Neural Networks: Convolutional Nets, Regularization, Data augmentation, Dropout

Machine Learning Course - CS-433
05 Nov 2025

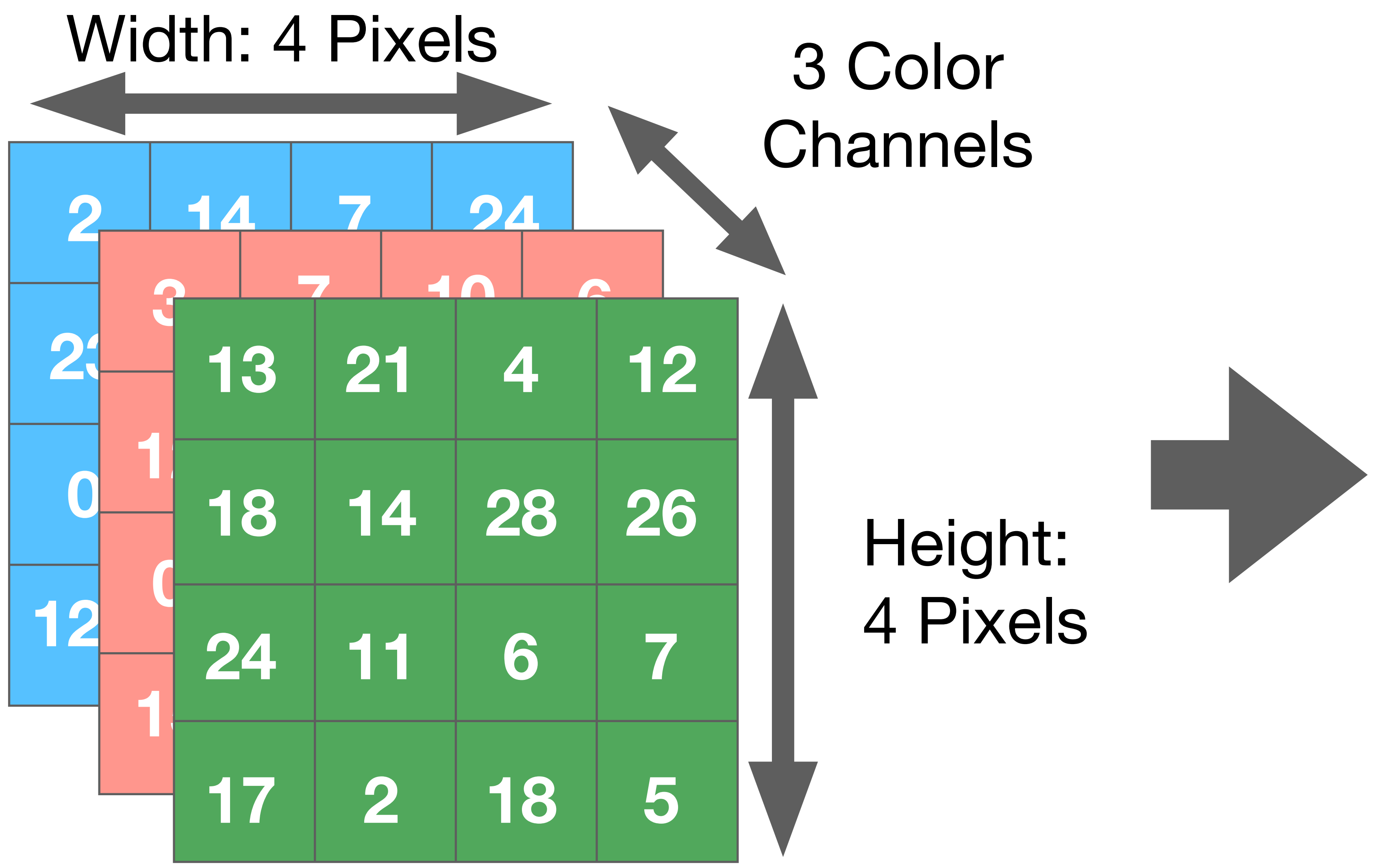
Robert West
(Slide credits: Nicolas Flammarion)

EPFL

Convolutional Networks

Fully connected NNs have many parameters and do not capture spatial dependencies

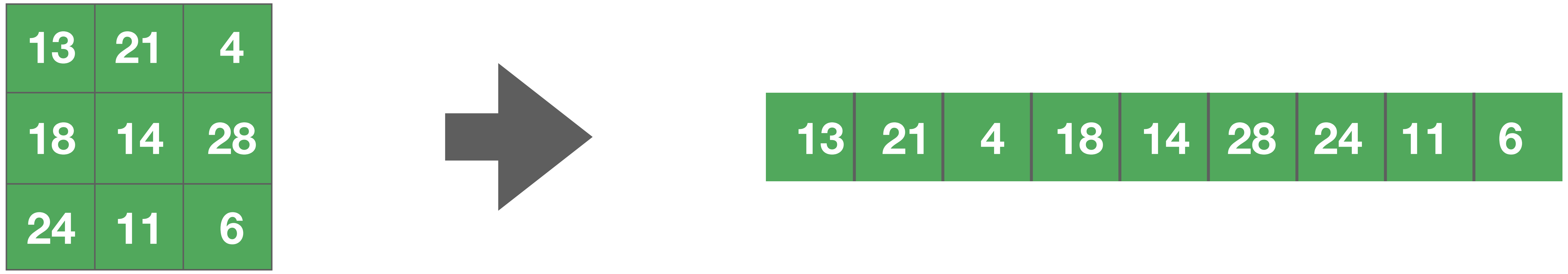
- Fully connected NNs have $O(K^2L)$ parameters: training requires a lot of data



ImageNet Dimension:

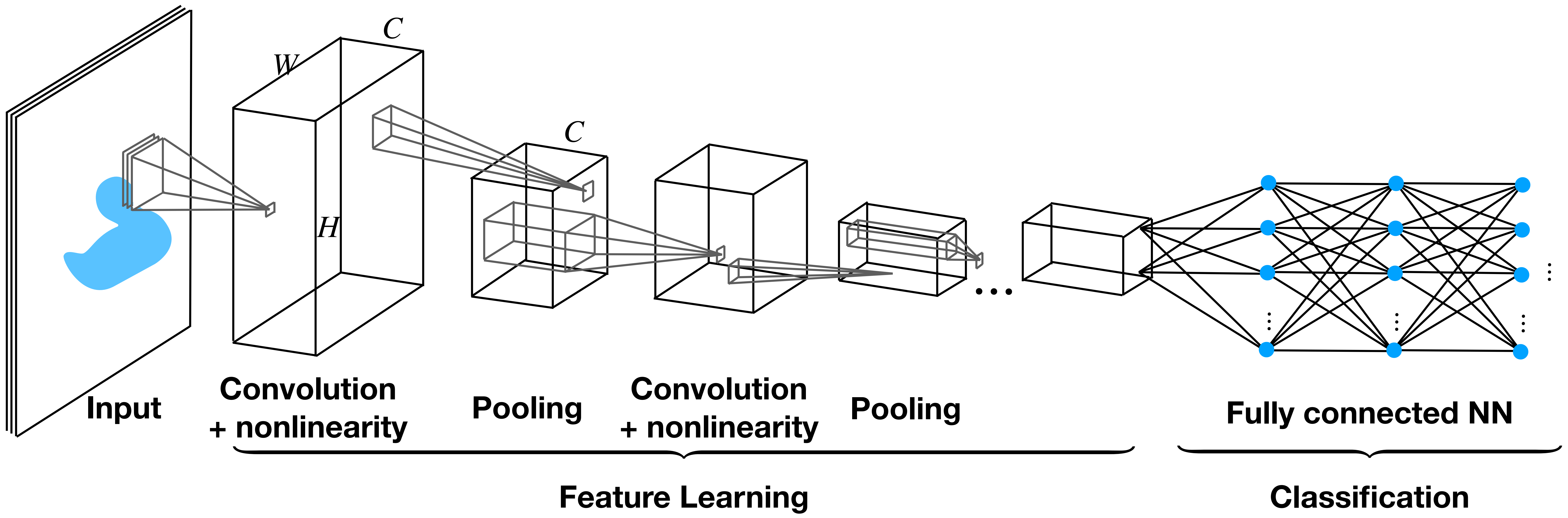
$$256 \times 256 \times 3 \sim 2 \cdot 10^5$$

- Fully connected neural networks interpret an image as a flattened vector, disregarding the original spatial dependencies



Flattening of a 3×3 picture into a 1×9 vector

Convolutional NNs: General Structure



- Convolutional networks consist of **sparsely connected convolutional layers** in place of fully-connected linear layers
- **Pooling layers** perform spatial downsampling (typically reducing the dimensions from $H \times W$ to $H/2 \times W/2$
- A fully-connected network at the end performs classification based on the extracted features

Convolution

For a filter f of size K and a stride S:

$$x^{(1)}_{n,m} = \sum_{k,l=0}^{K-1} f_{k,l} \cdot x^{(0)}_{nS+k,mS+l}$$

- For example, $K = 3$, $S = 1$

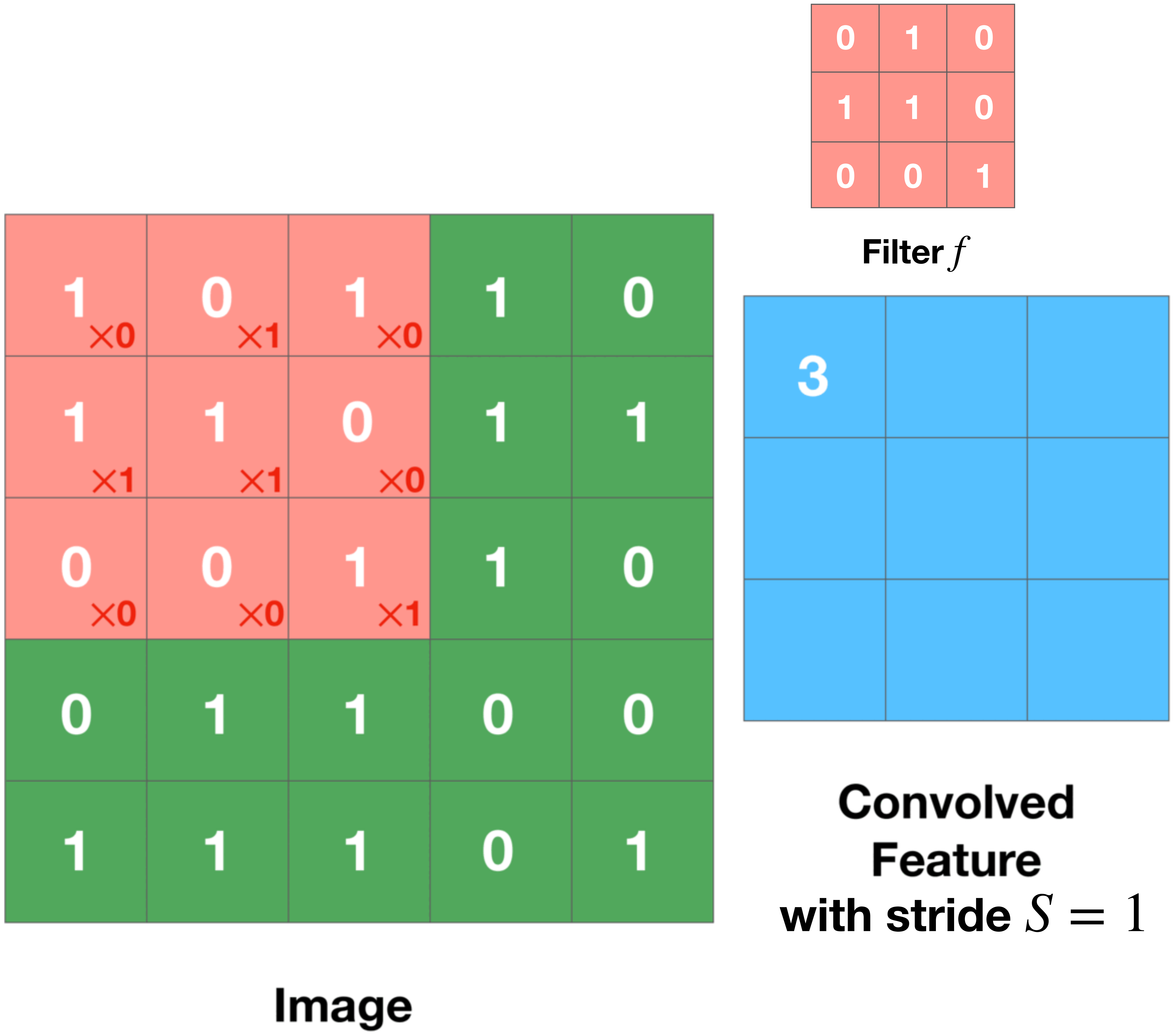
$$x^{(1)}_{0,0} = \boxed{\sum_{k=0}^{2} \sum_{l=0}^{2} f_{k,l}} \cdot x^{(0)}_{0+k,0+l}$$

$$x^{(1)}_{0,1} = \boxed{\sum_{k=0}^{2} \sum_{l=0}^{2} f_{k,l}} \cdot x^{(0)}_{0+k,1+l}$$

$$\vdots$$

Weight sharing

- Filter f represents the learnable weights, analogous to the weights W in an MLP
- We use the same filter at every position - **weight sharing**



(Video)

Convolution

For a filter f of size K and a stride S:

$$x^{(1)}_{n,m} = \sum_{k,l=0}^{K-1} f_{k,l} \cdot x^{(0)}_{nS+k,mS+l}$$

- For example, $K = 3$, $S = 1$

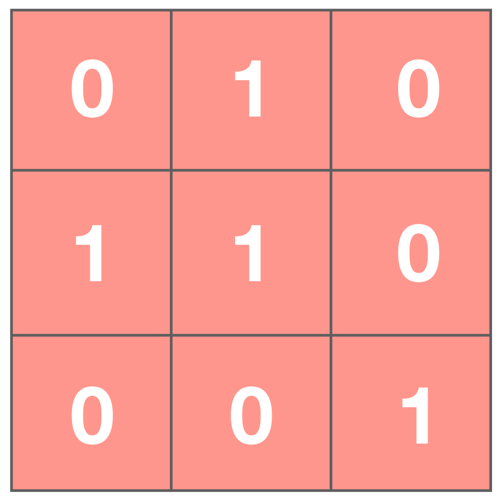
$$x^{(1)}_{0,0} = \sum_{k=0}^{2}\sum_{l=0}^{2} f_{k,l} \cdot x^{(0)}_{0+k,0+l}$$

$$x^{(1)}_{0,1} = \sum_{k=0}^{2}\sum_{l=0}^{2} f_{k,l} \cdot x^{(0)}_{0+k,1+l}$$

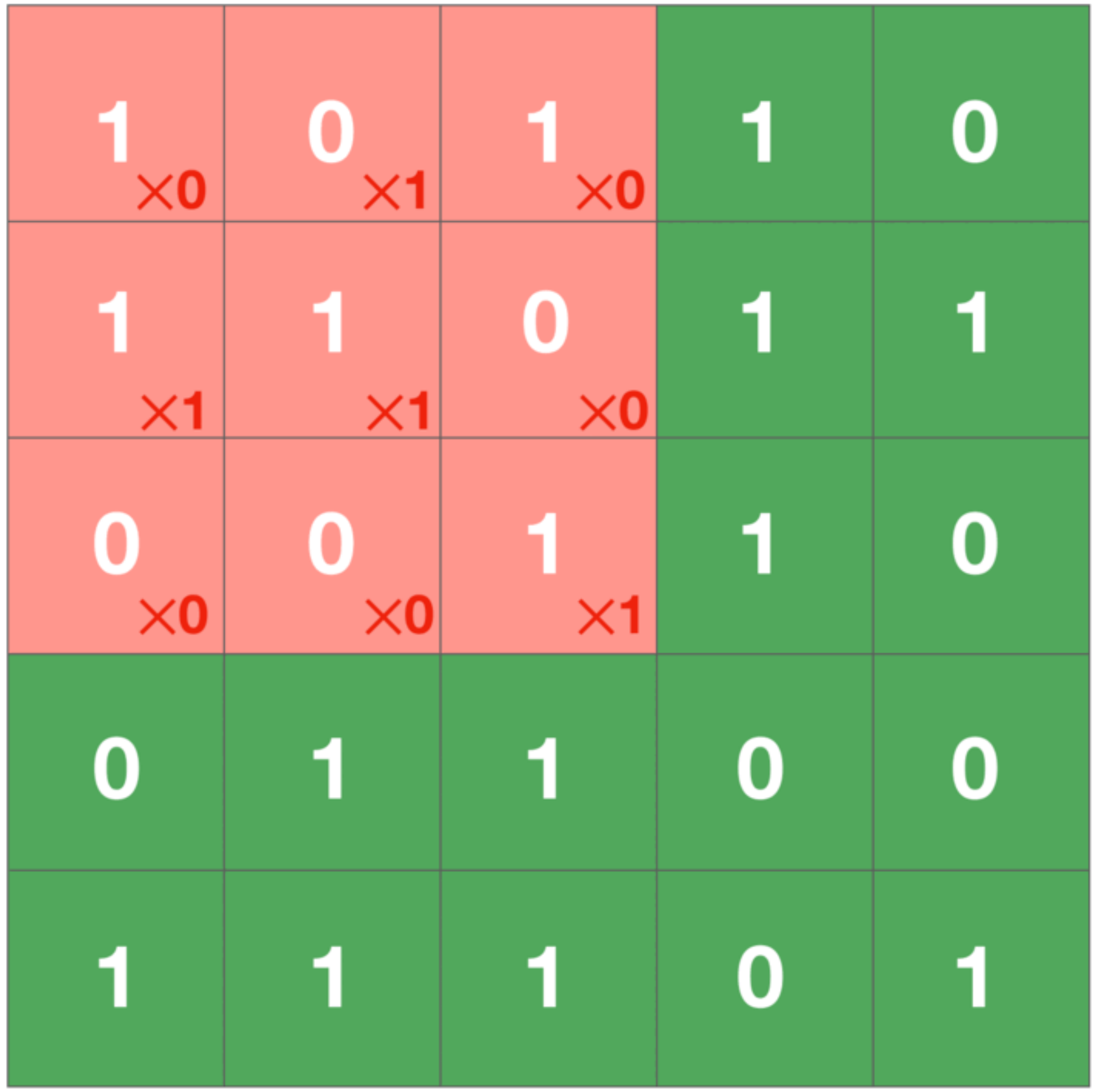
$$\vdots$$

Weight sharing

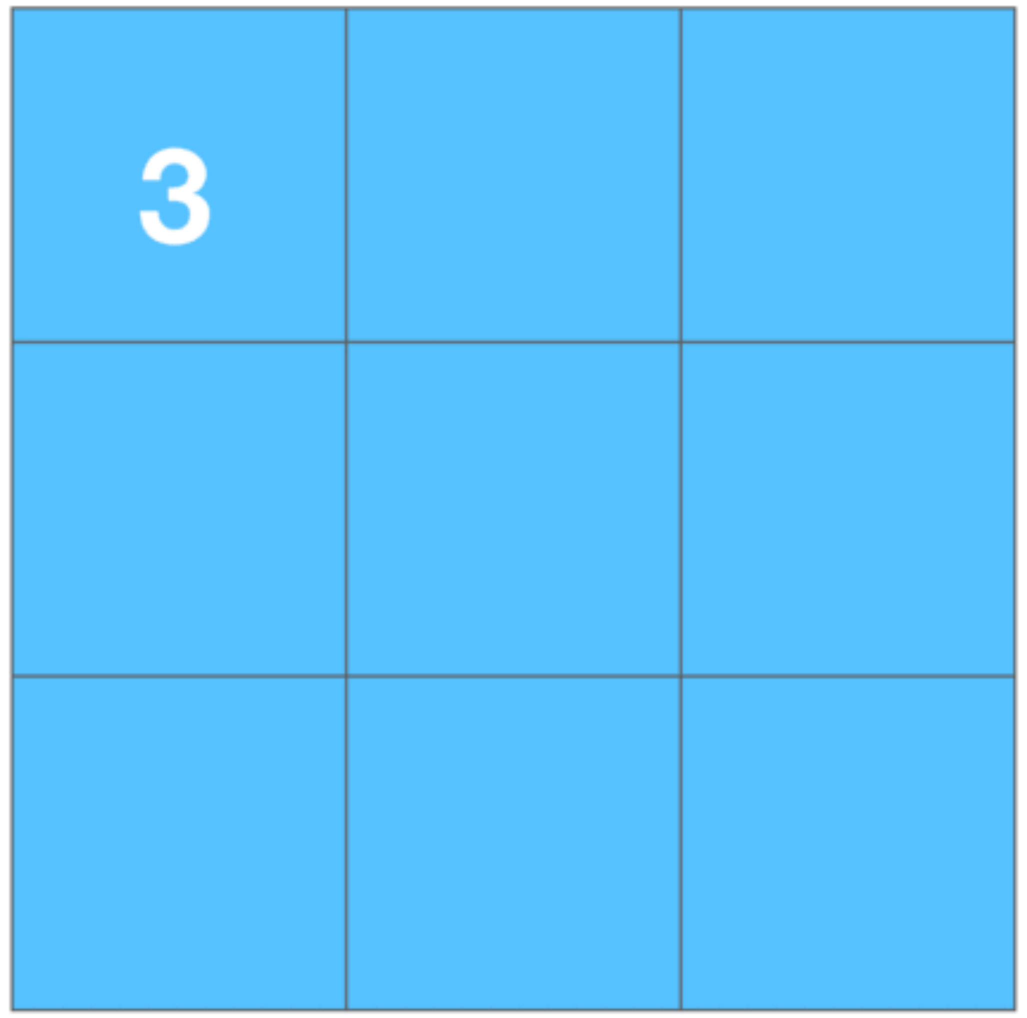
- Filter f represents the learnable weights, analogous to the weights W in an MLP
- We use the same filter at every position - **weight sharing**



Filter f



Image



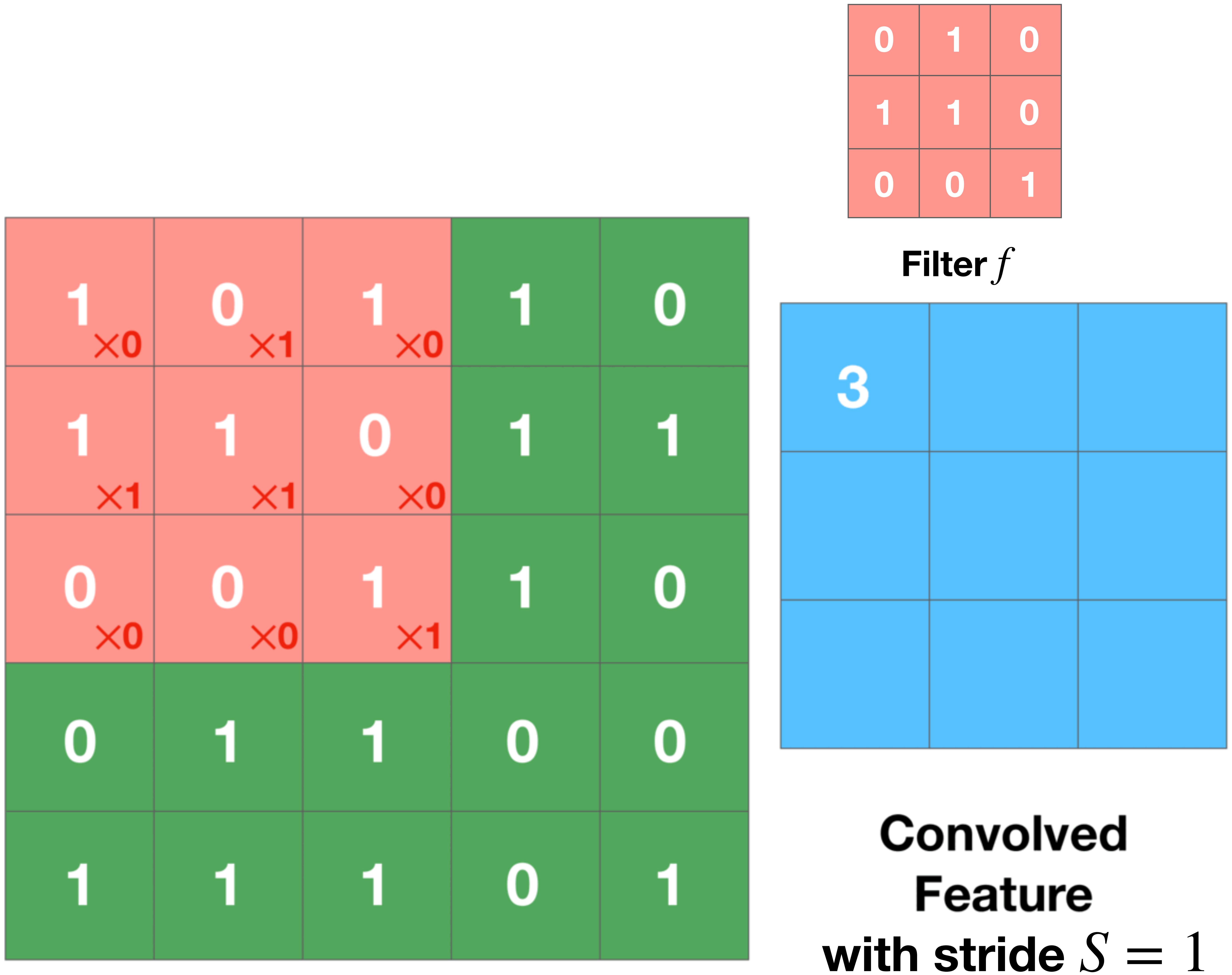
Convolved Feature with stride $S = 1$

(Video)

Convolution

$$x^{(1)}_{n,m} = \sum_{k,l=0}^{K-1} f_{k,l} \cdot x^{(0)}_{nS+k,mS+l}$$

- **Local connectivity** - $x^{(1)}_{n,m}$ only depends on the value of $x^{(0)}$ close to (nS, mS) for small K
- **Translation equivariance** - a shifted input results in a shifted output



Filter f

Convolved Feature with stride $S = 1$

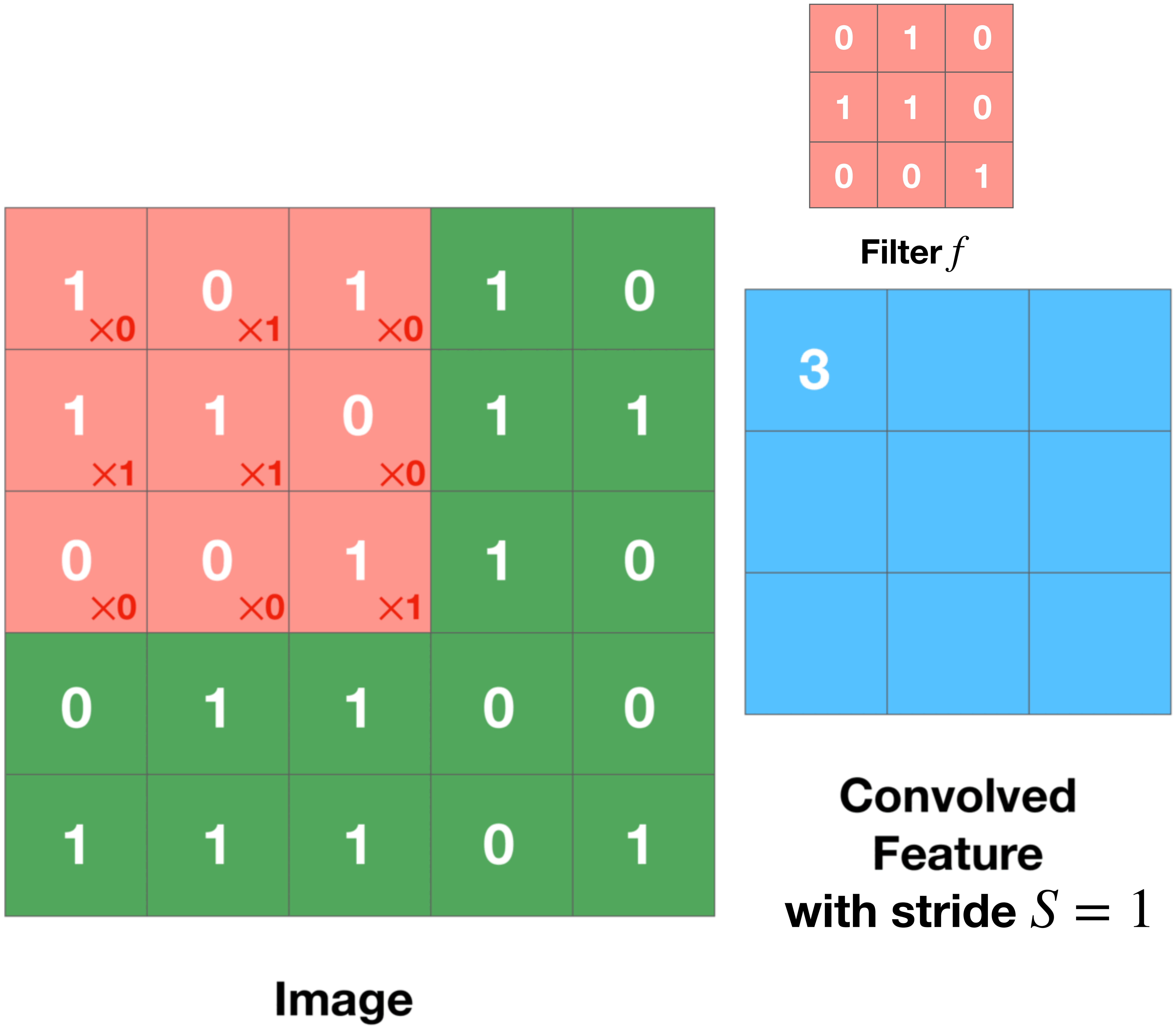
Image

➡ Convolution requires fewer parameters which are universal across different locations

Convolution

$$x^{(1)}_{n,m} = \sum_{k,l=0}^{K-1} f_{k,l} \cdot x^{(0)}_{nS+k,mS+l}$$

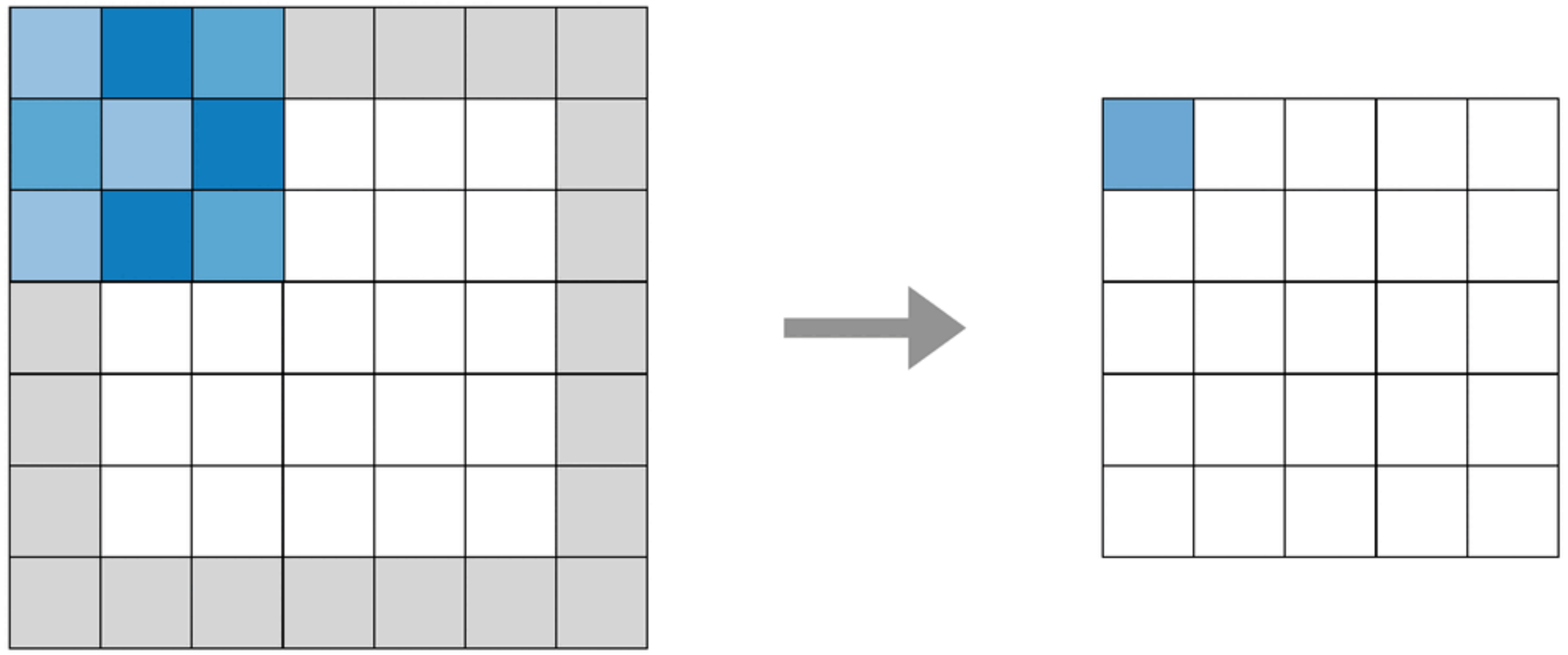
- **Local connectivity** - $x^{(1)}_{n,m}$ only depends on the value of $x^{(0)}$ close to (nS, mS) for small K
- **Translation equivariance** - a shifted input results in a shifted output



➡ Convolution requires fewer parameters which are universal across different locations

Handling of borders

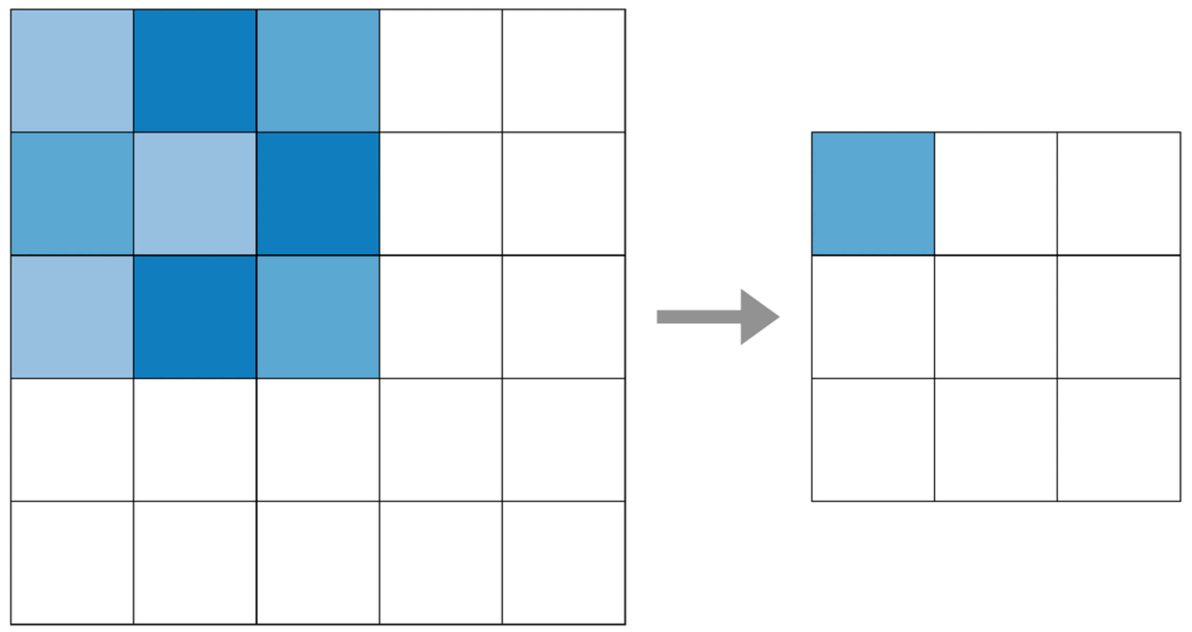
Zero padding:



Add zeros to each side of the input's boundaries

➡ The convolved feature has the same dimension as the input

Valid padding:



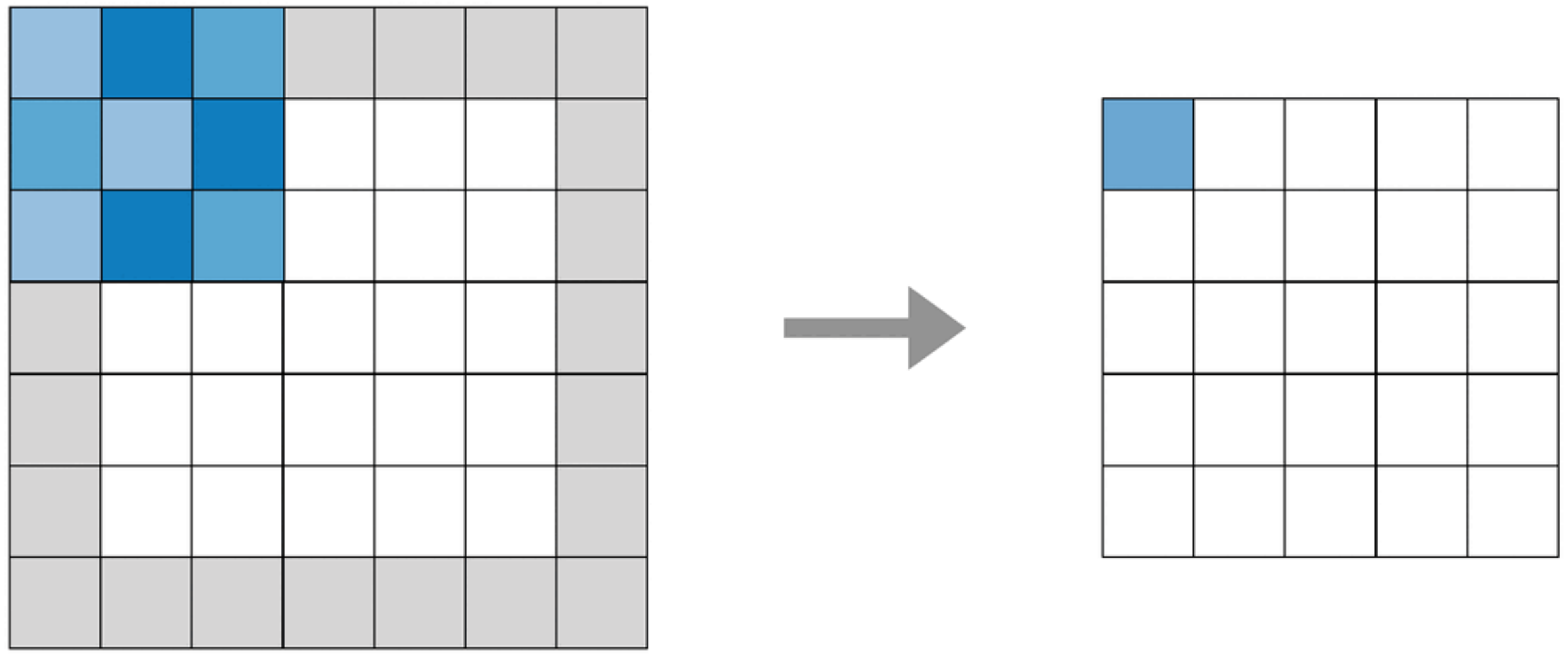
Perform the convolution only where the entire filter fits inside the original data

➡ The convolved feature has smaller dimensions than the input

(Video)

Handling of borders

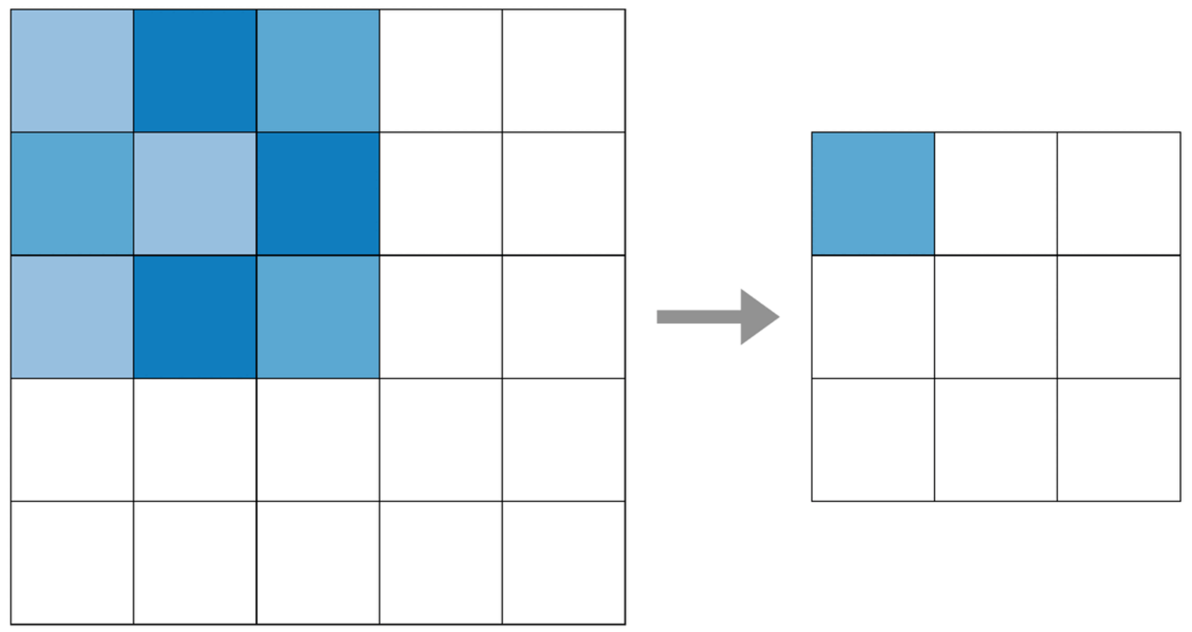
Zero padding:



Add zeros to each side of the input's boundaries

➡ The convolved feature has the same dimension as the input

Valid padding:



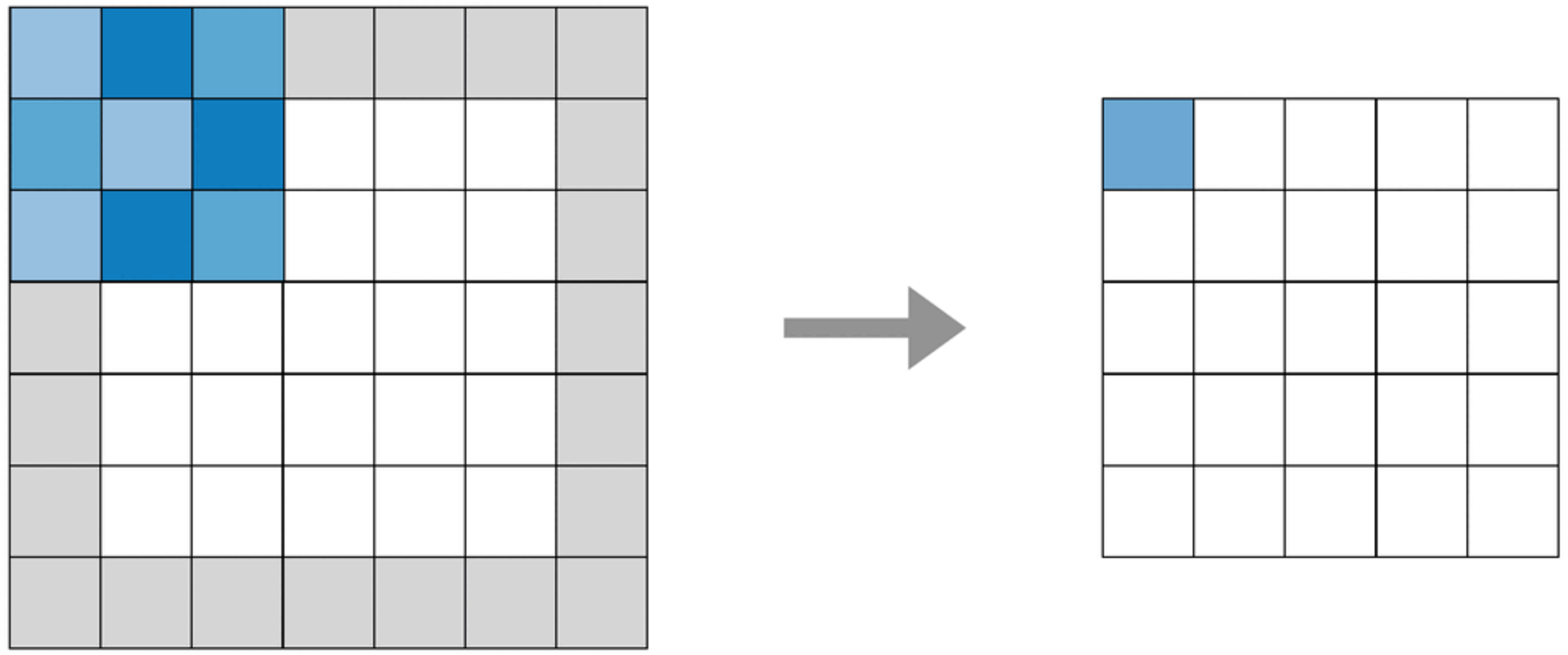
Perform the convolution only where the entire filter fits inside the original data

➡ The convolved feature has smaller dimensions than the input

(Video)

Handling of borders

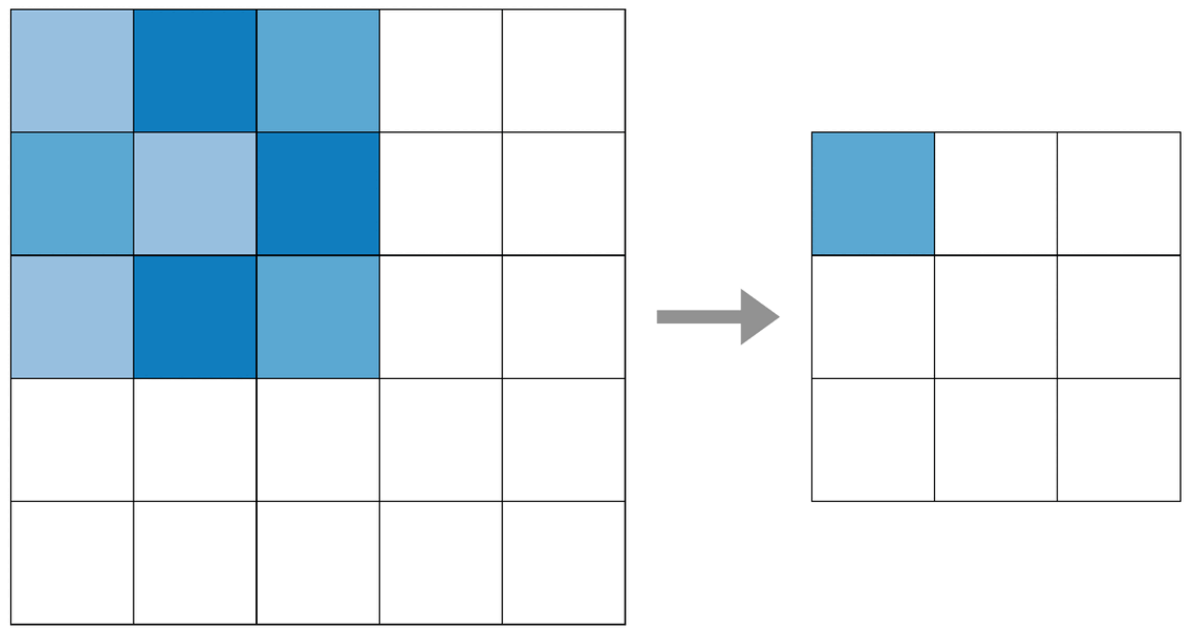
Zero padding:



Add zeros to each side of the input's boundaries

➡ The convolved feature has the same dimension as the input

Valid padding:

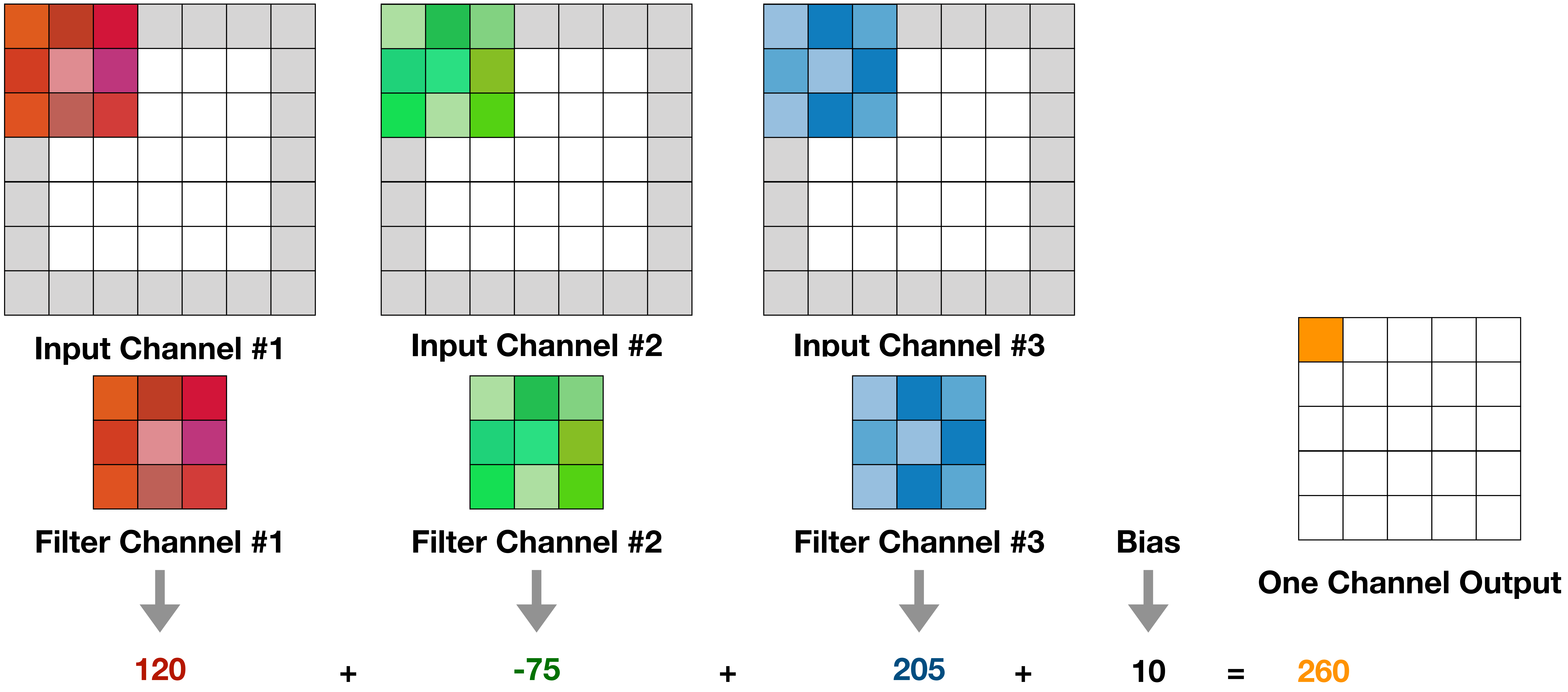


Perform the convolution only where the entire filter fits inside the original data

➡ The convolved feature has smaller dimensions than the input

(Video)

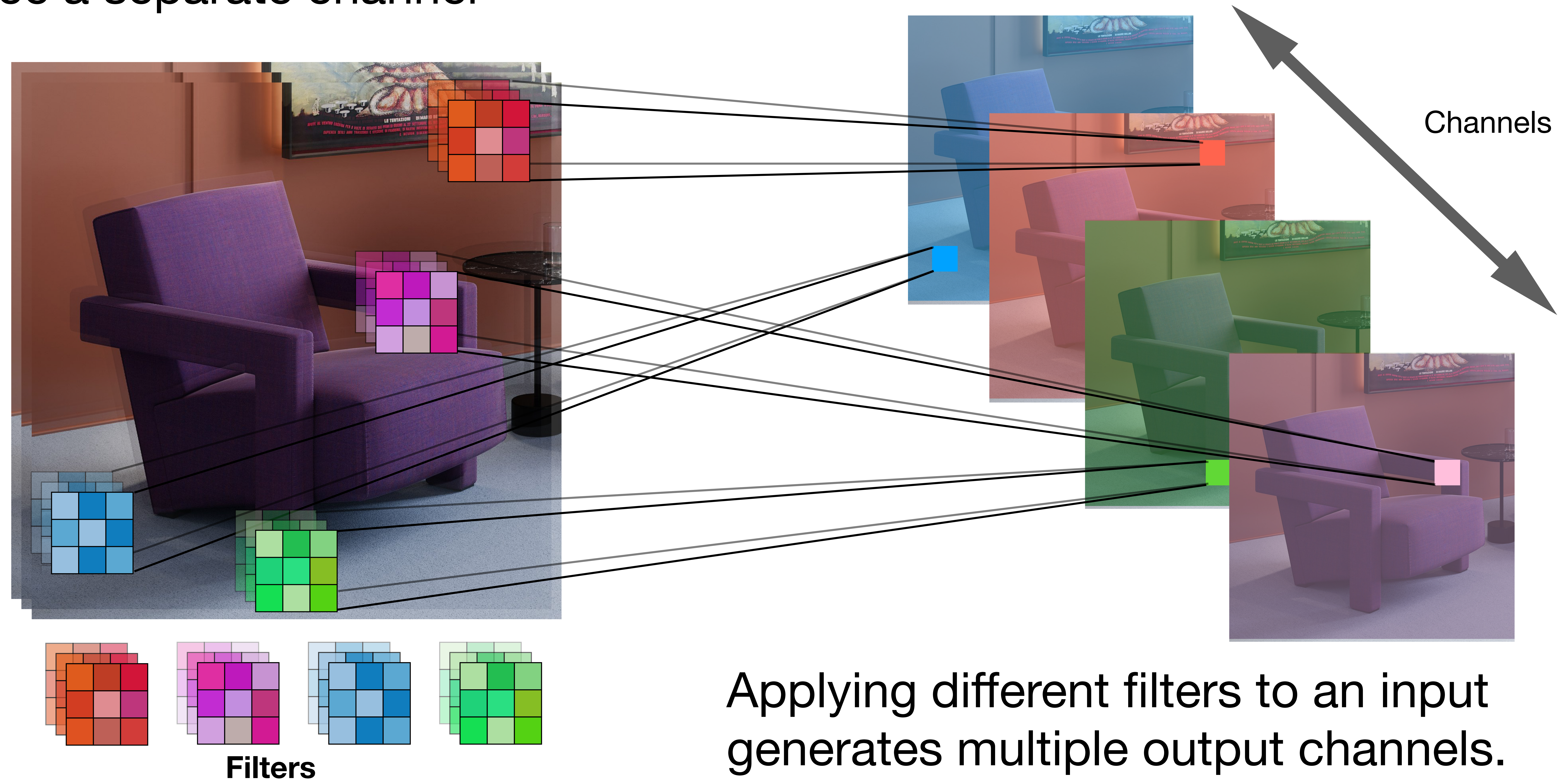
Filter for Multi-Channel Convolution



- For multi-channel inputs, the filter has the same number of channels as the input
- The filter channels and the bias are the learnable parameters of the filter

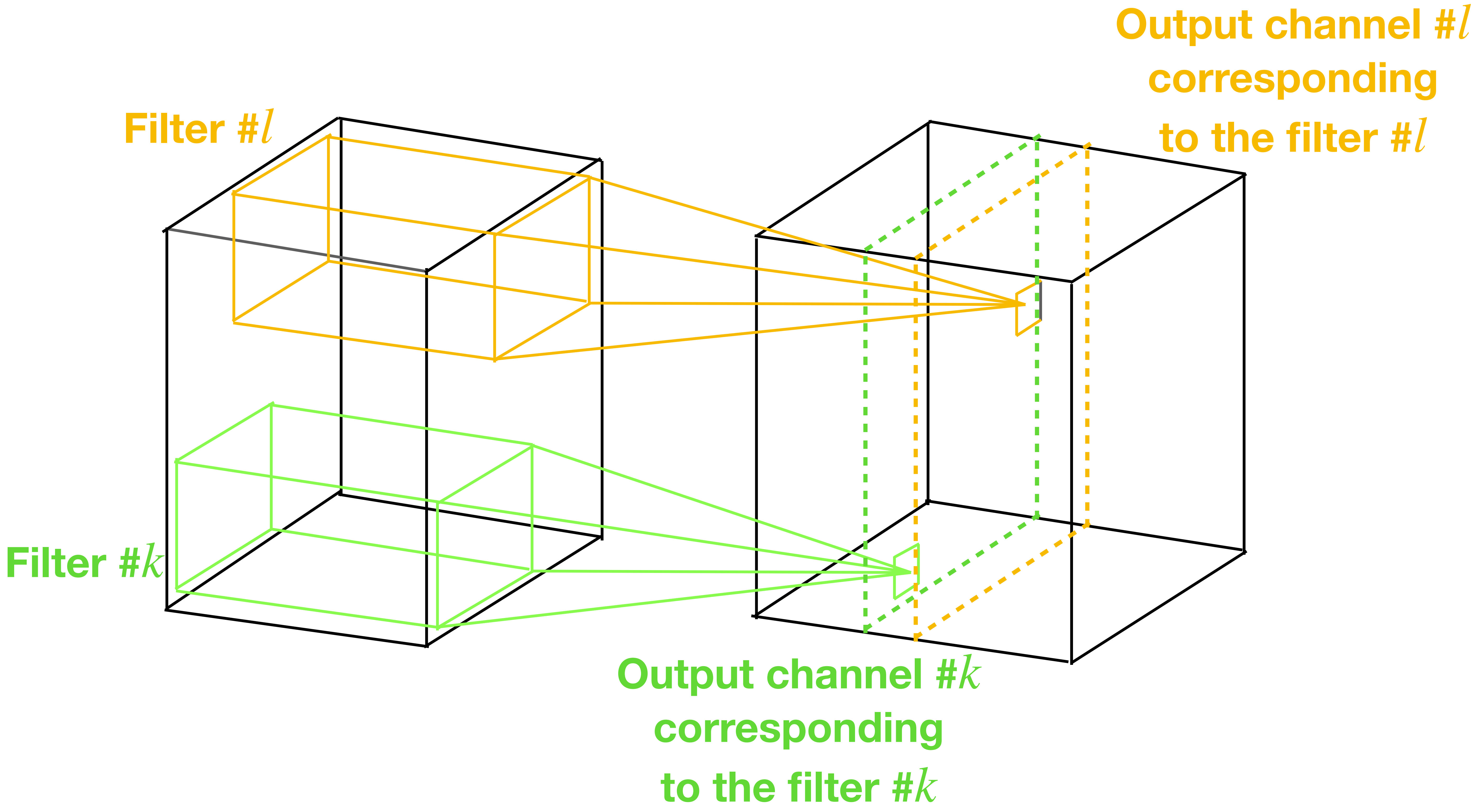
Multi-Channel Output from Multiple Filters

It is common to use multiple filters. Each filter processes the input to produce a separate channel



Applying different filters to an input generates multiple output channels.

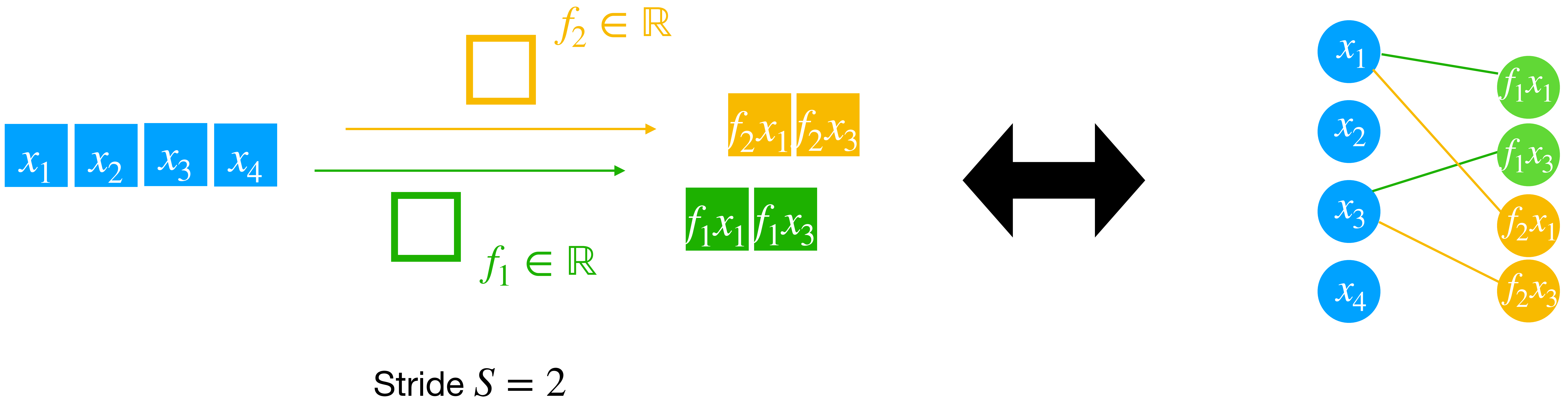
Convolutional Layer



- A convolutional layer is composed of multiple filters
- Each output channel corresponds to its own independent filter
- Hyper-parameters of the convolutional layer: size, padding, stride

Equivalence between Convolution and Fully Connected Layers

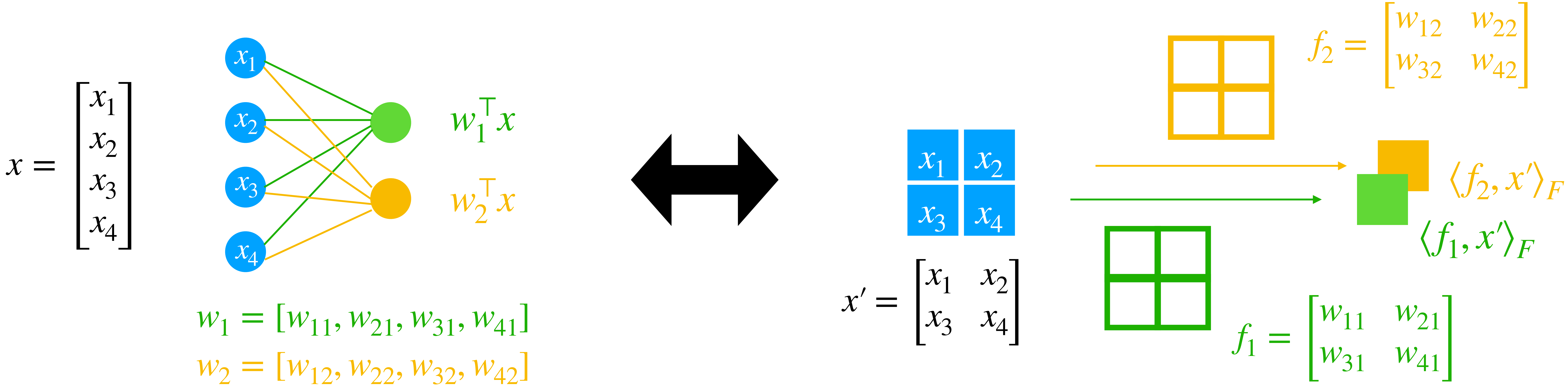
- For any Convolution layer there is an Fully Connected (FC) layer that implements the same forward function:
  - The FC weight matrix would be a large matrix that is mostly zero except for at certain blocks (due to local connectivity) where the weights in many of the blocks are equal (weight sharing).



Equivalence between Convolution and Fully Connected Layers

- For any Convolution layer there is an Fully Connected (FC) layer that implements the same forward function:
    - The FC weight matrix would be a large matrix that is mostly zero except for at certain blocks (due to local connectivity) where the weights in many of the blocks are equal (weight sharing).

- Conversely, any FC layer can be converted to a Convolution layer



Pooling: often applied after the convolutional layer

Max pooling: returns the maximum value of the portion of the convolved feature that is covered by the kernel

3	5	8	10	9	4
7	10	2	6	2	8
7	3	5	1	7	0
1	2	6	1	9	2
3	1	2	2	5	4
4	8	2	1	4	3

Convolved feature

10		

2×2 **max pooling**

Average pooling: returns the average value of the portion of the convolved feature that is covered by the kernel

3	5	8	10	9	4
7	10	2	6	2	8
7	3	5	1	7	0
1	2	6	1	9	2
3	1	2	2	5	4
4	8	2	1	4	3

Convolved feature

6.25		

2×2 **average pooling**

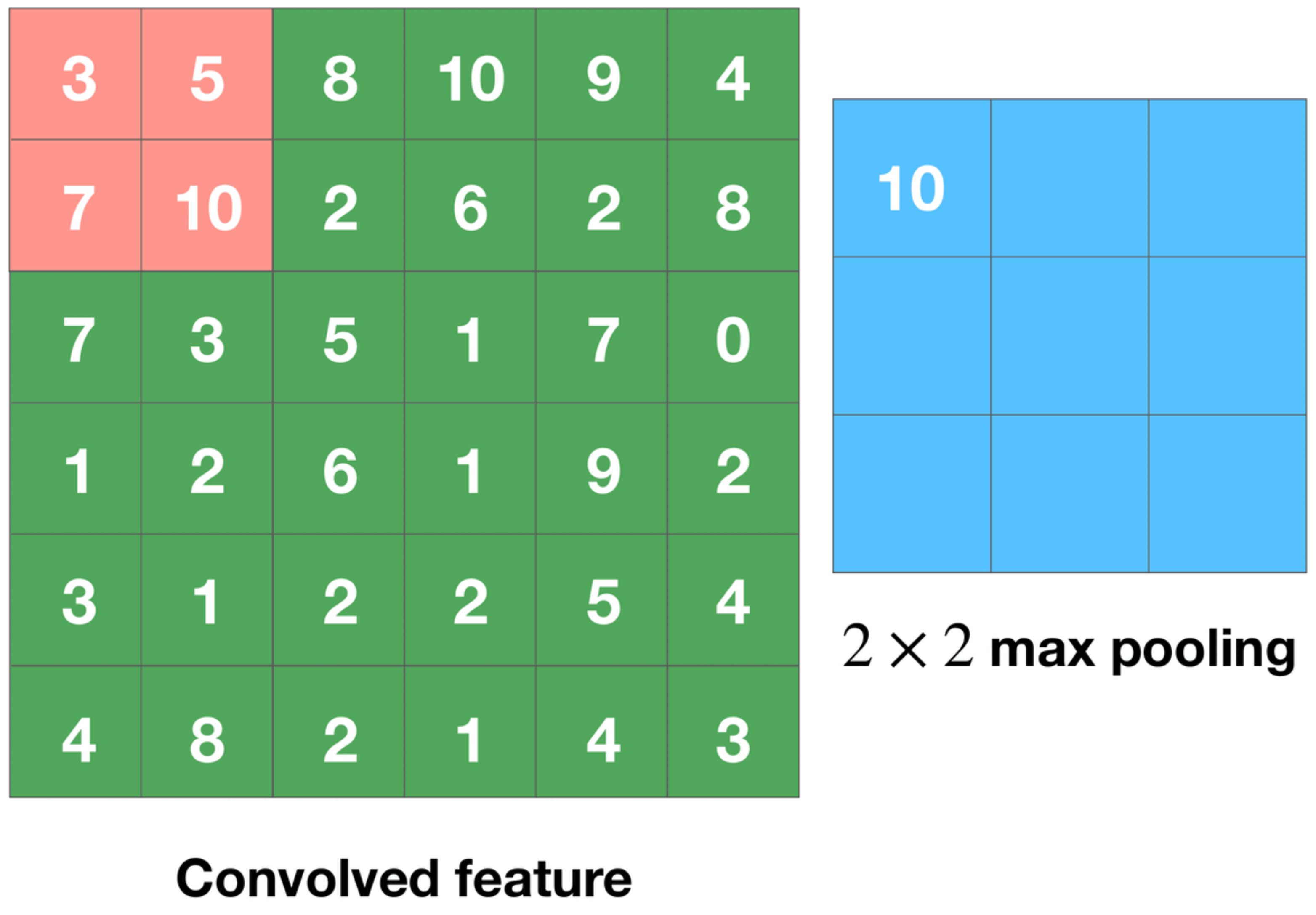
Pooling is a downsampling operation that reduces the spatial dimensions of the convolved feature

Remark: Pooling layers do not have learnable parameters

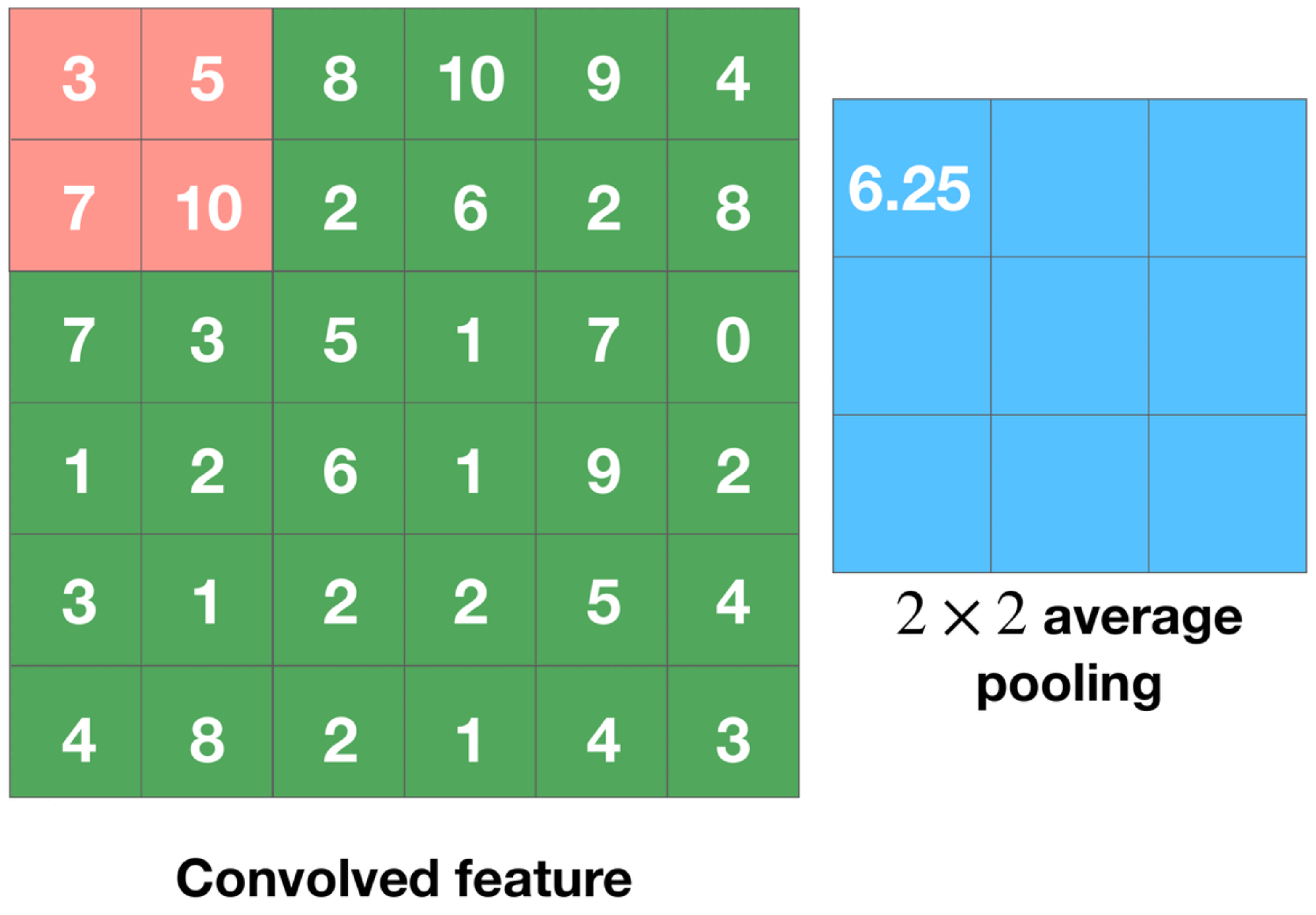
Hyperparameters are the size, type, and stride of the pooling operation **(Video)**

Pooling: often applied after the convolutional layer

Max pooling: returns the maximum value of the portion of the convolved feature that is covered by the kernel



Average pooling: returns the average value of the portion of the convolved feature that is covered by the kernel

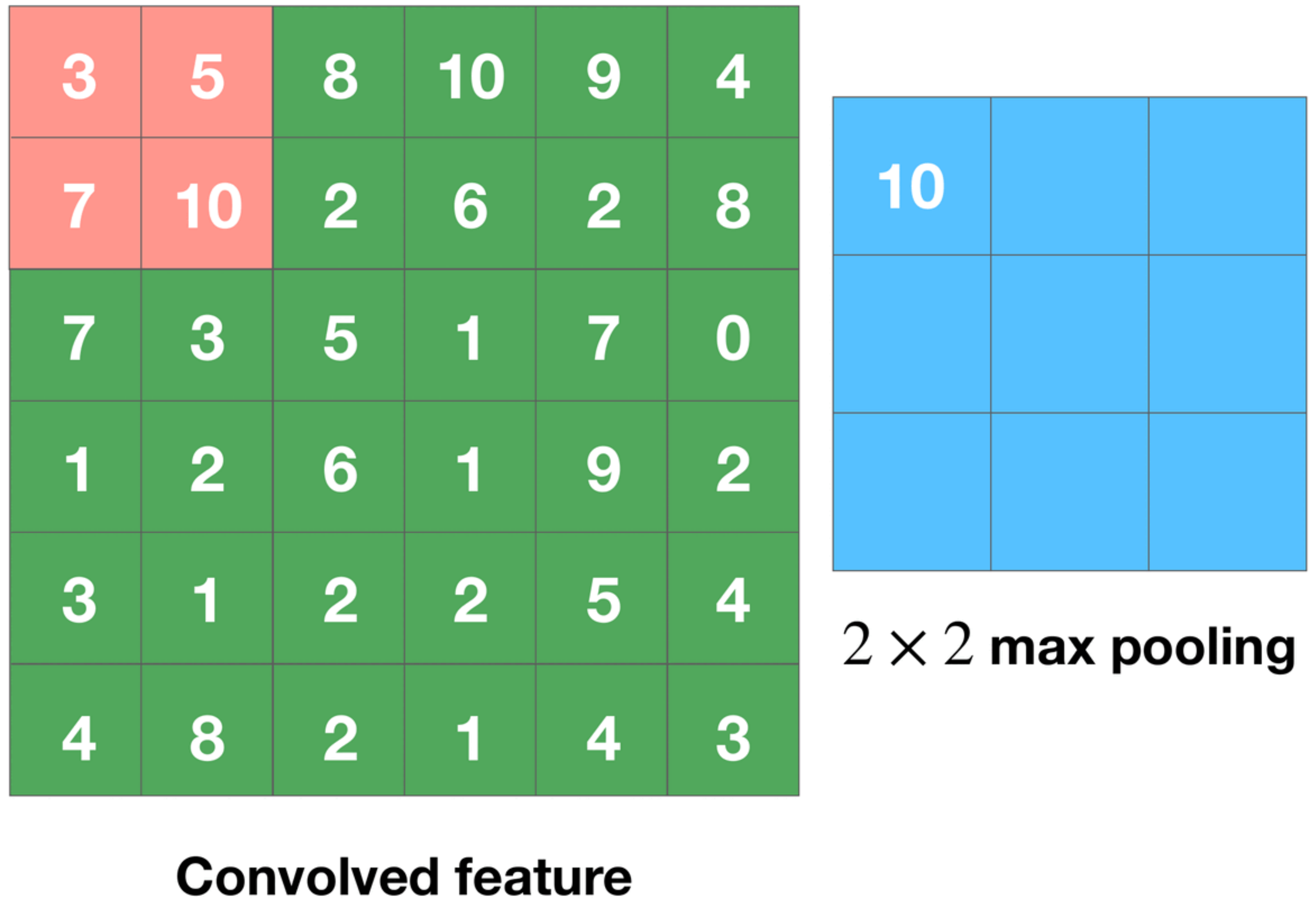


Pooling is a downsampling operation that reduces the spatial dimensions of the convolved feature

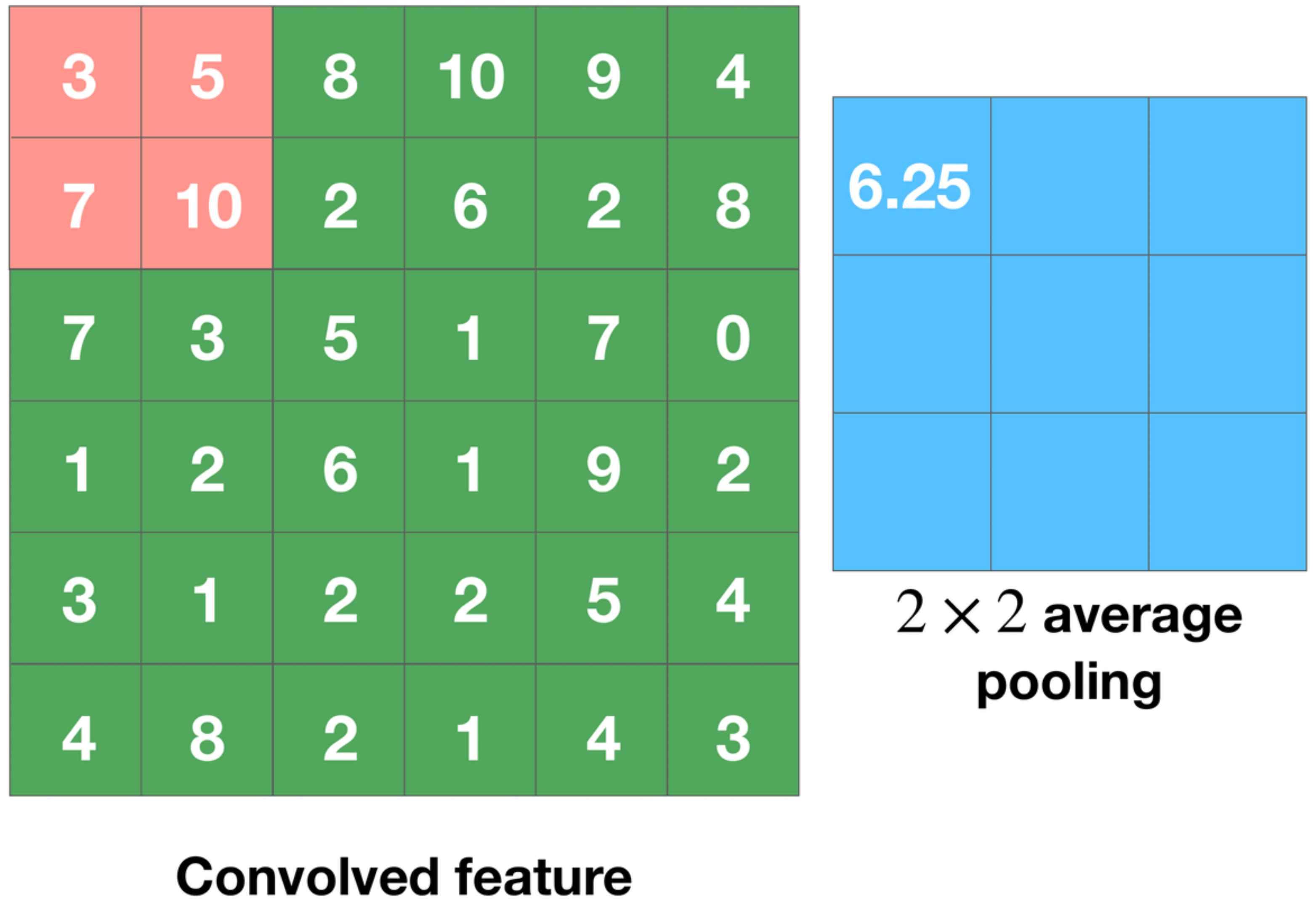
Remark: Pooling layers do not have learnable parameters
Hyperparameters are the size, type, and stride of the pooling operation (Video)

Pooling: often applied after the convolutional layer

<u>Max pooling</u>: returns the maximum value of the portion of the convolved feature that is covered by the kernel



<u>Average pooling</u>: returns the average value of the portion of the convolved feature that is covered by the kernel

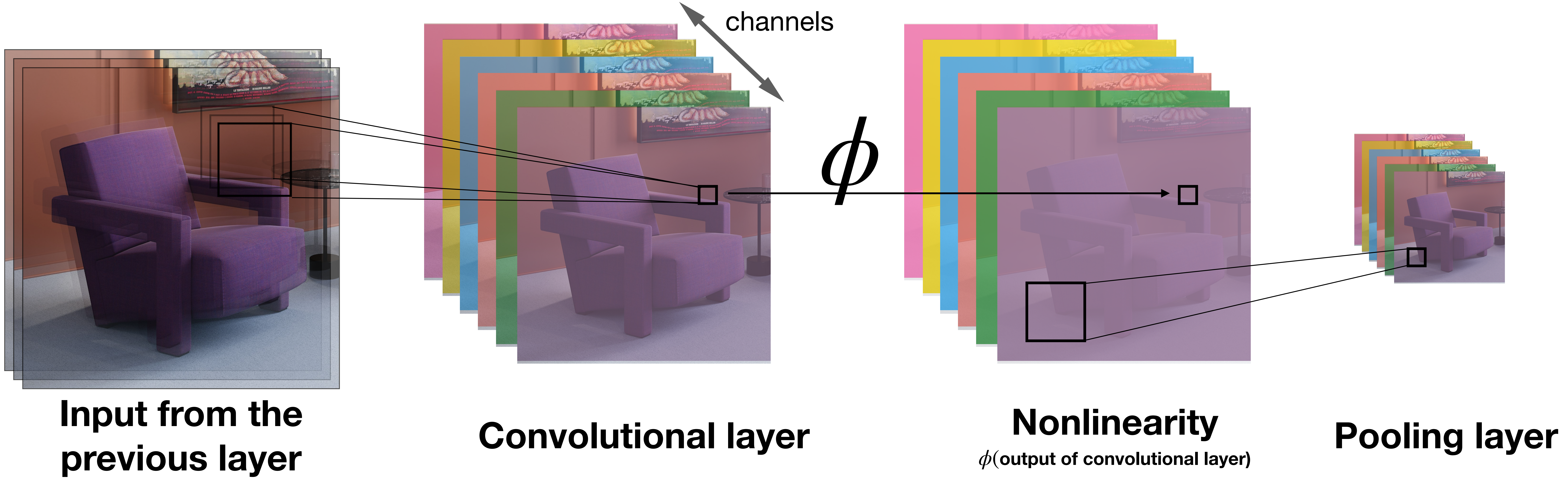


Pooling is a downsampling operation that reduces the spatial dimensions of the convolved feature

<u>Remark:</u> Pooling layers do not have learnable parameters

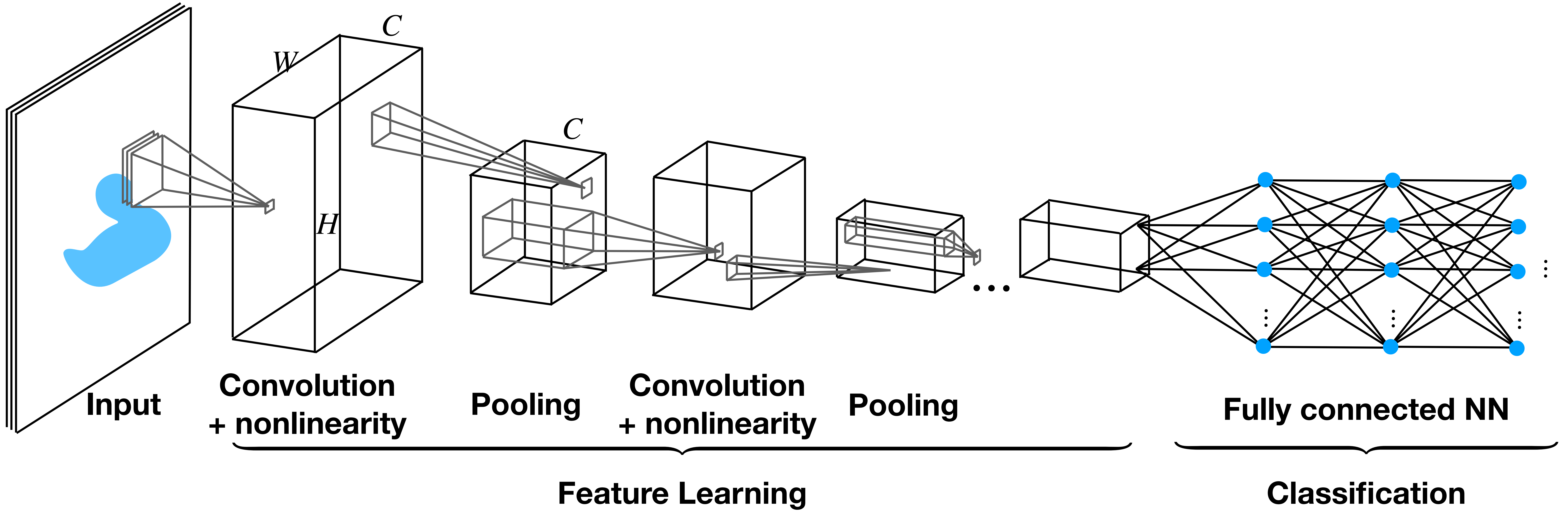
Hyperparameters are the size, type, and stride of the pooling operation (Video)

Non-linearity and convolutional NNs



Important: A non-linearity such as ReLU is included after each convolutional layer to make the model non-linear

Convolutional NNs: General Structure



Receptive field (area of input that affects a given neuron) increases with depth:

- First layers extract low-level features, e.g., edges, colors
- Subsequent layers extract high-level features, e.g., objects

➡ ConvNet reduces the images to a form easier to process without losing essential features

Backpropagation with weight sharing

Weight sharing is used in CNN: many edges use the same weights

Training:

1. Run backpropagation as if the weights were not shared (treat each weight as an independent variable)
2. Once the gradient is computed, sum the gradients of all edges that share the same weight

Why: let $f(x, y, z) : \mathbb{R}^3 \to \mathbb{R}$ and $g(x, y) = f(x, y, x)$

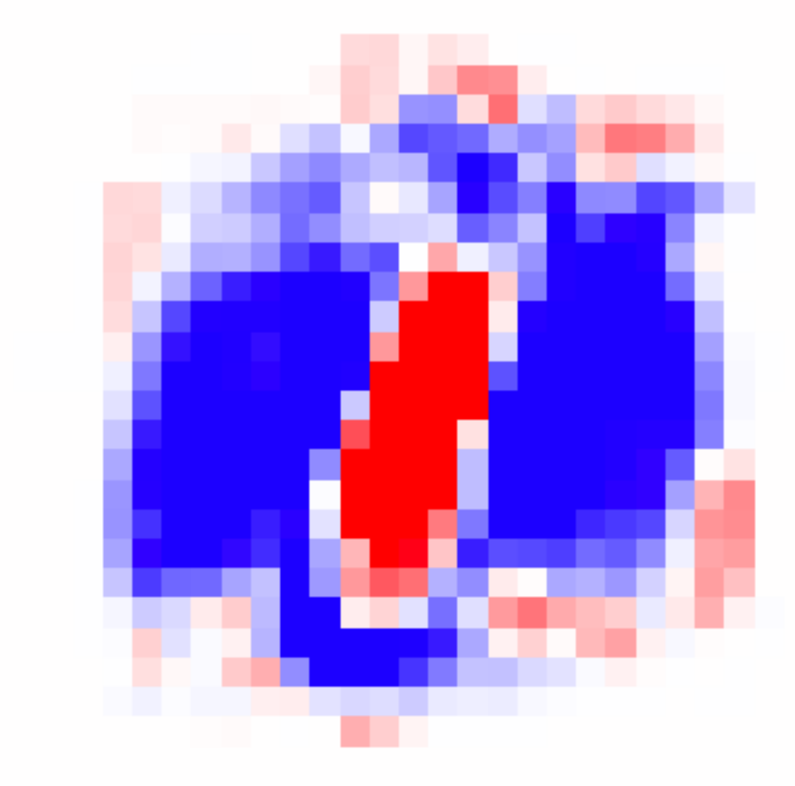
$$\left(\frac{\partial g}{\partial x}(x, y), \frac{\partial g}{\partial y}(x, y)\right) \underset{\text{Chain rule}}{=} \left(\frac{\partial f}{\partial x}(x, y, x) + \frac{\partial f}{\partial z}(x, y, x), \frac{\partial f}{\partial y}(x, y, x)\right)$$

What do ConvNets learn?

Learned Convolutional Filters on MNIST

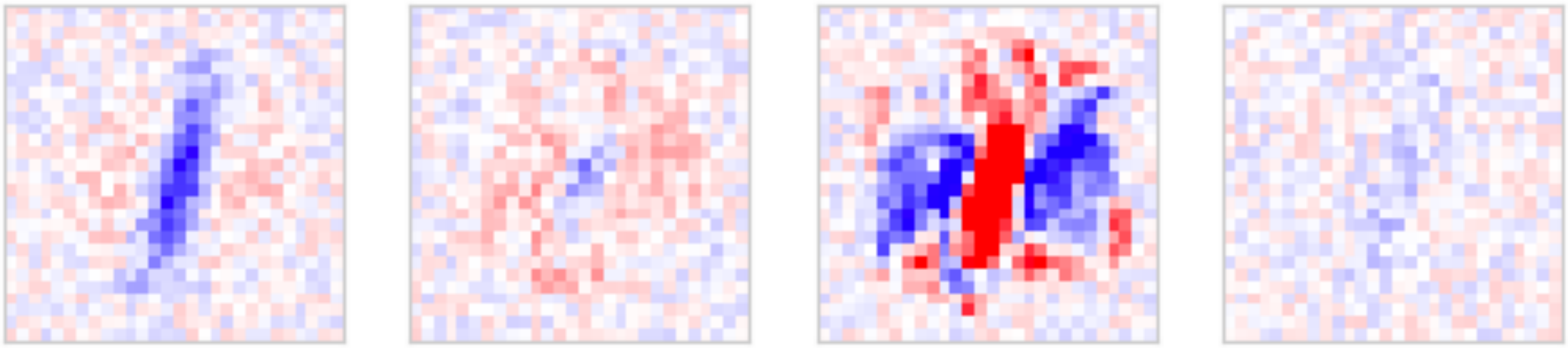
Linear model

parameter vector w



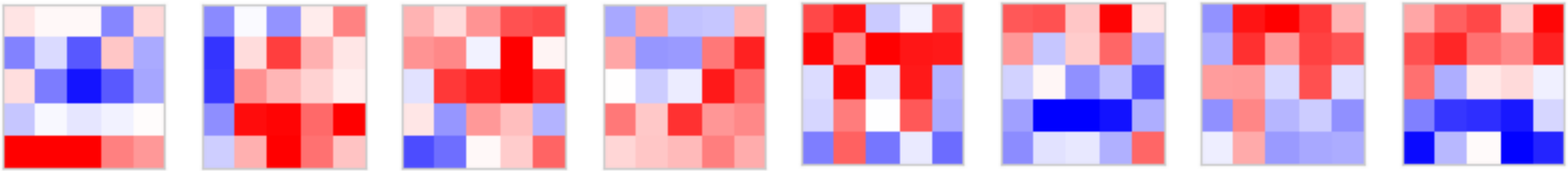
Fully-connected network

first-layer vector $w_{:,1}$ | first-layer vector $w_{:,2}$ | first-layer vector $w_{:,3}$ | first-layer vector $w_{:,4}$



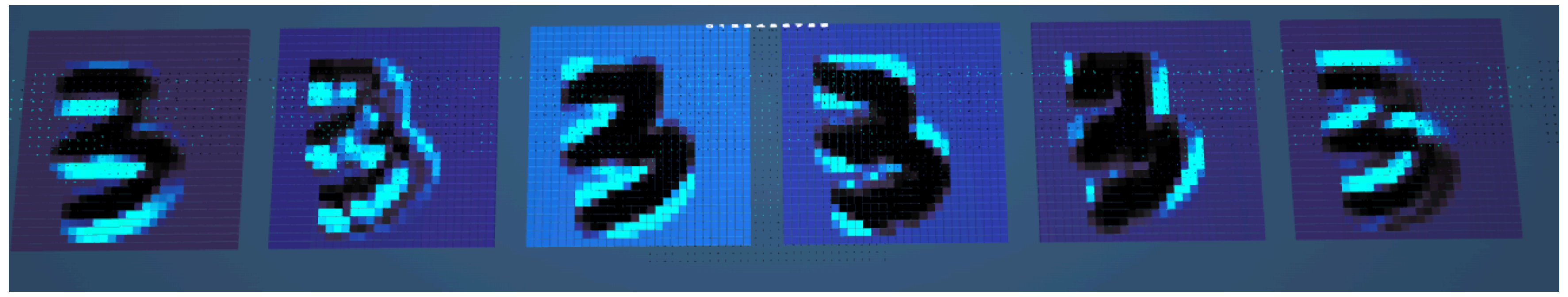
Convolutional networks

filter $f^{(1)}$ | filter $f^{(2)}$ | filter $f^{(3)}$ | filter $f^{(4)}$ | filter $f^{(5)}$ | filter $f^{(6)}$ | filter $f^{(7)}$ | filter $f^{(8)}$

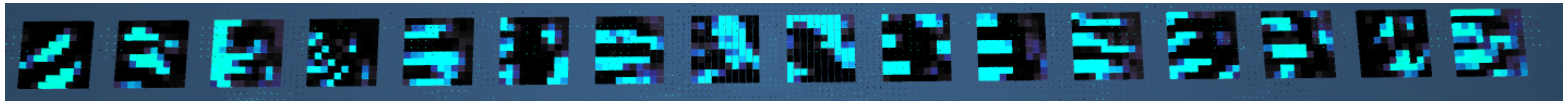


Layer-wise Activations on Hand-written Digits

Input



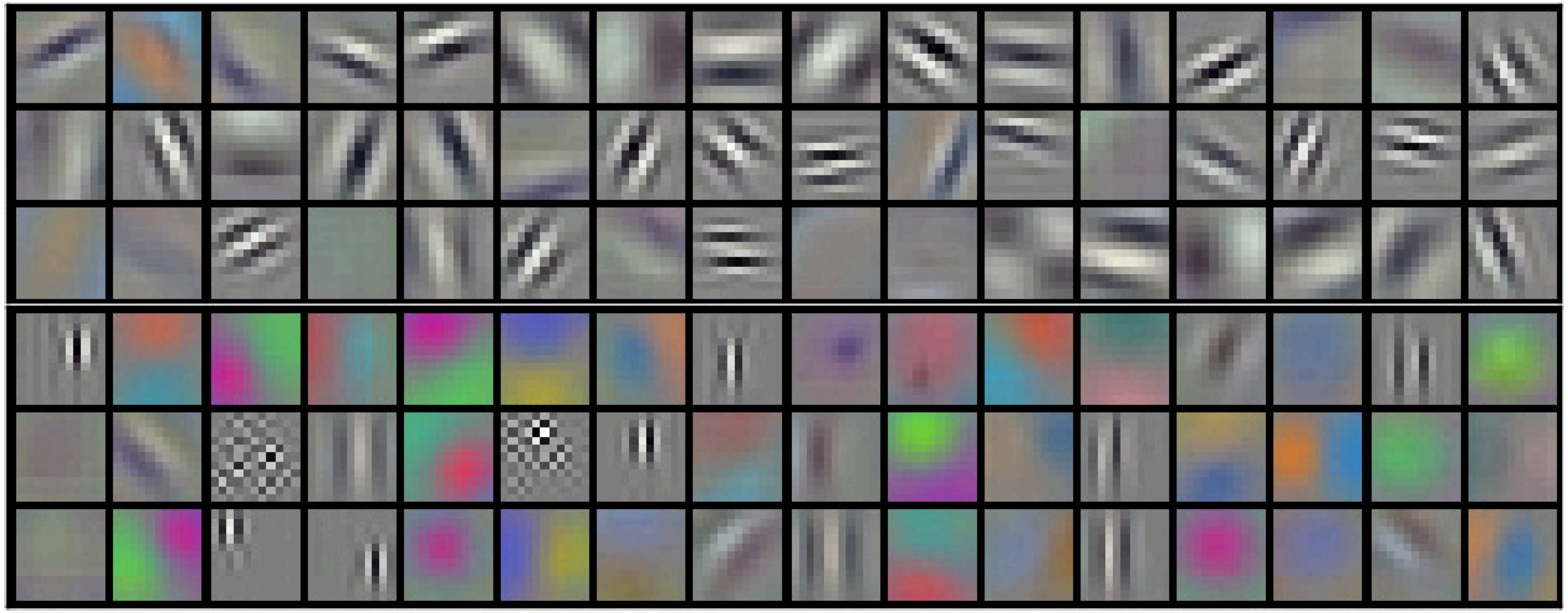
Conv Layer 1



Conv Layer 2

➡ Explore the visualization on your own: https://adamharley.com/nn_vis/cnn/3d.html

Learned Convolutional Filters on ImageNet



Source: ImageNet Classification with Deep Convolutional Neural Networks (NeurIPS 2012)

- Filters of the **first** layer can be interpretable
- Edge and color detectors typically emerge when trained on large datasets

Individual Activations Can Be Interpretable

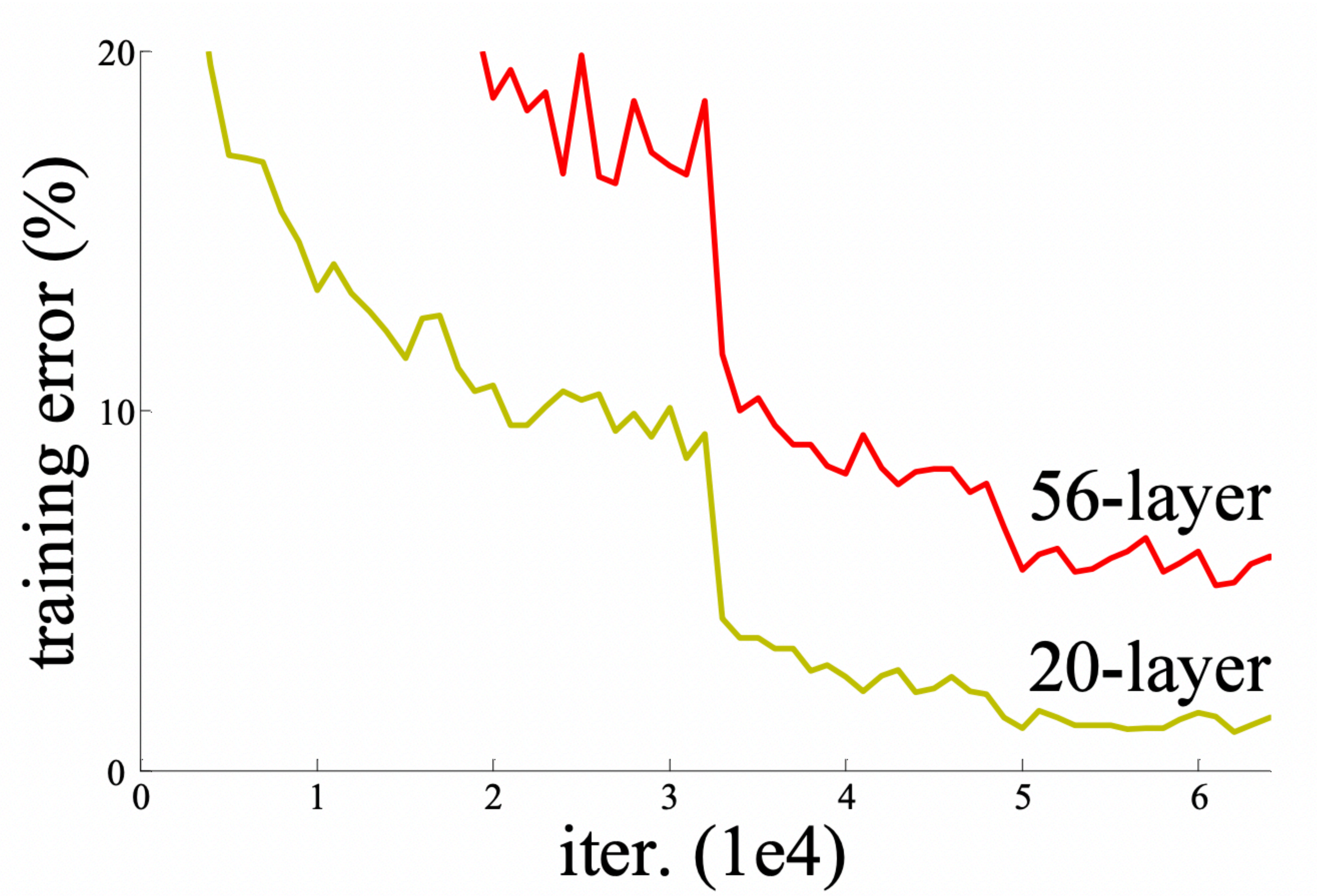
Source: Rich feature hierarchies for accurate object detection and semantic segmentation (CVPR 2014)

- Receptive fields and activation values are drawn in white
- Each activation detects some **pattern** or **object** (not always interpretable)
- Activations in later layers detect more complex patterns

Residual Networks

Skip Connections and Residuals

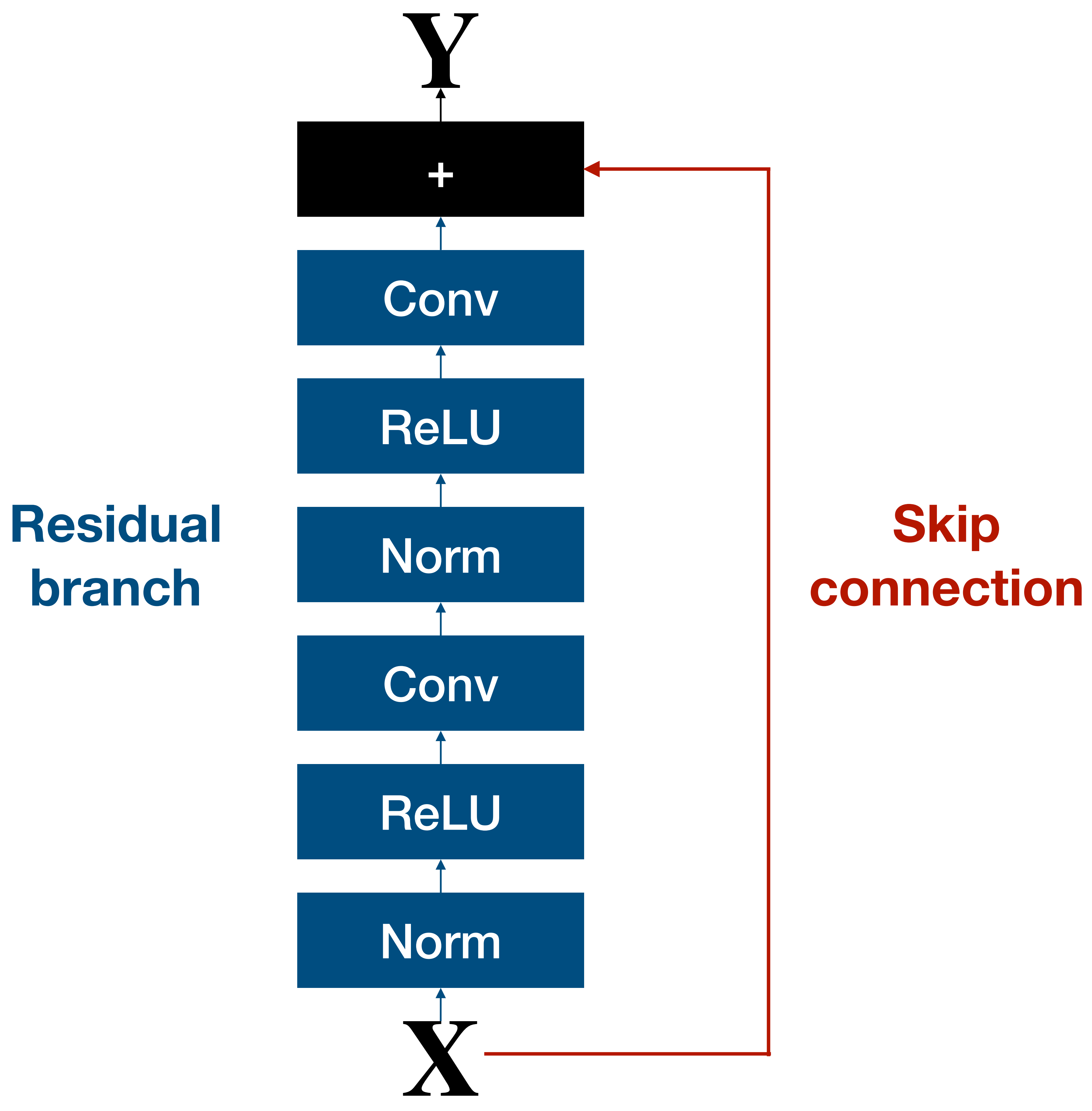
- **Starting point**: Adding more layers should lead to the same or lower training loss as they can learn the identity function
- **ResNet paper** indicates that this is not always the case
- **Solution**: add a *skip connection* around some layers $F(\mathbf{X})$
- **Standard network**: $\mathbf{Y} = F(\mathbf{X})$
- **Residual network**: $\mathbf{Y} = R(\mathbf{X}) + \mathbf{X}$ where $R(\mathbf{X})$ is called a residual branch



Observation from the ResNet paper: deeper CNNs are harder to train

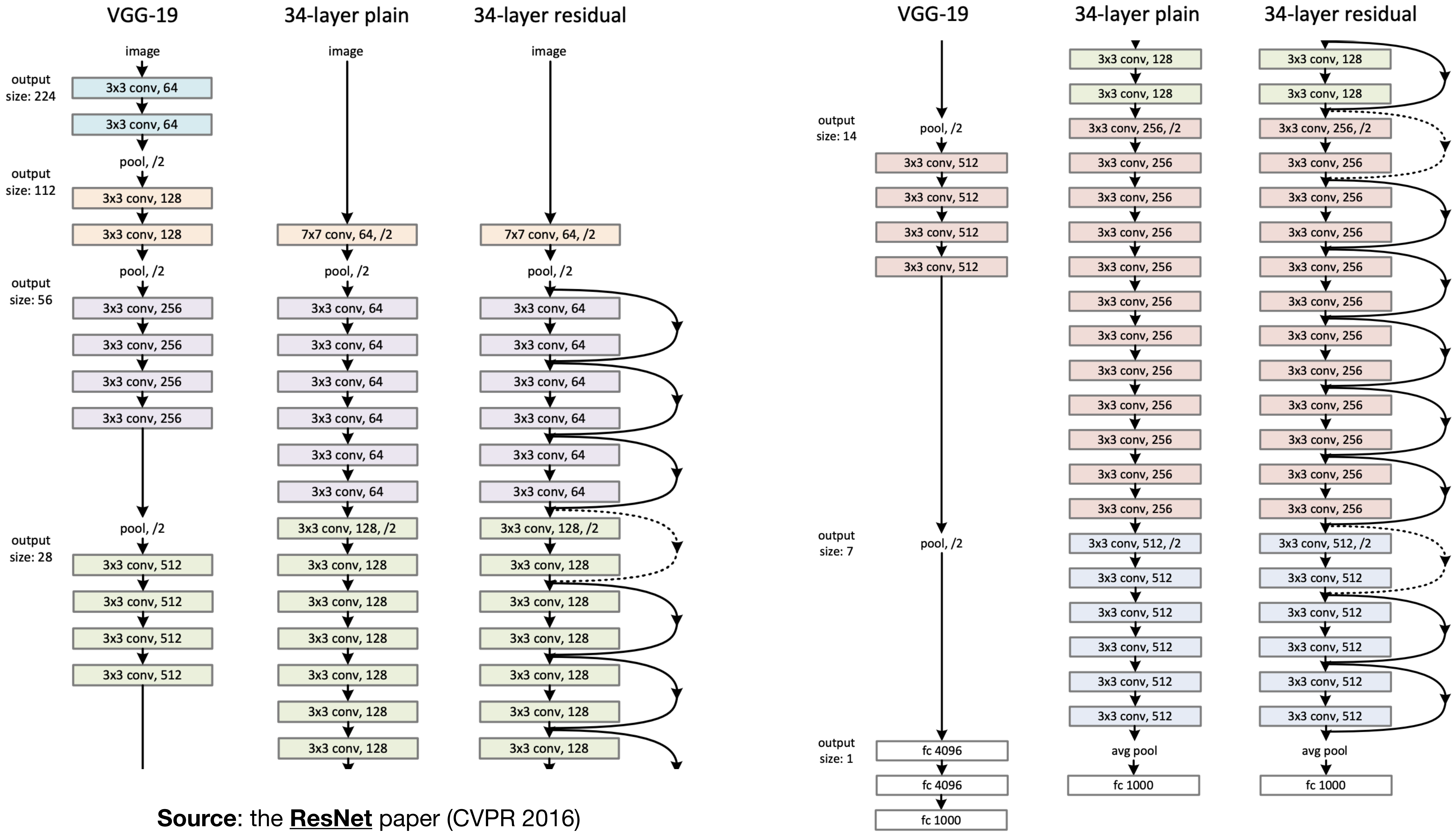
Skip Connections and Residuals

- **Example of** $R(\mathbf{X})$: see on the right
- **Technical detail**: If size$(\mathbf{Y}) \neq$ size$(\mathbf{X})$, additional operations are needed on the skip connection to match the dimensions
- Skip connections address the observed convergence issue, making the training of very deep networks (with hundreds of layers) feasible
- Skip connections are used in almost all modern neural networks (including CNNs and transformers)



Example residual block

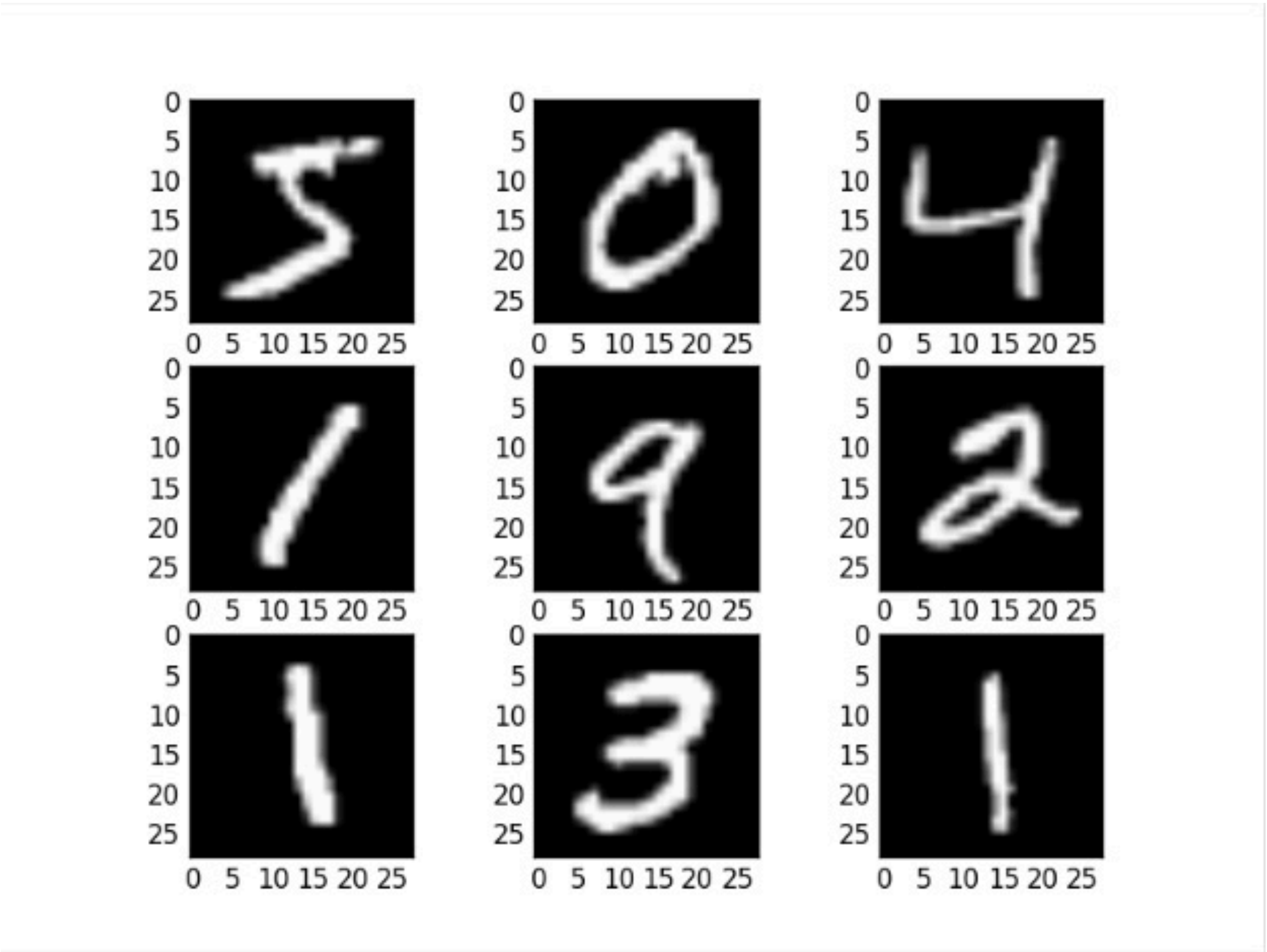
Popular architectures: VGG vs. ResNet



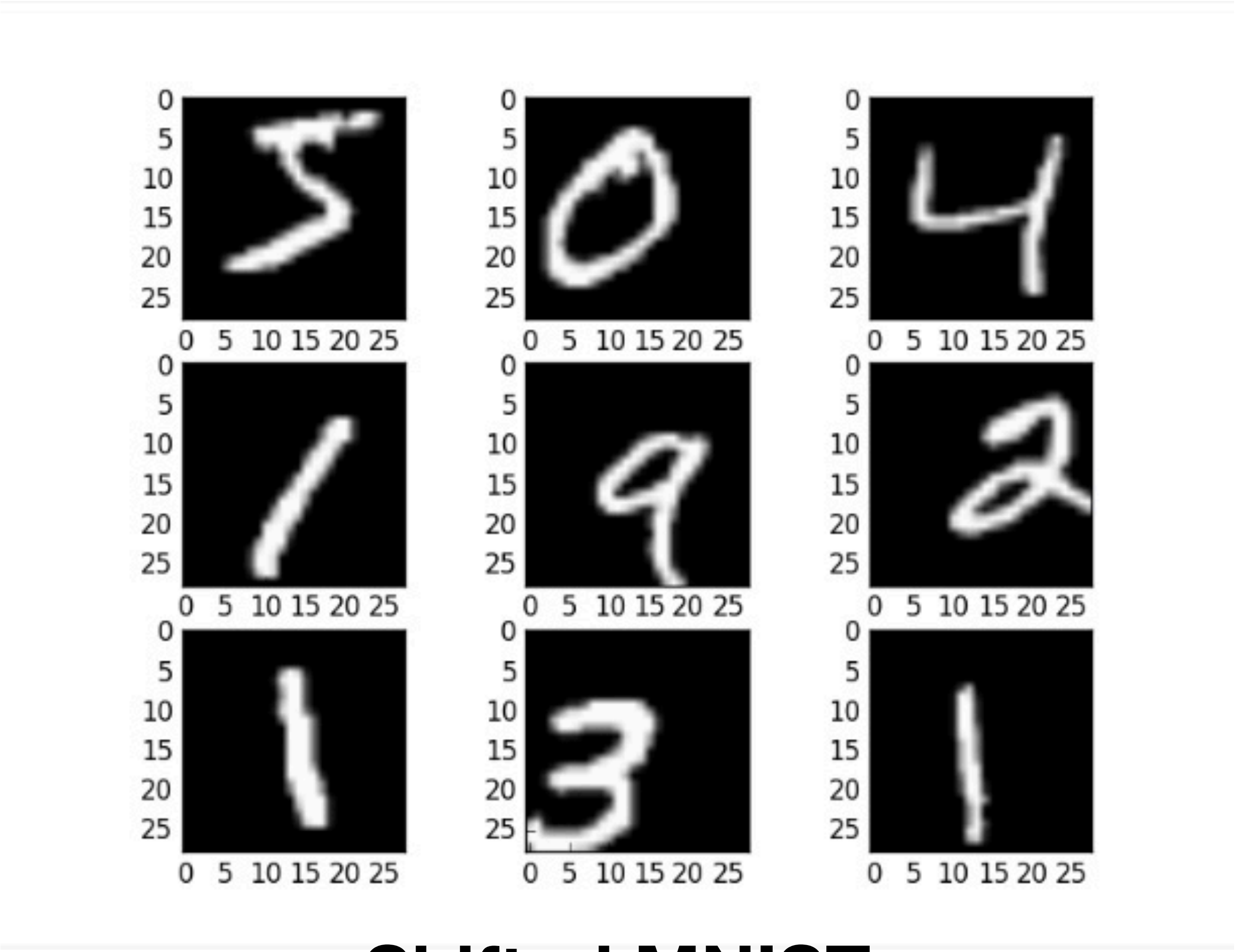
Source: the **ResNet** paper (CVPR 2016)

Data Augmentation

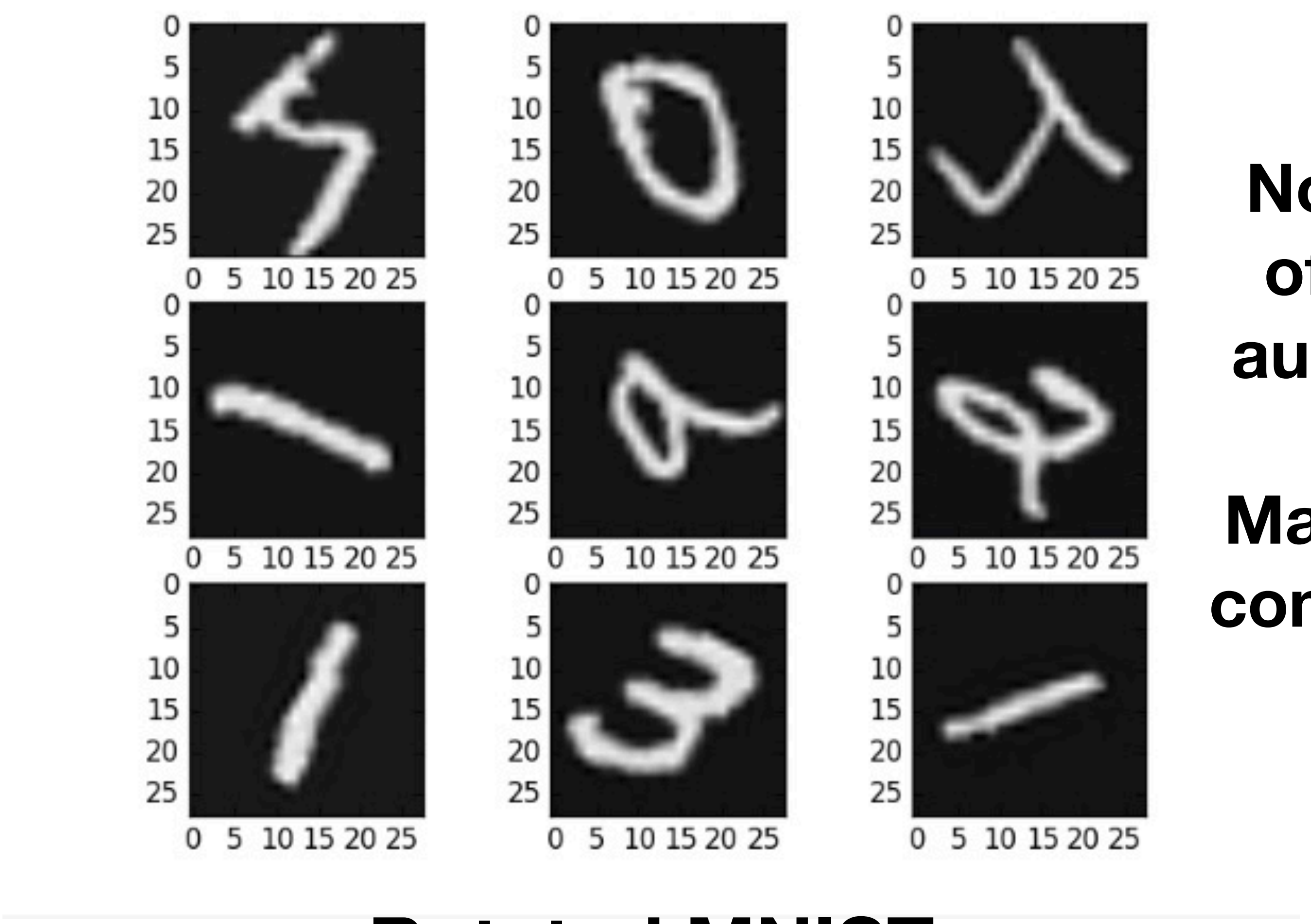
Data augmentation: generate new data from the data



MNIST



Shifted MNIST



Rotated MNIST

Note dangers of excessive augmentation!

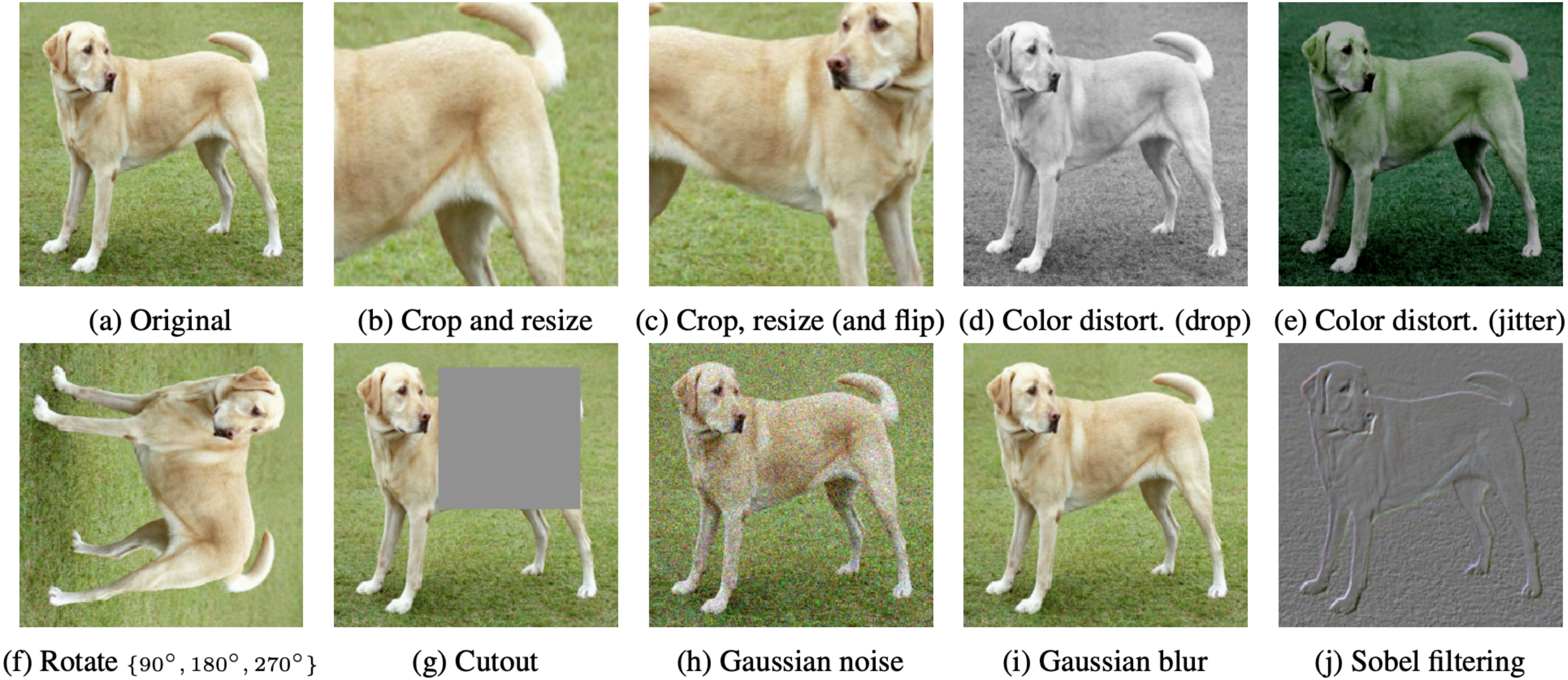
May eventually confuse 6 and 9

Transformation $\tau : \mathbb{R}^d \to \mathbb{R}^d$ which preserves the labels (i.e., $y_x = y_{\tau(x)}$)

$$S = S_{train} \cup \{(\tau(x_i), y_i)\}_{i=1}^{n}$$

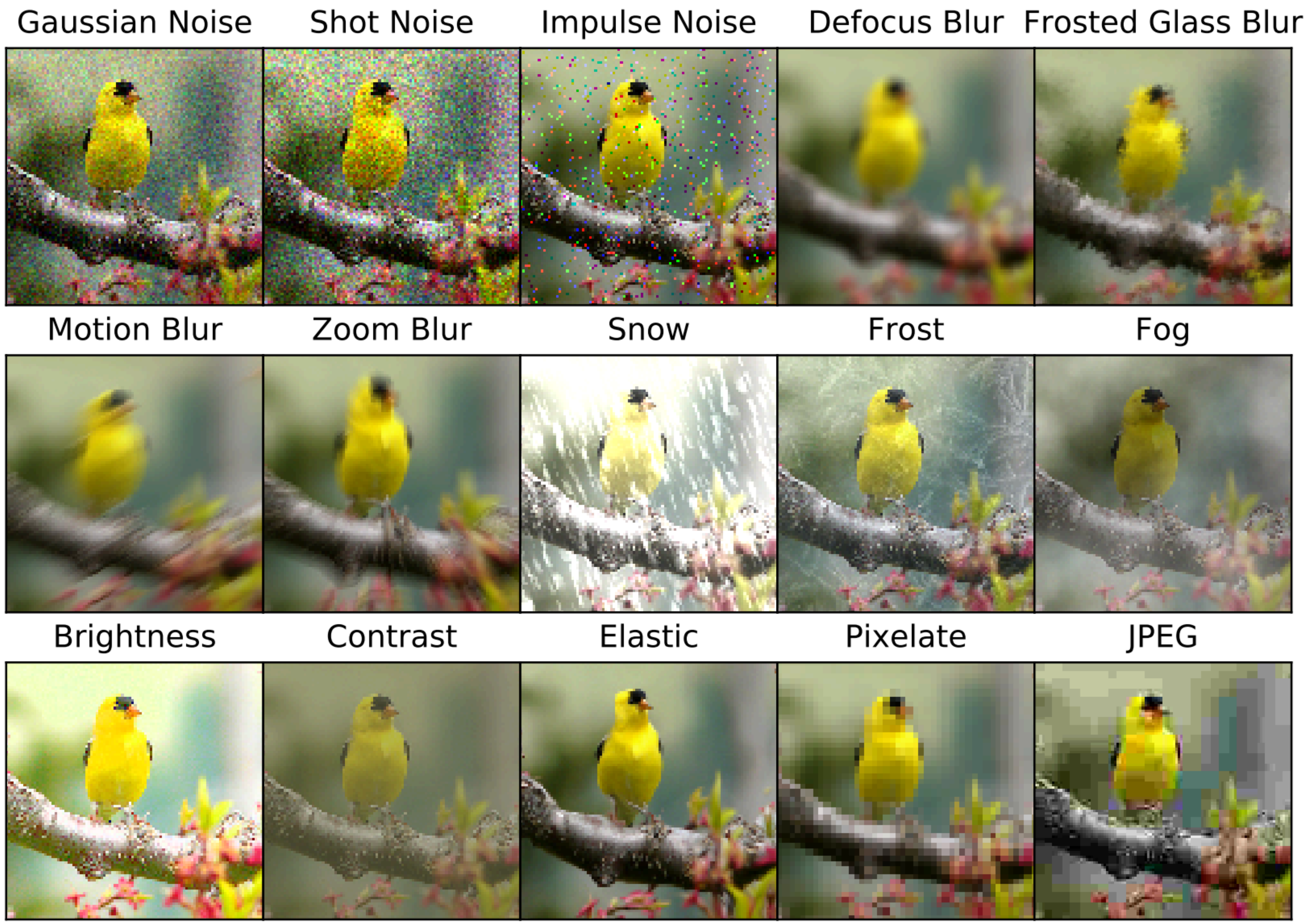
- We train on more data
- Encourages models to be invariant to τ
- It can be seen as regularization
- These transformations are task and dataset specific

Data augmentation: pictures can also be cropped, resized, or perturbed by a small amount of noise



T Chen, S Kornblith, M Norouzi, G Hinton. A simple framework for contrastive learning of visual representations. ICML, 2020

Data augmentation: generated corruptions



D. Hendrycks and T. Dietterich. Benchmarking Neural Network Robustness to Common Corruptions and Perturbations. ICLR, 2019

Weight Decay

Weight decay: ℓ_2-regularization for NNs

It is standard practice to regularize weights without regularizing bias terms:

$$\min \mathscr{L} + \frac{\lambda}{2} \sum_l \|\mathbf{W}^{(l)}\|_F^2$$

Weight decay [1] favors small weights which can aid in generalization and optimization

Optimization with gradient descent:

$$\left(w_{i,j}^{(l)}\right)_{t+1} = \left(w_{i,j}^{(l)}\right)_t - \eta \nabla \mathscr{L} - \eta\lambda\left(w_{i,j}^{(l)}\right)_t = \underbrace{(1-\eta\lambda)}_{\text{weight decay}}\left(w_{i,j}^{(l)}\right)_t - \eta \nabla \mathscr{L}$$

Interaction with BatchNorm:

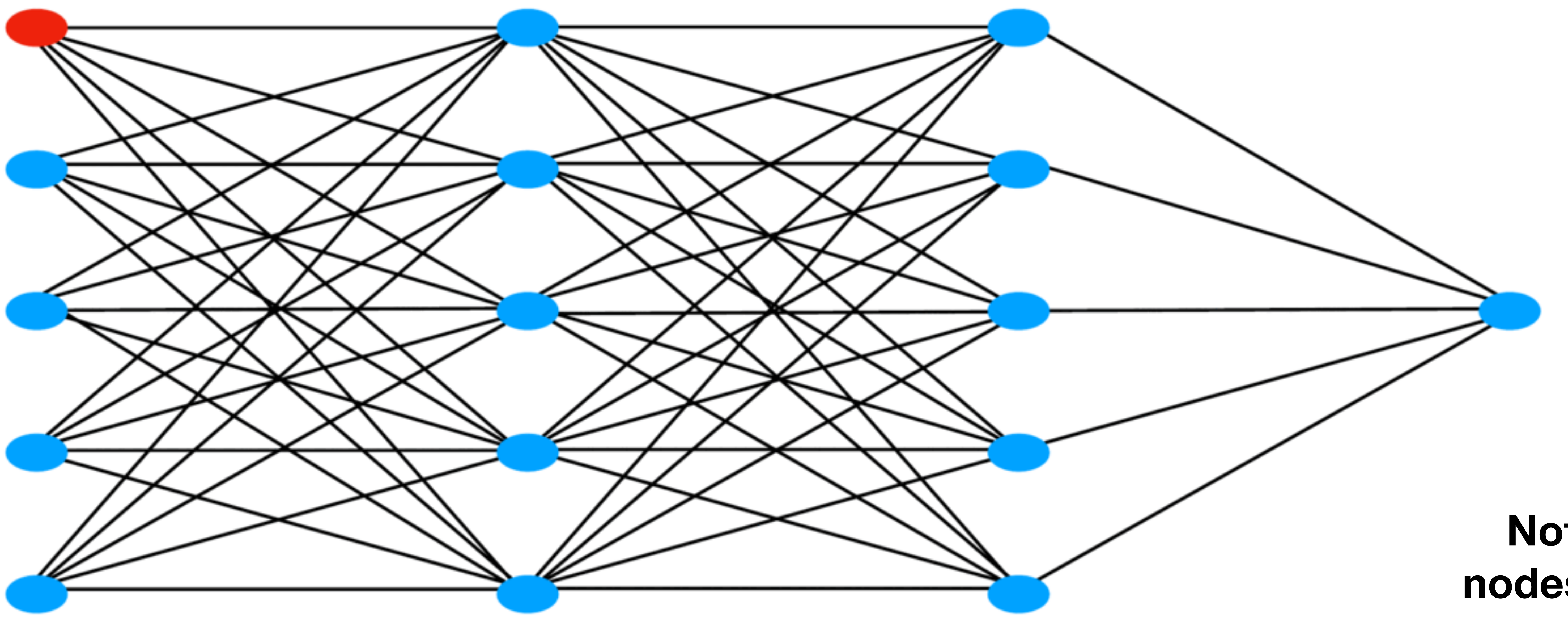
- BN$(\mathbf{WX})$ = BN$(\alpha\mathbf{WX})$ for $\alpha \in \mathbb{R}_{>0}$ (assuming $\varepsilon \approx 0$)
- BN is scale invariant in $\mathbf{W}$, hence there is **no direct regularization effect from WD**
- However, the training dynamics differ [2]

[1] A. Krogh and J. A Hertz. A simple weight decay can improve generalization. NeurIPS, 1992
[2] R. Wan, et al. Spherical Motion Dynamics: Learning Dynamics of Normalized Neural Network using SGD and Weight Decay. NeurIPS, 2021

Dropout

Dropout: randomly drop nodes

<u>Def:</u> At each training step, retain with probability $p^{(l)}$ each node in layer (l):



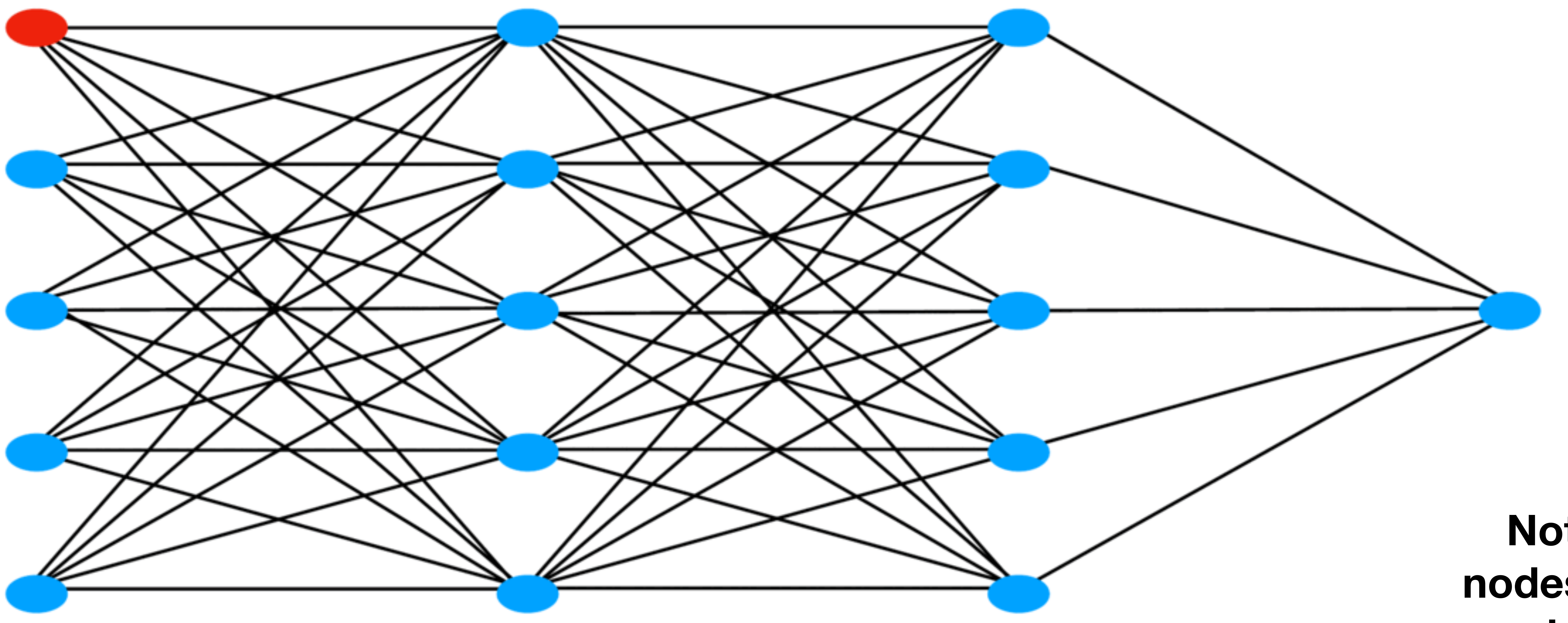
Note: In practice we drop nodes independently for each element of a mini-batch

N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, R. Salakhutdinov. Dropout: A Simple Way to Prevent Neural Networks from Overfitting, JMLR, 2014

(Video)

Dropout: randomly drop nodes

Def: At each training step, retain with probability $p^{(l)}$ each node in layer (l):



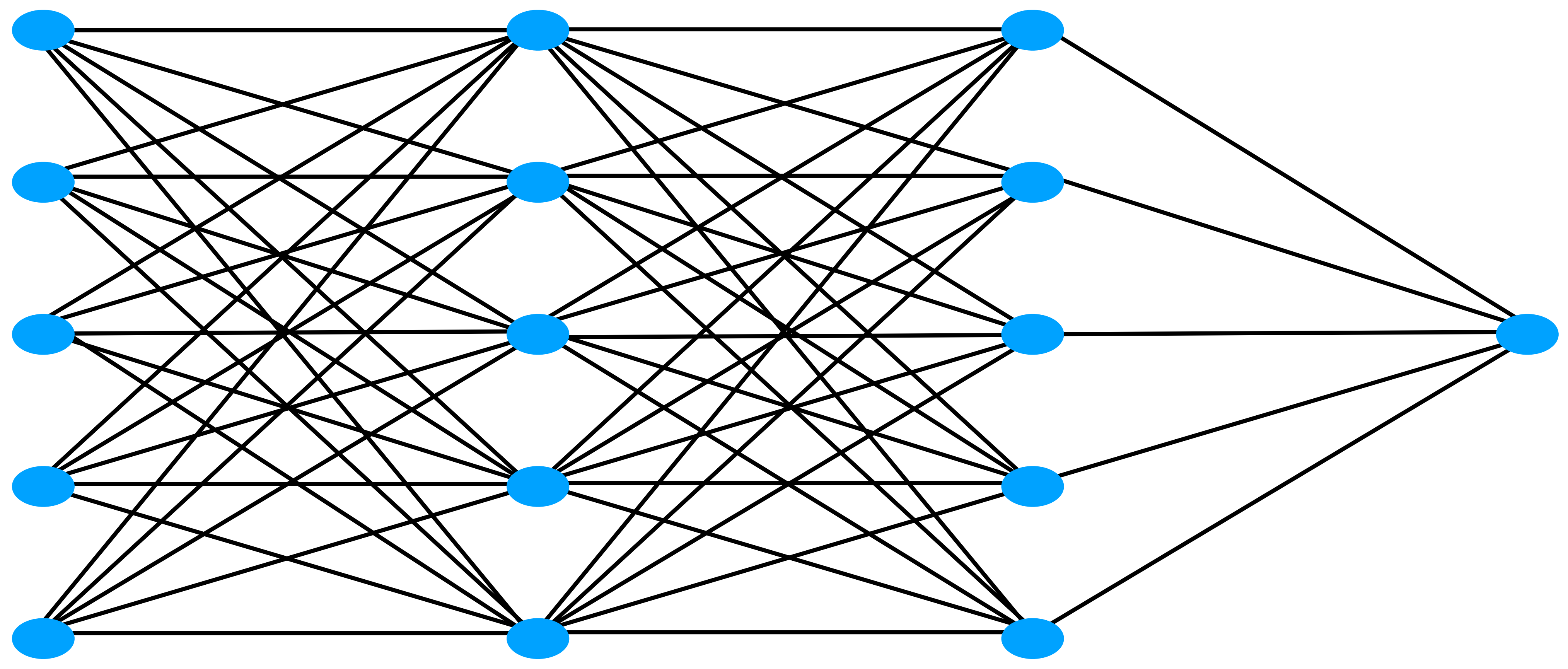
Note: In practice we drop nodes independently for each element of a mini-batch

N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, R. Salakhutdinov. Dropout: A Simple Way to Prevent Neural Networks from Overfitting, JMLR, 2014

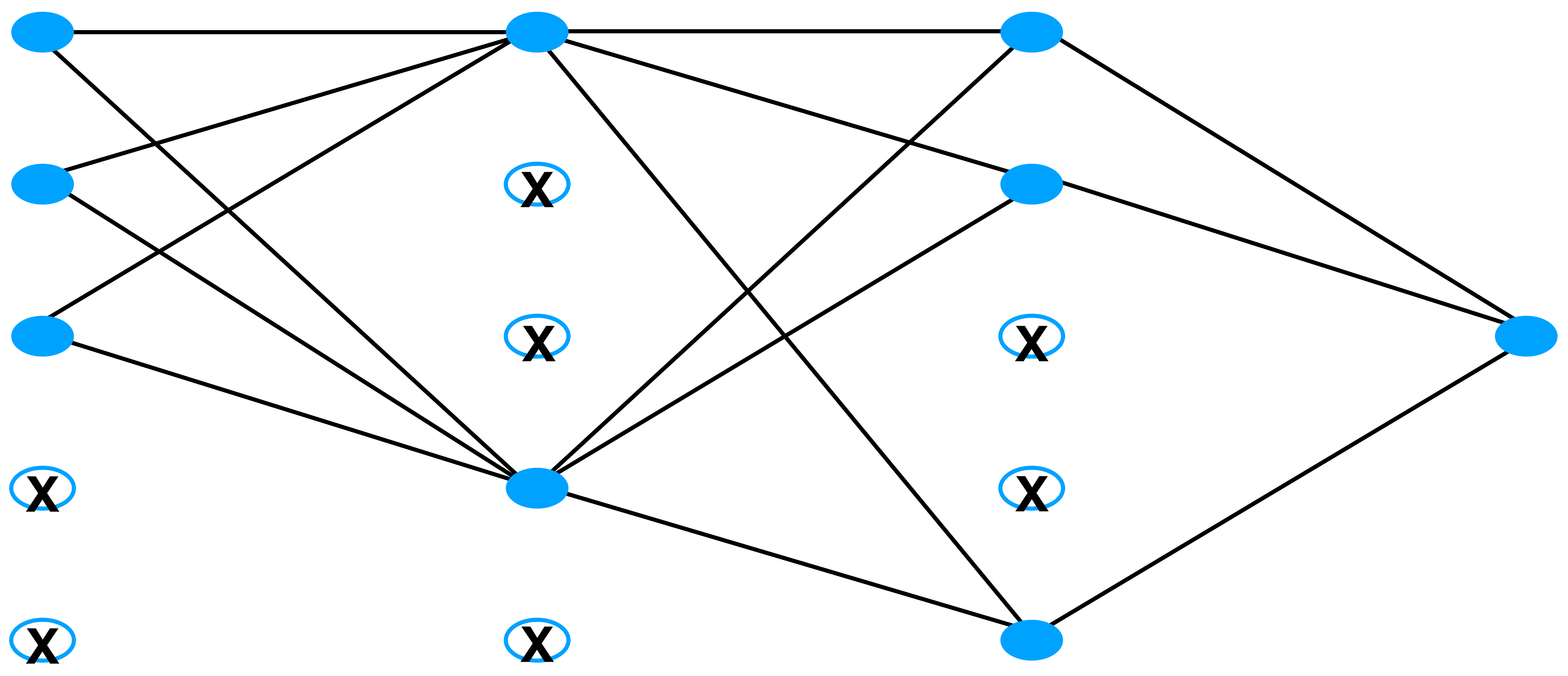
(Video)

Dropout: training phase

<u>Def:</u> At each training step, retain with probability $p^{(l)}$ each node in layer (l):



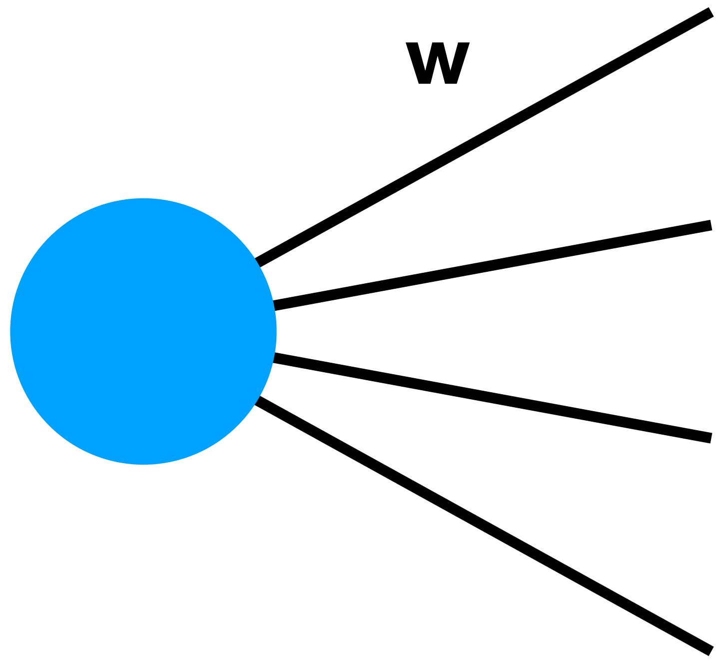
Original network



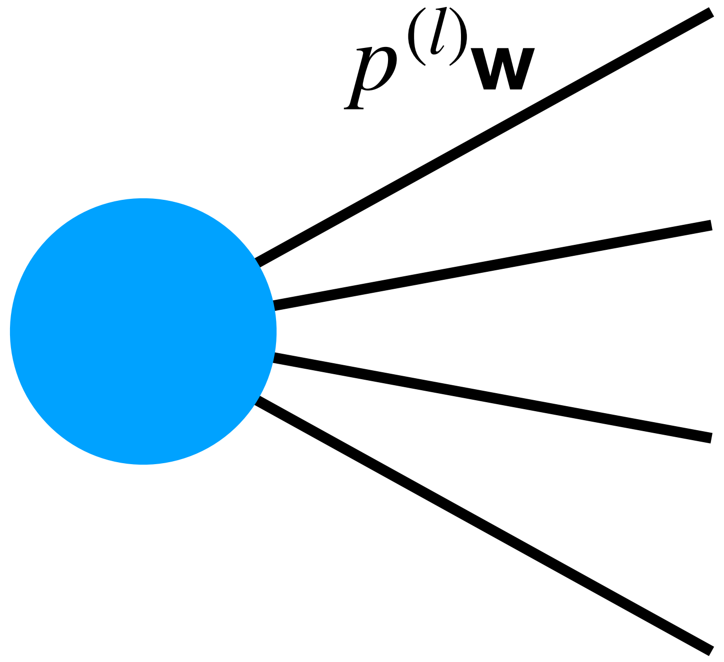
Random subnetwork

Run one step of SGD on the subnetwork and update the weights

Dropout: testing phase



At training time



At test time

Note: Variance is generally not preserved and as a result Dropout often works poorly with normalization

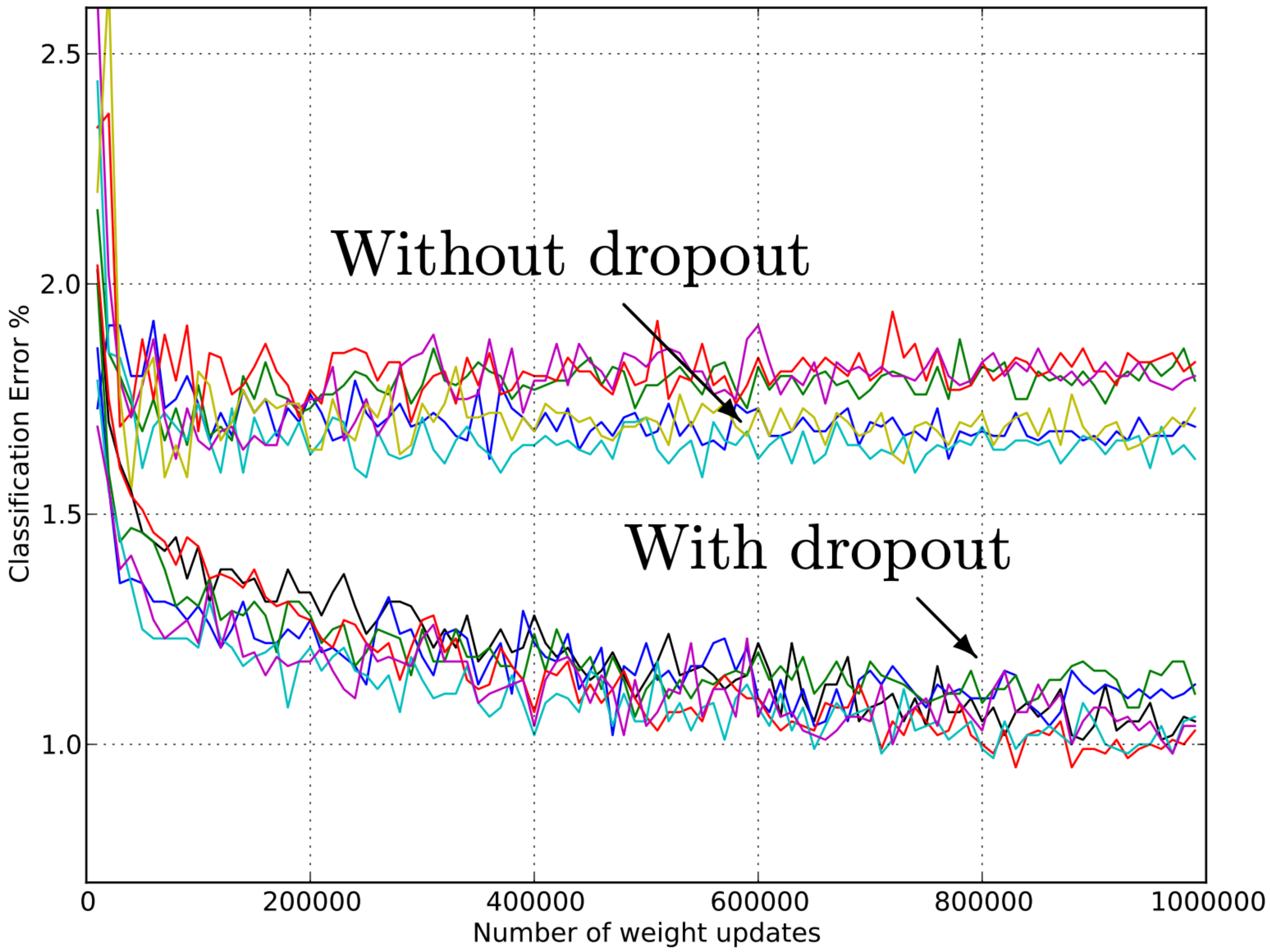
When testing:

- Use all nodes
- Scale each of them by the factor $p^{(l)}$ to ensure that the expected output (when considering the probability of dropping nodes during training) matches the actual output at test time

Remark: Alternatively, weight rescaling can be implemented during training time by scaling the weights by $1/p^{(l)}$ after each weight update and no changes are needed in test time - this is how it is implemented in practice

Dropout: results

- **Setting**: Fully-connected networks of different width and depth on MNIST
- Dropout results in **lower test error**
- However, dropout typically requires more iterations to converge due to increased stochastic noise



Source: Dropout: A Simple Way to Prevent Neural Networks from Overfitting (JMLR 2014)

Conclusion

Entangled effects of various methods: CIFAR10

model	# parameters	Random crop	Weight decay	Train accuracy	Test accuracy
Inception	1'649'402	Yes	Yes	100.0	89.05
		Yes	No	100.0	89.31
		No	Yes	100.0	86.03
		No	No	100.0	85.75
Inception w/o BatchNorm	1'649'402	No	Yes	100.0	83.00
		No	No	100.0	82.00
Alexnet	1'387'786	Yes	Yes	99.90	81.22
		Yes	No	99.82	79.66
		No	Yes	100.0	77.36
		No	No	100.0	76.07
MLP 3x512	1'735'178	No	Yes	100.0	53.35
		No	No	100.0	52.39
MLP 1x512	1'209'866	No	Yes	99.80	50.39
		No	No	100.0	50.51

C Zhang, S Bengio, M Hardt, B Recht, O Vinyals. Understanding deep learning requires rethinking generalization. ICLR, 2017

Entangled effects of various methods: ImageNet

model	Data aug	Dropout	Weight decay	Batch Norm	Skip Connections	Top-1 train	Top-1 test
ResNet200 (v2)	Yes	No	Yes	Yes	Yes	?	79.9
Inception (v3)	Yes	Yes	Yes	Yes	No	92.18	77.84
	Yes	No	No	Yes	No	92.33	72.95
	No	No	Yes	Yes	No	90.60	67.18(72.57)
	No	No	No	Yes	No	99.53	59.80(63.16)
VGG19	Yes	Yes	Yes	No	No	?	72.7

( ) : best test accuracy during training, i.e., with early stopping

C Zhang, S Bengio, M Hardt, B Recht, O Vinyals. Understanding deep learning requires rethinking generalization. ICLR, 2017

Recap

- Convolutional networks are composed of **sparsely connected convolutional layers** instead of fully-connected linear layers

- The same convolution is applied as a sliding window across all spatial locations

- **Data augmentation** usually results in a significant improvement in the model's generalization performance

- **Weight decay** and **dropout** can further enhance the performance, though typically to a lesser extent