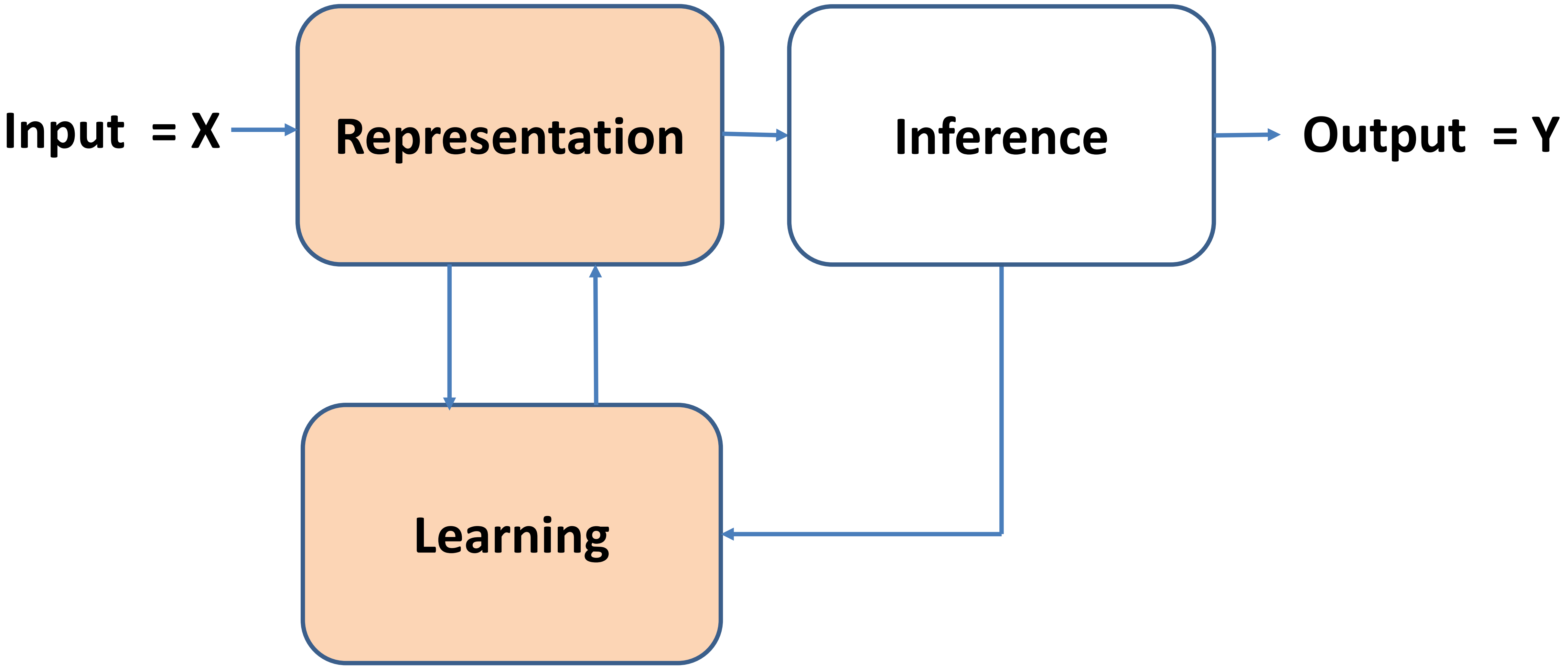
06 | Convolutional Neural Networks

Microsoft

Stephen F Elston | Principle Consultant, Quantia Analytics, LLC

Convolutional Neural Networks



Convolutional Neural Networks

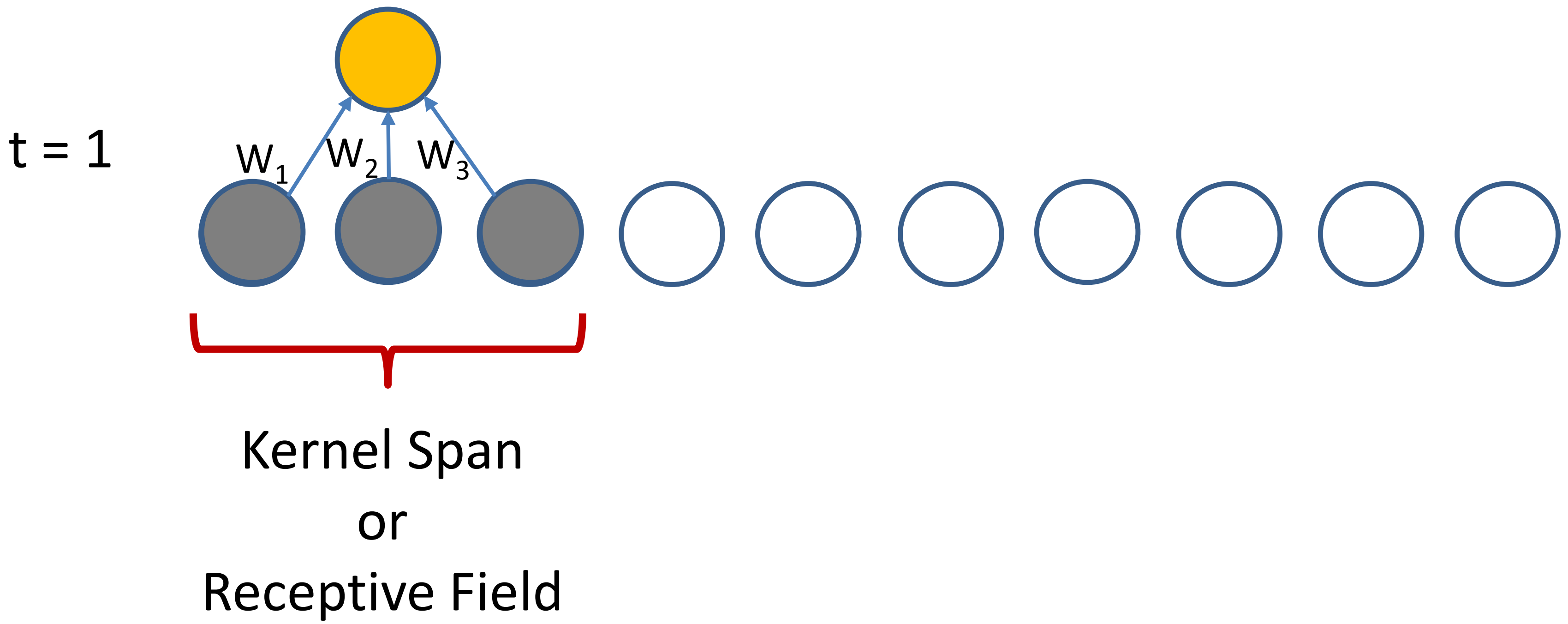
- **CNN**s are used to **learn complex feature maps**
  - **Invariant** to translation and distortion of features
  - **Reduce the dimensionality** of input tensors
  - **Share weights** and are relatively easy to train
- CNNs have a long history
  - LeCun et. al. (1998) first employed CNNs for automatic check handling
  - Era of general use started when Krizhevsky et. al. (2012) won an ImageNet object recognition competition
  - Now commonly used for image, speech and text problems

1-D Convolution

- 1-D CNNs are a simple, but useful example
  - Time series data
  - Text data
  - Speech
- Convolution kernel is moved along the input tensor
  - Kernel has a small span compared to dimension of input tensor
  - At each step a weighted output value is computed

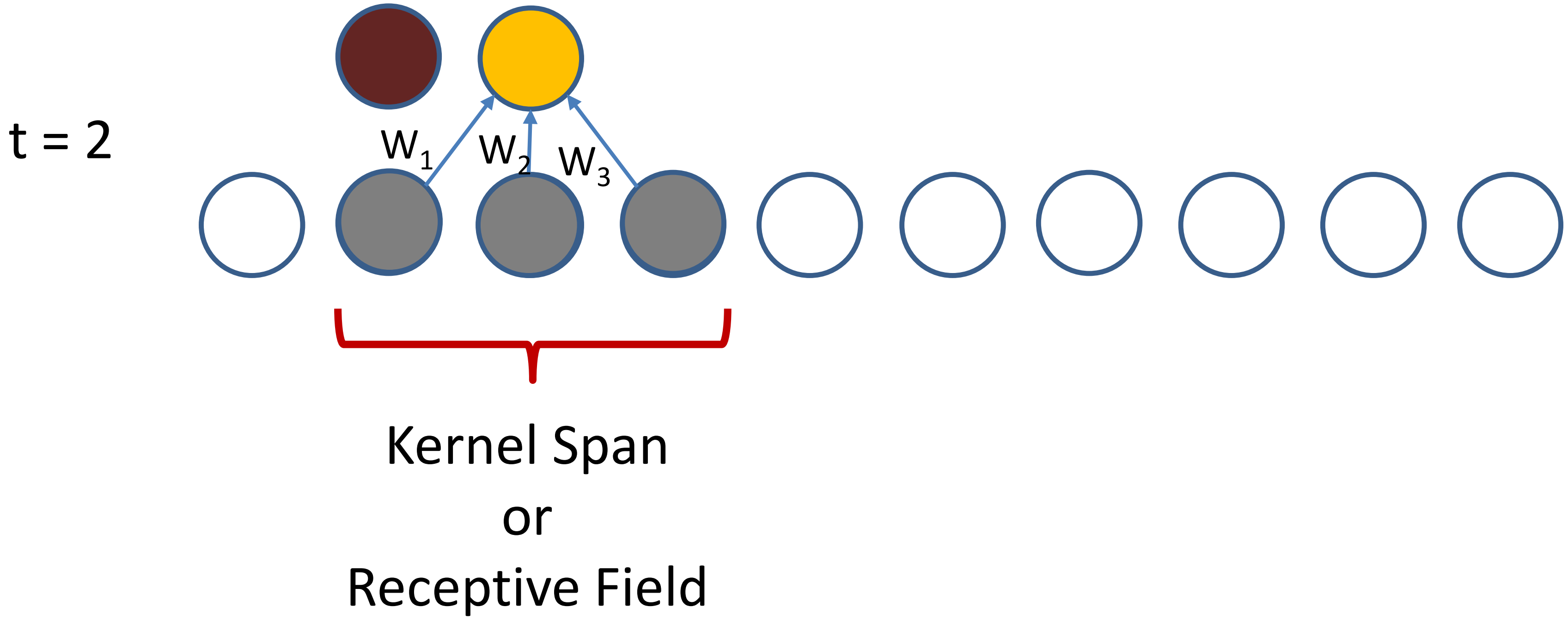
1-D Convolution

1-D CNNs are a simple, but useful example



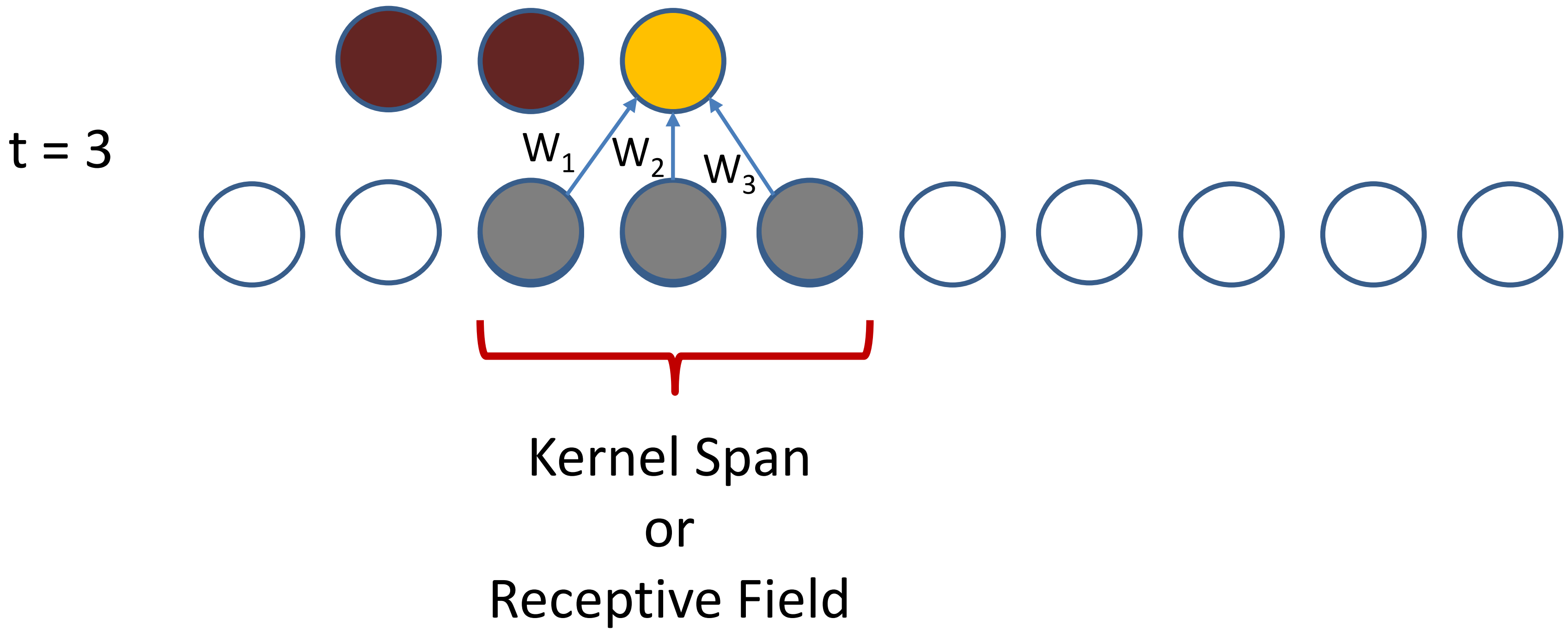
1-D Convolution

1-D CNNs are a simple, but useful example



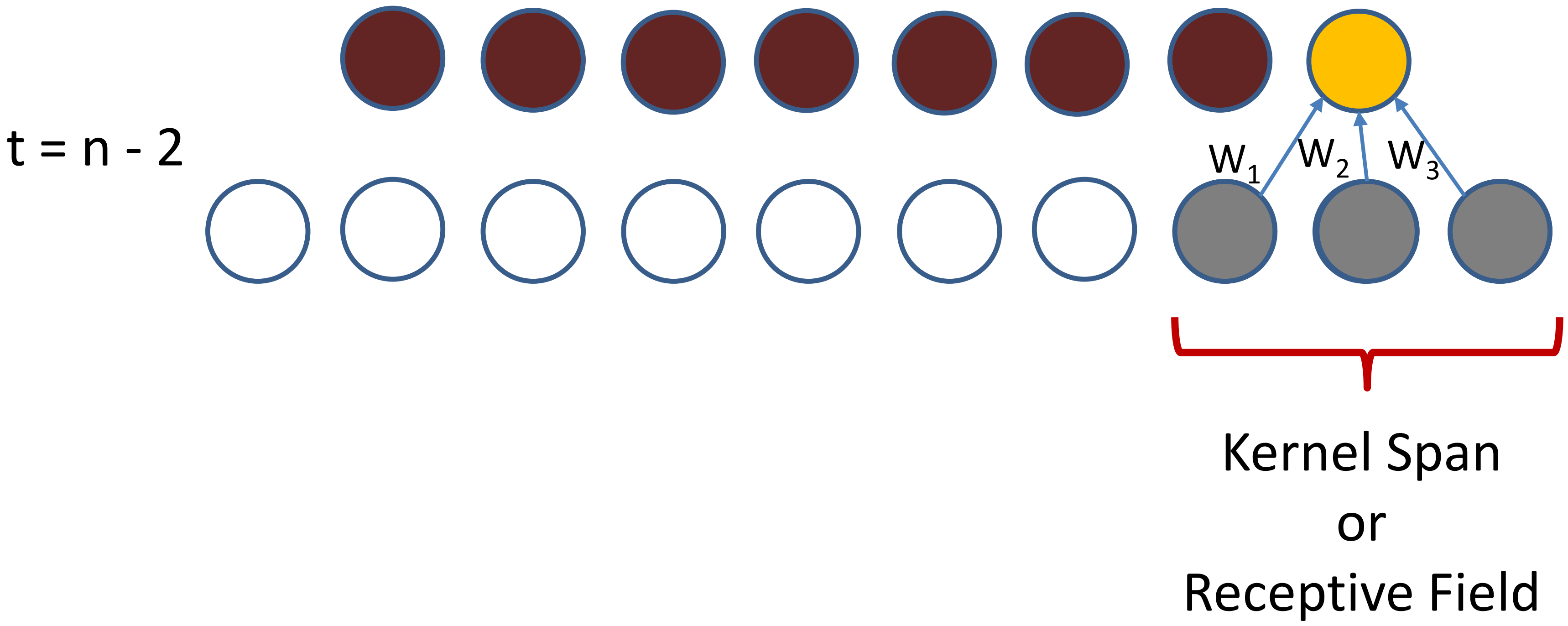
1-D Convolution

1-D CNNs are a simple, but useful example



1-D Convolution

1-D CNNs are a simple, but useful example



1-D Convolution

Mathematically, express 1-d convolution as a weighted sum over a set of discrete kernel values:

$$s(t) = (x * k)(t) = \sum_{\tau=t-a}^{t+a} x(t)k(t-\tau)$$

Where:

$s(t)$ is the output of the convolution operation at time t,

x is the series of values,

k is the convolution kernel

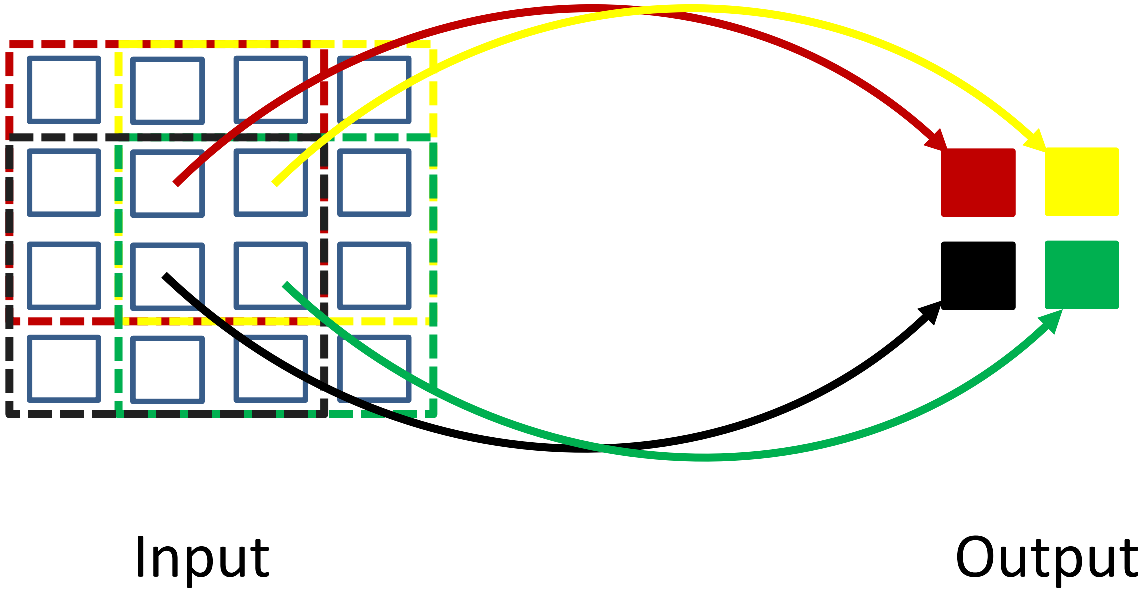
$*$ is the convolution operator

τ is the time difference of the convolution operator.

$a = \frac{1}{2}(kernel\_span + 1)$, for an odd length kernel

2-D Convolution

- 4 x 4 input tensor
- 3 x 3 convolution operator
- 2 x 2 output tensor



2-D Convolution

- Mathematically, we express 2-d convolution as a weighted sum over a discrete rectangular kernel:

$$S(i,j) = (I * K)(i,j) = \sum_{m=i-a}^{i+a} \sum_{n=j-a}^{j+a} I(i,j)K(i-m, j-n)$$

- Where S, I and K are now tensors

2-D Convolution

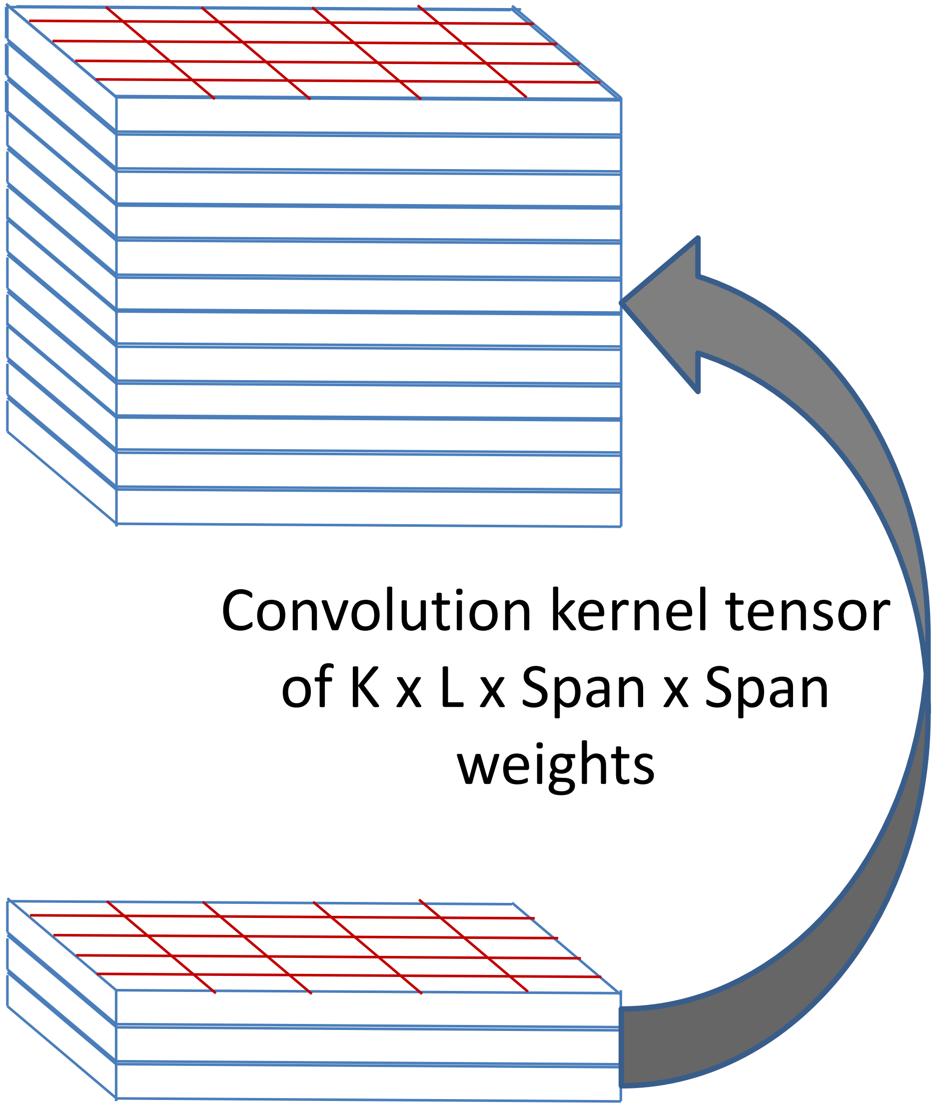
- The image and kernel tensors are **commutative** in the convolution relationship
- This allows an operation known as **kernel flipping** with the following alternative result:

$$s(i,j) = (I * K)(i,j) = \sum_{m=i-a}^{i+a} \sum_{n=j-a}^{j+a} I(i-m, j-n)K(i,j)$$

Convolution in Higher Dimensions

- Tensor notation allows easy extension to higher dimensions
  - Input tensor has **multiple input channels**
  - 3-D for color image
  - 4-D for video
- Create **multiple feature maps**
  - Convolution kernel tensor has **multiple output channels**
  - Each output channel is a different feature map
  - Feature in a channel might be vertical lines, horizontal lines, corners, etc

Convolution in Higher Dimensions



Convolution in Higher Dimensions

A multi-dimensional convolution relationship can be written:

$$Z_{i,j,k} = (V * Z)(i,j,k,l) = \sum_{l} \sum_{m=-a}^{a} \sum_{n=-a}^{a} V_{i,j,l} \cdot K_{i-m,j-n,k,l}$$

Where: i, j are the spatial dimenstions

l is the index of the input channel

k is the index of the output channel

$K_{i,j,k,l}$ is the kernel connecting the lth channel of the input to the kth channel of the output for pixel offsets i and j

$V_{i,j,l}$ is the i, j input pixel offsets from channel l of the input,

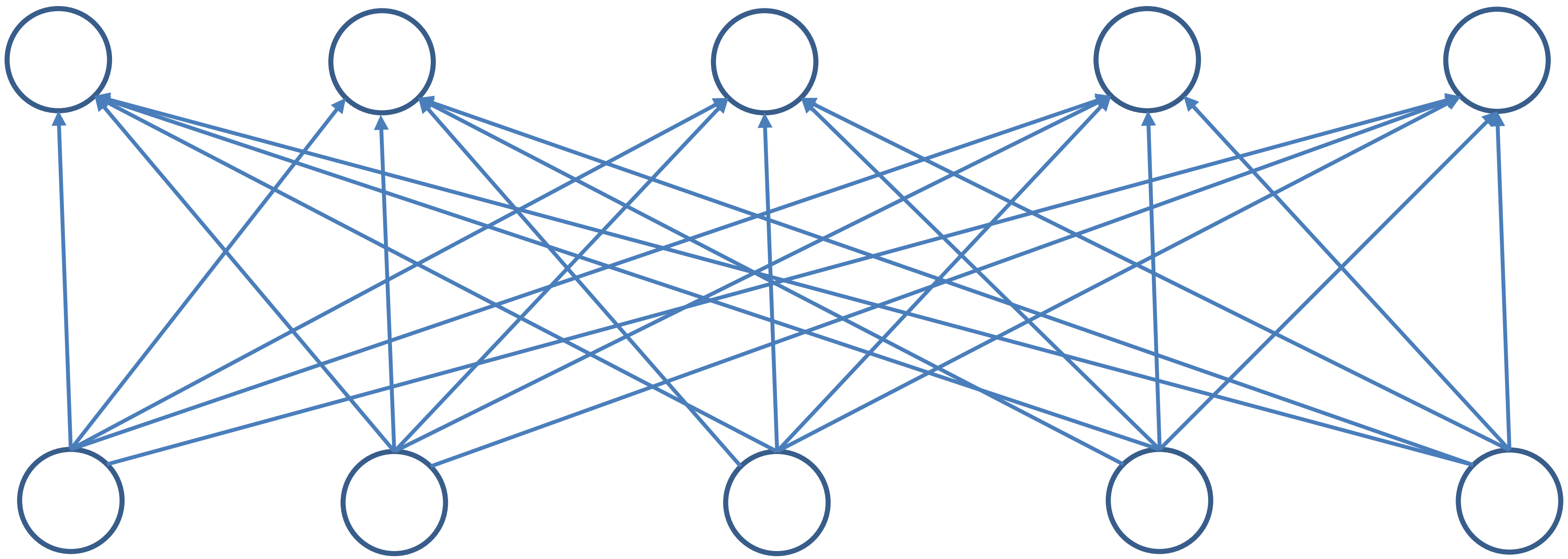
$a = \frac{1}{2}(kernel\_span + 1)$, for an odd length kernel

Parameter Sharing

- The weights of the convolutional kernel must be **learned**
- Each weight of a fully connected network must be learned independently
- CNNs are efficient to train
- CNNs use **parameter sharing**
  - **Statistical strength** from more samples per weight
  - Reduced variance of parameter estimates
- Weights are learned using backpropagation and gradient descent methods
- Also called **tied weights** or **spare interaction**

Parameter Sharing

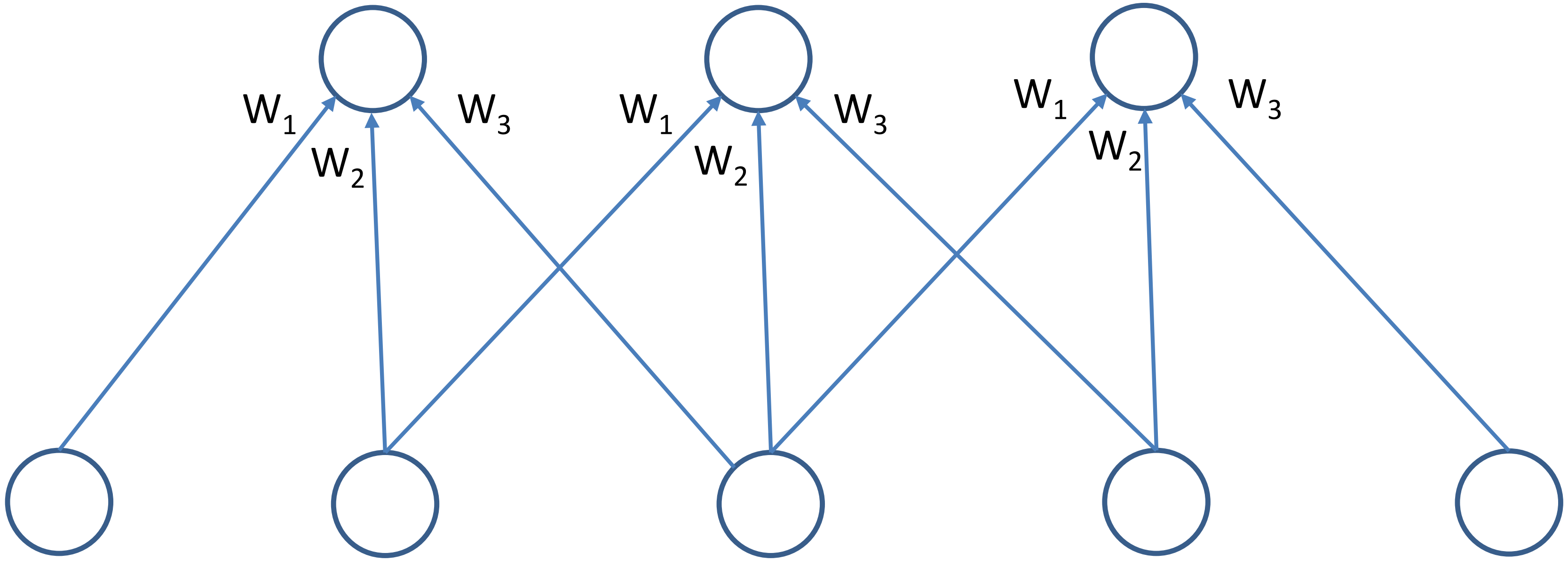
Weights of fully connected network are independent



e.g. requires $5^2 = 25$ weights

Parameter Sharing

Weights are shared in CNN



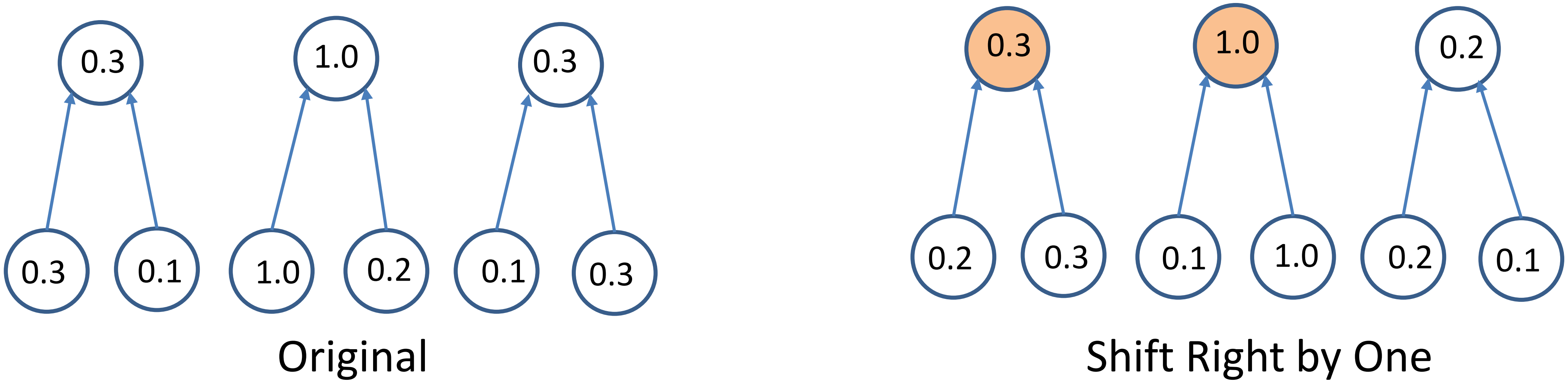
e.g. requires 3 weights

Pooling and Invariance

- Convolution provides reduced dimensionality of input tensor
- How can we obtain greater reduction in dimensionality?
- **Pooling** of convolution kernel output values reduces dimensionality
  - e.g. 2x2 operator pools the 4 values into 1
- How to pool?
  - Average?
  - Max pooling; simple and highly effective

Pooling and Invariance

Max pooling provides **invariance** to small shifts of the input tensor:

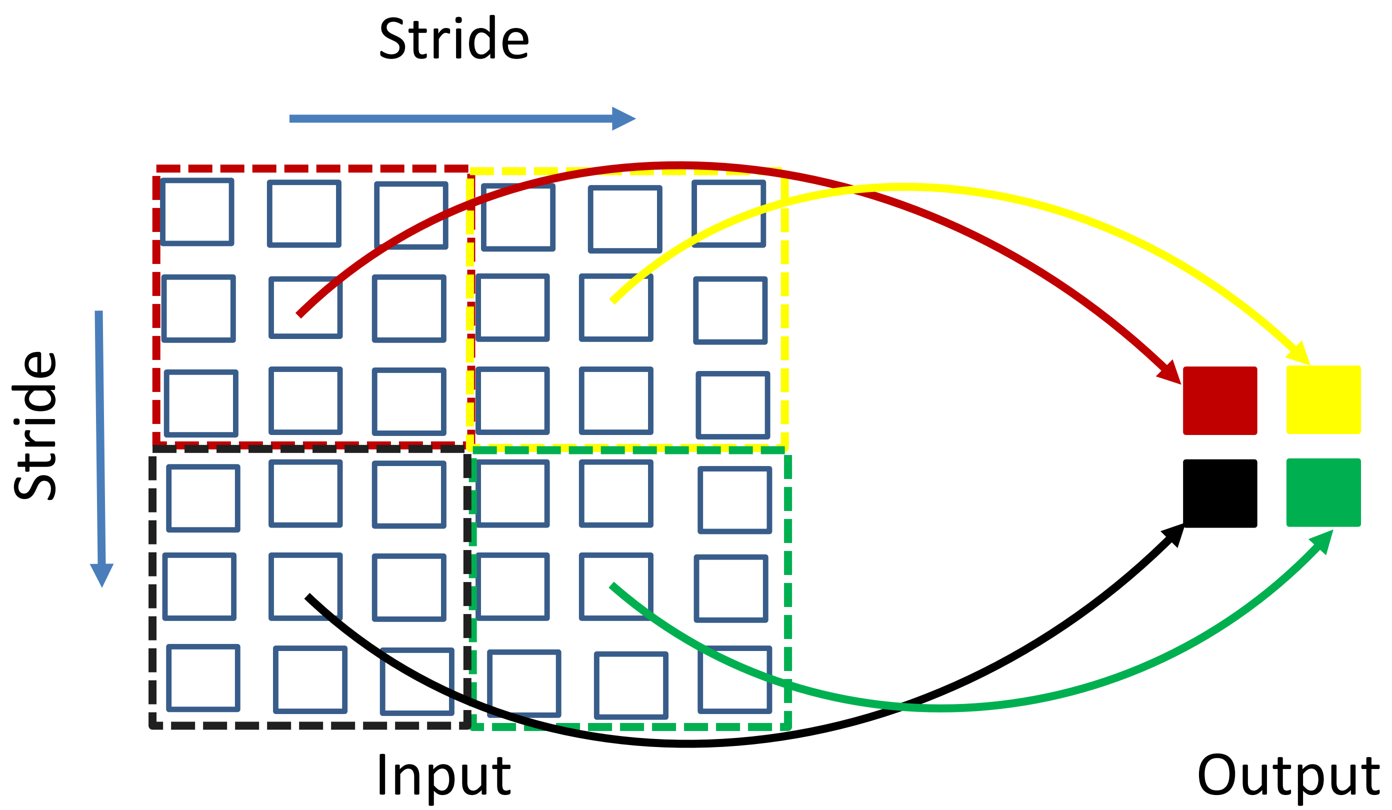


Stride and Tiling

- Do we always move the convolution kernel by 1 data value?
- **No!**
  - May not need the full resolution of the input
- We can choose a **stride > 1** for the convolution operator
  - Convolution kernel moved by stride at each step
- Stride > 1 **down-samples** the data
  - Reduces dimensionality

Stride and Tiling

Tiling is a special case when stride = span:



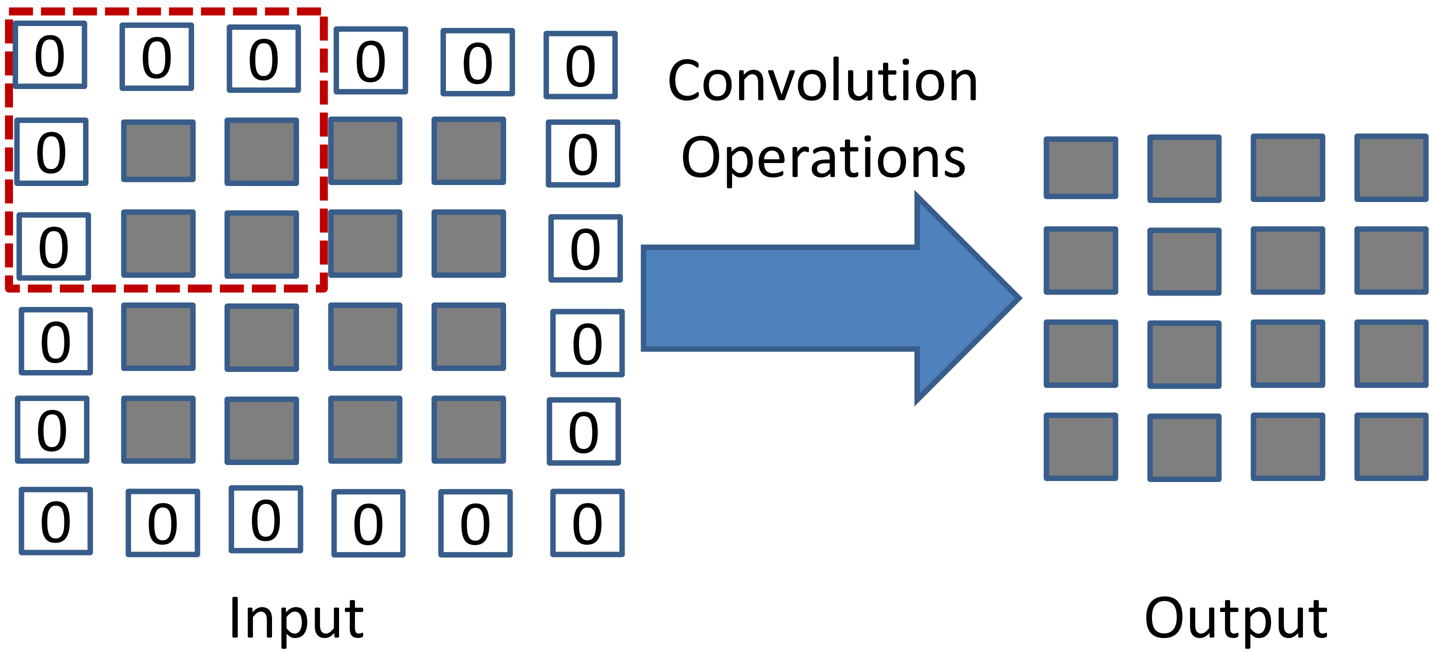
Stride >1 reduces dimensionality

Padding for Convolution

- How can we best deal with **edges of the input** tensor when performing convolution?
- A **valid convolution** confines the kernel to the input tensor dimensions
  - For odd kernel shape, output dimension is (span + 1)/2 less than input dimension
  - After many convolution layers, dimension is reduced further
- We can **zero pad** the input tensor
  - Dimensionality is maintained

Padding for Convolution

- Example: 4x4 input tensor with 3x3 kernel



- Dimension of output tensor = dimension of input tensor
- 0 values have little effect with max pooling

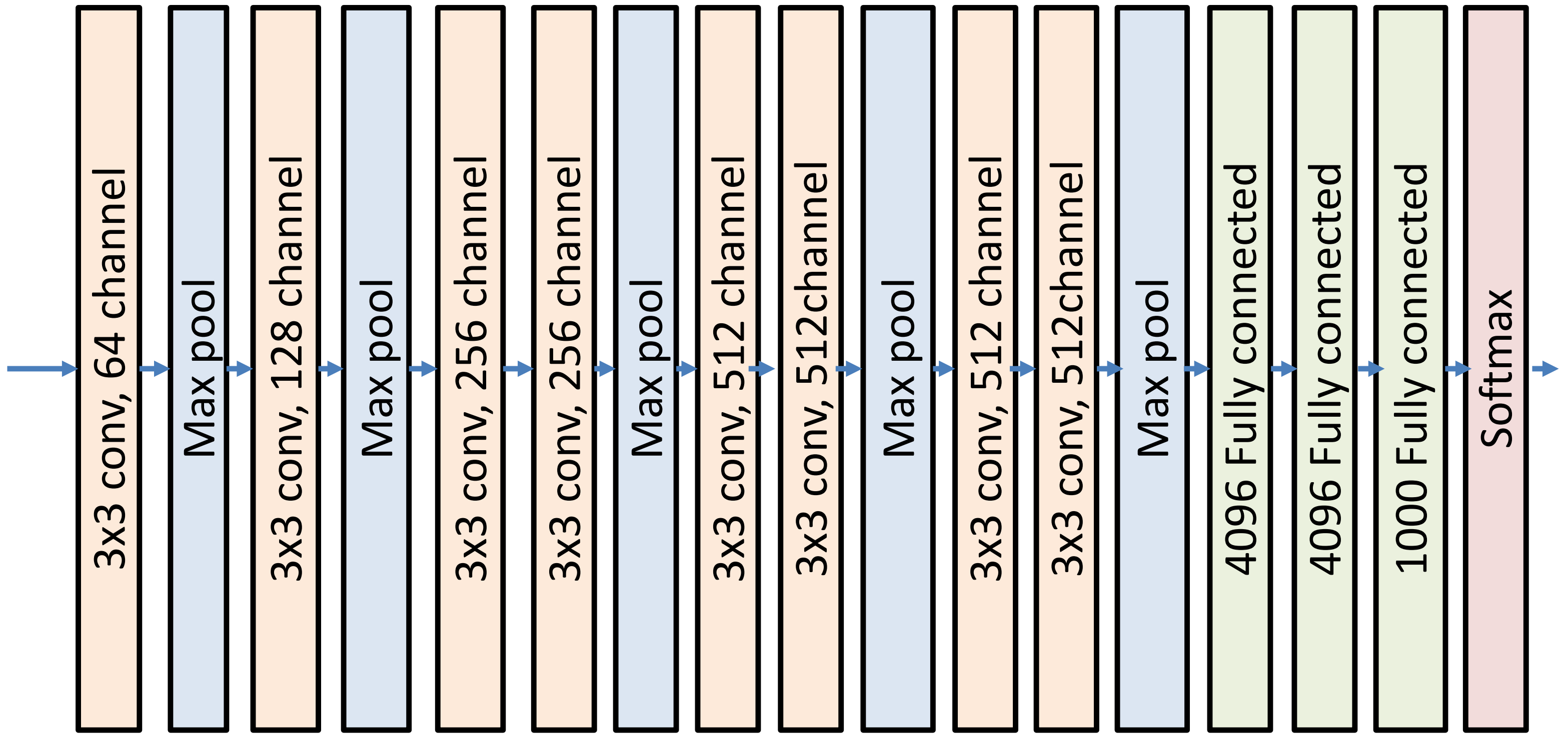
Deep and Multi-Scale Architectures

- Deep architectures create large and complex feature maps
- Feature maps have a higher number of channels
- Trained on very large benchmark datasets
- Classification accuracy found to improve with depth

Deep and Multi-Scale Architectures

VGG11

Simanyan and Zdisserman, 2015



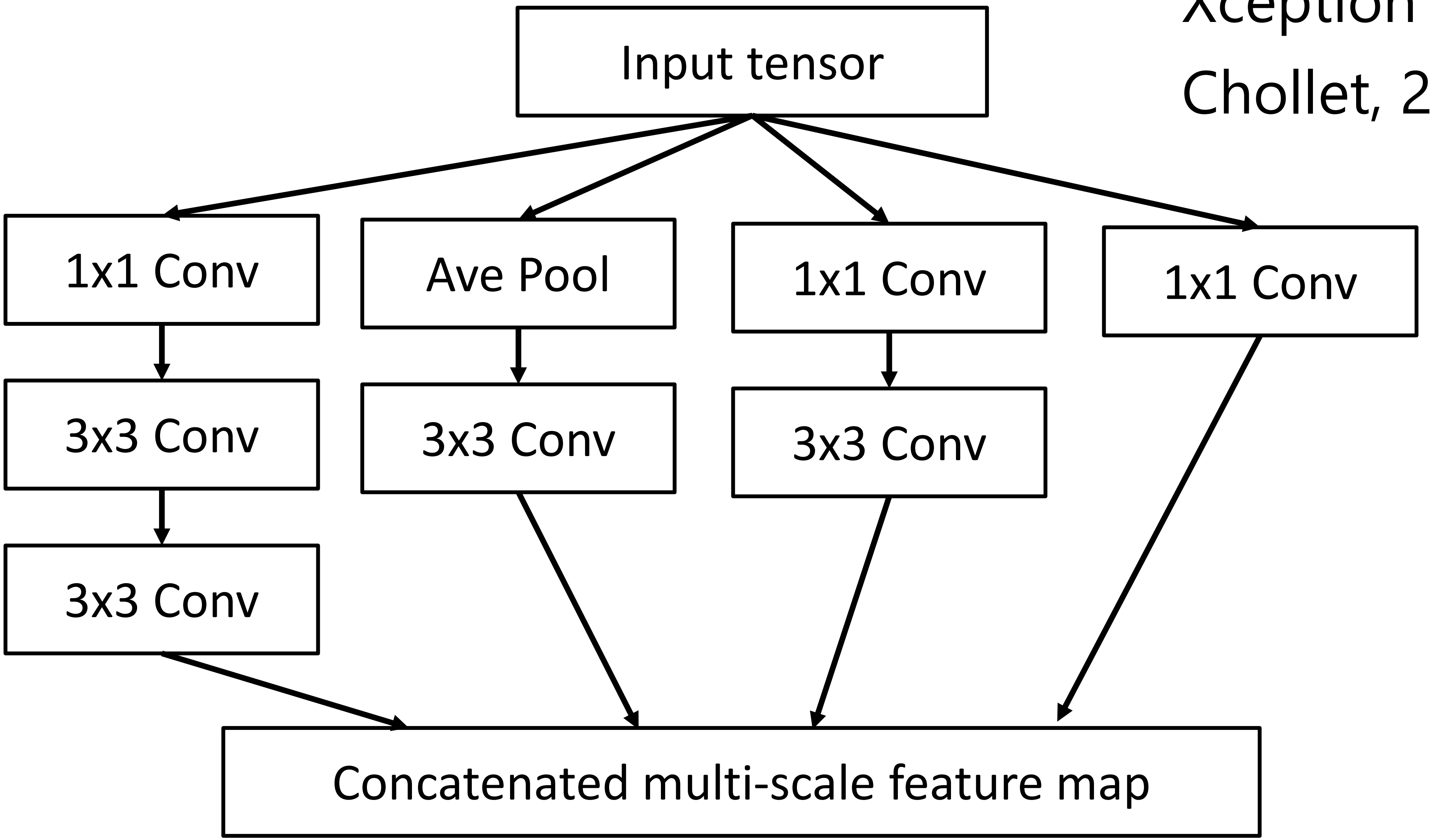
Deep and Multi-Scale Architectures

- Single convolutional layers work at single scale
- But, real-world images contain objects with different scales
- Need architecture that supports multiple scales
  - CNN layers with different scales in parallel
  - Concatenate the results
- Numerous pretrained models, using complex architectures, are now available
  - Trained on very large benchmark datasets
  - Built into deep learning frameworks; Keras, PyTorch, etc.

Deep and Multi-Scale Architectures

Xception

Chollet, 2017



Microsoft