COMS0018: Convolutional Neural Networks

Dima Damen
Dima.Damen@bristol.ac.uk

Bristol University, Department of Computer Science
Bristol BS8 1UB, UK

October 11, 2019

Introduction

- Not every Deep Neural Network (DNN) is a Convolutional Neural Network (CNN)

[1] Arguably!

Introduction

- Not every Deep Neural Network (DNN) is a Convolutional Neural Network (CNN)
- By the end of this course you will be familiar with 3 types of DNNs
  - Fully-Connected DNN
  - Convolutional DNN
  - Recurrent DNN

[1] Arguably!

Introduction

- Not every Deep Neural Network (DNN) is a Convolutional Neural Network (CNN)
- By the end of this course you will be familiar with 3 types of DNNs
  - Fully-Connected DNN
  - Convolutional DNN
  - Recurrent DNN
- CNNs could be credited for the recent success of Neural Networks[1]
- The term was first used by LeCun in his technical report: "Generalization and network design strategies" (1989).

[1] Arguably!

When to use CNNs?

- CNNs expect the **input** data x has a known **grid-like topology**
- i.e. if the order of element in the grid is shuffled, the data point is considered distinct.

When to use CNNs?

- CNNs expect the **input** data x has a known **grid-like topology**
- i.e. if the order of element in the grid is shuffled, the data point is considered distinct.
- Typical examples:
  - Audio: 1-D recurring data at regular time intervals
  - Images: 2-D grid of pixels
  - Video: 3-D (sequence of 2-D grid of pixels)

When to use CNNs?

- CNNs expect the **input** data x has a known **grid-like topology**
- i.e. if the order of element in the grid is shuffled, the data point is considered distinct.
- Typical examples:
  - Audio: 1-D recurring data at regular time intervals
  - Images: 2-D grid of pixels
  - Video: 3-D (sequence of 2-D grid of pixels)
- As the input is grid-like, operations might apply to individual or groups of grid cells.

When to use CNNs?

- CNNs expect the **input** data x has a known **grid-like topology**
- i.e. if the order of element in the grid is shuffled, the data point is considered distinct.
- Typical examples:
  - Audio: 1-D recurring data at regular time intervals
  - Images: 2-D grid of pixels
  - Video: 3-D (sequence of 2-D grid of pixels)
- As the input is grid-like, operations might apply to individual or groups of grid cells.
- Accordingly, CNN is a neural network that uses *convolution* in place of general matrix multiplication *in at least one of its layers*.

Kernels vs Tensors

- The *convolution* operation is typically denoted with $*$

$$\mathrm{x} * \omega \tag{1}$$

  where x is the **input** and ω is the **kernel**, also known as the **feature map**
- Traditionally, these kernels were manually defined for specific purposes, e.g. edge detection, tracking
- In CNNs, kernels are trained/learnt from data, for one or multiple tasks

Kernels vs Tensors

- The *convolution* operation is typically denoted with $*$

$$\mathrm{x} * \omega \tag{1}$$

  where x is the **input** and ω is the **kernel**, also known as the **feature map**
- Traditionally, these kernels were manually defined for specific purposes, e.g. edge detection, tracking
- In CNNs, kernels are trained/learnt from data, for one or multiple tasks
- Moreover, multiple *dependent* kernels are trained/learnt in one go

Kernels vs Tensors

- The *convolution* operation is typically denoted with $*$

$$\mathrm{x} * \omega \tag{1}$$

  where x is the **input** and ω is the **kernel**, also known as the **feature map**
- Traditionally, these kernels were manually defined for specific purposes, e.g. edge detection, tracking
- In CNNs, kernels are trained/learnt from data, for one or multiple tasks
- Moreover, multiple *dependent* kernels are trained/learnt in one go
- In CNN, x is a multidimensional array of data, and ω is a multidimensional array of kernels - referred to as **a tensor**

Convolution vs Correlation

- Using the convolution operator, for x and ω , the result S would be

$$S(i,j) = (\mathrm{x} * \omega)(i,j) = \sum_m \sum_n \mathrm{x}(m,n)\omega(i-m, j-n) \tag{2}$$

- A main property of convolution is that it is cummutative

$$S(i,j) = (\mathrm{x} * \omega)(i,j) = (\omega * \mathrm{x})(i,j) = \sum_m \sum_n \mathrm{x}(i-m, j-n)\omega(m,n) \tag{3}$$

Convolution vs Correlation

- Using the convolution operator, for x and ω , the result S would be

$$S(i,j) = (\mathrm{x} * \omega)(i,j) = \sum_m \sum_n \mathrm{x}(m,n)\omega(i-m, j-n) \tag{2}$$

- A main property of convolution is that it is cummutative

$$S(i,j) = (\mathrm{x} * \omega)(i,j) = (\omega * \mathrm{x})(i,j) = \sum_m \sum_n \mathrm{x}(i-m, j-n)\omega(m,n) \tag{3}$$

- The commutative property of the *convolution* operator is because we have **flipped** the kernel relative to the input - when m increases, the index of x increases but the index of ω decreases

Convolution vs Correlation

- Using the convolution operator, for x and ω , the result S would be

$$S(i,j) = (\mathrm{x} * \omega)(i,j) = \sum_m \sum_n \mathrm{x}(m,n)\omega(i-m, j-n) \tag{2}$$

- A main property of convolution is that it is cummutative

$$S(i,j) = (\mathrm{x} * \omega)(i,j) = (\omega * \mathrm{x})(i,j) = \sum_m \sum_n \mathrm{x}(i-m, j-n)\omega(m,n) \tag{3}$$

- The commutative property of the *convolution* operator is because we have **flipped** the kernel relative to the input - when m increases, the index of x increases but the index of ω decreases
- *The only reason to flip the kernel is to obtain the cummutative property - helpful in writing proofs*

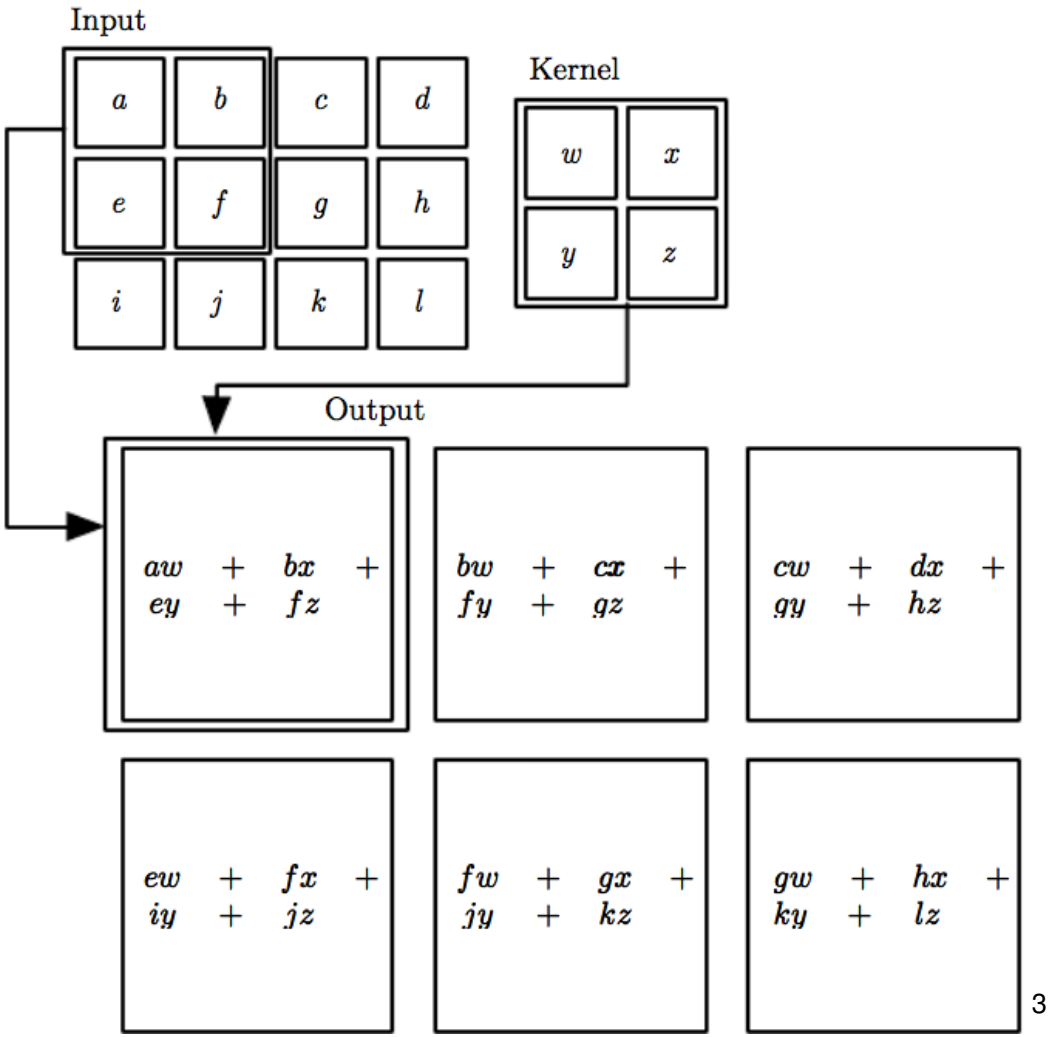
Convolution vs Correlation

- However, *most* DNN libraries implement the convolution as a **cross-correlation** operation, without flipping the kernel[2]

$$S(i,j) = (\mathrm{x} * \omega)(i,j) = \sum_m \sum_n \mathrm{x}(i+m, j+n)\omega(m,n) \tag{4}$$

[2] We do not have a good reason to call them CNNs really!

Convolution vs Correlation



[3]

[3] Reference: Goodfellow et al (2016) p325

Convolutional Neural Networks

- And now... to the main attraction **Convolutional Neural Networks (CNN)**

Convolutional Neural Networks

- And now... to the main attraction **Convolutional Neural Networks (CNN)**
- Three primary properties distinguish fully-connected networks from convolutional neural networks:

Convolutional Neural Networks

- And now... to the main attraction **Convolutional Neural Networks (CNN)**
- Three primary properties distinguish fully-connected networks from convolutional neural networks:
  1. Sparse Interactions
  2. Parameter Sharing
  3. Equi-variant Representations

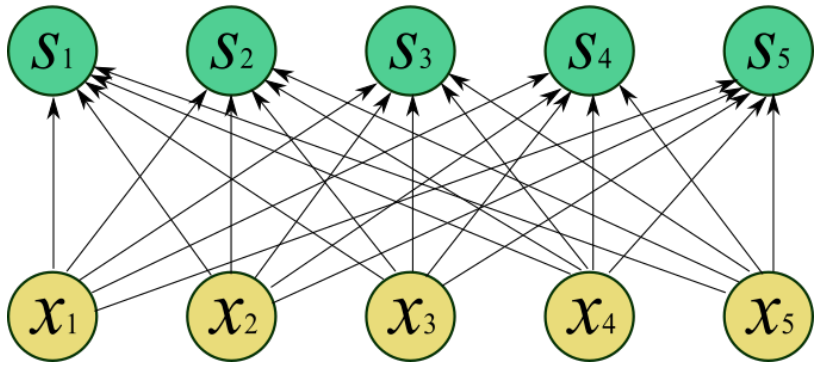
CNN Properties: 1- Sparse Interactions[4]

- A major difference between fully connected neural networks and CNNs are the contributions of input units to output units.

[4] Also referred to as multi-scale interactions

CNN Properties: 1- Sparse Interactions[4]

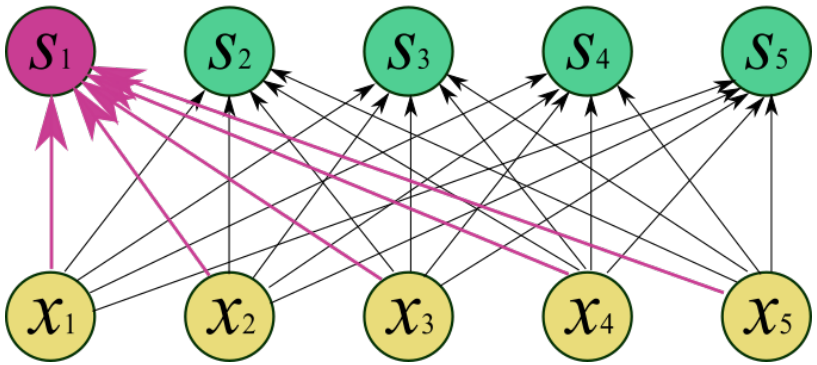
- A major difference between fully connected neural networks and CNNs are the contributions of input units to output units.
- Consider this two-layer fully-connected network, with 5 input units,



[4] Also referred to as multi-scale interactions

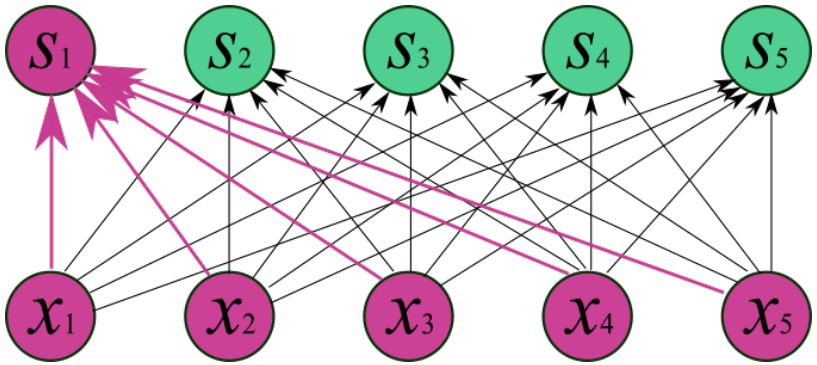
CNN Properties: 1- Sparse Interactions

- A major difference between fully connected neural networks and CNNs are the contributions of input units to output units.
- For one output unit s_1,



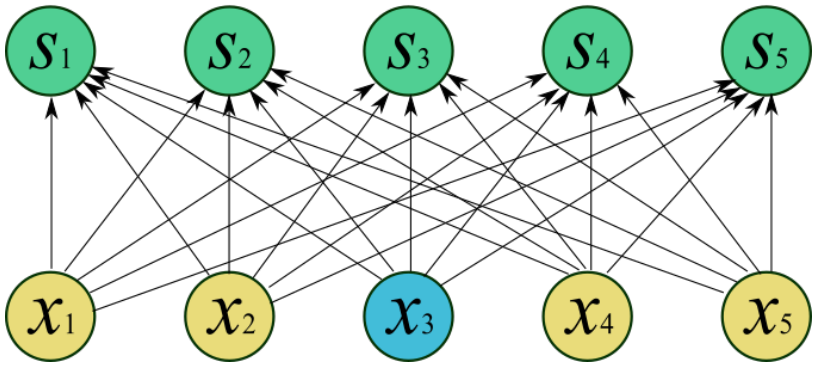
CNN Properties: 1- Sparse Interactions

- A major difference between fully connected neural networks and CNNs are the contributions of input units to output units.
- its value is decided from all 5 input units
  $s_1 = f(x_1, x_2, x_3, x_4, x_5; \omega_1, \omega_2, \omega_3, \omega_4, \omega_5)$.



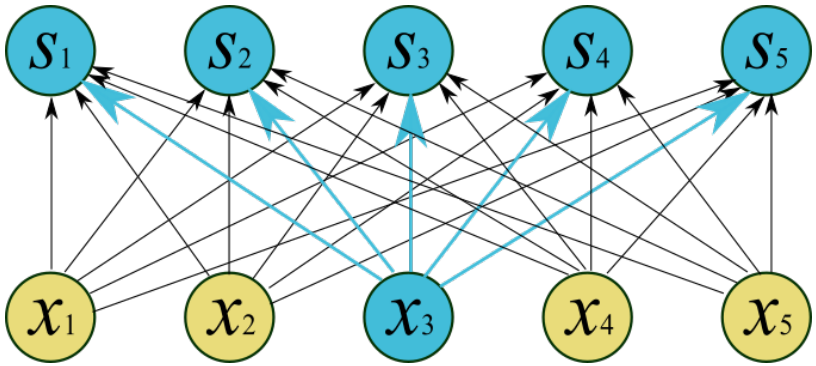
CNN Properties: 1- Sparse Interactions

- A major difference between fully connected neural networks and CNNs are the contributions of input units to output units.
- similarly, each input unit, e.g. x_3,



CNN Properties: 1- Sparse Interactions

- A major difference between fully connected neural networks and CNNs are the contributions of input units to output units.
- similarly, each input unit, e.g. x_3, contributes to all output units

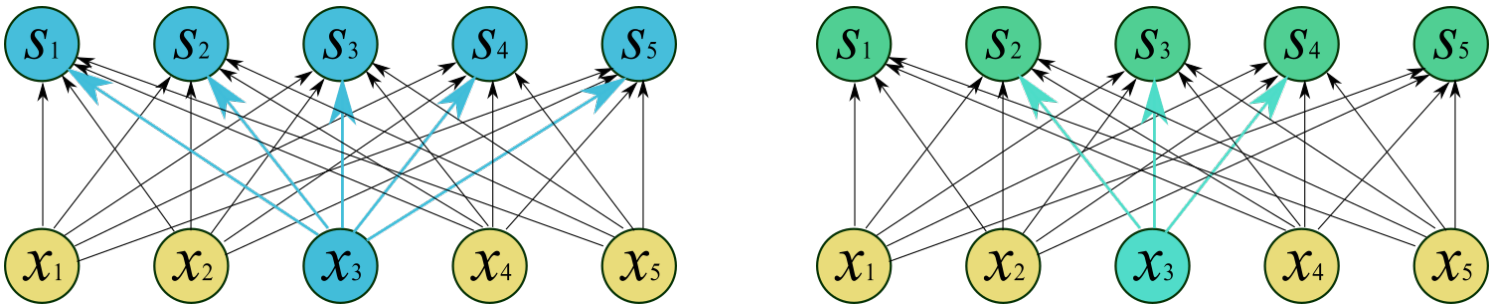


CNN Properties: 1- Sparse Interactions

- In **CNNs**, due to the grid structure, it is sufficient to limit the number of connections from each input unit unit to k,

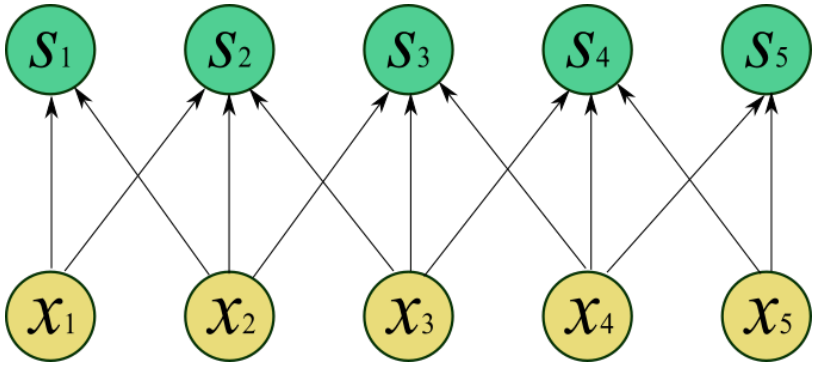
CNN Properties: 1- Sparse Interactions

- In **CNNs**, due to the grid structure, it is sufficient to limit the number of connections from each input unit unit to k,
- See the connections from x_3



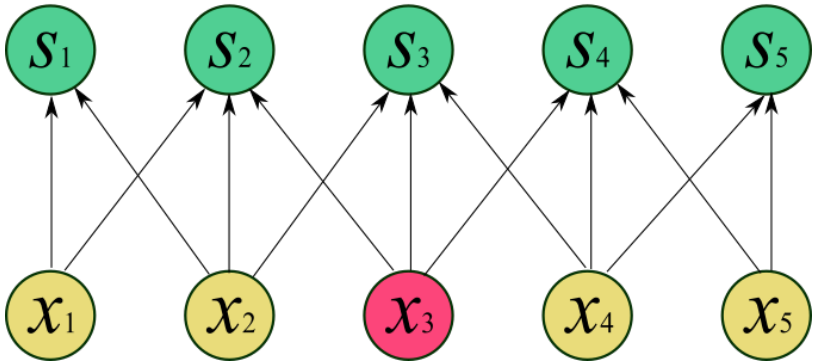
CNN Properties: 1- Sparse Interactions

- In **CNNs**, due to the grid structure, it is sufficient to limit the number of connections from each input unit unit to k,
- resulting in sparse weights - and sparse interactions between input and output



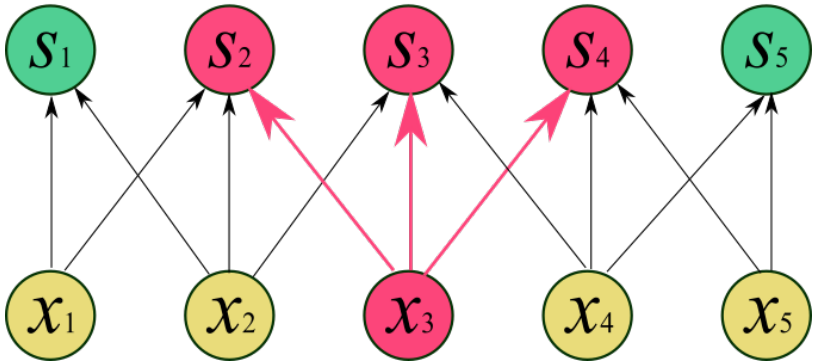
CNN Properties: 1- Sparse Interactions

- In **CNNs**, one input unit x_3,



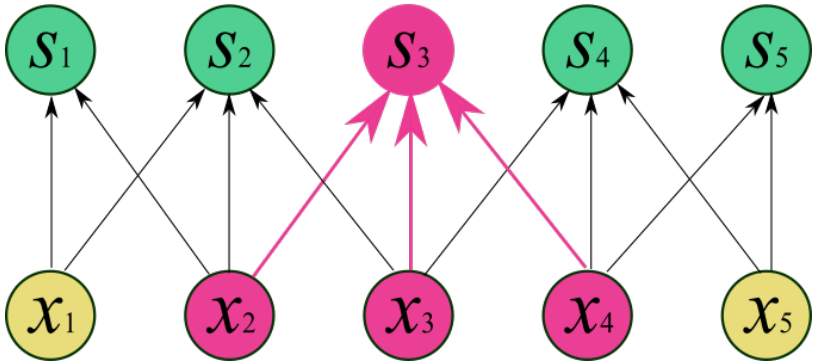
CNN Properties: 1- Sparse Interactions

- In **CNNs**, one input unit x_3, affects a limited number of output units



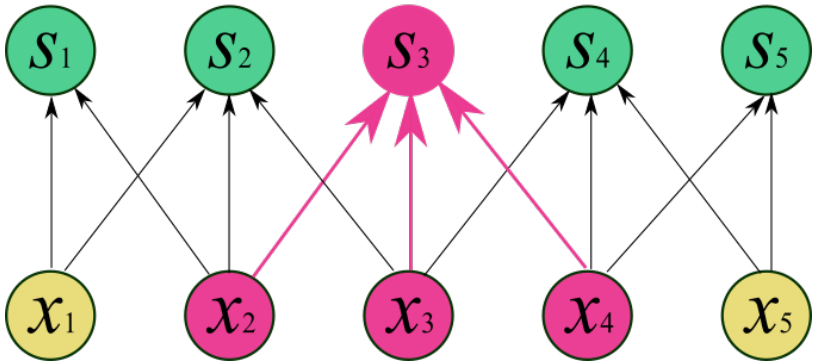
CNN Properties: 1- Sparse Interactions

- Similarly, the input units affecting a certain output unit (e.g. s_3),



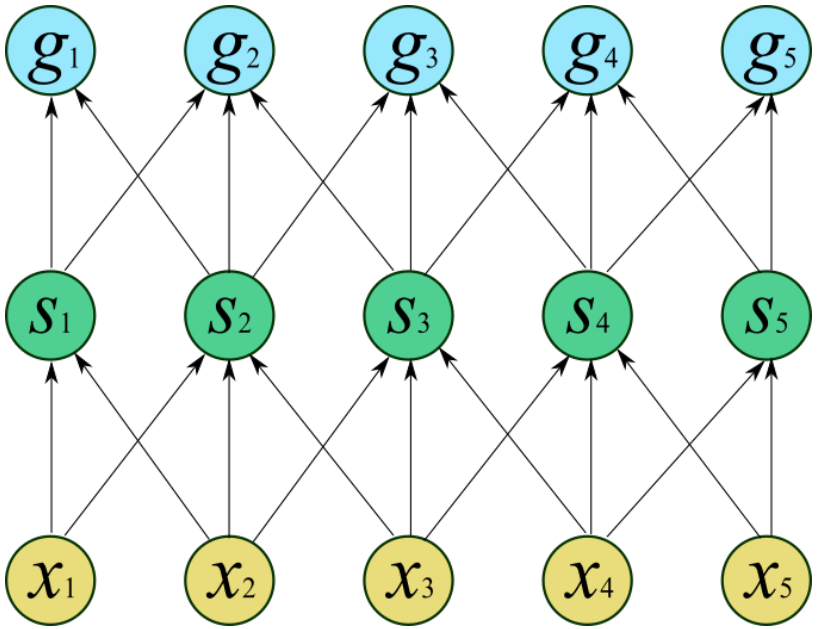
CNN Properties: 1- Sparse Interactions

- The input units affecting a certain output unit (e.g. s_3), are known as the unit's **receptive field**.



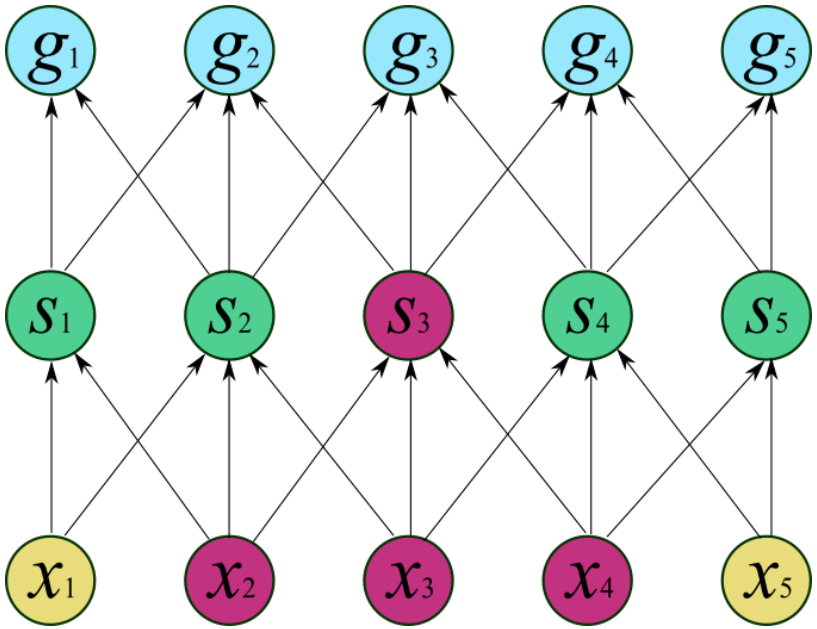
CNN Properties: 1- Sparse Interactions

- Interestingly, as more layers are added,



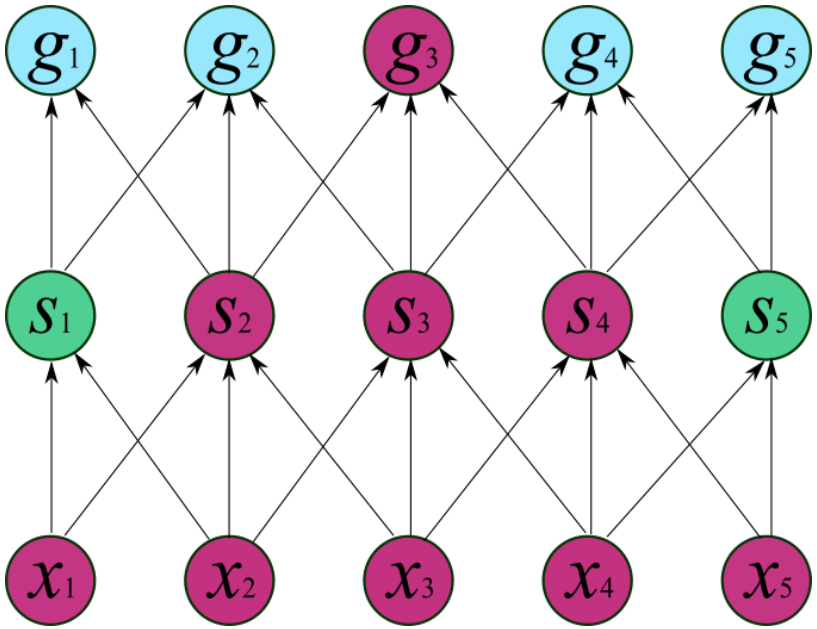
CNN Properties: 1- Sparse Interactions

- Interestingly, as more layers are added,



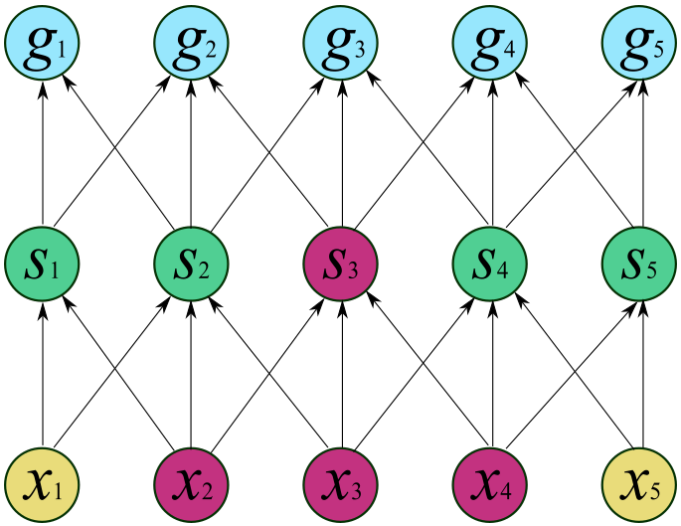
CNN Properties: 1- Sparse Interactions

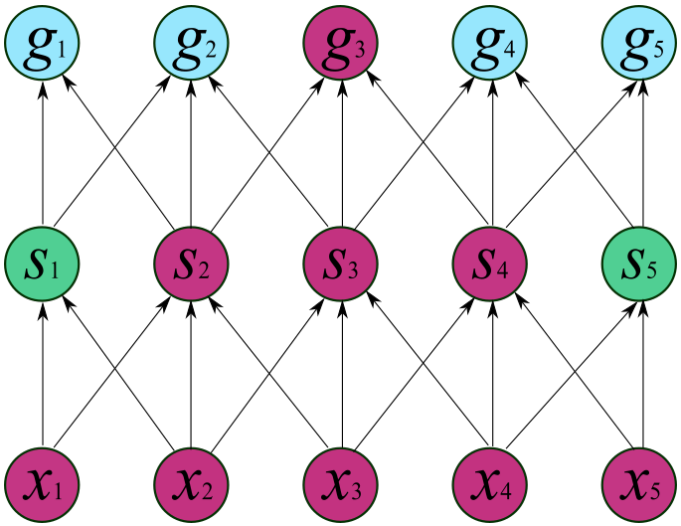
- Interestingly, as more layers are added,



CNN Properties: 1- Sparse Interactions

- **The receptive field of the units in the deeper layers of a CNN is larger than the receptive field of the units in the shallow layers**





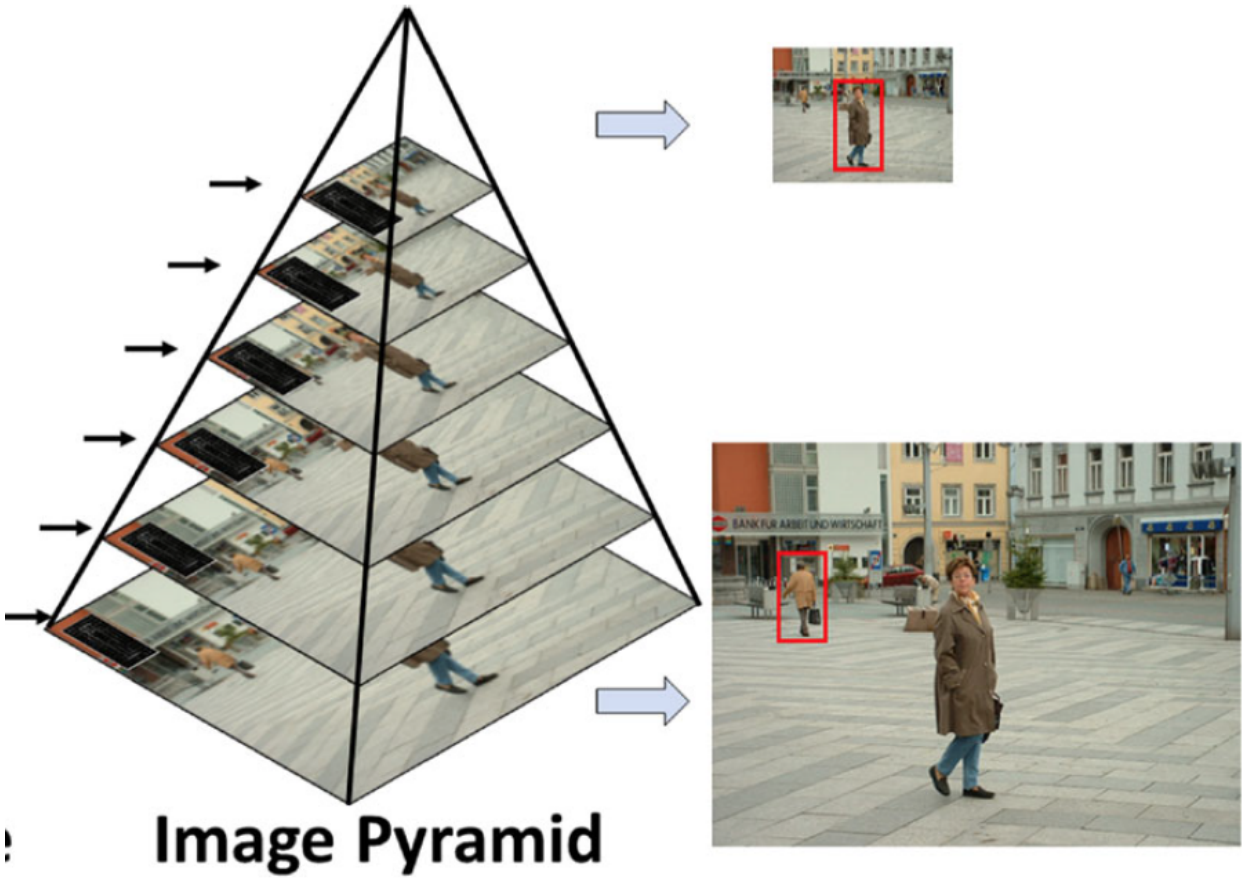
CNN Properties: 1- Sparse Interactions

- Is this new?

Suleiman and Sze, An Energy-Efficient Hardware Implementation of HOG-Based Object Detection, 2015

CNN Properties: 1- Sparse Interactions

- Is this new?



Suleiman and Sze, An Energy-Efficient Hardware Implementation of HOG-Based Object Detection, 2015

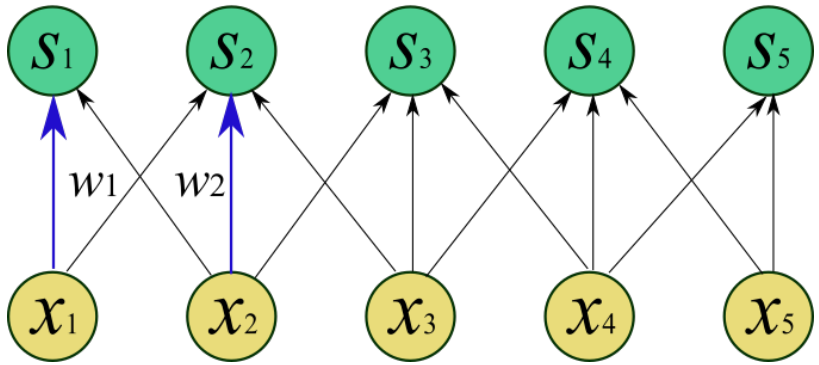
CNN Properties: 2- Parameter Sharing

CNN Properties: 2- Parameter Sharing

- **Paramter sharing** refers to using the same parameter for more than one function in the network

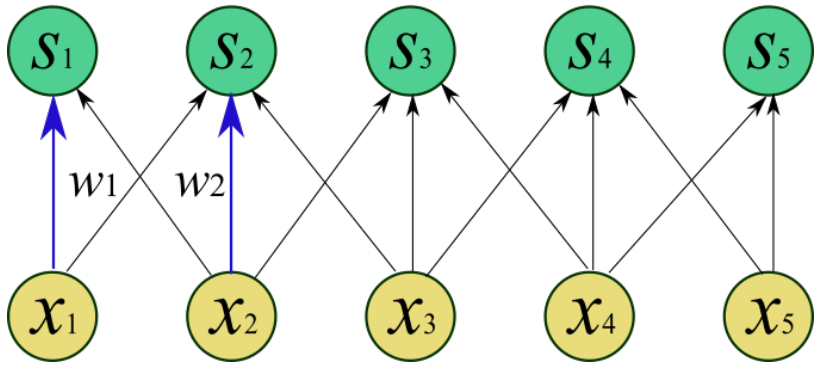
CNN Properties: 2- Parameter Sharing

- **Paramter sharing** refers to using the same parameter for more than one function in the network
- You can consider this as tying two paramters w_1 and w_2 together, so they can only have the same value



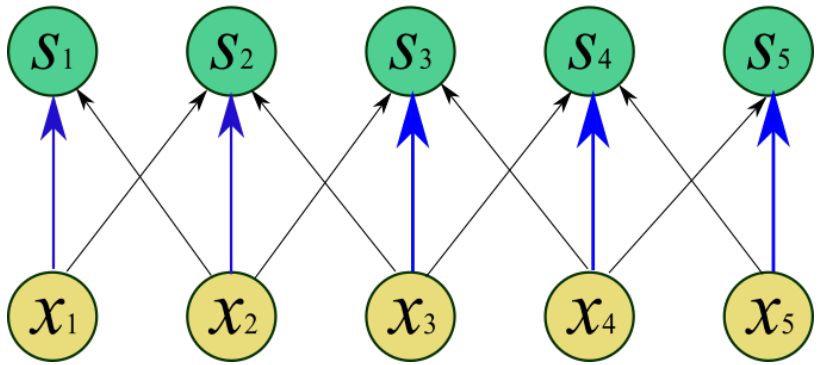
CNN Properties: 2- Parameter Sharing

- **Paramter sharing** refers to using the same parameter for more than one function in the network
- You can consider this as tying two paramters w_1 and w_2 together, so they can only have the same value
- You have dropped the number of parameters you need to train by 1 (!)



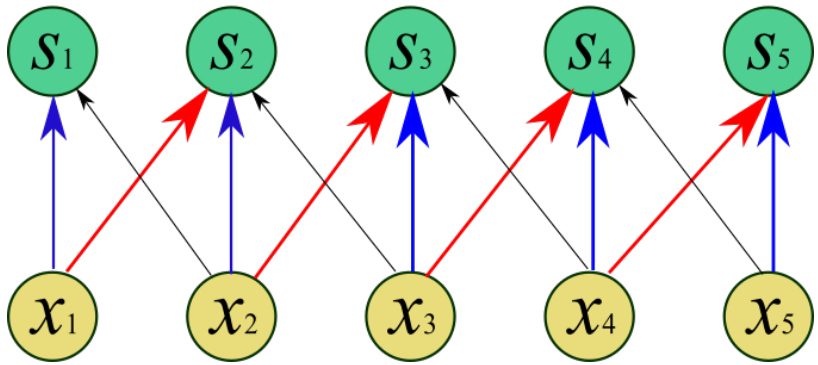
CNN Properties: 2- Parameter Sharing

- You can similarly think about sharing more parameters



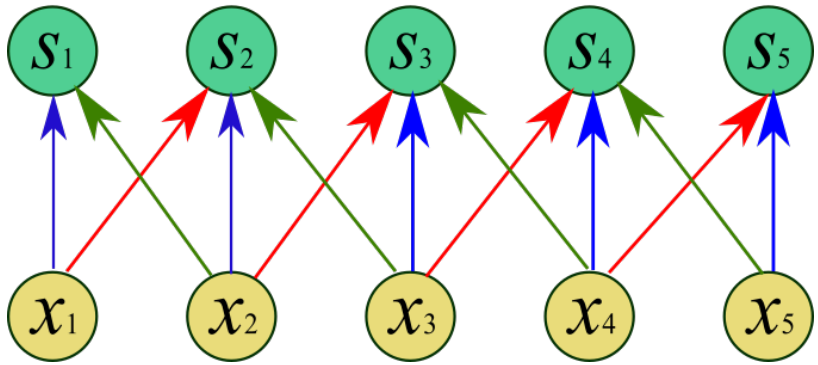
CNN Properties: 2- Parameter Sharing

- You can similarly think about sharing more parameters



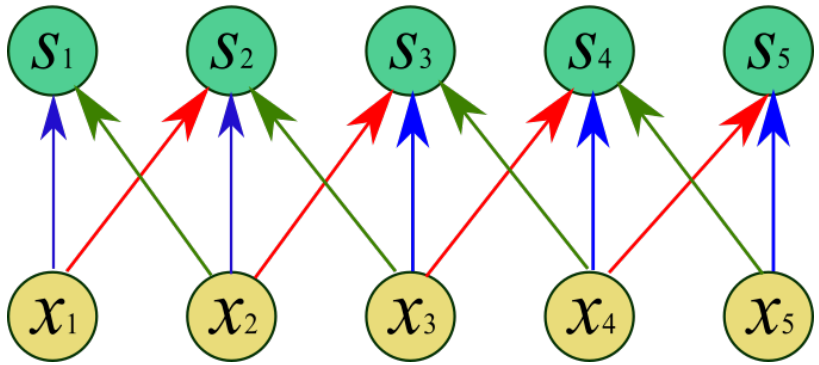
CNN Properties: 2- Parameter Sharing

- You can similarly think about sharing more parameters



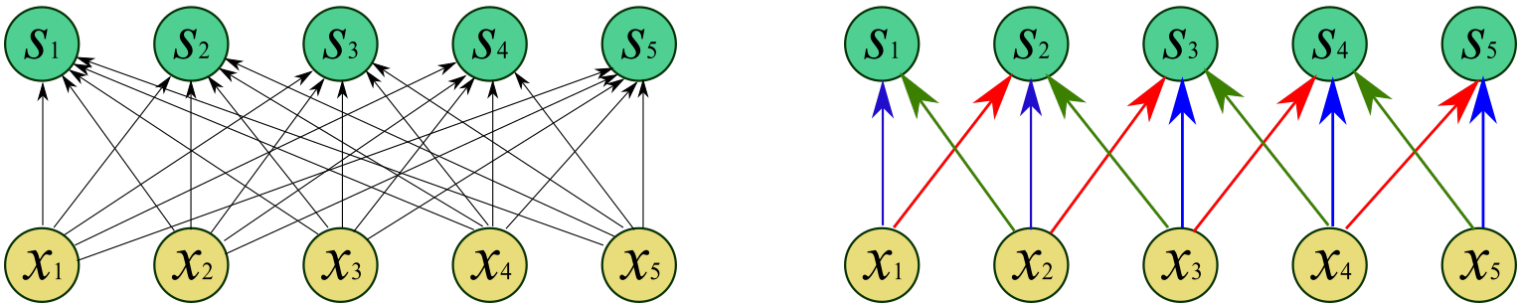
CNN Properties: 2- Parameter Sharing

- Though parameter sharing on this network - with sparse interactions - the number of parameters to train is... 3 !!!



CNN Properties: 2- Parameter Sharing

- Compare the number of parameters in the fully-connected network to this CNN with sparse interactions and parameter sharing!
- Only 12% !!! :-)



CNN Properties: 2- Parameter Sharing

- Parameter sharing is also known as **tied weights**, because the weight applied to one input is **tied** to the weight applied elsewhere.

CNN Properties: 2- Parameter Sharing

- Parameter sharing is also known as **tied weights**, because the weight applied to one input is **tied** to the weight applied elsewhere.
- Does not affect the runtime of the forward pass
- Does significantly reduce the memory requirements for the model

CNN Properties: 2- Parameter Sharing

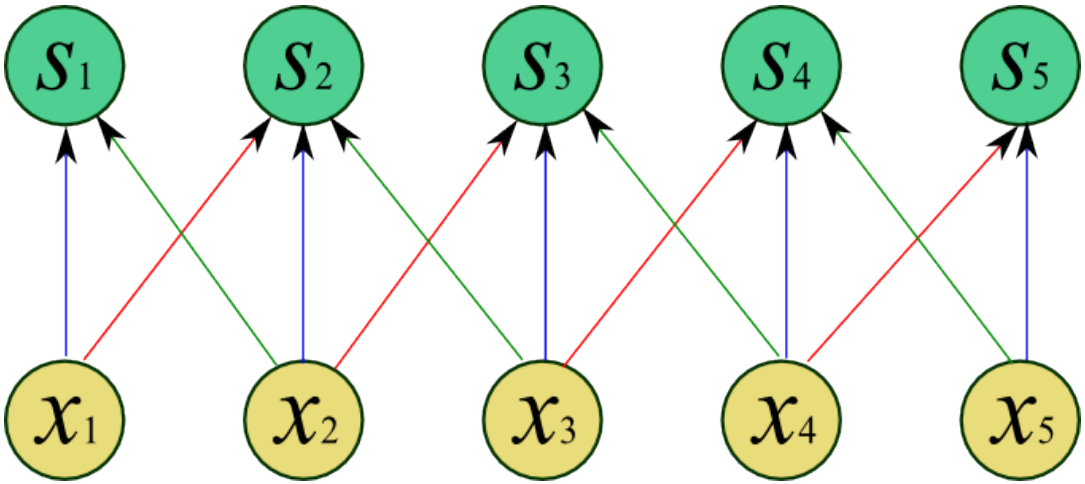
- Parameter sharing is also known as **tied weights**, because the weight applied to one input is **tied** to the weight applied elsewhere.
- Does not affect the runtime of the forward pass
- Does significantly reduce the memory requirements for the model
- You have significantly less parameters to train, and thus you need less data

CNN Properties: 2- Parameter Sharing

- Parameter sharing is also known as **tied weights**, because the weight applied to one input is **tied** to the weight applied elsewhere.
- Does not affect the runtime of the forward pass
- Does significantly reduce the memory requirements for the model
- You have significantly less parameters to train, and thus you need less data
- But only works on the assumption that the data is grid-like and thus sharing the weights is a sensible idea!

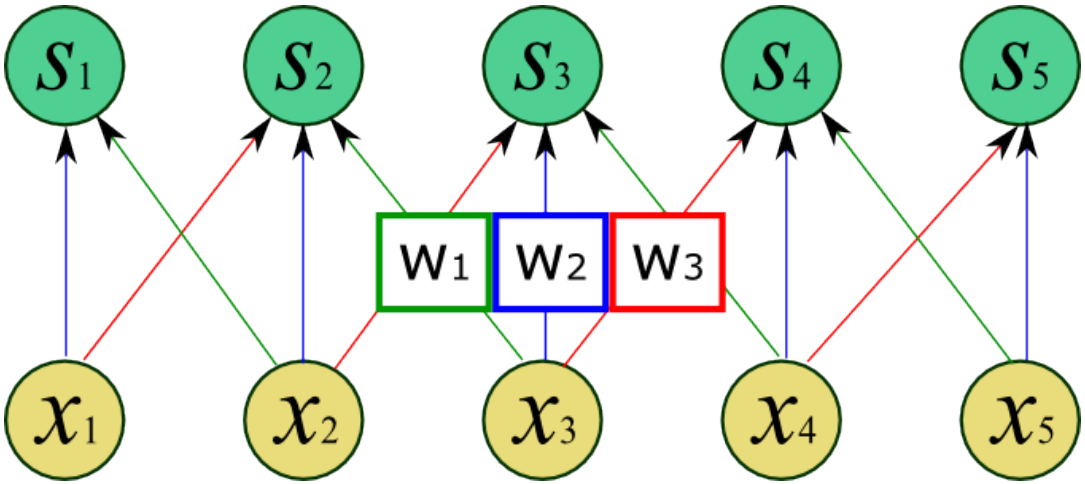
CNN Properties: 2- Parameter Sharing

- It this new??



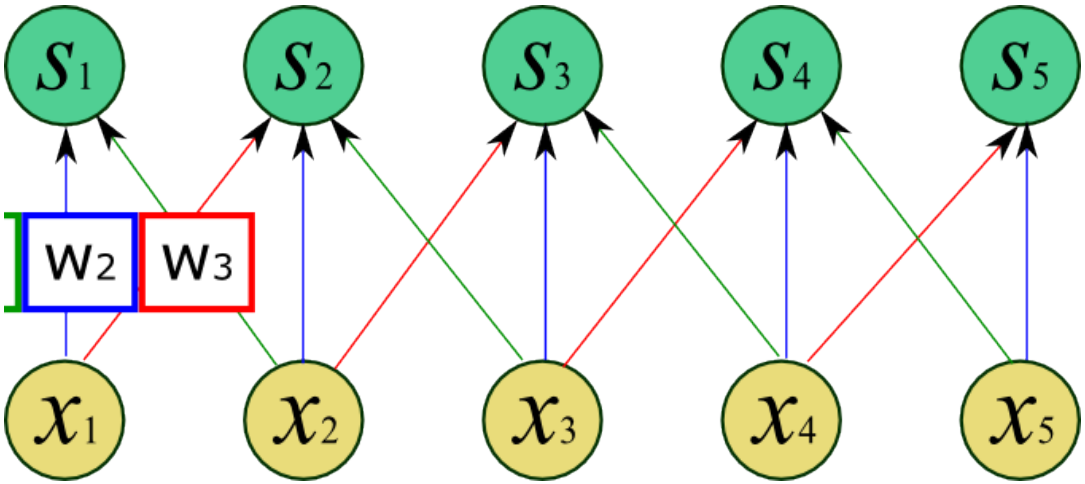
CNN Properties: 2- Parameter Sharing

- It this new??



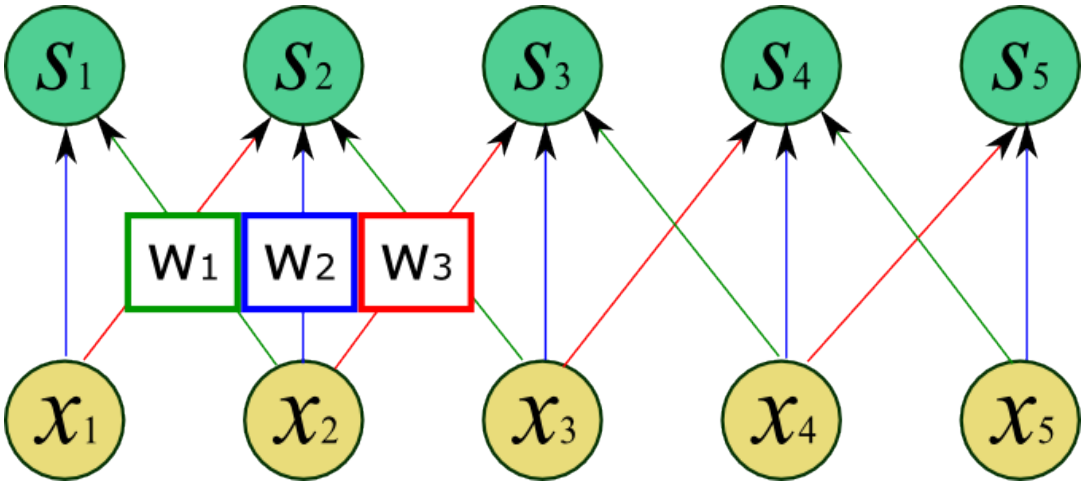
CNN Properties: 2- Parameter Sharing

- It this new??



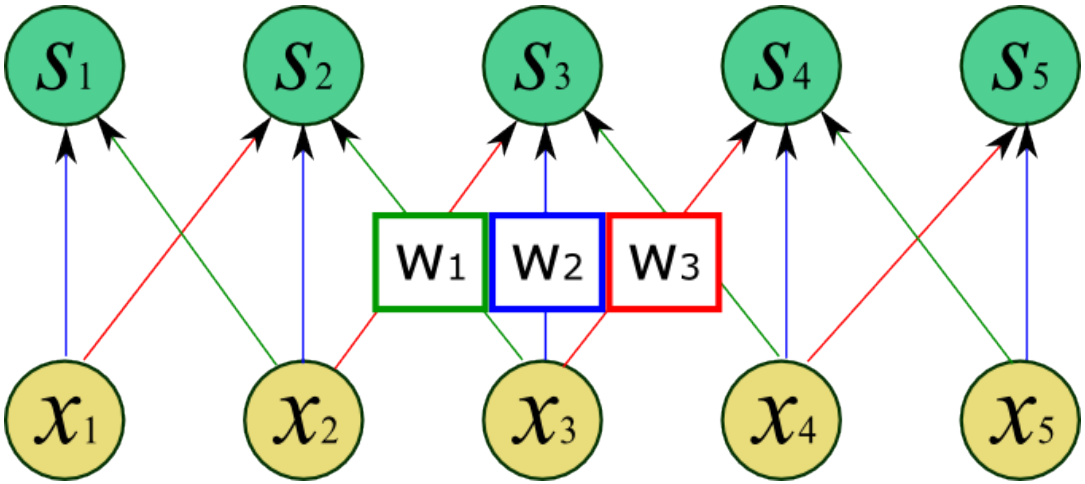
CNN Properties: 2- Parameter Sharing

- It this new??



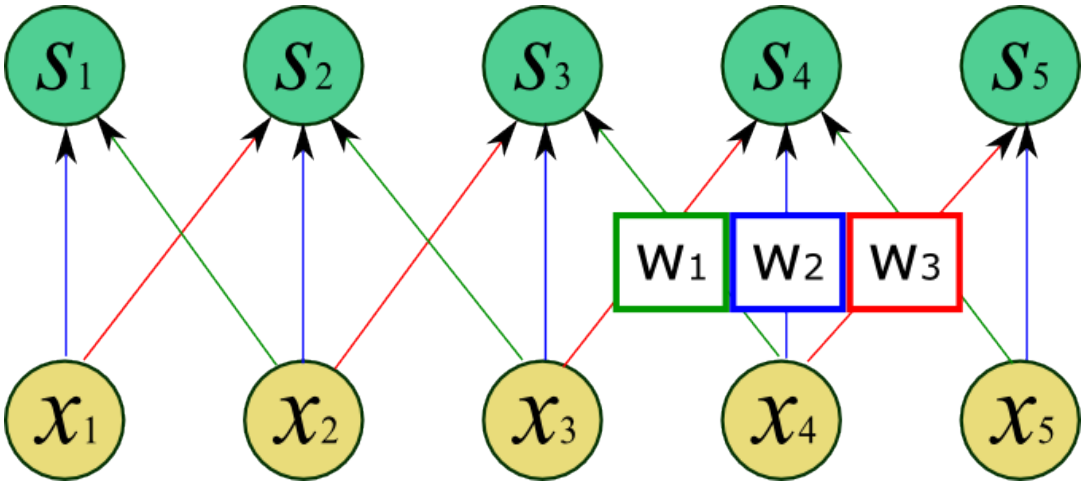
CNN Properties: 2- Parameter Sharing

- It this new??



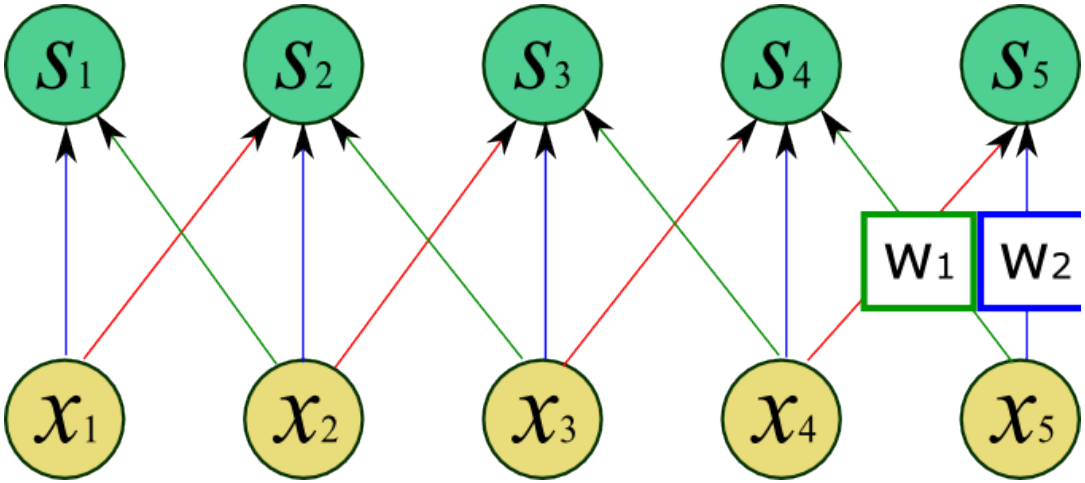
CNN Properties: 2- Parameter Sharing

- It this new??



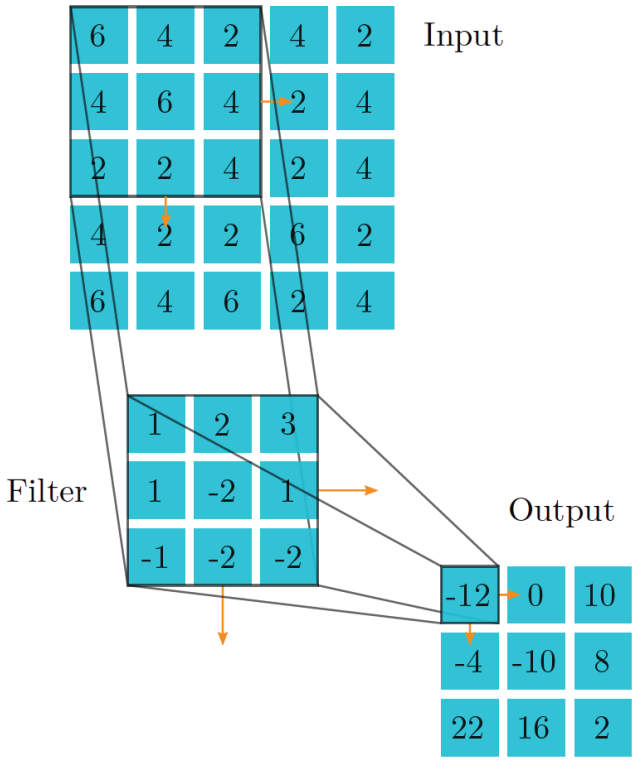
CNN Properties: 2- Parameter Sharing

- It this new?? **CONVOLUTION!!!** - or cross-correlation :-)



CNN Properties: 2- Parameter Sharing

- And in 2-D



Source: BSc Thesis, Will Price, Univ of Bristol, May 2017

CNN Properties: 3- Equi-variant Representations

CNN Properties: 3- Equi-variant Representations

- As a result of the first two properties, CNNs exhibit some equivariance properties.

CNN Properties: 3- Equi-variant Representations

- As a result of the first two properties, CNNs exhibit some equivariance properties.
- A function is equivariant if when the input changes (or shifts) in a certain way, the output also changes in exactly the same way.

CNN Properties: 3- Equi-variant Representations

- As a result of the first two properties, CNNs exhibit some equivariance properties.
- A function is equivariant if when the input changes (or shifts) in a certain way, the output also changes in exactly the same way.
- CNNs are equi-variant to... translation
- This is of immense value in images for example. If an object moves in the input, its representation will move in the output in the same direction.

CNN Properties: 3- Equi-variant Representations

- As a result of the first two properties, CNNs exhibit some equivariance properties.
- A function is equivariant if when the input changes (or shifts) in a certain way, the output also changes in exactly the same way.
- CNNs are equi-variant to... translation
- This is of immense value in images for example. If an object moves in the input, its representation will move in the output in the same direction.
- However CNNs are NOT equivariant to...

CNN Properties: 3- Equi-variant Representations

- As a result of the first two properties, CNNs exhibit some equivariance properties.
- A function is equivariant if when the input changes (or shifts) in a certain way, the output also changes in exactly the same way.
- CNNs are equi-variant to... translation
- This is of immense value in images for example. If an object moves in the input, its representation will move in the output in the same direction.
- However CNNs are NOT equivariant to... rotation or scale

Your first CNN Layer

- Multiple convolutional layers $\rightarrow$ You can learn multiple features, e.g.

Source: Rob Fergus, NN, MLSS2015 Summer School Presentation

Your first CNN Layer

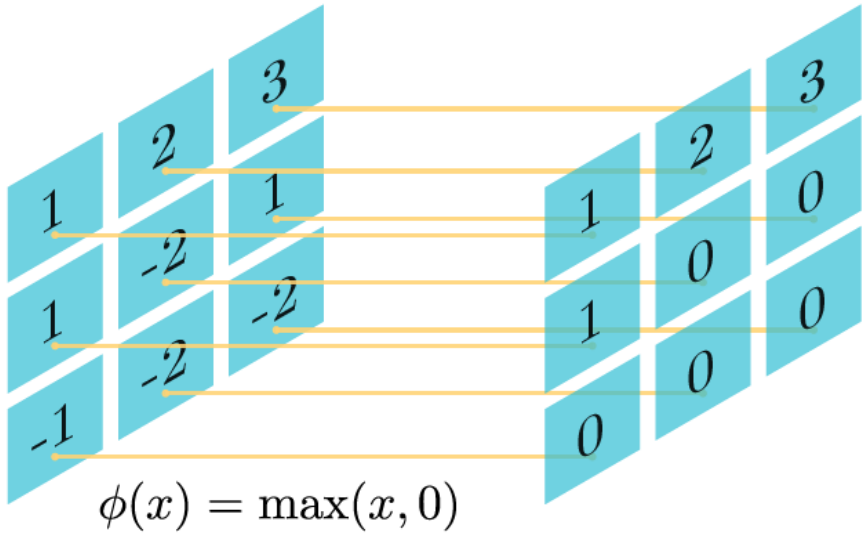
- However, to make the most of the input, particularly around the edges/borders, one essential feature of any CNN implementation is **zero padding** the input to make it wider
- Without zero-padding, the input shrinks by one pixel less than the kernel width at each layer
- With zero-padding, the input and output are of the same size, unlike example below

Your first CNN Layer

- However, to make the most of the input, particularly around the edges/borders, one essential feature of any CNN implementation is **zero padding** the input to make it wider
- Without zero-padding, the input shrinks by one pixel less than the kernel width at each layer
- With zero-padding, the input and output are of the same size, unlike example below
- Without zero-padding, the number of convolutional layers that can be included in a network will be capped

Your first CNN Layer

- The convolutions are directly followed by activation functions, in the same fashion as fully-connected CNNs
- RELU activation function is shown in the example below

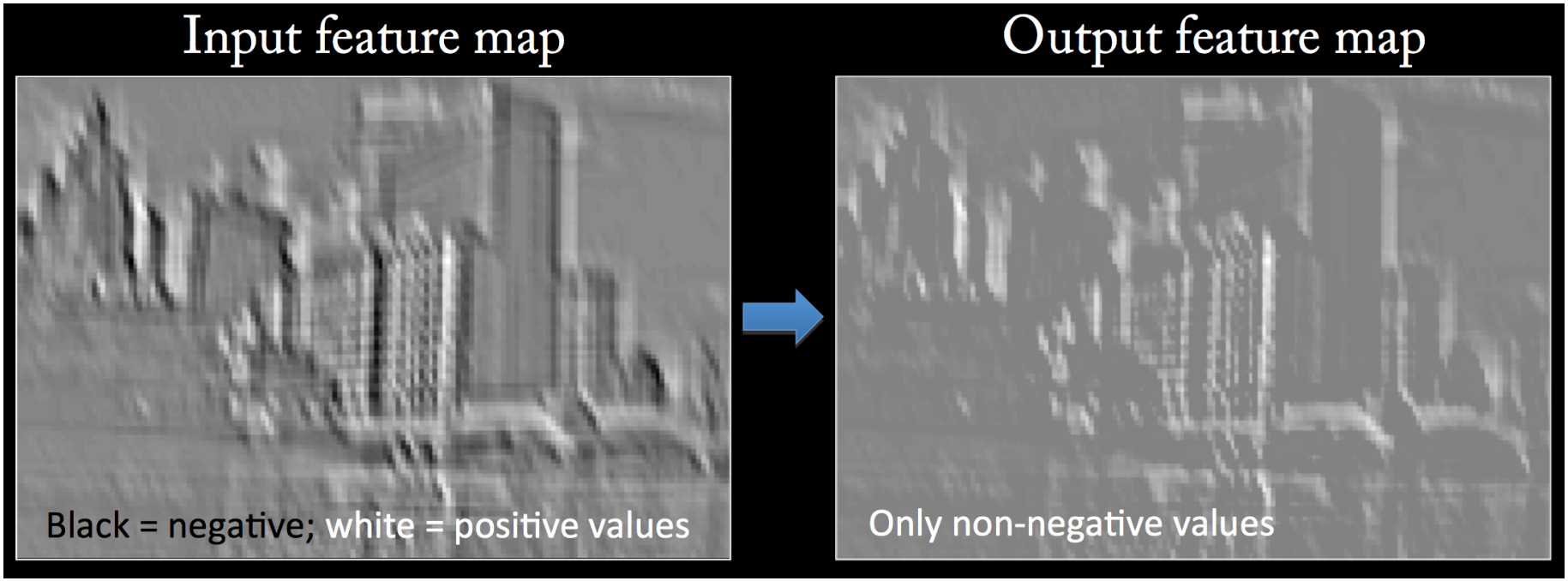


[5]

[5] Source: BSc Thesis, Will Price, Univ of Bristol, May 2017

Your first CNN Layer

- The convolutions are directly followed by activation functions, in the same fashion as fully-connected CNNs
- RELU activation function is shown in the example below

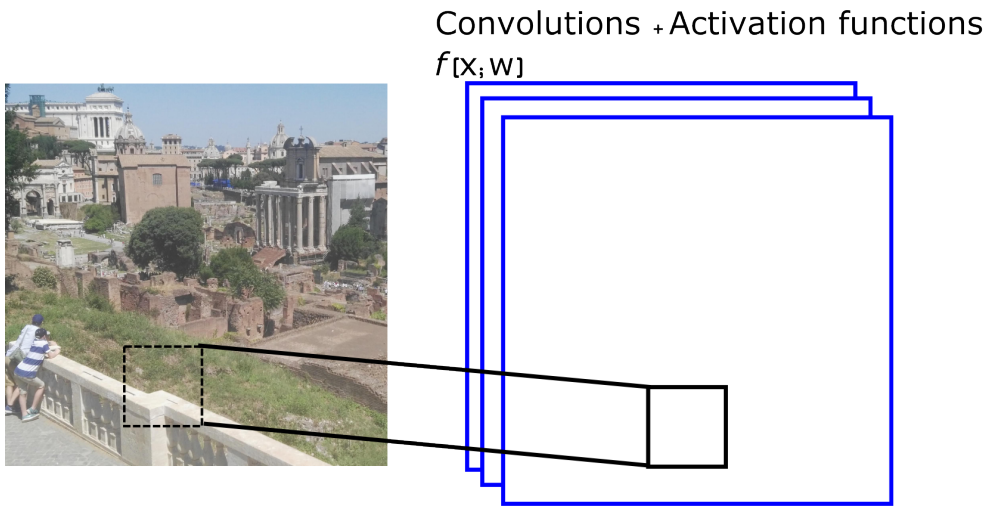


6

[6] Source: Rob Fergus, NN, MLSS2015 Summer School Presentation

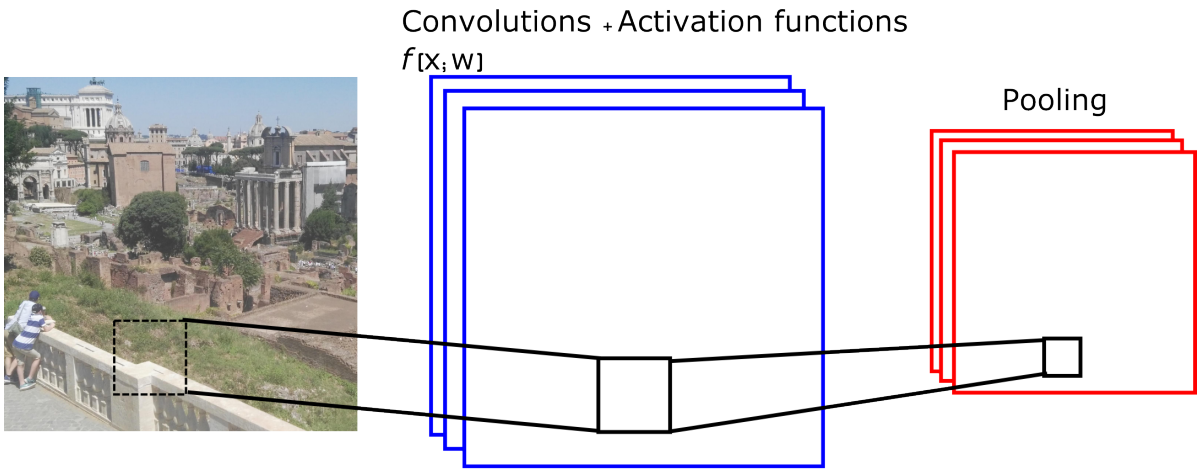
Your first CNN Layer

- Multiple convolutions can be piled
- Convolving a single kernel can extract one kind of feature
- We want to extract many kinds of features at many locations



Your first CNN Layer

- **Pooling functions** are added to modify the output layer further, typically its size.
- A pooling function **replaces** the output of the net at a certain location, with a **summary** of the outputs in nearby outputs.



Your first CNN Layer

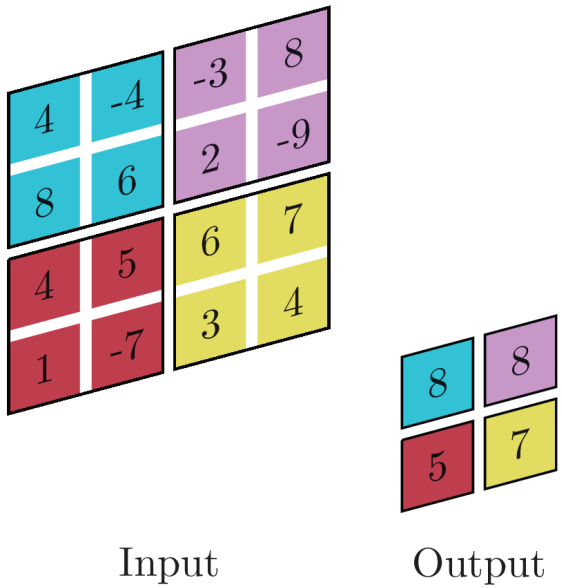
- **Max pooling**[7], for example, takes the maximum output within a rectangular neighbourhood.
- Pooling is almost always associated with downsampling,

[7] First proposed by Zhou and Chellappa, 1988

Source: BSc Thesis, Will Price, Univ of Bristol, May 2017

Your first CNN Layer

- **Max pooling**[7], for example, takes the maximum output within a rectangular neighbourhood.
- Pooling is almost always associated with downsampling,



[7] First proposed by Zhou and Chellappa, 1988

Source: BSc Thesis, Will Price, Univ of Bristol, May 2017

Your first CNN Layer

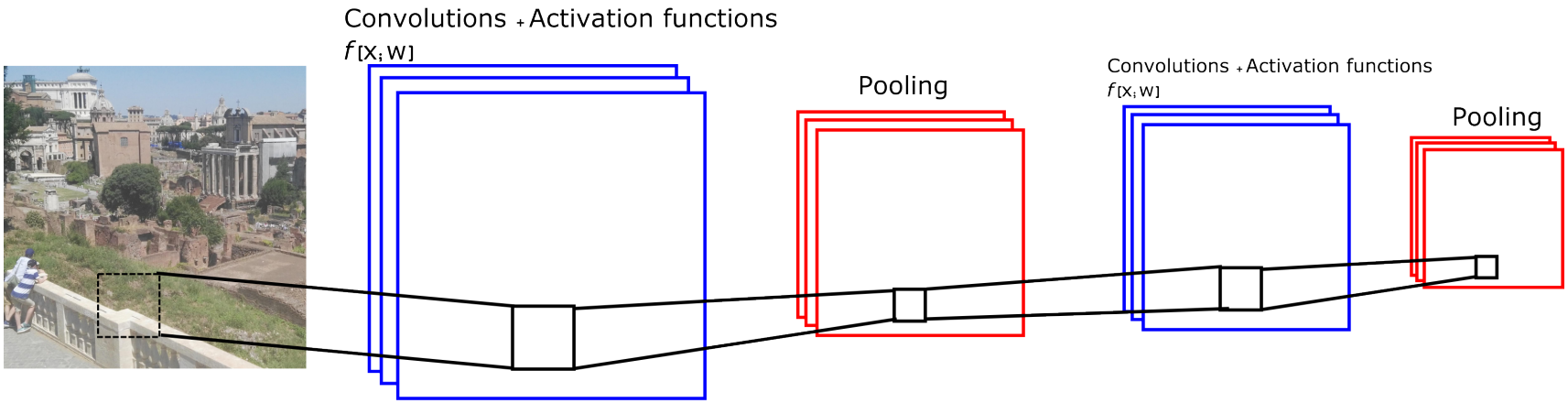
- Other pooling functions are:
  - average pooling
  - weighted average pooling
  - L^2 norm

Your first CNN Layer

- Other pooling functions are:
  - average pooling
  - weighted average pooling
  - L^2 norm
- Pooling allows invariance to small translations in input

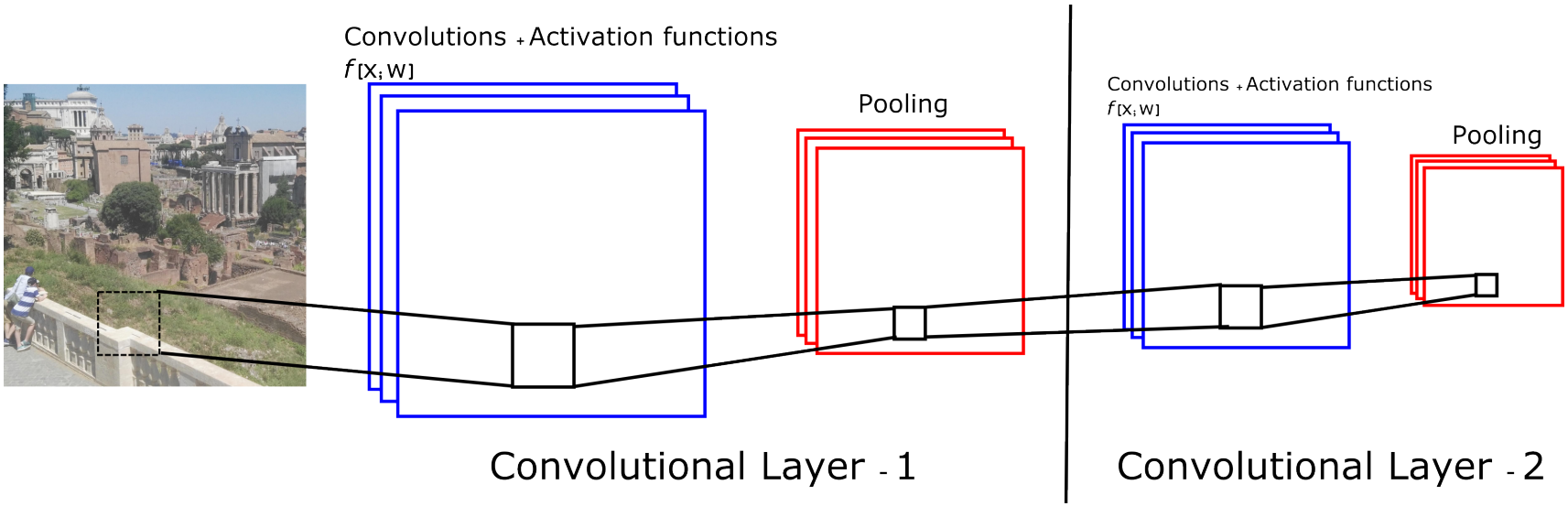
Your first CNN Architecture

- Further convolution $\rightarrow$ activation and $\rightarrow$ pooling with downsampling layers can be added



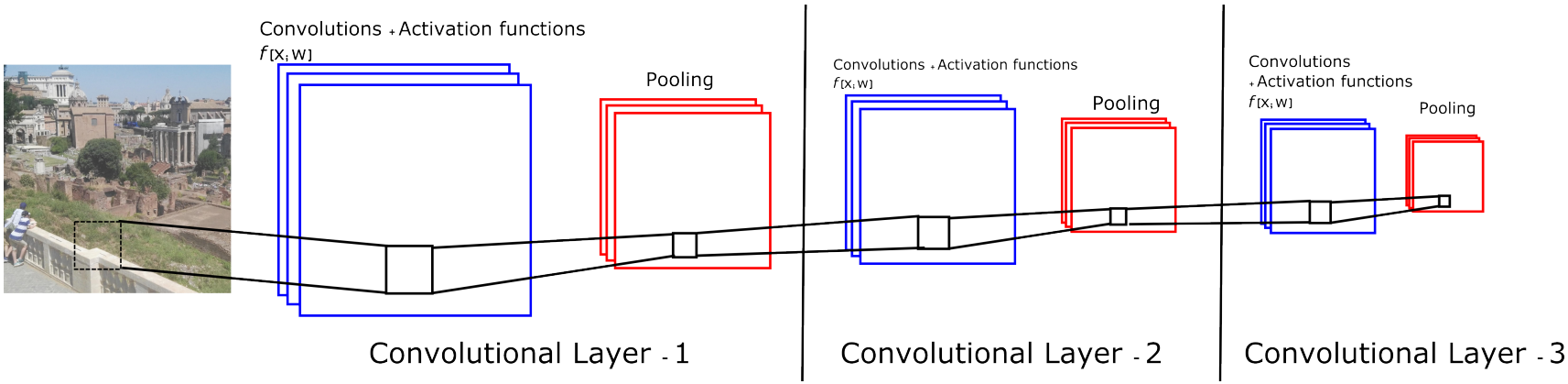
Your first CNN Architecture

- Technically, we would refer to these as the first and second convolutional layers of a deep CNN



Your first CNN Architecture

- As multiple convolutional layers as added, filters with larger receptive fields are learnt

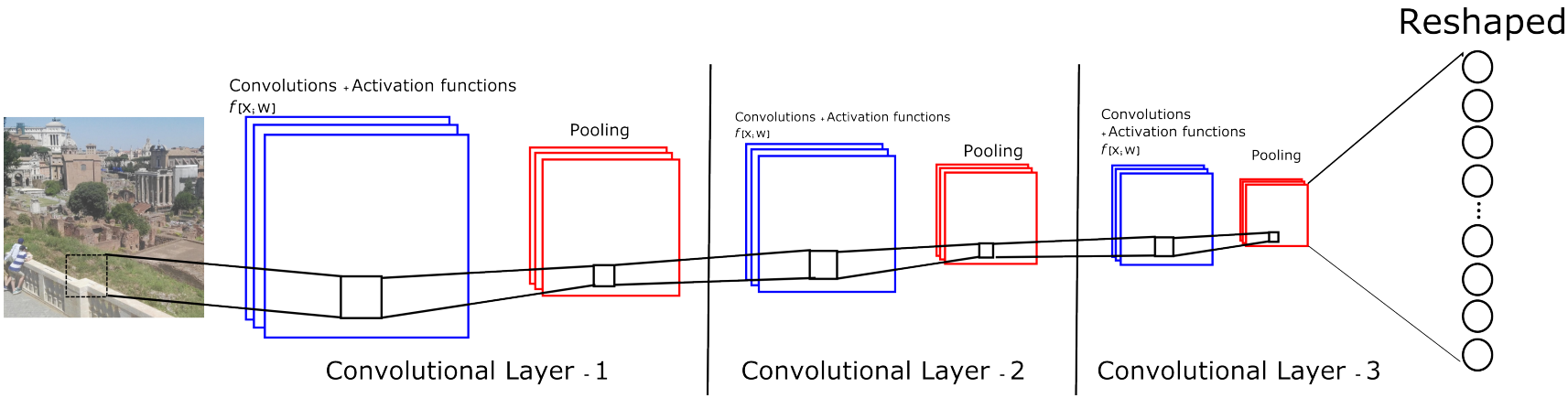


Your first CNN Architecture

- However, CNN architectures do not only have convolutions layers, they also have fully connected layers

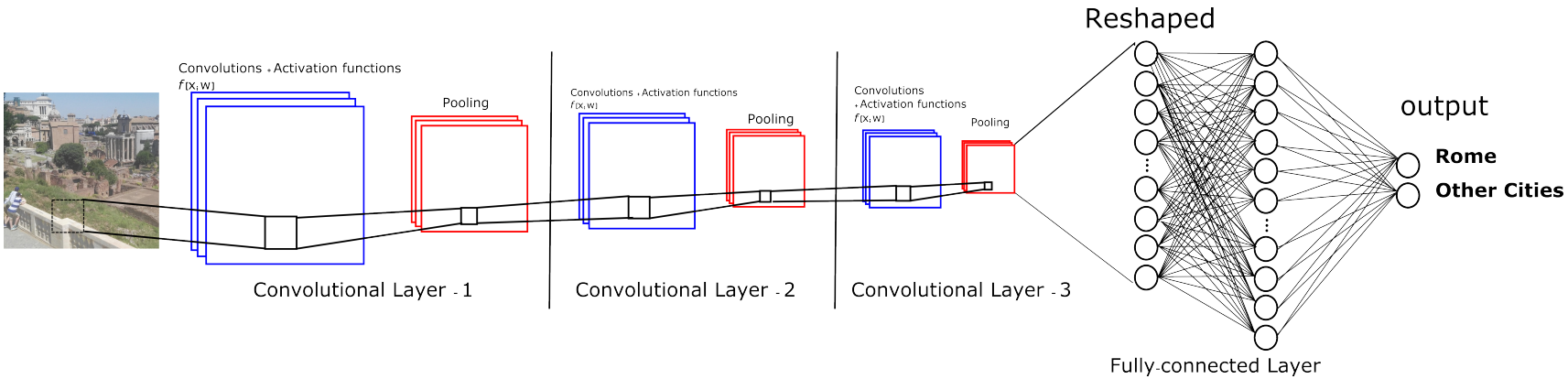
Your first CNN Architecture

- However, CNN architectures do not only have convolutions layers, they also have fully connected layers
- To use fully connecter layers, matrices are usually reshaped into 1-D



Your first CNN Architecture

- One or more fully connected layers can then be added



CNN Architecture Considerations

- In images for example, we have 3 channels (R/G/B)
- This means the input is 3D, and thus our convolutions are necessarily 3-D tensors

CNN Architecture Considerations

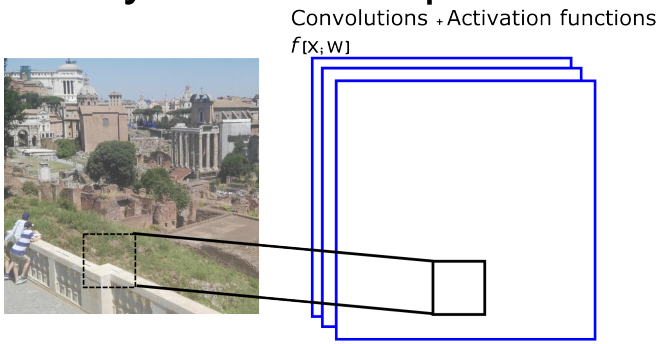
- In images for example, we have 3 channels (R/G/B)
- This means the input is 3D, and thus our convolutions are necessarily 3-D tensors
- Moreover, these are run in batch mode, so are typically 4-D tensors, with the fourth dimension indexing different examples in the batch

CNN Architecture Considerations

- In images for example, we have 3 channels (R/G/B)
- This means the input is 3D, and thus our convolutions are necessarily 3-D tensors
- Moreover, these are run in batch mode, so are typically 4-D tensors, with the fourth dimension indexing different examples in the batch
- All CNN representations omit the fourth dimension, for batch-based optimisation, for simplicity

CNN Architecture Considerations

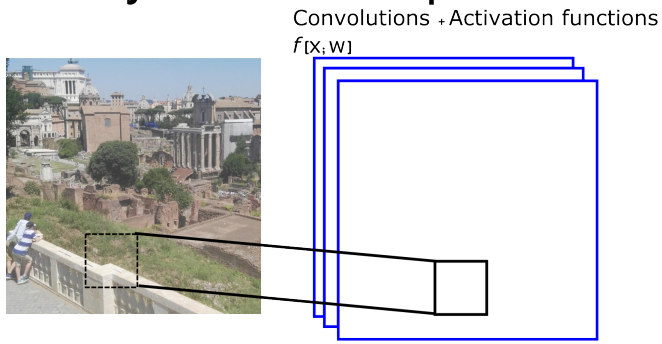
- In the previous example, the input and output sizes of the convolution+activation layers were equal



- However, practically we do not convolve densely, but instead we move the convolution skipping certain pixels.

CNN Architecture Considerations

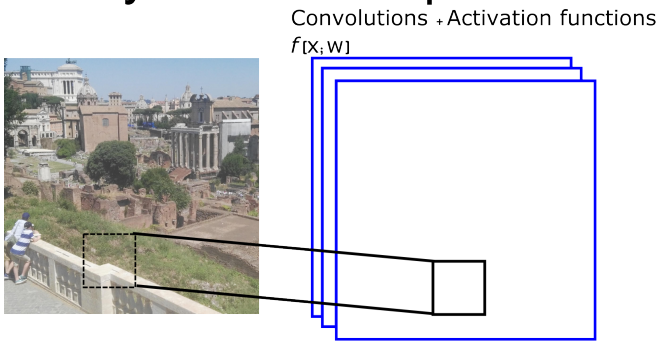
- In the previous example, the input and output sizes of the convolution+activation layers were equal



- However, practically we do not convolve densely, but instead we move the convolution skipping certain pixels.
- The number of pixels we skip, is referred to as the **stride** of the layer

CNN Architecture Considerations

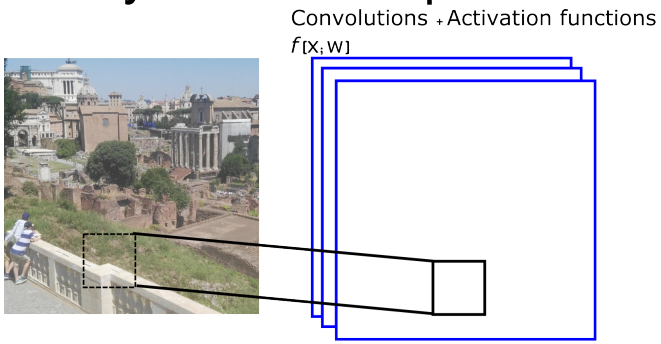
- In the previous example, the input and output sizes of the convolution+activation layers were equal



- However, practically we do not convolve densely, but instead we move the convolution skipping certain pixels.
- The number of pixels we skip, is referred to as the **stride** of the layer
- This results in downsampled convolutions

CNN Architecture Considerations

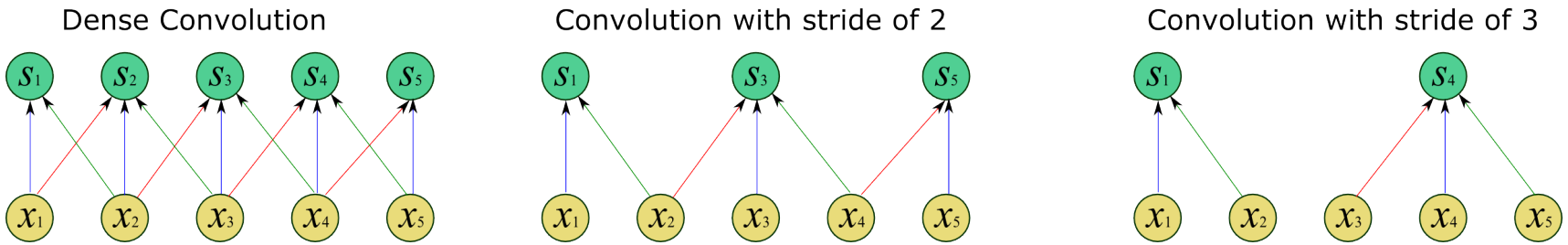
- In the previous example, the input and output sizes of the convolution+activation layers were equal



- However, practically we do not convolve densely, but instead we move the convolution skipping certain pixels.
- The number of pixels we skip, is referred to as the **stride** of the layer
- This results in downsampled convolutions
- It is possible to use separate strides for each dimension

CNN Architecture Considerations

- An example of coonvolution with a stride of two, and a stride of three, is shown below



CNN Architecture Considerations

- Care should be taken when backpropagating CNNs with zero padding or stride of more than one.

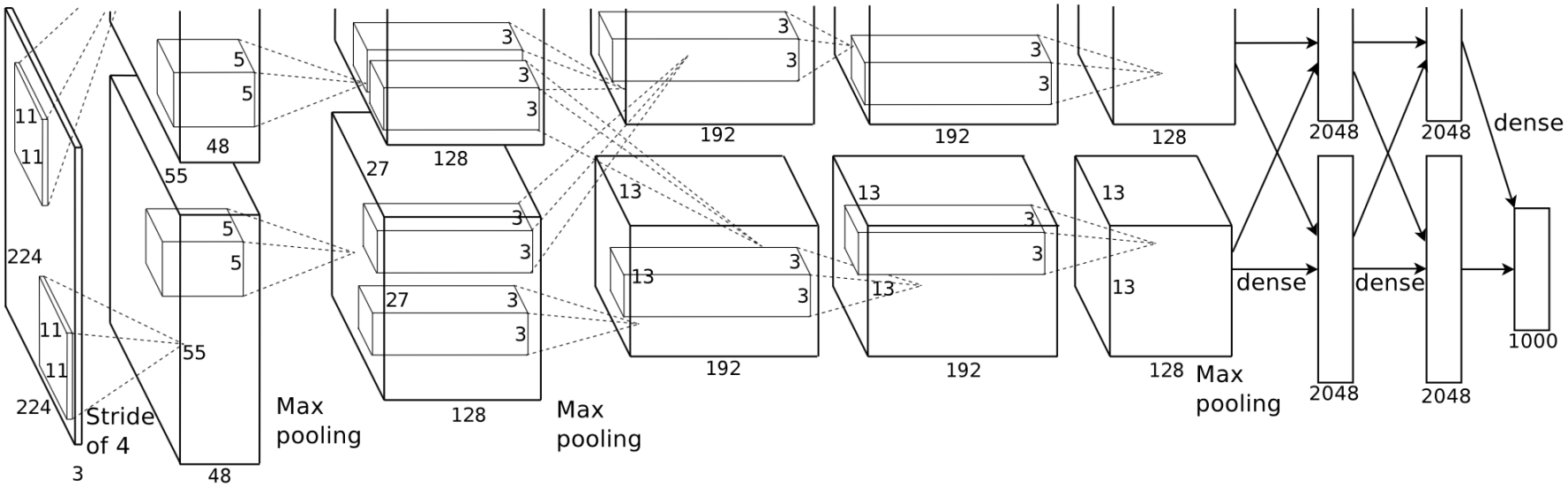
CNN Architectures - AlexNet

- When AlexNet won the most challenging computer vision task - Classifying 1000 classes by training from 10,000,000 images (The ImageNet Challenge), the new wave of CNN architectures started

Alex Krizhevsky, Sutskever and Hinton (2012) ImageNet Classification with Deep Convolutional Neural Networks, NIPS

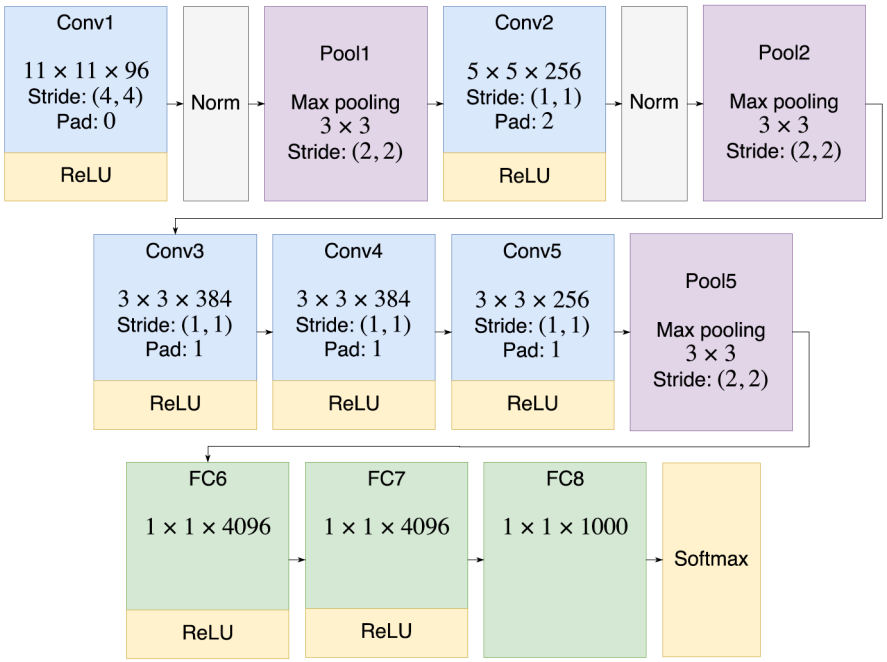
CNN Architectures - AlexNet

- When AlexNet won the most challenging computer vision task - Classifying 1000 classes by training from 10,000,000 images (The ImageNet Challenge), the new wave of CNN architectures started



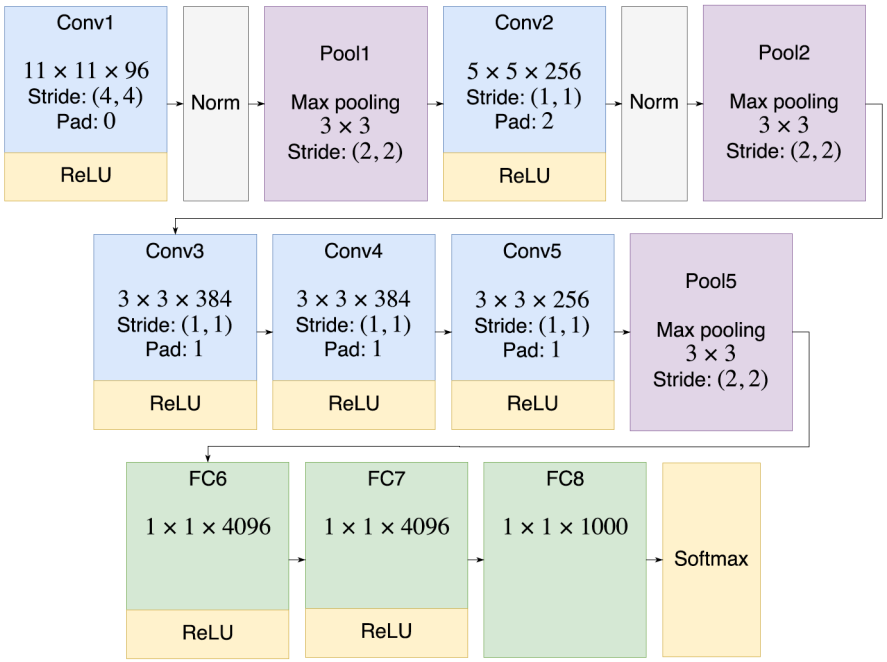
Alex Krizhevsky, Sutskever and Hinton (2012) ImageNet Classification with Deep Convolutional Neural Networks, NIPS

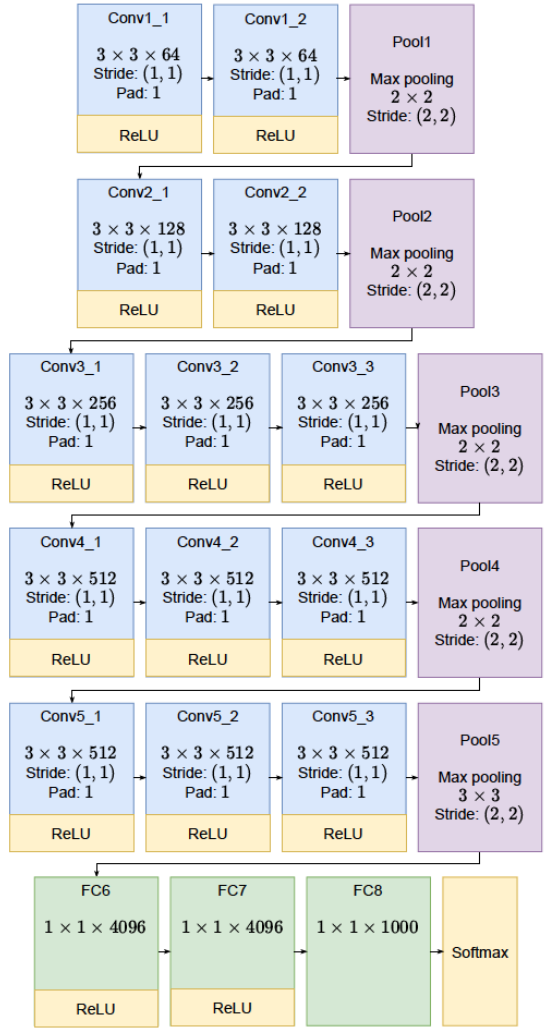
CNN Architectures - AlexNet vs VGG-16



Source: BSc Thesis, Will Price, Univ of Bristol, May 2017

CNN Architectures - AlexNet vs VGG-16





Source: BSc Thesis, Will Price, Univ of Bristol, May 2017

Training CNNs

- The most expensive part of CNN training is learning the features
- The fully-connected layers are usually relatively inexpensive to train because of the small number of features provided as input

Training CNNs

- The most expensive part of CNN training is learning the features
- The fully-connected layers are usually relatively inexpensive to train because of the small number of features provided as input
- When performing gradient descent, every gradient step requires a complete run of forward propagation and backward propagation through the entire network

Training CNNs

- The most expensive part of CNN training is learning the features
- The fully-connected layers are usually relatively inexpensive to train because of the small number of features provided as input
- When performing gradient descent, every gradient step requires a complete run of forward propagation and backward propagation through the entire network
- Several approaches have been proposed to solve this:

Training CNNs

- The most expensive part of CNN training is learning the features
- The fully-connected layers are usually relatively inexpensive to train because of the small number of features provided as input
- When performing gradient descent, every gradient step requires a complete run of forward propagation and backward propagation through the entire network
- Several approaches have been proposed to solve this:
  - Greedily training one layer at a time, freezing the others

Training CNNs

- The most expensive part of CNN training is learning the features
- The fully-connected layers are usually relatively inexpensive to train because of the small number of features provided as input
- When performing gradient descent, every gradient step requires a complete run of forward propagation and backward propagation through the entire network
- Several approaches have been proposed to solve this:
  - Greedily training one layer at a time, freezing the others
  - Use pre-trained features, only training the last convolutional layer with the fully-connected layers

Training CNNs

- The most expensive part of CNN training is learning the features
- The fully-connected layers are usually relatively inexpensive to train because of the small number of features provided as input
- When performing gradient descent, every gradient step requires a complete run of forward propagation and backward propagation through the entire network
- Several approaches have been proposed to solve this:
  - Greedily training one layer at a time, freezing the others
  - Use pre-trained features, only training the last convolutional layer with the fully-connected layers
  - Use random features, only training the fully-connected layers

Training CNNs

- The most expensive part of CNN training is learning the features
- The fully-connected layers are usually relatively inexpensive to train because of the small number of features provided as input
- When performing gradient descent, every gradient step requires a complete run of forward propagation and backward propagation through the entire network
- Several approaches have been proposed to solve this:
  - Greedily training one layer at a time, freezing the others
  - Use pre-trained features, only training the last convolutional layer with the fully-connected layers
  - Use random features, only training the fully-connected layers
  - Selected hand-crafted features (not recommended)

Training CNNs

- The most expensive part of CNN training is learning the features
- The fully-connected layers are usually relatively inexpensive to train because of the small number of features provided as input
- When performing gradient descent, every gradient step requires a complete run of forward propagation and backward propagation through the entire network
- Several approaches have been proposed to solve this:
  - Greedily training one layer at a time, freezing the others
  - Use pre-trained features, only training the last convolutional layer with the fully-connected layers
  - Use random features, only training the fully-connected layers
  - Selected hand-crafted features (not recommended)
  - Apply k-means clustering to image patches and use the cluster centres for convolutions (Coates et al 2011)

Training CNNs

- The most expensive part of CNN training is learning the features
- The fully-connected layers are usually relatively inexpensive to train because of the small number of features provided as input
- When performing gradient descent, every gradient step requires a complete run of forward propagation and backward propagation through the entire network
- Several approaches have been proposed to solve this:
  - Greedily training one layer at a time, freezing the others
  - Use pre-trained features, only training the last convolutional layer with the fully-connected layers
  - Use random features, only training the fully-connected layers
  - Selected hand-crafted features (not recommended)
  - Apply k-means clustering to image patches and use the cluster centres for convolutions (Coates et al 2011)
- All these approaches were popular before mega-size datasets were introduced, and remain relevant for problems with small data sizes

CNN Architectures - Further architectures

- See live demos at: `http://cs231n.stanford.edu`
- Visualise recent architectures at: `http://josephpcohen.com/w/visualizing-cnn-architectures-side-by-side-with-mxnet/`

Further Reading

- Deep Learning

  Ian Goodfellow, Yoshua Bengio, and Aaron Courville
  MIT Press, ISBN: 9780262035613.
  - Chapter 9 – Convolutional Networks