Lecture 6: Convolutional Neural Networks

Ali Harakeh

University of Waterloo

WAVE Lab
ali.harakeh@uwaterloo.ca

July 11, 2017

Overview

Section 1

Introduction

Introduction

- Convolutional Neural Networks (ConvNets) are a specialized kind of neural networks for processing data that has a **known grid like topology**.
- Example of such data can be 1-D time series data sampled at regular intervals, or 2-D images.
- As the name suggests, these networks employ the mathematical **convolution** operator.
- **Convolutions** are a special kind of **linear** operators that an be used instead of general matrix multiplication.

Section 2

The Convolution Operator

Mathematical Definition

- The convolution operator is mathematically defined as:

$$s(t) = (x * w)(t) = \int x(a)w(t-a)da$$
$$= \sum_{a=-\infty}^{\infty} x(a)w(t-a)$$

- Note that the infinite summation can be implemented as a finite one as it is assumed that these functions are zero everywhere except at t where a measurement is provided.

Terminology

- I is usually a multidimensional array of data termed the **input**.
- K is usually a multidimensional array of parameters termed the **kernel** or the **filter**.
- S is the **output** or **feature map**.

Mathematical Definition : 2D Case

- The convolution operator is mathematically defined as:

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

- Usually in convolutions, we **flip** the kernel, which gives rise to the above commutative property.
- The commutative property above is useful for writing proofs. It is not so useful for neural networks. Why?

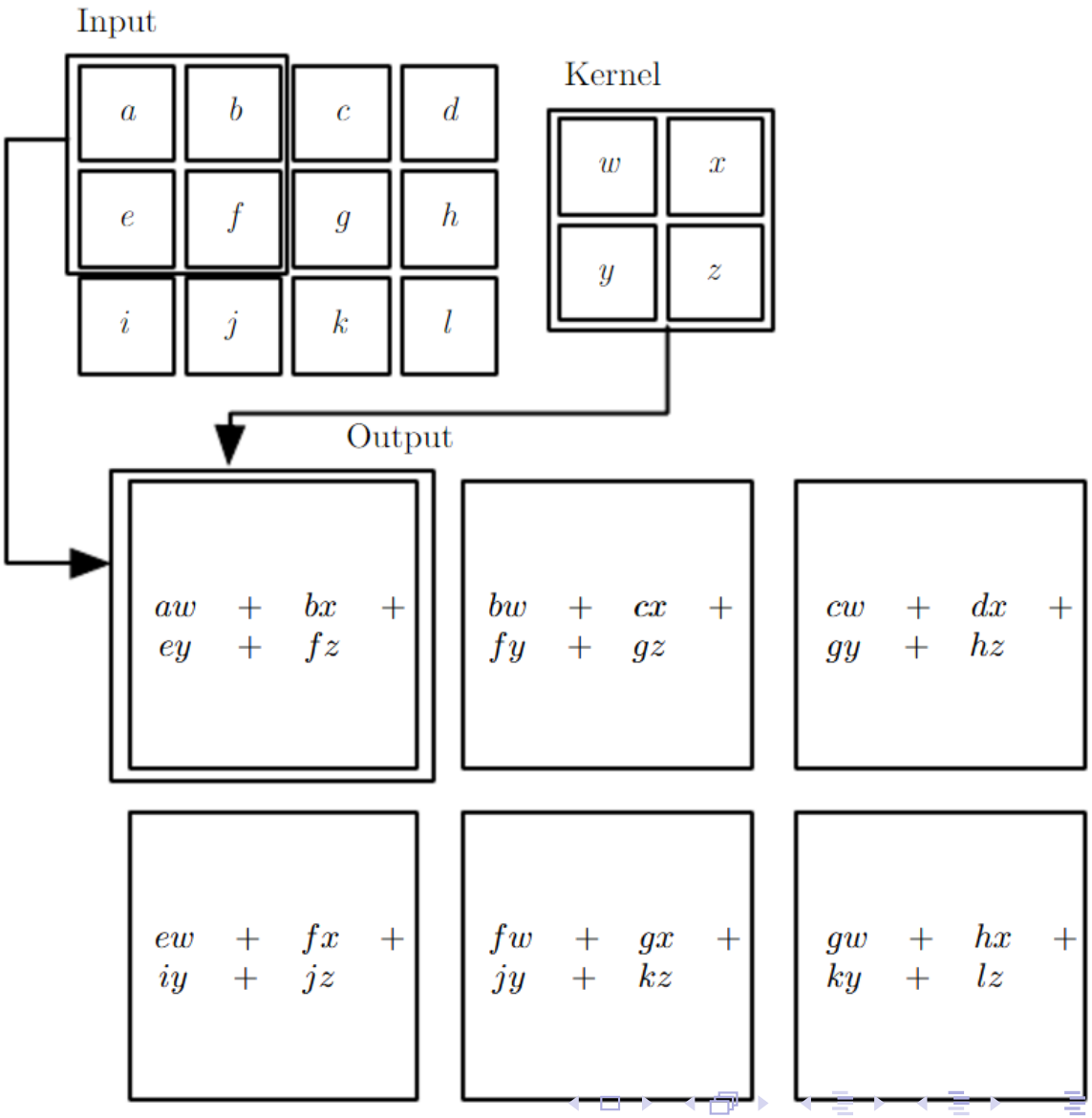
Mathematical Definition : 2D Case

- Most machine learning libraries implement **cross-correlation** while calling it convolutions.
- Cross correlation is defined as:

$$S(i,g) = (I * K)(i,j) = \sum_m \sum_n I(m,n)K(i+m, j+n)$$
$$= \sum_m \sum_n I(i-m, j-n)K(m,n)$$

- The logic behind the above is that usually, we **learn** the kernel and thus it does not matter if it is flipped or not.
- One second reason is that we usually employ convolutions with functions that do not commute regardless of the convolution flipping the kernel or not.

Example


Input
a
b
c
d
e
f
g
h
i
j
k
l
Kernel
w
x
y
z
Output
aw + bx +
ey + fz
bw + cx +
fy + gz
cw + dx +
gy + hz
ew + fx +
iy + jz
fw + gx +
jy + kz
gw + hx +
ky + lz

Section 3

Motivation: Why Use Convolutions

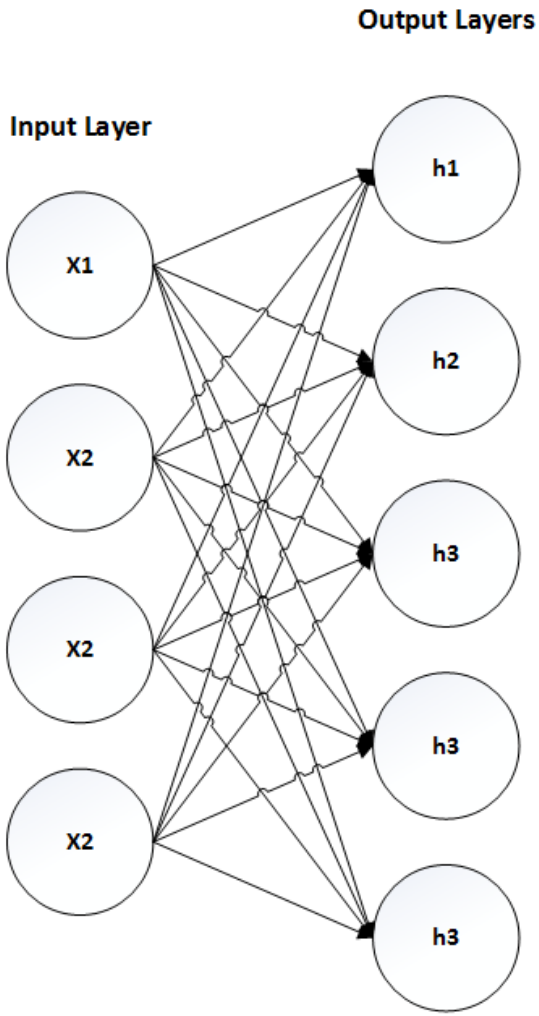
Advantages For ML Systems

- Convolutions leverage three important ideas that can help improve a machine learning system:
  - **Sparse Interactions**
  - **Parameter Sharing**
  - **Equivariant Representation**
- Moreover, convolutions provide a way to handle input of different sizes.

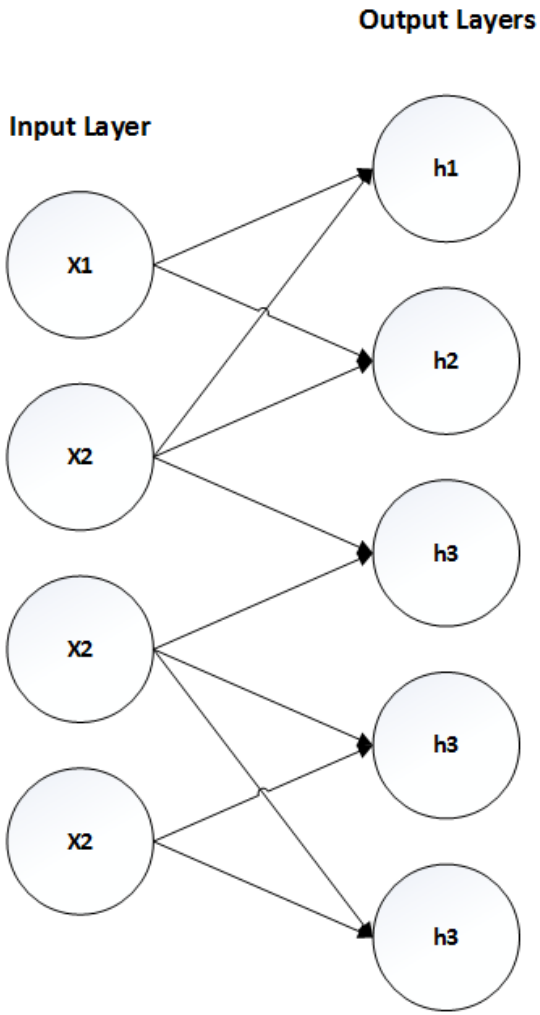
Sparse Connectivity

- Traditional Neural Networks have a parameter to model the interaction of every element of the input vector with every element of the output vector.
- Convolutional Networks typically have **sparse connectivity**.
- Sparse connectivity is achieved by making the kernel smaller than the input.

Sparse Connectivity



Fully Connected Layer



Convolutional Layer

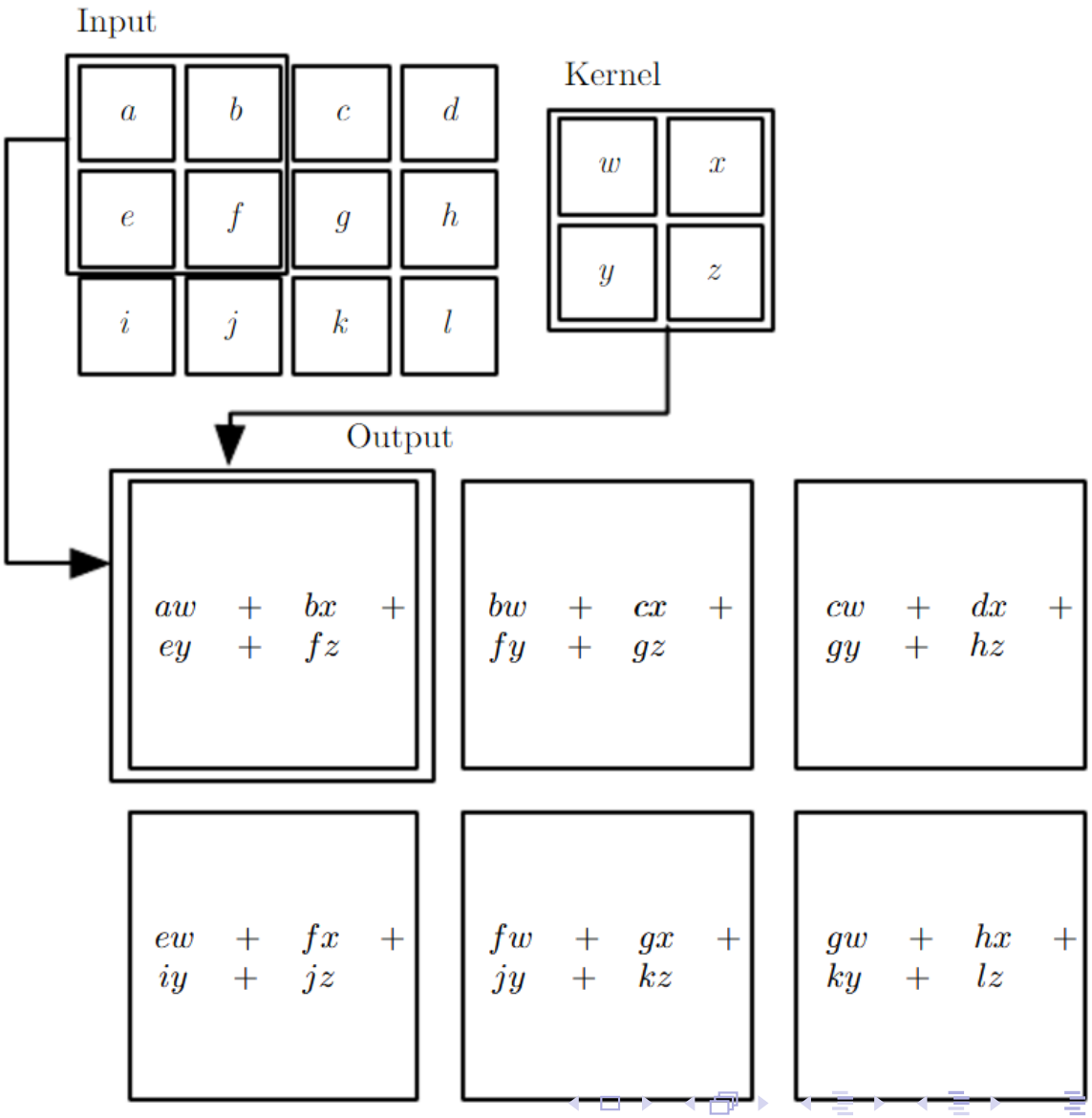
Sparse Connectivity

- If there are m inputs and n outputs, the fully connected layer will have $m \times n$ parameters and the matrix multiplication algorithm used in practice will have $O(m \times n)$ runtime per example.
- In the case of a convolutional layer, this bound is reduced to $O(k \times n)$, where $k < m$ is the number of sparse connections.
- In a deep convolutional network,units in the deeper layers may indirectly interact with a larger portion of the input.
- This allows the network to efficiently describe complicated interactions between many variables by constructing such interactions from simple building blocks that each describe only sparse interactions.

Parameter Sharing

- **Parameter sharing** refers to using the same parameter for more than one function in a model.
- Convolutional layers heavily apply this concept through applying the same kernel to every position of the input.
- This does not affect the runtime at inference, but it affects the memory requirement per layer as we now have to store only k parameters.
- Convolution is dramatically more efficient than dense matrix multiplication in terms of memory requirements and statistical efficiency.

Parameter Sharing



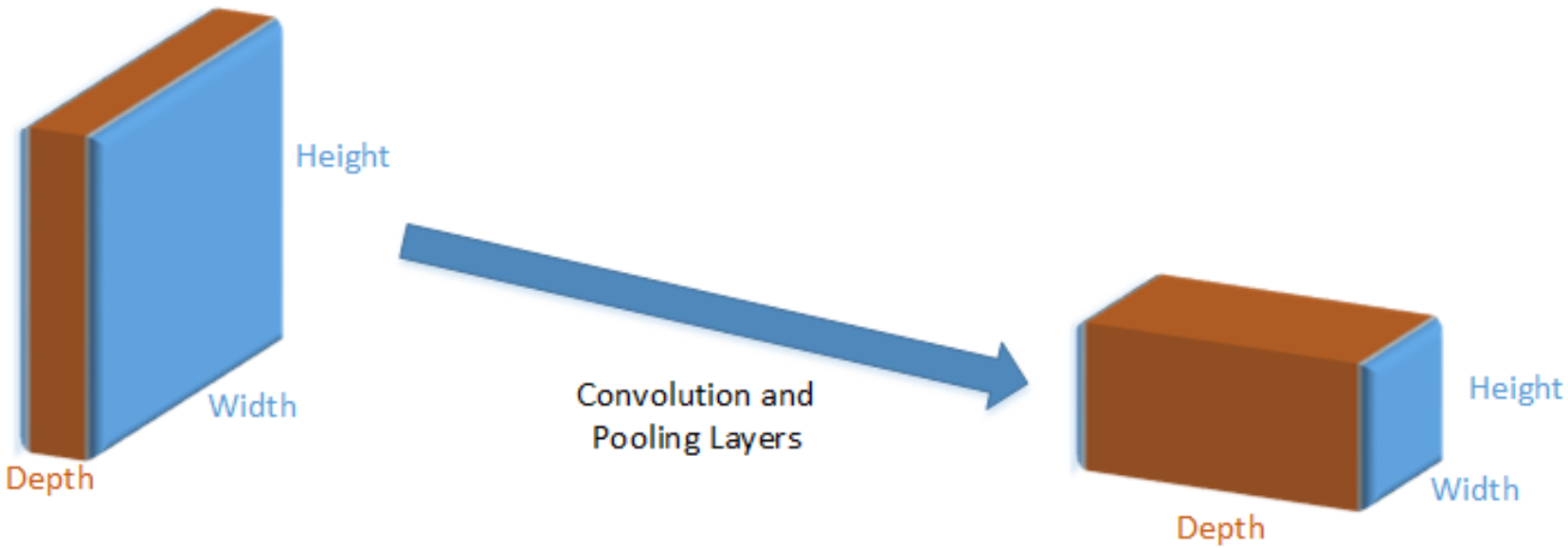
Equivariance

- A function $f(x)$ is **equivariant** to a function $g(x)$ if $f(g(x)) = g(f(x))$.
- Convolution operators are equivariant to translation of the input.
- Convolution is not naturally equivariant to some other transformations, such as changes in the scale or rotation of an image. Other mechanisms are necessary for handling these kinds of transformations.

Section 4

Convolutions In Pract1ice

Output Volume Size



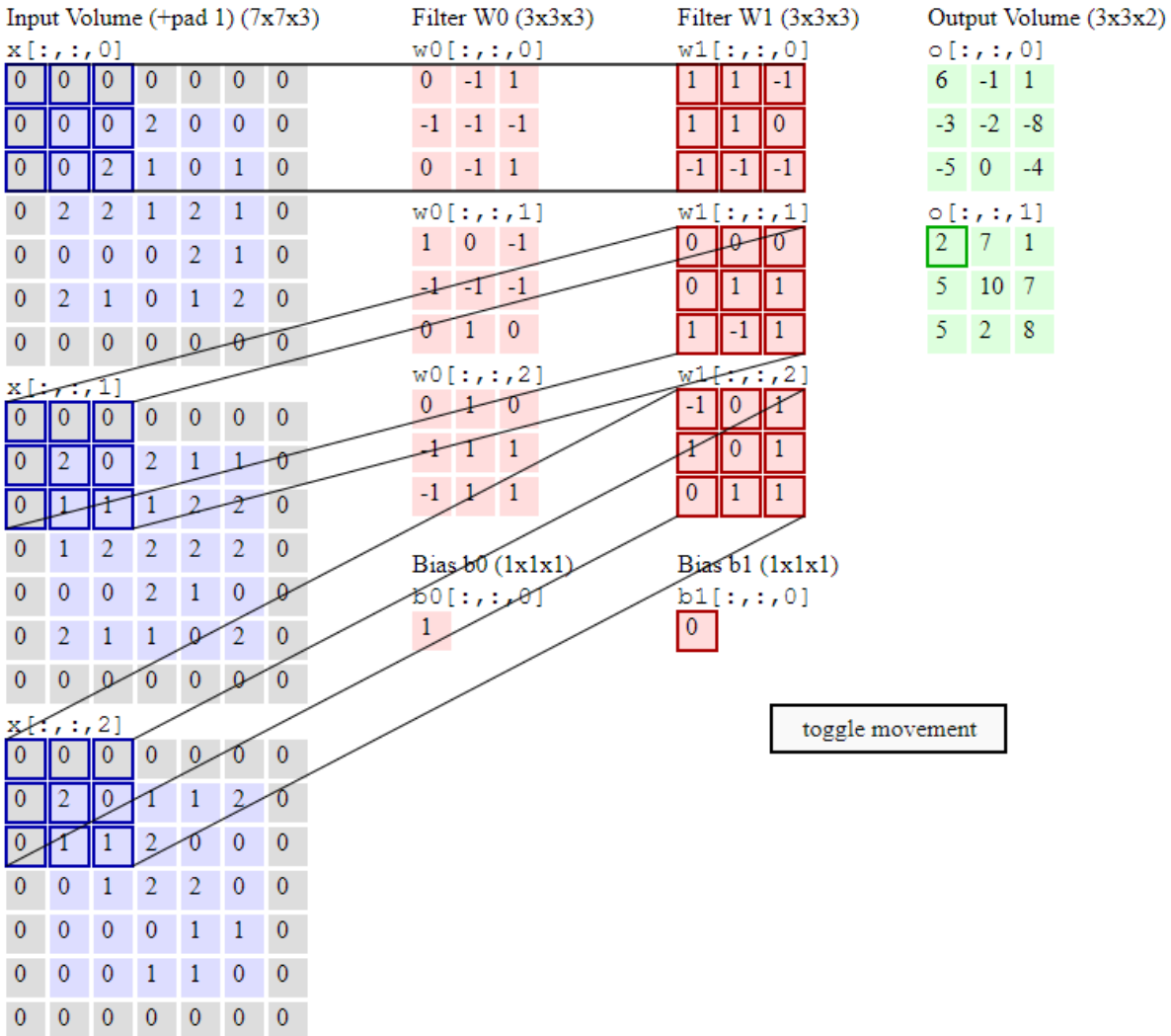
Output Volume Size

- In practice, three hyperparameters control the size of the output of a convolutional layer.
- The output **depth** of the volume is a hyper parameter

Number Of Parameters In a Convolutional Layer

- In practice, three hyperparameters control the size of the output of a convolutional layer.
- The output **depth** of the volume is a hyperparameter and corresponds to the number of filters we would like to use, each learning to look for something different in the input.
- The output width and height are controlled by the **stride** we use to slide each filter and the **padding** we use to expand the input.

Output Volume Size



Computing The Size Of The Output Volume

- Assuming that the filters are $m \times m$ and we have K filters, we can compute the size of our output volume according to the following:

$$W_{out} = \frac{W_{in} - m + 2Padding}{Stride + 1}$$
$$H_{out} = \frac{H_{in} - m + 2Padding}{Stride + 1}$$
$$D_{out} = K$$

Computing The Size Of The Output Volume

- Assuming that the filters are $m \times m$ and we have K filters, we can compute the size of our output volume according to the following:

$$W_{out} = \frac{W_{in} - m + 2 \times Padding}{Stride + 1}$$
$$H_{out} = \frac{H_{in} - m + 2 \times Padding}{Stride + 1}$$
$$D_{out} = K$$

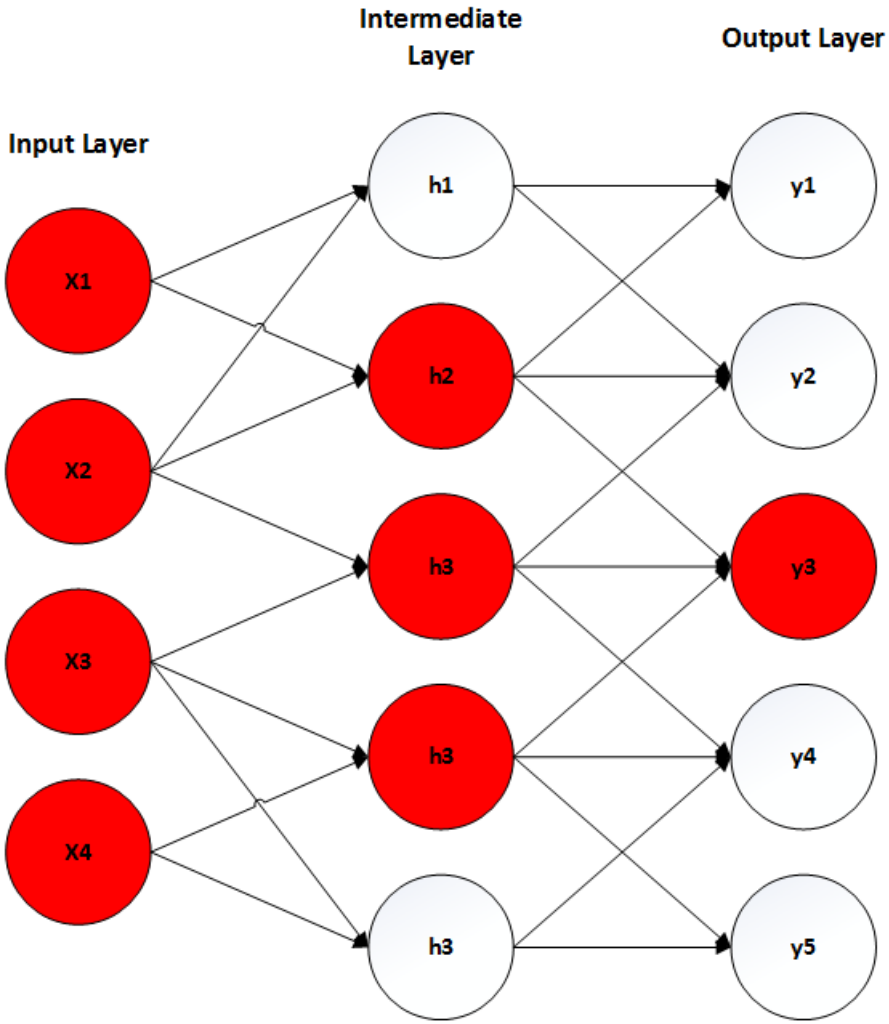
Number Of Parameters In Convolutional Layers

- The number of parameters in a convolutional layer can be computed according to the following:

$$Parameters = m^2 \times k \times D_{in} + k$$

- This is due to the layer having $m^2 \times D_{in}$ weights for k filters and k biases.

Receptive Field



Additional Notes

- The backward pass for a convolution operation (for both the data and the weights) is also a convolution (but with spatially-flipped filters). Libraries do that for you so no need to worry about it.
- 1×1 convolutions are used to manipulate the depth of the output volume. They can be thought of as a dot product through a depth slice.
- Many variations of convolution exist such as **Dilated Convolutions** and **Deformable Convolutions**.

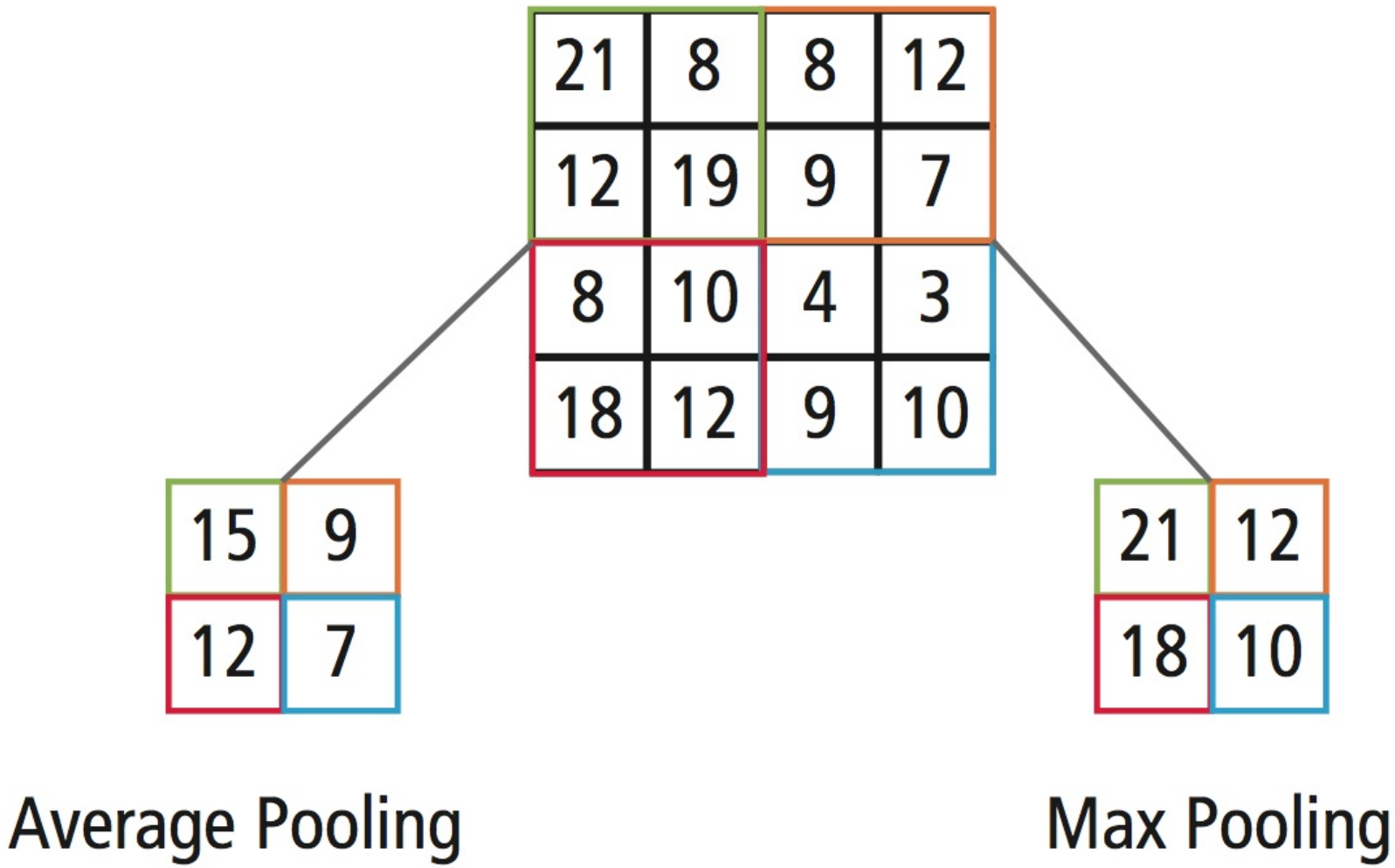
Section 5

Pooling Layers

Pooling

- A pooling function replaces the output of the previous layer, with a summary statistic of the nearby outputs.
- Pooling helps to make representations become **approximately invariant** to small translation of the input. If we translate the input a small amount, the output of the pooling layer will not change.
- The use of pooling can be viewed as adding an infinitely strong prior that the function the layer learns must be invariant to small translations. When this assumption is correct, it can greatly improve the statistical efficiency of the network.

Max Pooling vs Average Pooling



Geoffrey Hinton On Pooling

- The pooling operation used in convolutional neural networks is a big mistake and the fact that it works so well is a disaster.
- If the pools do not overlap, pooling loses valuable information about where things are. We need this information to detect precise relationships between the parts of an object.
- He proposes "capsules" (subnetworks in networks) as an alternative to pooling.

Additional Notes

- Many people dislike the pooling operation and think that we can get away without it.
- For example, *Striving for Simplicity: The All Convolutional Net* proposes to discard the pooling layer in favour of architecture that only consists of repeated convolutional layers.
- Just use a larger stride every once in a while to shrink down the input size.
- Discarding pooling layers has also been found to be important in training good generative models, such as variational autoencoders (VAEs) or generative adversarial networks (GANs).
- It seems likely that future architectures will feature very few to no pooling layers.

Section 6

Region Of Interest (ROI) Pooling

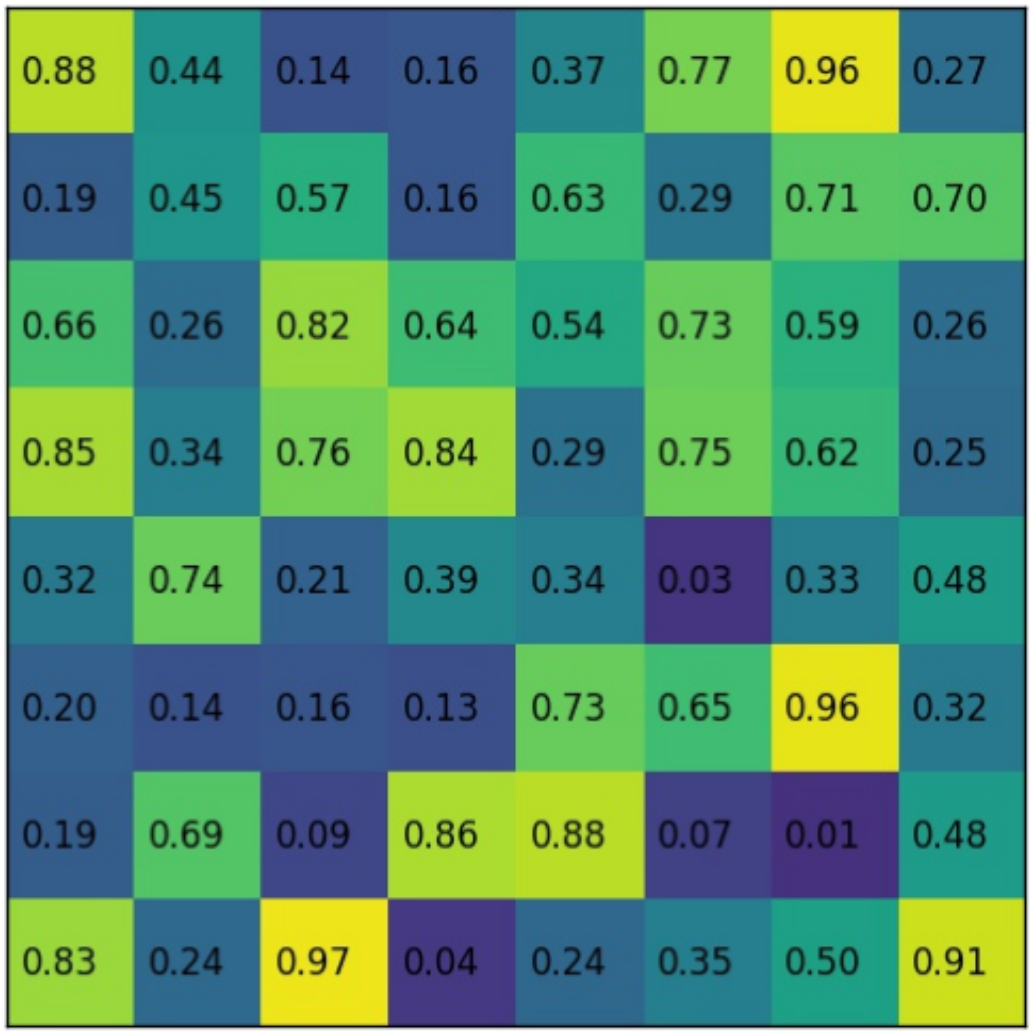
ROI Pooling

- **Region of interest pooling** (also known as RoI pooling) is an operation widely used in object detection tasks using convolutional neural networks.
- The operation was proposed in *Fast RCNN* paper in 2015.
- Its purpose is to perform max pooling on inputs of non-uniform sizes to obtain fixed-size feature maps.
- This enables training architectures containing RPNs in an end-to-end fashion.

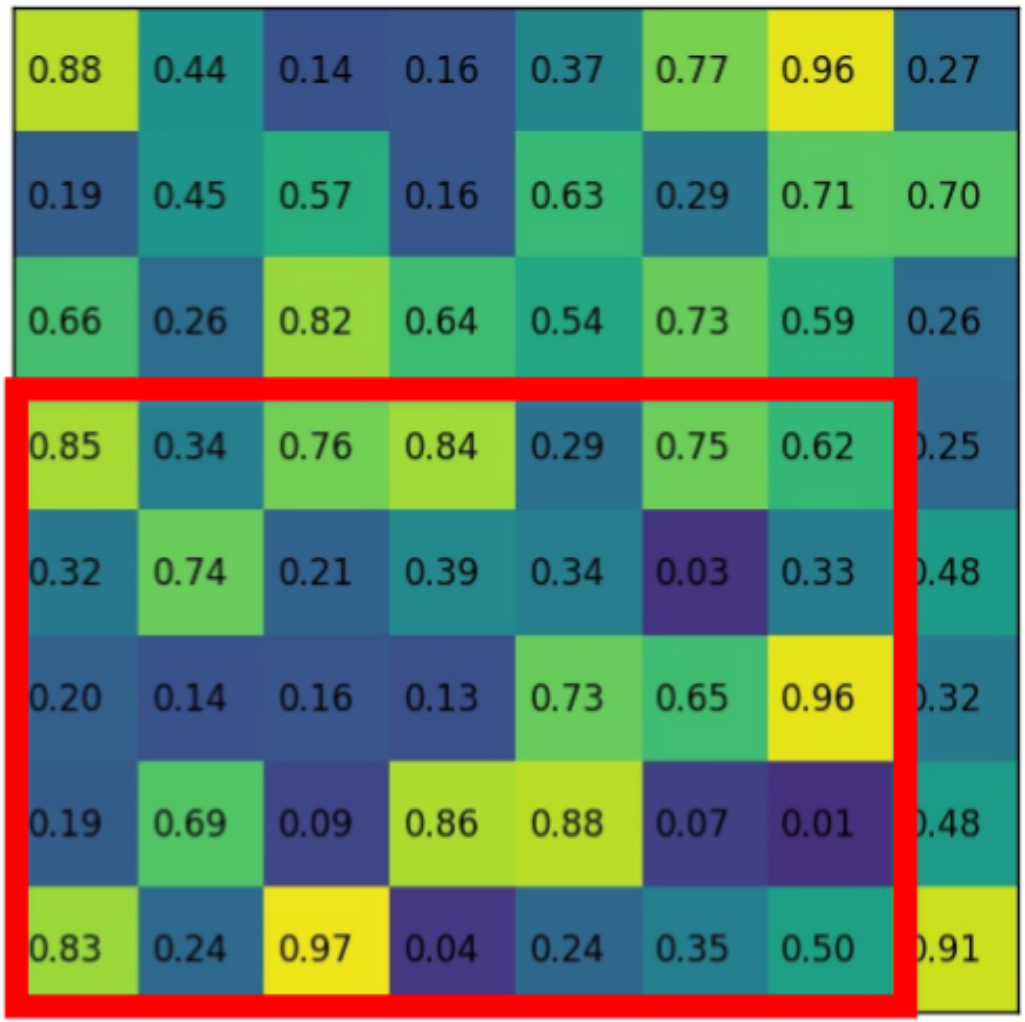
ROI Pooling

- ROI pooling employes three steps to transform the input regions to similar size feature vectors:
  - Divide the region proposal into equal-sized sections (the number of which is the same as the dimension of the output).
  - Find the largest value in each section.
  - Copy these max values to the output buffer.

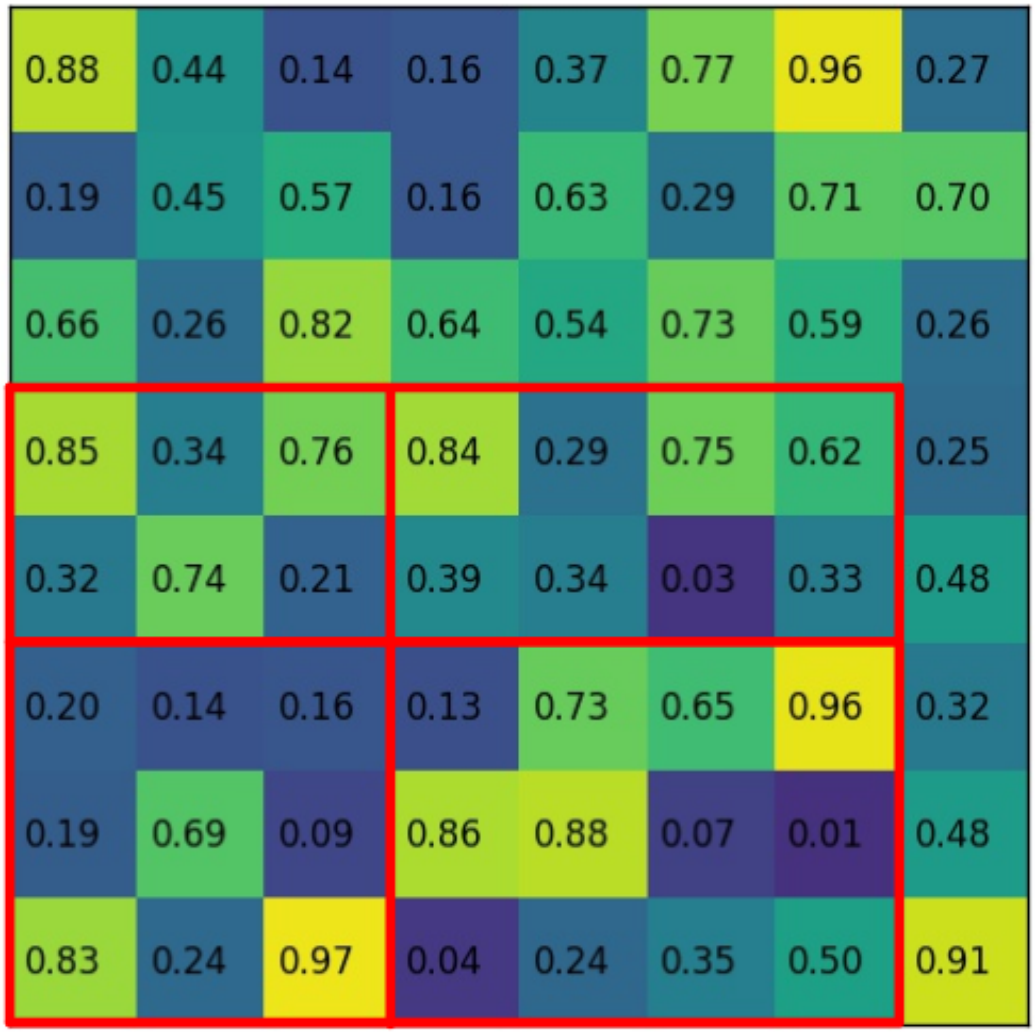
ROI Pooling: Input



ROI Pooling: Region Of Interest



ROI Pooling: Devide Region To Compartments



ROI Pooling: Max Pool Compartments

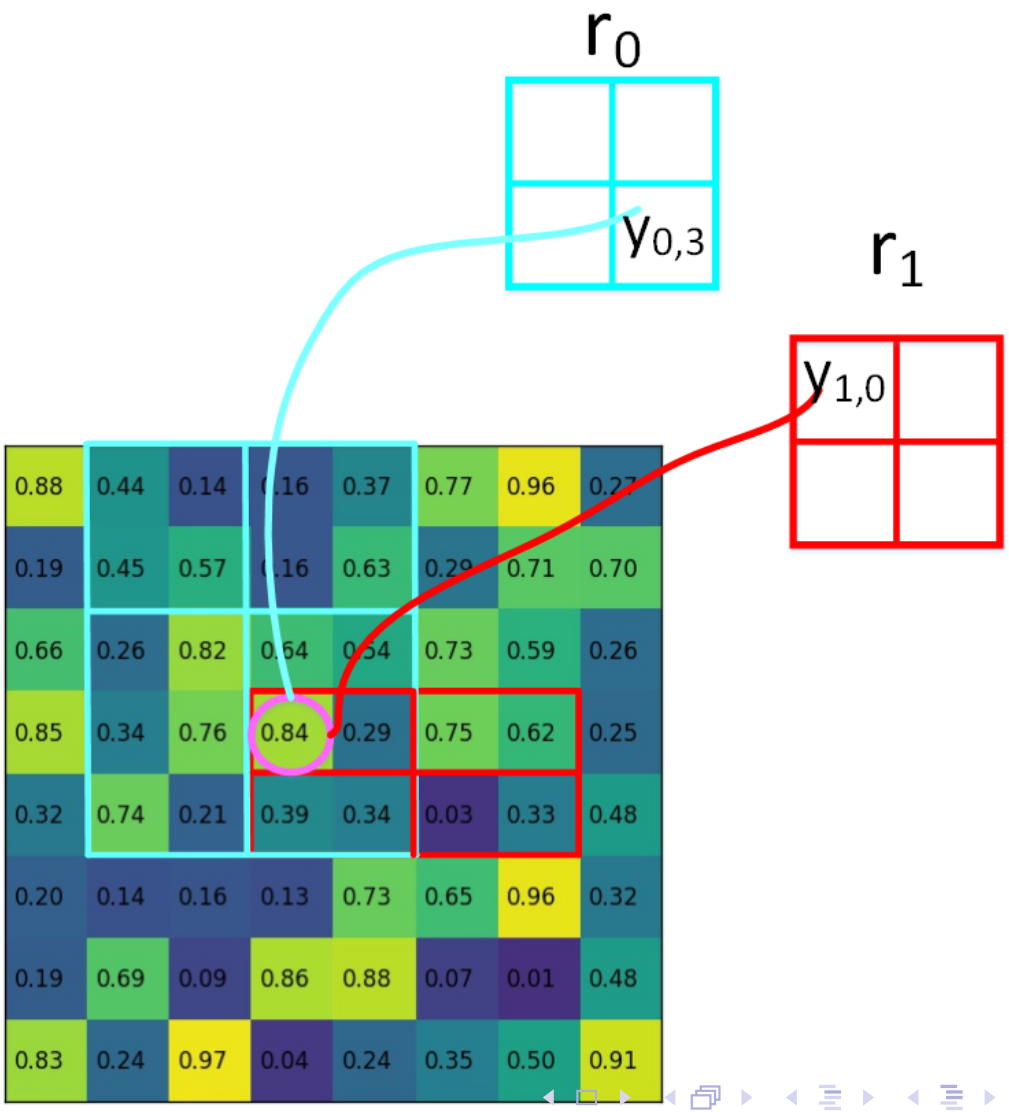
ROI Pooling: Feature Map Creation

Backporpagation Through ROI Pooling

- For training a network in an end-to-end fashion, ROI pooling layers need to be (sub)differentiable.
- To compute the gradient across the ROI pooling layer, we use the following:

$$\frac{\partial L}{\partial x_i} = \sum_{r \in regions} \sum_{j \in locations} \mathbb{I}(i = i^*(r,j)) \frac{\partial L}{\partial y_{r,j}}$$

Backpropagation Through ROI Pooling



Conclusion

- ROI pooling has different goals than regular pooling.
- It allows the network to reuse the feature map for all ROIs.
- It also allows obtaining same size feature vectors from multimodal input.
- Implementation now available in Tensor Flow. (https://github.com/deepsense-io/roi-pooling)

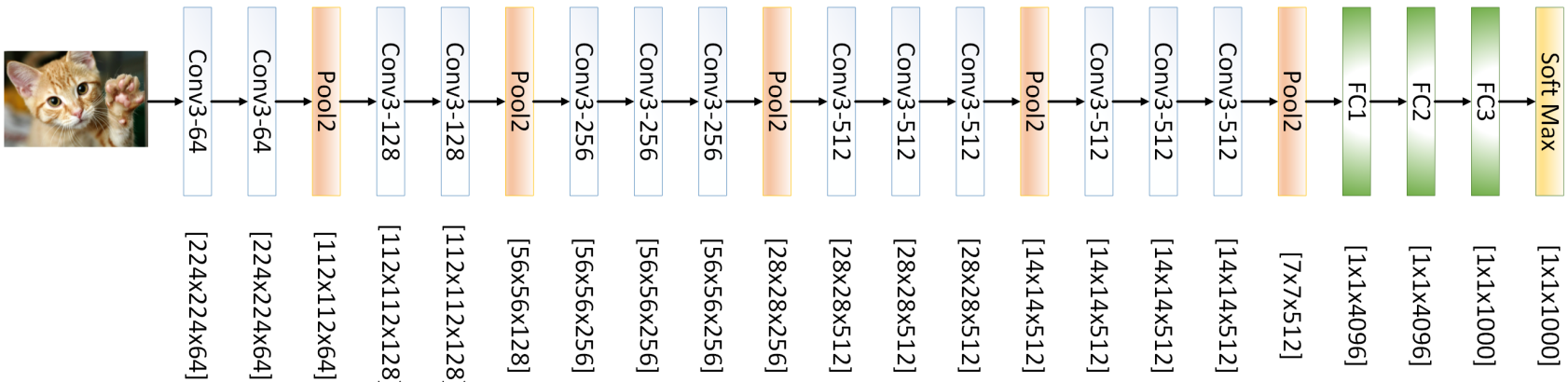
Section 7

Example of A Convolutional Neural Network: VGG16

VGG-16 Network

- Proposed in 2014 by *K.Simonyan and A. Zisserman*.
- Came in second place at the ImageNet ILSVRC-2014 challenge.
- The surprise was the overwhelming simplicity of this network.
- It only relied on 3×3 convolutional layers and $2x2$ max pooling layers, with the final 3 layers being fully connected layers.

VGG-16 Network



Section 8

Conclusion

Conclusion

- Convolutional networks have played an important role in the history of deep learning.
- They were some of the first deep models to perform well, long before arbitrary deep models were considered viable.
- Convolutional networks were also some of the first neural networks to solve important commercial applications and remain at the fore front of commercial applications of deep learning today.
- An example is Yann LeCun's Cheque reading network developed at AT&T labs in 1998.

Conclusion

- Furthermore, convolutional nets were some of the first working deep networks trained with back-propagation.
- It is not entirely clear why convolutional networks succeeded when general back-propagation networks were considered to have failed. *(Might be psychological)*
- Whatever the case, it is fortunate that convolutional networks performed well decades ago.
- In many ways, they carried the torch for the rest of deep learning and paved the way to the acceptance of neural networks in general.