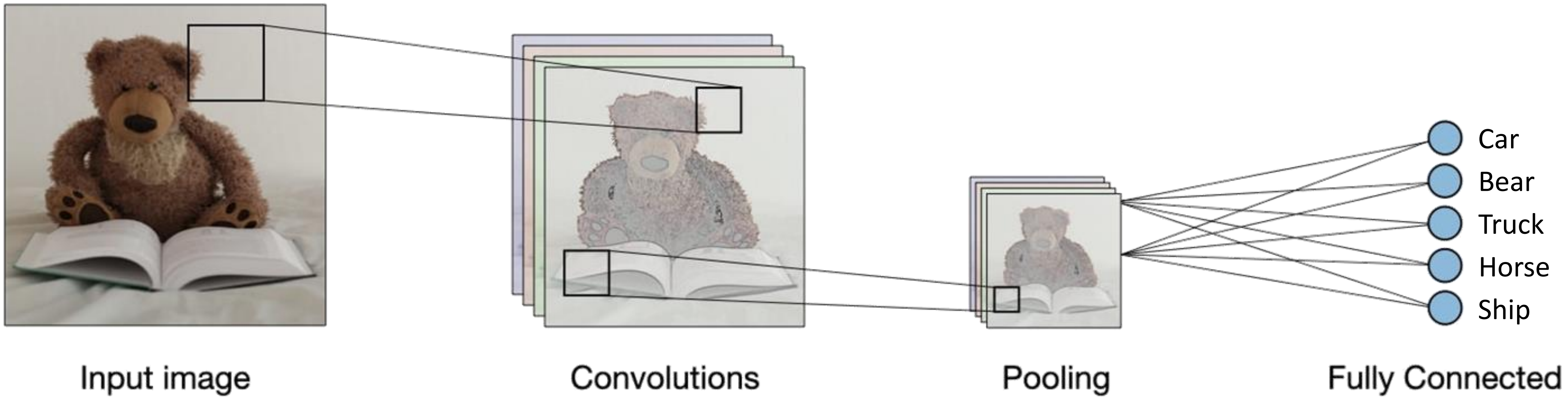
# Architecture of **CNNs**

- **Three main types of layers to build CNNs (ConvNet):**

— Convolutional layer

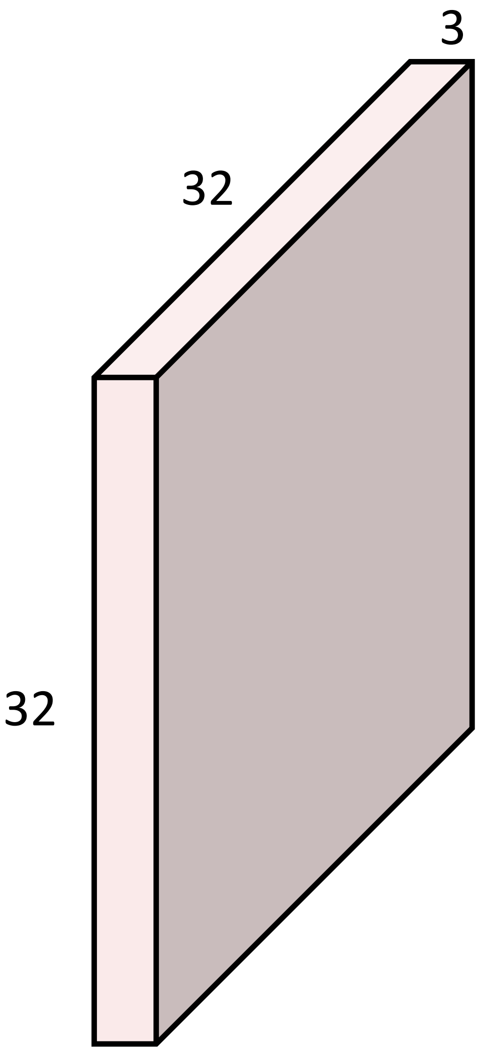
— Pooling layer

— Fully-Connected layer



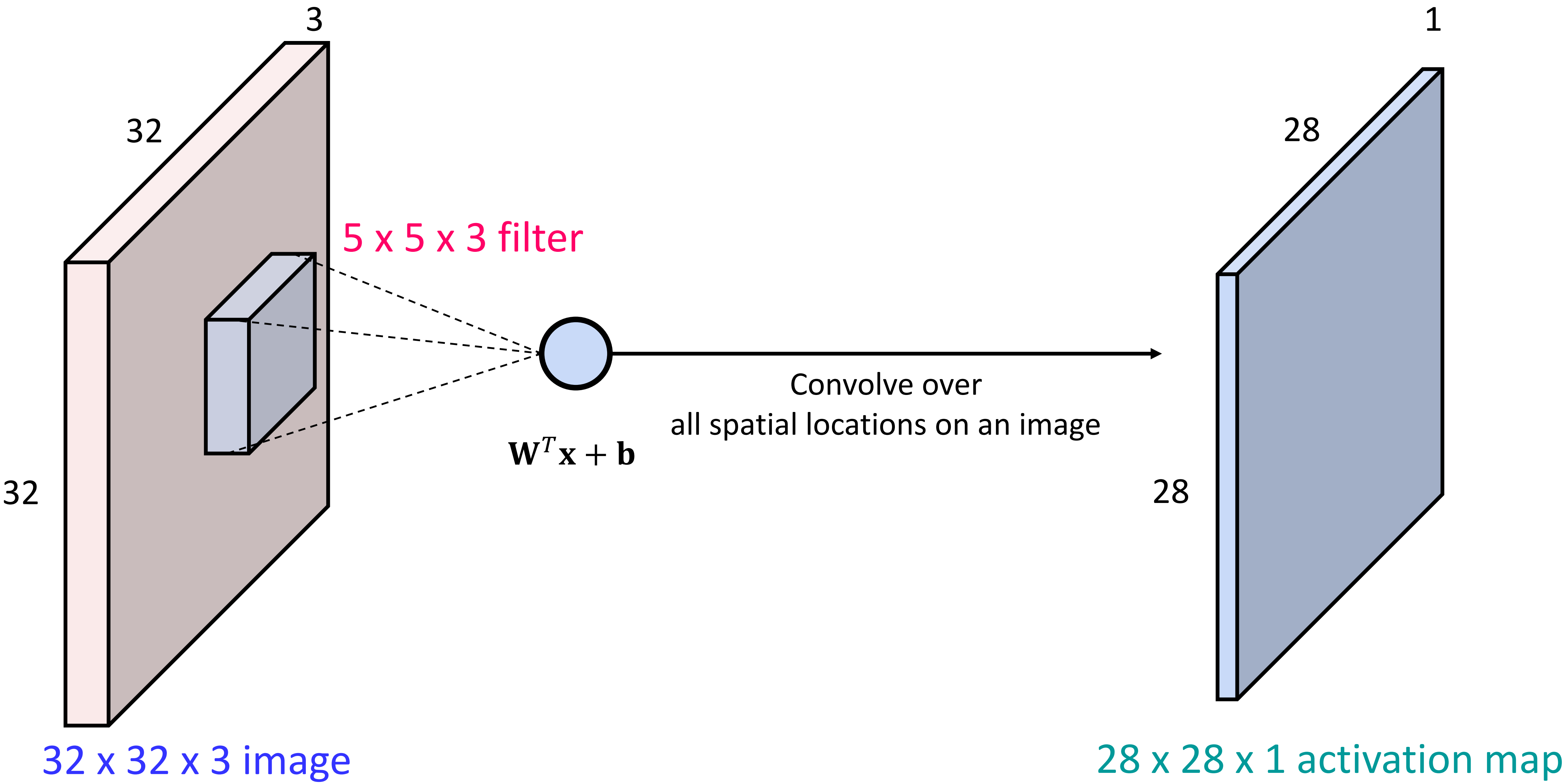

Example of ConvNet architecture

# Architecture of CNNs: **Convolution Layer**

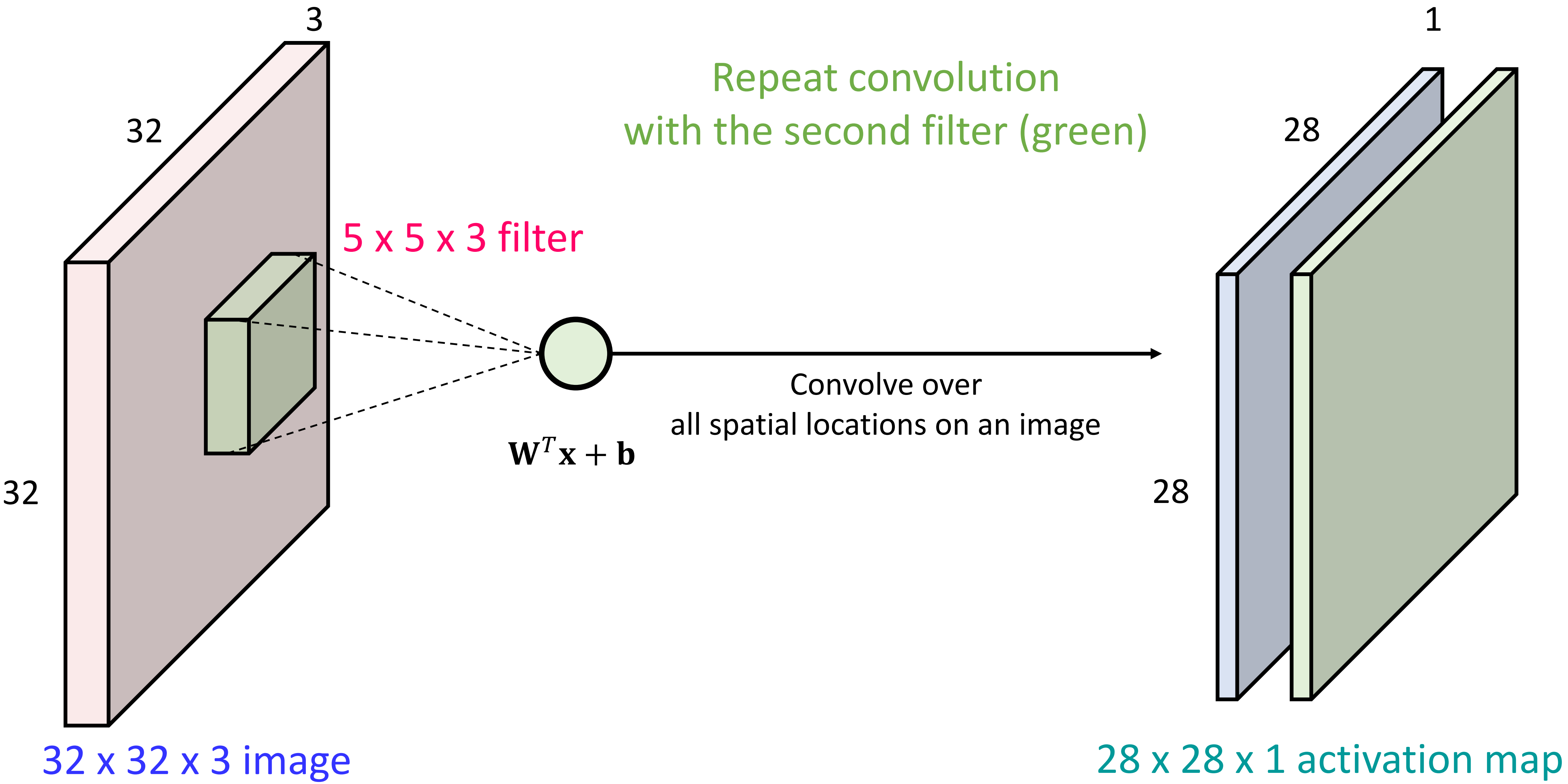



32 x 32 x 3 image

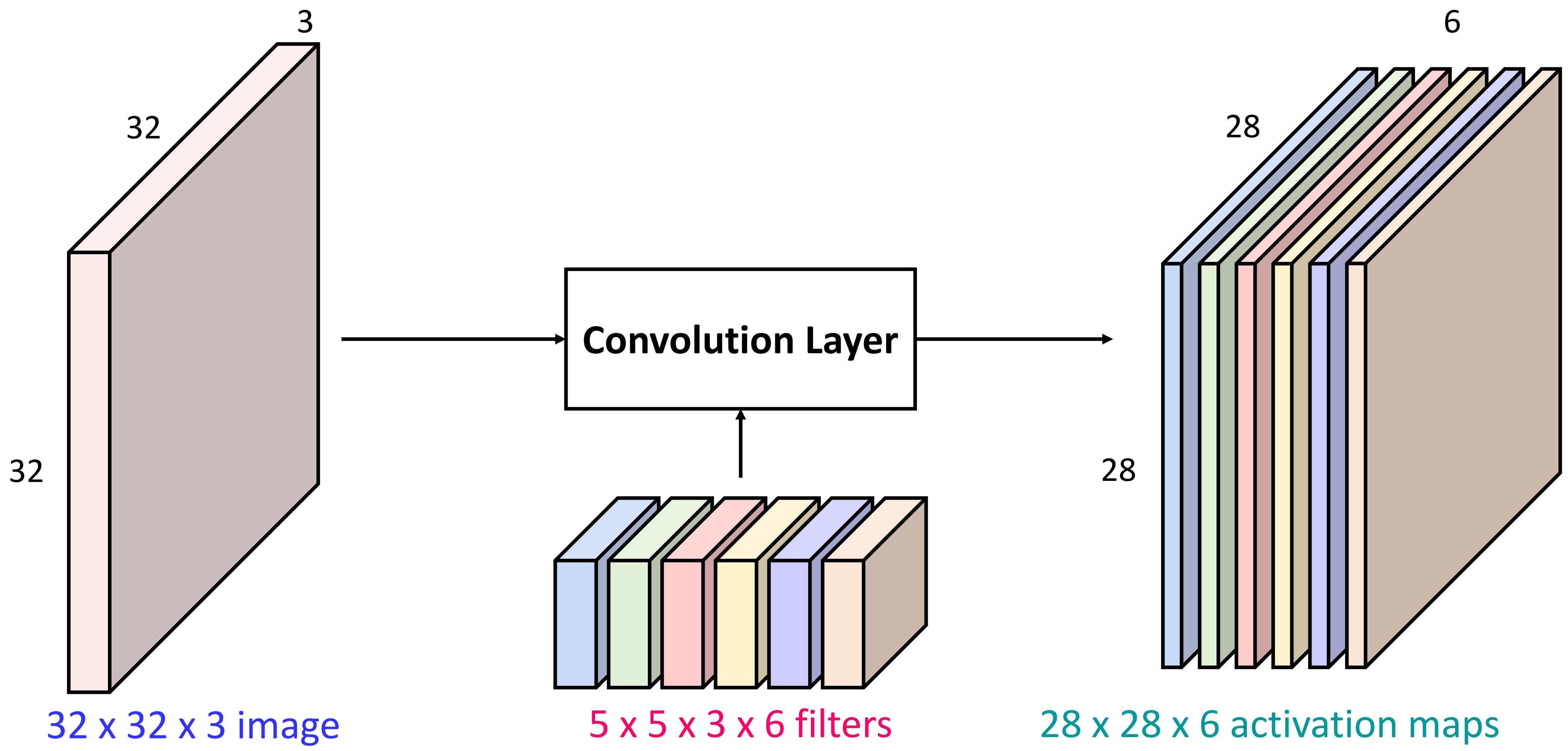
# Architecture of CNNs: **Convolution Layer**



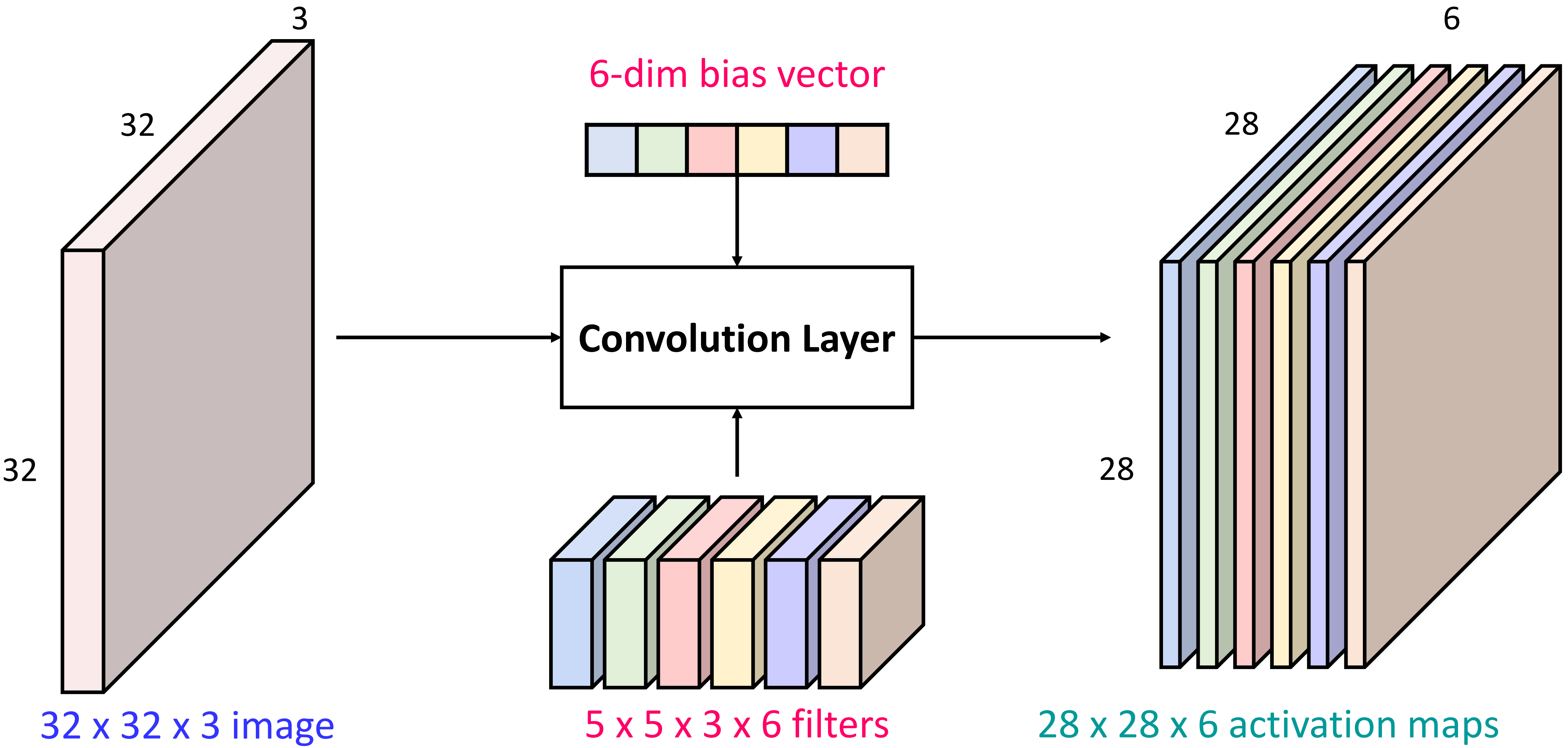
# Architecture of CNNs: **Convolution Layer**



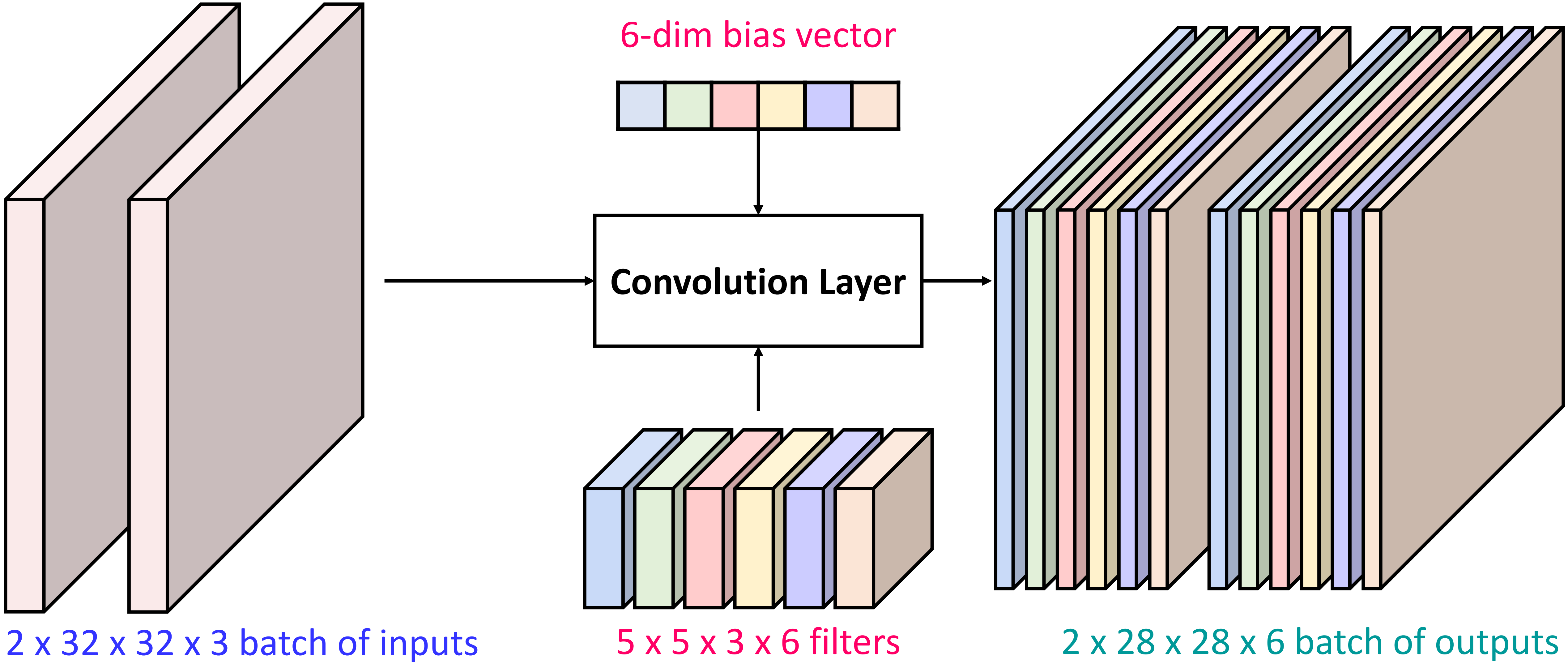
# Architecture of CNNs: **Convolution Layer**



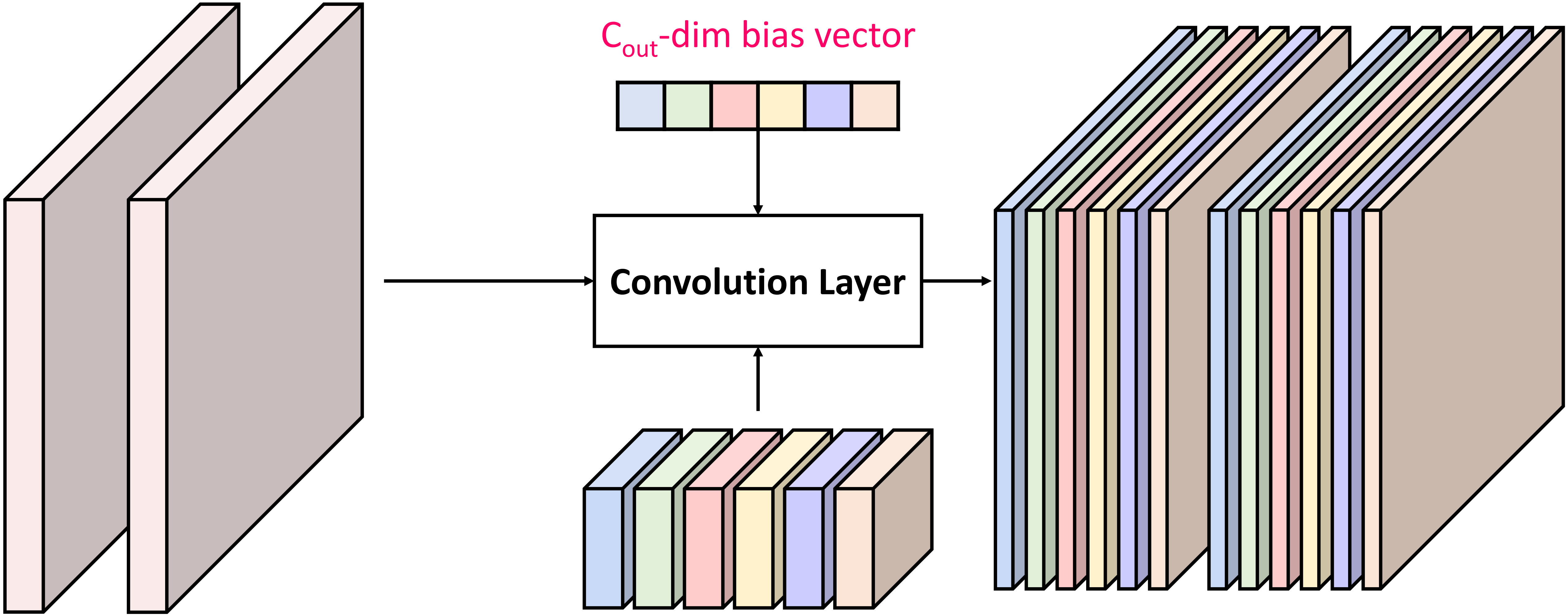
# Architecture of CNNs: **Convolution Layer**



# Architecture of CNNs: **Convolution Layer**



# Architecture of CNNs: **Convolution Layer**



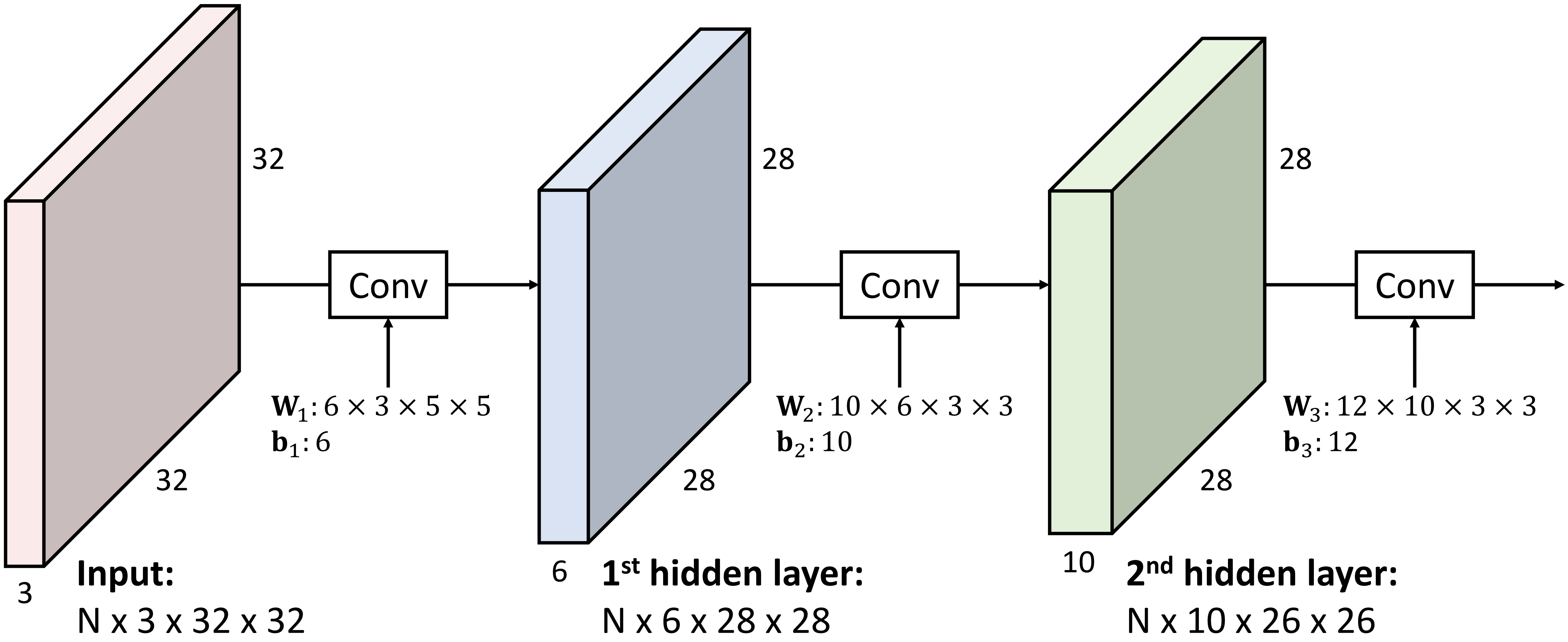

$N \times C_{in} \times H \times W$ batch of inputs

$C_{out} \times C_{in} \times K_h \times K_w$ filters

$N \times C_{out} \times H' \times W'$ batch of outputs

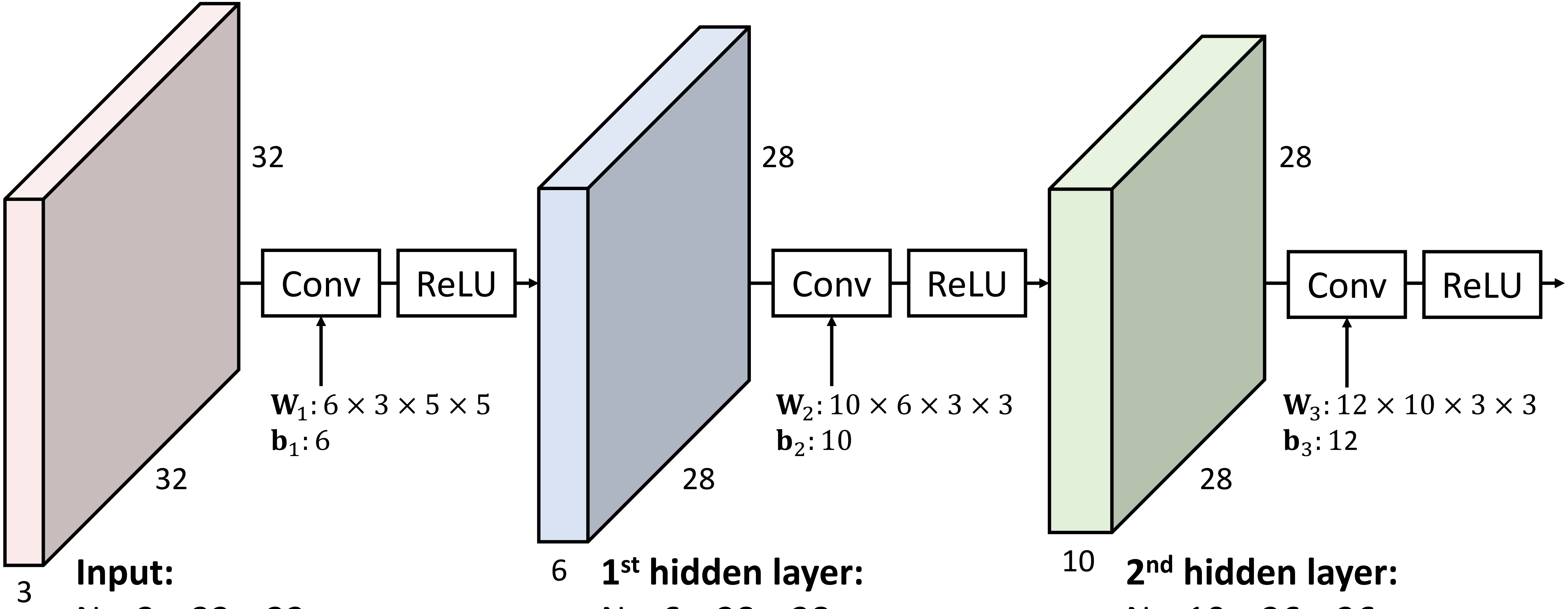
# Architecture of CNNs: **Convolution Layer**

- Stacking Convolutions



# Architecture of CNNs: **Convolution Layer**

- Stacking Convolutions



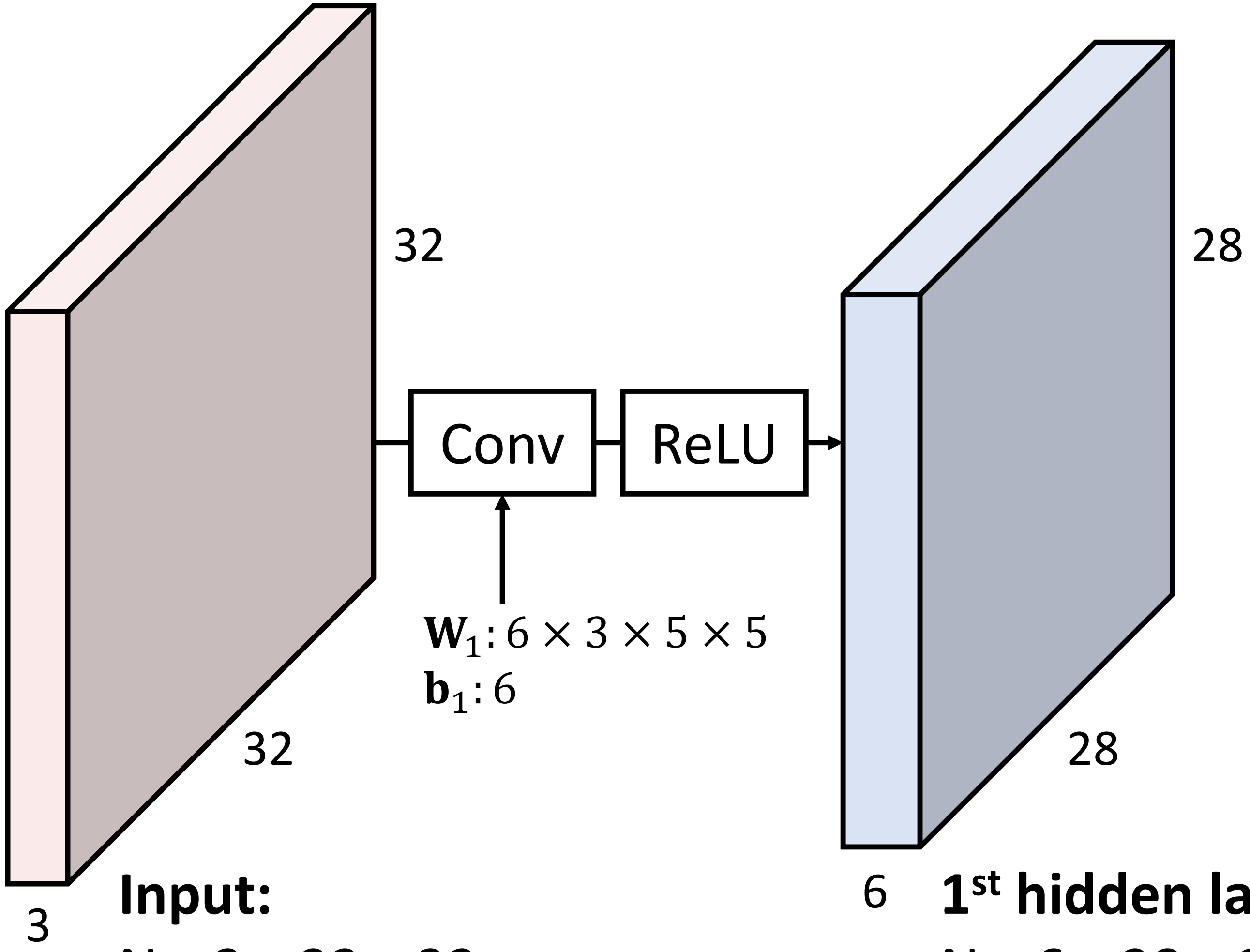

**Input:**
N x 3 x 32 x 32

**1st hidden layer:**
N x 6 x 28 x 28

**2nd hidden layer:**
N x 10 x 26 x 26

# Architecture of CNNs: **Convolution Layer**

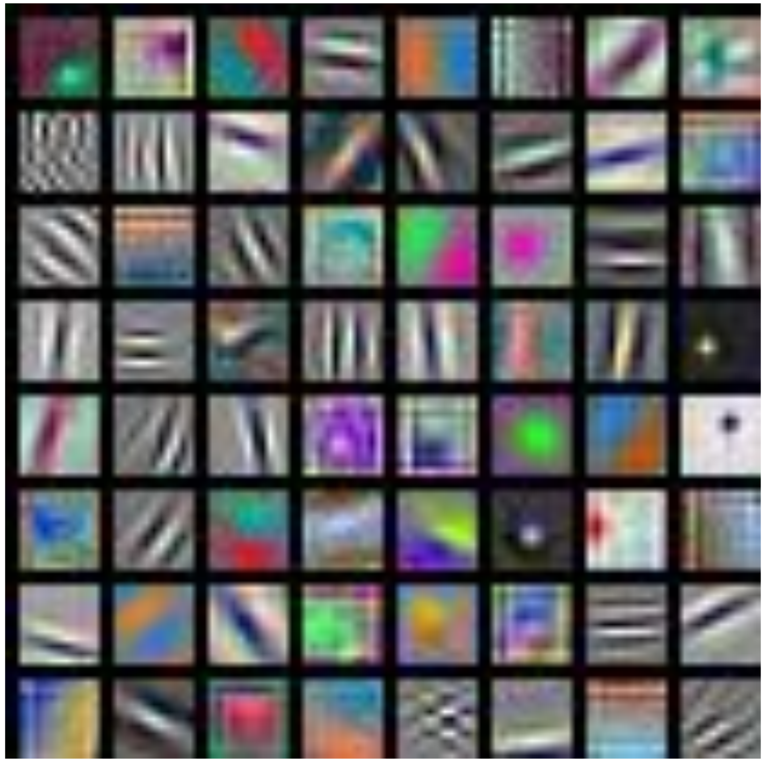
- What do convolution filters learn?




**Input:**
N x 3 x 32 x 32

**1st hidden layer:**
N x 6 x 28 x 28

Convolution filter at 1st layer learns the local image templates
(e.g., oriented edges, opposite colors, etc.)

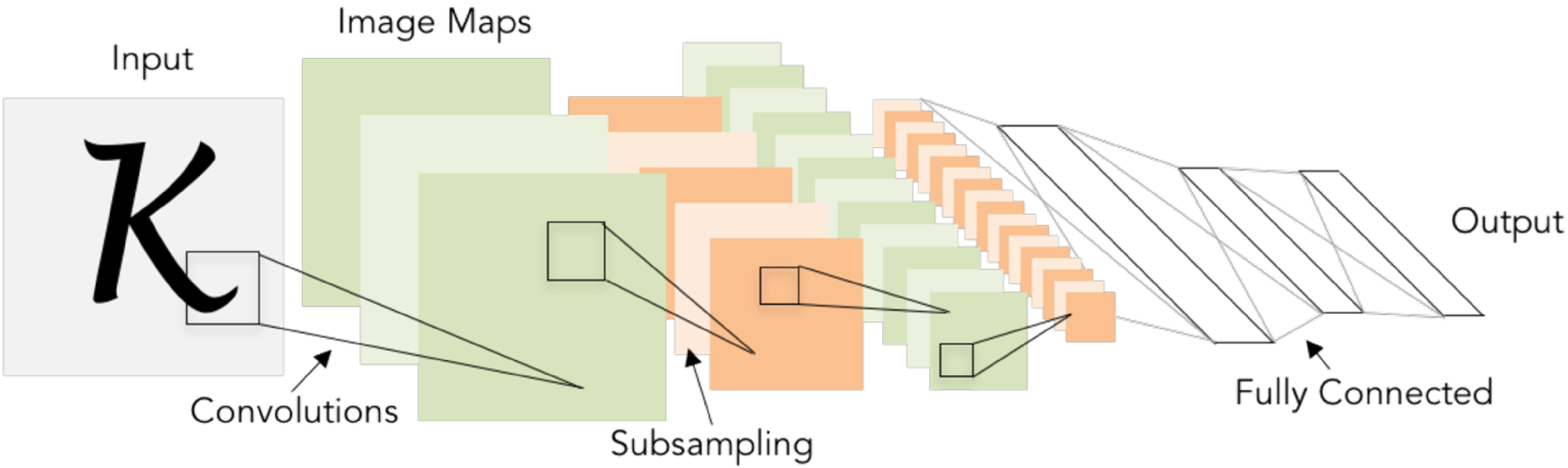


64 filters (3x11x11) in AlexNet

# Architecture of CNNs: **Convolution Layer**

- Receptive Fields (Kernel/filter size)

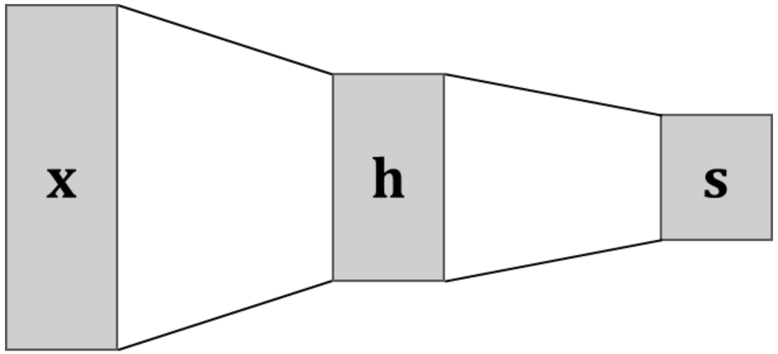
— For convolution with kernel size $K$, each element in the output depends on a $K \times K$ receptive field in the input



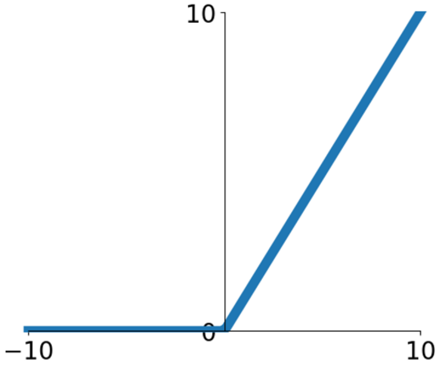

Y. Lecun *et al*., Gradient-based learning applied to document recognition, **Proceeding of IEEE, 1998**

# Components of **Convolutional Neural Networks** (CNNs)

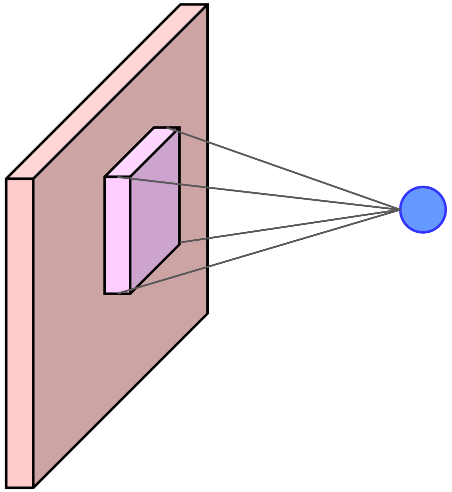
## Fully-Connected Layers (FC layers)



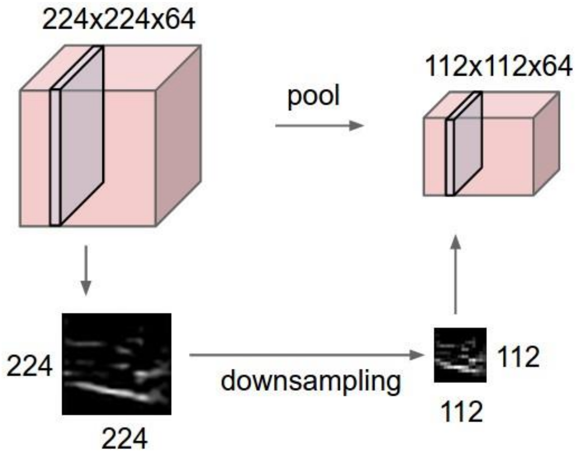

## Activation Function



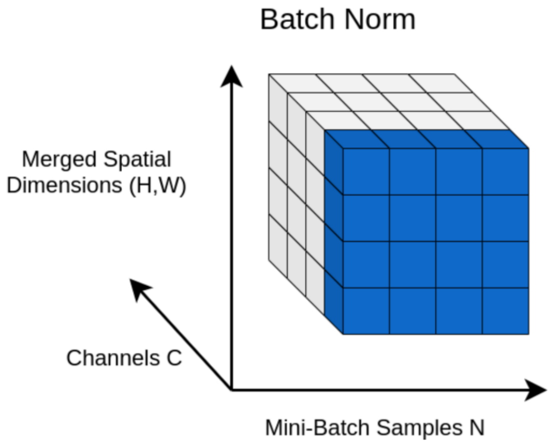

## Convolution Layers



## Pooling Layers

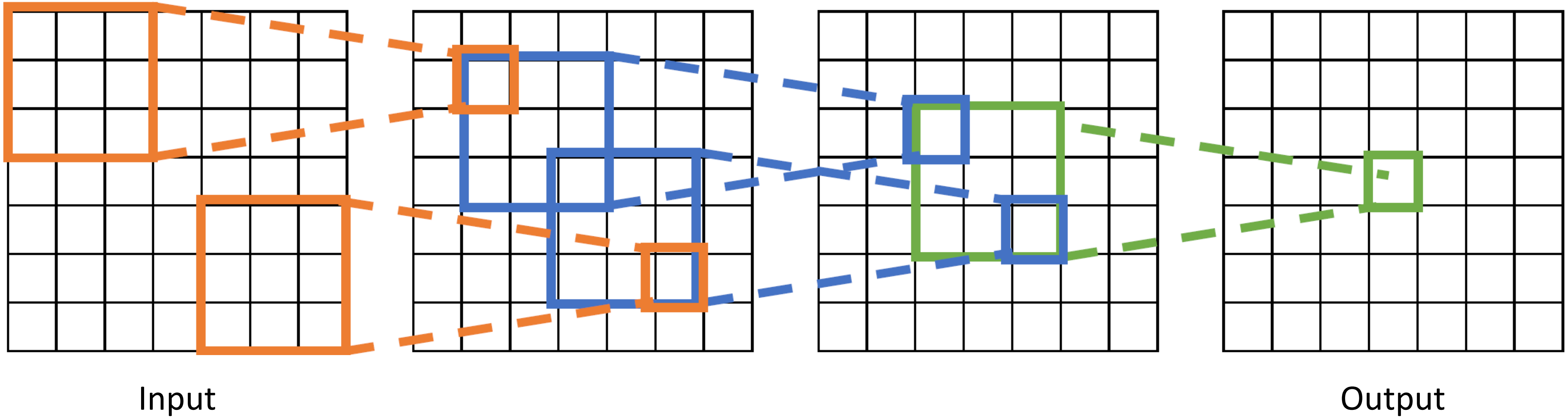



## Normalization



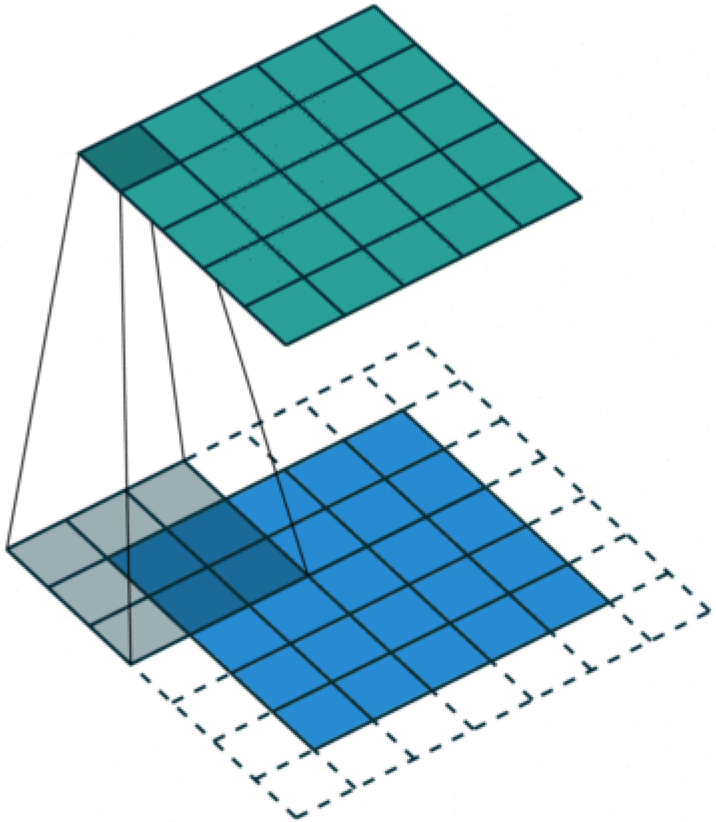
# Architecture of CNNs: **Convolution Layer**

- Receptive Fields (Kernel/filter size)
  - **Problem:** For large images, we need may layers for each output to see the whole size of image
  - **Solution:** Down-sampling in side the network

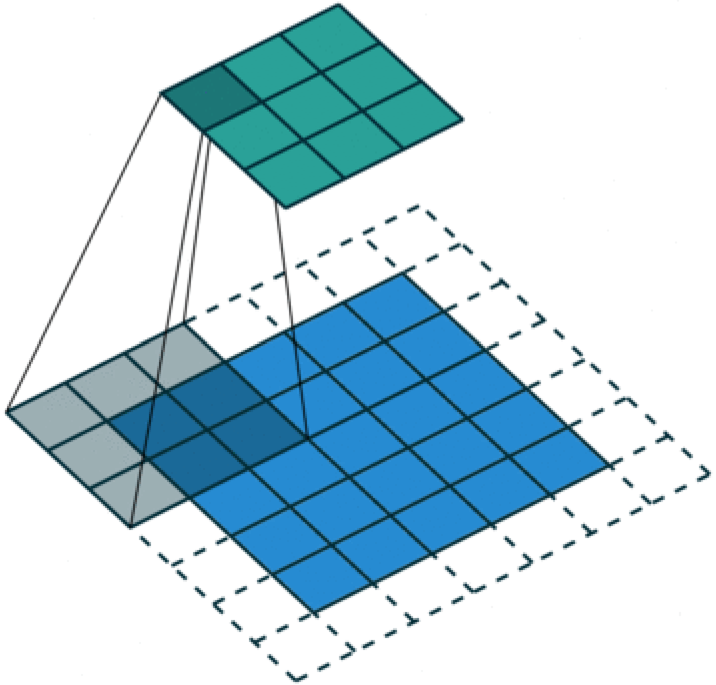


# Architecture of CNNs: **Convolution Layer**

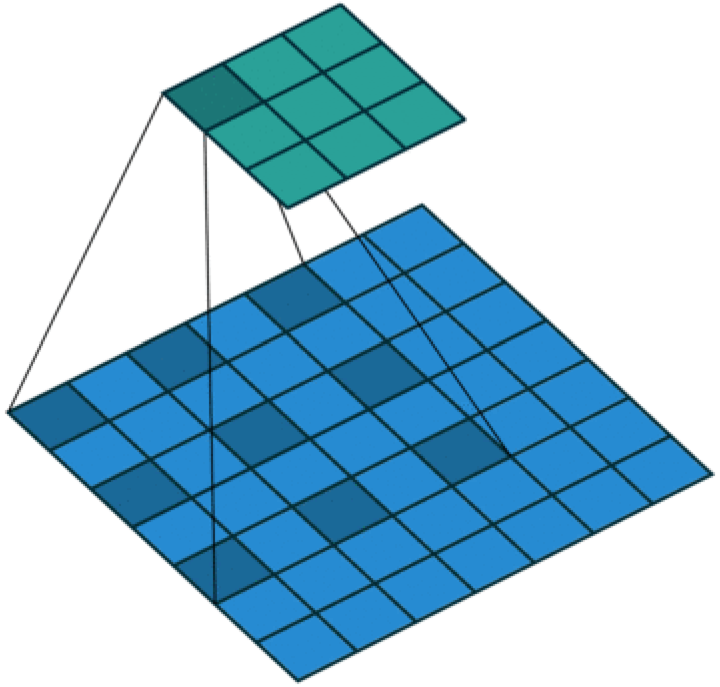
- **Various Convolutions (for down-sampling):**



Basic convolution



Strided convolution



Dilated convolution

https://github.com/vdumoulin/conv_arithmetic

# Architecture of CNNs: **Convolution Layer**

- Convolution Layer Functions in PyTorch

## CONV2D

CLASS `torch.nn.Conv2d(in_channels, out_channels, kernel_size, stride=1, padding=0, dilation=1, groups=1, bias=True, padding_mode='zeros', device=None, dtype=None)` [SOURCE]

Applies a 2D convolution over an input signal composed of several input planes.

In the simplest case, the output value of the layer with input size $(N, C_{\text{in}}, H, W)$ and output $(N, C_{\text{out}}, H_{\text{out}}, W_{\text{out}})$ can be precisely described as:

$$\text{out}(N_i, C_{\text{out}_j}) = \text{bias}(C_{\text{out}_j}) + \sum_{k=0}^{C_{\text{in}}-1} \text{weight}(C_{\text{out}_j}, k) \star \text{input}(N_i, k)$$

where $\star$ is the valid 2D cross-correlation operator, $N$ is a batch size, $C$ denotes a number of channels, $H$ is a height of input planes in pixels, and $W$ is width in pixels.

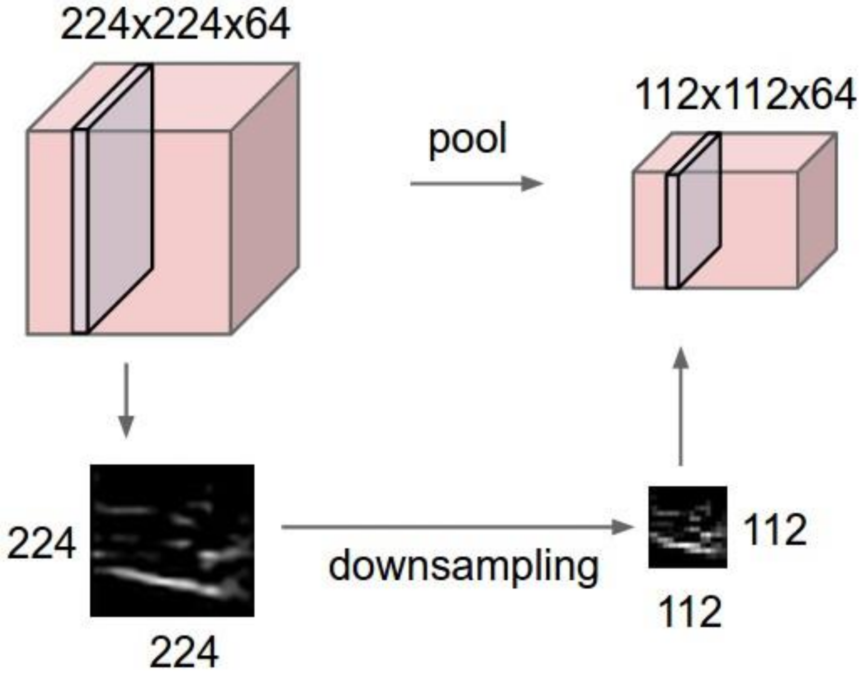
**Examples:**

```
>>> # With square kernels and equal stride
>>> m = nn.Conv2d(16, 33, 3, stride=2)
>>> # non-square kernels and unequal stride and with padding
>>> m = nn.Conv2d(16, 33, (3, 5), stride=(2, 1), padding=(4, 2))
>>> # non-square kernels and unequal stride and with padding and dilation
>>> m = nn.Conv2d(16, 33, (3, 5), stride=(2, 1), padding=(4, 2), dilation=(3, 1))
```

https://pytorch.org/docs/stable/generated/torch.nn.Conv2d.html

# Architecture of CNNs: **Pooling Layer**

- Pooling layer is to progressively reduce the spatial size of the representation to reduce the amount of parameters and computation in the network
- It operates independently on every depth slice of the input and resizes it spatially, using the MAX operation



# Architecture of CNNs: **Pooling Layer**

- MAX Pooling

| | | | |
|---|---|---|---|
| 2 | 3 | 5 | 4 |
| 5 | 6 | 7 | 8 |
| 3 | 2 | 1 | 0 |
| 1 | 2 | 3 | 4 |

MAX pooling with
2x2 kernel size and stride 2

| | |
|---|---|
| **6** | **8** |
| **3** | **4** |

# Architecture of CNNs: **Pooling Layer**

- Average Pooling

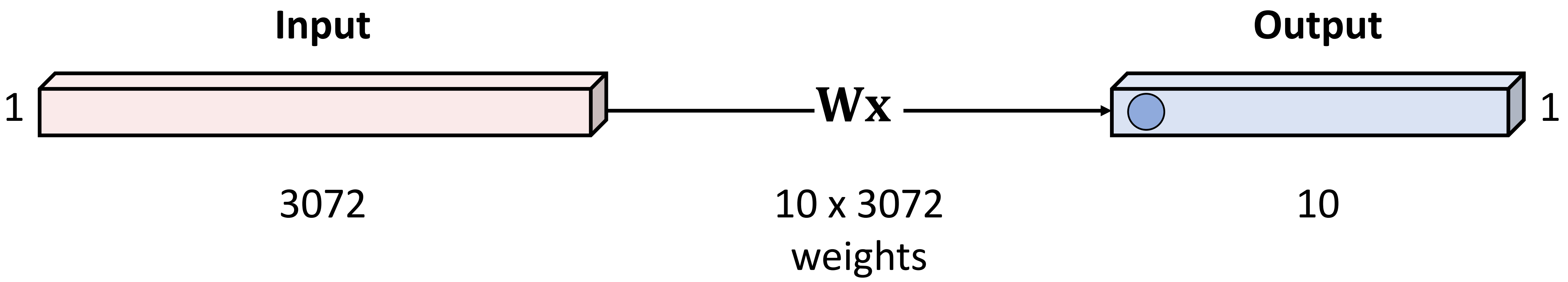
| | | | |
|---|---|---|---|
| 2 | 3 | 5 | 4 |
| 5 | 6 | 7 | 8 |
| 3 | 2 | 1 | 0 |
| 1 | 2 | 3 | 4 |

Average pooling with
2x2 kernel size and stride 2

| | |
|---|---|
| **4** | **6** |
| **2** | **2** |

# Architecture of CNNs: **Fully-Connected Layer**

- Neurons in a fully connected layer have full connections to all activations in the previous layer, as seen in regular Neural Networks



# **Example:** Architecture of CNNs

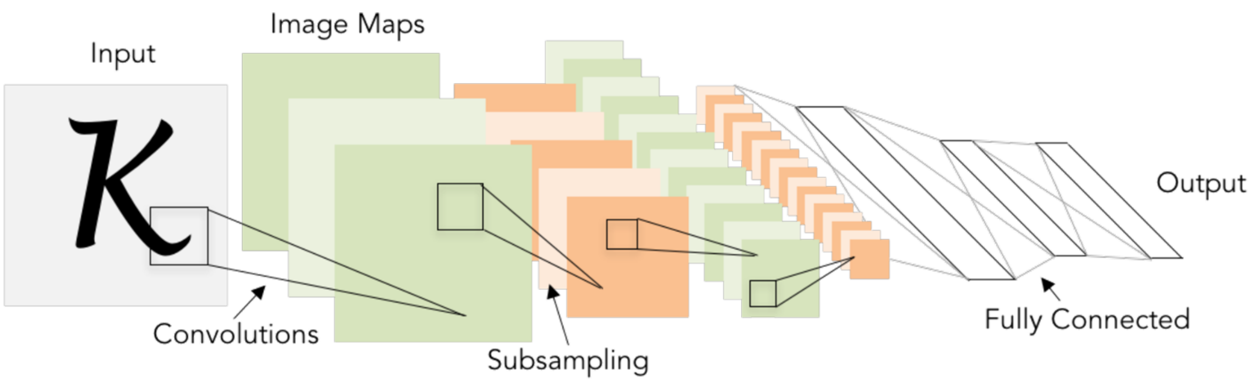
- LeNet-5

— Spartial Size **decreases**

  - By pooling and strided convolution

— Number of channels **increases**

  - Total volume is preserved

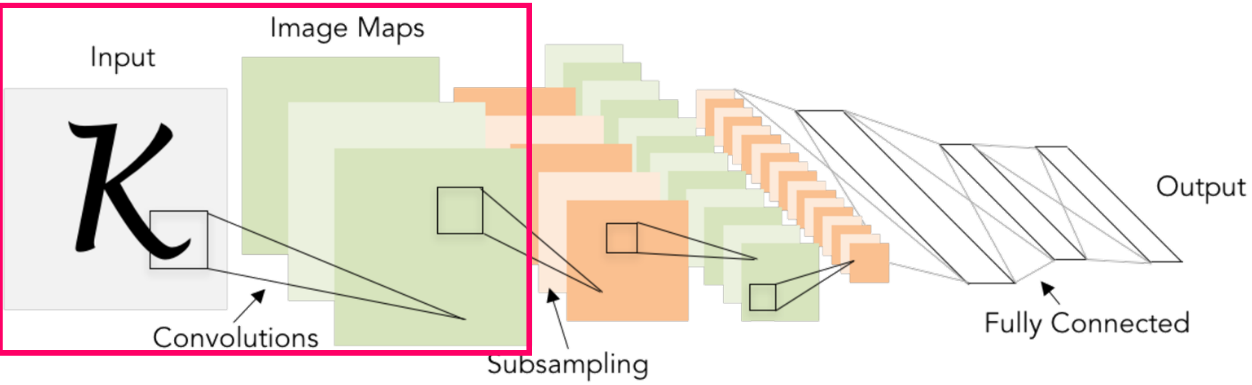



| Layer | Output Size | Weight Size |
|---|---|---|
| Input | 1 x 28 x 28 | |
| Conv ($C_{out}$=20, K=5, P=2, S=1) | 20 x 28 x 28 | 20 x 1 x 5 x 5 |
| ReLU | 20 x 28 x 28 | |
| MaxPool (K=2, S=2) | 20 x 14 x 14 | |
| Conv ($C_{out}$=50, K=5, P=2, S=1) | 50 x 14 x 14 | 50 x 20 x 5 x 5 |
| ReLU | 50 x 14 x 14 | |
| MaxPool (K=2, S=2) | 50 x 7 x 7 | |
| Flatten | 2,450 | |
| FC layer (2,450→500) | 500 | 2,450 x 500 |
| ReLU | 500 | |
| FC layer (500→10) | 10 | 500 x 10 |

Y. Lecun *et al*., Gradient-based learning applied to document recognition, **Proceeding of IEEE, 1998**

# **Example:** Architecture of CNNs

- LeNet-5
  - — Spartial Size **decreases**
    - By pooling and strided convolution
  - — Number of channels **increases**
    - Total volume is preserved

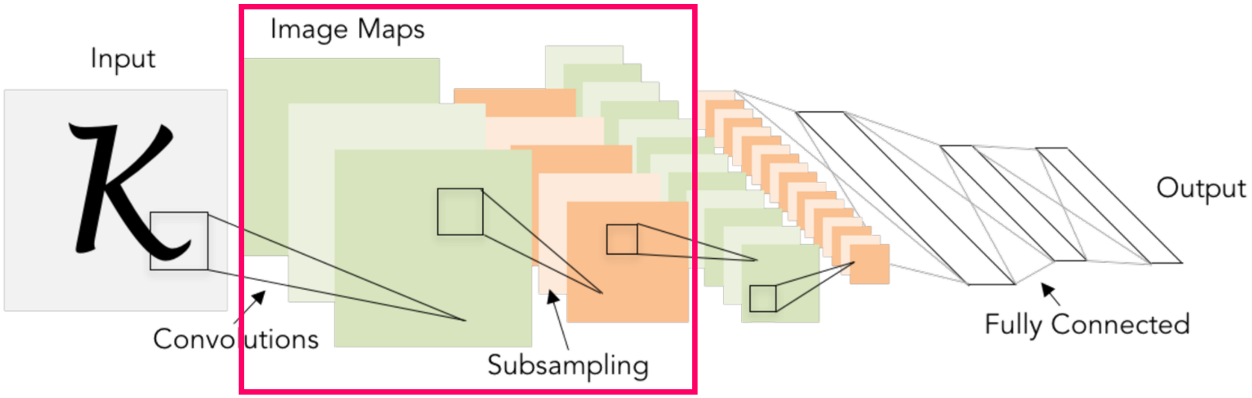



| Layer | Output Size | Weight Size |
|---|---|---|
| Input | 1 x 28 x 28 | |
| Conv ($C_{out}$=20, K=5, P=2, S=1) | 20 x 28 x 28 | 20 x 1 x 5 x 5 |
| ReLU | 20 x 28 x 28 | |
| MaxPool (K=2, S=2) | 20 x 14 x 14 | |
| Conv ($C_{out}$=50, K=5, P=2, S=1) | 50 x 14 x 14 | 50 x 20 x 5 x 5 |
| ReLU | 50 x 14 x 14 | |
| MaxPool (K=2, S=2) | 50 x 7 x 7 | |
| Flatten | 2,450 | |
| FC layer (2,450→500) | 500 | 2,450 x 500 |
| ReLU | 500 | |
| FC layer (500→10) | 10 | 500 x 10 |

Y. Lecun *et al*., Gradient-based learning applied to document recognition, **Proceeding of IEEE, 1998**

# **Example:** Architecture of CNNs

- LeNet-5
  - — Spartial Size **decreases**
    - By pooling and strided convolution
  - — Number of channels **increases**
    - Total volume is preserved

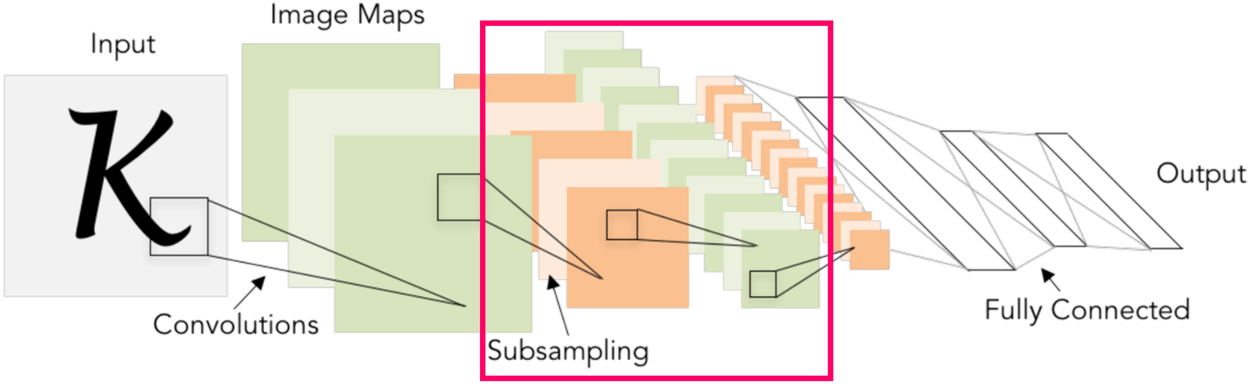



| Layer | Output Size | Weight Size |
|---|---|---|
| Input | 1 x 28 x 28 | |
| Conv ($C_{out}$=20, K=5, P=2, S=1) | 20 x 28 x 28 | 20 x 1 x 5 x 5 |
| ReLU | 20 x 28 x 28 | |
| MaxPool (K=2, S=2) | 20 x 14 x 14 | |
| Conv ($C_{out}$=50, K=5, P=2, S=1) | 50 x 14 x 14 | 50 x 20 x 5 x 5 |
| ReLU | 50 x 14 x 14 | |
| MaxPool (K=2, S=2) | 50 x 7 x 7 | |
| Flatten | 2,450 | |
| FC layer (2,450→500) | 500 | 2,450 x 500 |
| ReLU | 500 | |
| FC layer (500→10) | 10 | 500 x 10 |

Y. Lecun *et al.*, Gradient-based learning applied to document recognition, **Proceeding of IEEE, 1998**

# **Example:** Architecture of CNNs

- LeNet-5
- — Spartial Size **decreases**
  - By pooling and strided convolution
- — Number of channels **increases**
  - Total volume is preserved

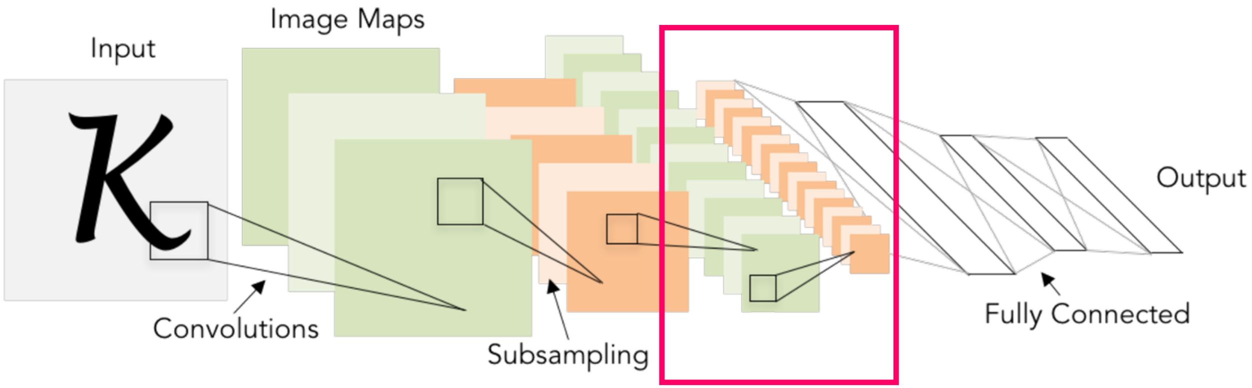



| Layer | Output Size | Weight Size |
|---|---|---|
| Input | 1 x 28 x 28 | |
| Conv ($C_{out}$=20, K=5, P=2, S=1) | 20 x 28 x 28 | 20 x 1 x 5 x 5 |
| ReLU | 20 x 28 x 28 | |
| MaxPool (K=2, S=2) | 20 x 14 x 14 | |
| Conv ($C_{out}$=50, K=5, P=2, S=1) | 50 x 14 x 14 | 50 x 20 x 5 x 5 |
| ReLU | 50 x 14 x 14 | |
| MaxPool (K=2, S=2) | 50 x 7 x 7 | |
| Flatten | 2,450 | |
| FC layer (2,450→500) | 500 | 2,450 x 500 |
| ReLU | 500 | |
| FC layer (500→10) | 10 | 500 x 10 |

Y. Lecun *et al*., Gradient-based learning applied to document recognition, **Proceeding of IEEE, 1998**

# **Example:** Architecture of CNNs

- LeNet-5
  - Spartial Size **decreases**
    - By pooling and strided convolution
  - Number of channels **increases**
    - Total volume is preserved

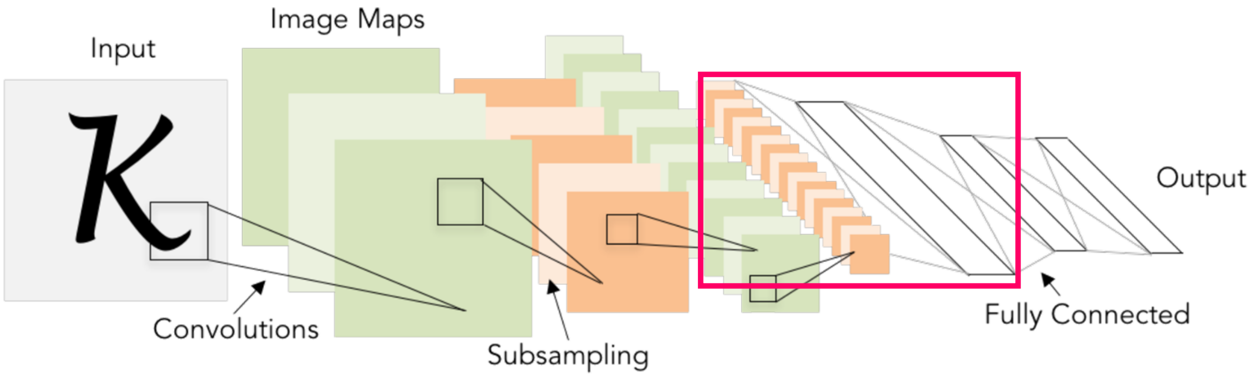



| Layer | Output Size | Weight Size |
|---|---|---|
| Input | 1 x 28 x 28 | |
| Conv ($C_{out}$=20, K=5, P=2, S=1) | 20 x 28 x 28 | 20 x 1 x 5 x 5 |
| ReLU | 20 x 28 x 28 | |
| MaxPool (K=2, S=2) | 20 x 14 x 14 | |
| Conv ($C_{out}$=50, K=5, P=2, S=1) | 50 x 14 x 14 | 50 x 20 x 5 x 5 |
| ReLU | 50 x 14 x 14 | |
| MaxPool (K=2, S=2) | 50 x 7 x 7 | |
| Flatten | 2,450 | |
| FC layer (2,450→500) | 500 | 2,450 x 500 |
| ReLU | 500 | |
| FC layer (500→10) | 10 | 500 x 10 |

Y. Lecun *et al.*, Gradient-based learning applied to document recognition, **Proceeding of IEEE, 1998**

# **Example:** Architecture of CNNs

- LeNet-5
  - Spartial Size **decreases**
    - By pooling and strided convolution
  - Number of channels **increases**
    - Total volume is preserved

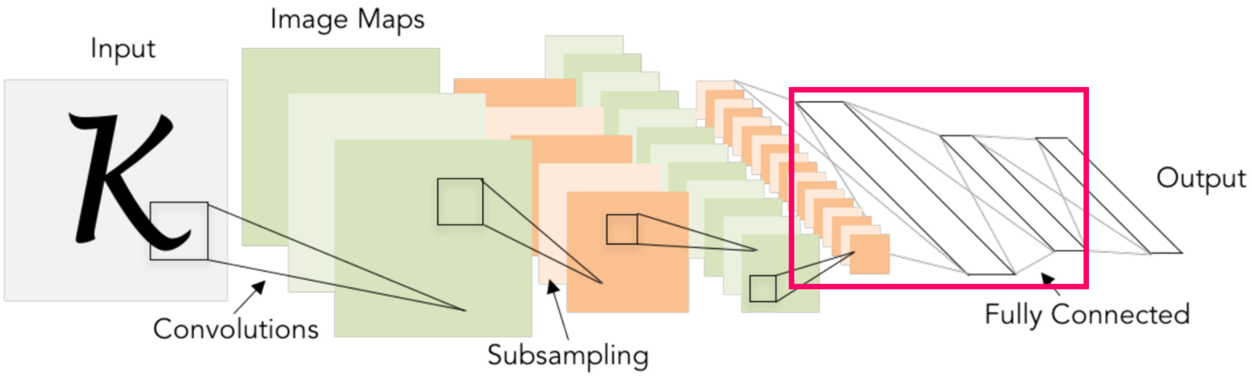



| Layer | Output Size | Weight Size |
|---|---|---|
| Input | 1 x 28 x 28 | |
| Conv ($C_{out}$=20, K=5, P=2, S=1) | 20 x 28 x 28 | 20 x 1 x 5 x 5 |
| ReLU | 20 x 28 x 28 | |
| MaxPool (K=2, S=2) | 20 x 14 x 14 | |
| Conv ($C_{out}$=50, K=5, P=2, S=1) | 50 x 14 x 14 | 50 x 20 x 5 x 5 |
| ReLU | 50 x 14 x 14 | |
| MaxPool (K=2, S=2) | 50 x 7 x 7 | |
| Flatten | 2,450 | |
| FC layer (2,450→500) | 500 | 2,450 x 500 |
| ReLU | 500 | |
| FC layer (500→10) | 10 | 500 x 10 |

Y. Lecun *et al.*, Gradient-based learning applied to document recognition, **Proceeding of IEEE, 1998**

# **Example:** Architecture of CNNs

- LeNet-5
  - Spartial Size **decreases**
    - By pooling and strided convolution
  - Number of channels **increases**
    - Total volume is preserved




| Layer | Output Size | Weight Size |
|---|---|---|
| Input | 1 x 28 x 28 | |
| Conv ($C_{out}$=20, K=5, P=2, S=1) | 20 x 28 x 28 | 20 x 1 x 5 x 5 |
| ReLU | 20 x 28 x 28 | |
| MaxPool (K=2, S=2) | 20 x 14 x 14 | |
| Conv ($C_{out}$=50, K=5, P=2, S=1) | 50 x 14 x 14 | 50 x 20 x 5 x 5 |
| ReLU | 50 x 14 x 14 | |
| MaxPool (K=2, S=2) | 50 x 7 x 7 | |
| Flatten | 2,450 | |
| FC layer (2,450→500) | 500 | 2,450 x 500 |
| ReLU | 500 | |
| FC layer (500→10) | 10 | 500 x 10 |

Y. Lecun *et al*., Gradient-based learning applied to document recognition, **Proceeding of IEEE, 1998**

# **Example:** Architecture of CNNs

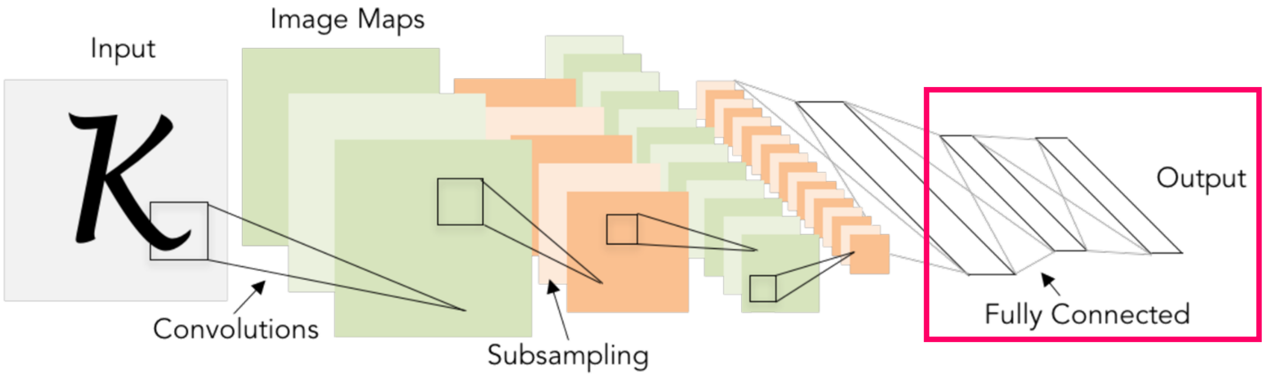
- LeNet-5

— Spartial Size **decreases**

- By pooling and strided convolution

— Number of channels **increases**

- Total volume is preserved

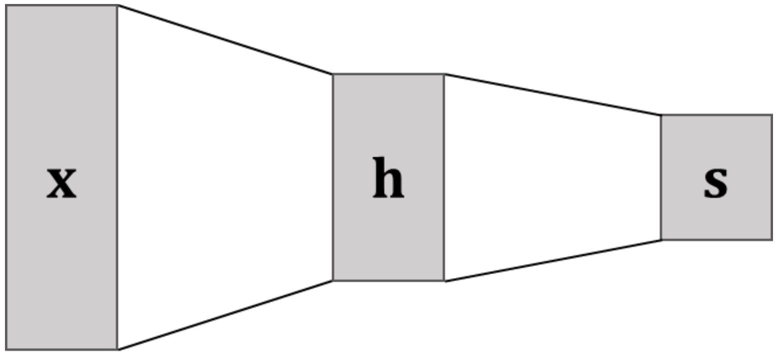



| Layer | Output Size | Weight Size |
|---|---|---|
| Input | 1 x 28 x 28 | |
| Conv ($C_{out}$=20, K=5, P=2, S=1) | 20 x 28 x 28 | 20 x 1 x 5 x 5 |
| ReLU | 20 x 28 x 28 | |
| MaxPool (K=2, S=2) | 20 x 14 x 14 | |
| Conv ($C_{out}$=50, K=5, P=2, S=1) | 50 x 14 x 14 | 50 x 20 x 5 x 5 |
| ReLU | 50 x 14 x 14 | |
| MaxPool (K=2, S=2) | 50 x 7 x 7 | |
| Flatten | 2,450 | |
| FC layer (2,450→500) | 500 | 2,450 x 500 |
| ReLU | 500 | |
| FC layer (500→10) | 10 | 500 x 10 |

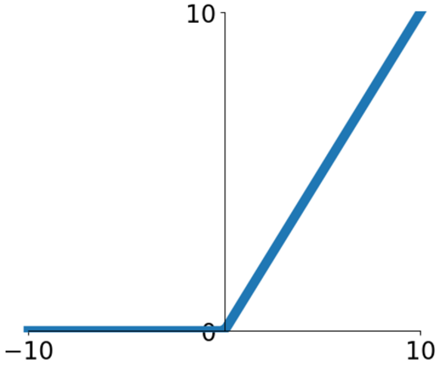
Y. Lecun *et al.*, Gradient-based learning applied to document recognition, **Proceeding of IEEE, 1998**

# Components of **Convolutional Neural Networks** (CNNs)

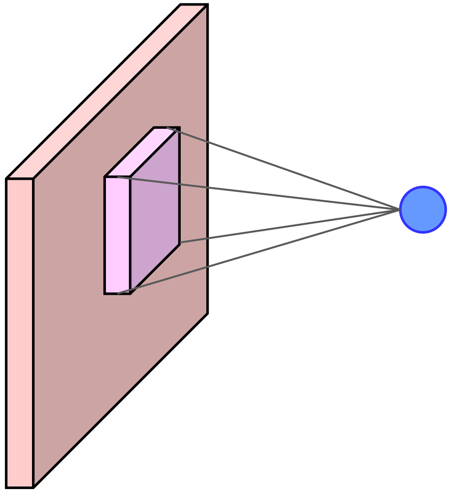
## Fully-Connected Layers (FC layers)



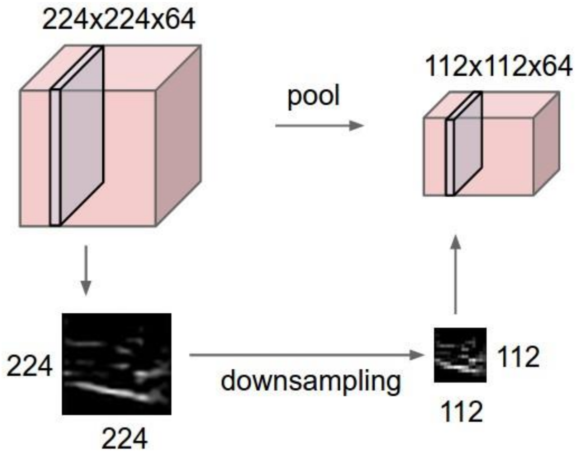

## Activation Function



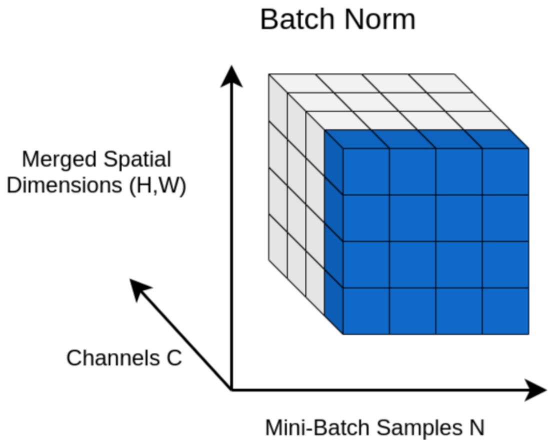

## Convolution Layers



## Pooling Layers

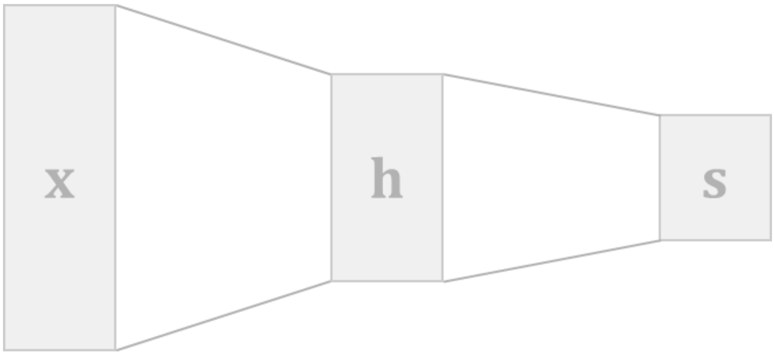



## Normalization

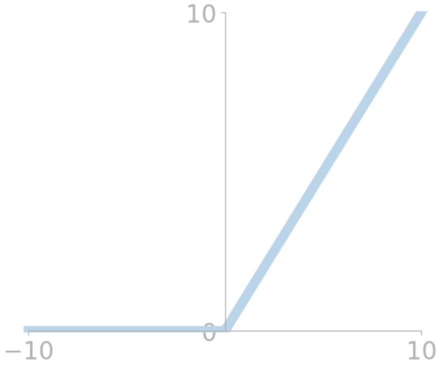


# Components of **Convolutional Neural Networks** (CNNs)

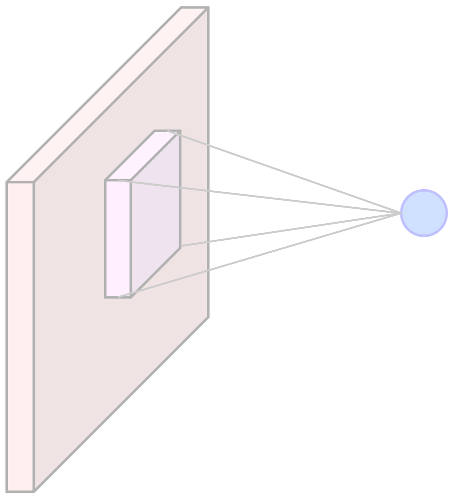
## Fully-Connected Layers (FC layers)



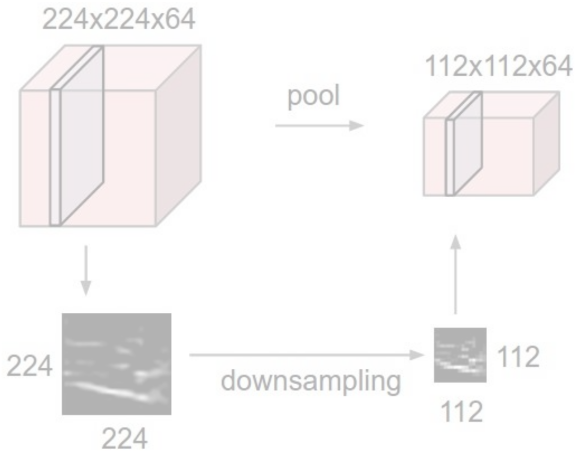

## Activation Function




## Convolution Layers

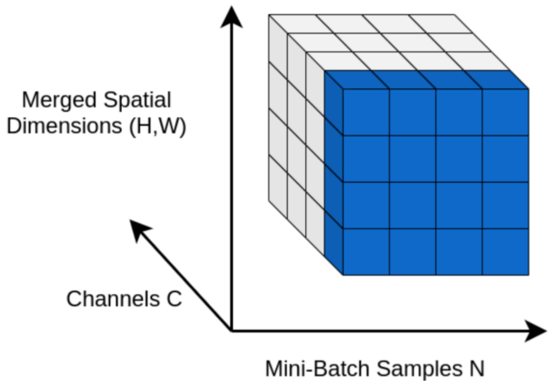


## Pooling Layers




## Normalization

Batch Norm



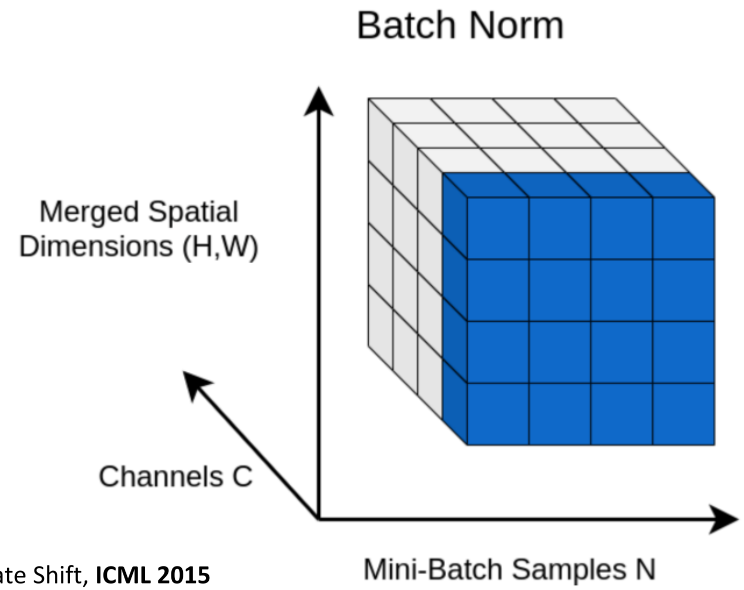
# Architecture of CNNs: **Batch Normalization**

- Normalize the outputs of a layer so they have zero mean and unit variance

— It helps **reduce** *internal covariate shift* and **improves** optimization

- Batch normalization is a **differentiable function**, so we can use it as an operator in our networks and backpropagation through it

$$\hat{x} = \frac{x - E[x]}{\sqrt{Var[x]}}$$




S. Ioffe and C. Szegedy, Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift, **ICML 2015**

# Architecture of CNNs: **Batch Normalization**

- Input: $\mathbf{x} \in \mathbb{R}^{N \times D}$

— Mean (per each channel)

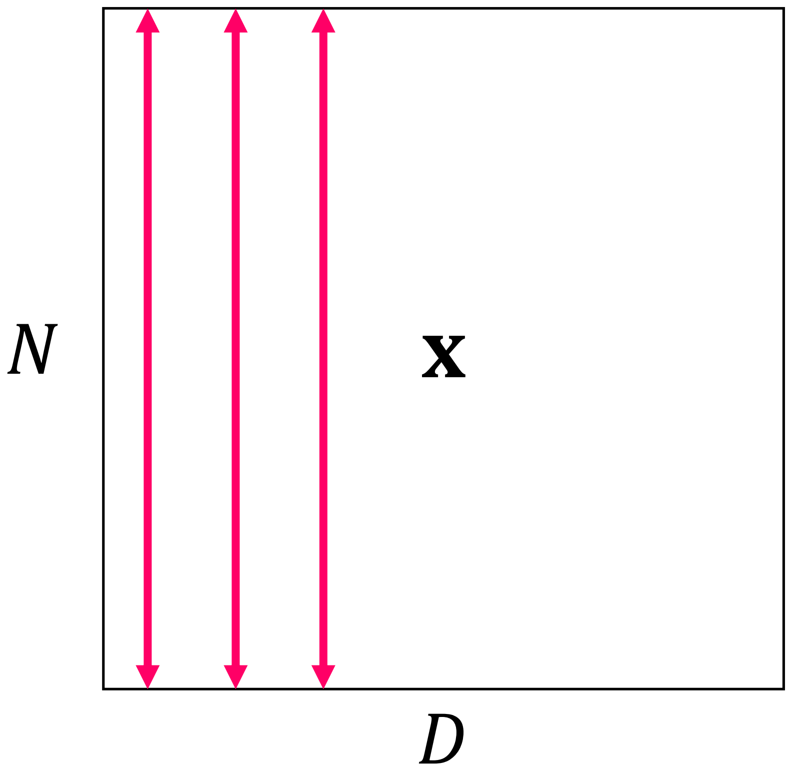
$$\mu_j = \frac{1}{N} \sum_{i=1}^{N} x_{i,j}$$

— Standard deviation (per each channel)

$$\sigma_j^2 = \frac{1}{N} \sum_{i=1}^{N} \left(x_{i,j} - \mu_j\right)^2$$

— Normalized input

$$\hat{x}_{i,j} = \frac{x_{i,j} - \mu_j}{\sqrt{\sigma_j^2 + \epsilon}}$$




S. Ioffe and C. Szegedy, Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift, **ICML 2015**

# Architecture of CNNs: **Batch Normalization**

- Input: $\mathbf{x} \in \mathbb{R}^{N \times D}$

— Mean (per each channel)

$$\mu_j = \frac{1}{N} \sum_{i=1}^{N} x_{i,j}$$

— Standard deviation (per each channel)

$$\sigma_j^2 = \frac{1}{N} \sum_{i=1}^{N} \left(x_{i,j} - \mu_j\right)^2$$

— Normalized input

$$\hat{x}_{i,j} = \frac{x_{i,j} - \mu_j}{\sqrt{\sigma_j^2 + \epsilon}}$$

- What if zero-mean and unit variance is too hard of a constraint?

— Learnable scale($\gamma \in \mathbb{R}^D$) & shift ($\beta \in \mathbb{R}^D$) parameters are used:

$$y_{i,j} = \gamma_j \hat{x}_{i,j} + \beta_j$$

S. Ioffe and C. Szegedy, Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift, **ICML 2015**

# Architecture of CNNs: **Batch Normalization**

- Input: $\mathbf{x} \in \mathbb{R}^{N \times D}$
  - Mean (per each channel)

$$\mu_j = \frac{1}{N}\sum_{i=1}^{N} x_{i,j}$$

  - Standard deviation (per each channel)

$$\sigma_j^2 = \frac{1}{N}\sum_{i=1}^{N} \left(x_{i,j} - \mu_j\right)^2$$

  - Normalized input

$$\hat{x}_{i,j} = \frac{x_{i,j} - \mu_j}{\sqrt{\sigma_j^2 + \epsilon}}$$

- We cannot estimate the mean, standard deviation, and normalized input of each minibatch at test time

S. Ioffe and C. Szegedy, Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift, **ICML 2015**

# Architecture of CNNs: **Batch Normalization at Test Time**

- Input: $\mathbf{x} \in \mathbb{R}^{N \times D}$
  - Mean (per each channel)

    $\mu_j =$ (Running) average of values seen during training

  - Standard deviation (per each channel)

    $\sigma_j^2 =$ (Running) average of values seen during training

  - Normalized input

    $$\hat{x}_{i,j} = \frac{x_{i,j} - \mu_j}{\sqrt{\sigma_j^2 + \epsilon}}$$

- We cannot estimate the mean, standard deviation, and normalized input of each minibatch at test time
  - During test, batch normalization becomes a linear operation, which can be fused with the previous Conv or FC layers

S. Ioffe and C. Szegedy, Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift, **ICML 2015**

# Architecture of CNNs: **Batch Normalization**

- **Advantages of Batch Normalization:**

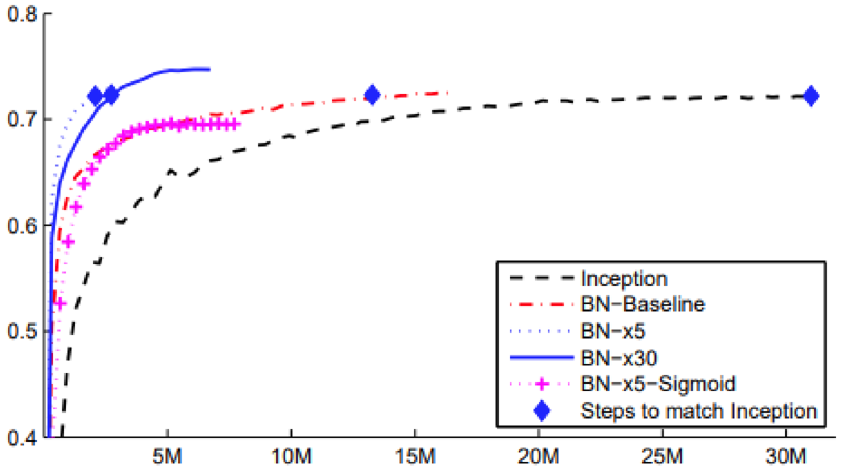
— Makes deep networks much easier to train

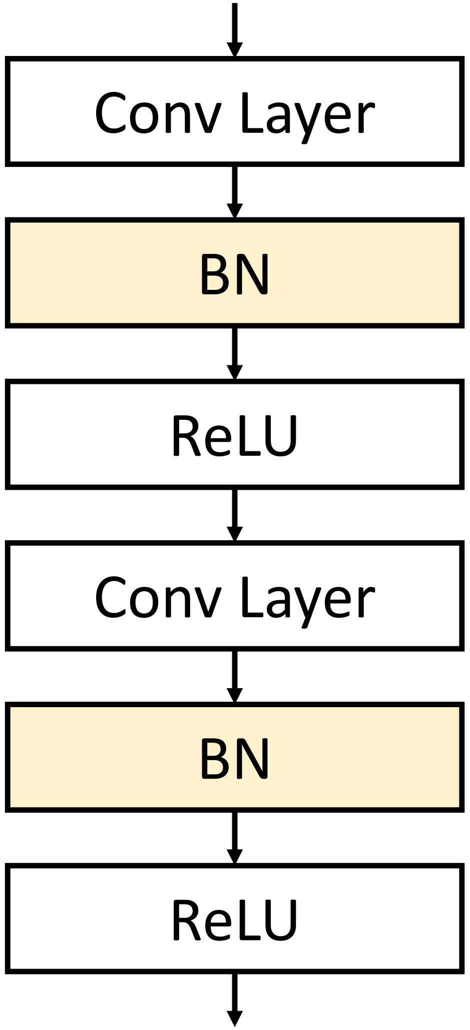
— Allows higher learning rates, faster convergence

— Networks become more robust to initialization

— Acts as regularization during training

— Zero overhead at test-time

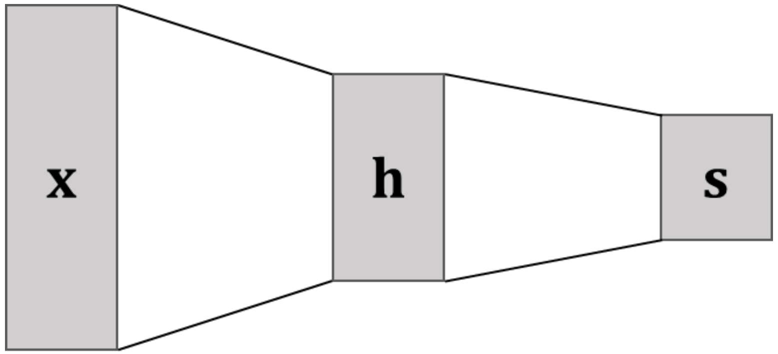


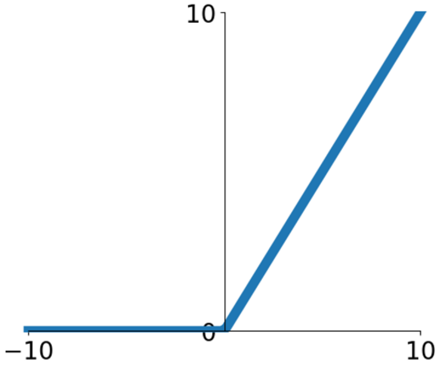

S. Ioffe and C. Szegedy, Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift, **ICML 2015**

# **Summary:** Components of CNNs

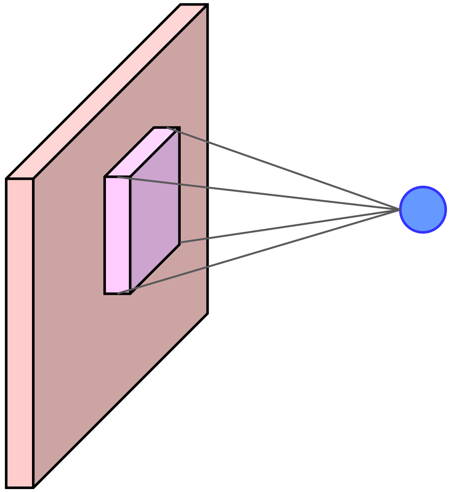
## Fully-Connected Layers (FC layers)



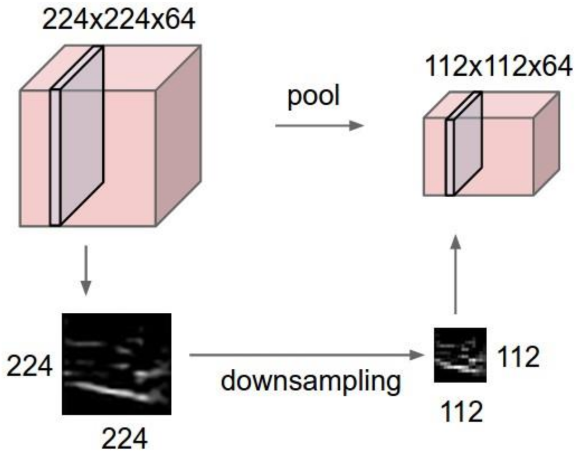

## Activation Function



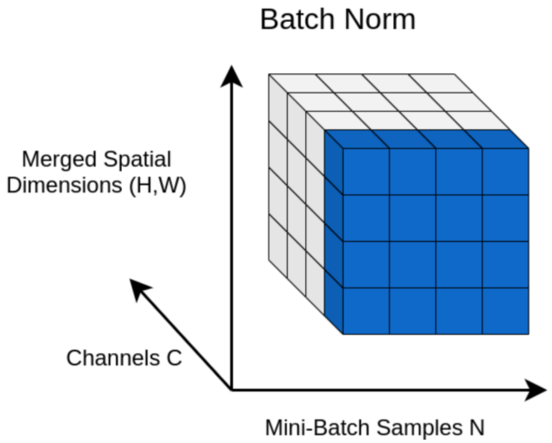

## Convolution Layers



## Pooling Layers




## Normalization



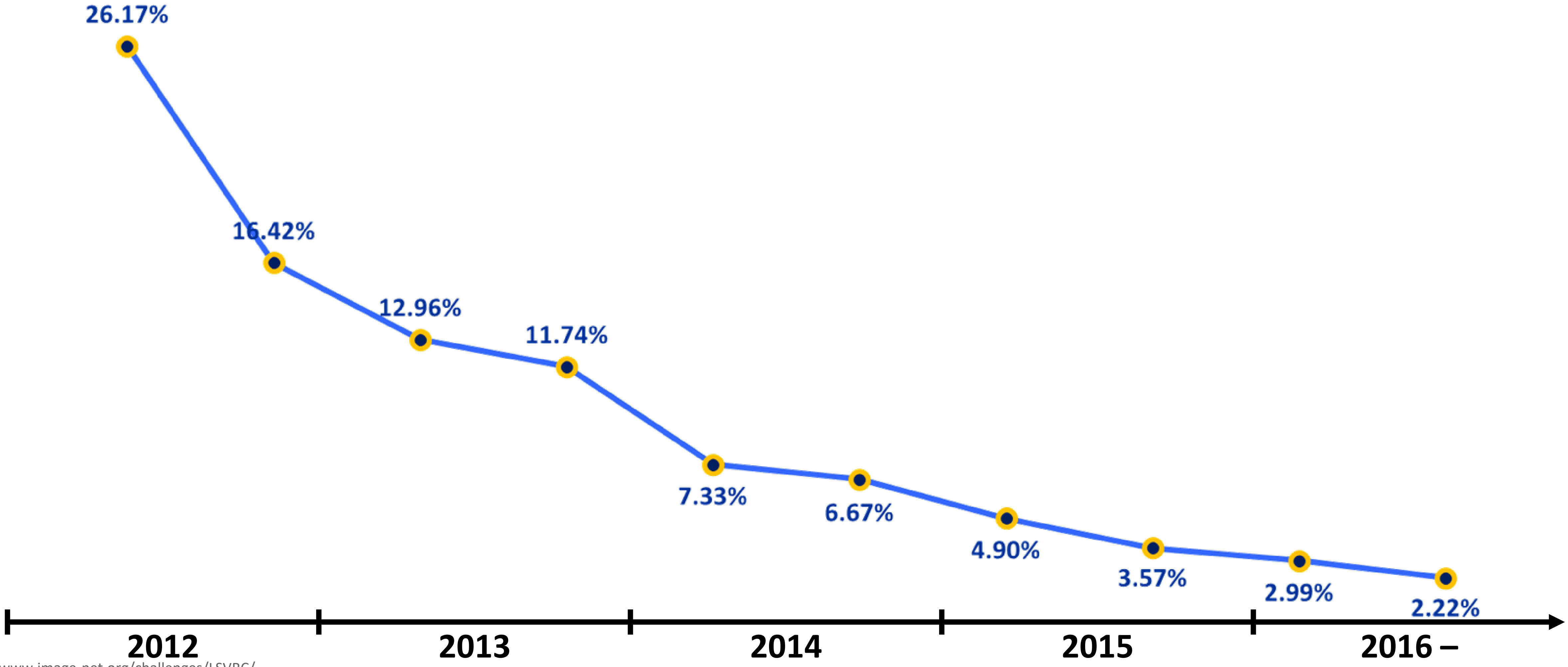
# Topics

- Convolutional Neural Networks (CNNs)
  - Convolution Layers
  - Pooling Layers
  - Normalization

- Evolution of CNNs

# **Evolution** of CNNs

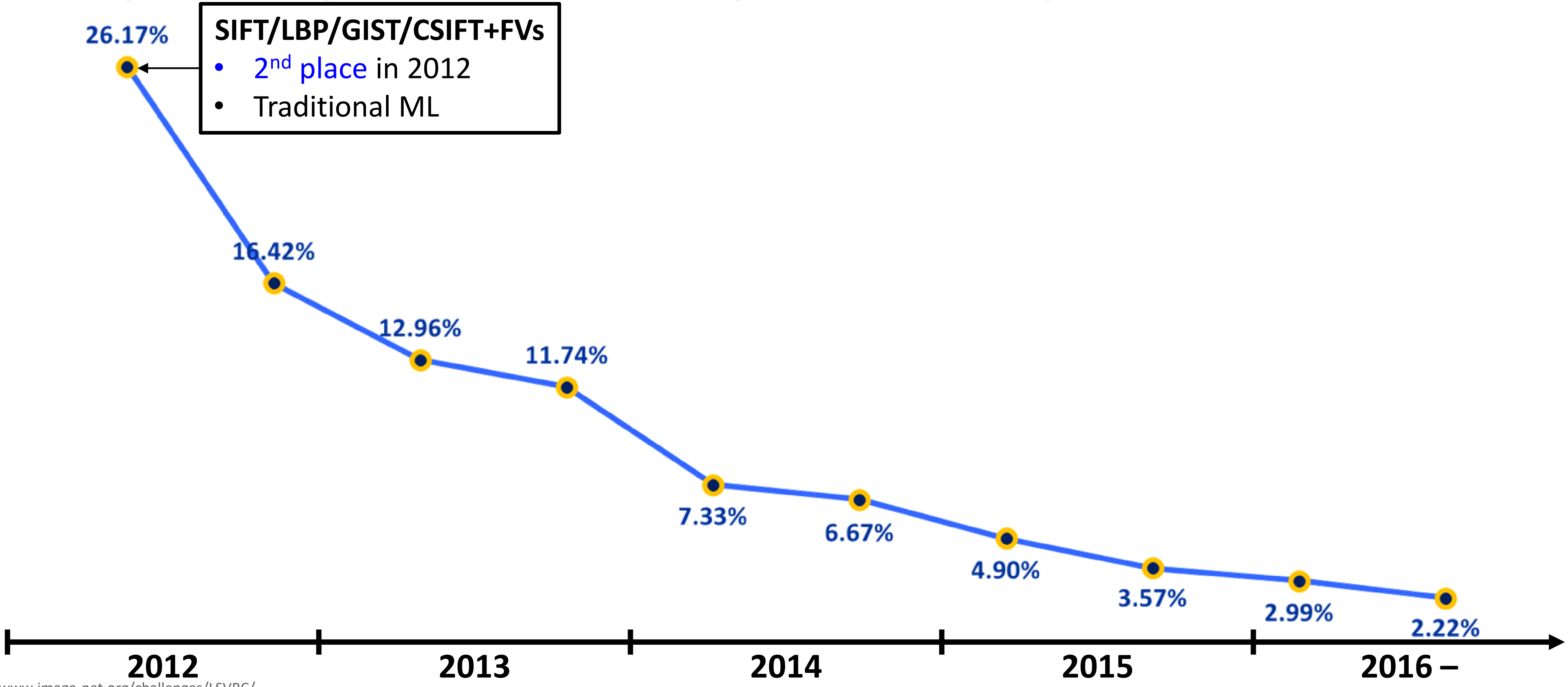
- **ImageNet Large Scale Visual Recognition Challenge (ILSVRC)**




https://www.image-net.org/challenges/LSVRC/

# **Evolution** of CNNs

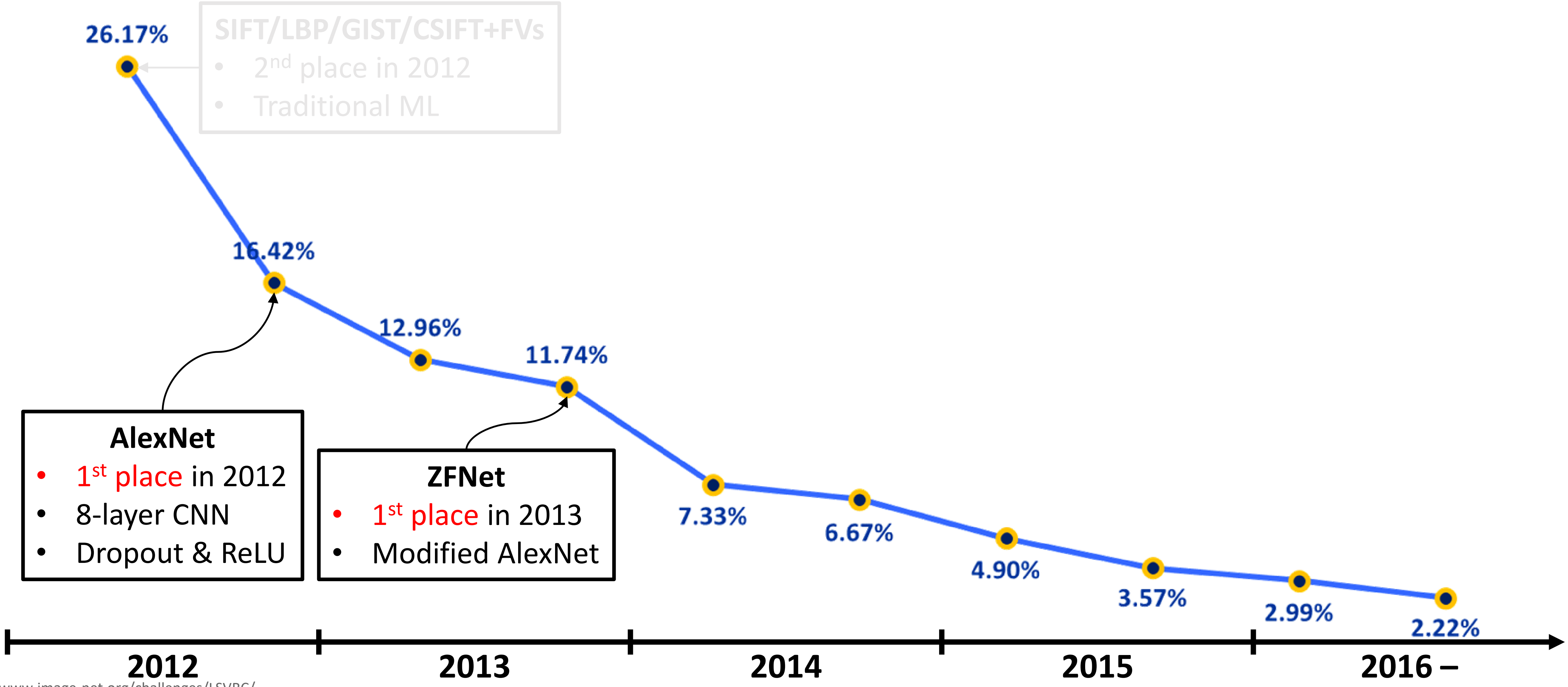
- **ImageNet Large Scale Visual Recognition Challenge (ILSVRC)**




https://www.image-net.org/challenges/LSVRC/

# **Evolution** of CNNs

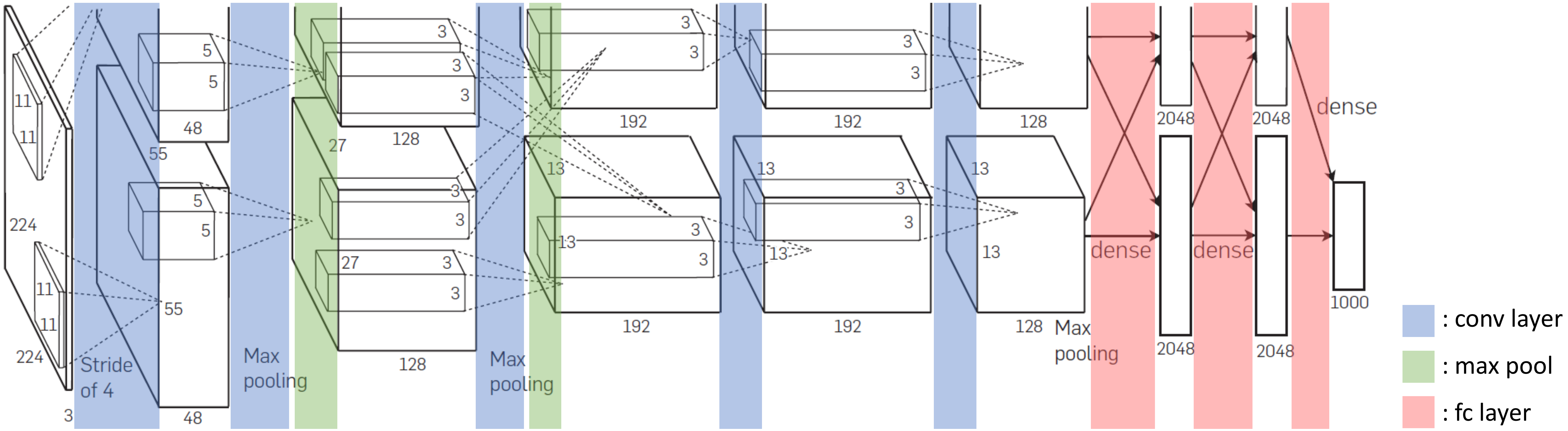
- **ImageNet Large Scale Visual Recognition Challenge (ILSVRC)**




https://www.image-net.org/challenges/LSVRC/

# Evolution of CNNs: **AlexNet (2012)**

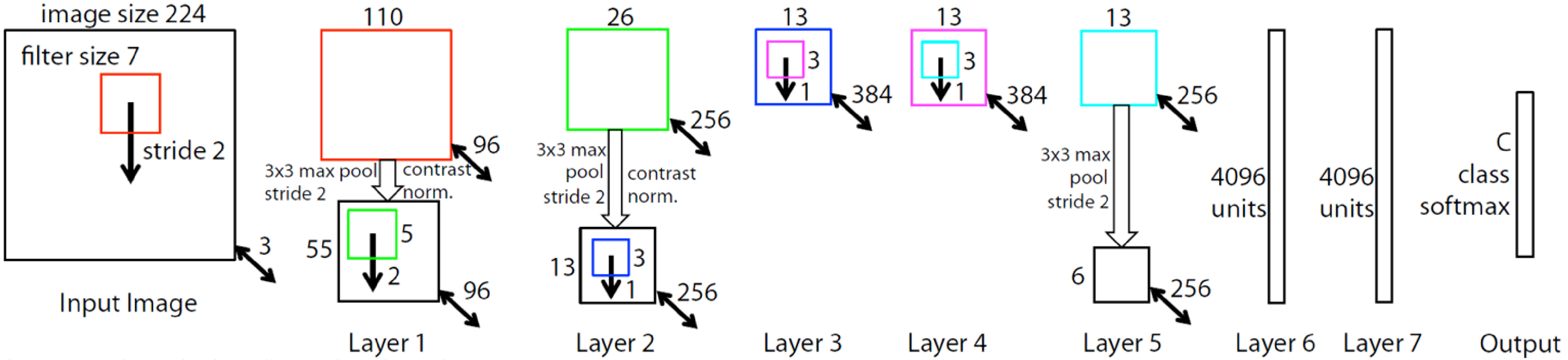
- The 1st winner to use CNN in ILSVRC 2012
  - **Astounding** improvement: 25.78% (1st place in 2011) → **15.3%** (Best in 2012)
  - The 2nd place record in 2012: 26.17%
- AlexNet
  - 8-layer CNN: 5 conv layers + 3 fc layers




A. Krizhevsky et al., ImageNet Classification with Deep Convolutional Neural Networks, **NeurIPS 2012**

# Evolution of CNNs: **ZFNet (2014)**

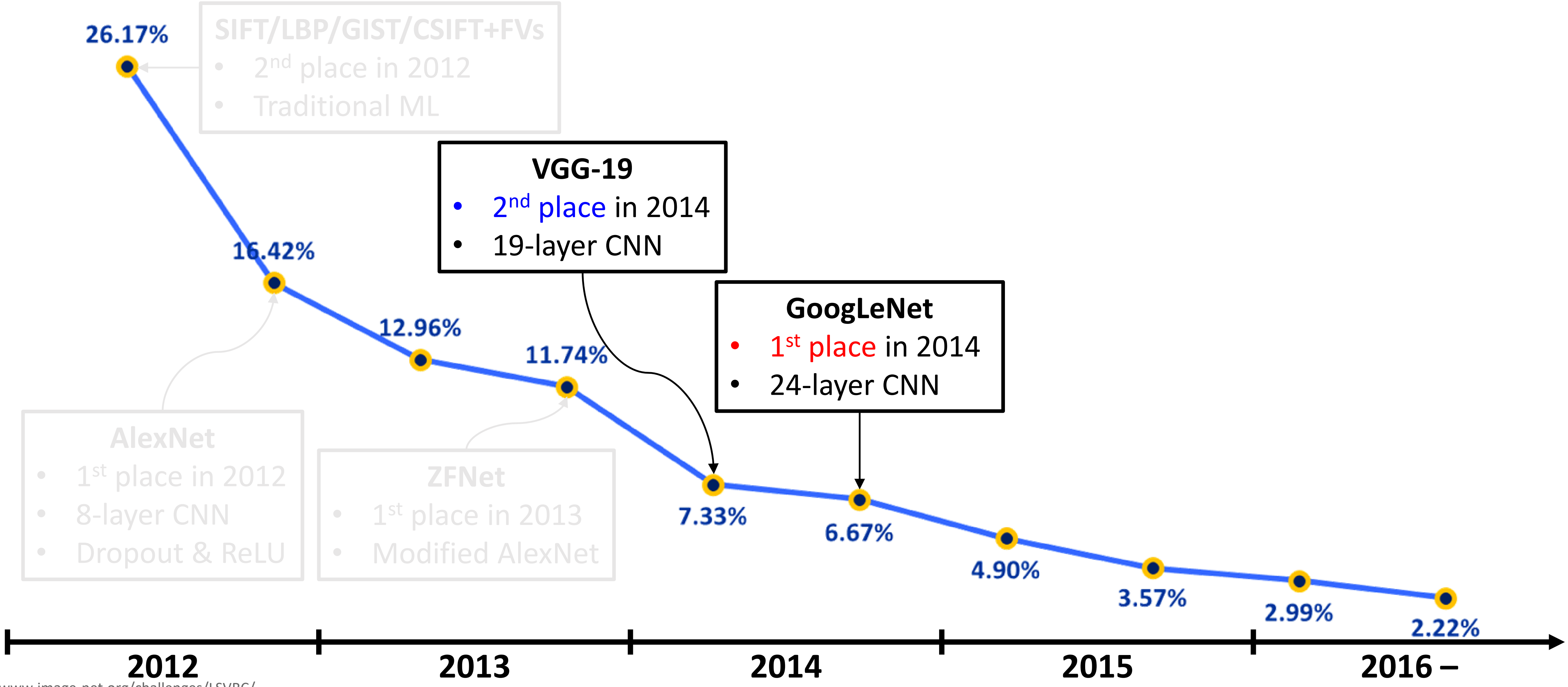
- The 1st place in ILSVRC 2013
  - Small improvement: 15.3% (1st place in 2012) → **11.7%** (Best in 2013)
- ZFNet is a *simple variant* of AlexNet
  - Smaller kernel: 11 x 11 → 7 x 7
  - Smaller stride: 4 → 2
  - Doubled the number of kernels




M. D. Zeiler and R. Fergus, Visualizing and Understanding Convolutional Networks, **ECCV 2014**

# **Evolution** of CNNs

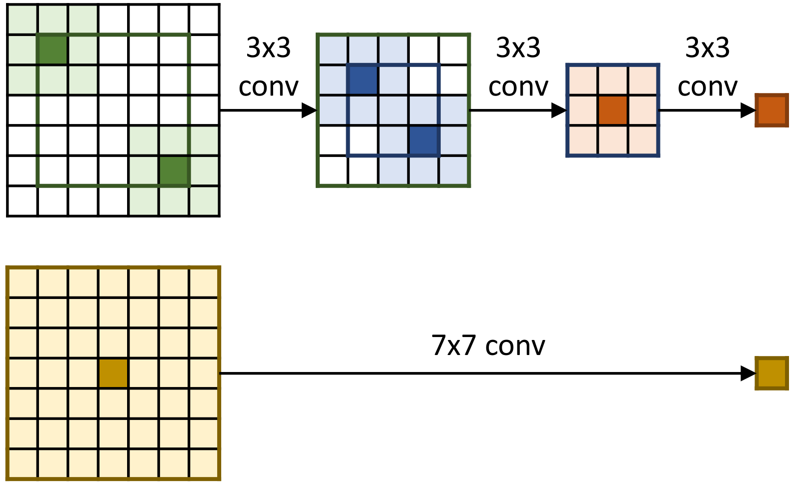
- **ImageNet Large Scale Visual Recognition Challenge (ILSVRC)**

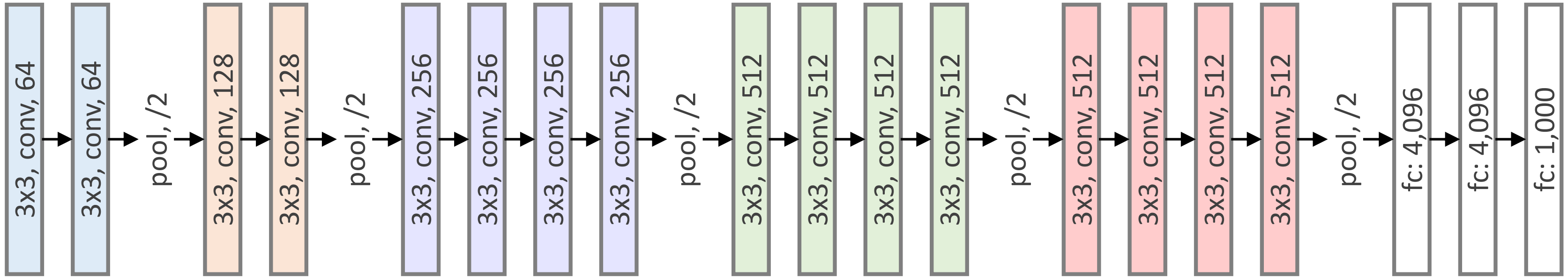



https://www.image-net.org/challenges/LSVRC/

# Evolution of CNNs: **VGG-19 (2014)**

- The 2nd place in ILSVRC 2014
  - Big improvement: 11.7% (1st place in 2013) → **7.33%** (2nd place in 2014)
- VGG-19
  - Only 3 x 3 kernels for convolutions
  - **Stacking multiple 3 x 3 kernels** is better than others
    - The same receptive field: 3 x (3 x 3) = 7 x 7
    - Small number of parameters: $27C^2 < 49C^2$
    - More non-linearity

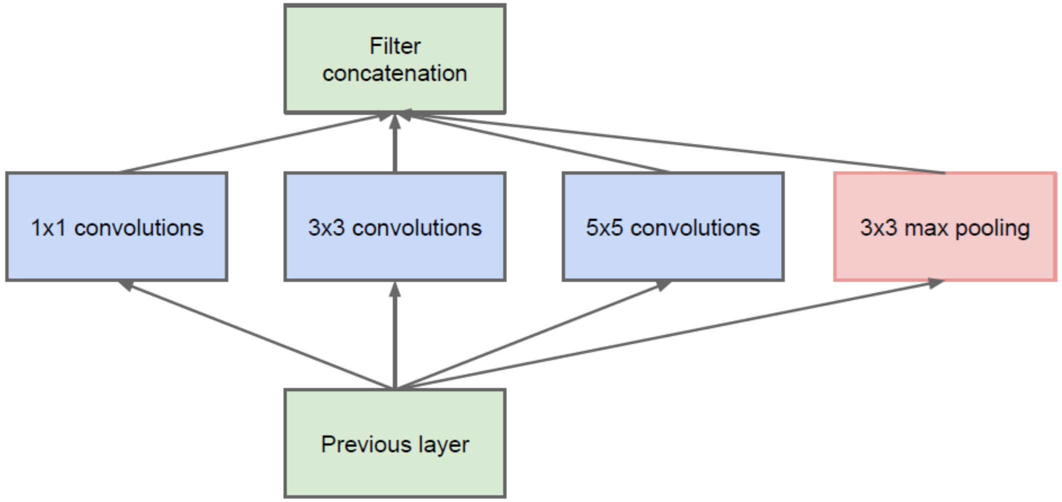


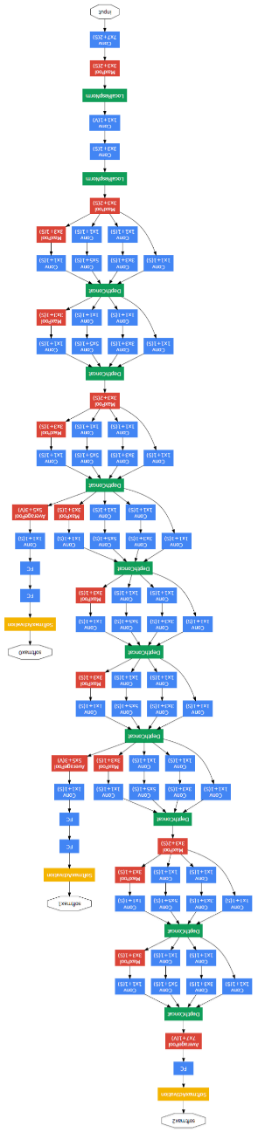

K. Simonyan and A. Zisserman, Very Deep Convolutional Networks for Large-Scale Image Recognition, **ICLR 2015**

# Evolution of CNNs: **GoogLeNet (2014)**

- The 1st place in ILSVRC 2014
  - Big improvement: 11.7% (1st place in 2013) → **6.67%** (Best in 2014)
  - With 12x fewer parameters than AlexNet
- **Inception Module** in GoogLeNet
  - Multiple convolution operations with different receptive fields
  - Capturing sparse patterns in a stack of features



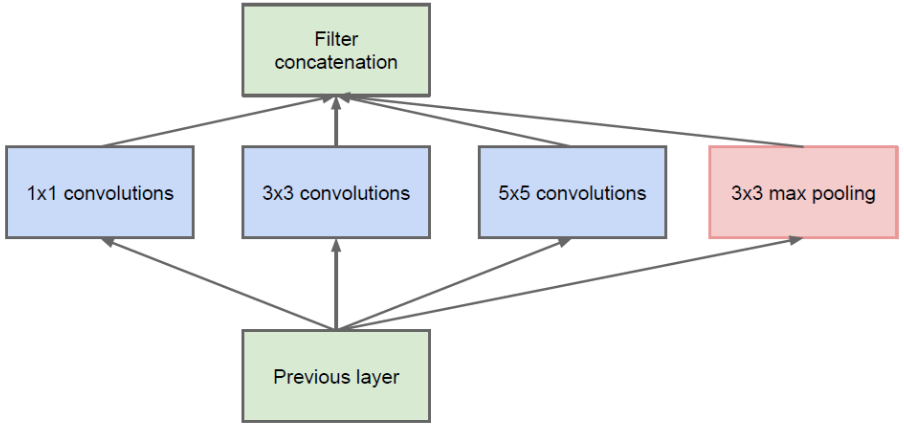

Inception module (naïve version)



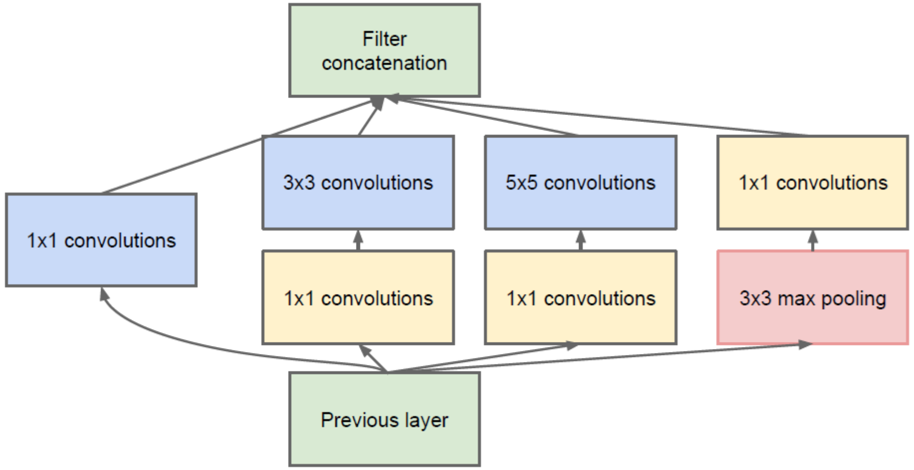
C. Szegedy et al., Going Deeper with Convolutions, **CVPR 2015**

# Evolution of CNNs: **GoogLeNet (2014)**

- Inception Module: **Naïve vs Dimension Reduction**




Inception module (naïve version)




Inception module with dimension reductions

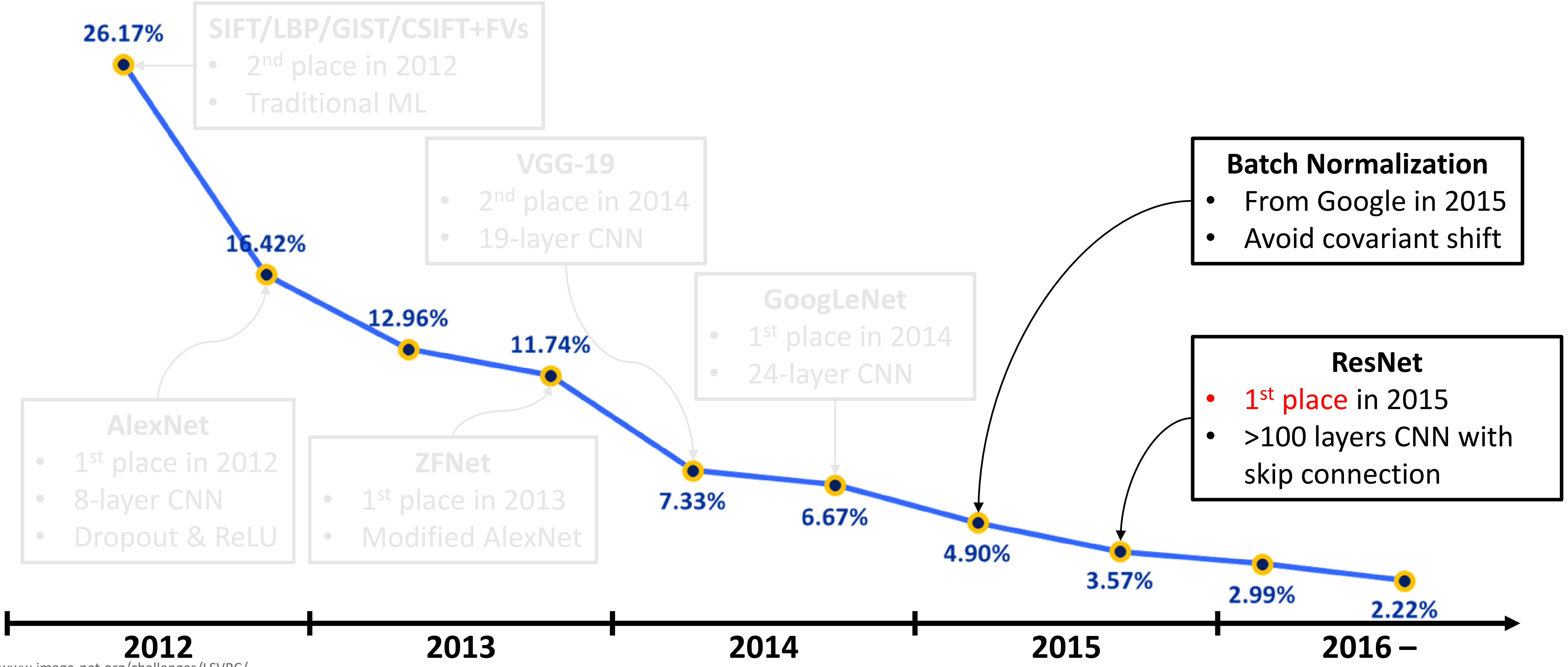
- (+) Encoding different receptive fields
- (+) Capturing sparse patterns
- (–) Computational blow up

- (+) Reductions before the expensive convolutions
- (+) Useful for changing #.channels
- (+) More non-linearities

C. Szegedy et al., Going Deeper with Convolutions, **CVPR 2015**

# **Evolution** of CNNs

- **ImageNet Large Scale Visual Recognition Challenge (ILSVRC)**



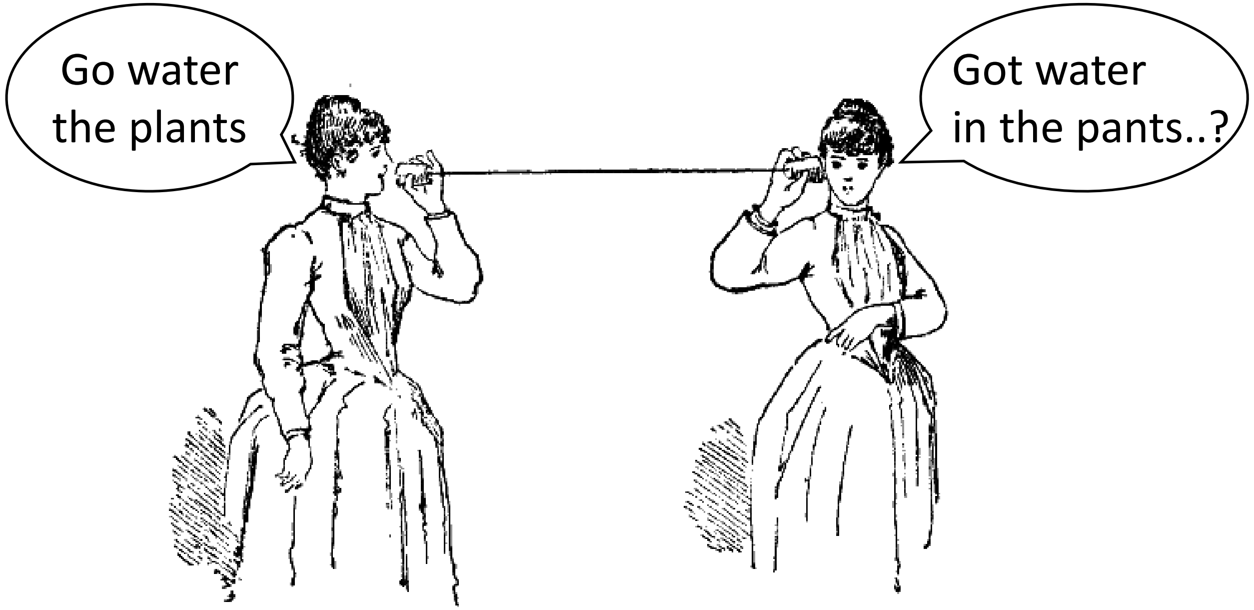

https://www.image-net.org/challenges/LSVRC/

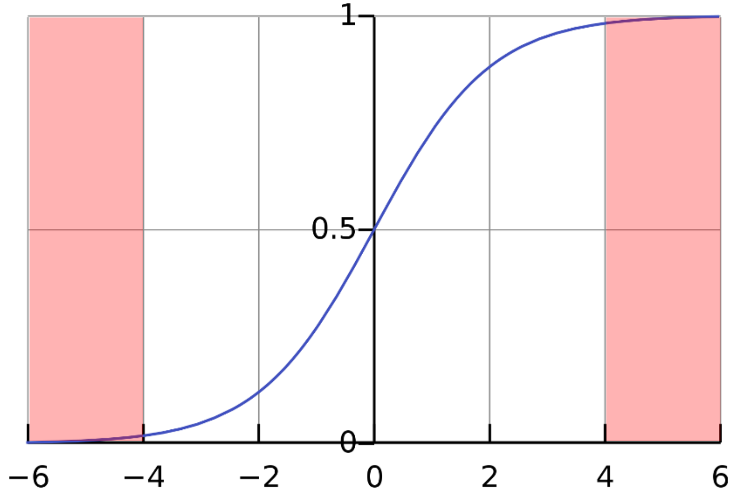
# Evolution of CNNs: **Batch Normalization (2015)**

- **Internal Covariate Shift**
  - The change in the distribution of network activations due to the change in network parameters during training

  → It requires lower learning rates and careful initialization

  → It makes it notoriously hard to train models with saturating nonlinearities



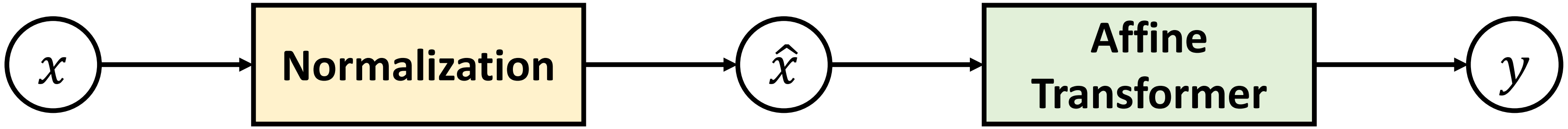




Saturated regime in activation

S. Ioffe and C. Szegedy, Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift, **ICML 2015**

# Evolution of CNNs: **Batch Normalization (2015)**

- **Batch normalization (BN)** dramatically accelerates the training of deep networks by reducing internal covariate shift
- Normalization via Mini-Batch Statistics
  - Normalization: Normalize each dimension for a layer
  - Transformation: Scale and shift the normalized value



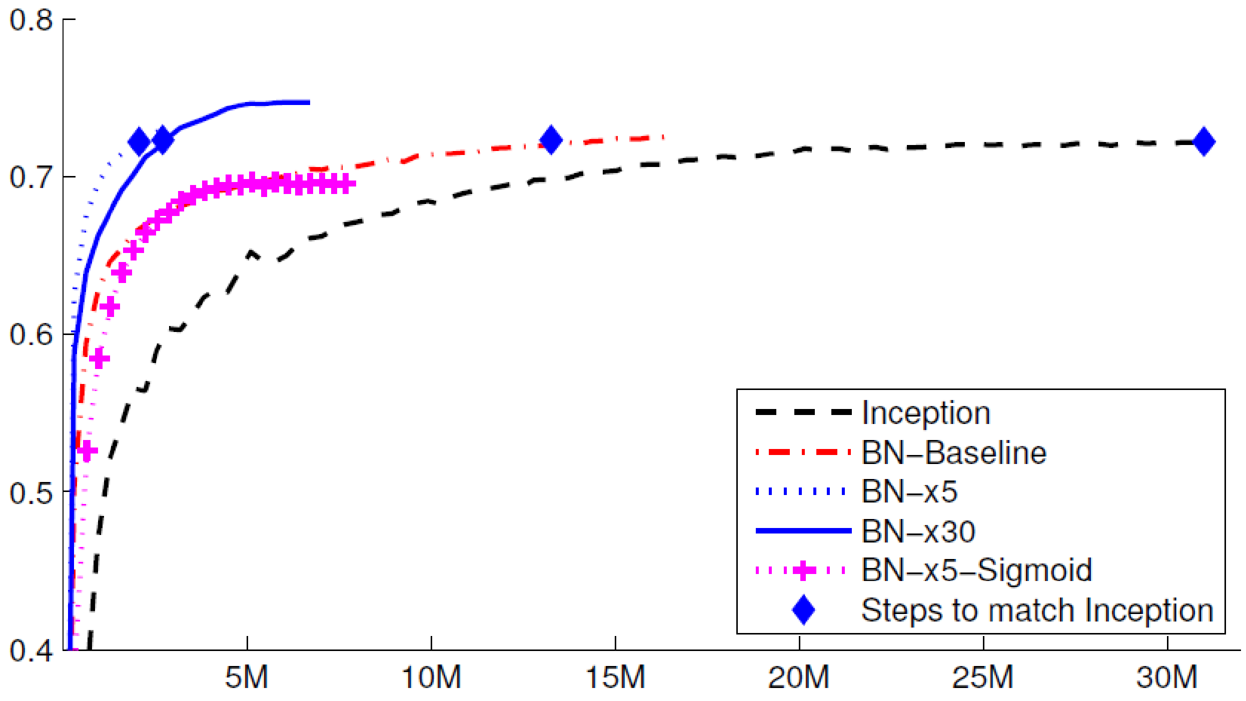

$$\hat{x}_i = \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \varepsilon}}$$

$$y_i = \gamma \hat{x}_i + \beta$$

S. Ioffe and C. Szegedy, Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift, **ICML 2015**

# Evolution of CNNs: **Batch Normalization (2015)**

- **Batch normalization (BN)** dramatically accelerates the training of deep networks by reducing internal covariate shift
  - Much higher learning rates
  - Less careful about initialization




S. Ioffe and C. Szegedy, Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift, **ICML 2015**

# Evolution of CNNs: **Batch Normalization (2015)**

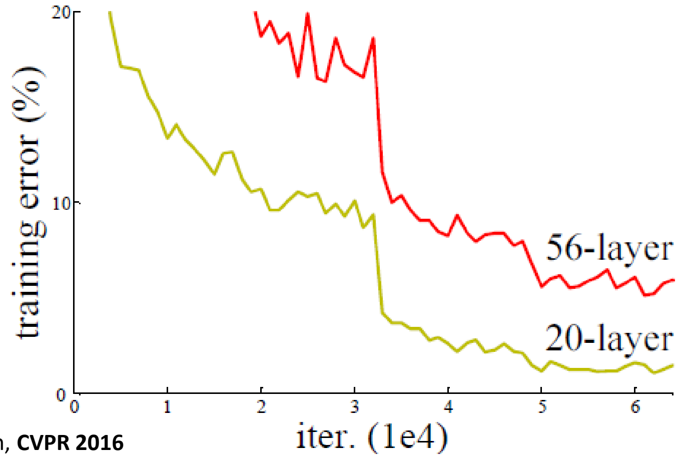
- **Batch normalization (BN)** dramatically accelerates the training of deep networks by reducing internal covariate shift
  - Allows much **higher learning rates** → Faster training
  - **Stabilizes** gradient vanishing → Saturating non-linearities
  - Has **regularization** effects → No need dropout layers

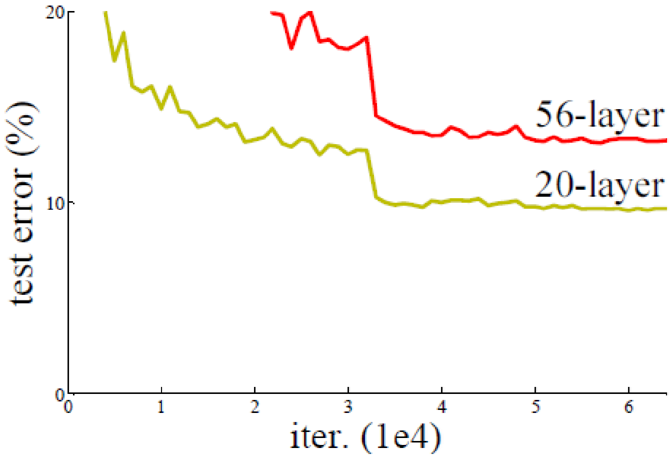
| Model | Resolution | Crops | Models | Top-1 error | Top-5 error |
|---|---|---|---|---|---|
| GoogLeNet ensemble | 224 | 144 | 7 | - | 6.67% |
| Deep Image low-res | 256 | - | 1 | - | 7.96% |
| Deep Image high-res | 512 | - | 1 | 24.88 | 7.42% |
| Deep Image ensemble | up to 512 | - | - | - | 5.98% |
| MSRA multicrop | up to 480 | - | - | - | 5.71% |
| MSRA ensemble | up to 480 | - | - | - | 4.94%* |
| BN-Inception single crop | 224 | 1 | 1 | 25.2% | 7.82% |
| BN-Inception multicrop | 224 | 144 | 1 | 21.99% | 5.82% |
| BN-Inception ensemble | 224 | 144 | 6 | 20.1% | **4.82%*** |

S. Ioffe and C. Szegedy, Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift, **ICML 2015**

# Evolution of CNNs: **ResNet (2016)**

- The 1st place in ILSVRC 2015
  - **Astounding** improvement: 6.67% (1st place in 2015) → **3.57%** (Best in 2015)
  - ResNet is the first architecture succeeded to train > 100-layer
- **Degradation** in Training Deeper Networks
  - Adding more layers leads to higher training error
  - Such degradation is not caused by overfitting
  - Deeper networks are much harder to optimize even if we use BNs

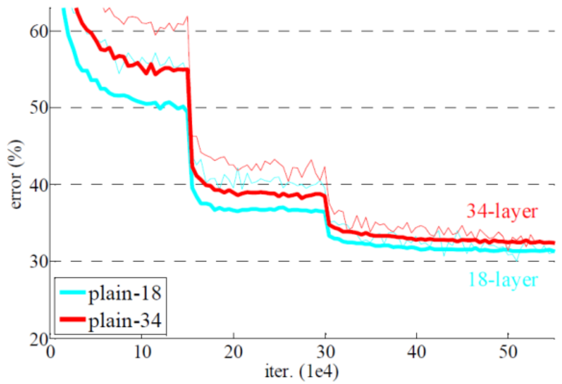


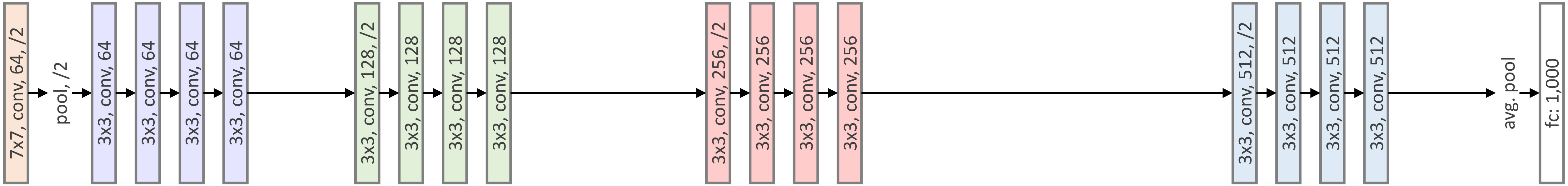




K. He et al., Deep Residual Learning for Image Recognition, **CVPR 2016**

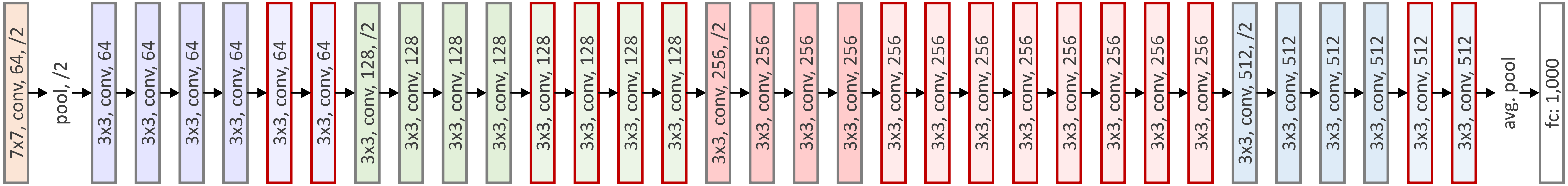
# Evolution of CNNs: **ResNet (2016)**

- **Non-linear layers** may struggle to represent an identity func
  - Due to internal non-linearities such as ReLU
  - This may cause the optimization difficulty on deeper networks




Shallow network: plain-18

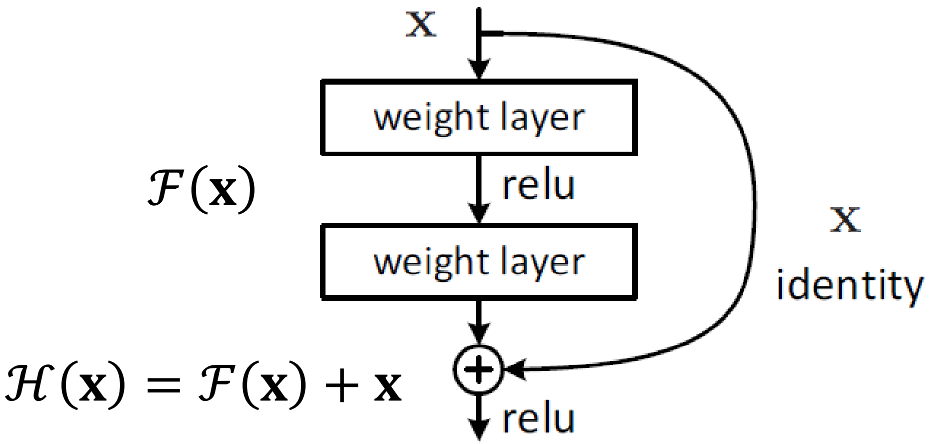


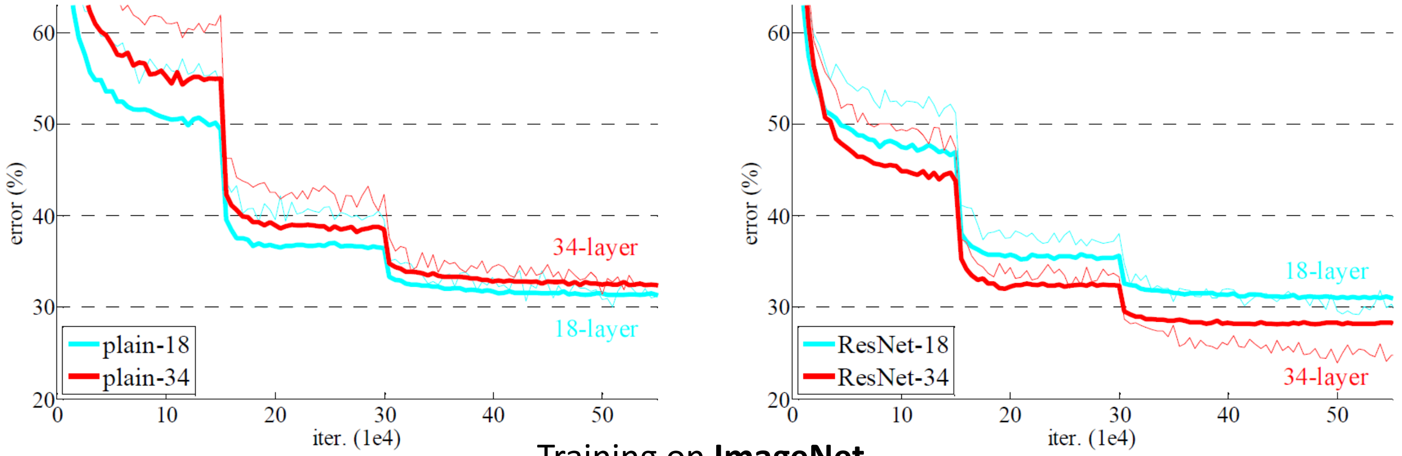

K. He et al., Deep Residual Learning for Image Recognition, **CVPR 2016**

Deeper network: plain-34

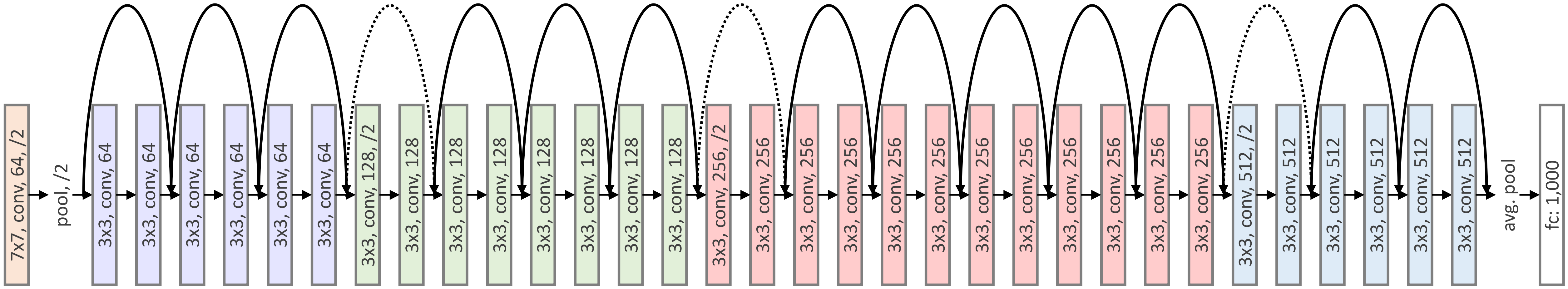
# Evolution of CNNs: **ResNet (2016)**

- **Solution:** Explicitly let the stacked layers fit a residual mapping




Training on **ImageNet**




Deeper network: ResNet-34

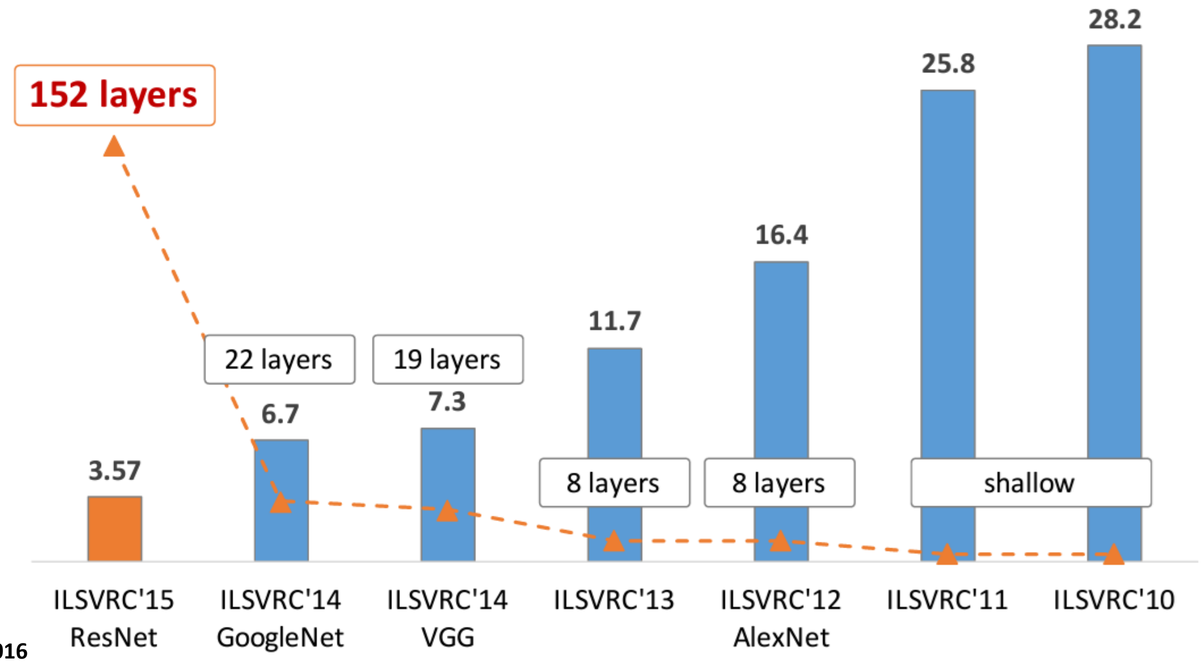
K. He et al., Deep Residual Learning for Image Recognition, **CVPR 2016**

# Evolution of CNNs: **ResNet (2016)**

- **Revolution of Depth**

— Identity connection resolved a major difficulty on optimizing large networks
  - Residual learning makes it possible to train >100-layer without optimization difficulty

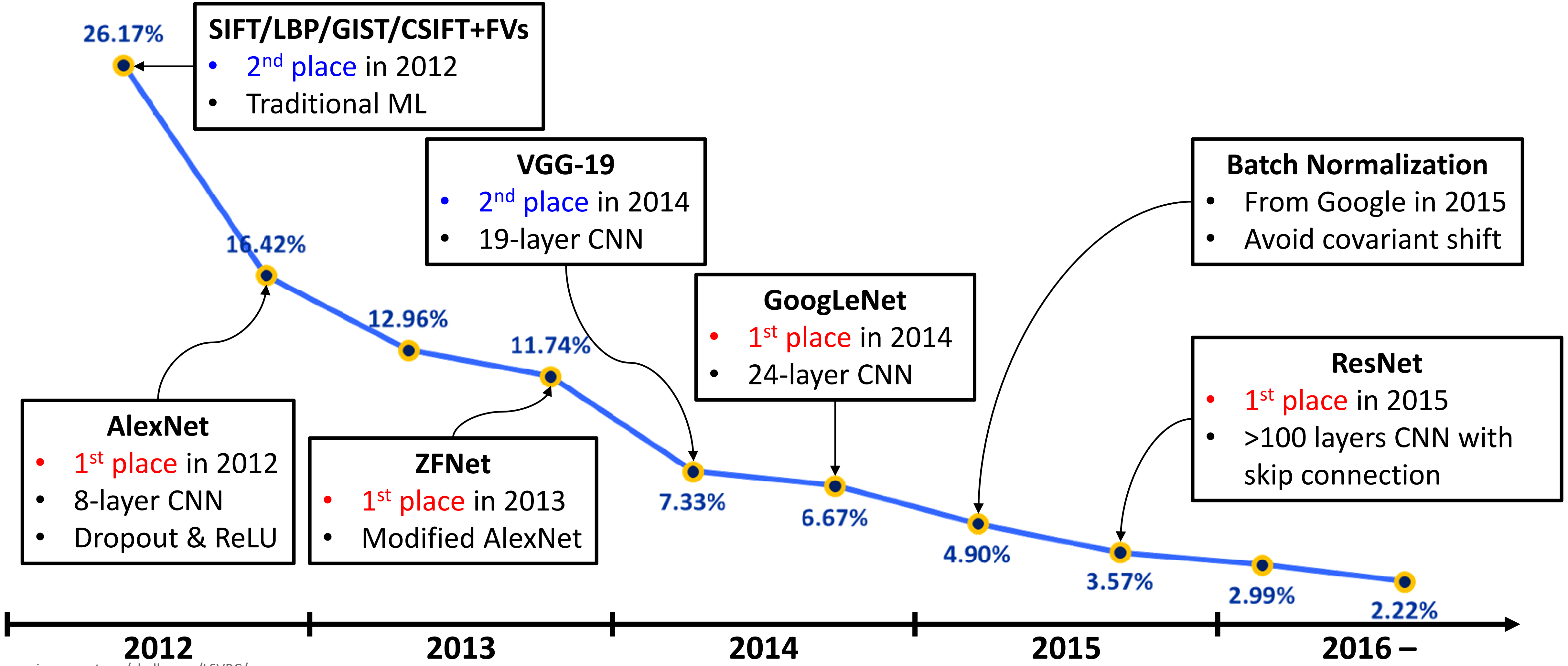
— ResNet shows good generalization ability as well




K. He et al., Deep Residual Learning for Image Recognition, **CVPR 2016**
K. He, Deep Residual Networks: Deep Learning Gets Way Deeper, **ICML 2016 Tutorial**

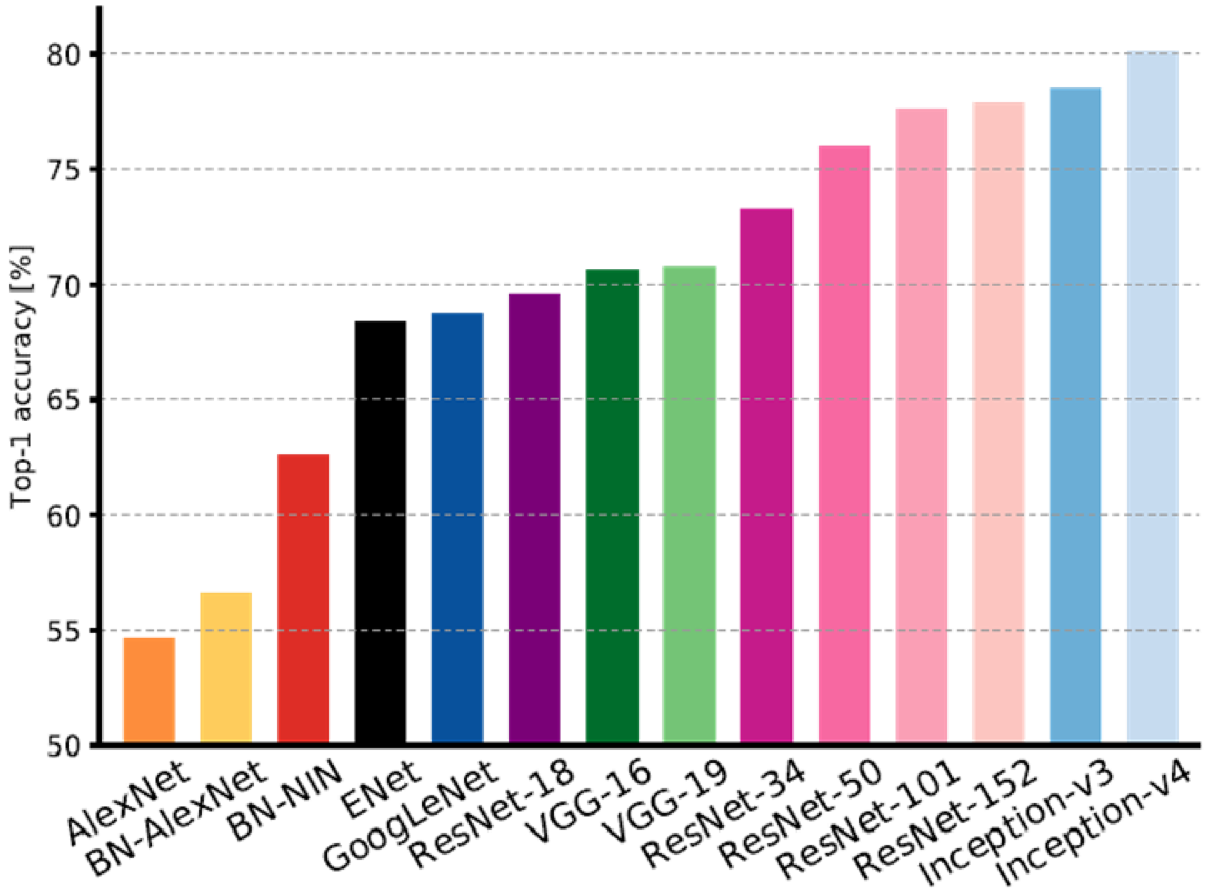
# **Evolution** of CNNs

- **ImageNet Large Scale Visual Recognition Challenge (ILSVRC)**

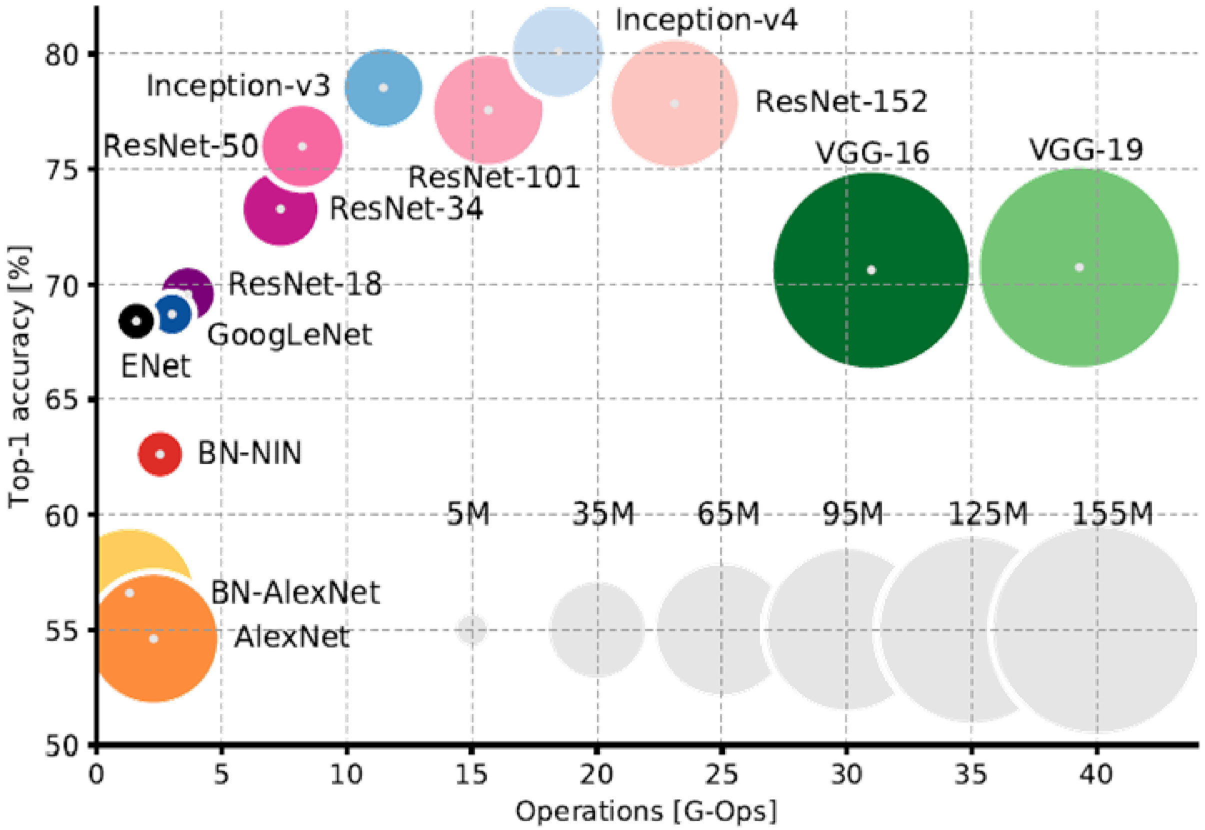



https://www.image-net.org/challenges/LSVRC/

# **Evolution** of CNNs: Performance Comparisons




**Top1 vs. network**
Single-crop top-1 validation accuracies
for top scoring single-model architectures




**Top1 vs. operations, size / parameters.**
Top-1 one-crop accuracy versus amount
of operations required for a single forward pass

A. Canziani et al., An Analysis of Deep Neural Network Models for Practical Applications, **arXiv 2016**

# Evolution of CNNs: **Various** ResNet Architectures

- **Network Design Paradigm**: Optimization → Generalization

— How well does an architecture generalize as its scale grows?

- Various Architectures based on ResNet
  - Deep networks with stochastic depth
  - Wide ResNet
  - ResNet in ResNet
  - ResNeXt
  - Inception-v4
  - DenseNet
  - Dual Path Network
  - *Etc.*


G. Huang et al., Deep Networks with Stochastic Depth, **ECCV 2016**
S. Zagoruyko and N. Komodakis, Wide Residual Networks, **BMVC 2017**
S. Targ et al., ResNet in ResNet: Generalizing Residual Architectures, **ICLRW 2016**
S. Xie et al., Aggregated Residual Transformations for Deep Neural Networks, **CVPR 2017**
C. Szegedy et al., Inception-v4, Inception-ResNet and the Impact of Residual Connections on Learning, **AAAI 2017**
G. Huang et al., Densely Connected Convolutional Networks, **CVPR2017**
Y. Chen et al., Dual Path Networks, **NeurIPS 2017**