

# Convolution in Neural Networks

Kate Farrahi

Vision, Learning and Control  
University of Southampton

References: [https://github.com/vdumoulin/conv\\_arithmetic](https://github.com/vdumoulin/conv_arithmetic)  
Justin Johnson, Bernhard Kainz, Jon Hare

# Motivation

- So far, we have focused on MLPs
- How is a 2D image input into an MLP?
- How can we keep the spatial information?



n02097047 (196)



n01682714 (40)



n03134739 (522)



n04254777 (806)



n02859443 (449)



n02096177 (192)



n02107683 (239)



n01443537 (1)



n02264363 (318)

# **Components of a Convolutional Neural Network (CNN)**

- **Convolution Layers**
- Activation Functions
- **Pooling Layers**
- **Normalization**
- Fully Connected Layers

# The Convolution Operation

In the **time domain**, convolution is:

$$\begin{aligned}(f * g)(t) &\stackrel{\text{def}}{=} \int_{-\infty}^{\infty} f(\tau) g(t - \tau) d\tau \\&= \int_{-\infty}^{\infty} f(t - \tau) g(\tau) d\tau.\end{aligned}$$

Notice that the image or kernel is “flipped” in time, where  $f$  is the image and  $g$  is the kernel.

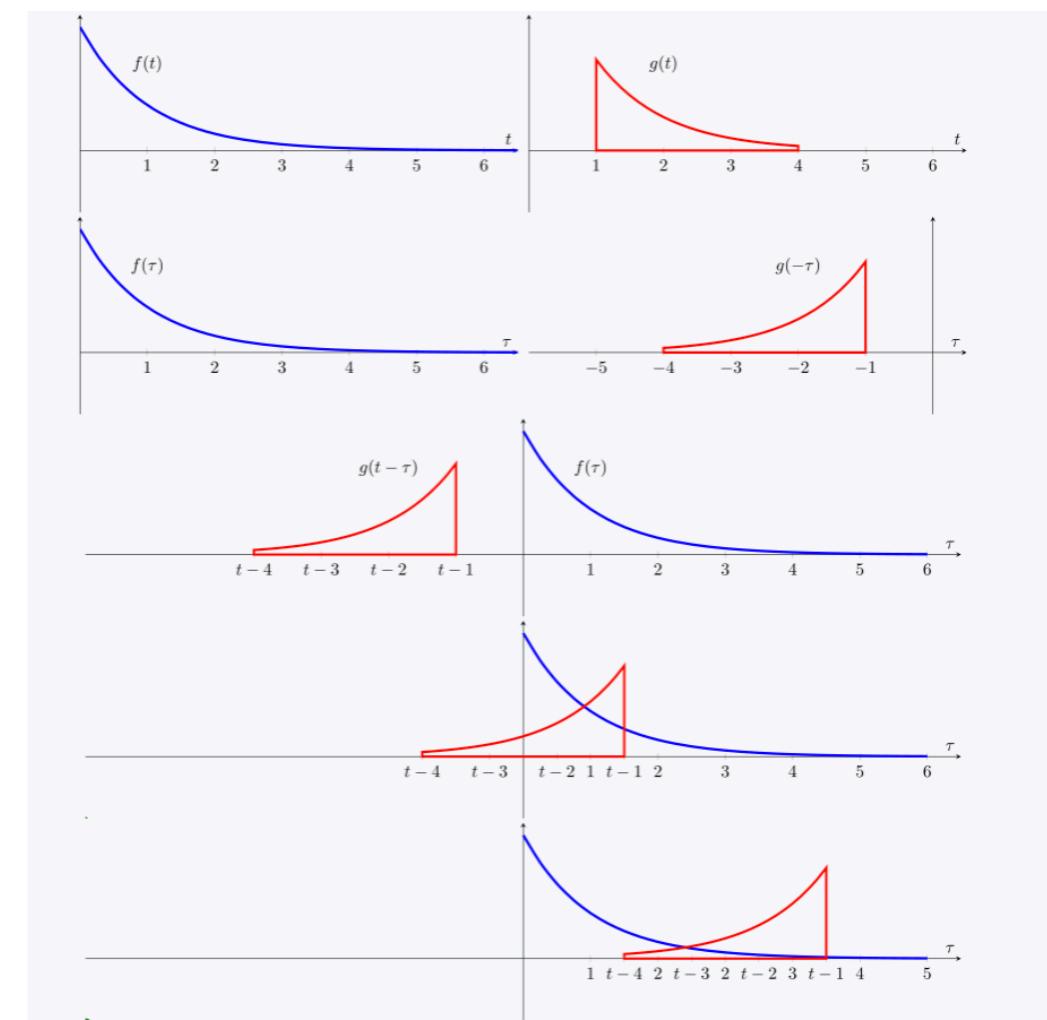
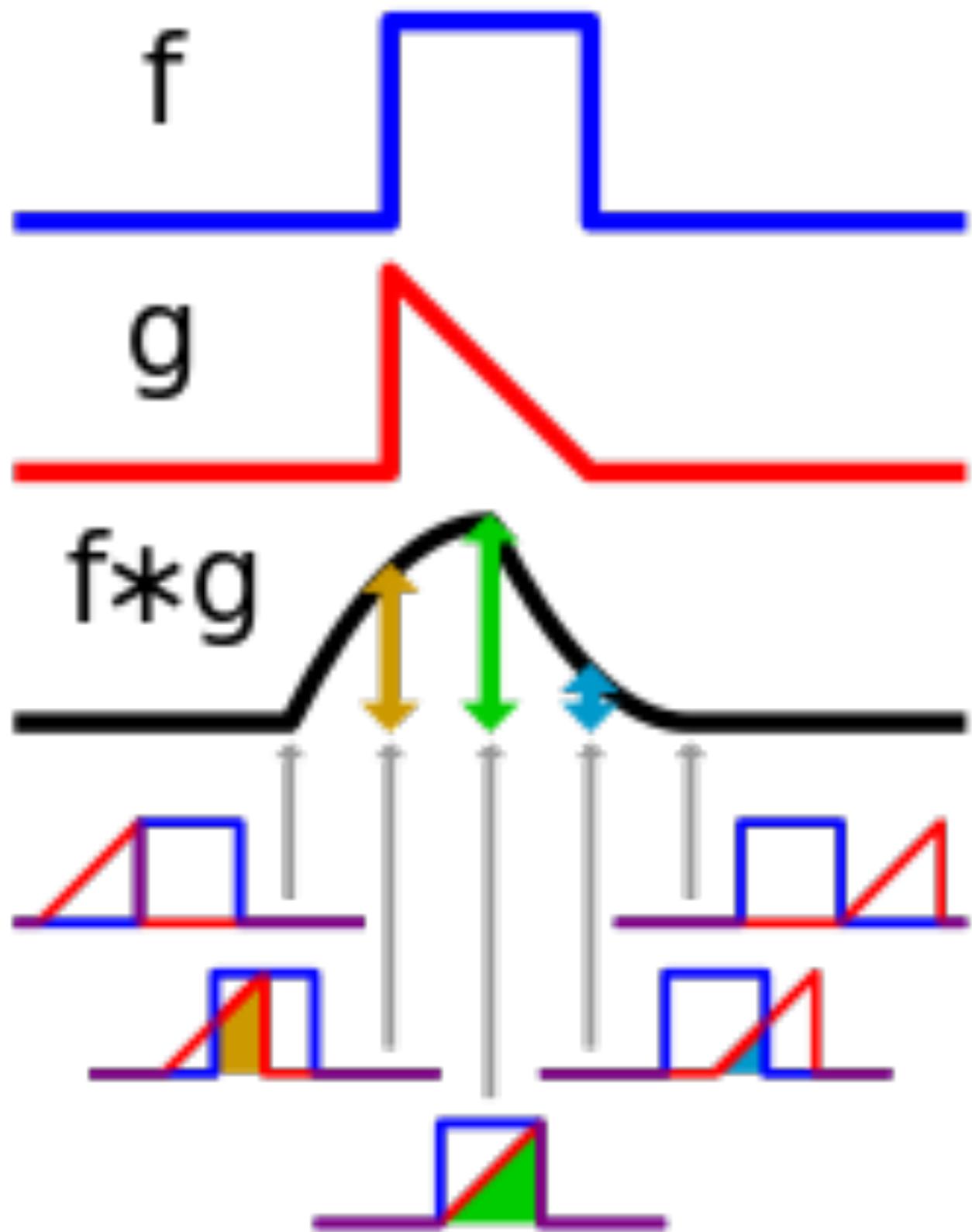


Image taken from: Wikipedia

# Convolution



# Cross-correlation

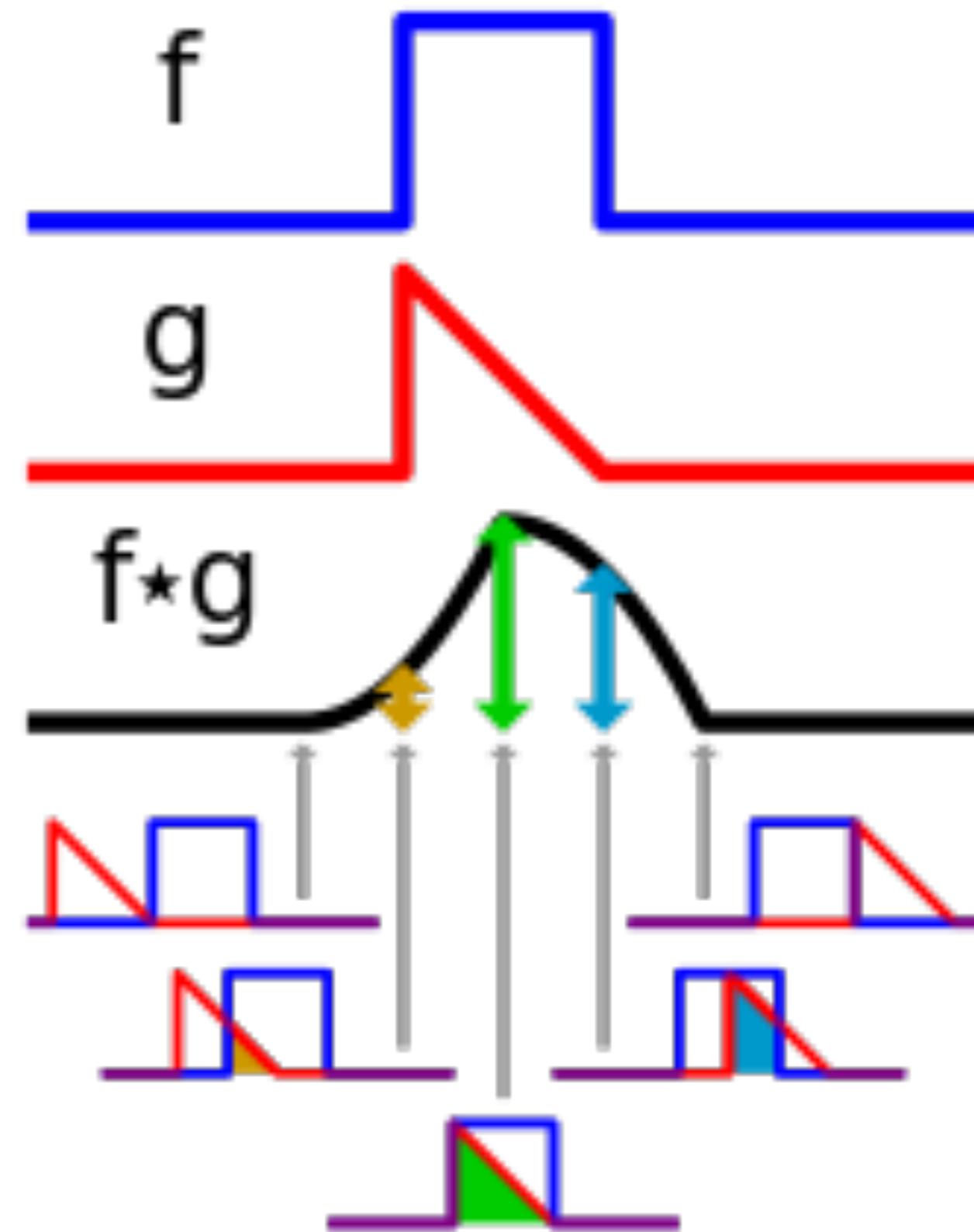


Image taken from: Wikipedia

# Cross-Correlation in Practice

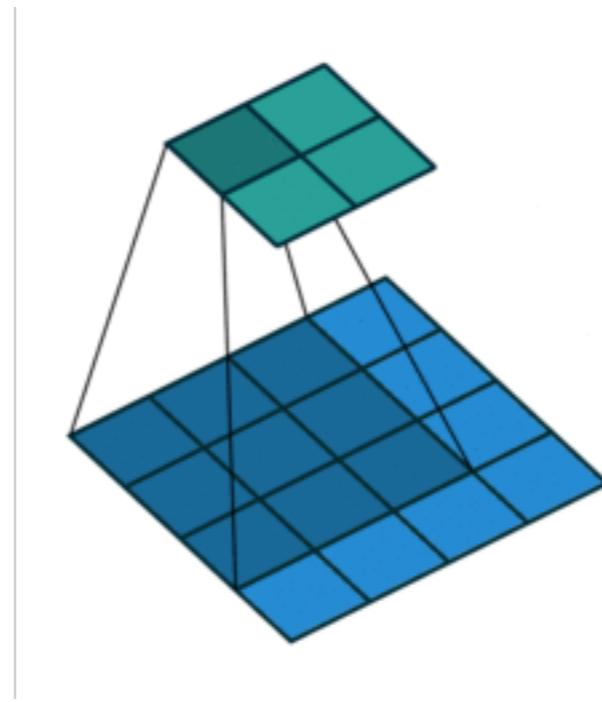
Convolution over a two-dimensional input image  $I$  and two-dimensional kernel  $K$  is defined as:

$$(1) \quad S(i, j) = (I^*K)(i, j) = \sum_m \sum_n I(i - m, j - n)K(m, n)$$

However, nearly all machine learning and deep learning libraries use the **simplified cross-correlation function**

$$(2) \quad S(i, j) = (I^*K)(i, j) = \sum_m \sum_n I(i + m, j + n)K(m, n)$$

# Convolution Visualised



**Visual link [https://github.com/vdumoulin/conv\\_arithmetic/blob/master/gif/no\\_padding\\_no\\_strides.gif](https://github.com/vdumoulin/conv_arithmetic/blob/master/gif/no_padding_no_strides.gif)**

# “Convolution” in Neural Networks

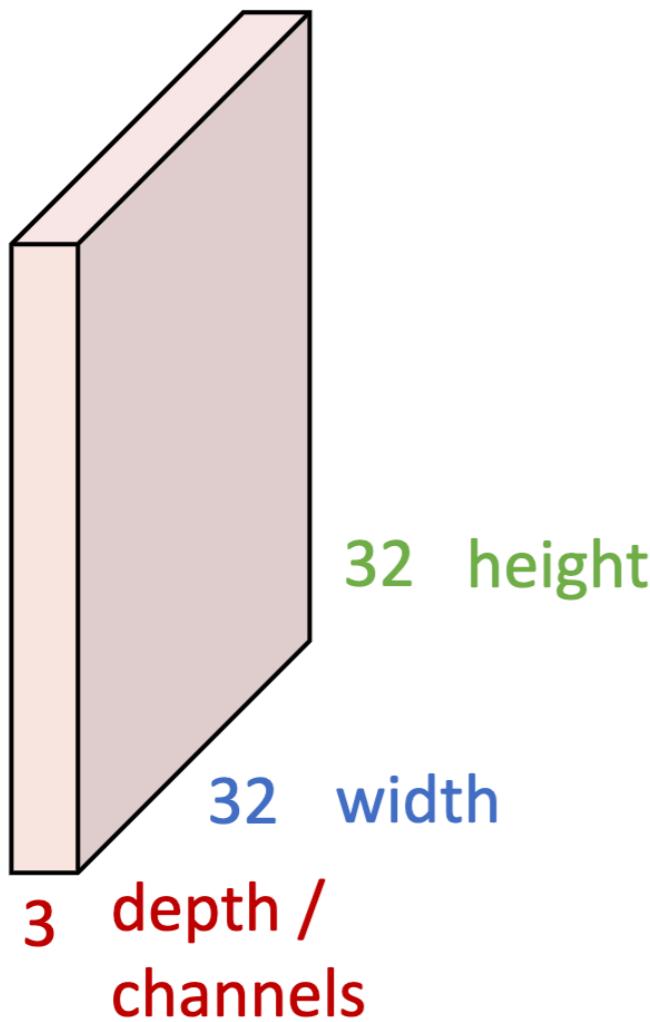
- “Convolution” in the neural network literature almost always refers to an operation akin cross-correlation
  - An element-wise multiplication of learned weights across a receptive field, which is repeated at various positions across the input.
- Normally, we also add an additional *bias term*; a single bias term for each *kernel*.
- There are also other parameters of these “convolutions”...

# Convolutional Layers

- In a convolutional layer, we have multiple kernels or filters which are learnt (plus the biases). This set of kernels can be called a bank of kernels.
- Each filter produces a single “Response Map” or “Feature Map” or "Activation Map". The activation maps are stacked together as “channels” of the resultant output tensor
- Each activation map tells us how much does each position in the input respond to the corresponding convolutional filter

# Convolution Layer

$3 \times 32 \times 32$  image



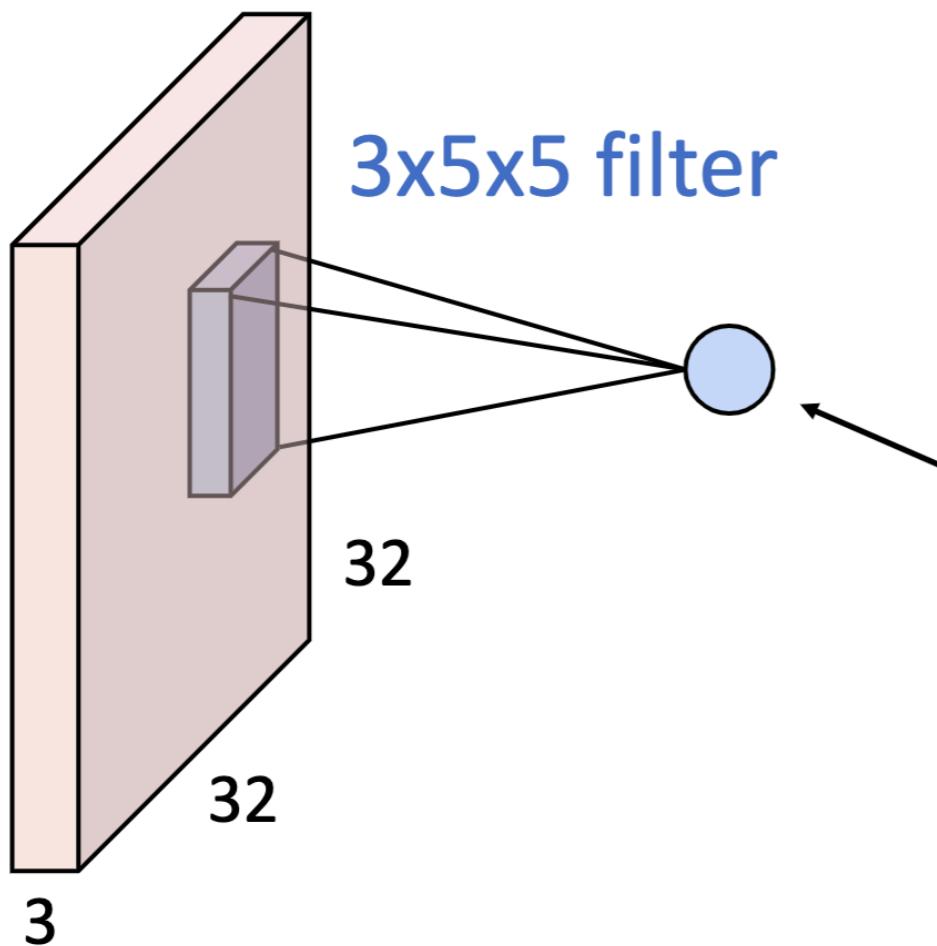
$3 \times 5 \times 5$  filter



**Convolve** the filter with the image  
i.e. “slide over the image spatially,  
computing dot products”

# Convolution Layer

3x32x32 image

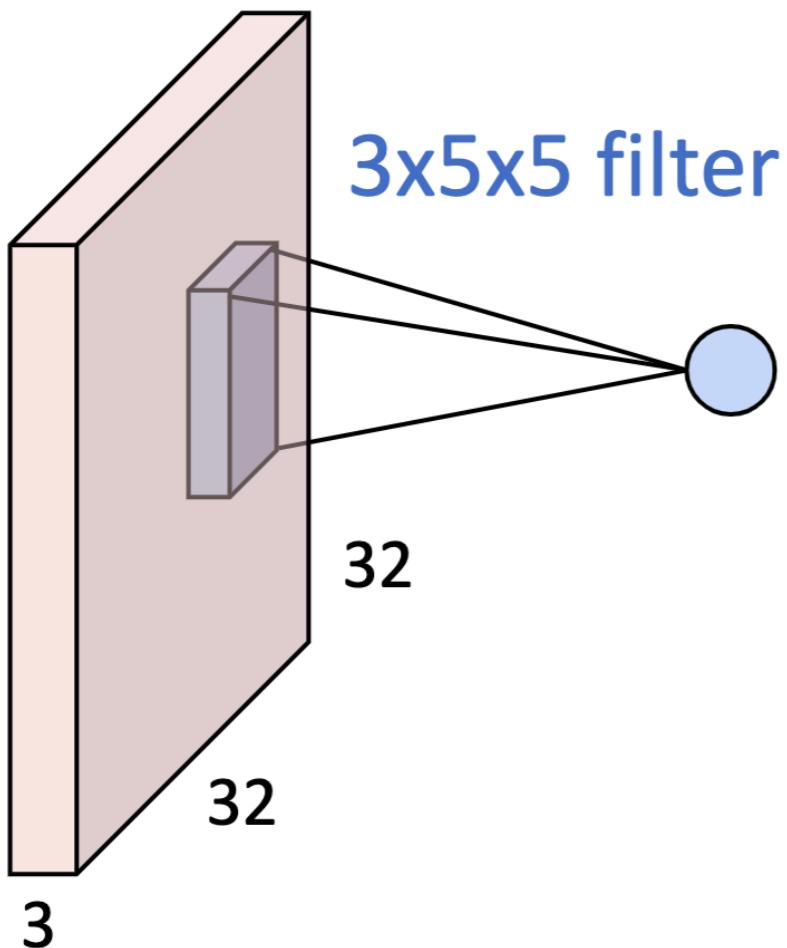


**1 number:**  
the result of taking a dot product between the filter  
and a small 3x5x5 chunk of the image  
(i.e.  $3 \times 5 \times 5 = 75$ -dimensional dot product + bias)

$$w^T x + b$$

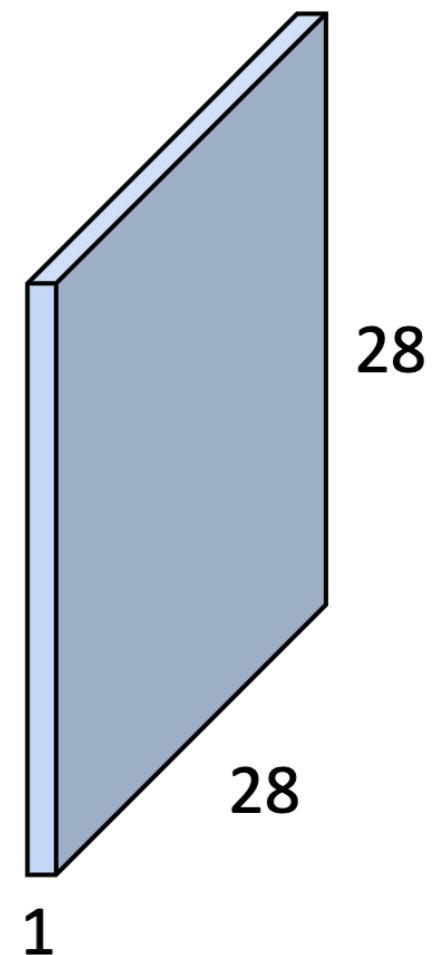
# Convolution Layer

3x32x32 image



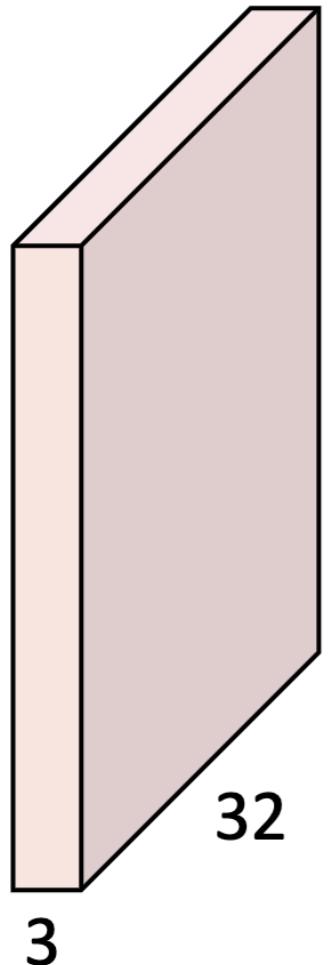
convolve (slide) over  
all spatial locations

1x28x28  
activation map

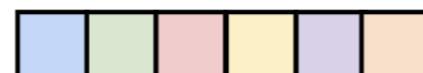


# Convolution Layer

3x32x32 image

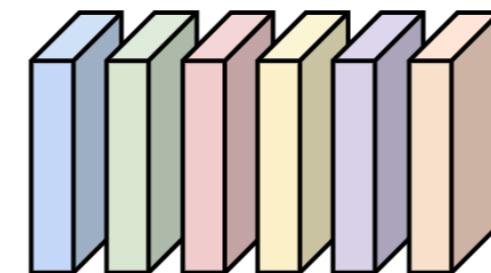


Also 6-dim bias vector:

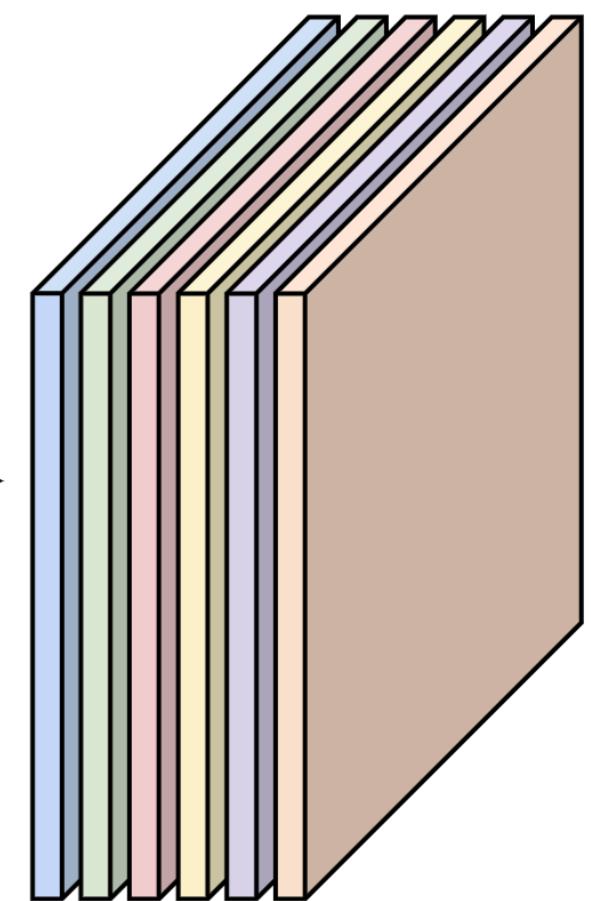


Convolution  
Layer

6x3x5x5  
filters



6 activation maps,  
each 1x28x28



Stack activations to get a  
6x28x28 output image!

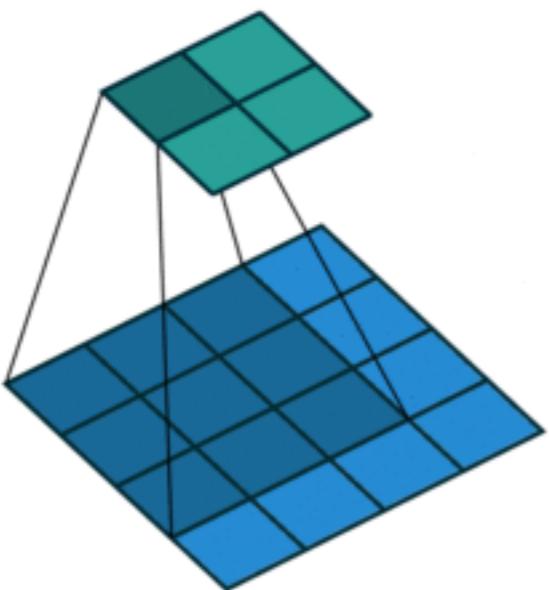
# 2D Convolutions with kernel size of 1

- 1x1 convolutions are a common place operation, but might seem non-sensical at first
  - They do not capture any local spatial information
  - They are used to change the number of feature maps without affecting the spatial resolution

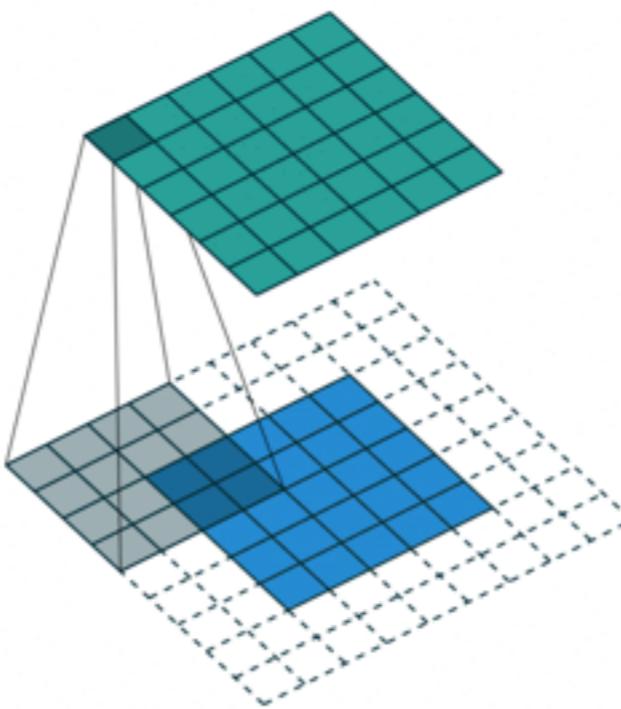
# Padding

- What happens to a convolution at the edges of its spatial extent?
- In spatial convolution if we do nothing, the output will be smaller...
- We often use zero-padding to retain the size
- With "same" padding,  $P = (K-1) / 2$  to make the output the same size as the input
- Output:  $W - K + 1 + 2P$  ( $K$  = kernel size,  $W$  = input size)

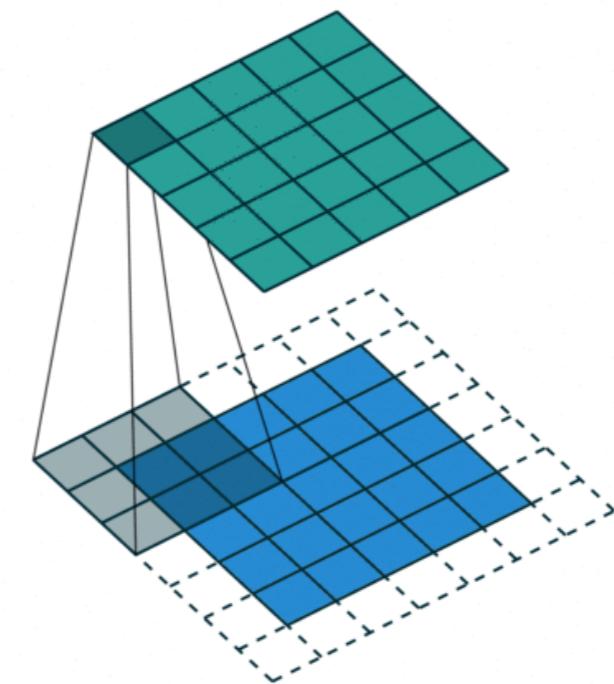
## Arbitrary padding



No padding

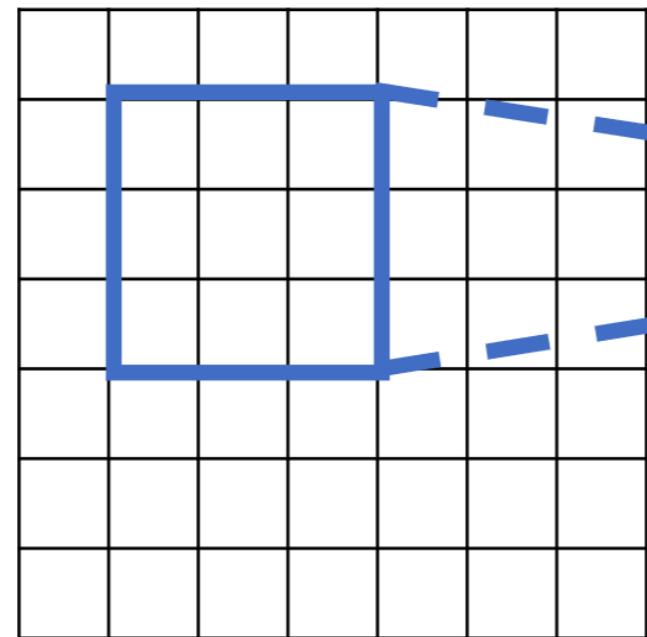


“same” padding

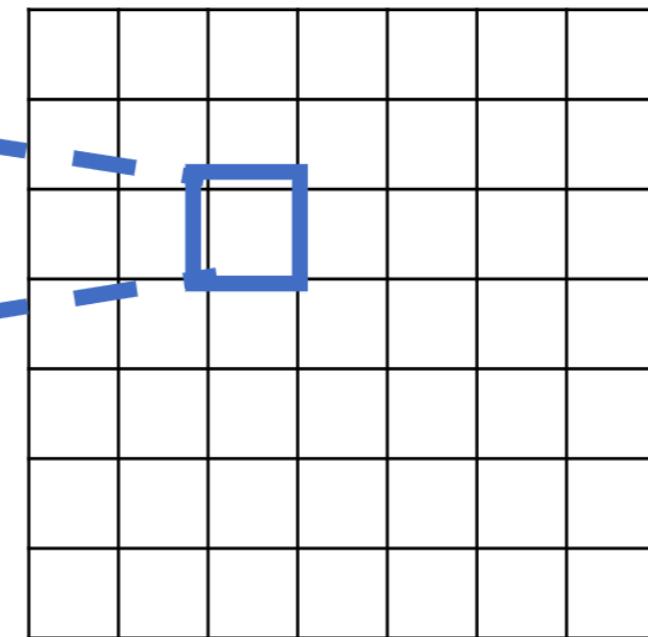


# Receptive Fields

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



Input

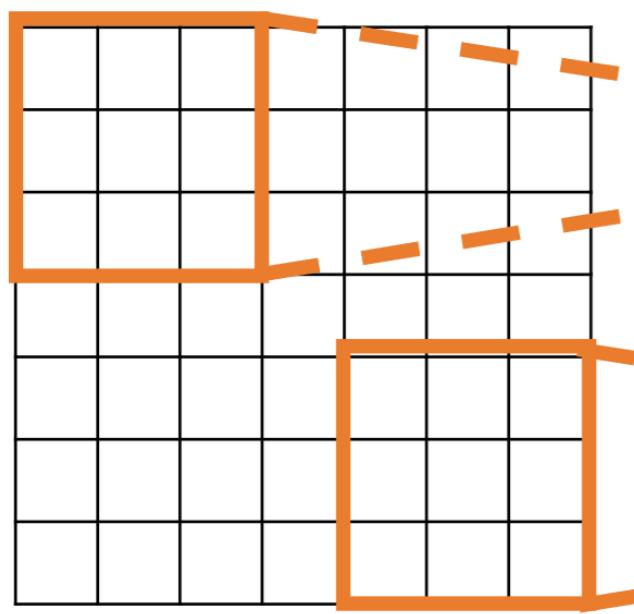


Output

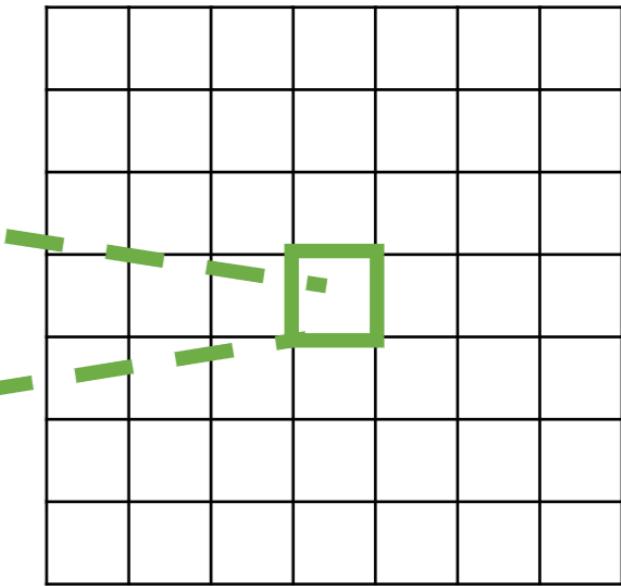
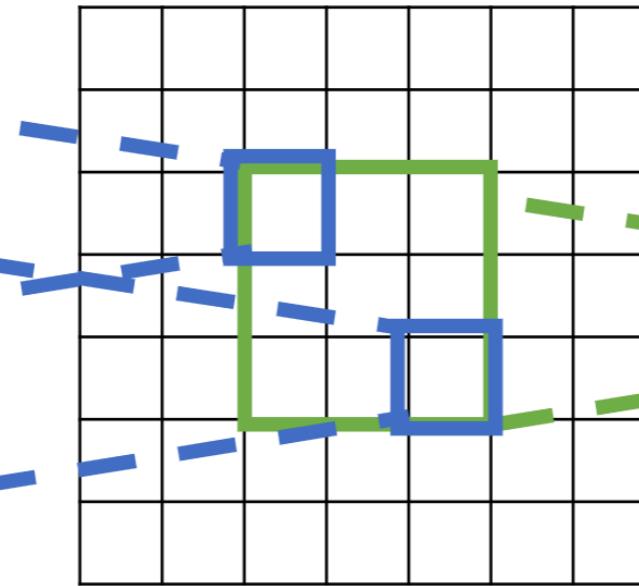
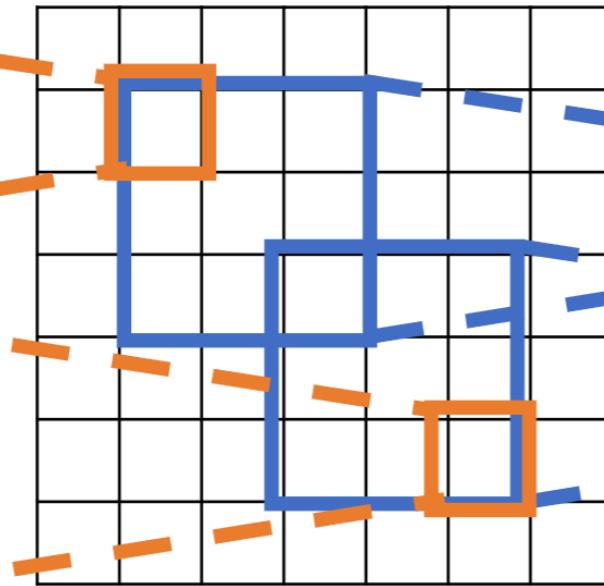
# Receptive Fields

Each successive convolution adds  $K - 1$  to the receptive field size

With  $L$  layers the receptive field size is  $1 + L * (K - 1)$



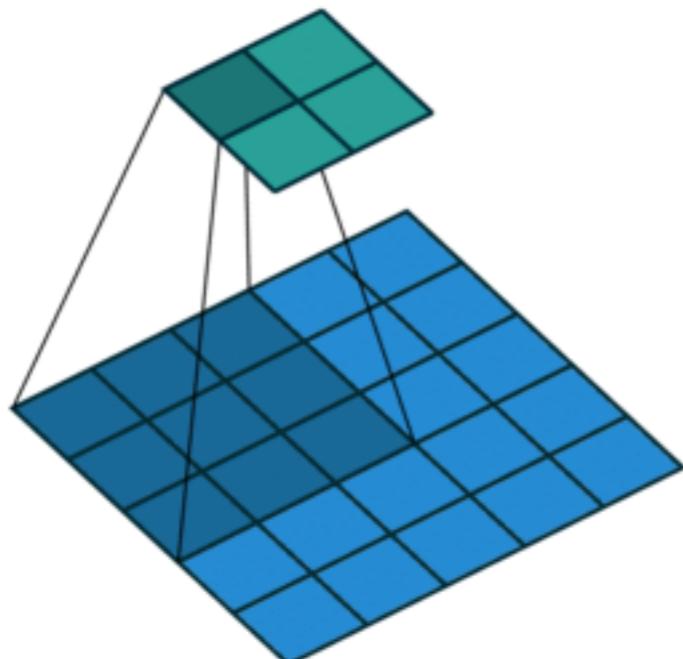
Input



Output

# Striding

- Convolution is expensive... could we make it cheaper by skipping over positions?



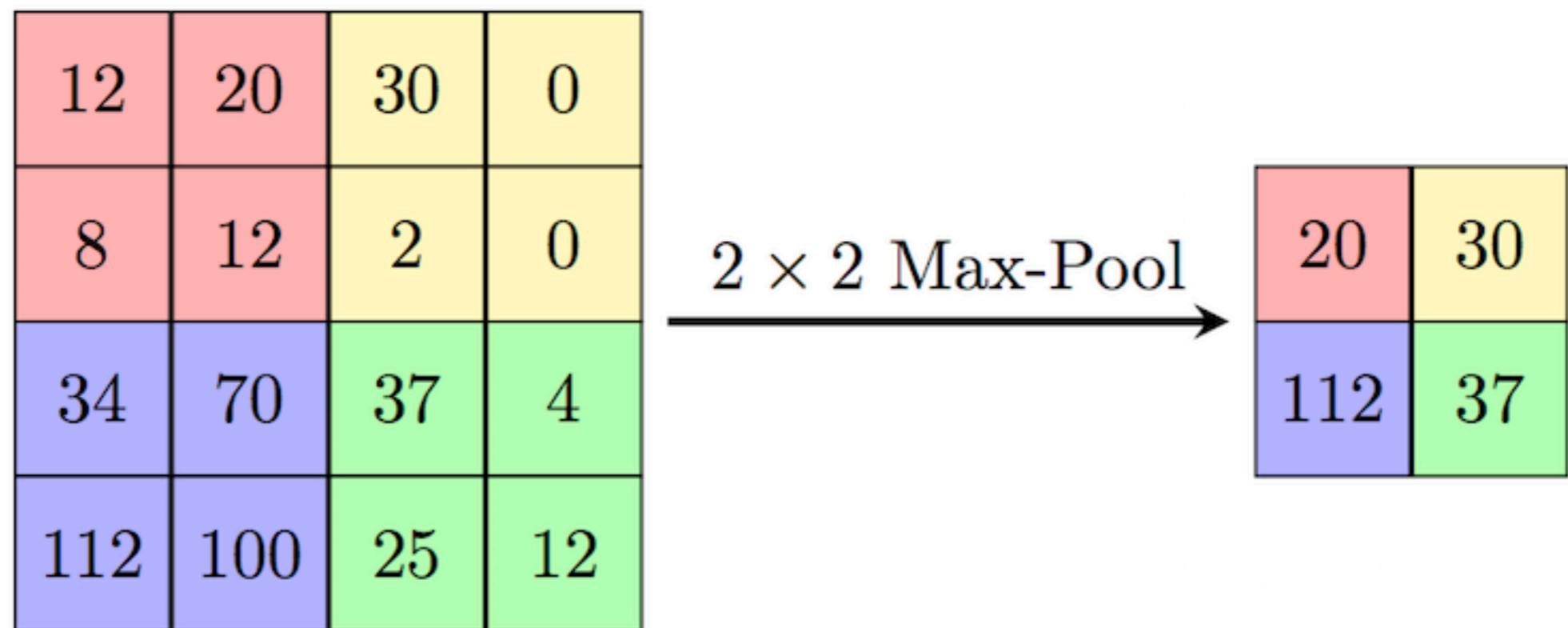
Stride=(2,2)

In general:  
Input: W  
Filter: K  
Padding: P  
Stride: S  
Output:  $(W - K + 2P) / S + 1$

# Pooling

- Striding is a popular way to reduce spatial dimensionality in modern networks
- Before striding was devised, **pooling**, was the defacto way of reducing dimensionality

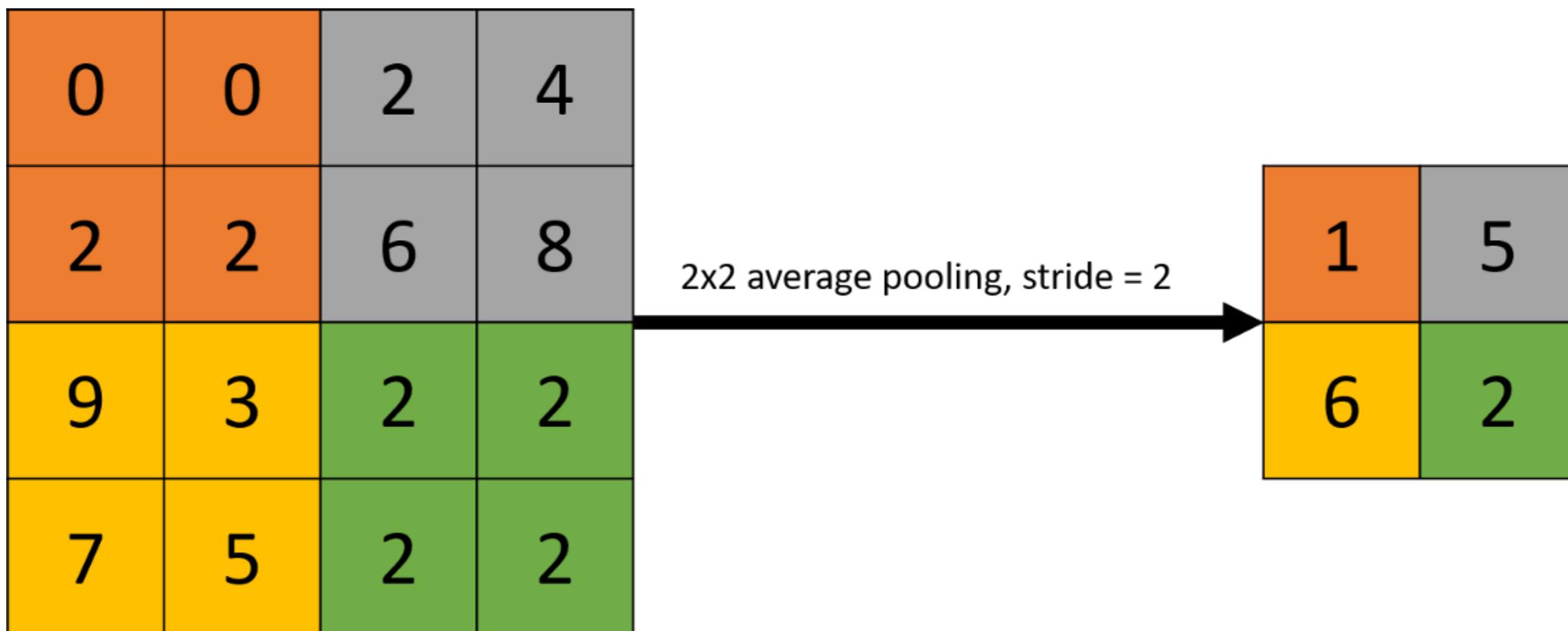
# Max Pooling, 2x2, stride=2



# Max Pooling Gradients

- The gradient of the max pooling operation is 1 everywhere a max value was selected, and zero elsewhere
- This means that implementations not only need to record the max values in the forward-pass, but also keep track of the positions of those maximums for the backward pass

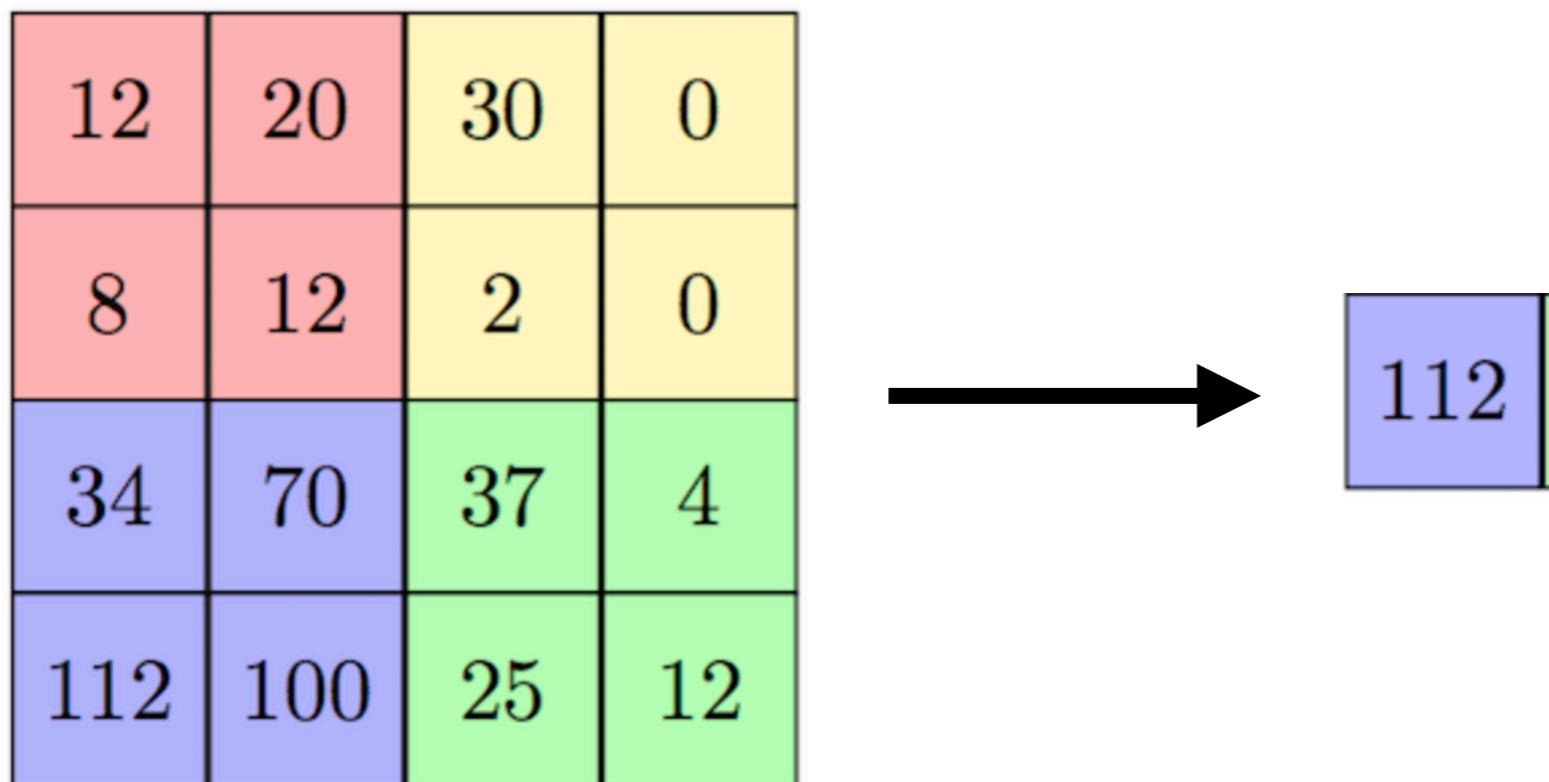
# Average Pooling



# Local Versus Global Pooling

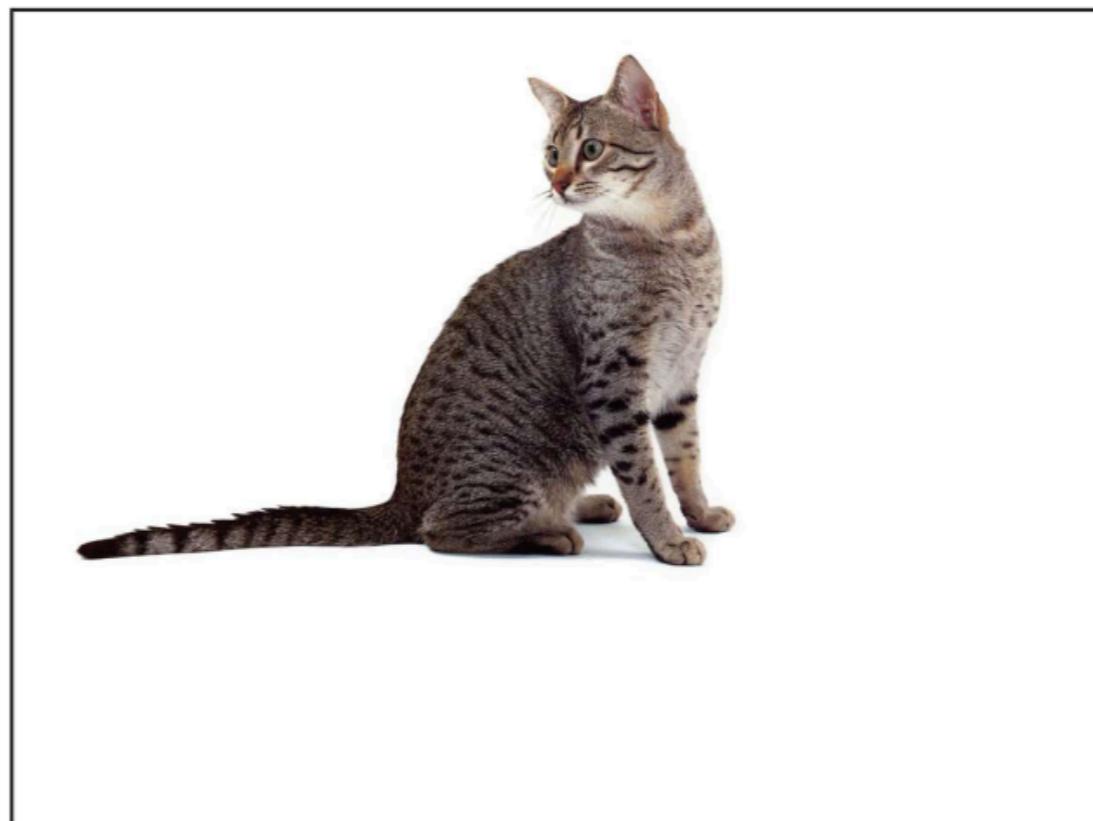
- The pooling operations on the previous slides are local
  - They result in a feature map reducing in spatial size
- Global pooling reduces a feature map to a scalar
  - So a tensor of many feature maps would be reduced to a single feature vector
  - Often used near the end of networks to flatten feature maps into feature vectors that can be fed into an MLP

# Global Max-Pooling



# Invariance and equivariance

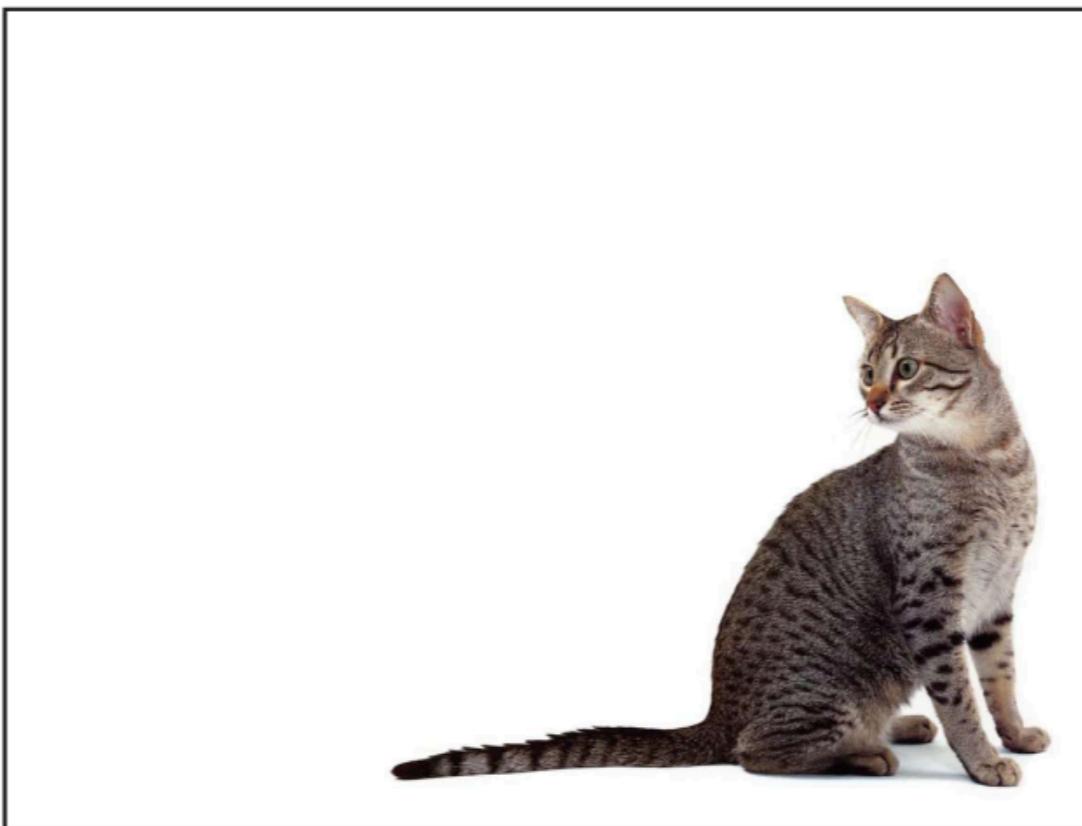
- Shift invariance



**'cat'**

# Invariance and equivariance

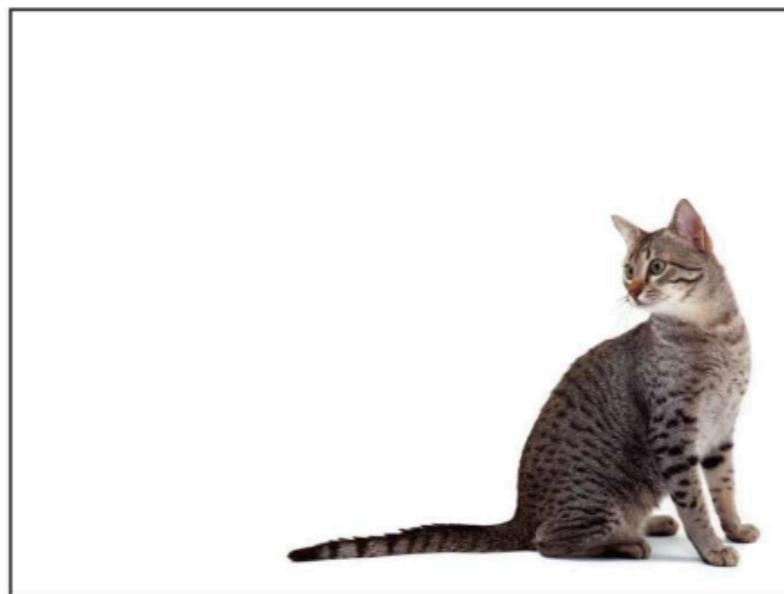
- Shift invariance



**'cat'**

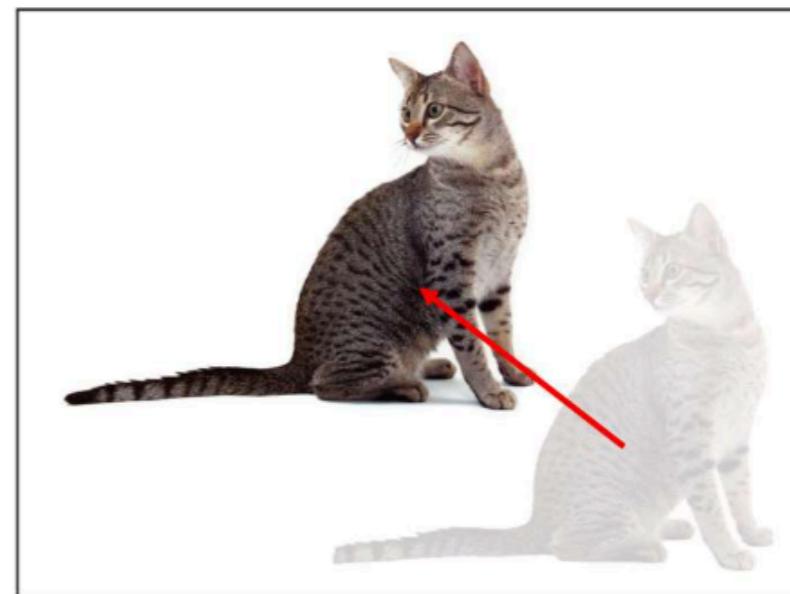
# Shift invariance

**Input  $x$**



**Output  $f(x) = 1$**

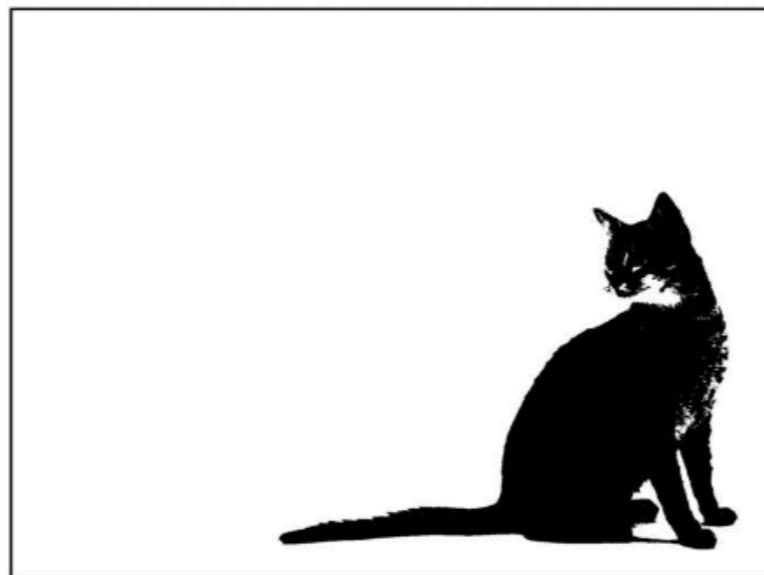
**Shifted input  $S_vx$**



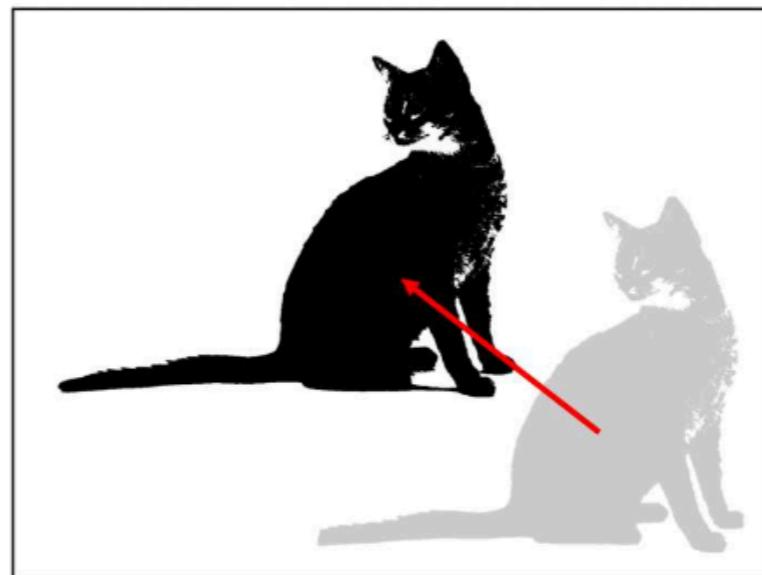
**Output  $f(S_vx) = 1$**

# Shift equivariance

**Input  $\mathbf{x}$**



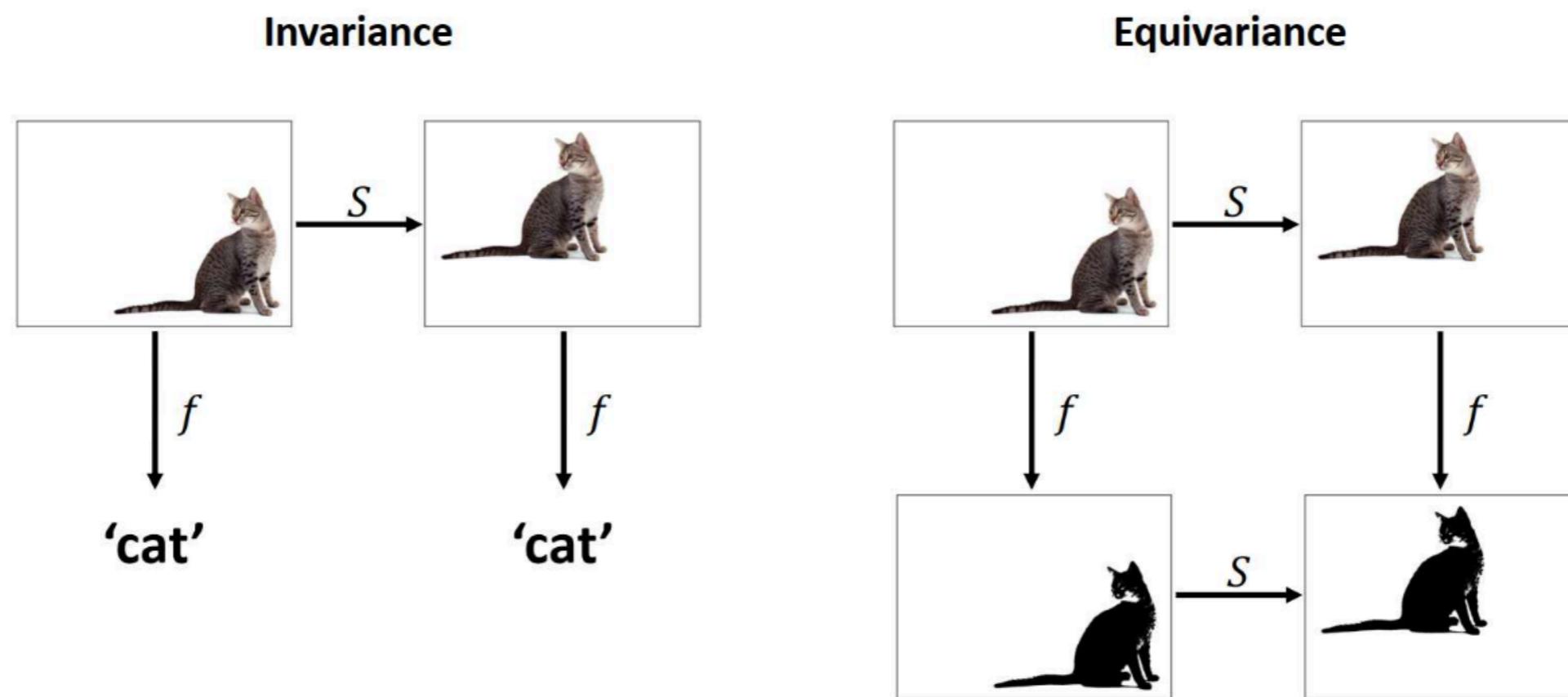
**Shifted input  $S_v\mathbf{x}$**



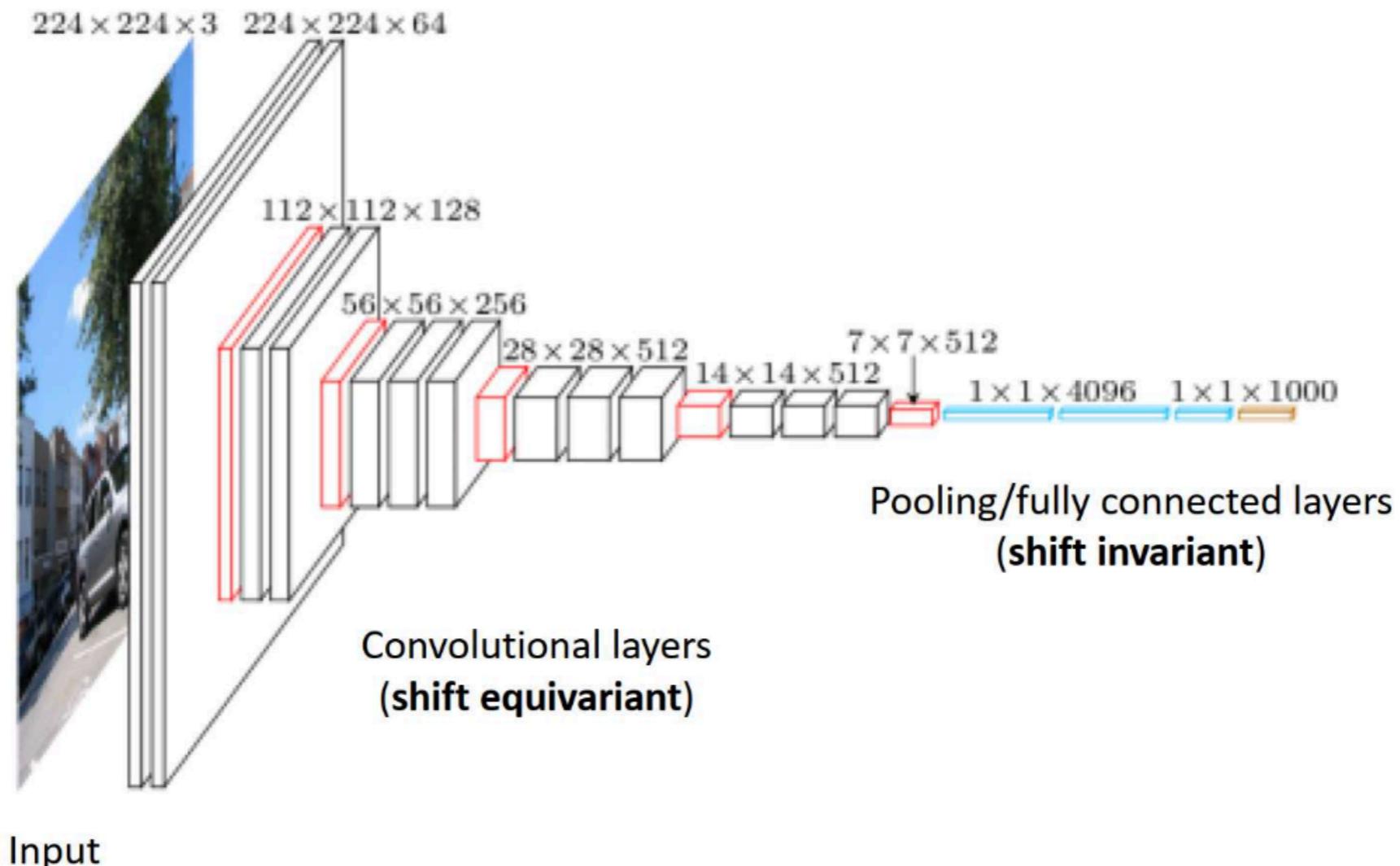
$$\text{Output } f_i(\mathbf{x}) = \begin{cases} 1 & \text{pixel } i \in \text{cat} \\ 0 & \text{otherwise} \end{cases}$$

$$\text{Output } f_i(S_v\mathbf{x}) = \begin{cases} 1 & \text{pixel } i \in \text{cat} \\ 0 & \text{otherwise} \end{cases}$$

# Invariance vs equivariance

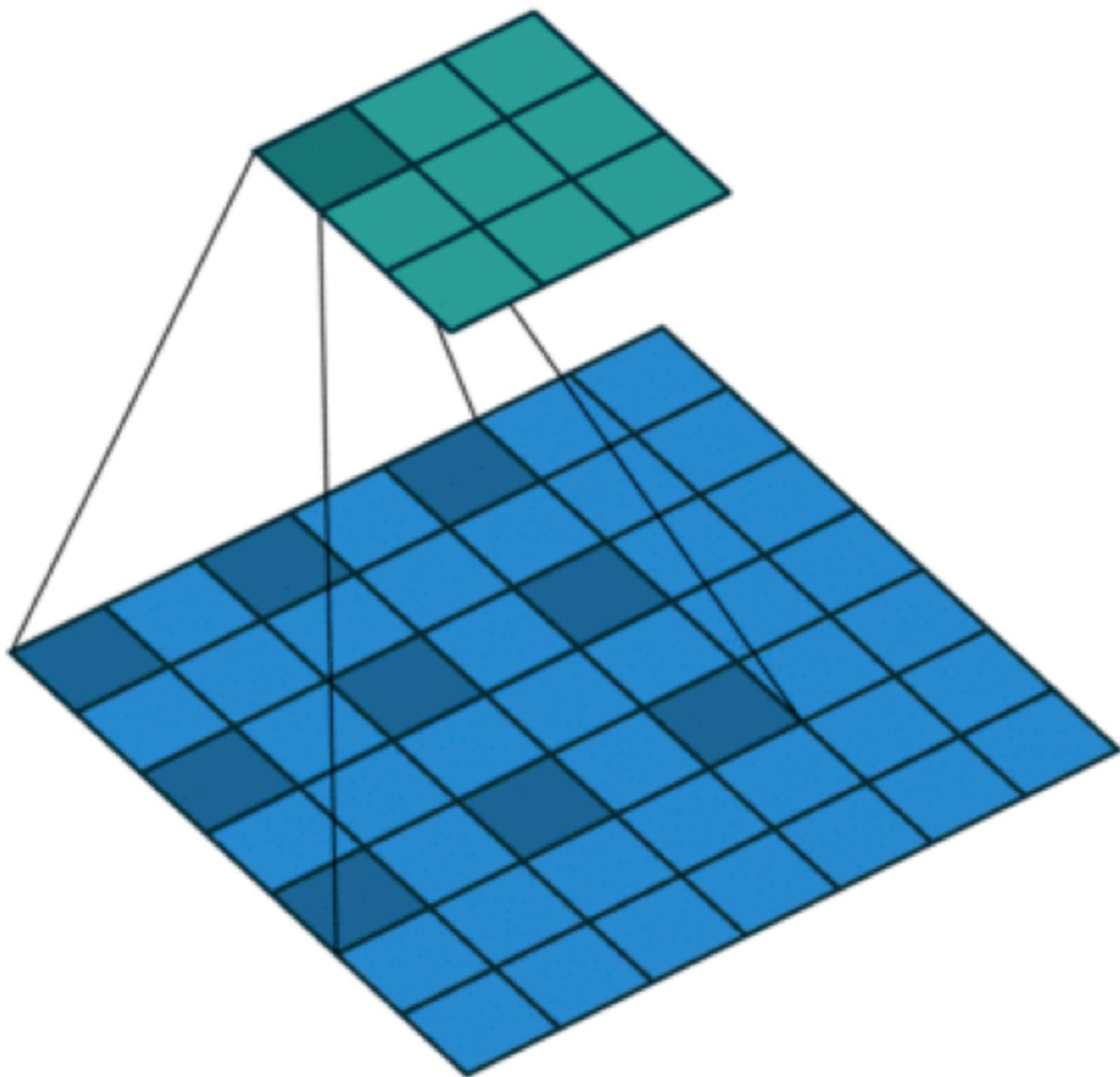


# How shift invariance is achieved in CNNs?



# Dilated Convolutions

- Sometimes we want to have larger receptive fields in our networks
  - We can increase the kernel size to achieve this, but this introduces more weights
  - We can downsample/pool the input, but this decreases spatial resolution
  - Or we could ‘pad’ the kernel with zeros throughout to increase the effective size without increasing the number of parameters



# Data Types

- Convolutions are applied to many dimensionalities and types of data - for example:

	<b>Single Channel</b>	<b>Multichannel</b>
<b>1-D</b>	Audio	Multiple sensor data over time
<b>2-D</b>	Audio data preprocessed into a spectrogram; greyscale images	Colour image data (e.g. RGB)
<b>3-D</b>	Volumetric data, e.g. CT scans	Colour video data