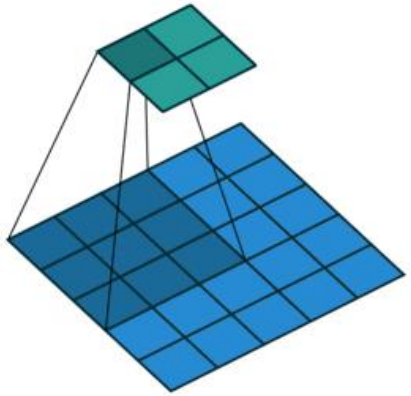
上海交通大学
SHANGHAI JIAO TONG UNIVERSITY

CNNs and Image Recognition

(2026)

Lecturer: Xue Yang

yangxue.site



上海交通大学
SHANGHAI JIAO TONG UNIVERSITY

Contents

- Convolutional Neural Network
- Training CNNs
- Classical CNN Architectures

Image Classification with NN

- Image Classification is a core task in Computer Vision.

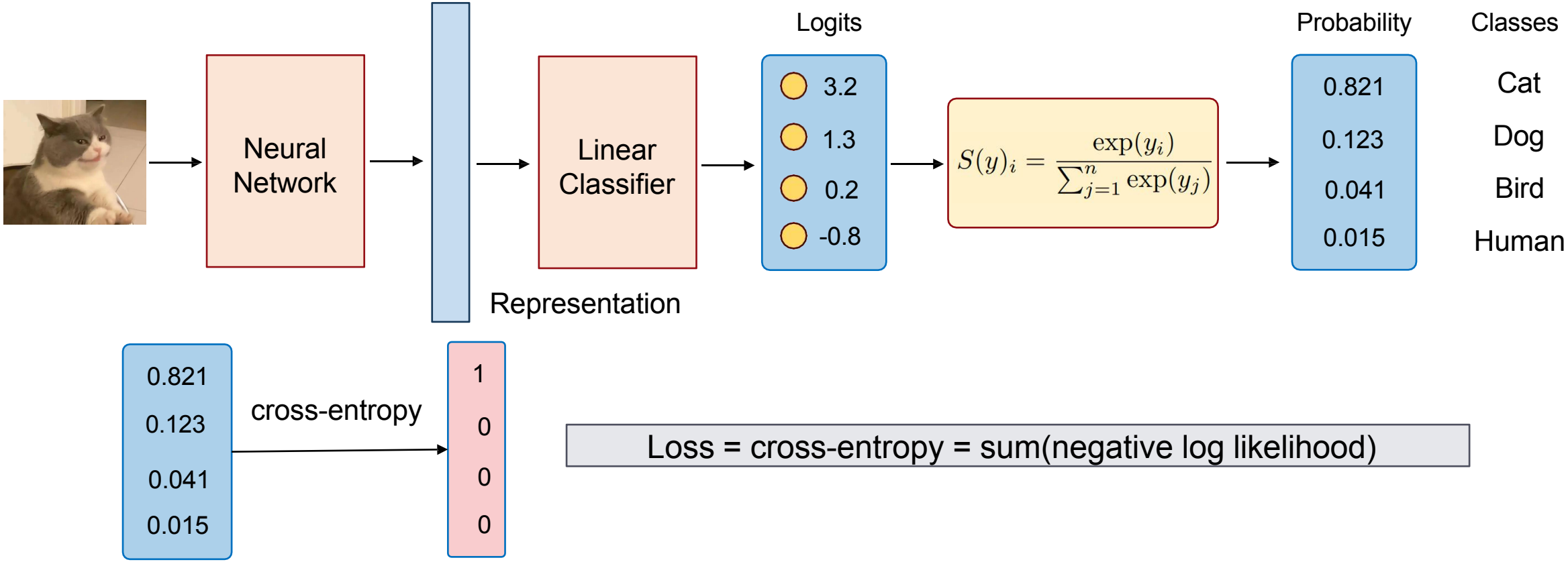
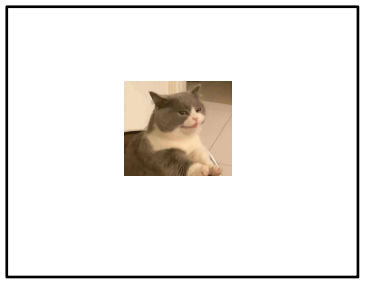
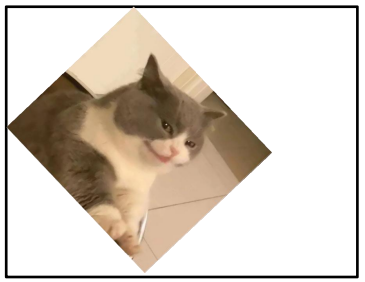


Image Classification with NN

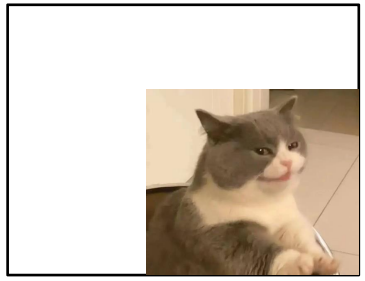
- Images after scale/rotation/translation should be classified into the same category!



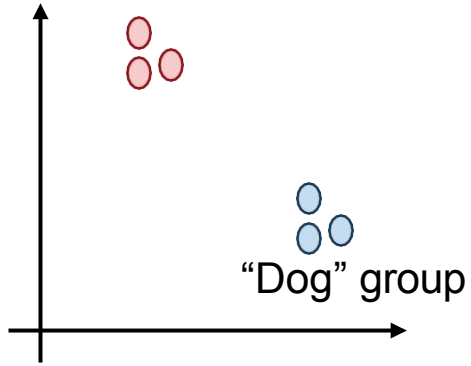
Scale Invariance



Rotation Invariance



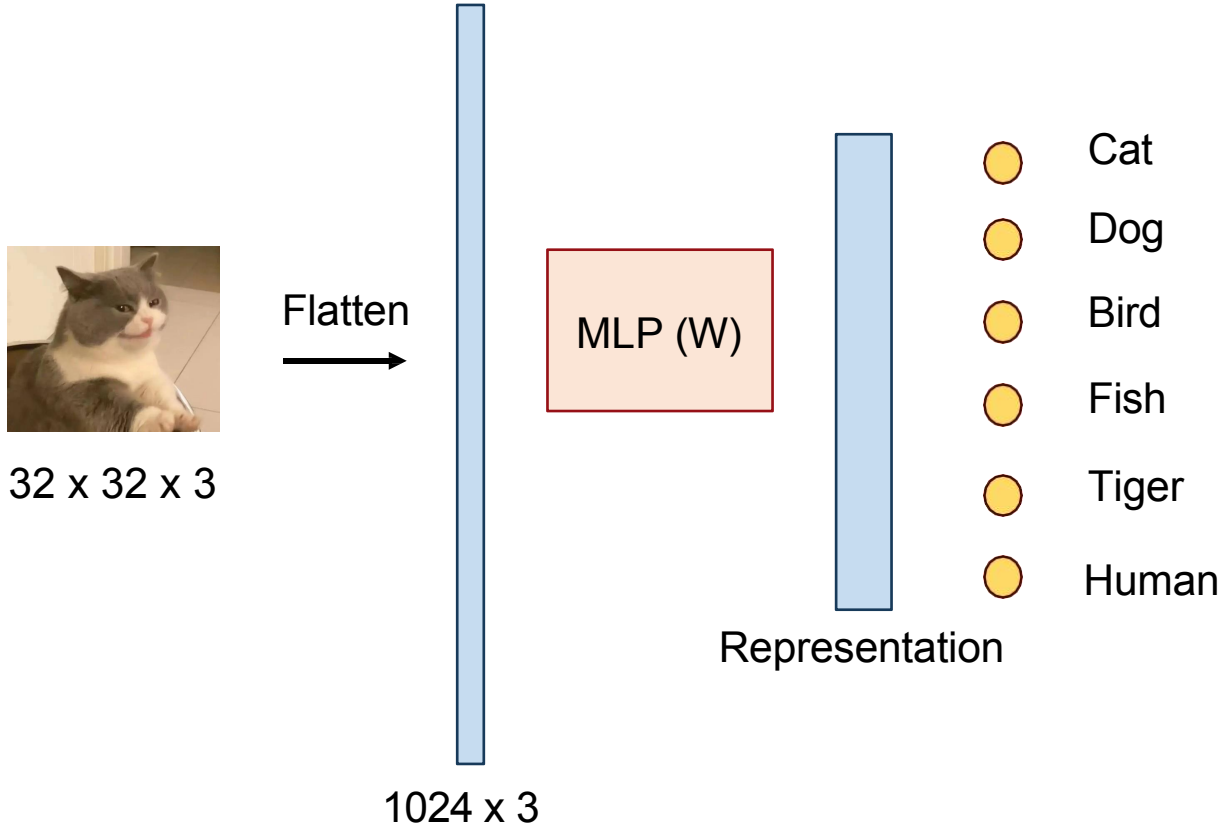
Translation Invariance



Transformation Invariance !

Image Classification with NN

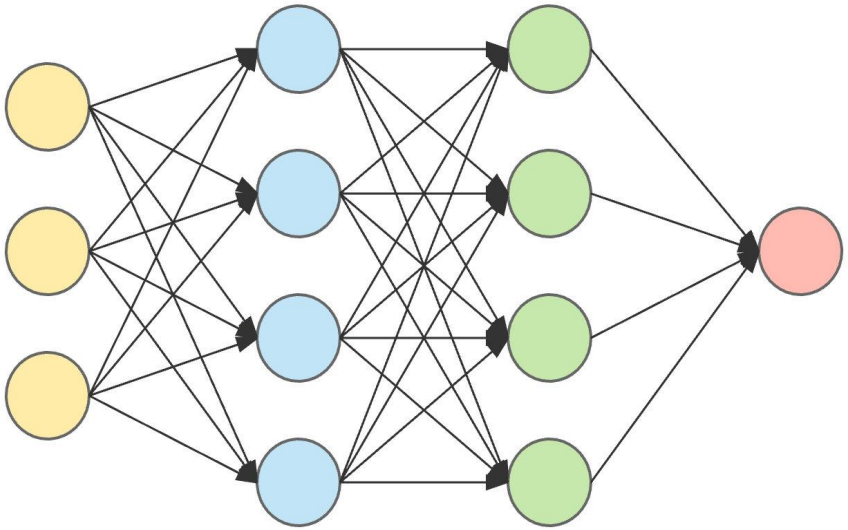
- What if we use MLP?



The number of parameters is unmanageable !

Image Classification with NN

- What if we use MLP?



- ✓ **Input image**: 32 * 32 pixel, RGB
- ✓ **Task**： 10-class classification
- ✓ **Networks**：

  Input Layer $32 \times 32 \times 3$ = 3,072 neurons

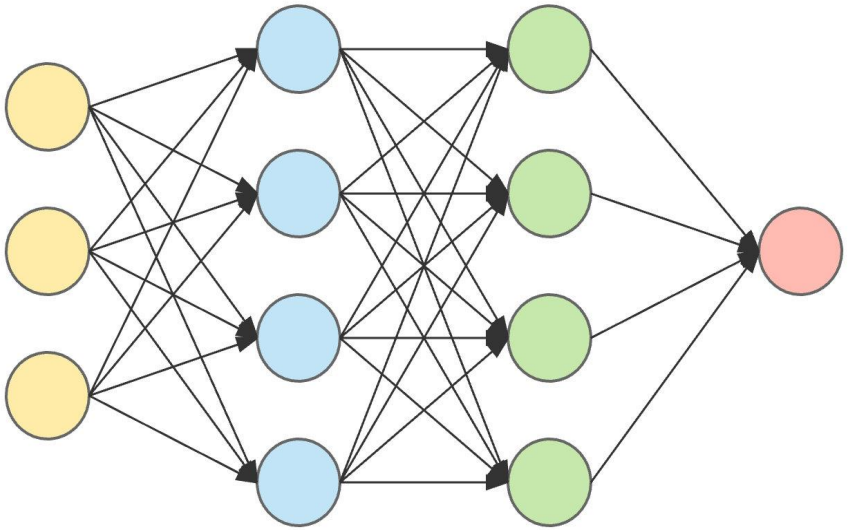
  Hidden Layer1 4,096 neurons

  Hidden Layer2 2,048 neurons

  Output Layer 10 neurons

Image Classification with NN

- What if we use MLP?



$(3{,}072+1) \times 4{,}096$

$(4{,}096+1) \times 2{,}048$

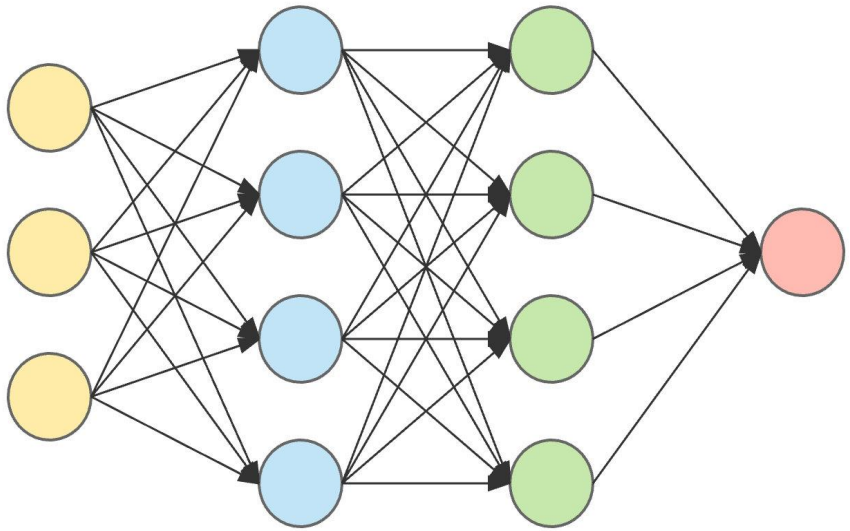
$(2{,}048+1) \times 10$

} ≈21 million

A shallow fully connected network has a massive number of parameters

Image Classification with NN

- What if we use MLP?



- ✓ In practical engineering applications, the number of network layers can reach **hundreds**, and the pixel count of input images is also much **larger than 32 * 32**.
- ✓ In such cases, using a fully connected neural network would result in an unmanageable number of parameters.

Image Classification with NN

- Intuition

What leads to the extremely high parameter complexity in fully connected neural networks?

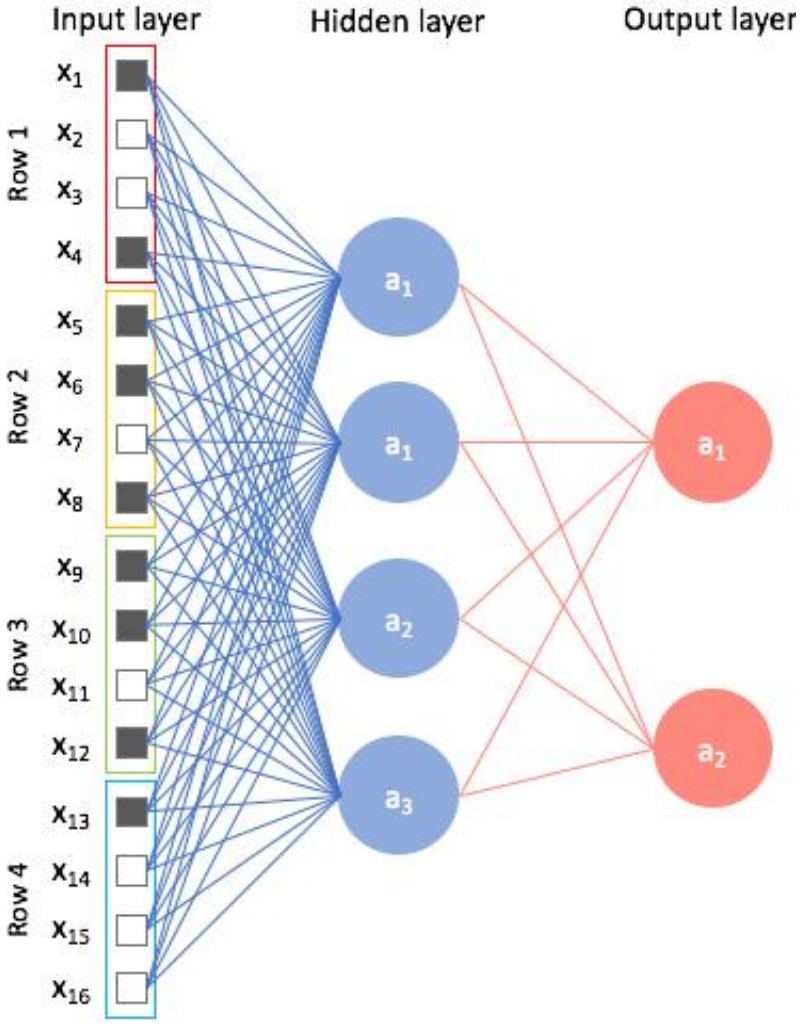
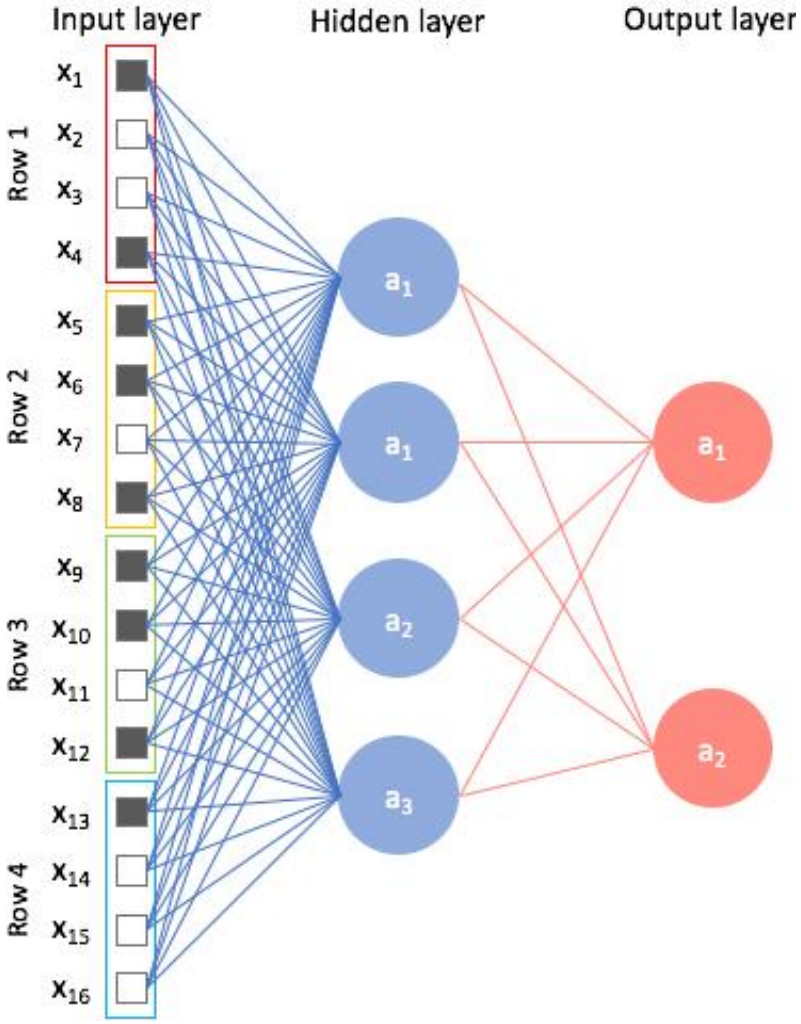


Image Classification with NN

- **Connection Method**: Every neuron in a layer connects to all neurons in the previous layer, resulting in a dense weight matrix.

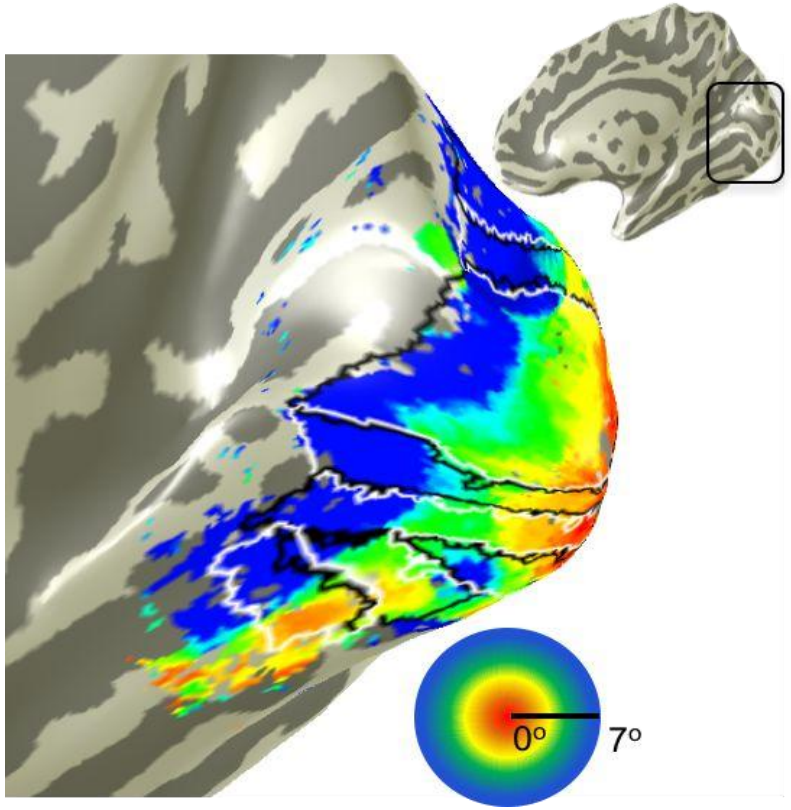
- **Absence of spatial information**: Flatten the image into a one-dimensional vector without utilizing the spatial distribution information between pixels



Case

Topographical mapping in the cortex:

- nearby cells in cortex represent
- nearby regions in the visual field



Convolution

- Given an input signal sequence x and a filter w, the output of one-dimensional convolution:

$$y_t = x * w = \sum_{k=0}^{K-1} w_k x_{t-k} = \sum_{n=t-K+1}^{t} x_n w_{t-n} = \sum_{k=t-K+1}^{t} x_k w_{t-k}$$

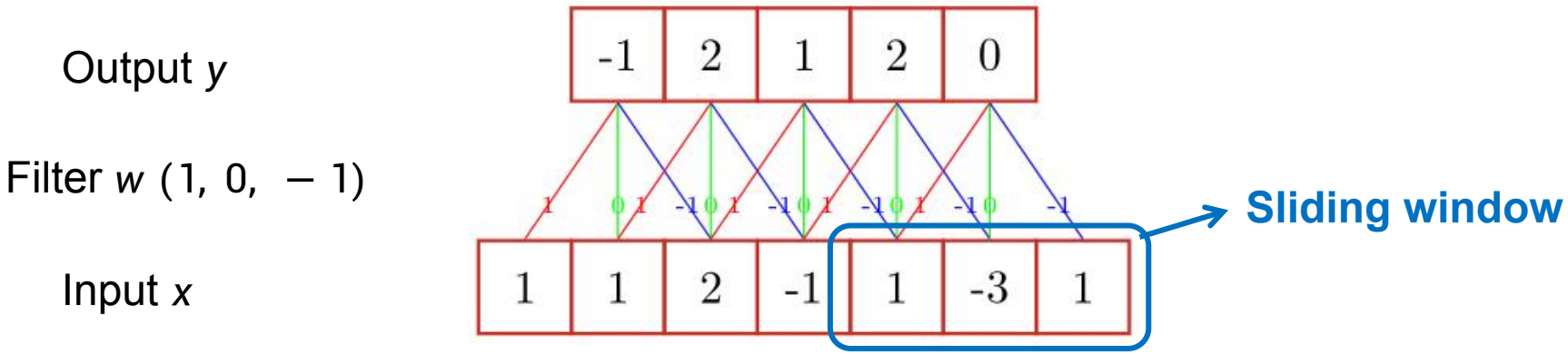
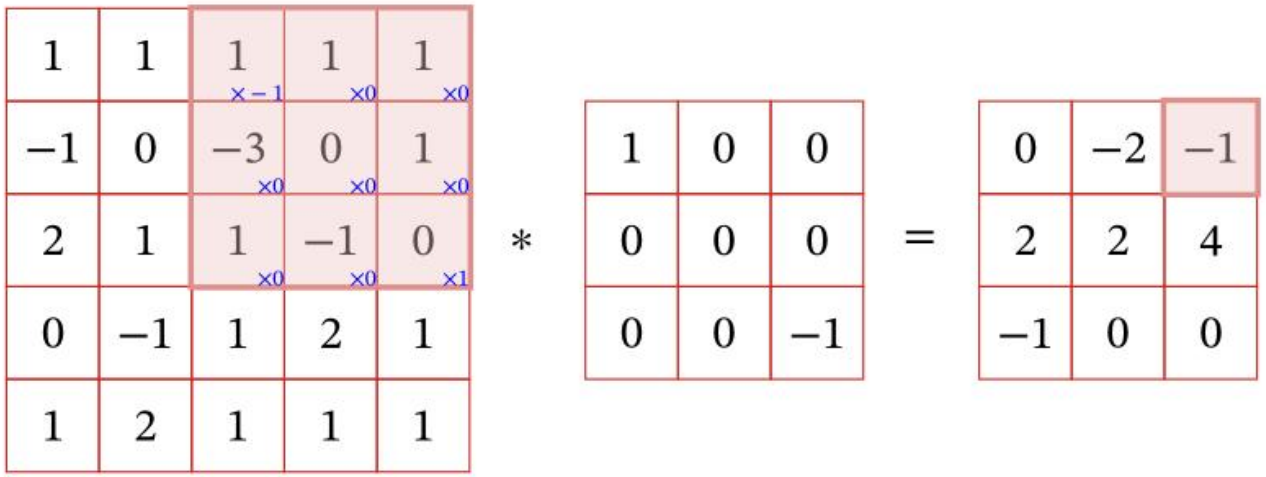


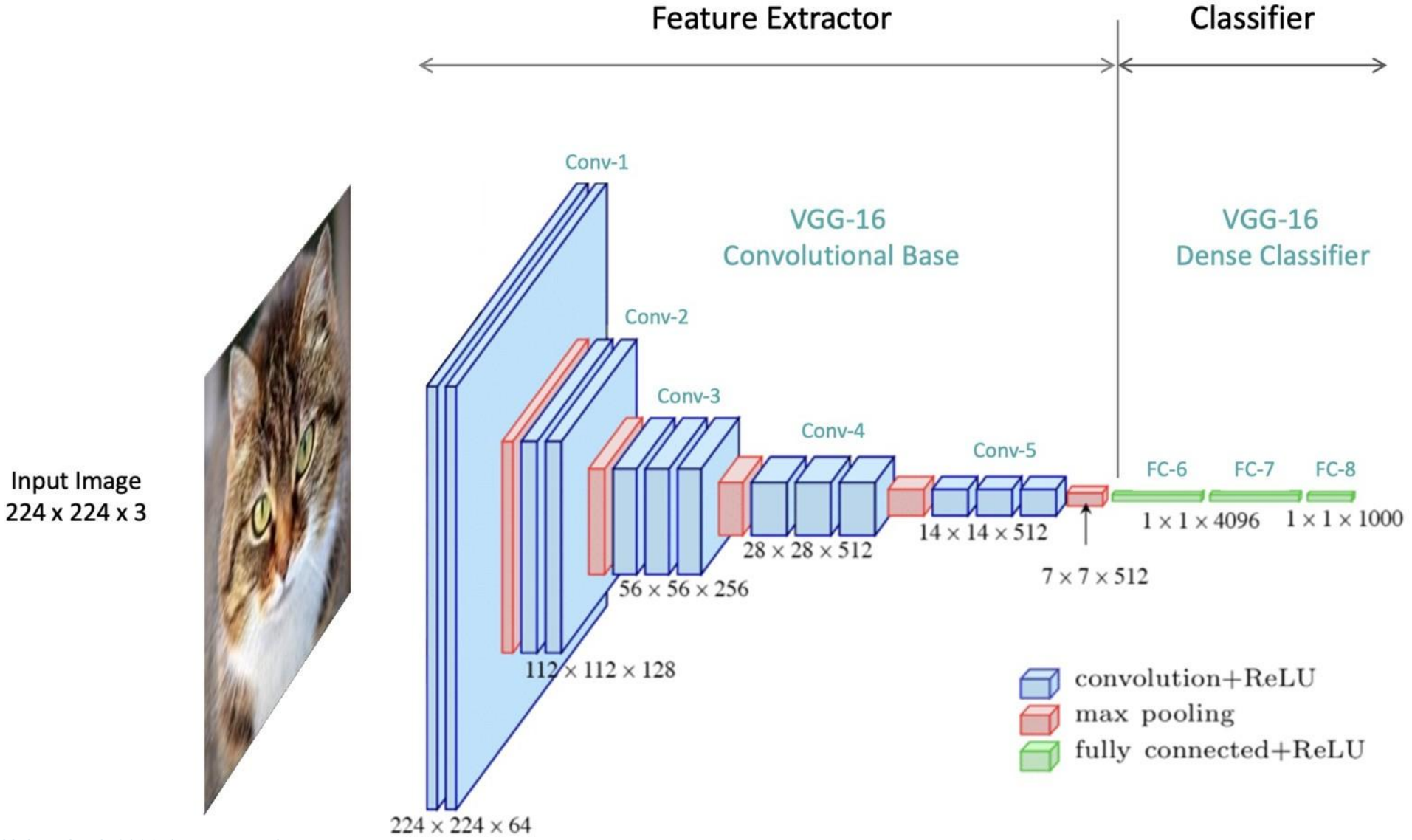
Image Convolution

- Two-dimensional image data → Extend convolution to two-dimensional

$$y_{ij} = \sum_{u=0}^{U-1} \sum_{v=0}^{V-1} w_{uv} x_{i-u,j-v}$$

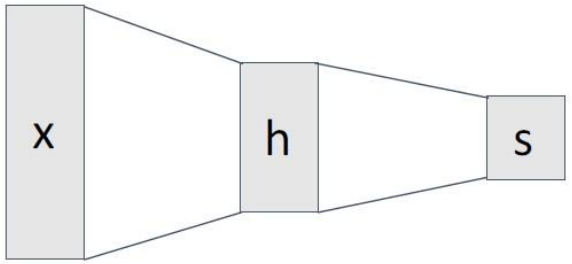


Convolutional Neural Network

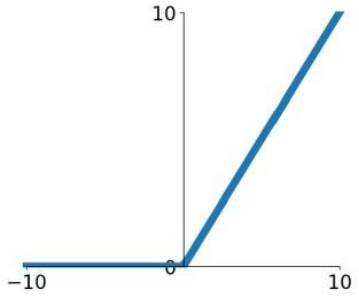


Components of a Convolutional Network

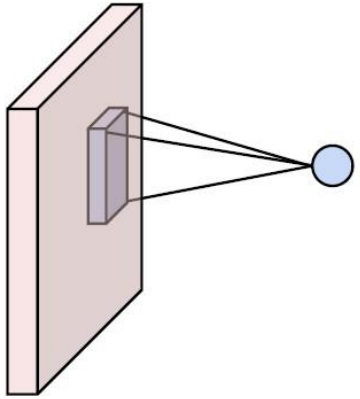
Fully-Connected Layers



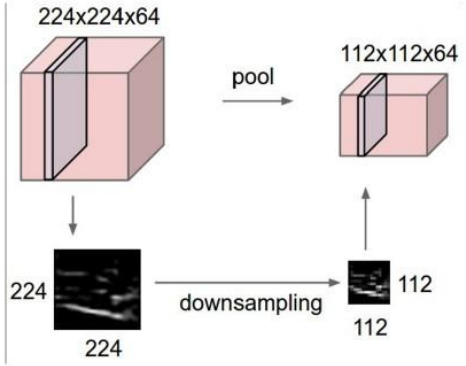
Activation Function



Convolution Layers



Pooling Layers



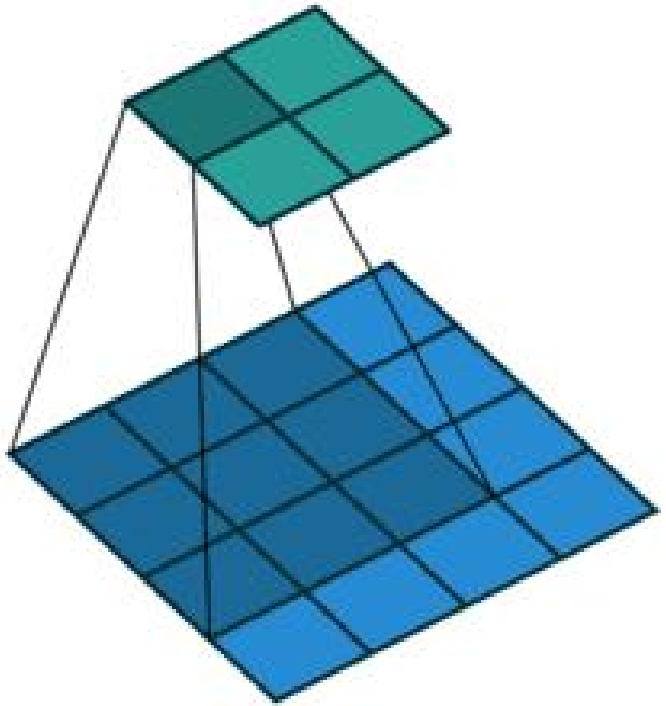
Normalization

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

Convolution Layers

Input				Kernel			Output	
0	1	2	*	0	1	=	19	25
3	4	5		2	3		37	43
6	7	8						

$$0\times0+1\times1+3\times2+4\times3=19,$$
$$1\times0+2\times1+4\times2+5\times3=25,$$
$$3\times0+4\times1+6\times2+7\times3=37,$$
$$4\times0+5\times1+7\times2+8\times3=43.$$



Convolution Kernel

(wikipedia)

$$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$$

Edge Detection

$$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$$

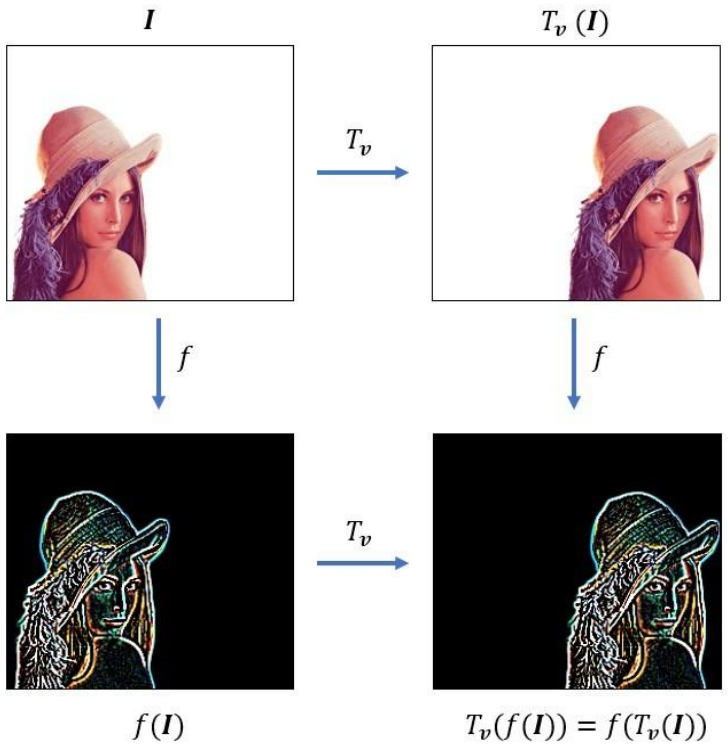
Sharpening

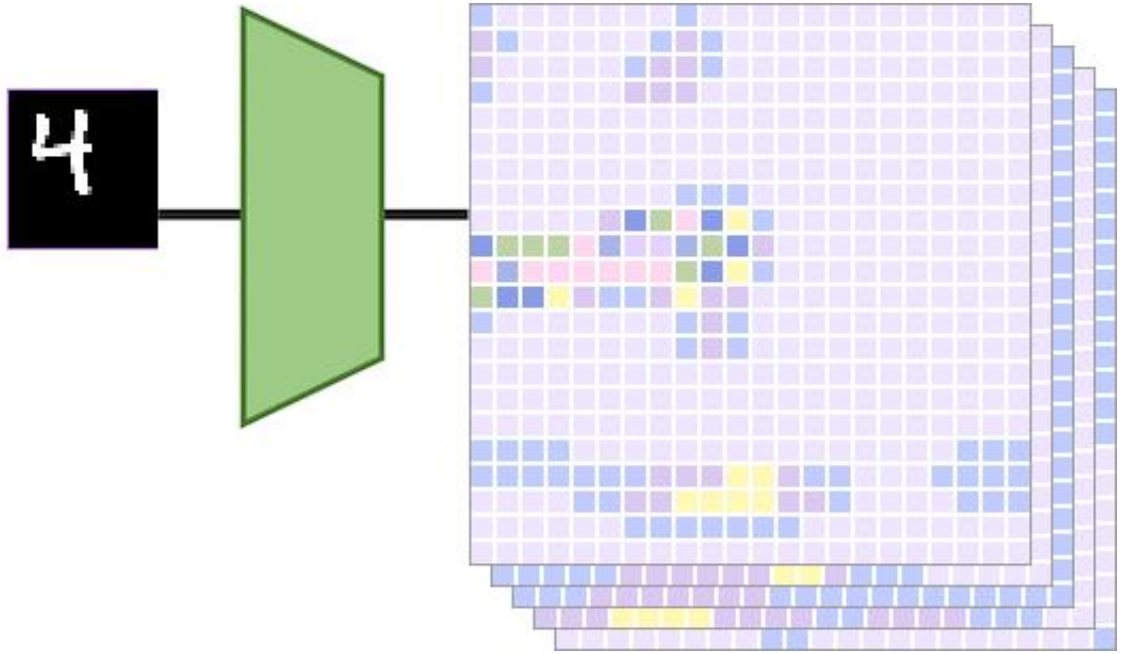
$$\frac{1}{16}\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$$

Gaussian Blur

Translation Equivariance

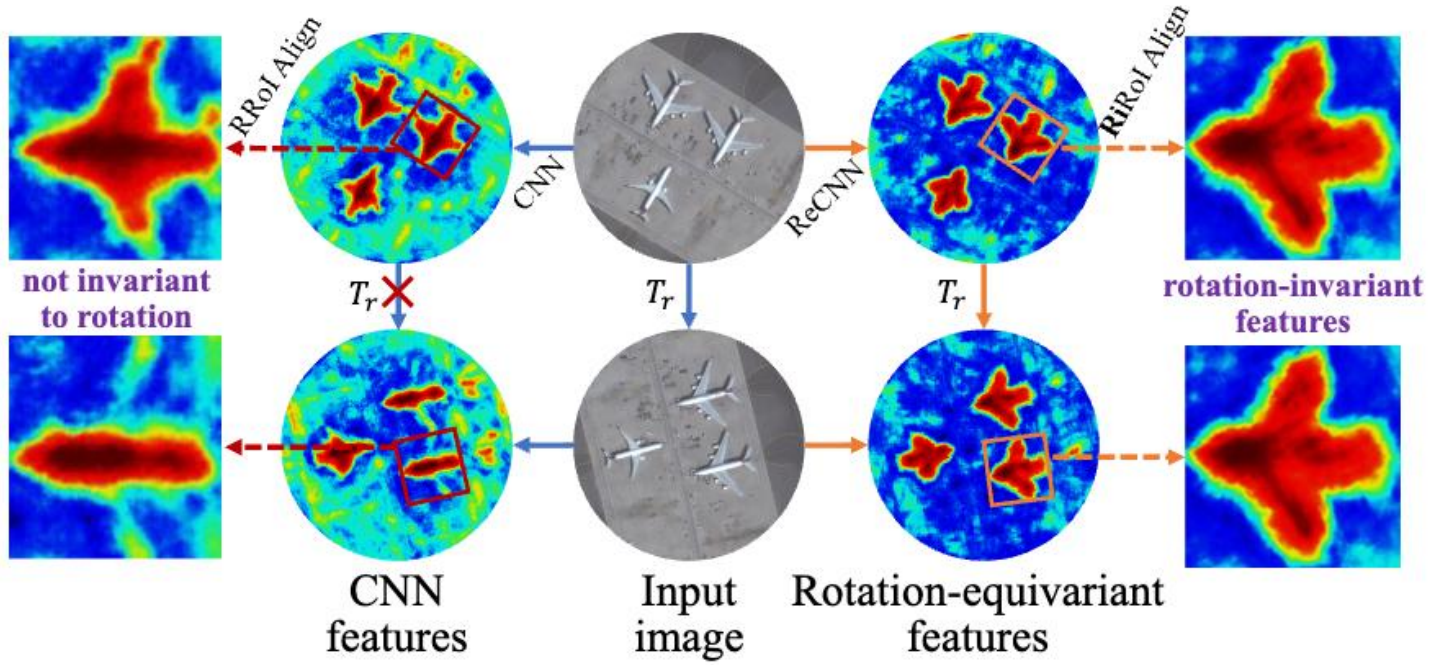
- The convolution operation is **translation equivariant**.

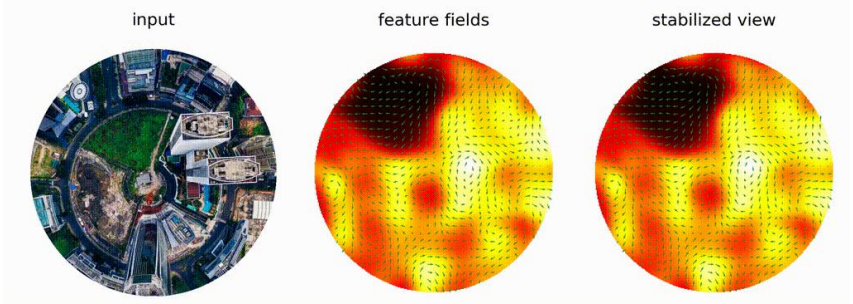


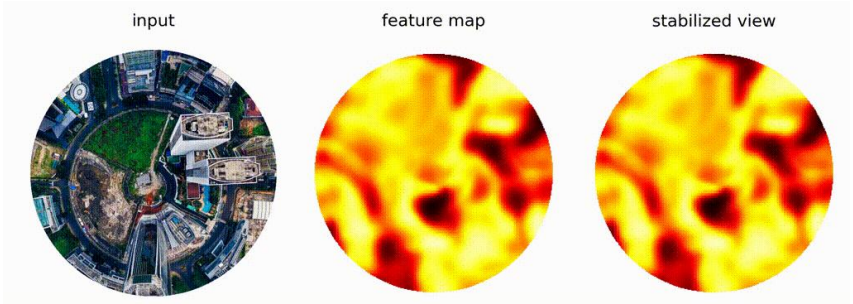


Translation Equivariance

- The convolution operation is **not rotation equivariant**.







Padding

Input

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

*

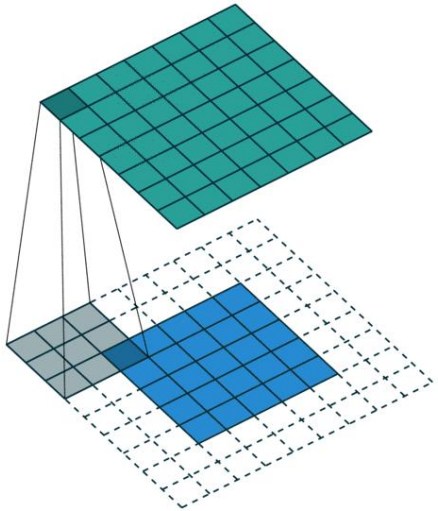
Kernel

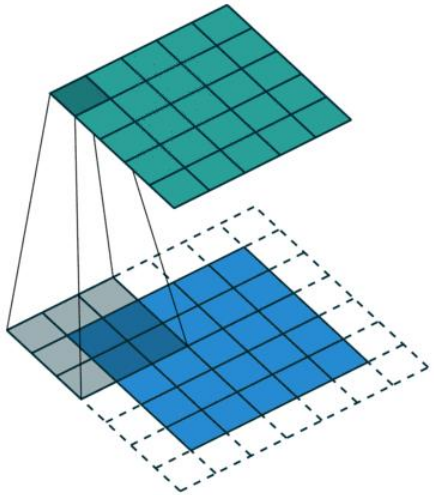
0	1
2	3

=

Output

0	3	8	4
9	19	25	10
21	37	43	16
6	7	8	0

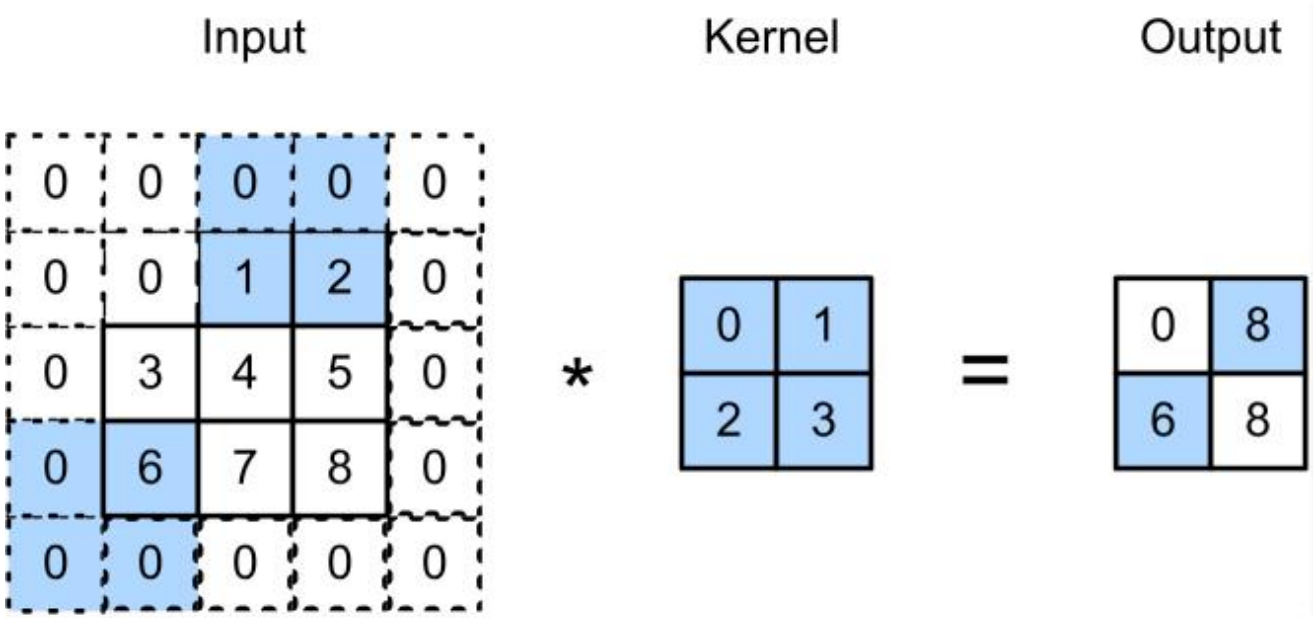


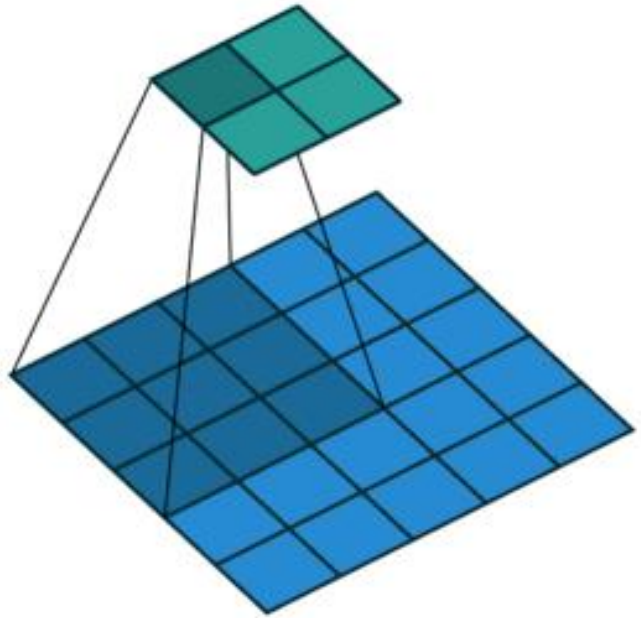


$$0 \times 0 + 0 \times 1 + 0 \times 2 + 0 \times 3 = 0$$

Stride

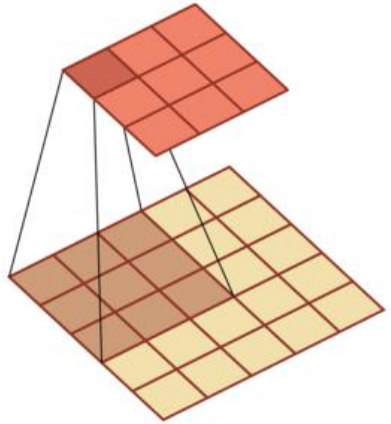
high=3, width=2



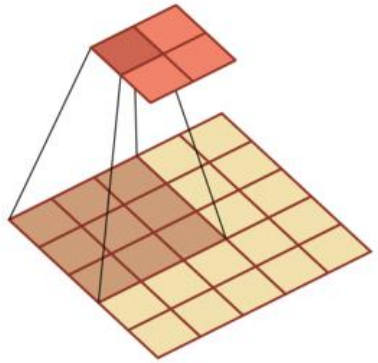


$$0\times0+0\times1+1\times2+2\times3=8$$
$$0\times0+6\times1+0\times2+0\times3=6$$

Stride & Padding

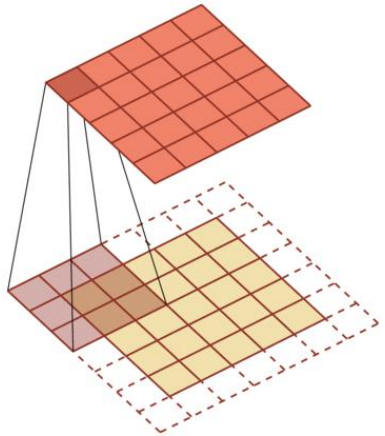


Slide = 1
Padding = 0

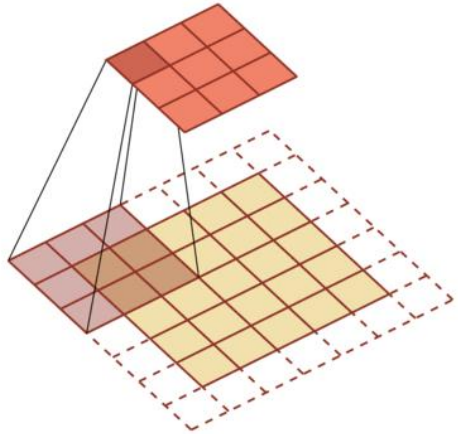


Slide = 2
Padding = 0

$$W_{out} = \left\lfloor \frac{W_{in} + 2P - K}{S} \right\rfloor + 1$$

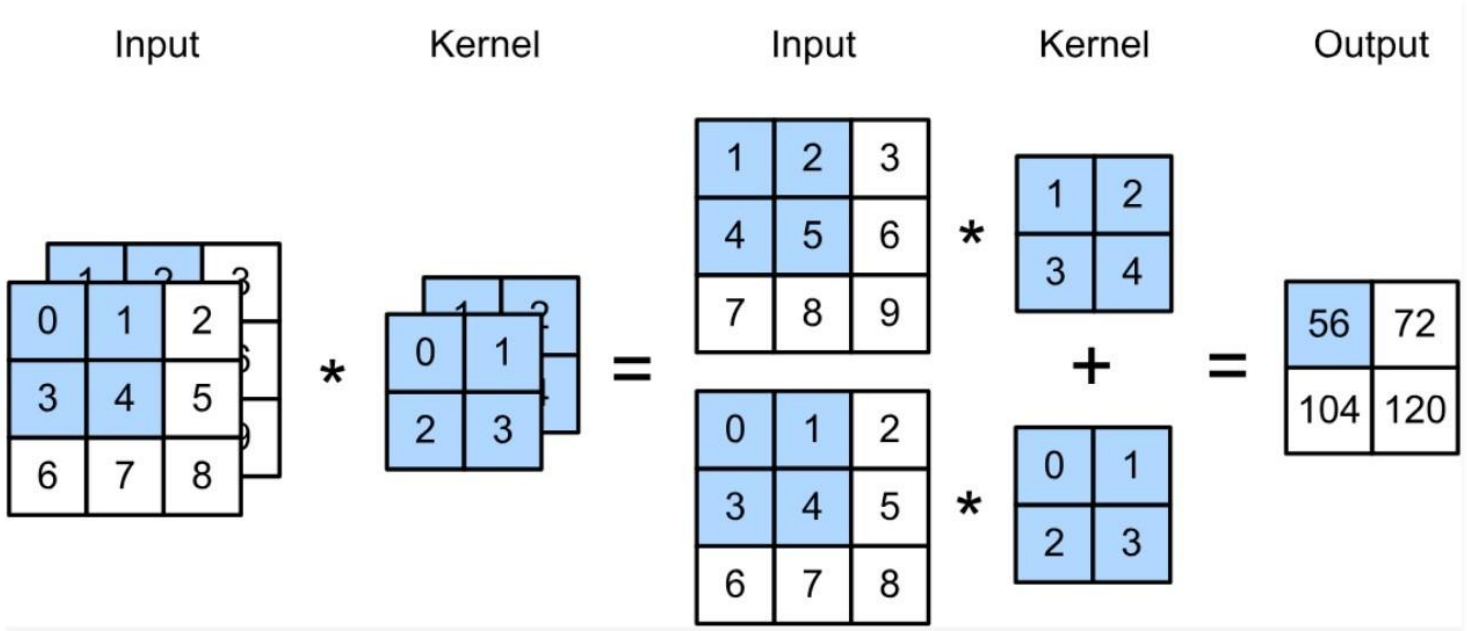


Slide = 1
Padding = 1



Slide = 2
Padding = 1

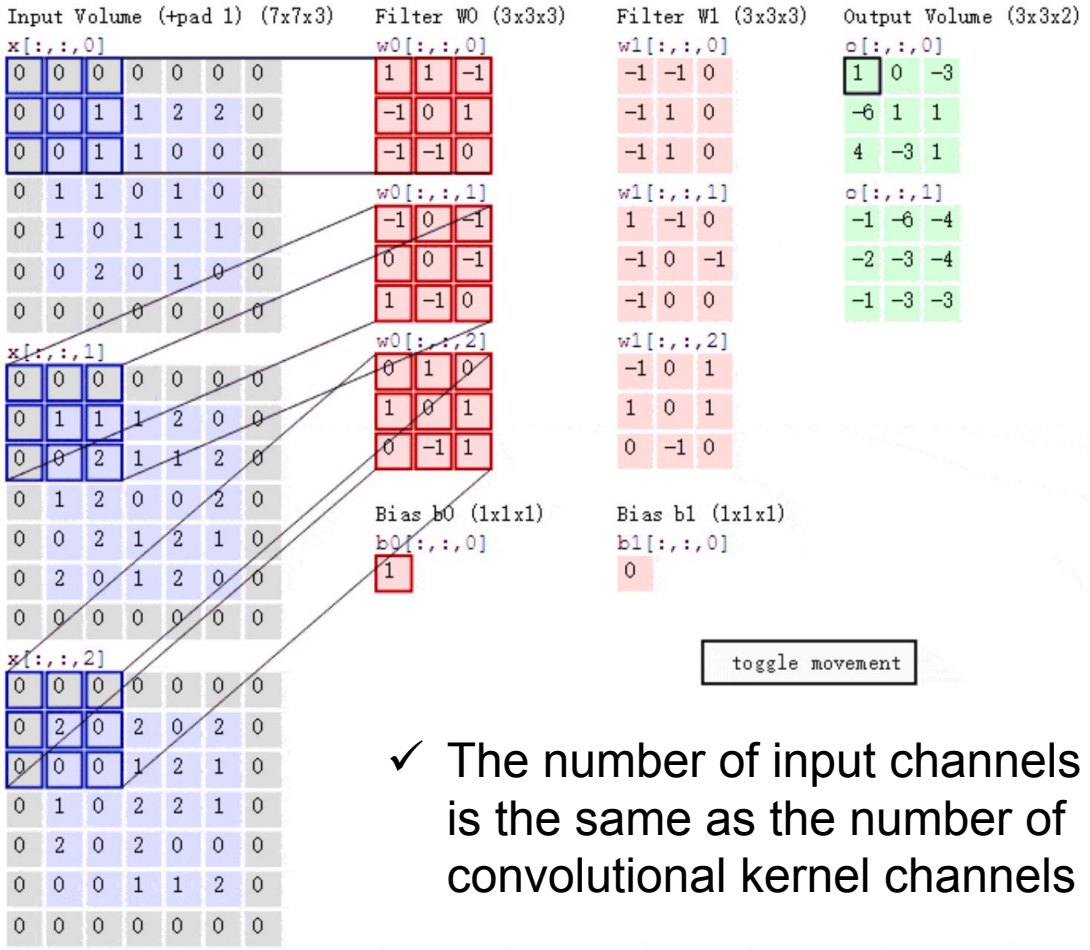
Multi-Channel



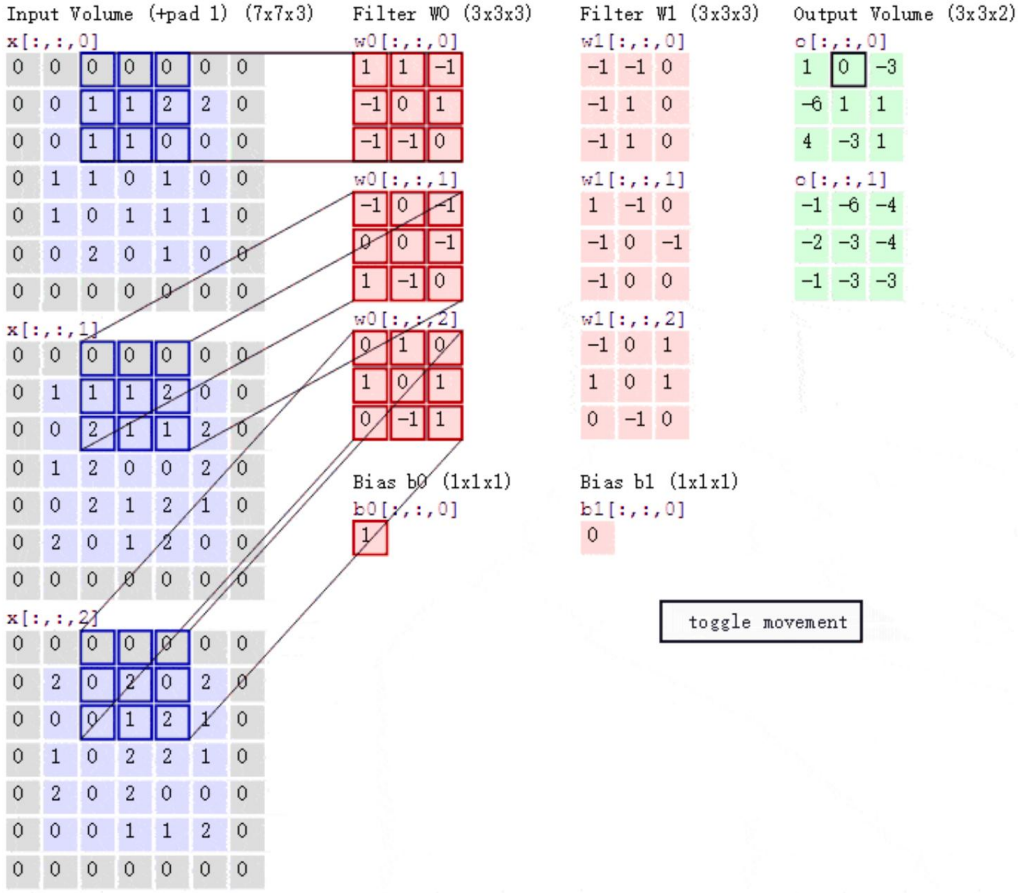
$$(1\times 1+2\times 2+4\times 3+5\times 4)$$
$$+(0\times 0+1\times 1+3\times 2+4\times 3)=56$$

上海交通大學 SHANGHAI JIAO TONG UNIVERSITY

Multi Channel Convolution

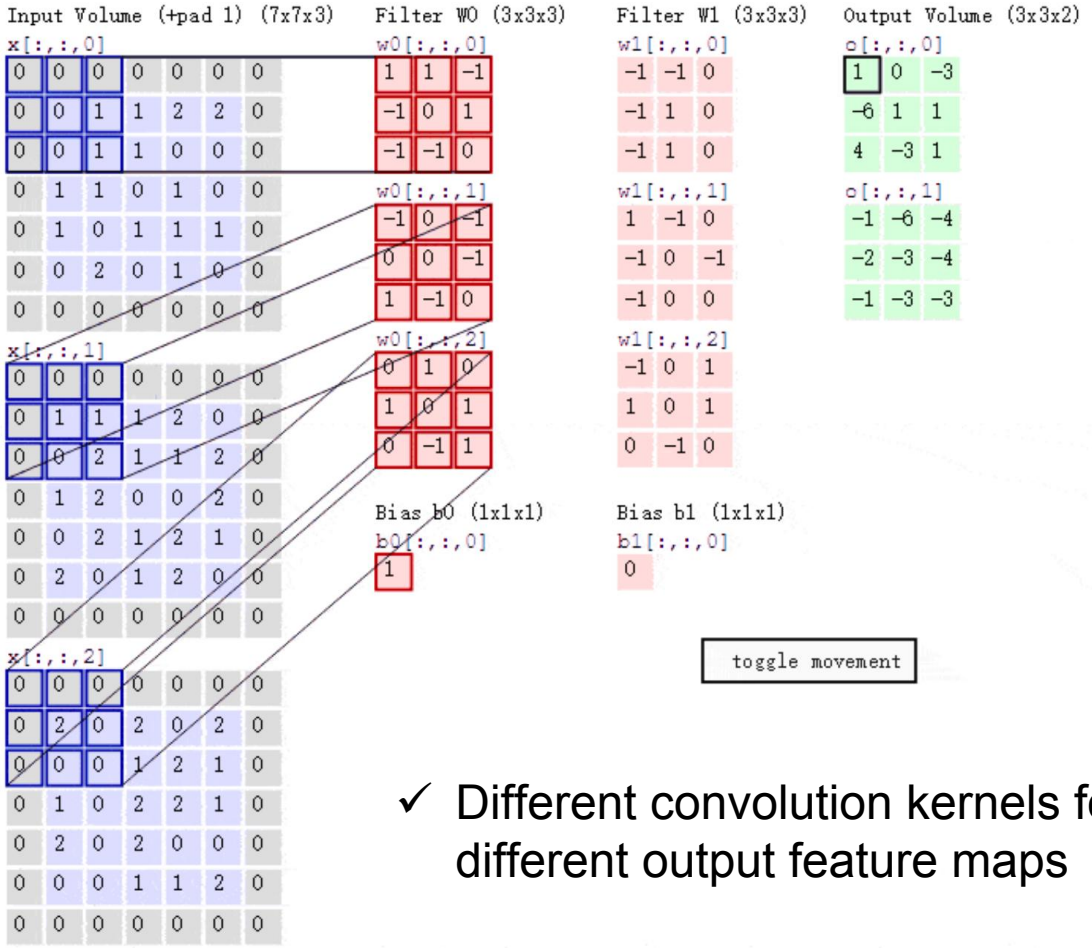


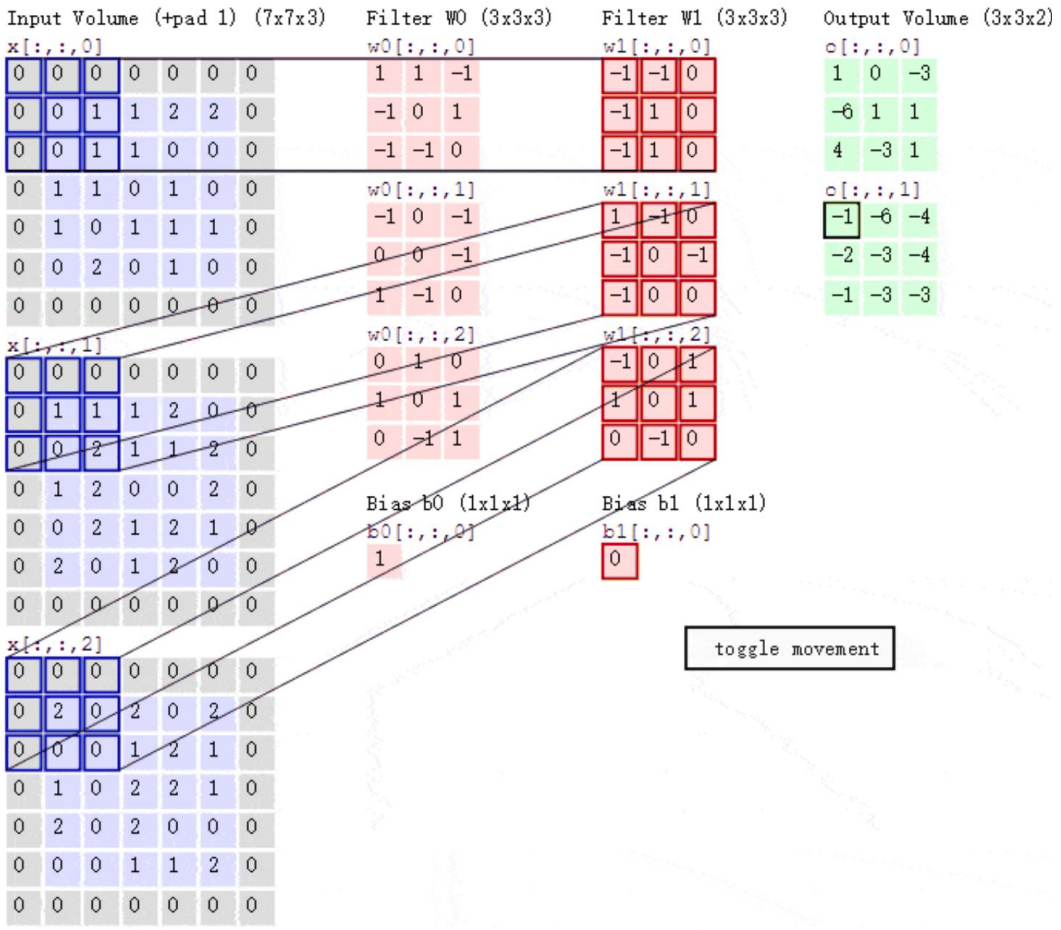
✓ The number of input channels is the same as the number of convolutional kernel channels



上海交通大学 SHANGHAI JIAO TONG UNIVERSITY

Multi Kernel Convolution



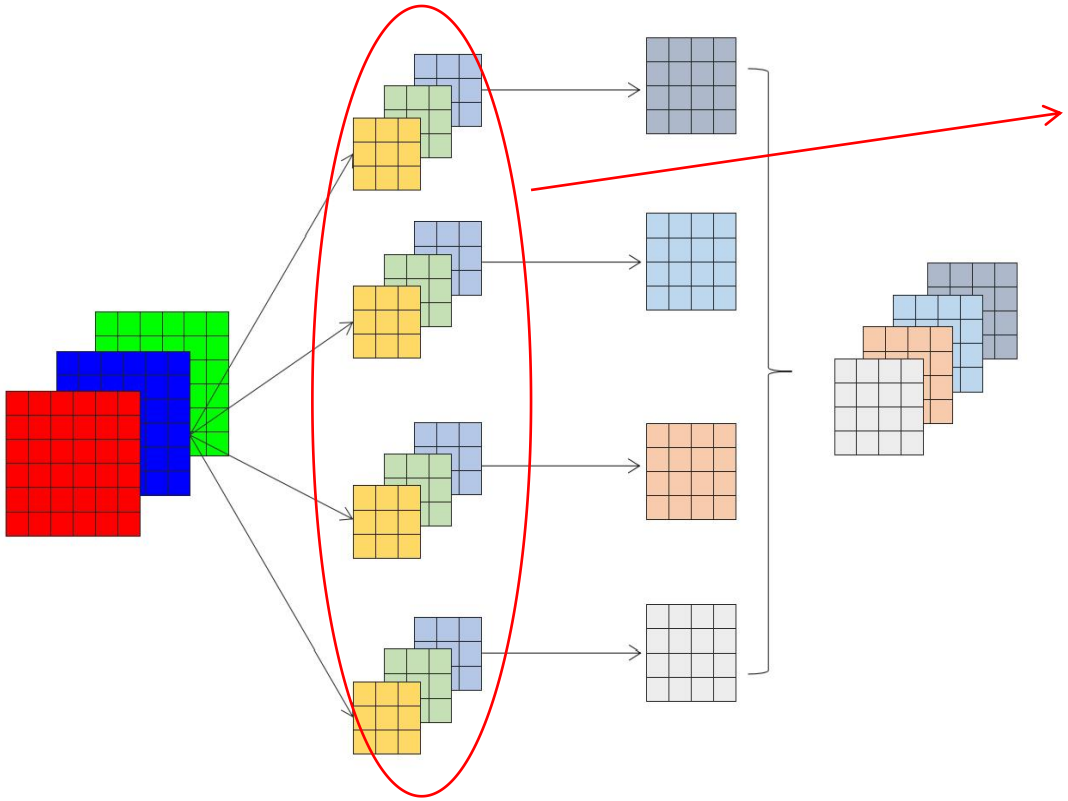


- ✓ Different convolution kernels form different output feature maps

Multi-Channel

- Each output channel can recognize one specific pattern.

Standard Convolution



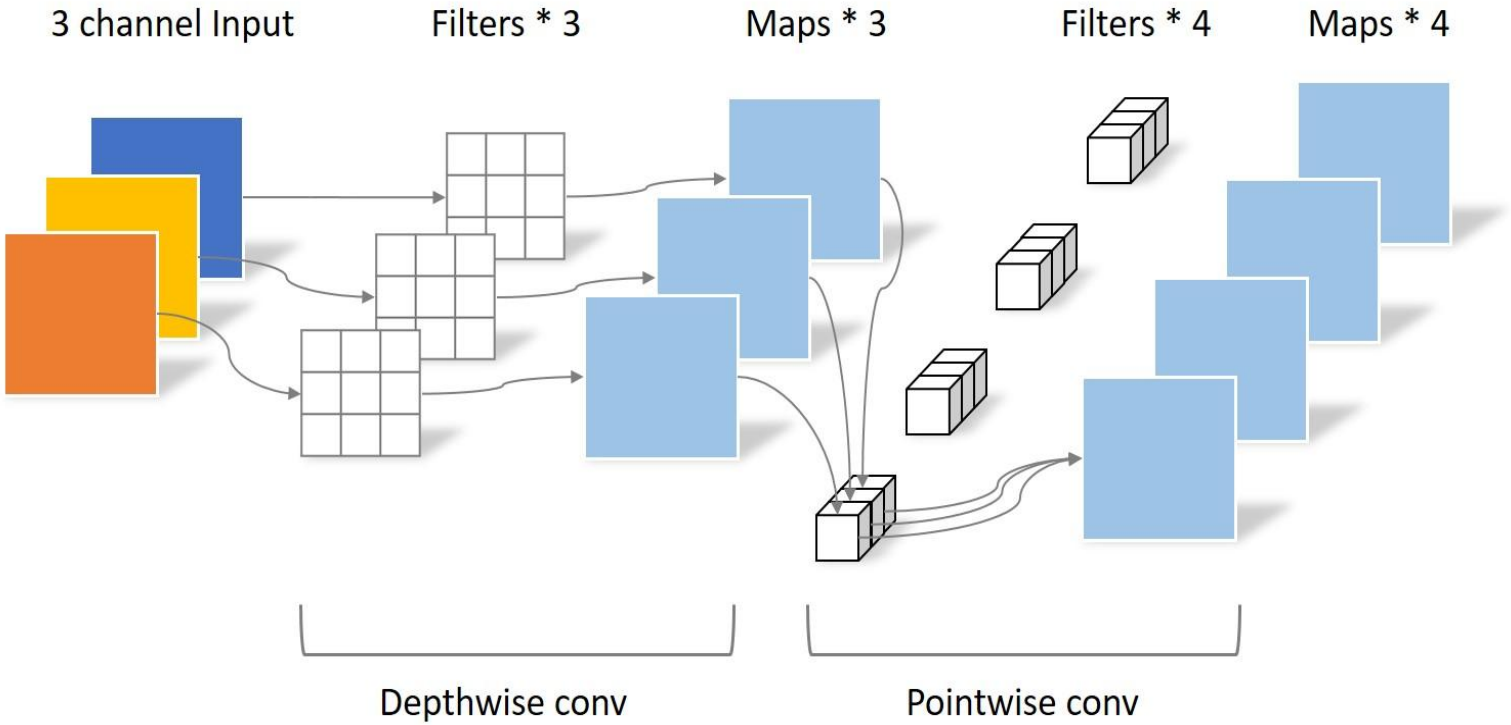
Two Stage:

1. Each channel corresponding to a convolution kernel process spatial information with wide and high dimensions;
2. Channel addition integrates channel information

It may result in parameter redundancy

3-channel input ⟶ 4-channel output: 4*3*3*3 = 108 paramaters

Depthwise Separable Convolution



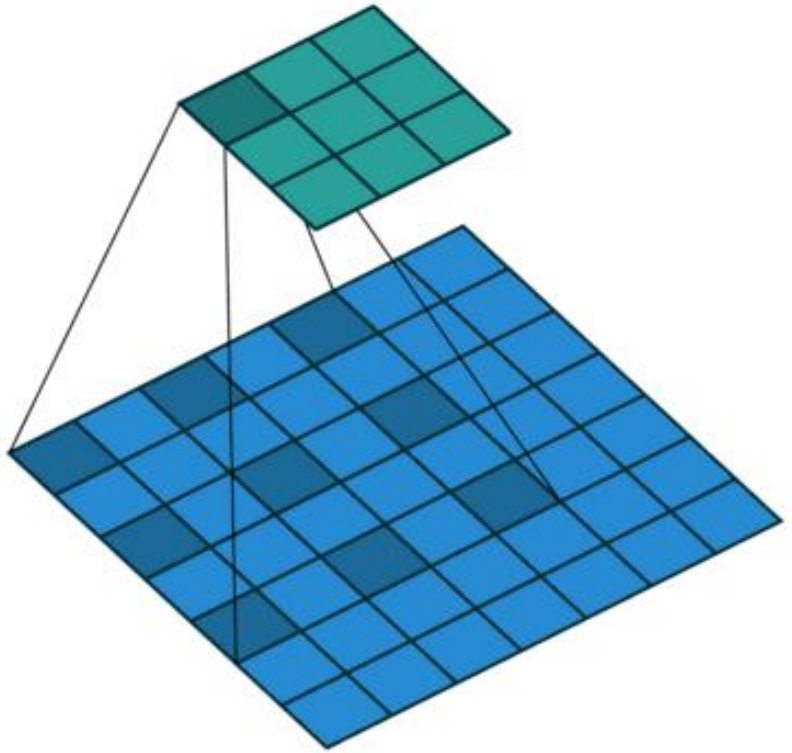
Only handle spatial information

Only Merge channel information

3-channel input ⟶4-channel output: (3*3*3 + 3*4*1) = 39 paramaters

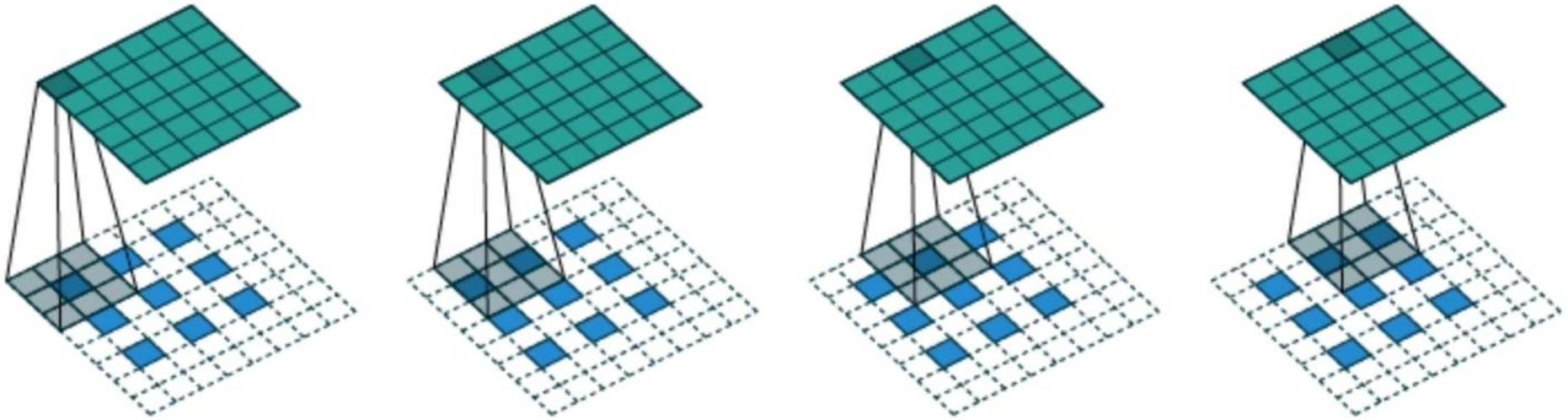
Variants: Dilated Convolution

- By sampling values at intervals, the receptive field is expanded, allowing the original convolutional kernel to cover a **larger receptive field** while maintaining the same number of parameters and computational cost.



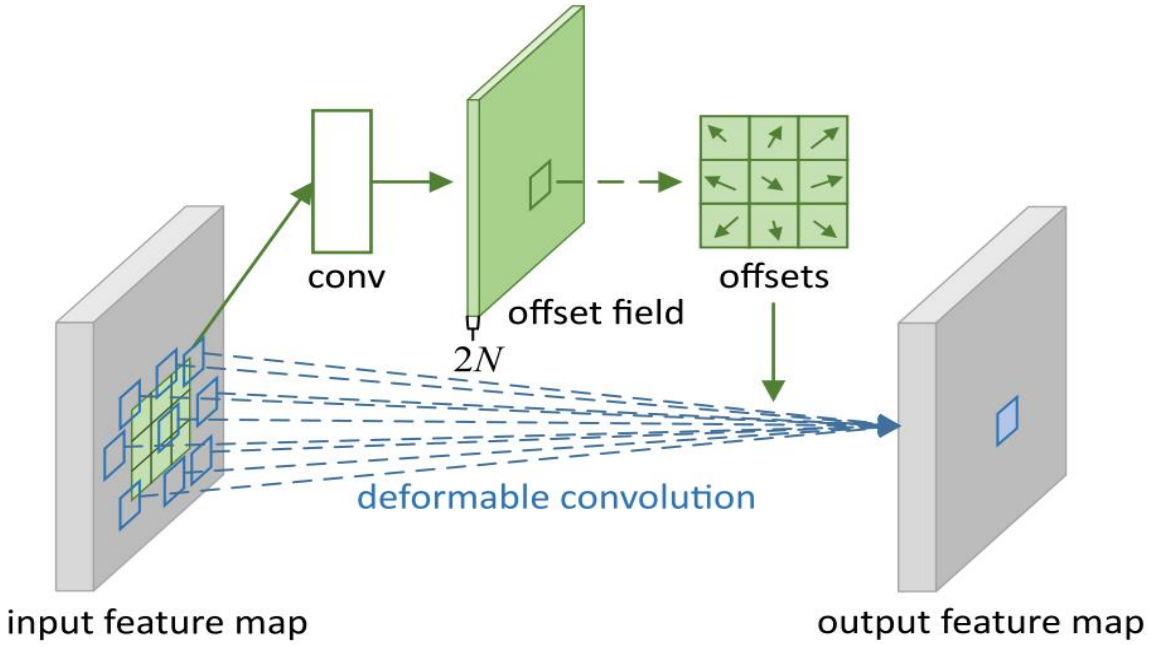
Variants: Deconvolution

- Upsample feature maps by reversing the spatial reduction of standard convolution, commonaly used in tasks like segmentation and image generation.



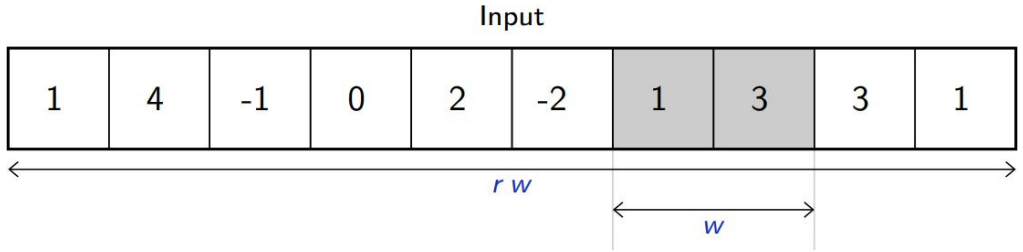
Variants: Deformable Convolution

- Adds learnable 2D offsets to regular grid sampling. The convolution input pixels are reorganized via offsets to achieve kernel shift.

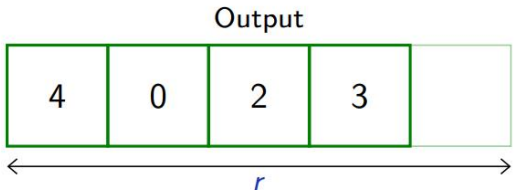


Pooling Layers

- To compute a low-dimension signal from a high-dimension one, grouping several activations into a single “more meaningful” one.
- Max Pooling computes max values over non-overlapping blocks. The average pooling computes average values per block instead of max values.

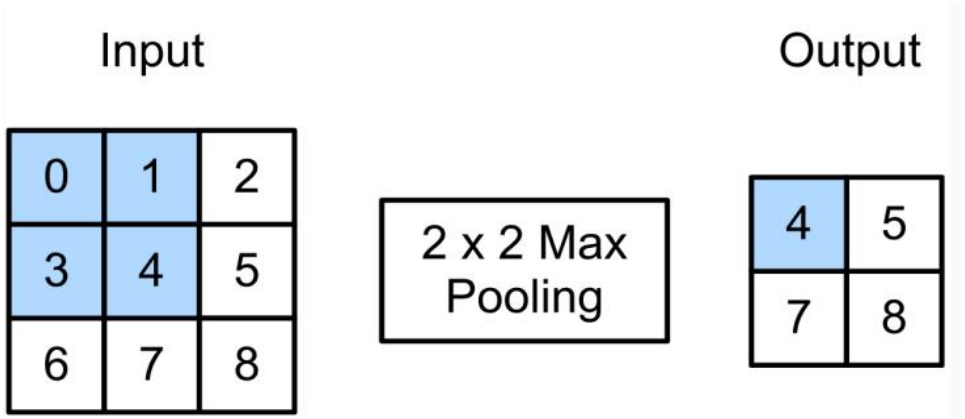


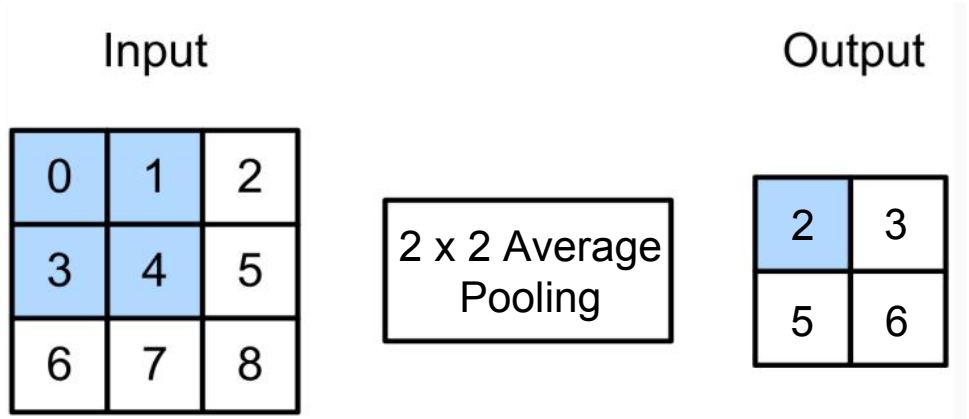
1d with a kernel of size 2



Pooling Layers

- Max Pooling v.s. Average Pooling



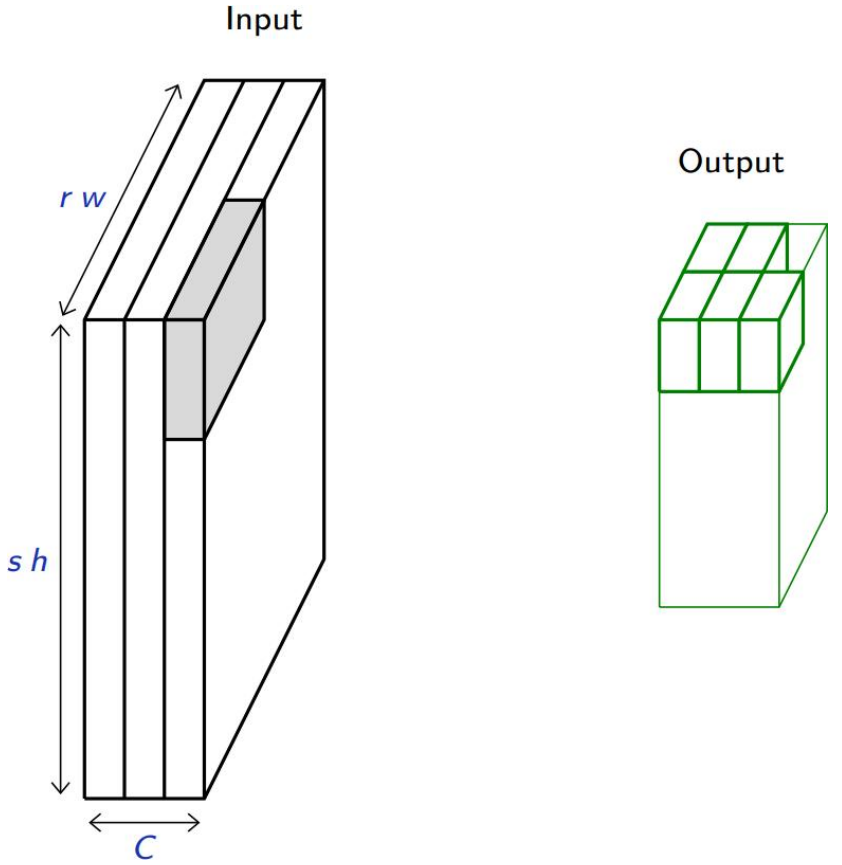


$$\max(0,1,3,4) = 4$$

$$\text{Average}(0,1,3,4) = 2$$

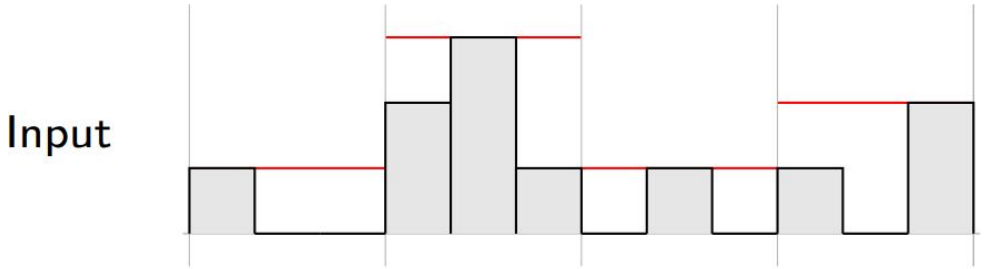
Pooling Layers

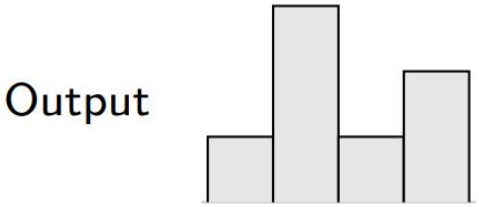
- Contrary to convolution, pooling is applied independently on each channel.
- There are as many channels as output.



Pooling provides invariance

- Pooling provides invariance to any permutation inside one of the cell.
- More practically, it provides a pseudo-invariance to deformations that result into local translations.





Invariance and Equivariance in CNN

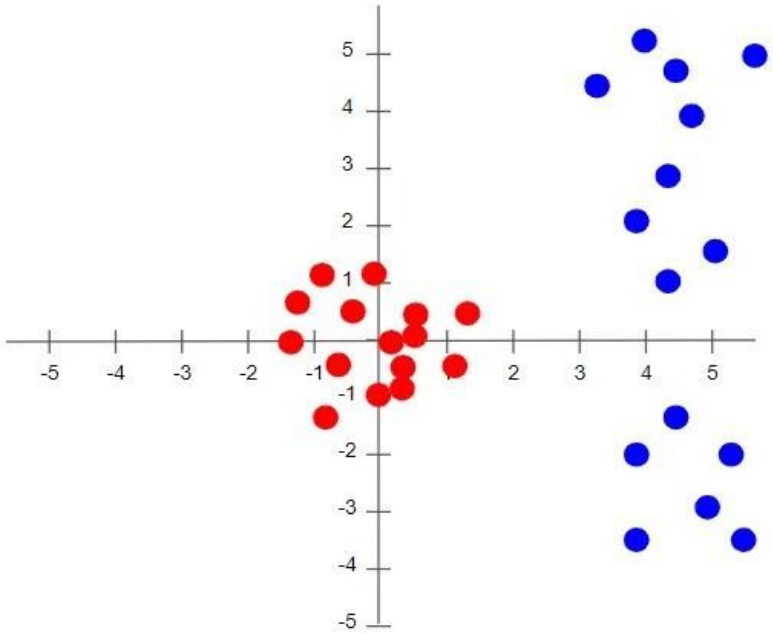
- The **translation equivariance** is obtained by means of the convolutional layers.
- **Translation invariance** is obtained in CNNs by means of the pooling layers.
- CNNs are **NOT** naturally equivariant and invariant to rotation, scaling, and affine transformations (if not local). Hence, **data augmentation** techniques must be used to make CNNs more robust to such geometric transformations.

Data Augmentation

Normalization

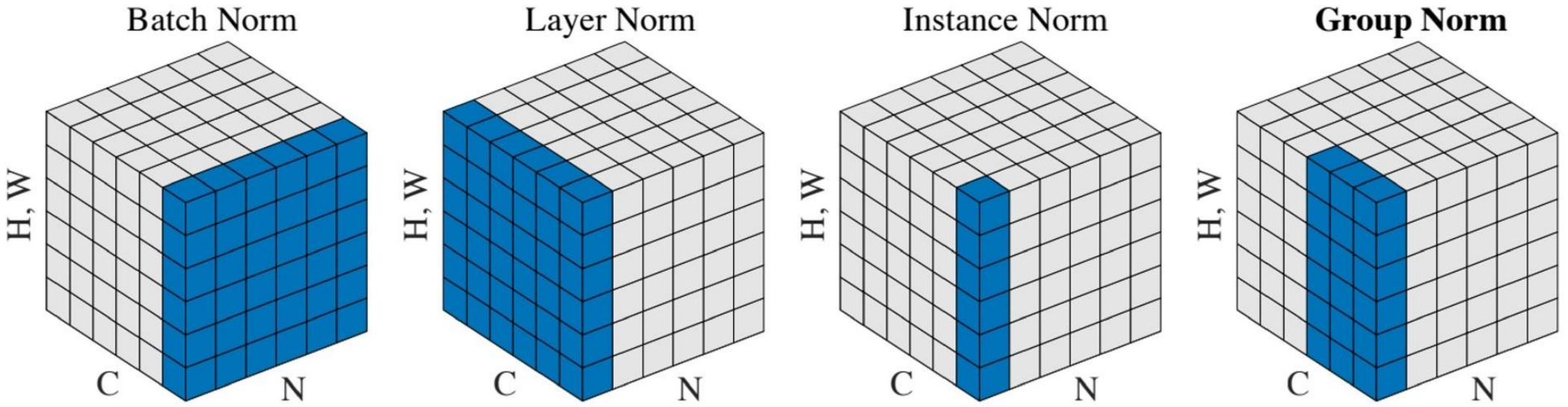
- **Idea**: “Normalize” the outputs of a layer so they have zero mean and unit variance
- **Types**: Batch normalization, Layer Normalization, Instance Normalization, …

$$\hat{x} = \frac{x - E(x)}{\sqrt{Var(x) + \mathcal{E}}}$$



Comparison of Normalization

- **Idea**: “Normalize” the outputs of a layer so they have zero mean and unit variance
- **Types**: Batch normalization, Layer Normalization, Instance Normalization, …



Batch Normalization: Train Time

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

Learnable scale and shift parameters:

$$\gamma, \beta \in \mathbb{R}^{D}$$

Learning $\gamma = \sigma, \beta = \mu$ will recover the identity function (in expectation)

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

Per-channel mean, shape is D

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

Per-channel std, shape is D

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

Normalized x, Shape is N x D

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

Output, Shape is N x D

Batch Normalization: Test time

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

Learnable scale and shift parameters:

$$\gamma, \beta \in \mathbb{R}^{D}$$

Learning $\gamma = \sigma, \beta = \mu$ will recover the identity function (in expectation)

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

Per-channel mean, shape is D

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

Per-channel std, shape is D

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

Normalized x, Shape is N x D

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

Output, Shape is N x D

Batch Normalization: Test time

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

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

Per-channel mean, shape is D

Learnable scale and shift parameters:

$$\gamma, \beta \in \mathbb{R}^D$$

Learning $\gamma = \sigma, \beta = \mu$ will recover the identity function (in expectation)

$$\mu_j^{test} = 0$$

For each training iteration:

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

$$\mu_j^{test} = 0.99\, \mu_j^{test} + 0.01\, \mu_j$$

(Similar for σ)

Batch Normalization for ConvNets

Batch Normalization for
fully-connected networks

$$x : N \times D$$

Normalize ↓

$$\mu, \sigma : 1 \times D$$

$$\gamma, \beta : 1 \times D$$

$$y = \frac{(x - \mu)}{\sigma}\gamma + \beta$$

Batch Normalization for
convolutional networks
(Spatial Batchnorm, BatchNorm2D)

$$x : N \times C \times H \times W$$

Normalize ↓ ↓ ↓

$$\mu, \sigma : 1 \times C \times 1 \times 1$$

$$\gamma, \beta : 1 \times C \times 1 \times 1$$

$$y = \frac{(x - \mu)}{\sigma}\gamma + \beta$$

Layer Normalization

Batch Normalization for convolutional networks

$$x : N \times C \times H \times W$$

Normalize

$$\mu, \sigma : 1 \times C \times 1 \times 1$$

$$\gamma, \beta : 1 \times C \times 1 \times 1$$

$$y = \frac{(x - \mu)}{\sigma}\gamma + \beta$$

Layer Normalization for fully-connected networks

Same behavior at train and test!

Used in RNNs, Transformers

$$x : N \times D$$

Normalize

$$\mu, \sigma : N \times 1$$

$$\gamma, \beta : 1 \times D$$

$$y = \frac{(x - \mu)}{\sigma}\gamma + \beta$$

Instance Normalization

Batch Normalization for convolutional networks

$$x : N \times C \times H \times W$$

Normalize ↓ ↓ ↓

$$\mu, \sigma : 1 \times C \times 1 \times 1$$

$$\gamma, \beta : 1 \times C \times 1 \times 1$$

$$y = \frac{(x - \mu)}{\sigma}\gamma + \beta$$

Instance Normalization for convolutional networks

$$x : N \times C \times H \times W$$

Normalize ↓ ↓ ↓

$$\mu, \sigma : N \times C \times 1 \times 1$$

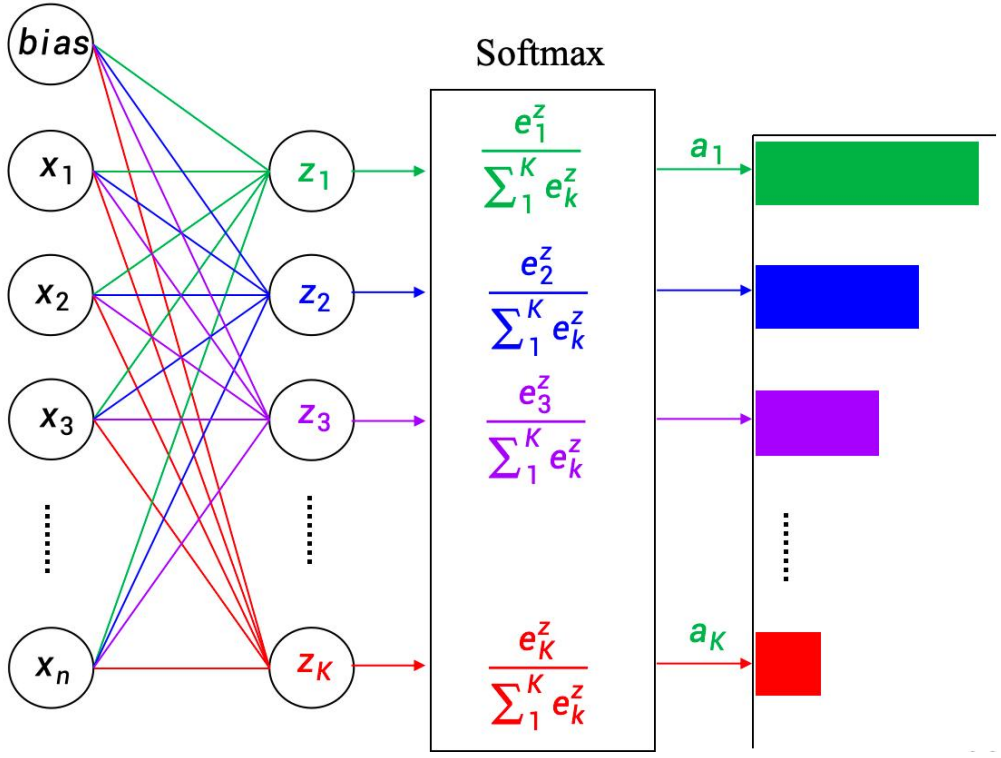
$$\gamma, \beta : 1 \times C \times 1 \times 1$$

$$y = \frac{(x - \mu)}{\sigma}\gamma + \beta$$

Fully-Connected Layers

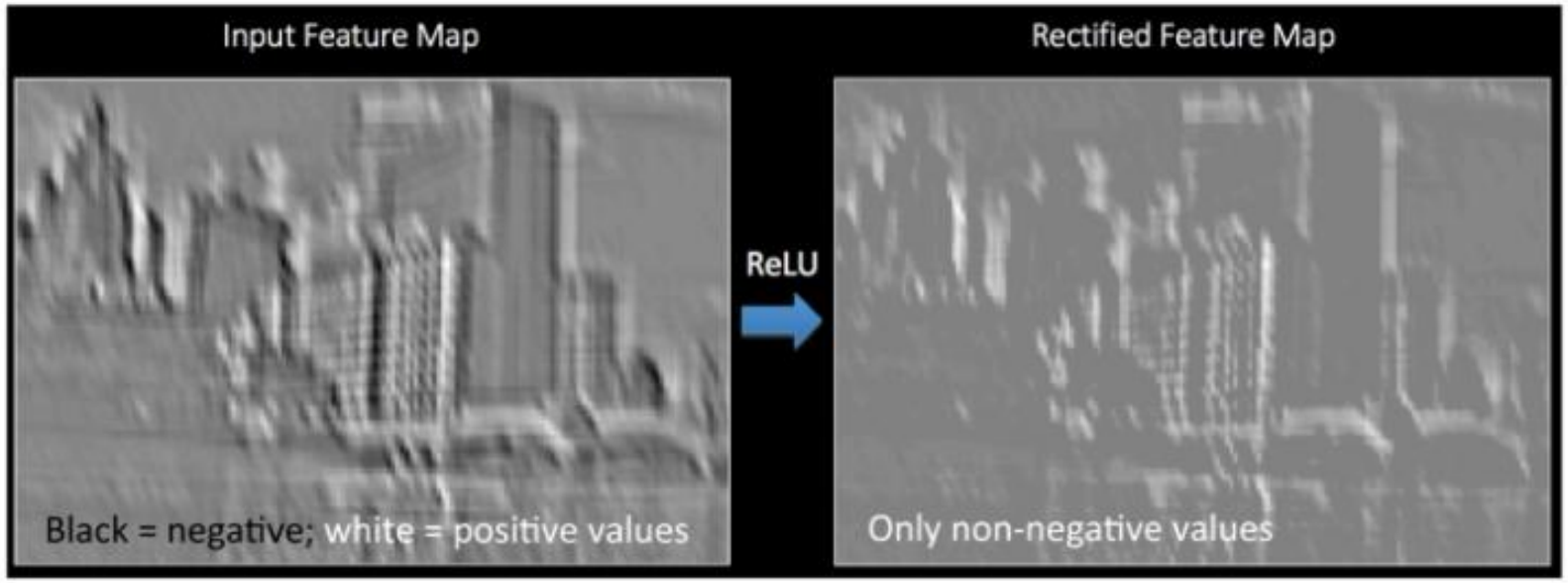
- Convolutional layers extract feature

- Fully connected layer enables global reasoning by combining all learned features into class scores or regression outputs



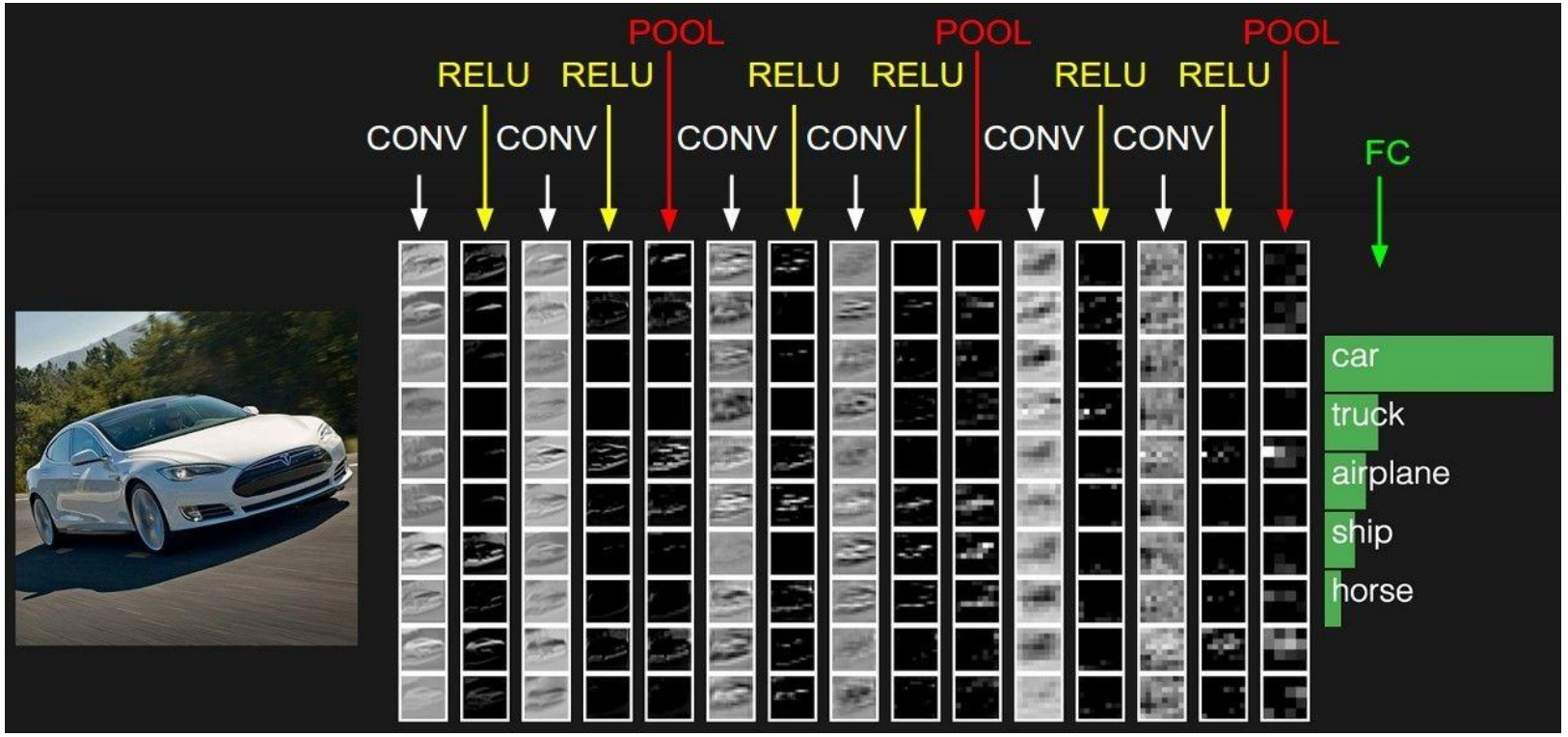
Activation Function

- Convolution operation is essentially a **linear operation**.
- Similar to fully connected neural networks, nonlinearity needs to be introduced through activation functions.



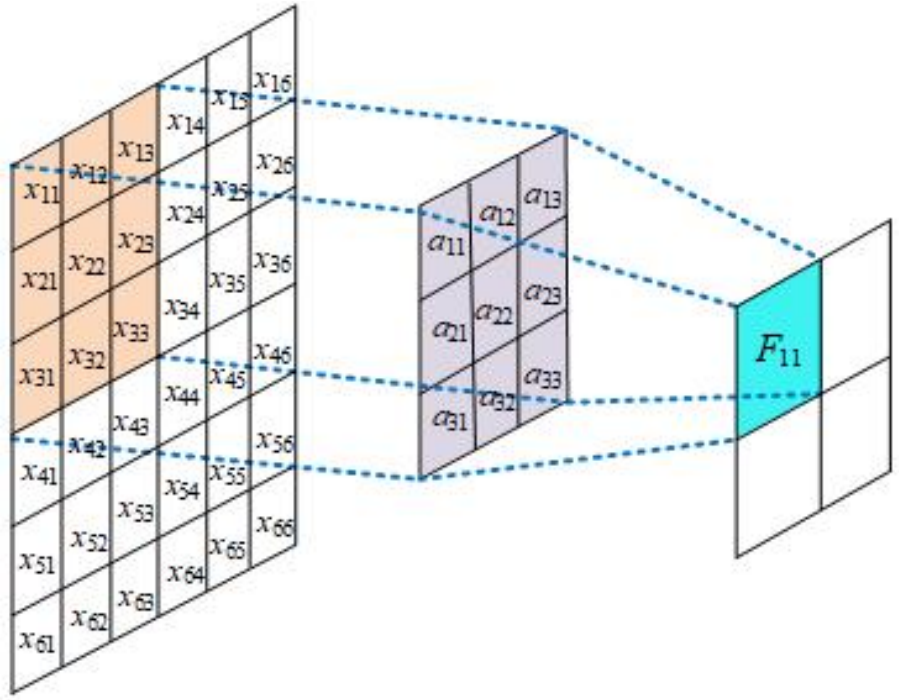
How Convolution works

- Extract features hierarchically



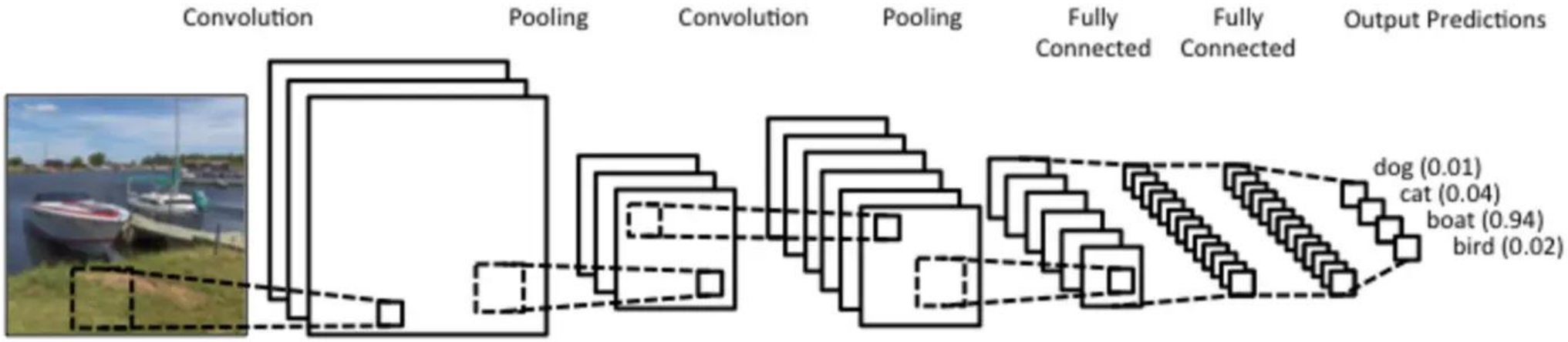
How Convolution works

- **Weight Sharing**: Each neuron in a convolutional layer connects only to a local receptive field, reducing parameters and computational cost.



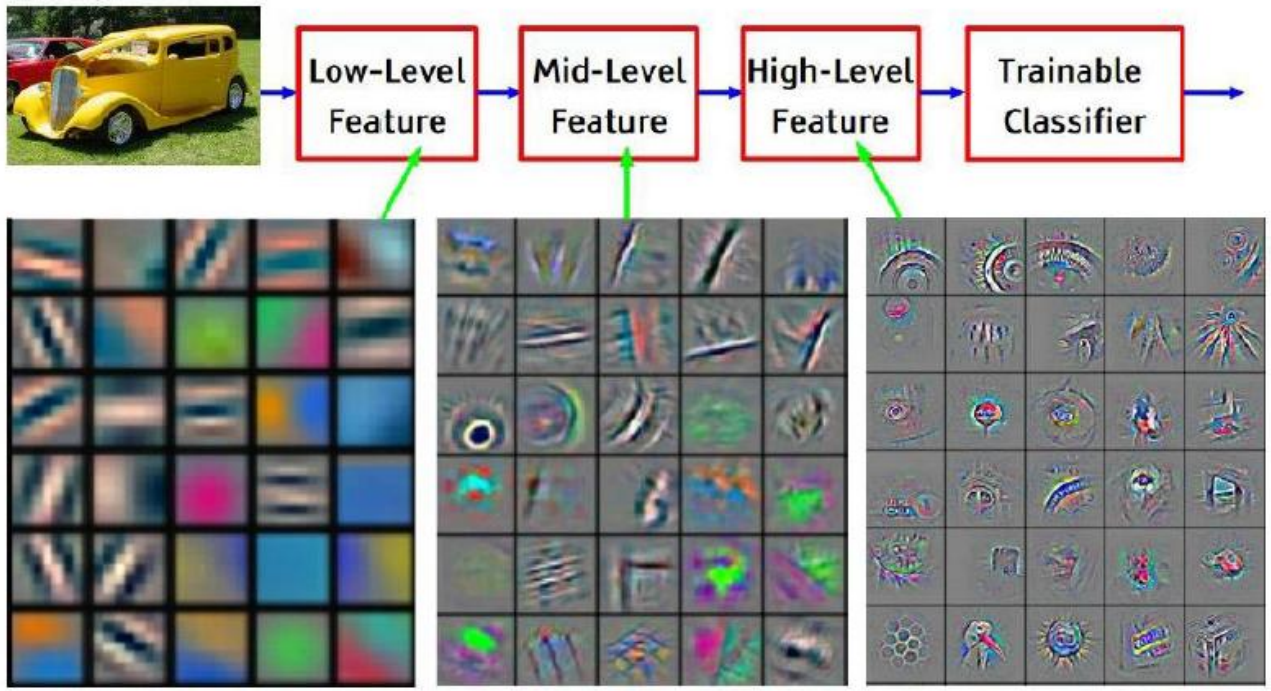
How Convolution works

- **Local Connectivity**: Unlike fully connected layers, convolution applies small kernels to local regions of the input.



How Convolution works

- **Hierarchical Feature Learn**: Early layers detect low-level features, while deeper layers combine them into high-level features.

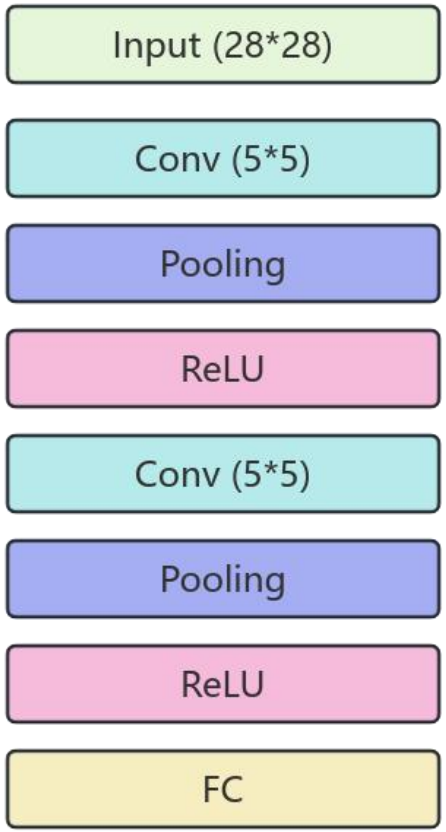


Example: Classification task on MNIST

- Build a simple convolutional neural network to achieve the same 10 classification tasks on the MNIST dataset.

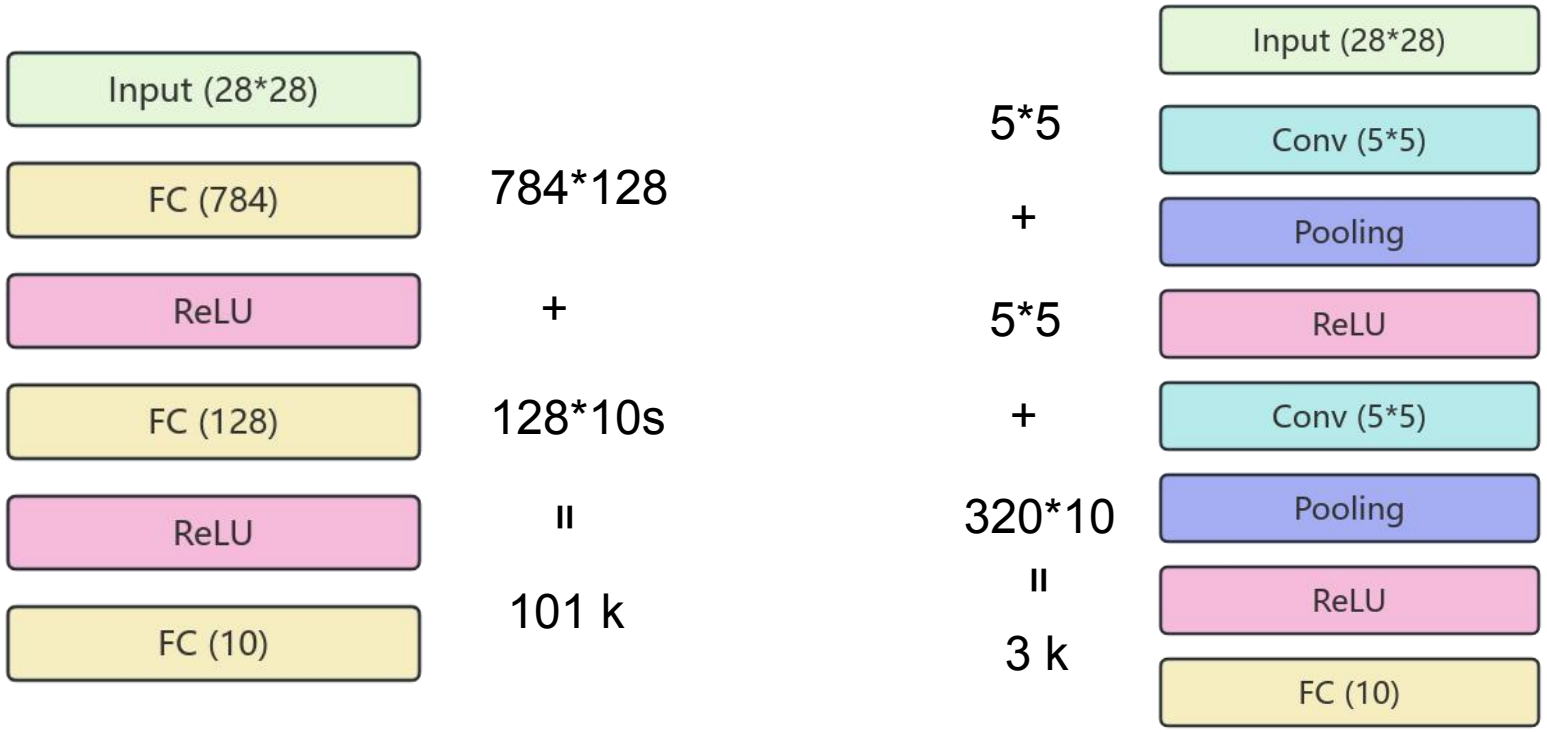
```python
class ImprovedCNN(nn.Module):
    def __init__(self):
        super(ImprovedCNN, self).__init__()
        self.conv1 = nn.Conv2d(1, 10, kernel_size=5)
        self.conv2 = nn.Conv2d(10, 20, kernel_size=5)
        self.pool = nn.MaxPool2d(2)
        self.relu = nn.ReLU()
        self.fc = nn.Linear(320, 10)

    def forward(self, x):
        x = self.pool(self.relu(self.conv1(x)))  # conv+relu+pooling
        x = self.pool(self.relu(self.conv2(x)))
        x = x.view(-1, 320)
        x = self.fc(x)  # fc to classification
        return x
```

code here



Comparison on parameter complexity: FCN vs. CNN

- Convolutional networks can greatly reduce the number of parameters.



上海交通大学
SHANGHAI JIAO TONG UNIVERSITY

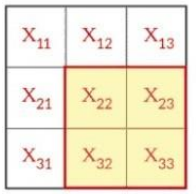
Contents

- Convolutional Neural Network
- **Training CNNs**
- Classical CNN Architectures

Back Propagation in CNN

- In the backward pass, we get the loss gradient with respect to the next layer
- In CNNs the loss gradient is computed w.r.t the input and also w.r.t the filter
  - ✓ Horizontal and vertical stride = 1
  - ✓ Convolution between Input X and Filter F, and output is O.

$$\begin{bmatrix} O_{11} & O_{12} \\ O_{21} & O_{22} \end{bmatrix} \text{ Output O} = \text{Convolution}\left(\begin{bmatrix} X_{11} & X_{12} & X_{13} \\ X_{21} & X_{22} & X_{23} \\ X_{31} & X_{32} & X_{33} \end{bmatrix} \text{ Input X}, \begin{bmatrix} F_{11} & F_{12} \\ F_{21} & F_{22} \end{bmatrix} \text{ Filter F} \right)$$



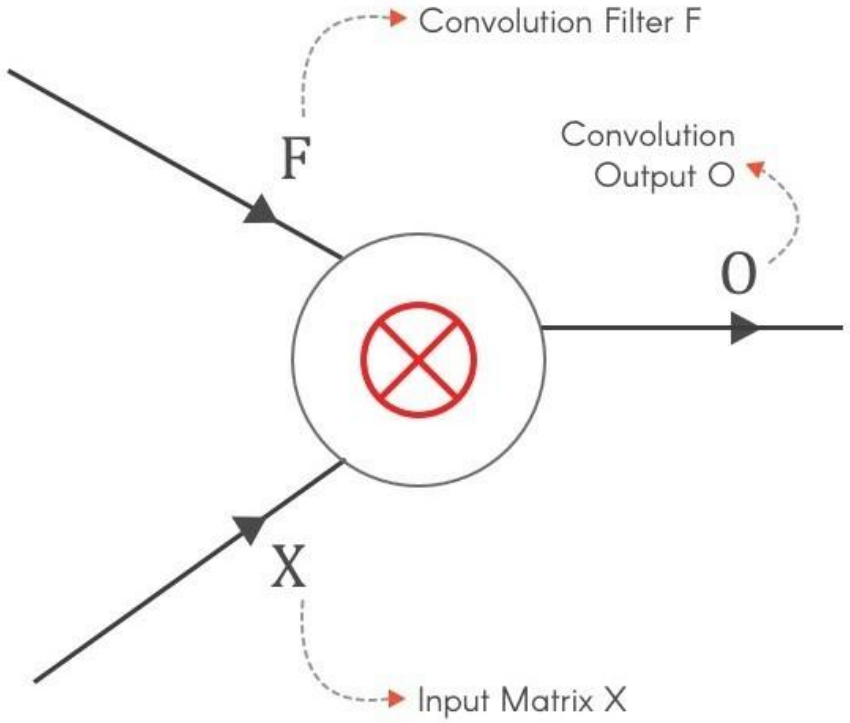
$$\begin{bmatrix} X_{11} & X_{12} & X_{13} \\ X_{21} & X_{22}F_{11} & X_{23}F_{12} \\ X_{31} & X_{32}F_{21} & X_{33}F_{22} \end{bmatrix}$$

$$O_{11} = X_{11}F_{11} + X_{12}F_{12} + X_{21}F_{21} + X_{22}F_{22}$$
$$O_{12} = X_{12}F_{11} + X_{13}F_{12} + X_{22}F_{21} + X_{23}F_{22}$$
$$O_{21} = X_{21}F_{11} + X_{22}F_{12} + X_{31}F_{21} + X_{32}F_{22}$$
$$O_{22} = X_{22}F_{11} + X_{23}F_{12} + X_{32}F_{21} + X_{33}F_{22}$$

https://deeplearning.cs.cmu.edu/F21/document/recitation/Recitation5/CNN_Backprop_Recitation_5_F21.pdf

Back Propagation in CNN

- Loss gradient: we want to calculate the gradients w.r.t to input ‘X’ and filter ‘F’



Back Propagation in CNN

- Loss gradient w.r.t the filter:
  - ✓ Use the chain rule to obtain the gradient w.r.t the filter

$$\frac{\partial L}{\partial F} = \frac{\partial L}{\partial O} * \frac{\partial O}{\partial F}$$

Gradient to update Filter F

Loss Gradient from previous layer

Local Gradients

For every element of F

$$\frac{\partial L}{\partial F_i} = \sum_{k=1}^{M} \frac{\partial L}{\partial O_k} * \frac{\partial O_k}{\partial F_i}$$

Back Propagation in CNN

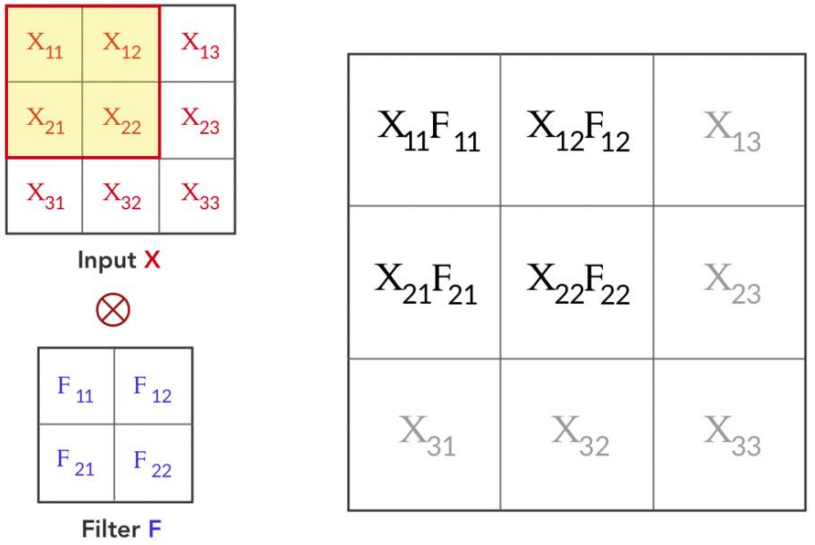
- Loss gradient w.r.t the filter:
  - ✓ The chain rule summation can be expanded as:

$$\frac{\partial L}{\partial F_{11}} = \frac{\partial L}{\partial O_{11}} * \frac{\partial O_{11}}{\partial F_{11}} + \frac{\partial L}{\partial O_{12}} * \frac{\partial O_{12}}{\partial F_{11}} + \frac{\partial L}{\partial O_{21}} * \frac{\partial O_{21}}{\partial F_{11}} + \frac{\partial L}{\partial O_{22}} * \frac{\partial O_{22}}{\partial F_{11}}$$

$$\frac{\partial L}{\partial F_{12}} = \frac{\partial L}{\partial O_{11}} * \frac{\partial O_{11}}{\partial F_{12}} + \frac{\partial L}{\partial O_{12}} * \frac{\partial O_{12}}{\partial F_{12}} + \frac{\partial L}{\partial O_{21}} * \frac{\partial O_{21}}{\partial F_{12}} + \frac{\partial L}{\partial O_{22}} * \frac{\partial O_{22}}{\partial F_{12}}$$

$$\frac{\partial L}{\partial F_{21}} = \frac{\partial L}{\partial O_{11}} * \frac{\partial O_{11}}{\partial F_{21}} + \frac{\partial L}{\partial O_{12}} * \frac{\partial O_{12}}{\partial F_{21}} + \frac{\partial L}{\partial O_{21}} * \frac{\partial O_{21}}{\partial F_{21}} + \frac{\partial L}{\partial O_{22}} * \frac{\partial O_{22}}{\partial F_{21}}$$

$$\frac{\partial L}{\partial F_{22}} = \frac{\partial L}{\partial O_{11}} * \frac{\partial O_{11}}{\partial F_{22}} + \frac{\partial L}{\partial O_{12}} * \frac{\partial O_{12}}{\partial F_{22}} + \frac{\partial L}{\partial O_{21}} * \frac{\partial O_{21}}{\partial F_{22}} + \frac{\partial L}{\partial O_{22}} * \frac{\partial O_{22}}{\partial F_{22}}$$



$$O_{11} = X_{11}F_{11} + X_{12}F_{12} + X_{21}F_{21} + X_{22}F_{22}$$

Back Propagation in CNN

- Loss gradient w.r.t the filter:
  - ✓ Replacing the local gradients of the filter, we can represent it as a convolution operation between input X and loss gradient ∂L/∂O as shown below:

$$\begin{bmatrix} \frac{\partial L}{\partial F_{11}} & \frac{\partial L}{\partial F_{12}} \\ \frac{\partial L}{\partial F_{21}} & \frac{\partial L}{\partial F_{22}} \end{bmatrix} = \text{Convolution} \left(\begin{bmatrix} X_{11} & X_{12} & X_{13} \\ X_{21} & X_{22} & X_{23} \\ X_{31} & X_{32} & X_{33} \end{bmatrix}, \begin{bmatrix} \frac{\partial L}{\partial O_{11}} & \frac{\partial L}{\partial O_{12}} \\ \frac{\partial L}{\partial O_{21}} & \frac{\partial L}{\partial O_{22}} \end{bmatrix} \right)$$

where

$$\begin{bmatrix} X_{11} & X_{12} & X_{13} \\ X_{21} & X_{22} & X_{23} \\ X_{31} & X_{32} & X_{33} \end{bmatrix} = \text{Input X} \qquad \begin{bmatrix} \frac{\partial L}{\partial O_{11}} & \frac{\partial L}{\partial O_{12}} \\ \frac{\partial L}{\partial O_{21}} & \frac{\partial L}{\partial O_{22}} \end{bmatrix} = \frac{\partial L}{\partial O} \text{ Loss gradient from previous layer}$$

$$\frac{\partial L}{\partial F_{11}} = \frac{\partial L}{\partial O_{11}} * X_{11} + \frac{\partial L}{\partial O_{12}} * X_{12} + \frac{\partial L}{\partial O_{21}} * X_{21} + \frac{\partial L}{\partial O_{22}} * X_{22}$$

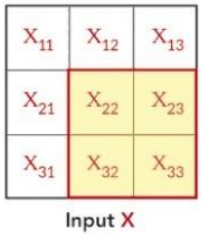
$$\frac{\partial L}{\partial F_{12}} = \frac{\partial L}{\partial O_{11}} * X_{12} + \frac{\partial L}{\partial O_{12}} * X_{13} + \frac{\partial L}{\partial O_{21}} * X_{22} + \frac{\partial L}{\partial O_{22}} * X_{23}$$

$$\frac{\partial L}{\partial F_{21}} = \frac{\partial L}{\partial O_{11}} * X_{21} + \frac{\partial L}{\partial O_{12}} * X_{22} + \frac{\partial L}{\partial O_{21}} * X_{31} + \frac{\partial L}{\partial O_{22}} * X_{32}$$

$$\frac{\partial L}{\partial F_{22}} = \frac{\partial L}{\partial O_{11}} * X_{22} + \frac{\partial L}{\partial O_{12}} * X_{23} + \frac{\partial L}{\partial O_{21}} * X_{32} + \frac{\partial L}{\partial O_{22}} * X_{33}$$

Back Propagation in CNN

- Loss gradient w.r.t the input:



X_{11}	X_{12}	X_{13}
X_{21}	$X_{22}F_{11}$	$X_{23}F_{12}$
X_{31}	$X_{32}F_{21}$	$X_{33}F_{22}$

$$O_{11} = X_{11}F_{11} + X_{12}F_{12} + X_{21}F_{21} + X_{22}F_{22}$$
$$O_{12} = X_{12}F_{11} + X_{13}F_{12} + X_{22}F_{21} + X_{23}F_{22}$$
$$O_{21} = X_{21}F_{11} + X_{22}F_{12} + X_{31}F_{21} + X_{32}F_{22}$$
$$O_{22} = X_{22}F_{11} + X_{23}F_{12} + X_{32}F_{21} + X_{33}F_{22}$$

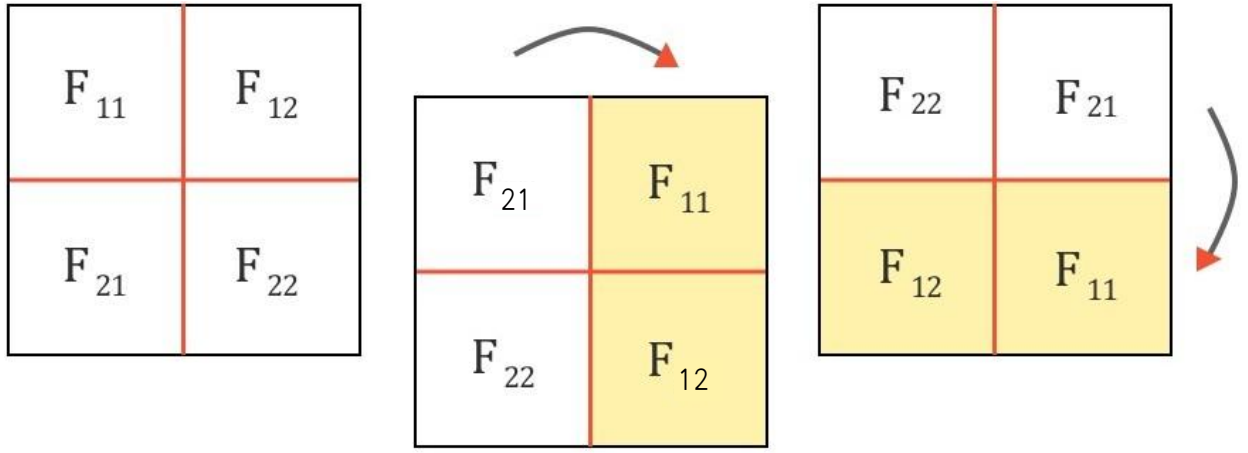
For every element of X_i

$$\frac{\partial L}{\partial X_i} = \sum_{k=1}^{M} \frac{\partial L}{\partial O_k} * \frac{\partial O_k}{\partial X_i}$$

$$\frac{\partial L}{\partial X_{11}} = \frac{\partial L}{\partial O_{11}} * F_{11}$$
$$\frac{\partial L}{\partial X_{12}} = \frac{\partial L}{\partial O_{11}} * F_{12} + \frac{\partial L}{\partial O_{12}} * F_{11}$$
$$\frac{\partial L}{\partial X_{13}} = \frac{\partial L}{\partial O_{12}} * F_{12}$$
$$\frac{\partial L}{\partial X_{21}} = \frac{\partial L}{\partial O_{11}} * F_{21} + \frac{\partial L}{\partial O_{21}} * F_{11}$$
$$\frac{\partial L}{\partial X_{22}} = \frac{\partial L}{\partial O_{11}} * F_{22} + \frac{\partial L}{\partial O_{12}} * F_{21} + \frac{\partial L}{\partial O_{21}} * F_{12} + \frac{\partial L}{\partial O_{22}} * F_{11}$$
$$\frac{\partial L}{\partial X_{23}} = \frac{\partial L}{\partial O_{12}} * F_{22} + \frac{\partial L}{\partial O_{22}} * F_{12}$$
$$\frac{\partial L}{\partial X_{31}} = \frac{\partial L}{\partial O_{21}} * F_{21}$$
$$\frac{\partial L}{\partial X_{32}} = \frac{\partial L}{\partial O_{21}} * F_{22} + \frac{\partial L}{\partial O_{22}} * F_{21}$$
$$\frac{\partial L}{\partial X_{33}} = \frac{\partial L}{\partial O_{22}} * F_{22}$$

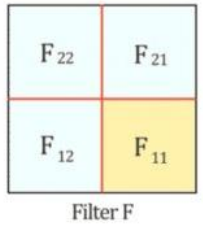
Back Propagation in CNN

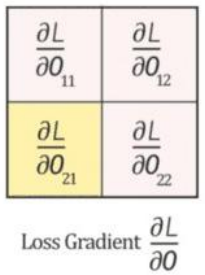
- Loss gradient w.r.t the input:
  - ✓ First, let us rotate the Filter F by 180 degrees. This is done by flipping it first vertically and then horizontally

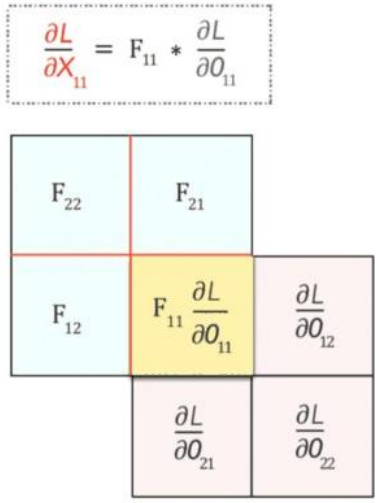


Back Propagation in CNN

- Loss gradient w.r.t the input:
  - ✓ We see that the loss gradient w.r.t the input is given as a full convolution between the filter and Loss gradien







$$\frac{\partial L}{\partial X_{11}} = \frac{\partial L}{\partial O_{11}} * F_{11}$$

$$\frac{\partial L}{\partial X_{12}} = \frac{\partial L}{\partial O_{11}} * F_{12} + \frac{\partial L}{\partial O_{12}} * F_{11}$$

$$\frac{\partial L}{\partial X_{13}} = \frac{\partial L}{\partial O_{12}} * F_{12}$$

$$\frac{\partial L}{\partial X_{21}} = \frac{\partial L}{\partial O_{11}} * F_{21} + \frac{\partial L}{\partial O_{21}} * F_{11}$$

$$\frac{\partial L}{\partial X_{22}} = \frac{\partial L}{\partial O_{11}} * F_{22} + \frac{\partial L}{\partial O_{12}} * F_{21} + \frac{\partial L}{\partial O_{21}} * F_{12} + \frac{\partial L}{\partial O_{22}} * F_{11}$$

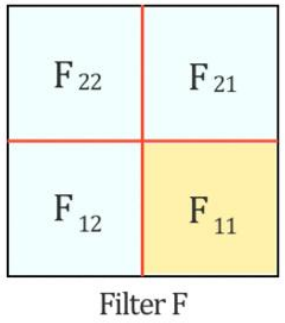
$$\frac{\partial L}{\partial X_{23}} = \frac{\partial L}{\partial O_{12}} * F_{22} + \frac{\partial L}{\partial O_{22}} * F_{12}$$

$$\frac{\partial L}{\partial X_{31}} = \frac{\partial L}{\partial O_{21}} * F_{21}$$

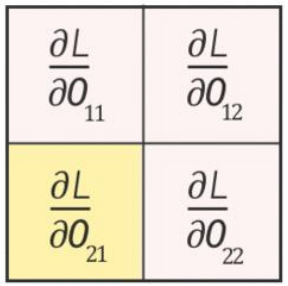
$$\frac{\partial L}{\partial X_{32}} = \frac{\partial L}{\partial O_{21}} * F_{22} + \frac{\partial L}{\partial O_{22}} * F_{21}$$

$$\frac{\partial L}{\partial X_{33}} = \frac{\partial L}{\partial O_{22}} * F_{22}$$

Back Propagation in CNN

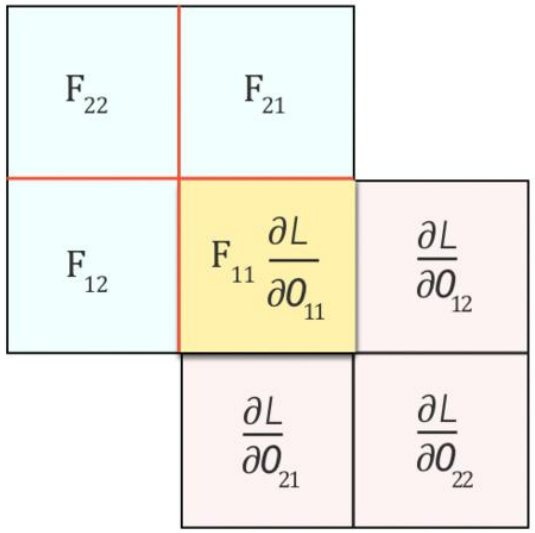


Filter F



Loss Gradient $\frac{\partial L}{\partial O}$

$$\frac{\partial L}{\partial X_{11}} = F_{11} * \frac{\partial L}{\partial O_{11}}$$



@pavisj

$$\frac{\partial L}{\partial X_{11}} = \frac{\partial L}{\partial O_{11}} * F_{11}$$

$$\frac{\partial L}{\partial X_{12}} = \frac{\partial L}{\partial O_{11}} * F_{12} + \frac{\partial L}{\partial O_{12}} * F_{11}$$

$$\frac{\partial L}{\partial X_{13}} = \frac{\partial L}{\partial O_{12}} * F_{12}$$

$$\frac{\partial L}{\partial X_{21}} = \frac{\partial L}{\partial O_{11}} * F_{21} + \frac{\partial L}{\partial O_{21}} * F_{11}$$

$$\frac{\partial L}{\partial X_{22}} = \frac{\partial L}{\partial O_{11}} * F_{22} + \frac{\partial L}{\partial O_{12}} * F_{21} + \frac{\partial L}{\partial O_{21}} * F_{12} + \frac{\partial L}{\partial O_{22}} * F_{11}$$

$$\frac{\partial L}{\partial X_{23}} = \frac{\partial L}{\partial O_{12}} * F_{22} + \frac{\partial L}{\partial O_{22}} * F_{12}$$

$$\frac{\partial L}{\partial X_{31}} = \frac{\partial L}{\partial O_{21}} * F_{21}$$

$$\frac{\partial L}{\partial X_{32}} = \frac{\partial L}{\partial O_{21}} * F_{22} + \frac{\partial L}{\partial O_{22}} * F_{21}$$

$$\frac{\partial L}{\partial X_{33}} = \frac{\partial L}{\partial O_{22}} * F_{22}$$

Back Propagation in CNN

- Loss gradient w.r.t the input:
  - ✓ We see that the loss gradient w.r.t the input is given as a full convolution between the filter and Loss gradient

$$\frac{\partial L}{\partial X} = \begin{bmatrix} \frac{\partial L}{\partial X_{11}} & \frac{\partial L}{\partial X_{12}} & \frac{\partial L}{\partial X_{13}} \\ \frac{\partial L}{\partial X_{21}} & \frac{\partial L}{\partial X_{22}} & \frac{\partial L}{\partial X_{23}} \\ \frac{\partial L}{\partial X_{31}} & \frac{\partial L}{\partial X_{32}} & \frac{\partial L}{\partial X_{33}} \end{bmatrix} = \text{Full Convolution} \left(\begin{bmatrix} F_{22} & F_{21} \\ F_{12} & F_{11} \end{bmatrix}, \begin{bmatrix} \frac{\partial L}{\partial O_{11}} & \frac{\partial L}{\partial O_{12}} \\ \frac{\partial L}{\partial O_{21}} & \frac{\partial L}{\partial O_{22}} \end{bmatrix} \right)$$

Filter F

Loss Gradient $\frac{\partial L}{\partial O}$

Padding=1

Back Propagation in CNN

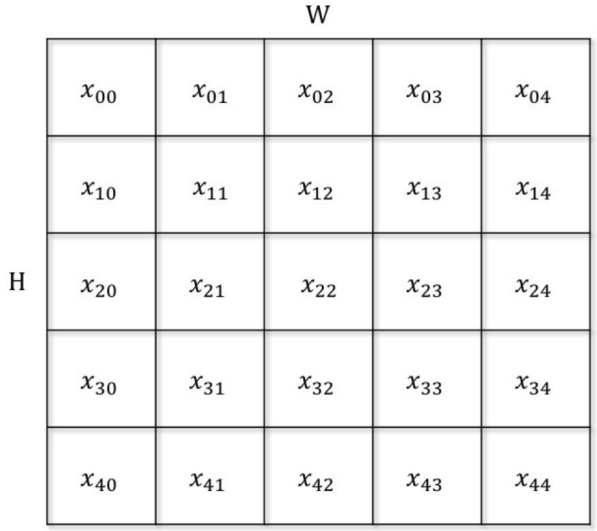
- Both the Forward pass and the Backpropagation of a Convolutional layer are Convolutions!

$$\frac{\partial L}{\partial F} = \text{Convolution}\left(\text{Input } X, \text{Loss gradient } \frac{\partial L}{\partial O}\right)$$

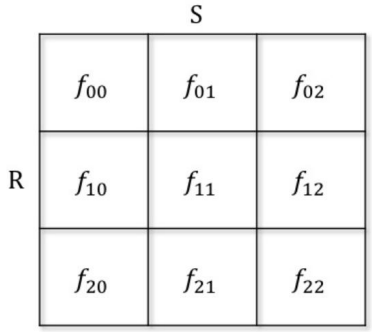
$$\frac{\partial L}{\partial X} = \text{Full Convolution}\left(180^\circ \text{rotated Filter } F, \text{Loss Gradient } \frac{\partial L}{\partial O}\right)$$

Back Propagation in CNN

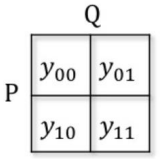
- Loss gradient w.r.t the **input**
  - ✓ Example: horizontal and vertical stride = 2



Input activations
Input channels C = 1, number of images N = 1,
Image height H = 5, width = 5



Filter (aka kernel)
Input channels C = 1, number of filters K = 1,
Filter height R = 3, width S = 3,
stride_R = stride_S = 2



Output
Output channels K = 1, number of outputs N = 1,
Output height P = 2, width Q = 2

Back Propagation in CNN

- Recap: Forward pass
  - ✓ This is how the forward pass looks like for the example:

$x_{00}f_{00}$	$x_{01}f_{01}$	$x_{02}f_{02}$	x_{03}	x_{04}
$x_{10}f_{10}$	$x_{11}f_{11}$	$x_{12}f_{12}$	x_{13}	x_{14}
$x_{20}f_{20}$	$x_{21}f_{21}$	$x_{22}f_{22}$	x_{23}	x_{24}
x_{30}	x_{31}	x_{32}	x_{33}	x_{34}
x_{40}	x_{41}	x_{42}	x_{43}	x_{44}

y_{00}	y_{01}
y_{10}	y_{11}

$$y_{00} = x_{00}f_{00} + x_{01}f_{01} + x_{02}f_{02} + x_{10}f_{10} + x_{11}f_{11} + x_{12}f_{12} + x_{20}f_{20} + x_{21}f_{21} + x_{22}f_{22}$$

Back Propagation in CNN

- Backward pass:
  - ✓ **Assumption**: we have the loss gradient w.r.t the output pixels.
  - ✓ **Requirement**: calculate the loss gradient w.r.t the input activations.

Loss gradients w.r.t input

$\frac{\partial L}{\partial x_{00}}$	$\frac{\partial L}{\partial x_{01}}$	$\frac{\partial L}{\partial x_{02}}$	$\frac{\partial L}{\partial x_{03}}$	$\frac{\partial L}{\partial x_{04}}$
$\frac{\partial L}{\partial x_{10}}$	$\frac{\partial L}{\partial x_{11}}$	$\frac{\partial L}{\partial x_{12}}$	$\frac{\partial L}{\partial x_{13}}$	$\frac{\partial L}{\partial x_{14}}$
$\frac{\partial L}{\partial x_{20}}$	$\frac{\partial L}{\partial x_{21}}$	$\frac{\partial L}{\partial x_{22}}$	$\frac{\partial L}{\partial x_{23}}$	$\frac{\partial L}{\partial x_{24}}$
$\frac{\partial L}{\partial x_{30}}$	$\frac{\partial L}{\partial x_{31}}$	$\frac{\partial L}{\partial x_{32}}$	$\frac{\partial L}{\partial x_{33}}$	$\frac{\partial L}{\partial x_{34}}$
$\frac{\partial L}{\partial x_{40}}$	$\frac{\partial L}{\partial x_{41}}$	$\frac{\partial L}{\partial x_{42}}$	$\frac{\partial L}{\partial x_{43}}$	$\frac{\partial L}{\partial x_{44}}$

Loss gradients w.r.t output

$\frac{\partial L}{\partial y_{00}}$	$\frac{\partial L}{\partial y_{01}}$
$\frac{\partial L}{\partial y_{10}}$	$\frac{\partial L}{\partial y_{11}}$

Back Propagation in CNN

- Backward pass:
  - ✓ Each input contributes to one or more outputs. The total gradient of the loss wrt to each input pixel is computed using the formula shown:
  - ✓ The gradient computation is done using chain rule and partial differentiation

$$\frac{\partial L}{\partial x_{mn}} = \Sigma_{ij} \frac{\partial L}{\partial y_{ij}} \frac{\partial y_{ij}}{\partial x_{mn}}$$

  - ✓ *i* and *j* represent the position of a single output pixel

Back Propagation in CNN

- Backward pass example:
  - ✓ Consider input x_{00} in the input shown. It contributed to the output y_{00}

x_{00}	x_{01}	x_{02}	x_{03}	x_{04}
x_{10}	x_{11}	x_{12}	x_{13}	x_{14}
x_{20}	x_{21}	x_{22}	x_{23}	x_{24}
x_{30}	x_{31}	x_{32}	x_{33}	x_{34}
x_{40}	x_{41}	x_{42}	x_{43}	x_{44}

$$\frac{\partial \mathrm{L}}{\partial x_{mn}} = \sum_{ij} \frac{\partial L}{\partial y_{ij}} \frac{\partial y_{ij}}{\partial x_{mn}}$$

y_{00}	y_{01}
y_{10}	y_{11}

Consider x_{00}. What output pixels y_{ij} does it contribute to?

$$y_{00} = x_{00}f_{00} + x_{01}f_{01} + x_{02}f_{02} + x_{10}f_{10} + x_{11}f_{11} + x_{12}f_{12} + x_{20}f_{20} + x_{21}f_{20} + x_{22}f_{22}$$

We see that x_{00} only contributes to y_{00}. Also, $\frac{\partial y_{00}}{\partial x_{00}} = f_{00}$. Thus, $\frac{\partial \mathrm{L}}{\partial x_{00}} = \frac{\partial L}{\partial y_{00}} f_{00}$

Back Propagation in CNN

- Backward pass example:
  - ✓ Input x_{01} also contributed to the output y_{00} so the loss gradient w.r.t x_{01} is computed as shown:

x_{00}	x_{01}	x_{02}	x_{03}	x_{04}
x_{10}	x_{11}	x_{12}	x_{13}	x_{14}
x_{20}	x_{21}	x_{22}	x_{23}	x_{24}
x_{30}	x_{31}	x_{32}	x_{33}	x_{34}
x_{40}	x_{41}	x_{42}	x_{43}	x_{44}

$$\frac{\partial \mathrm{L}}{\partial x_{mn}} = \sum_{ij} \frac{\partial L}{\partial y_{ij}} \frac{\partial y_{ij}}{\partial x_{mn}}$$

y_{00}	y_{01}
y_{10}	y_{11}

Next, consider x_{01}. What output pixels y_{ij} does it contribute to?

$$y_{00} = x_{00}f_{00} + x_{01}f_{01} + x_{02}f_{02} + x_{10}f_{10} + x_{11}f_{11} + x_{12}f_{12} + x_{20}f_{20} + x_{21}f_{20} + x_{22}f_{22}$$

Again, x_{01} only contributes to y_{00}. Also, $\frac{\partial y_{00}}{\partial x_{01}} = f_{01}$. Thus, $\frac{\partial \mathrm{L}}{\partial x_{01}} = \frac{\partial L}{\partial y_{00}} f_{01}$

Back Propagation in CNN

- Backward pass example:
  - ✓ Input x_{02} contributed to the output y_{00} and y_{01} so the loss gradient w.r.t x_{02} is computed as shown:

x_{00}	x_{01}	x_{02}	x_{03}	x_{04}
x_{10}	x_{11}	x_{12}	x_{13}	x_{14}
x_{20}	x_{21}	x_{22}	x_{23}	x_{24}
x_{30}	x_{31}	x_{32}	x_{33}	x_{34}
x_{40}	x_{41}	x_{42}	x_{43}	x_{44}

y_{00}	y_{01}
y_{10}	y_{11}

$$\frac{\partial \mathrm{L}}{\partial x_{mn}} = \sum_{ij} \frac{\partial L}{\partial y_{ij}} \frac{\partial y_{ij}}{\partial x_{mn}}$$

Next, consider x_{02}. It contributes to y_{00} and y_{01}.

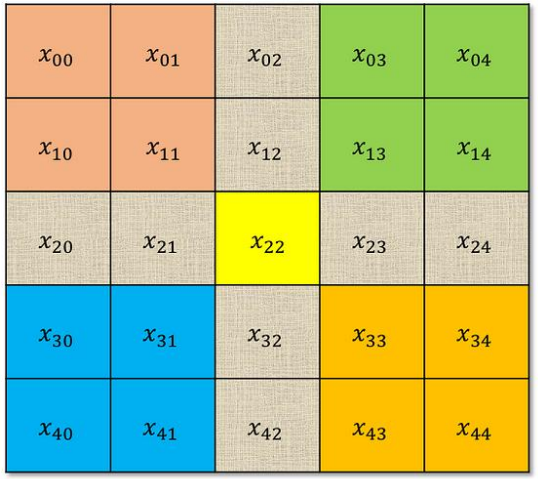
$$y_{00} = x_{00}f_{00} + x_{01}f_{01} + x_{02}f_{02} + x_{10}f_{10} + x_{11}f_{11} + x_{12}f_{12} + x_{20}f_{20} + x_{21}f_{20} + x_{22}f_{22}$$

$$y_{01} = x_{02}f_{00} + x_{03}f_{01} + x_{04}f_{02} + x_{12}f_{10} + x_{13}f_{11} + x_{14}f_{12} + x_{22}f_{20} + x_{23}f_{21} + x_{24}f_{22}$$

$$\text{Thus, } \frac{\partial \mathrm{L}}{\partial x_{02}} = \frac{\partial L}{\partial y_{00}} f_{02} + \frac{\partial L}{\partial y_{01}} f_{00}$$

Back Propagation in CNN

- Backward pass example:
  - ✓ Input x_{22} contributed to the output y_{00}, y_{01}, y_{10}, and y_{11} so the loss gradient w.r.t x_{22} is computed as shown:



$$\frac{\partial \mathrm{L}}{\partial x_{mn}} = \sum_{ij} \frac{\partial L}{\partial y_{ij}} \frac{\partial y_{ij}}{\partial x_{mn}}$$

Finally, consider x_{22}. It contributes to all outputs: y_{00} , y_{01}, y_{10}, and y_{11}

$$y_{00} = x_{00}f_{00} + x_{01}f_{01} + x_{02}f_{02} + x_{10}f_{10} + x_{11}f_{11} + x_{12}f_{12} + x_{20}f_{20} + x_{21}f_{20} + x_{22}f_{22}$$
$$y_{01} = x_{02}f_{00} + x_{03}f_{01} + x_{04}f_{02} + x_{12}f_{10} + x_{13}f_{11} + x_{14}f_{12} + x_{22}f_{20} + x_{23}f_{21} + x_{24}f_{22}$$
$$y_{10} = x_{20}f_{00} + x_{21}f_{01} + x_{22}f_{02} + x_{30}f_{10} + x_{31}f_{11} + x_{32}f_{12} + x_{40}f_{20} + x_{41}f_{20} + x_{42}f_{22}$$
$$y_{11} = x_{22}f_{00} + x_{23}f_{01} + x_{24}f_{02} + x_{32}f_{10} + x_{33}f_{11} + x_{34}f_{12} + x_{42}f_{20} + x_{43}f_{20} + x_{44}f_{22}$$

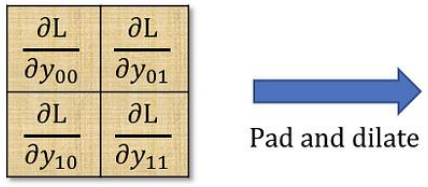
$$\text{Thus, } \frac{\partial \mathrm{L}}{\partial x_{22}} = \frac{\partial L}{\partial y_{00}} f_{22} + \frac{\partial L}{\partial y_{01}} f_{20} + \frac{\partial L}{\partial y_{10}} f_{02} + \frac{\partial L}{\partial y_{11}} f_{00}$$

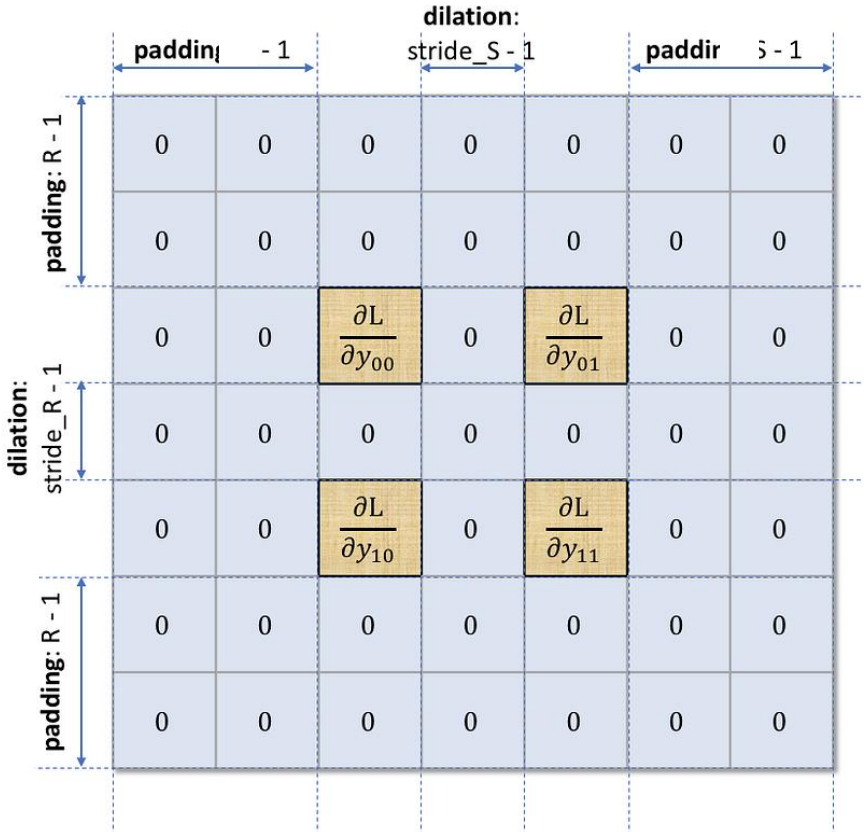
Back Propagation in CNN

- Backward pass example:
  - ✓ To visualize the pattern more clearly, we pad the gradient tensor with zeros at the top and bottom as well as to the left and right
  - ✓ The **number of zeros padded on either side is equal to the stride** (horizontal and vertical)
  - ✓ We also dilate the output gradient pixels with the stride–vertically and horizontally

Back Propagation in CNN

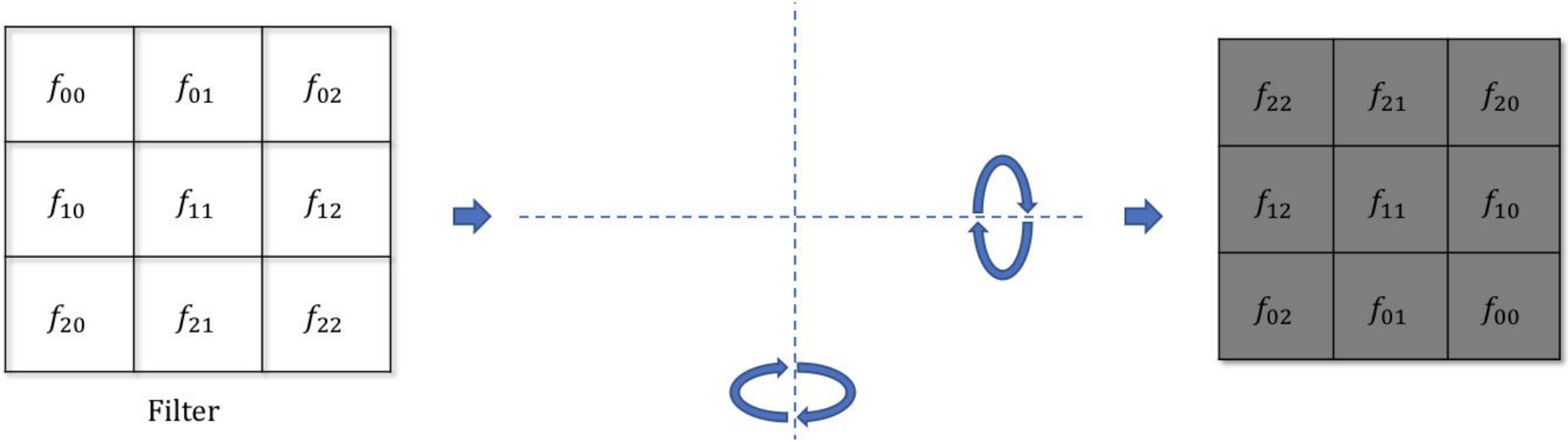
- Backward pass example:
  - ✓ To visualize the pattern more clearly, we pad the gradient tensor with zeros at the top and bottom as well as to the left and right





Back Propagation in CNN

- Backward pass example:
  - ✓ We also rotate the filter vertically and horizontally as shown:



Back Propagation in CNN

- Backward pass example:
  - ✓ After these modifications, we can now see the calculation of the gradient tensor as follows:

$$\frac{\partial L}{\partial x_{00}} = \frac{\partial L}{\partial y_{00}} f_{00}$$

$\frac{\partial L}{\partial x_{00}}$	$\frac{\partial L}{\partial x_{01}}$	$\frac{\partial L}{\partial x_{02}}$	$\frac{\partial L}{\partial x_{03}}$	$\frac{\partial L}{\partial x_{04}}$
$\frac{\partial L}{\partial x_{10}}$	$\frac{\partial L}{\partial x_{11}}$	$\frac{\partial L}{\partial x_{12}}$	$\frac{\partial L}{\partial x_{13}}$	$\frac{\partial L}{\partial x_{14}}$
$\frac{\partial L}{\partial x_{20}}$	$\frac{\partial L}{\partial x_{21}}$	$\frac{\partial L}{\partial x_{22}}$	$\frac{\partial L}{\partial x_{23}}$	$\frac{\partial L}{\partial x_{24}}$
$\frac{\partial L}{\partial x_{30}}$	$\frac{\partial L}{\partial x_{31}}$	$\frac{\partial L}{\partial x_{32}}$	$\frac{\partial L}{\partial x_{33}}$	$\frac{\partial L}{\partial x_{34}}$
$\frac{\partial L}{\partial x_{40}}$	$\frac{\partial L}{\partial x_{41}}$	$\frac{\partial L}{\partial x_{42}}$	$\frac{\partial L}{\partial x_{43}}$	$\frac{\partial L}{\partial x_{44}}$

=

$0 * f_{22}$	$0 * f_{21}$	$0 * f_{20}$	0	0	0	0
$0 * f_{12}$	$0 * f_{11}$	$0 * f_{10}$	0	0	0	0
$0 * f_{02}$	$0 * f_{01}$	$\frac{\partial L}{\partial y_{00}} f_{00}$	0	$\frac{\partial L}{\partial y_{01}}$	0	0
0	0	0	0	0	0	0
0	0	$\frac{\partial L}{\partial y_{10}}$	0	$\frac{\partial L}{\partial y_{11}}$	0	0
0	0	0	0	0	0	0
0	0	0	0	0	0	0

$$y_{00} = x_{00}f_{00} + x_{01}f_{01} + x_{02}f_{02} + x_{10}f_{10} + x_{11}f_{11} + x_{12}f_{12} + x_{20}f_{20} + x_{21}f_{21} + x_{22}f_{22}$$

$$y_{01} = x_{02}f_{00} + x_{03}f_{01} + x_{04}f_{02} + x_{12}f_{10} + x_{13}f_{11} + x_{14}f_{12} + x_{22}f_{20} + x_{23}f_{21} + x_{24}f_{22}$$

$$y_{10} = x_{20}f_{00} + x_{21}f_{01} + x_{22}f_{02} + x_{30}f_{10} + x_{31}f_{11} + x_{32}f_{12} + x_{40}f_{20} + x_{41}f_{21} + x_{42}f_{22}$$

$$y_{11} = x_{22}f_{00} + x_{23}f_{01} + x_{24}f_{02} + x_{32}f_{10} + x_{33}f_{11} + x_{34}f_{12} + x_{42}f_{20} + x_{43}f_{21} + x_{44}f_{22}$$

Back Propagation in CNN

- Backward pass example:
  - ✓ After these modifications, we can now see the calculation of the gradient tensor as follows:

$$\frac{\partial \mathrm{L}}{\partial x_{00}} = \frac{\partial L}{\partial y_{00}} f_{00}$$

$\frac{\partial \mathrm{L}}{\partial x_{00}}$	$\frac{\partial \mathrm{L}}{\partial x_{01}}$	$\frac{\partial \mathrm{L}}{\partial x_{02}}$	$\frac{\partial \mathrm{L}}{\partial x_{03}}$	$\frac{\partial \mathrm{L}}{\partial x_{04}}$
$\frac{\partial \mathrm{L}}{\partial x_{10}}$	$\frac{\partial \mathrm{L}}{\partial x_{11}}$	$\frac{\partial \mathrm{L}}{\partial x_{12}}$	$\frac{\partial \mathrm{L}}{\partial x_{13}}$	$\frac{\partial \mathrm{L}}{\partial x_{14}}$
$\frac{\partial \mathrm{L}}{\partial x_{20}}$	$\frac{\partial \mathrm{L}}{\partial x_{21}}$	$\frac{\partial \mathrm{L}}{\partial x_{22}}$	$\frac{\partial \mathrm{L}}{\partial x_{23}}$	$\frac{\partial \mathrm{L}}{\partial x_{24}}$
$\frac{\partial \mathrm{L}}{\partial x_{30}}$	$\frac{\partial \mathrm{L}}{\partial x_{31}}$	$\frac{\partial \mathrm{L}}{\partial x_{32}}$	$\frac{\partial \mathrm{L}}{\partial x_{33}}$	$\frac{\partial \mathrm{L}}{\partial x_{34}}$
$\frac{\partial \mathrm{L}}{\partial x_{40}}$	$\frac{\partial \mathrm{L}}{\partial x_{41}}$	$\frac{\partial \mathrm{L}}{\partial x_{42}}$	$\frac{\partial \mathrm{L}}{\partial x_{43}}$	$\frac{\partial \mathrm{L}}{\partial x_{44}}$

=

$0 * f_{22}$	$0 * f_{21}$	$0 * f_{20}$	0	0	0	0
$0 * f_{12}$	$0 * f_{11}$	$0 * f_{10}$	0	0	0	0
$0 * f_{02}$	$0 * f_{01}$	$\frac{\partial \mathrm{L}}{\partial y_{00}} f_{00}$	0	$\frac{\partial \mathrm{L}}{\partial y_{01}}$	0	0
0	0	0	0	0	0	0
0	0	$\frac{\partial \mathrm{L}}{\partial y_{10}}$	0	$\frac{\partial \mathrm{L}}{\partial y_{11}}$	0	0
0	0	0	0	0	0	0
0	0	0	0	0	0	0

$$y_{00} = x_{00}f_{00} + x_{01}f_{01} + x_{02}f_{02} + x_{10}f_{10} + x_{11}f_{11} + x_{12}f_{12} + x_{20}f_{20} + x_{21}f_{21} + x_{22}f_{22}$$

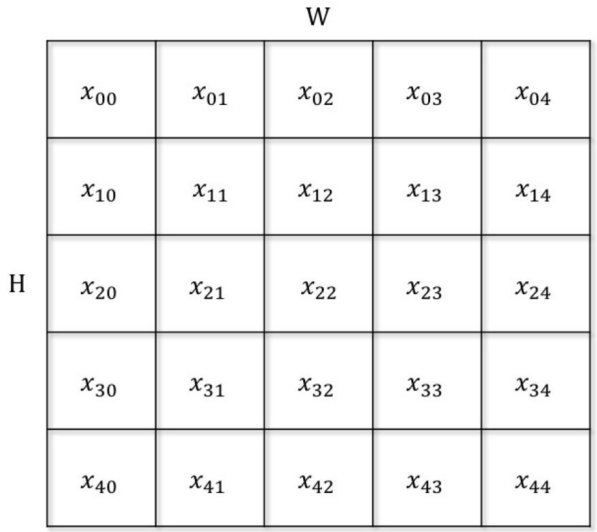
$$y_{01} = x_{02}f_{00} + x_{03}f_{01} + x_{04}f_{02} + x_{12}f_{10} + x_{13}f_{11} + x_{14}f_{12} + x_{22}f_{20} + x_{23}f_{21} + x_{24}f_{22}$$

$$y_{10} = x_{20}f_{00} + x_{21}f_{01} + x_{22}f_{02} + x_{30}f_{10} + x_{31}f_{11} + x_{32}f_{12} + x_{40}f_{20} + x_{41}f_{21} + x_{42}f_{22}$$

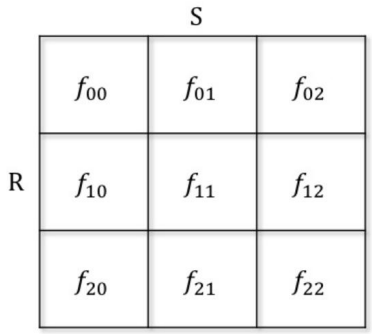
$$y_{11} = x_{22}f_{00} + x_{23}f_{01} + x_{24}f_{02} + x_{32}f_{10} + x_{33}f_{11} + x_{34}f_{12} + x_{42}f_{20} + x_{43}f_{21} + x_{44}f_{22}$$

Back Propagation in CNN

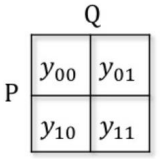
- Loss gradient w.r.t the **Filter**
  - ✓ Example: horizontal and vertical stride = 2



Input activations
Input channels C = 1, number of images N = 1,
Image height H = 5, width = 5



Filter (aka kernel)
Input channels C = 1, number of filters K = 1,
Filter height R = 3, width S = 3,
stride_R = stride_S = 2



Output
Output channels K = 1, number of outputs N = 1,
Output height P = 2, width Q = 2

Back Propagation in CNN

- Backward pass:
  - ✓ **Assumption**: we have the loss gradient w.r.t the output pixels.
  - ✓ **Requirement**: calculate the loss gradient w.r.t the filter.

Loss gradients w.r.t filter

$\frac{\partial L}{\partial f_{00}}$	$\frac{\partial L}{\partial f_{01}}$	$\frac{\partial L}{\partial f_{02}}$
$\frac{\partial L}{\partial f_{10}}$	$\frac{\partial L}{\partial f_{11}}$	$\frac{\partial L}{\partial f_{12}}$
$\frac{\partial L}{\partial f_{20}}$	$\frac{\partial L}{\partial f_{21}}$	$\frac{\partial L}{\partial f_{22}}$

Loss gradients w.r.t output

$\frac{\partial L}{\partial y_{00}}$	$\frac{\partial L}{\partial y_{01}}$
$\frac{\partial L}{\partial y_{10}}$	$\frac{\partial L}{\partial y_{11}}$

Back Propagation in CNN

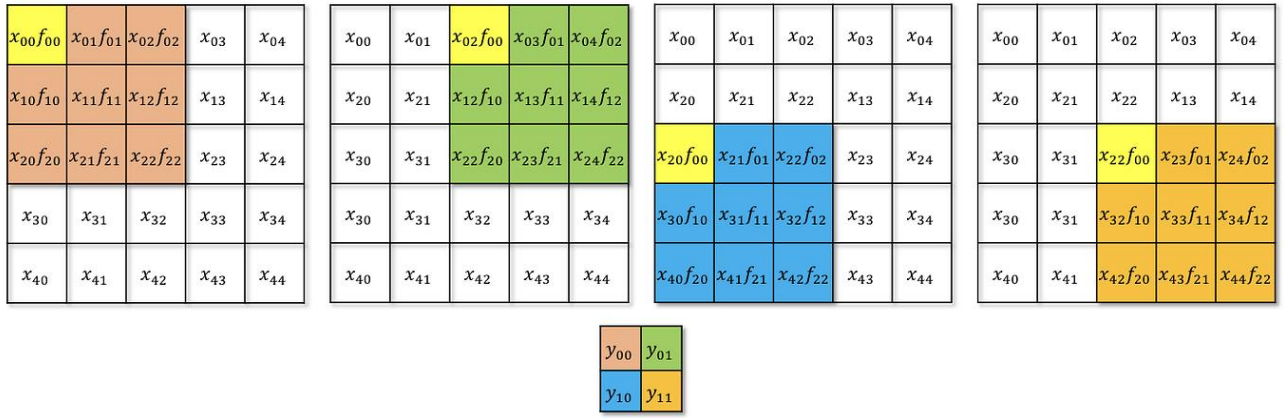
- Backward pass:
  - ✓ Unlike the inputs which contribute to some outputs, each filter contributes to all outputs:
  - ✓ The gradient computation is done using chain rule and partial differentiation

$$\frac{\partial L}{\partial f_{mn}} = \Sigma_{ij} \frac{\partial L}{\partial y_{ij}} \frac{\partial y_{ij}}{\partial f_{mn}}$$

  - ✓ *i* and *j* represent the position of a single output pixel

Back Propagation in CNN

- Backward pass example:
  - ✓ Considering the filter f_{00}, the loss gradient is computed as shown:
  - ✓ Notice the inputs involved in the computation



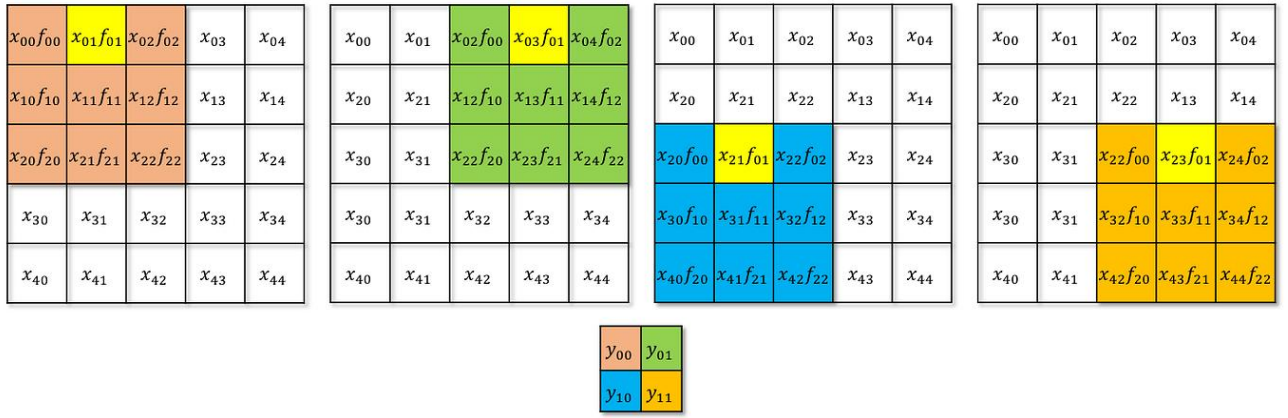
First, consider f_{00}. It contributes to all outputs: y_{00} , y_{01}, y_{10}, and y_{11}

$$y_{00} = x_{00}f_{00} + x_{01}f_{01} + x_{02}f_{02} + x_{10}f_{10} + x_{11}f_{11} + x_{12}f_{12} + x_{20}f_{20} + x_{21}f_{20} + x_{22}f_{22}$$
$$y_{01} = x_{02}f_{00} + x_{03}f_{01} + x_{04}f_{02} + x_{12}f_{10} + x_{13}f_{11} + x_{14}f_{12} + x_{22}f_{20} + x_{23}f_{21} + x_{24}f_{22}$$
$$y_{10} = x_{20}f_{00} + x_{21}f_{01} + x_{22}f_{02} + x_{30}f_{10} + x_{31}f_{11} + x_{32}f_{12} + x_{40}f_{20} + x_{41}f_{20} + x_{42}f_{22}$$
$$y_{11} = x_{22}f_{00} + x_{23}f_{01} + x_{24}f_{02} + x_{32}f_{10} + x_{33}f_{11} + x_{34}f_{12} + x_{42}f_{20} + x_{43}f_{20} + x_{44}f_{22}$$

$$\frac{\partial \mathrm{L}}{\partial f_{mn}} = \sum_{ij} \frac{\partial L}{\partial y_{ij}} \frac{\partial y_{ij}}{\partial f_{mn}}. \text{ Thus, } \frac{\partial \mathrm{L}}{\partial f_{00}} = \frac{\partial L}{\partial y_{00}} x_{00} + \frac{\partial L}{\partial y_{01}} x_{02} + \frac{\partial L}{\partial y_{10}} x_{20} + \frac{\partial L}{\partial y_{11}} x_{22}$$

Back Propagation in CNN

- Backward pass example:
  - ✓ Considering the filter f_{22}, the loss gradient is computed as shown:
  - ✓ Notice the inputs involved in the computation



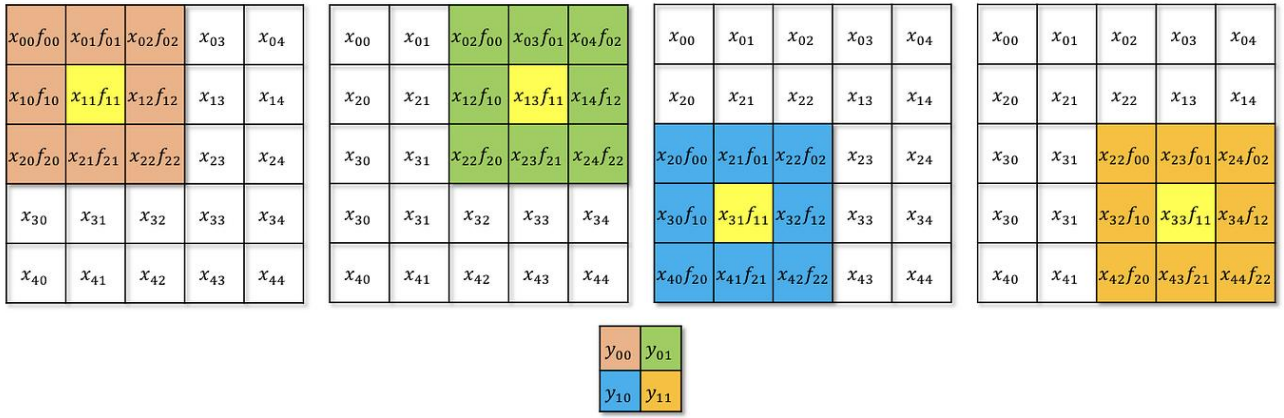
Next, consider f_{01}. It contributes to all outputs: y_{00} , y_{01}, y_{10}, and y_{11}

$$y_{00} = x_{00}f_{00} + x_{01}f_{01} + x_{02}f_{02} + x_{10}f_{10} + x_{11}f_{11} + x_{12}f_{12} + x_{20}f_{20} + x_{21}f_{20} + x_{22}f_{22}$$
$$y_{01} = x_{02}f_{00} + x_{03}f_{01} + x_{04}f_{02} + x_{12}f_{10} + x_{13}f_{11} + x_{14}f_{12} + x_{22}f_{20} + x_{23}f_{21} + x_{24}f_{22}$$
$$y_{10} = x_{20}f_{00} + x_{21}f_{01} + x_{22}f_{02} + x_{30}f_{10} + x_{31}f_{11} + x_{32}f_{12} + x_{40}f_{20} + x_{41}f_{20} + x_{42}f_{22}$$
$$y_{11} = x_{22}f_{00} + x_{23}f_{01} + x_{24}f_{02} + x_{32}f_{10} + x_{33}f_{11} + x_{34}f_{12} + x_{42}f_{20} + x_{43}f_{20} + x_{44}f_{22}$$

$$\frac{\partial \mathrm{L}}{\partial f_{mn}} = \sum_{ij} \frac{\partial L}{\partial y_{ij}} \frac{\partial y_{ij}}{\partial f_{mn}}. \text{ Thus, } \frac{\partial \mathrm{L}}{\partial f_{01}} = \frac{\partial L}{\partial y_{00}} x_{01} + \frac{\partial L}{\partial y_{01}} x_{03} + \frac{\partial L}{\partial y_{10}} x_{21} + \frac{\partial L}{\partial y_{11}} x_{23}$$

Back Propagation in CNN

- Backward pass example:
  - ✓ Considering the filter f_{22}, the loss gradient is computed as shown:
  - ✓ Notice the inputs involved in the computation



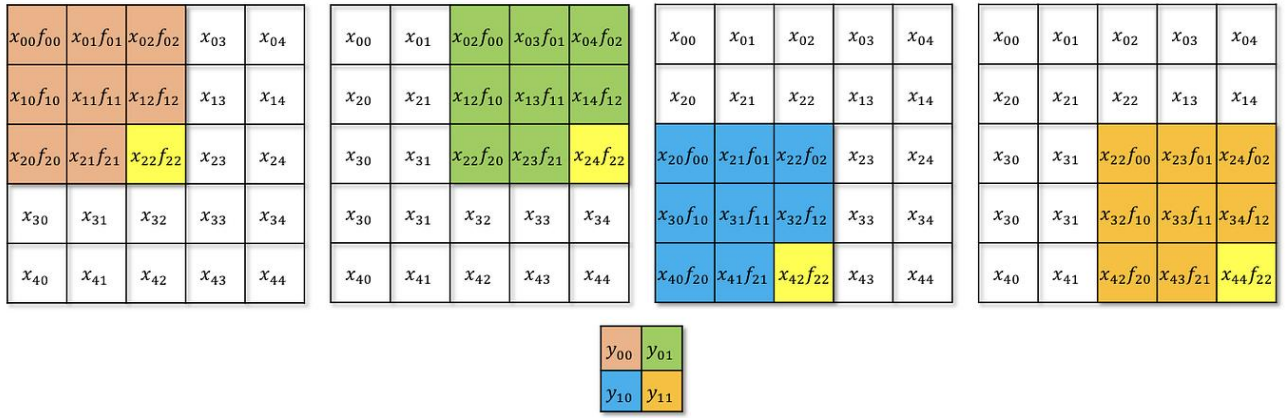
Next, consider f_{11}. It contributes to all outputs: y_{00} , y_{01}, y_{10}, and y_{11}

$$y_{00} = x_{00}f_{00} + x_{01}f_{01} + x_{02}f_{02} + x_{10}f_{10} + x_{11}f_{11} + x_{12}f_{12} + x_{20}f_{20} + x_{21}f_{20} + x_{22}f_{22}$$
$$y_{01} = x_{02}f_{00} + x_{03}f_{01} + x_{04}f_{02} + x_{12}f_{10} + x_{13}f_{11} + x_{14}f_{12} + x_{22}f_{20} + x_{23}f_{21} + x_{24}f_{22}$$
$$y_{10} = x_{20}f_{00} + x_{21}f_{01} + x_{22}f_{02} + x_{30}f_{10} + x_{31}f_{11} + x_{32}f_{12} + x_{40}f_{20} + x_{41}f_{20} + x_{42}f_{22}$$
$$y_{11} = x_{22}f_{00} + x_{23}f_{01} + x_{24}f_{02} + x_{32}f_{10} + x_{33}f_{11} + x_{34}f_{12} + x_{42}f_{20} + x_{43}f_{20} + x_{44}f_{22}$$

$$\frac{\partial \mathrm{L}}{\partial f_{mn}} = \sum_{ij} \frac{\partial L}{\partial y_{ij}} \frac{\partial y_{ij}}{\partial f_{mn}}. \text{ Thus, } \frac{\partial \mathrm{L}}{\partial f_{11}} = \frac{\partial L}{\partial y_{00}} x_{11} + \frac{\partial L}{\partial y_{01}} x_{13} + \frac{\partial L}{\partial y_{10}} x_{31} + \frac{\partial L}{\partial y_{11}} x_{33}$$

Back Propagation in CNN

- Backward pass example:
  - ✓ Considering the filter f_{22}, the loss gradient is computed as shown:
  - ✓ Notice the inputs involved in the computation



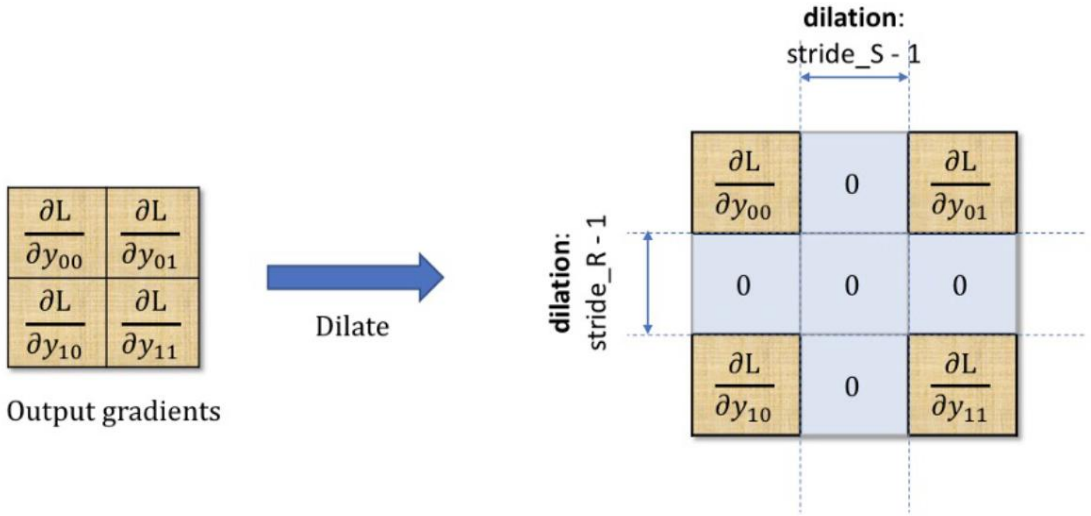
Finally, consider f_{22}. It contributes to all outputs: y_{00} , y_{01}, y_{10}, and y_{11}

$$y_{00} = x_{00}f_{00} + x_{01}f_{01} + x_{02}f_{02} + x_{10}f_{10} + x_{11}f_{11} + x_{12}f_{12} + x_{20}f_{20} + x_{21}f_{20} + x_{22}f_{22}$$
$$y_{01} = x_{02}f_{00} + x_{03}f_{01} + x_{04}f_{02} + x_{12}f_{10} + x_{13}f_{11} + x_{14}f_{12} + x_{22}f_{20} + x_{23}f_{21} + x_{24}f_{22}$$
$$y_{10} = x_{20}f_{00} + x_{21}f_{01} + x_{22}f_{02} + x_{30}f_{10} + x_{31}f_{11} + x_{32}f_{12} + x_{40}f_{20} + x_{41}f_{20} + x_{42}f_{22}$$
$$y_{11} = x_{22}f_{00} + x_{23}f_{01} + x_{24}f_{02} + x_{32}f_{10} + x_{33}f_{11} + x_{34}f_{12} + x_{42}f_{20} + x_{43}f_{20} + x_{44}f_{22}$$

$$\frac{\partial \mathrm{L}}{\partial f_{mn}} = \sum_{ij} \frac{\partial L}{\partial y_{ij}} \frac{\partial y_{ij}}{\partial f_{mn}} \text{. Thus, } \frac{\partial \mathrm{L}}{\partial f_{22}} = \frac{\partial L}{\partial y_{00}} x_{22} + \frac{\partial L}{\partial y_{01}} x_{24} + \frac{\partial L}{\partial y_{10}} x_{42} + \frac{\partial L}{\partial y_{11}} x_{44}$$

Back Propagation in CNN

- Backward pass example:
  - ✓ To visualize the underlying pattern, we will modify the output gradient tensor by dilating the pixels with the stride vertically and horizontally:



Back Propagation in CNN

- Backward pass example:
  - ✓ After these modifications, we can now see the calculation of the filter gradient tensor as follows:

$$\frac{\partial \mathrm{L}}{\partial f_{00}} = \frac{\partial L}{\partial y_{00}} x_{00} + \frac{\partial L}{\partial y_{01}} x_{02} + \frac{\partial L}{\partial y_{10}} x_{20} + \frac{\partial L}{\partial y_{11}} x_{22}$$

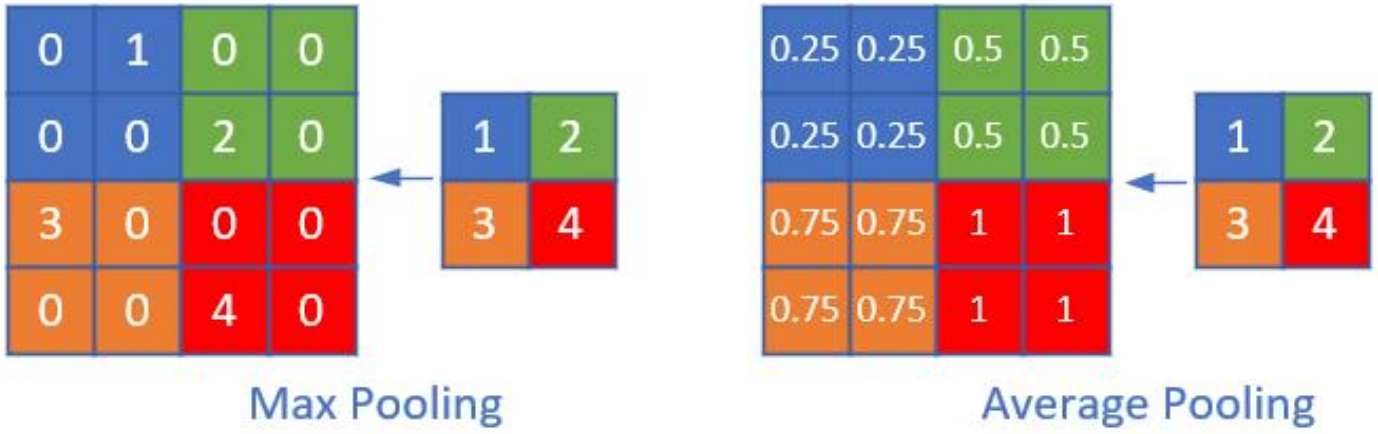
$\frac{\partial \mathrm{L}}{\partial f_{00}}$	$\frac{\partial \mathrm{L}}{\partial f_{01}}$	$\frac{\partial \mathrm{L}}{\partial f_{02}}$
$\frac{\partial \mathrm{L}}{\partial f_{10}}$	$\frac{\partial \mathrm{L}}{\partial f_{11}}$	$\frac{\partial \mathrm{L}}{\partial f_{12}}$
$\frac{\partial \mathrm{L}}{\partial f_{20}}$	$\frac{\partial \mathrm{L}}{\partial f_{21}}$	$\frac{\partial \mathrm{L}}{\partial f_{22}}$

=

$\frac{\partial \mathrm{L}}{\partial y_{00}} x_{00}$	$0 * x_{01}$	$\frac{\partial \mathrm{L}}{\partial y_{01}} x_{02}$	x_{03}	x_{04}
$0 * x_{10}$	$0 * x_{11}$	$0 * x_{12}$	x_{13}	x_{14}
$\frac{\partial \mathrm{L}}{\partial y_{10}} x_{20}$	$0 * x_{21}$	$\frac{\partial \mathrm{L}}{\partial y_{11}} x_{22}$	x_{23}	x_{24}
x_{30}	x_{31}	x_{32}	x_{33}	x_{34}
x_{40}	x_{41}	x_{42}	x_{43}	x_{44}

Back Propagation in Pooling

- Pooling layers have no learnable parameters, so no weights need updated.
- Max pooling: the forward pass stores the argmax index; the backward pass sends the upstream gradient only to the max location and zeros elsewhere.
- Average pooling: the upstream gradient is distributed equally to all elements in the pooling window.

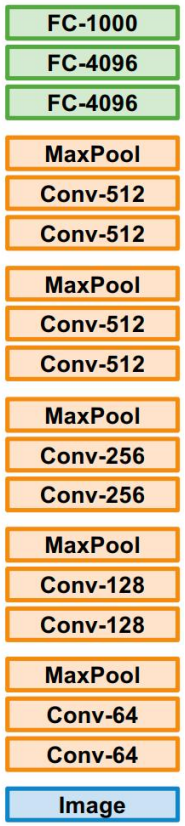


Training CNNs

What if you don’t have a lot of data?
Can you still train CNNs?

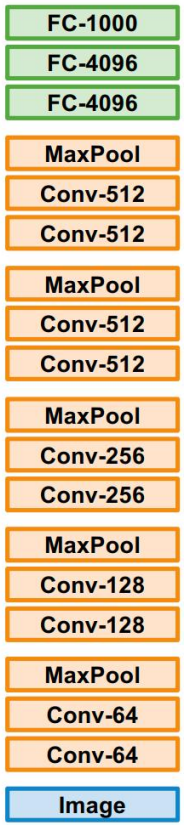
Transfer Learning with CNNs

1. Train on Imagenet (or internet scale data)

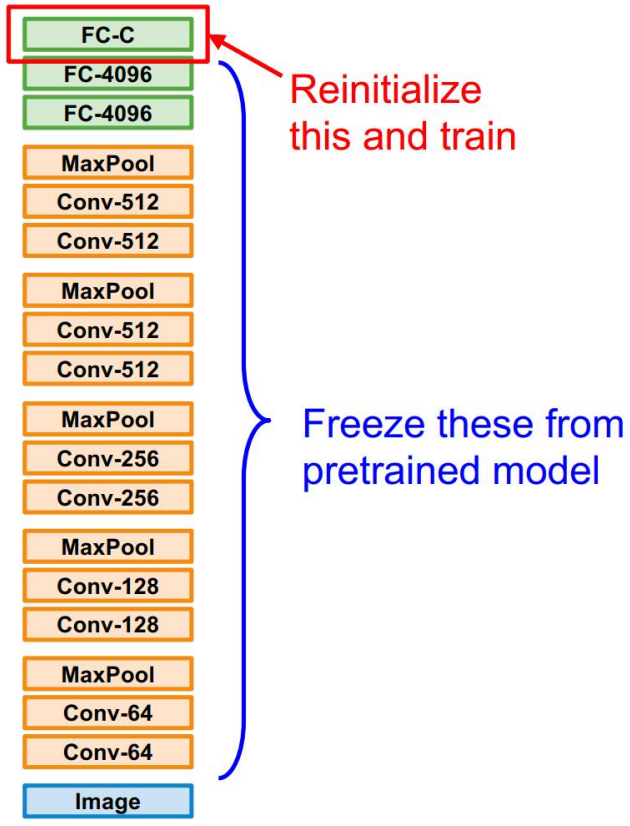


Transfer Learning with CNNs

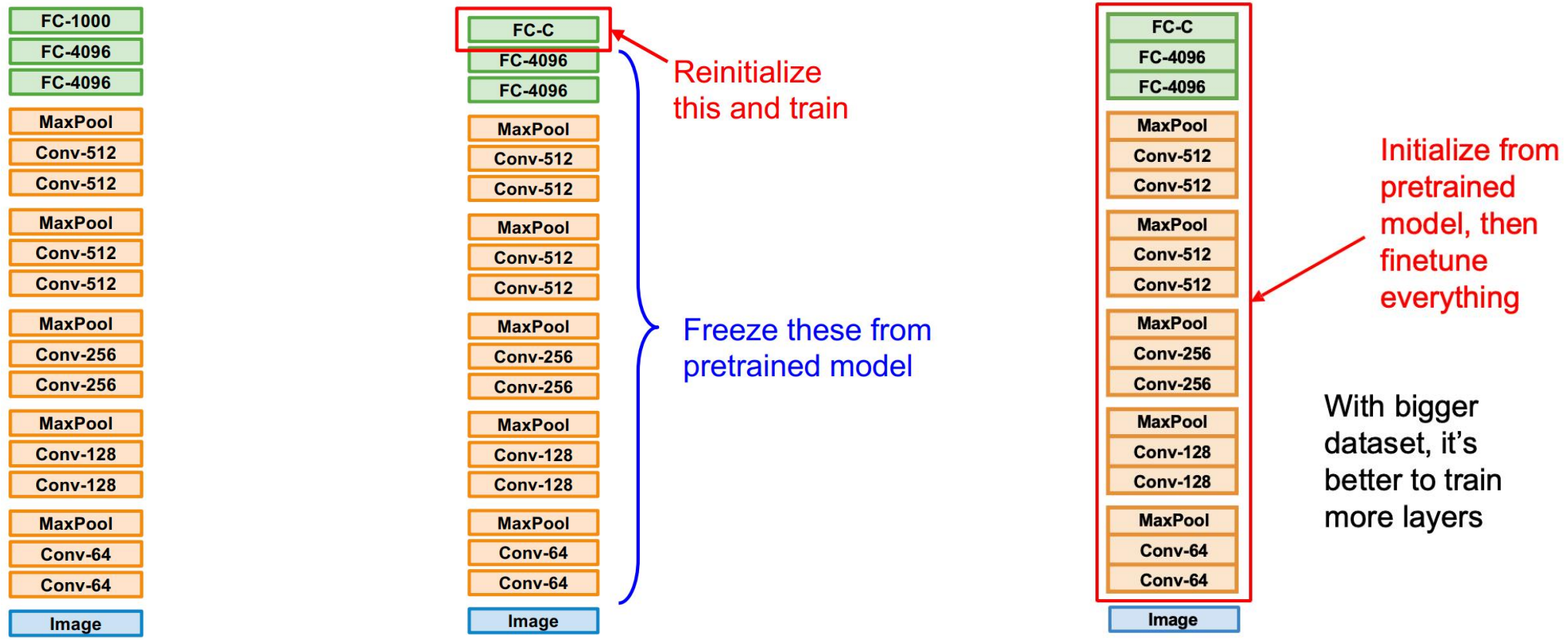
1. Train on Imagenet



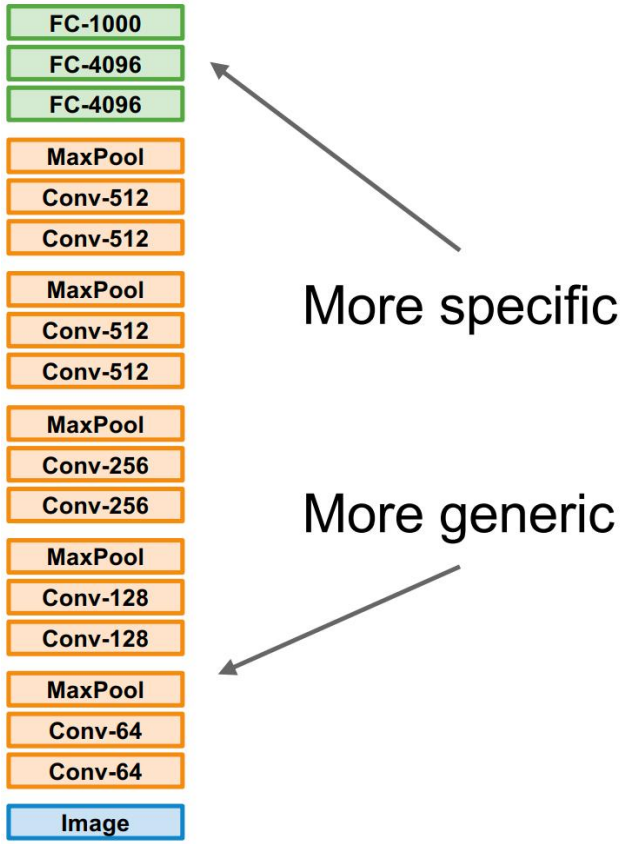
2. Small Dataset (C classes)



Transfer Learning with CNNs

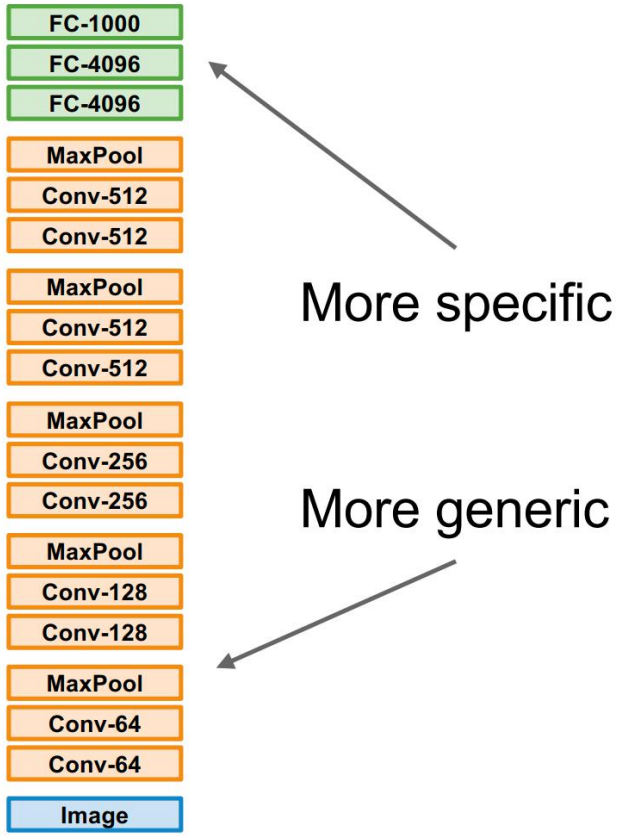


Transfer Learning with CNNs



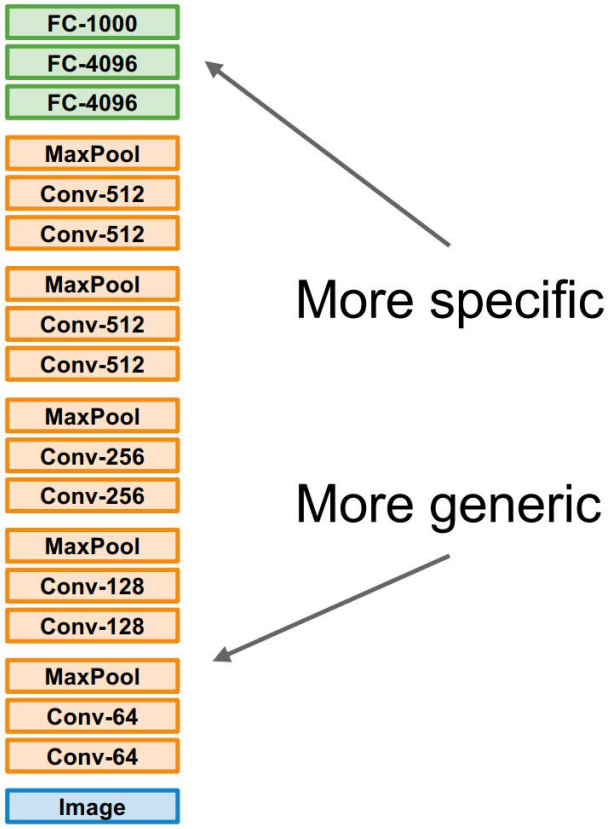
	very similar dataset	very different dataset
very little data	?	?
quite a lot of data	?	?

Transfer Learning with CNNs



	very similar dataset	very different dataset
very little data	Use Linear Classifier on final layer	?
quite a lot of data	Finetune all model layers	?

Transfer Learning with CNNs



	very similar dataset	very different dataset
very little data	Use Linear Classifier on final layer	Try another model or collect more data ☹
quite a lot of data	Finetune all model layers	Either finetune all model layers or train from scratch!

Transfer Learning with CNNs

Takeaway for your projects and beyond: Have some dataset of interest but it has < ~1M images?

1. Find a very large dataset that has similar data, train a big model there
2. Transfer learn to your dataset

Deep learning frameworks provide a “Model Zoo” of pretrained models so you don’t need to train your own

PyTorch: https://github.com/pytorch/vision

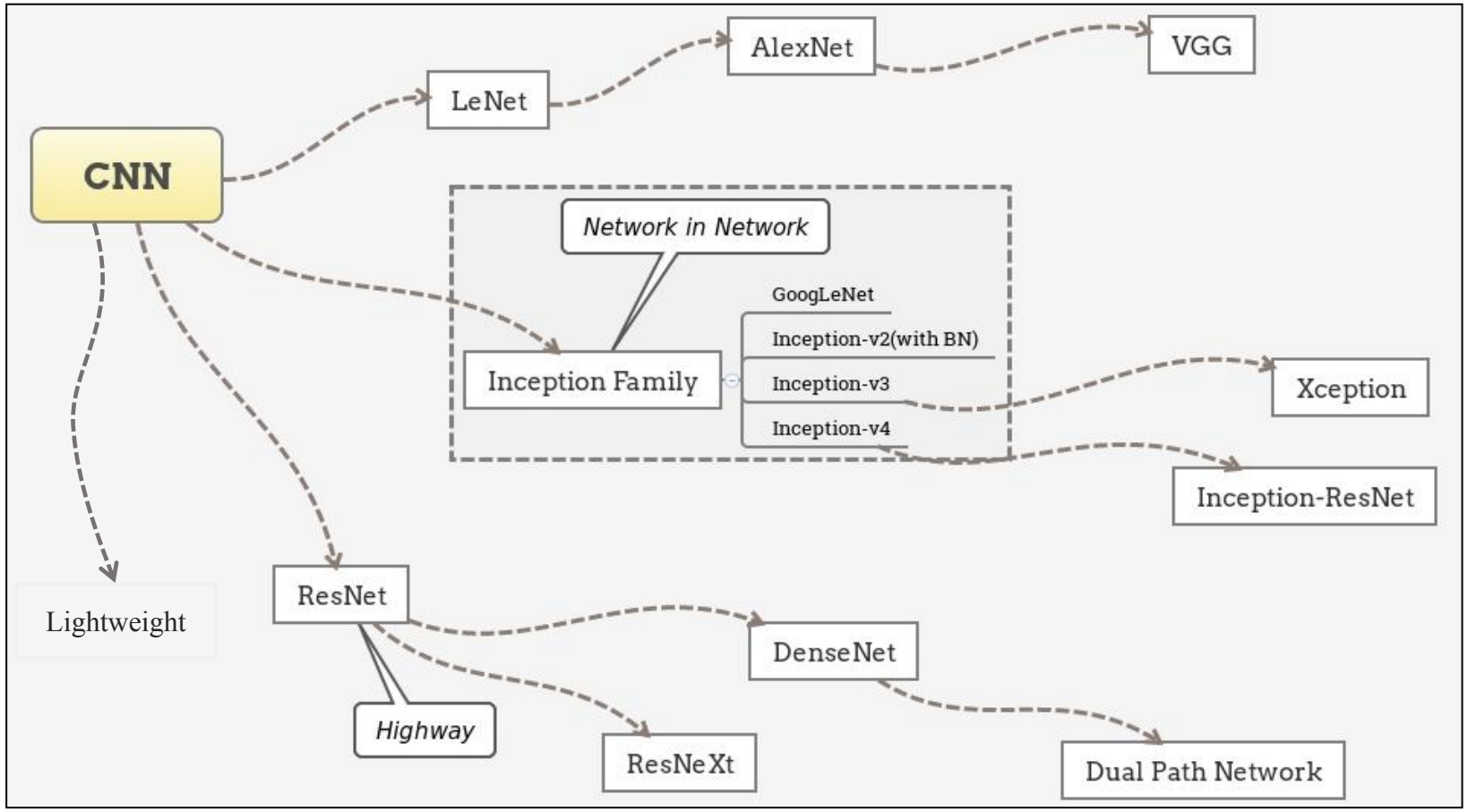
Huggingface: https://github.com/huggingface/pytorch-image-models

上海交通大学
SHANGHAI JIAO TONG UNIVERSITY

Contents

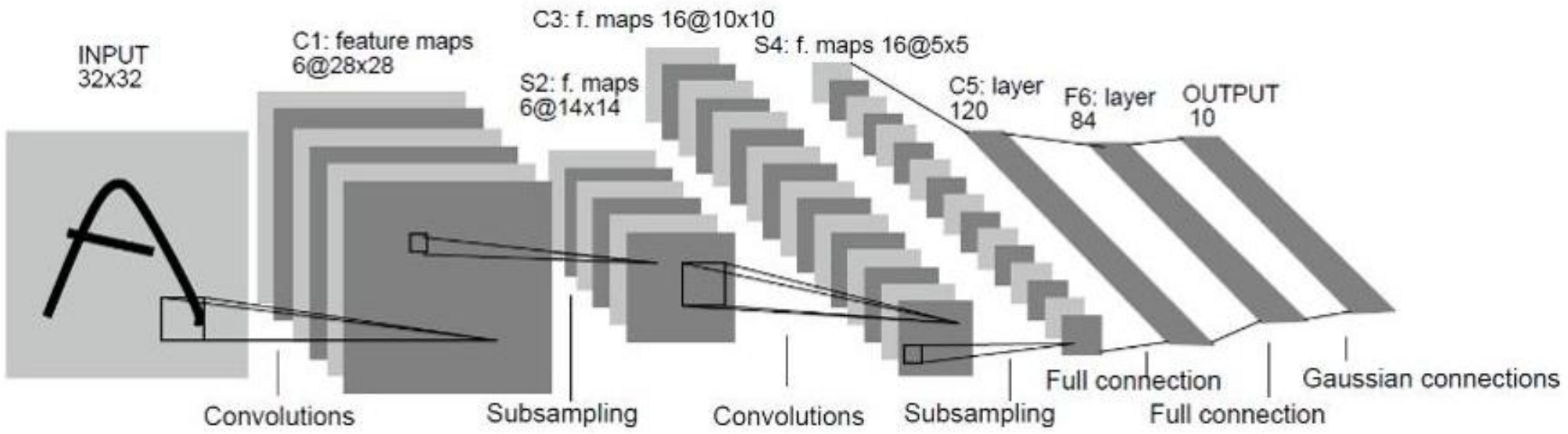
- Convolutional Neural Network
- Training CNNs
- **Classical CNN Architectures**

Preview


CNN
LeNet
AlexNet
VGG
Network in Network
Inception Family
GoogLeNet
Inception-v2(with BN)
Inception-v3
Inception-v4
Xception
Inception-ResNet
Lightweight
ResNet
Highway
ResNeXt
DenseNet
Dual Path Network

LeNet

- LeNet mainly consists of 3 convolutional layers, 2 pooling layers, and 1 fully connected layer

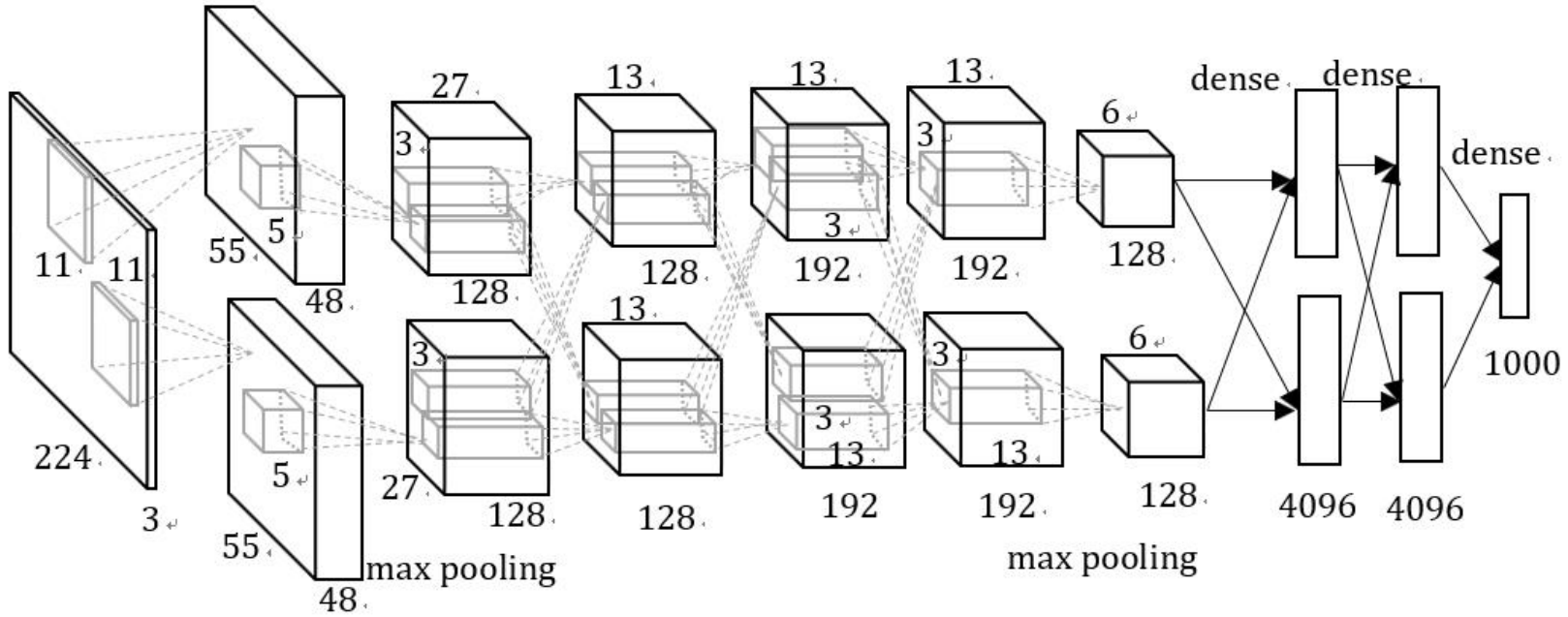


LeNet

- It possesses basic components such as convolution, activation, pooling, and fully connected layers, forming the basic framework of CNN.
- GPUs did not exist at that time, and CPUs had extremely low performance.
- LeNet is only used in simple scenarios such as handwriting recognition

AlexNet

- AlexNet, designed by Hinton and his student Alex Krizhevsky and Ilya Sutskever, won the ILSVRC championship in 2012 with a significant margin of 16.4%, containing 5 convolutional layers, 3 pooling layers, and 3 fully connected layers

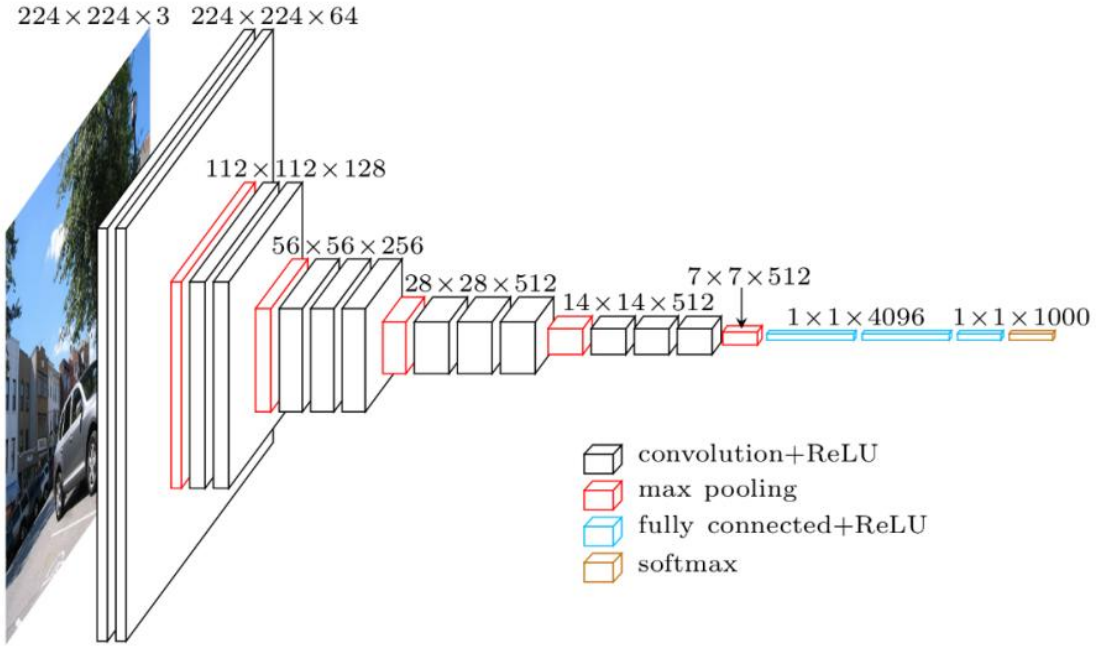


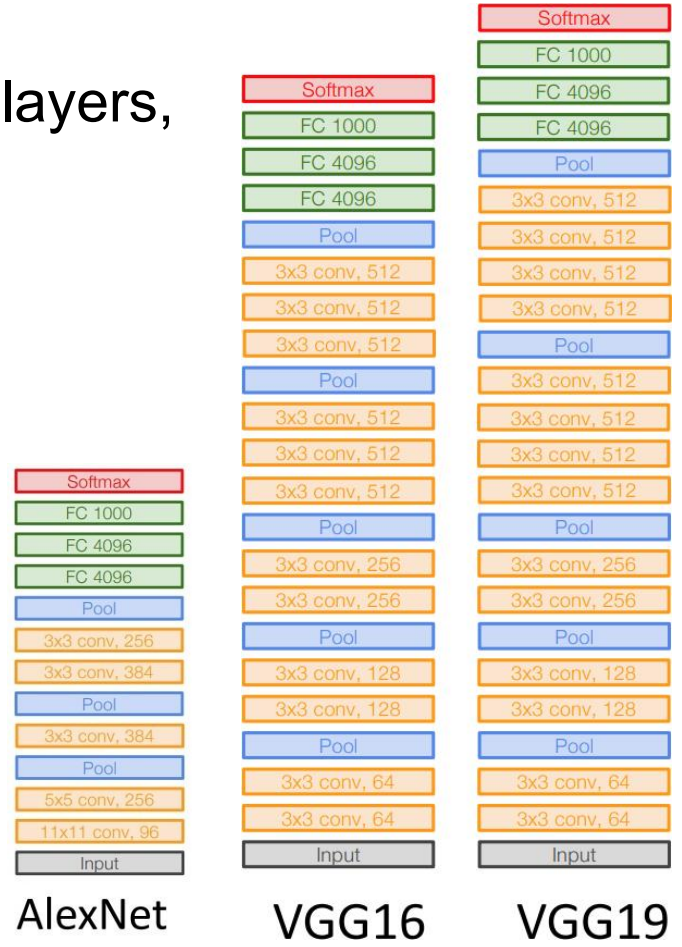
AlexNet

- Advanced techniques such as **dropout**, and **GPU acceleration** were utilized.
- A dual-GPU parallel computing acceleration mode was adopted. It refers to the process where the model structure remains the same across different GPUs, but the training data is split and trained separately to obtain different models, which are then fused.
- Parameters: 60 million; Model size: ≈240M.

VGG16

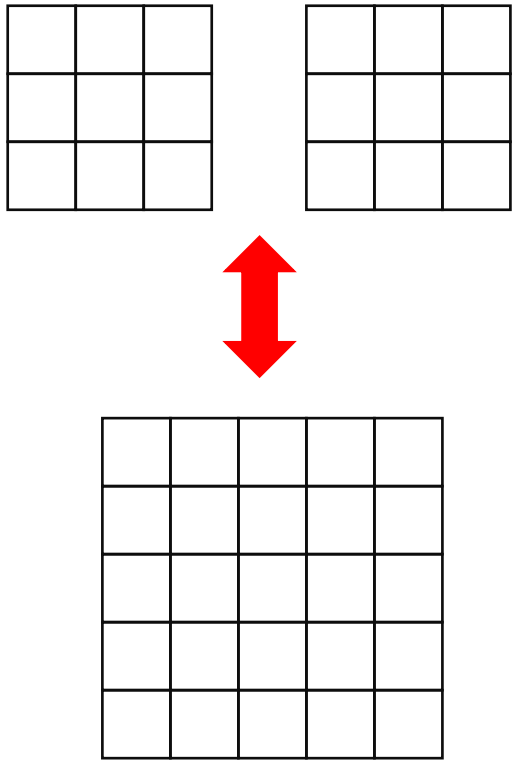
- The VGG16 network comprises 13 convolutional layers, 5 pooling layers, and 3 fully connected layers.



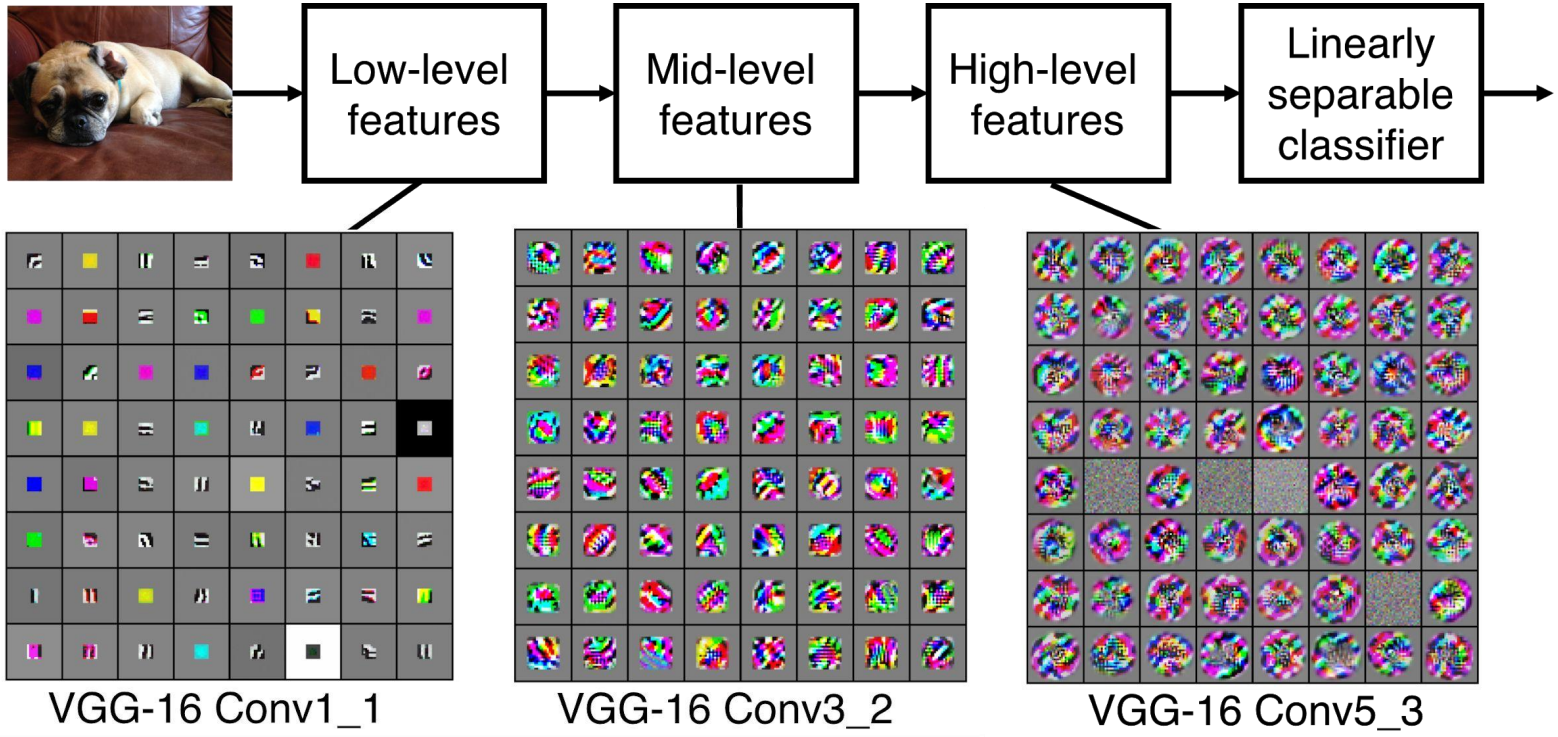


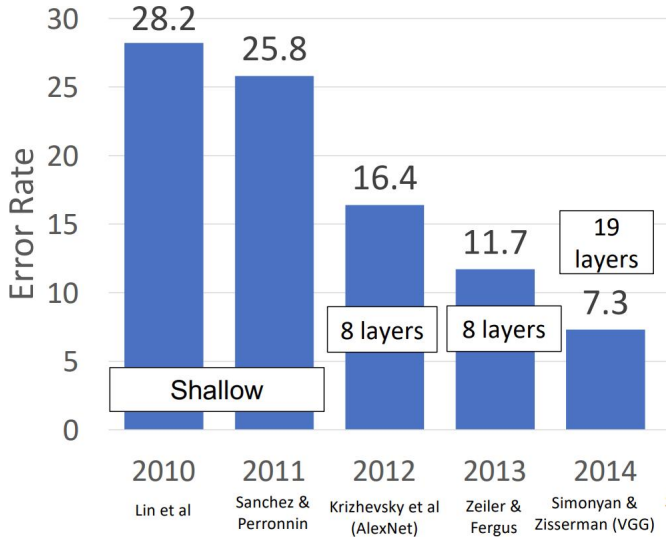
VGG16

- **Small filters**: Connecting multiple small convolutions in series is equivalent to one large convolution, e.g., using two 3*3 convolutions in series achieves a 5*5 effect (the same output feature size), but the number of parameters is only 9/25 of the previous one.
- **Deeper network**: By repeatedly stacking 3x3 convolutions and 2x2 pooling, a maximum depth of 16 layers was achieved.
- **Parameters**: 138 million; Model size: ≈500M.



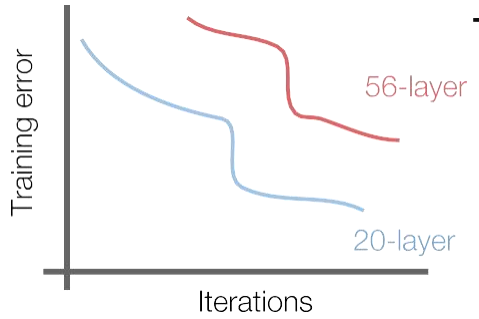
VGG16


Low-level features
Mid-level features
High-level features
Linearly separable classifier
VGG-16 Conv1_1
VGG-16 Conv3_2
VGG-16 Conv5_3


Error Rate
30
25
20
15
10
5
0
28.2
25.8
16.4
11.7
7.3
Shallow
8 layers
8 layers
19 layers
2010
Lin et al
2011
Sanchez & Perronnin
2012
Krizhevsky et al (AlexNet)
2013
Zeiler & Fergus
2014
Simonyan & Zisserman (VGG)

Problem

Simply deepening a neural network



Too many parameters can easily lead to overfitting if the training dataset is limited

The deeper the network, the easier it is for gradients to vanish when propagating backwards, making it difficult to optimize the model

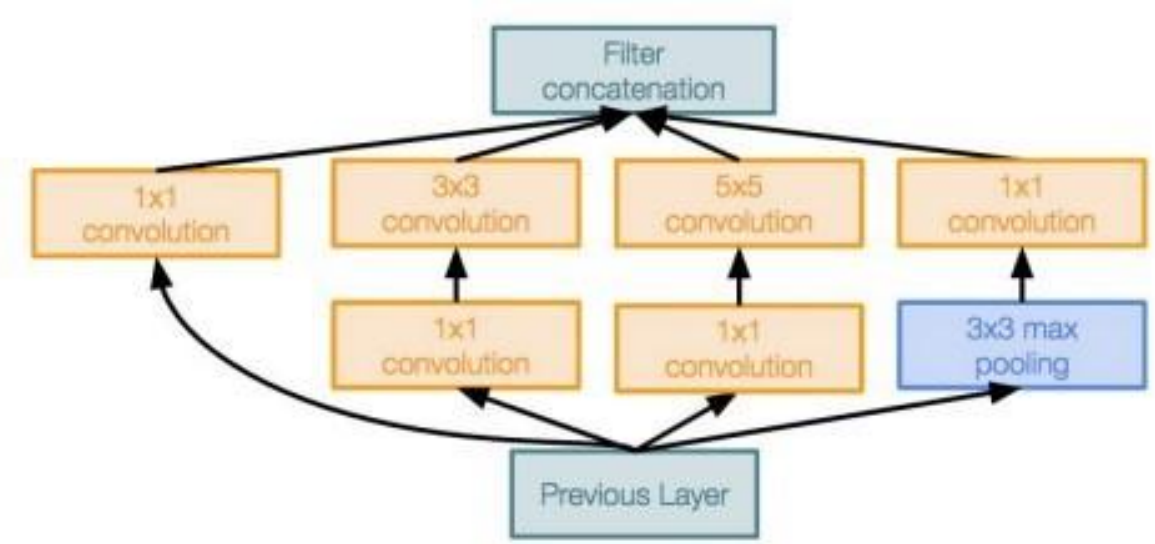
How to enhance the feature extraction capability of the network

Increase “width”

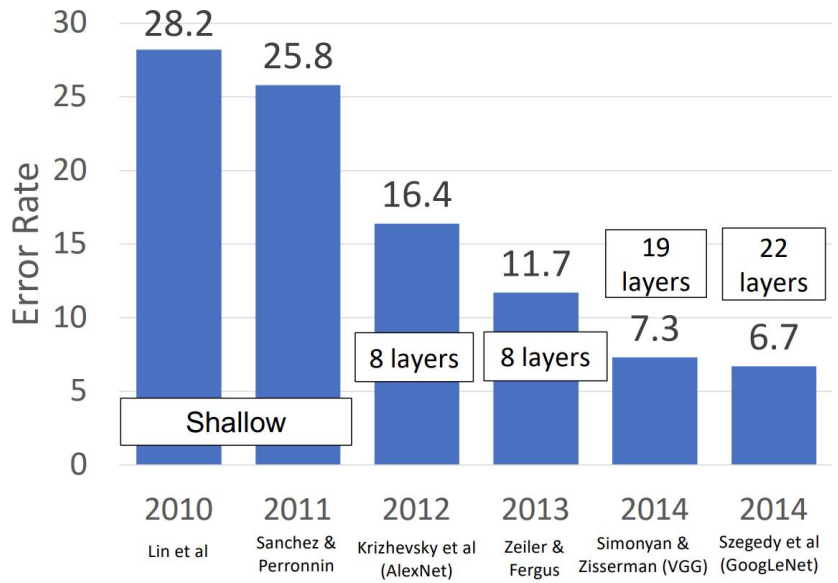
“Skip connection”

GoogLeNet

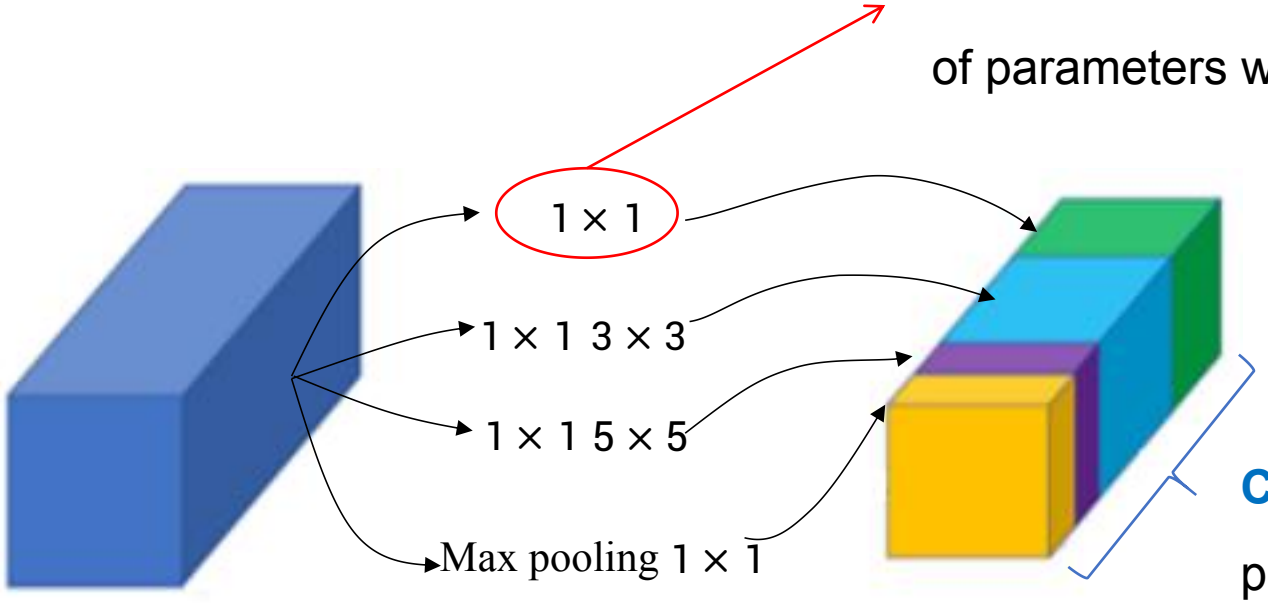
- **Inception Block**: Simultaneously employing 1x1, 3x3, and 5x5 convolution and pooling operations to extract features with different receptive fields



Increase the width of network



Inception

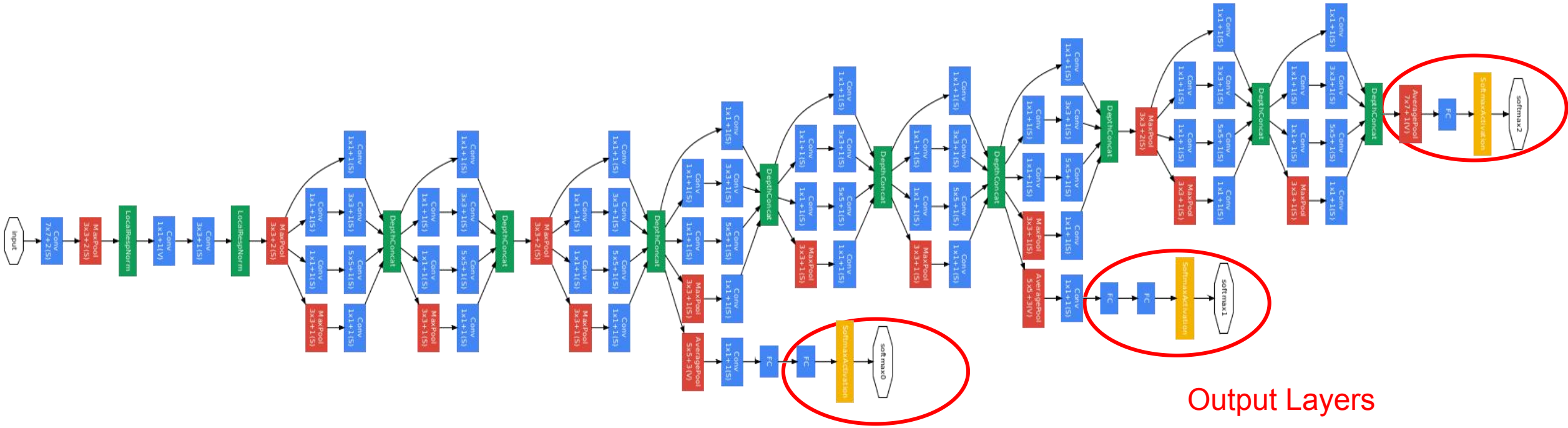


1*1 convolution: Reduce the number of input channels to avoid an explosion in the number of parameters while increasing the width

Concatenation: By configuring different padding sizes to achieve features of identical spatial dimensions, the features can be concatenated along the channel

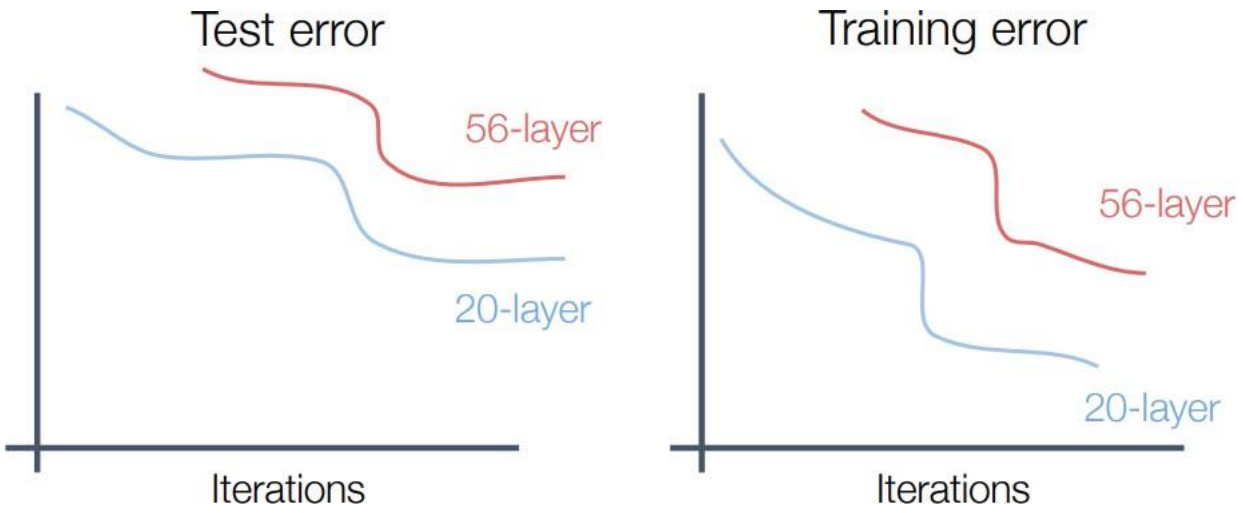
Deeper Structure

- 22 layers: To mitigate the vanishing gradient problem, extra softmax outputs are strategically added at different depths to ensure stable gradient backpropagation.

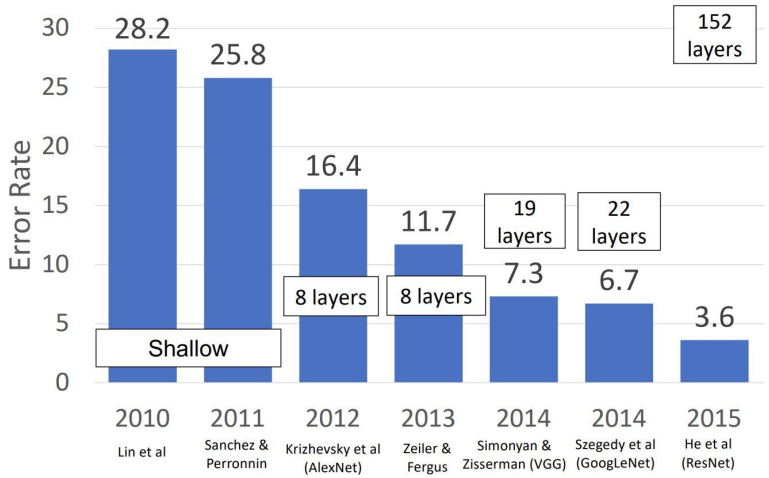


ResNet

Deep ConvNets tend to underfit.



How to train such a deep ConvNet?

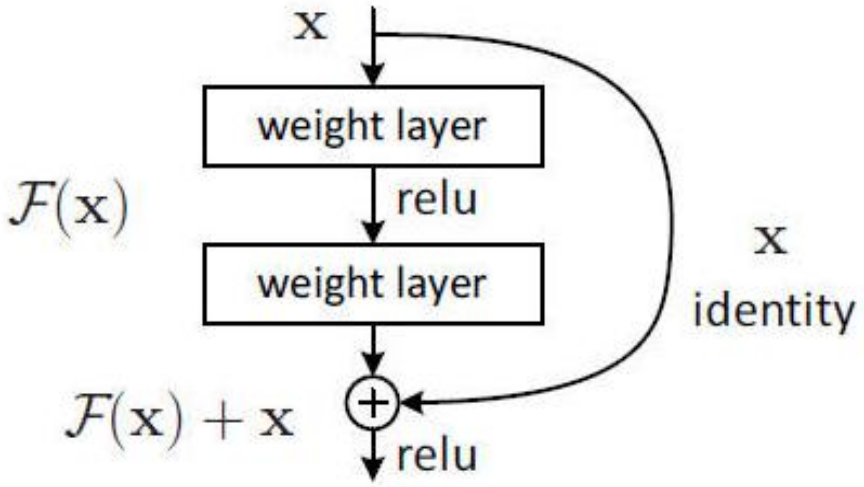


ResNet

- **Skip Connection**: if a layer is redundant, the network can trivially zero its weights (F(x)=0) to recover identity mapping (just like the shortcut between A and D).

MSRA @ ILSVRC & COCO 2015 Competitions

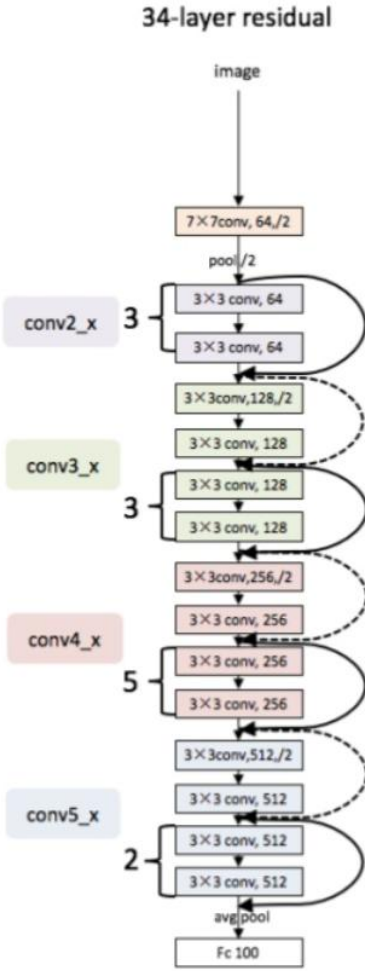
- **1st places in all five main tracks**
  - ImageNet Classification: "*Ultra-deep*" (quote Yann) 152-layer nets
  - ImageNet Detection: 16% better than 2nd
  - ImageNet Localization: 27% better than 2nd
  - COCO Detection: 11% better than 2nd
  - COCO Segmentation: 12% better than 2nd



Residual Block

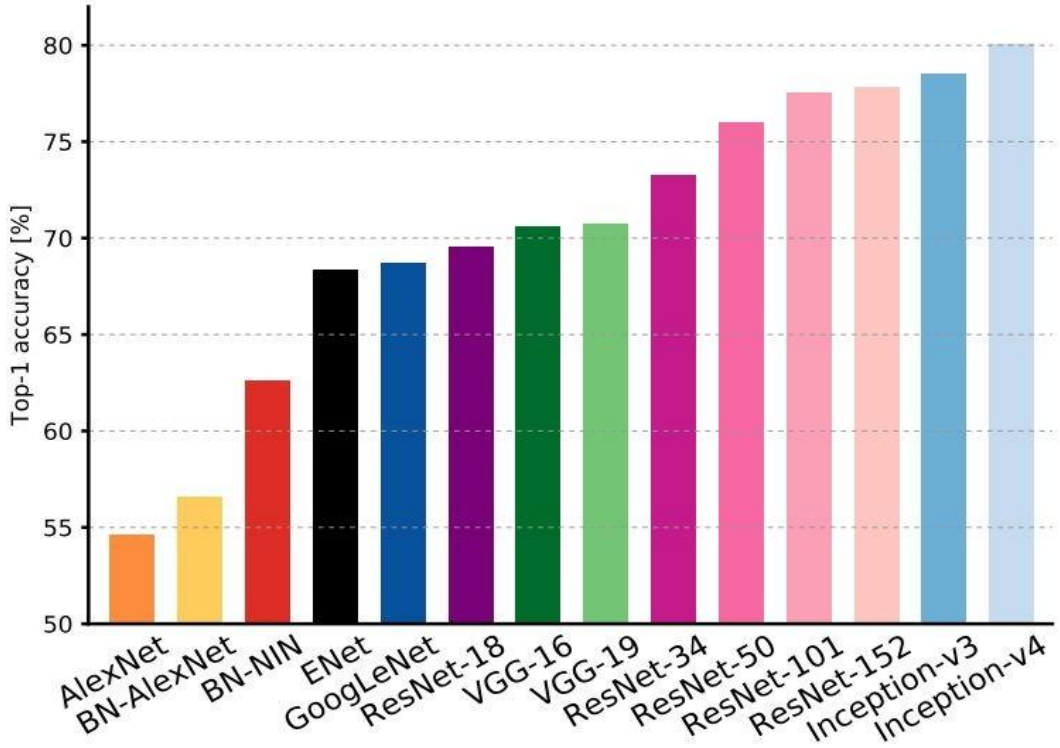
ResNet

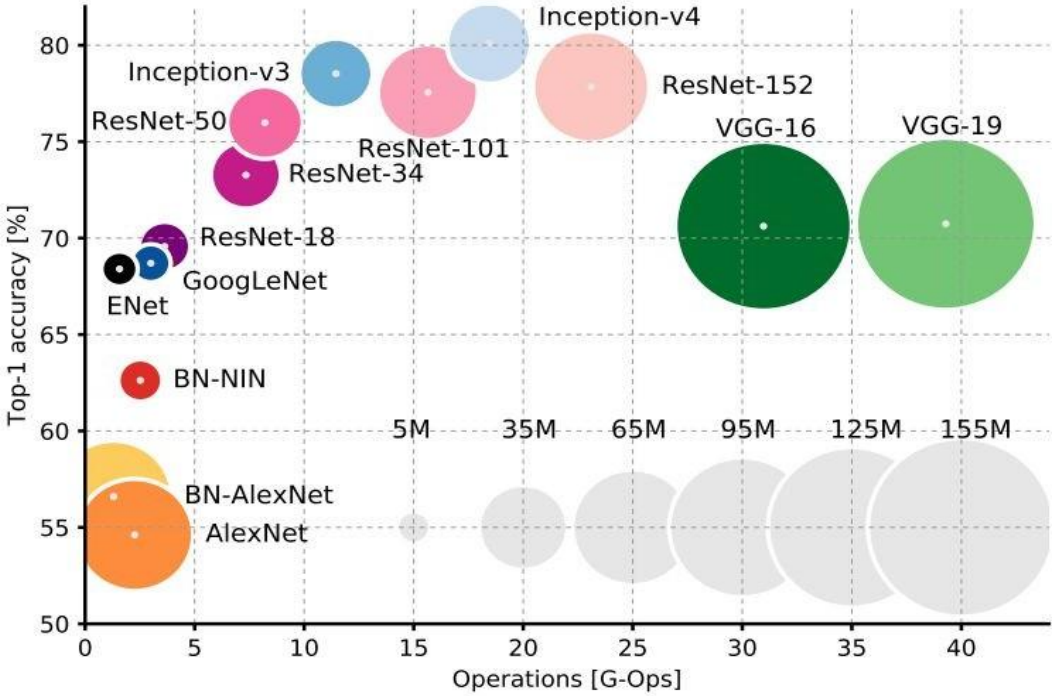
layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112	7×7, 64, stride 2				
conv2_x	56×56	3×3 max pool, stride 2				
		$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$
conv3_x	28×28	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times8$
conv4_x	14×14	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times23$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times36$
conv5_x	7×7	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$
	1×1	average pool, 1000-d fc, softmax				
FLOPs		1.8×10^9	3.6×10^9	3.8×10^9	7.6×10^9	11.3×10^9



上海交通大學 SHANGHAI JIAO TONG UNIVERSITY

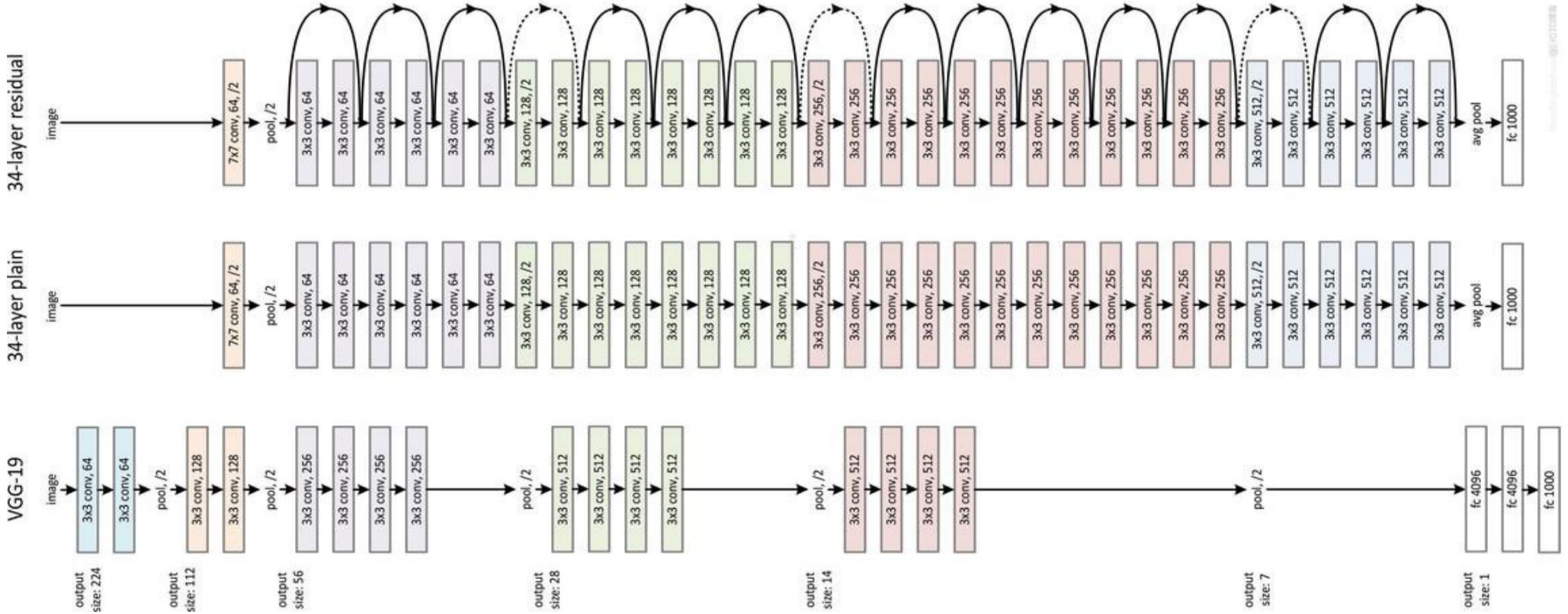
Comparison



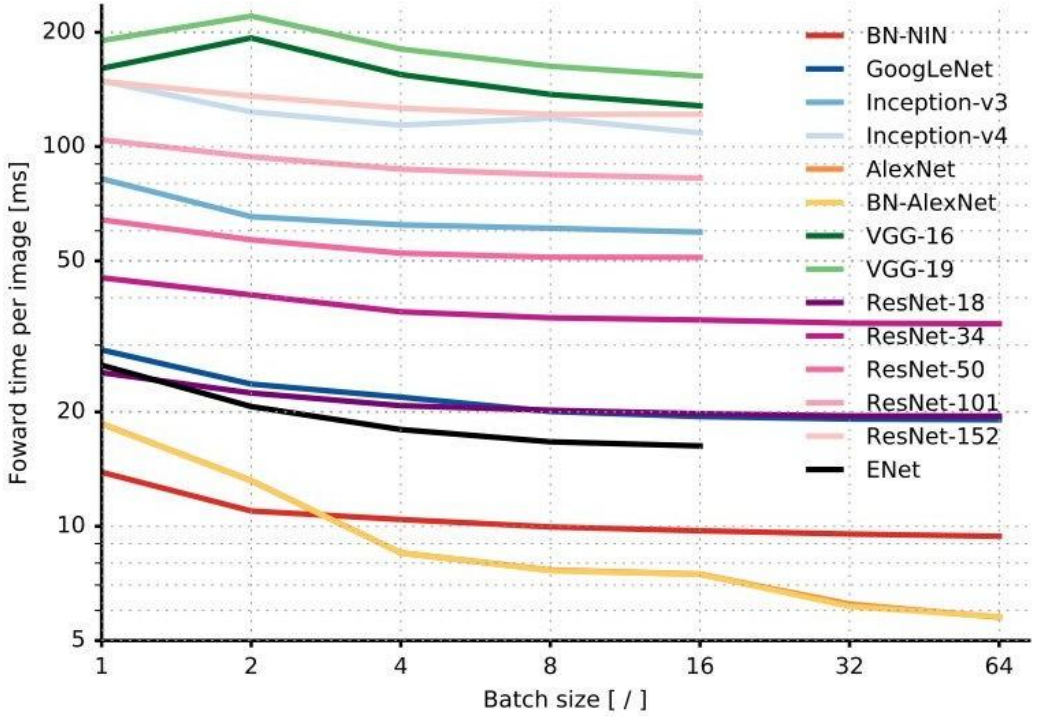


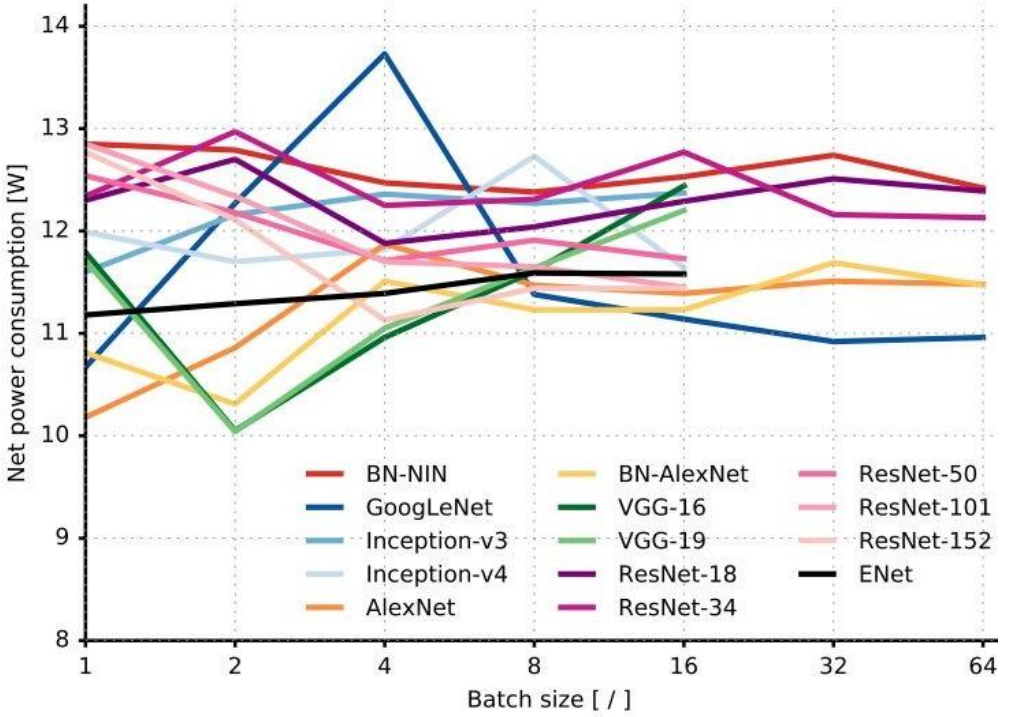
Canziani et al, An Analysis of Deep Neural Network Models for Practical Applications, https://arxiv.org/abs/1605.07678.

Comparison


34-layer residual
image
7x7 conv, 64, /2
pool, /2
3x3 conv, 64
3x3 conv, 128
3x3 conv, 256
3x3 conv, 512
avg pool
fc 1000
34-layer plain
VGG-19
fc 4096
output size: 224
output size: 112
output size: 56
output size: 28
output size: 14
output size: 7
output size: 1

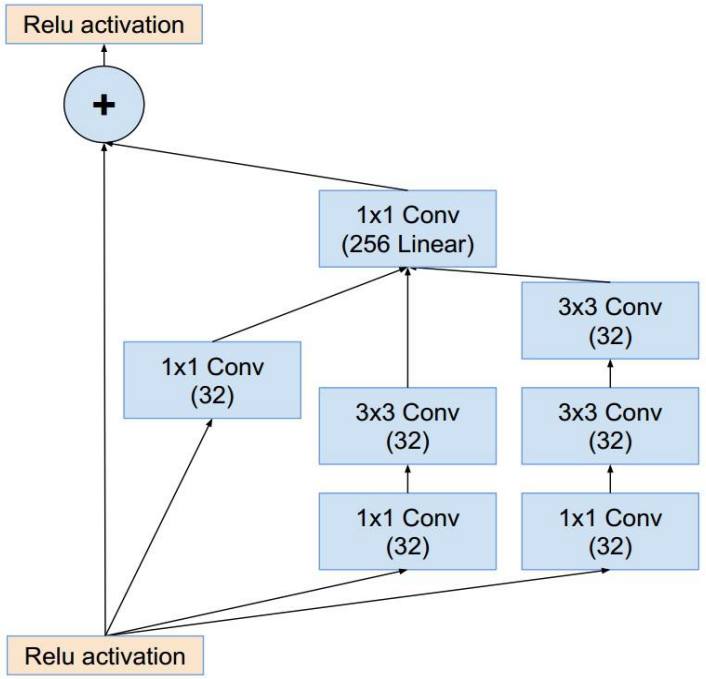
Comparison

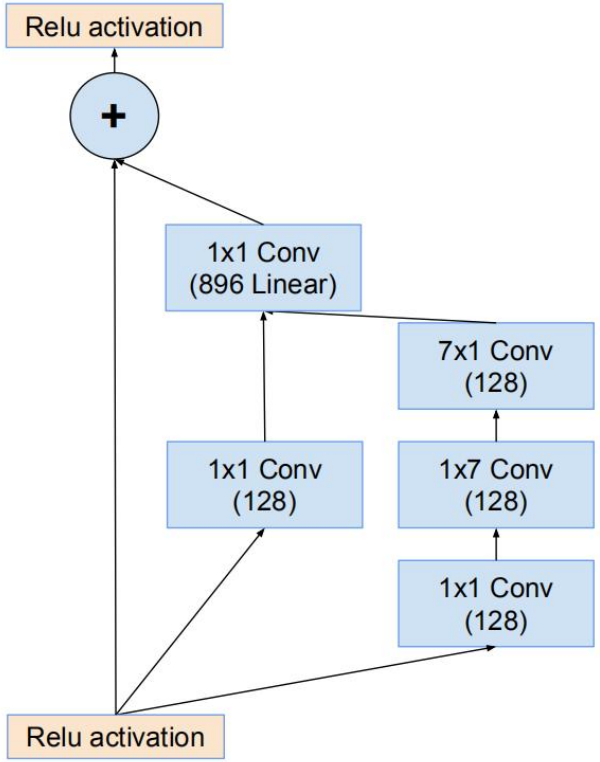

Foward time per image [ms]
Batch size [/]
200
100
50
20
10
5
1
2
4
8
16
32
64
BN-NIN
GoogLeNet
Inception-v3
Inception-v4
AlexNet
BN-AlexNet
VGG-16
VGG-19
ResNet-18
ResNet-34
ResNet-50
ResNet-101
ResNet-152
ENet


Net power consumption [W]
Batch size [/]
14
13
12
11
10
9
8
1
2
4
8
16
32
64
BN-NIN
GoogLeNet
Inception-v3
Inception-v4
AlexNet
BN-AlexNet
VGG-16
VGG-19
ResNet-18
ResNet-34
ResNet-50
ResNet-101
ResNet-152
ENet

Inception-ResNet

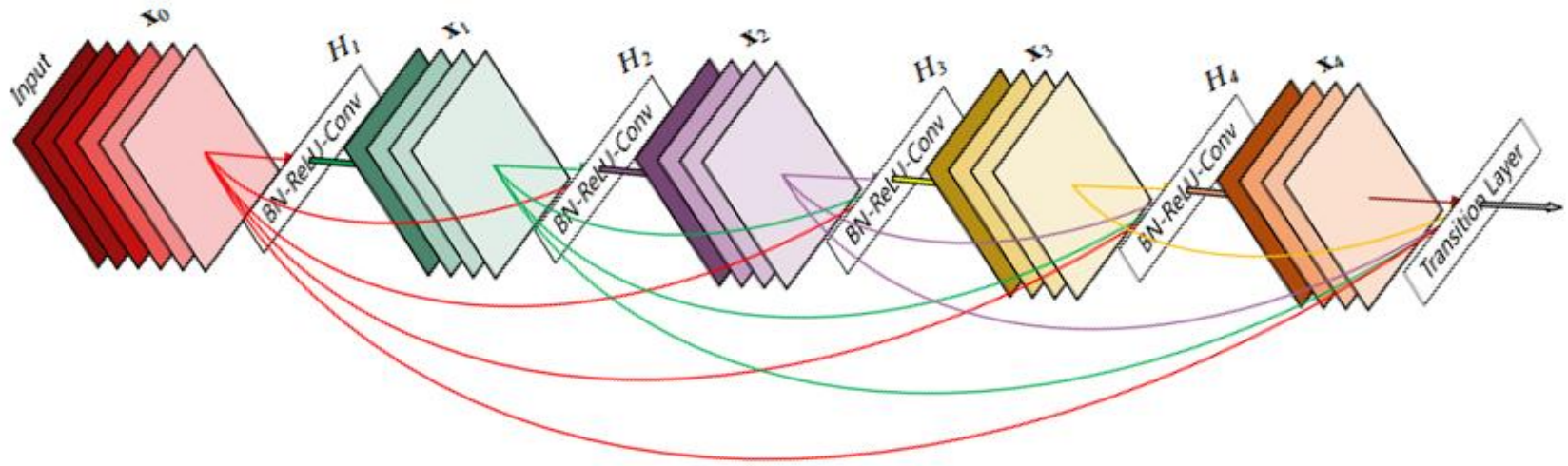
- Combining the concepts of Inception and ResNet



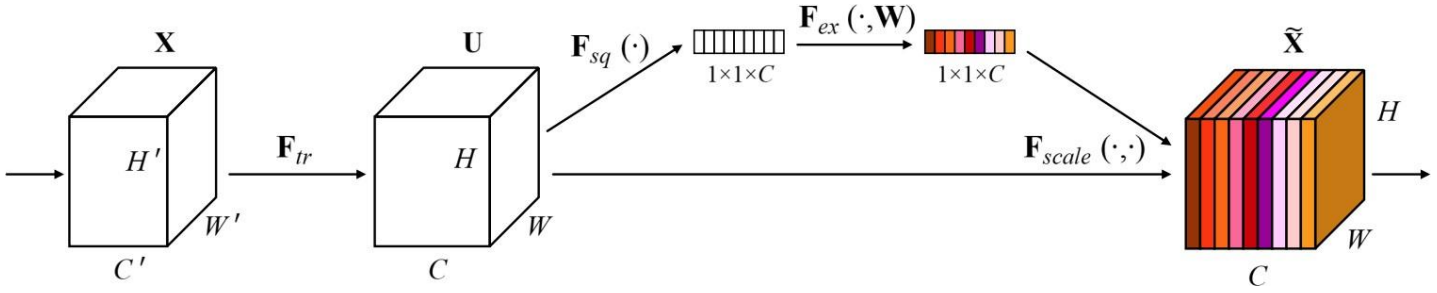


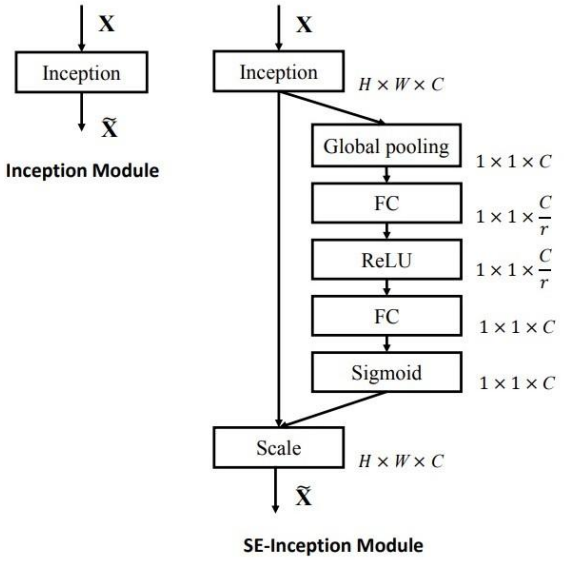
DenseNet

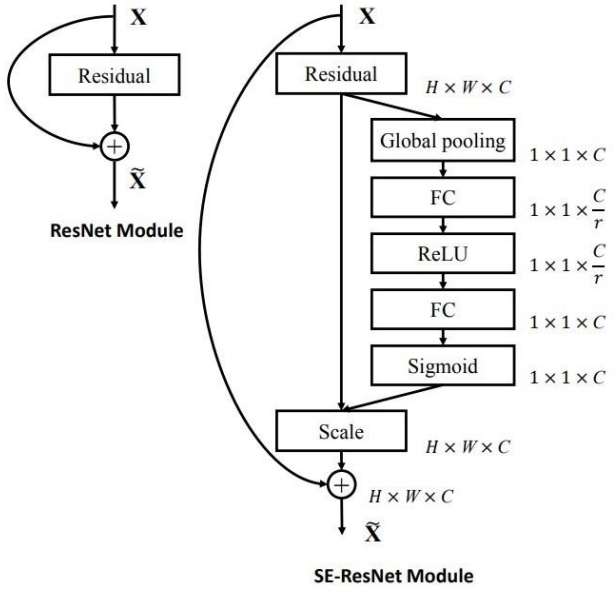
- Connect the output with the input in parallel, enabling each layer to directly receive the outputs from all preceding layers.
- Utilizing shortcut connections to directly link all layers together, this architecture not only mitigates the vanishing gradient problem but also promotes feature reuse.

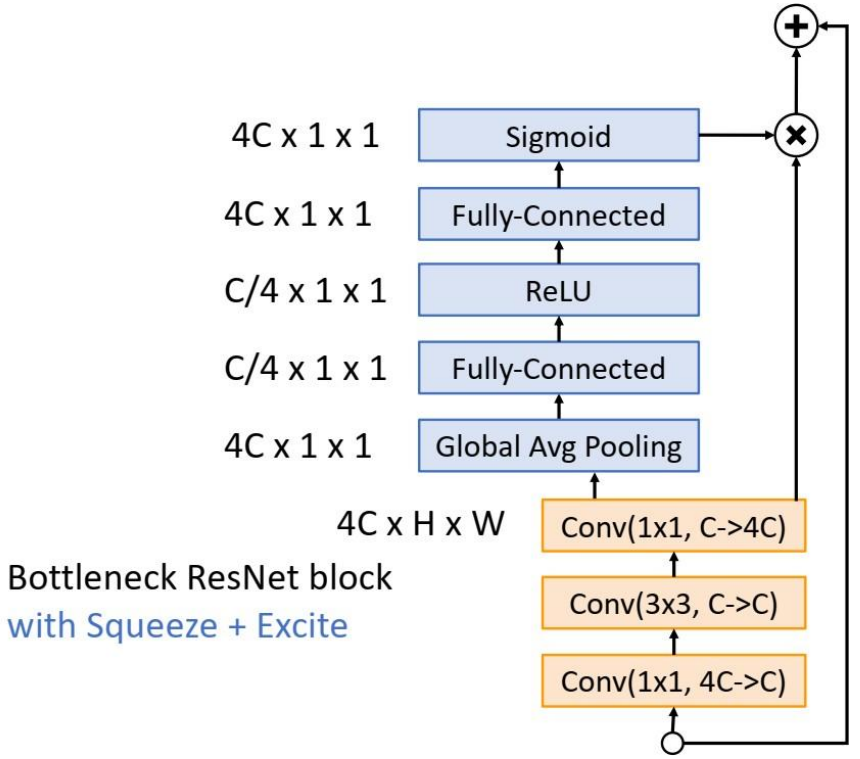


SENet









Bottleneck ResNet block
with Squeeze + Excite

Classification with CNN, Transformer, etc.

考核要求：

- ✓ **实验目标**：使用卷积神经网络（CNN）进行图像分类任务，利用PyTorch、TensorFlow或Jittor实现完整流程，确保代码可运行。
- ✓ **数据集要求**：选择标准分类数据集（如MNIST、CIFAR-10）或自行准备标注数据集，进行预处理和划分。
- ✓ **模型设计**：设计单层或多层CNN网络，可创新设计（如加入ResNet模块），需说明设计思路。
- ✓ **性能评估**：记录训练过程中的损失和准确率，提供测试集分类准确率，探讨超参数影响。
- ✓ **代码要求**：PyTorch或TensorFlow框架代码实现，清晰可读并附注释；Jittor实现为加分项，需提交源码。
- ✓ **实验报告**：包含实验背景、数据集说明、网络结构、实验过程、结果分析和总结，给出训练曲线和性能指标，可附加创新点或失败实验反思。

Thanks

Welcome to join VisionXLab

yangxue.site