COMSM0045: Convolutional Neural Networks (Part 1)

Dima Damen
Dima.Damen@bristol.ac.uk

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

October 10, 2020

Introduction

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

[1] Arguably!

Dima Damen
Dima.Damen@bristol.ac.uk

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**

When to use CNNs?

- CNNs expect the **input** data x has a known **grid-like topology**
- 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**
- 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**
- 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

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**

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

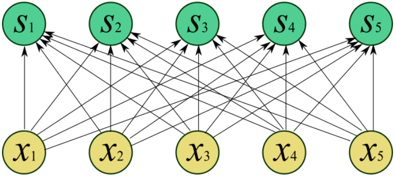
CNN Properties: 1- Sparse Interactions[2]

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

[2] Also referred to as multi-scale interactions

CNN Properties: 1- Sparse Interactions[2]

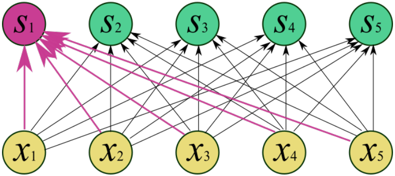
- A major difference between fully connected neural networks and CNNs are the contributions of input units to output units.
- Consider this two-layer fully-connected network, with 5 input units,



[2] Also referred to as multi-scale interactions

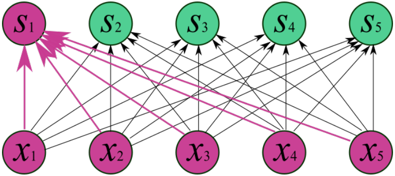
CNN Properties: 1- Sparse Interactions

- 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,



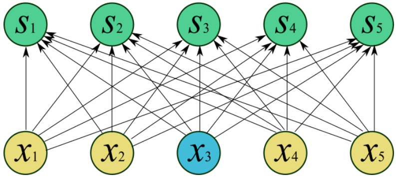
CNN Properties: 1- Sparse Interactions

- 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 $s_1 = f(x_1, x_2, x_3, x_4, x_5; \omega_1, \omega_2, \omega_3, \omega_4, \omega_5)$.



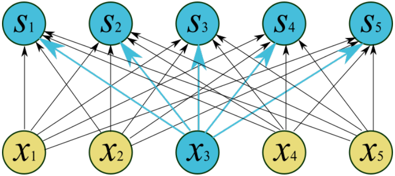
CNN Properties: 1- Sparse Interactions

- 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

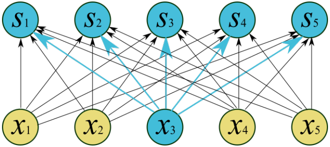


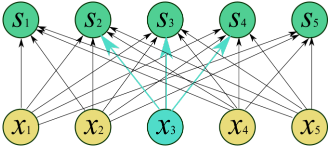
CNN Properties: 1- Sparse Interactions

- 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

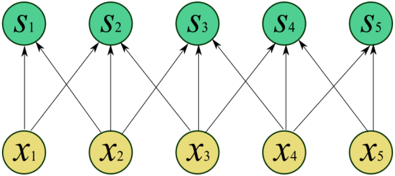
- In **CNNs**, due to the grid structure, it is sufficient to limit the number of connections from each input unit unit to k,
- See the connections from x_3





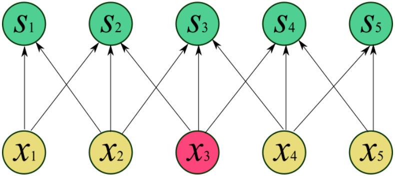
CNN Properties: 1- Sparse Interactions

- 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



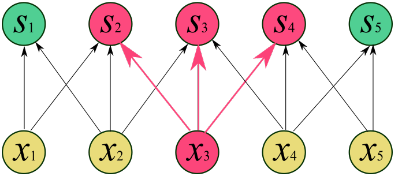
CNN Properties: 1- Sparse Interactions

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



CNN Properties: 1- Sparse Interactions

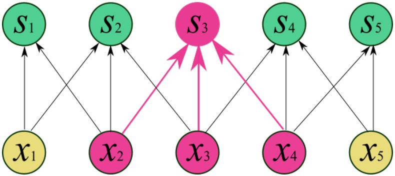
- In **CNNs**, one input unit x_3, affects a limited number of output units



Dima Damen
Dima.Damen@bristol.ac.uk

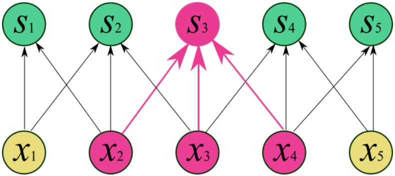
CNN Properties: 1- Sparse Interactions

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



CNN Properties: 1- Sparse Interactions

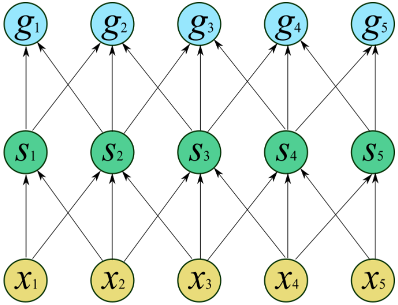
- The input units affecting a certain output unit (e.g. s_3), are known as the unit's **receptive field**.



Dima Damen
Dima.Damen@bristol.ac.uk

CNN Properties: 1- Sparse Interactions

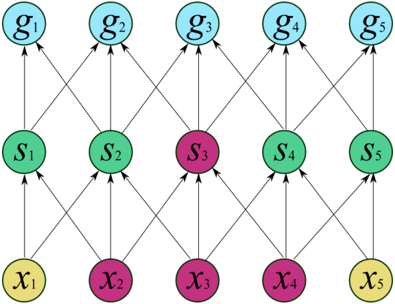
- Interestingly, as more layers are added,



Dima Damen
Dima.Damen@bristol.ac.uk

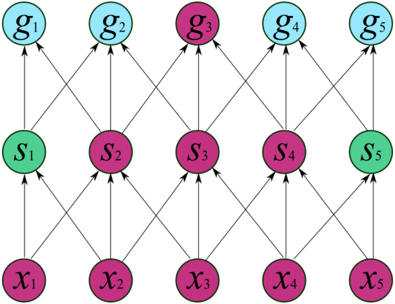
CNN Properties: 1- Sparse Interactions

- Interestingly, as more layers are added,



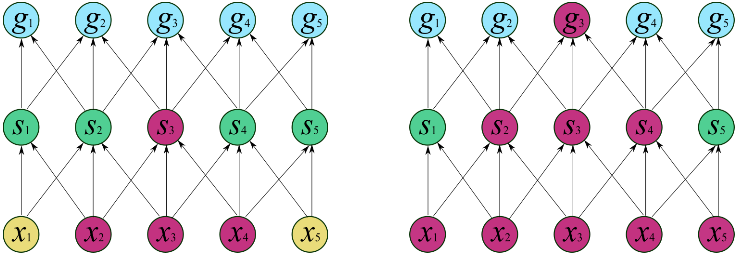
CNN Properties: 1- Sparse Interactions

- 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**



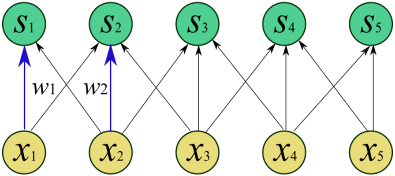
CNN Properties: 2- Parameter Sharing

CNN Properties: 2- Parameter Sharing

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

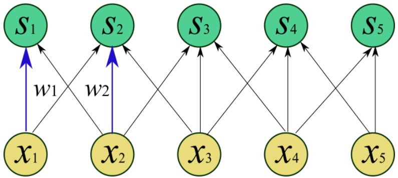
CNN Properties: 2- Parameter Sharing

- **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

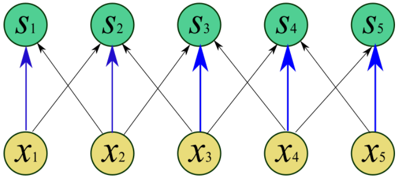
- **Paramter sharing** refers to using the same parameter for more than one function in the network
- You can consider this as tying two paramters w_1 and w_2 together, so they can only have the same value
- You have dropped the number of parameters you need to train by 1 (!)



Dima Damen
Dima.Damen@bristol.ac.uk

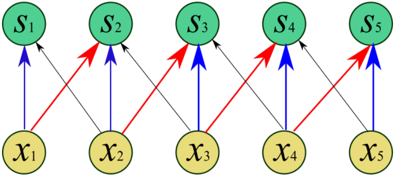
CNN Properties: 2- Parameter Sharing

- You can similarly think about sharing more parameters



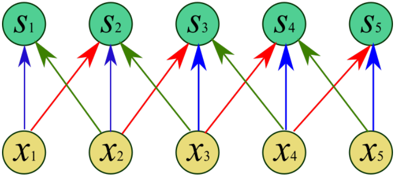
CNN Properties: 2- Parameter Sharing

- You can similarly think about sharing more parameters



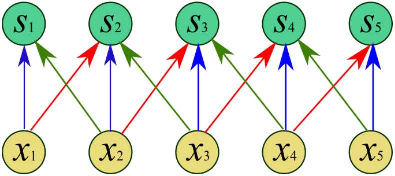
CNN Properties: 2- Parameter Sharing

- You can similarly think about sharing more parameters



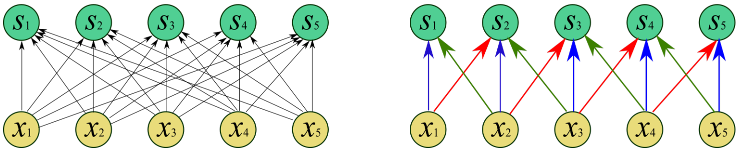
CNN Properties: 2- Parameter Sharing

- 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

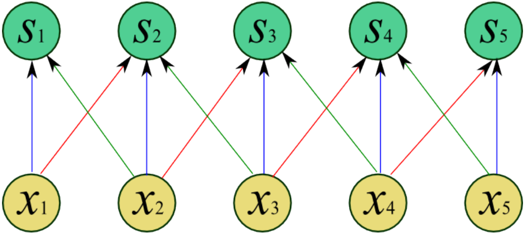
- Parameter sharing is also known as **tied weights**, because the weight applied to one input is **tied** to the weight applied elsewhere.
- Does not affect the runtime of the forward pass
- Does significantly reduce the memory requirements for the model
- You have significantly less parameters to train,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

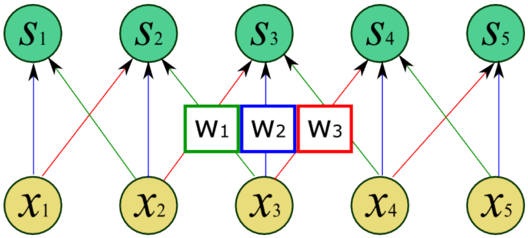
CNN Properties: 2- Parameter Sharing

- It this new??



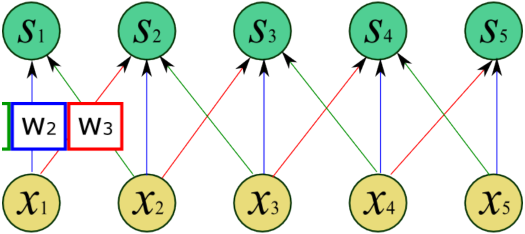
CNN Properties: 2- Parameter Sharing

- It this new??



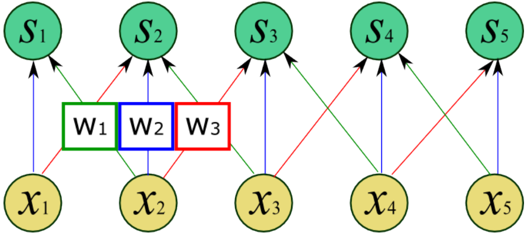
CNN Properties: 2- Parameter Sharing

- It this new??



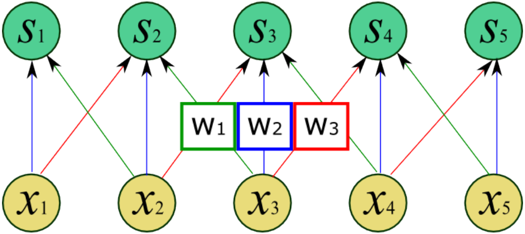
CNN Properties: 2- Parameter Sharing

- It this new??



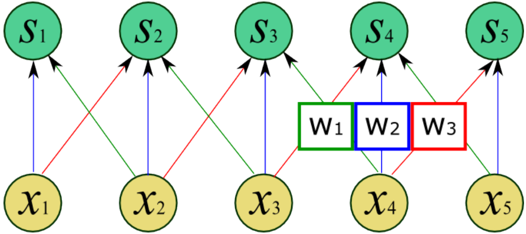
CNN Properties: 2- Parameter Sharing

- It this new??



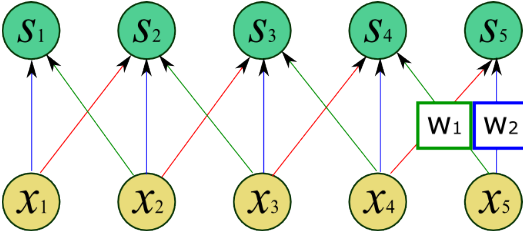
CNN Properties: 2- Parameter Sharing

- It this new??



CNN Properties: 2- Parameter Sharing

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



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*

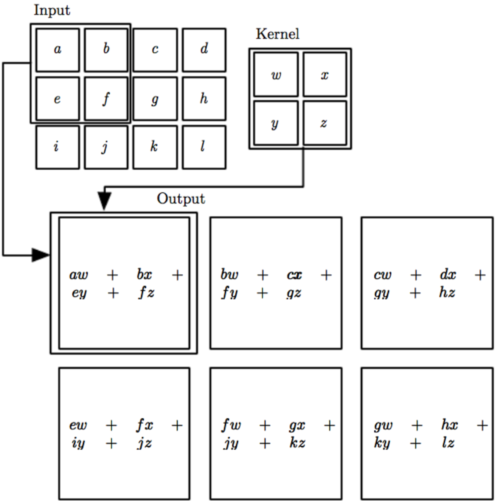
Convolution vs Correlation

- However, *most* DNN libraries implement the convolution as a **cross-correlation** operation, without flipping the kernel[3]

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

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

Convolution vs Correlation

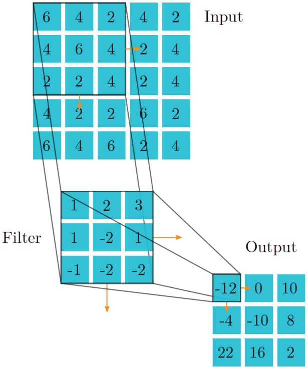


[4]

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

CNN Properties: 2- Parameter Sharing

- And in 2-D



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

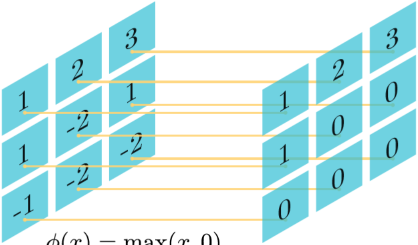
Your first CNN Layer

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

Source: Rob Fergus, NN, MLSS2015 Summer School Presentation

Your first CNN Layer

- 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

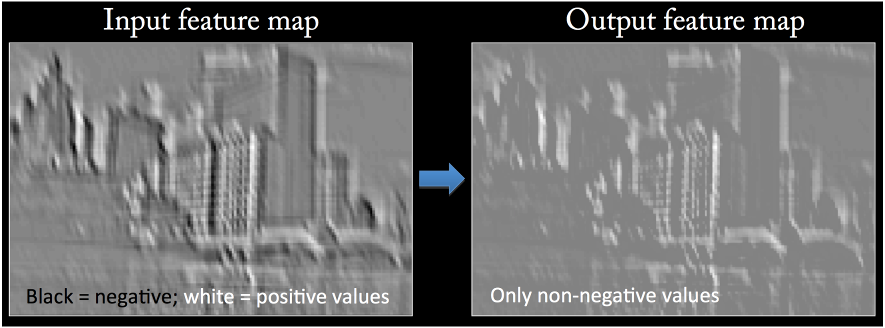


$\phi(x) = \max(x, 0)$ [5]

[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

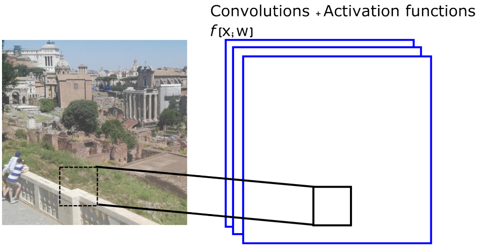


[6]

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

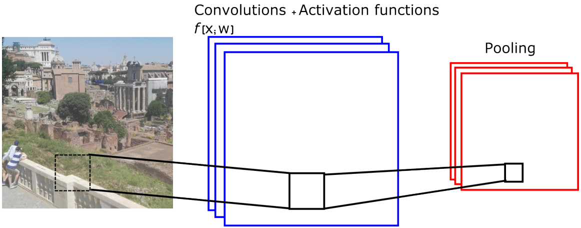
Your first CNN Layer

- 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

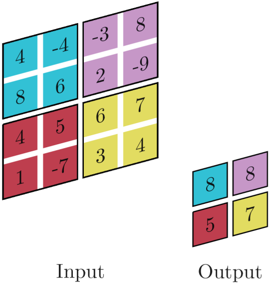
- **Max pooling**[7], for example, takes the maximum output within a rectangular neighbourhood.
- Pooling is almost always associated with downsampling,

[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

Further Reading

- Deep Learning

  Ian Goodfellow, Yoshua Bengio, and Aaron Courville
  MIT Press, ISBN: 9780262035613.

  - Chapter 9 – Convolutional Networks