Convolutional Neural Networks

Generative and Deep Learning (GDL)

Davide Bacciu (davide.bacciu@unipi.it)

UNIVERSITÀ DI PISA

Lecture(s) Outline

- Deep learning module introduction
- Introduction and historical perspective on CNNs
- Dissecting the components of a CNN
  - Convolution, stride, pooling
- CNN architectures for machine vision
  - Putting components back together
  - From LeNet to ResNet
- Advanced topics
  - Dilated convolutions
  - Preview of vision applications

Split in two lectures

Module Introduction

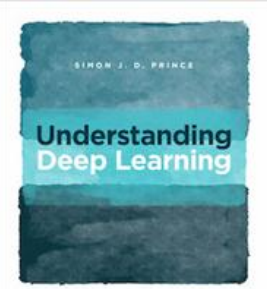
Module Outline

Module focus: foundational deep learning models and methods

- Convolutional Neural Networks
- Gated Recurrent Networks (LSTM, GRU, …)
- Neural attention
- Transformers and self-supervised training
- Propagation issues and architectures addressing them
- Coding lectures: Keras/TF and Pytorch

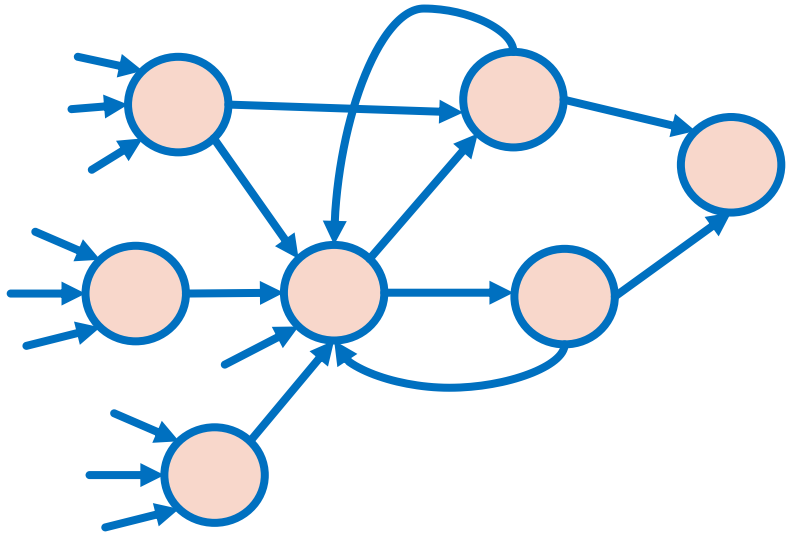
More models and in the generative DL module and in the final module

Reference Book



Simon J.D. Prince, Understanding Deep Learning, MIT Press (2023)

Module's Prerequisites



- Formal model of neuron
- Neural network
  - Feed-forward
  - Recurrent

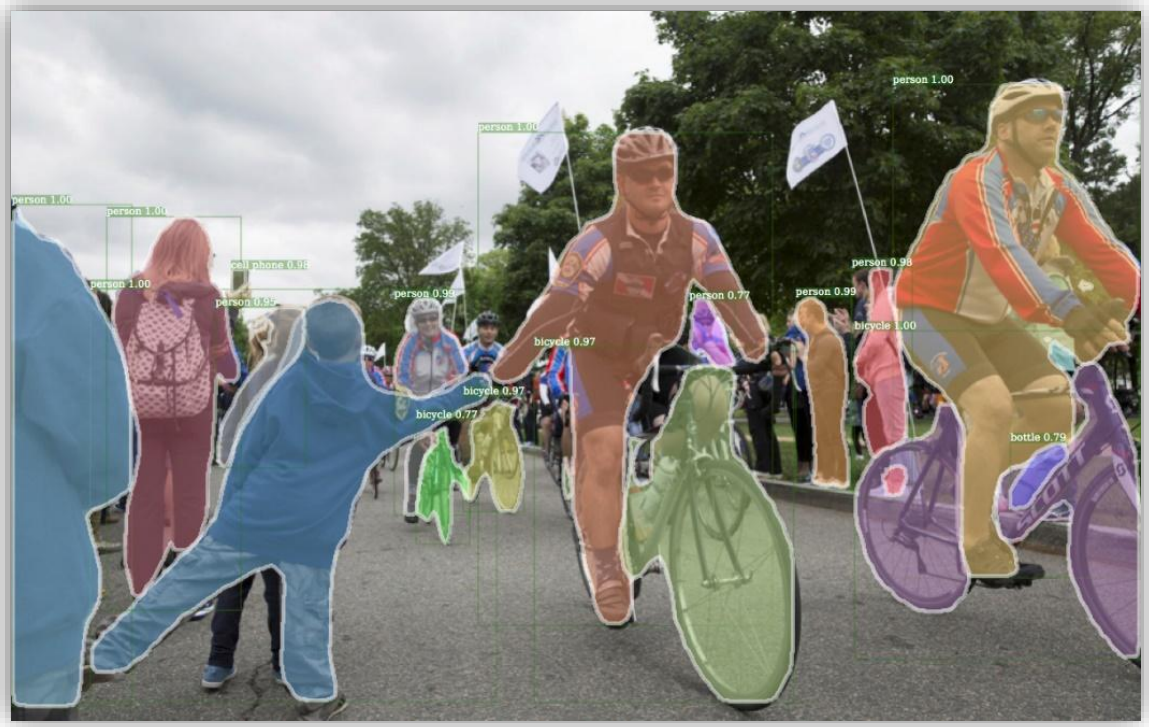
- Cost function optimization
  - Backpropagation/SGD and optimizers
  - Regularization
- Neural network hyper-parameters and model selection

Introduction to CNNs

Part I

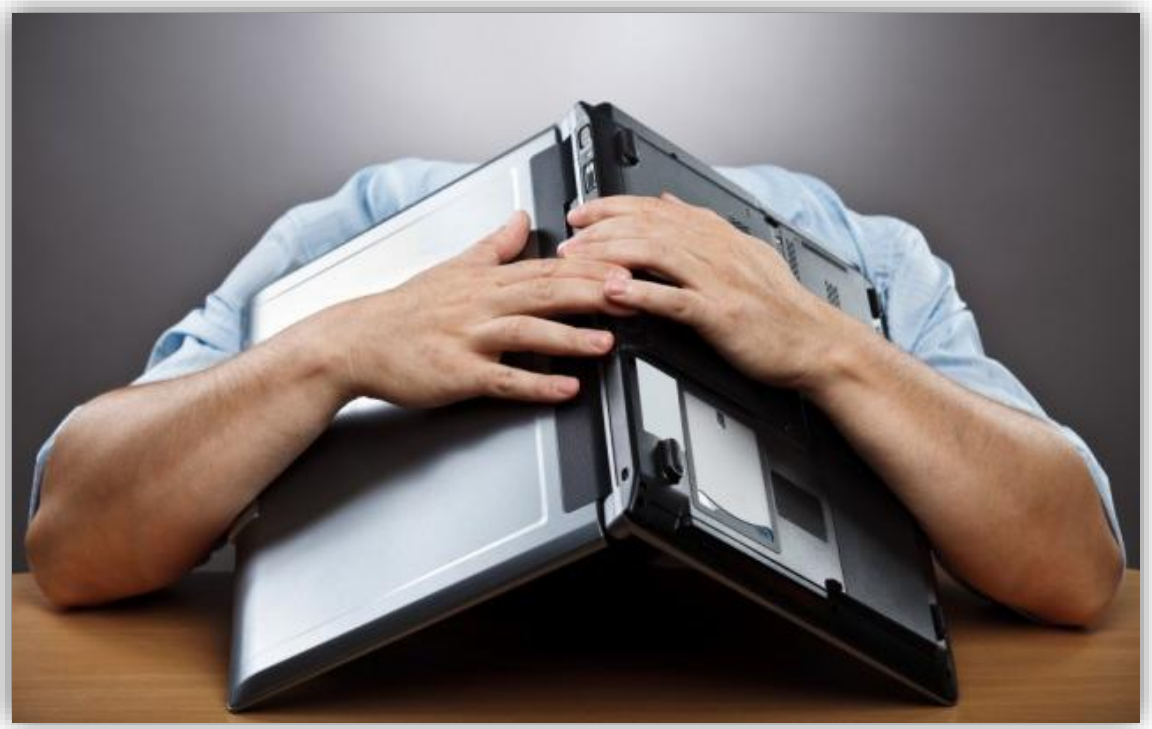
Introduction

Convolutional Neural Networks



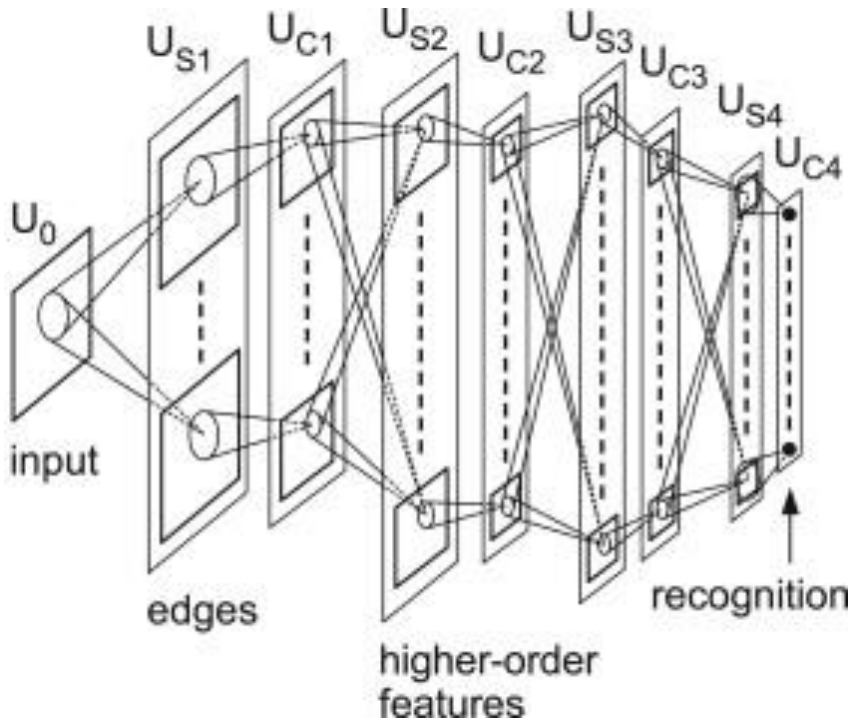
Introduction

Convolutional Neural Networks



Destroying Machine Vision research(ers) since 2012

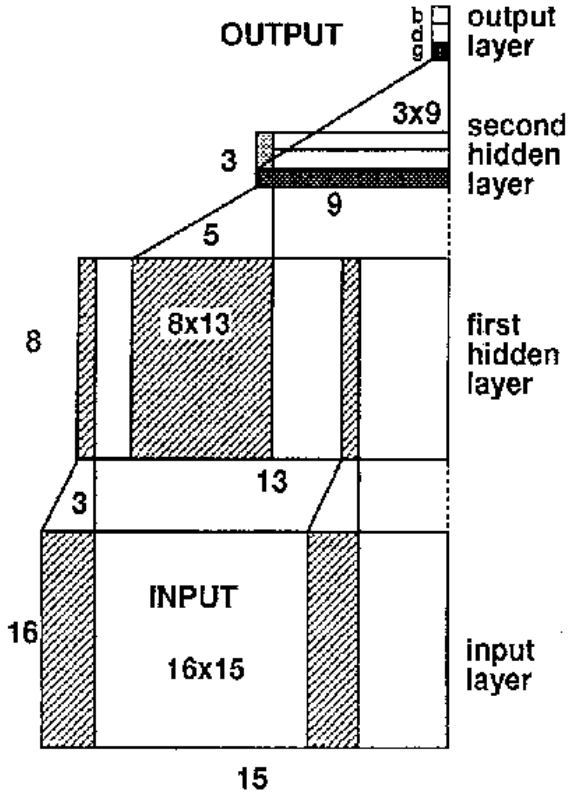
Neocognitron



Trained by unsupervised learning

- Hubel-Wiesel ('59) model of brain visual processing
  - Simple cells responding to localized features
  - Complex cells pooling responses of simple cells for invariance
- Fukushima ('80) built the first hierarchical image processing architecture exploiting this model

CNN for Sequences

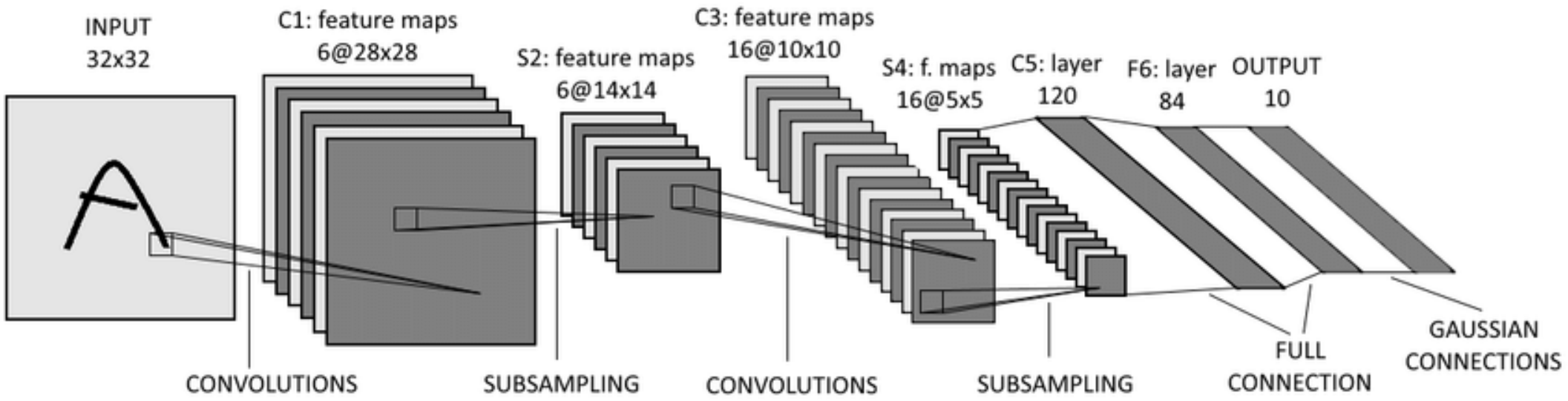


Time delay neural network (Waibel & Hinton, 1987)

- Apply a bank of 16 convolution kernels to sequences (windows of 15 elements)
- Trained by backpropagation with parameter sharing
- Guess who introduced it?

  …yeah, HIM!

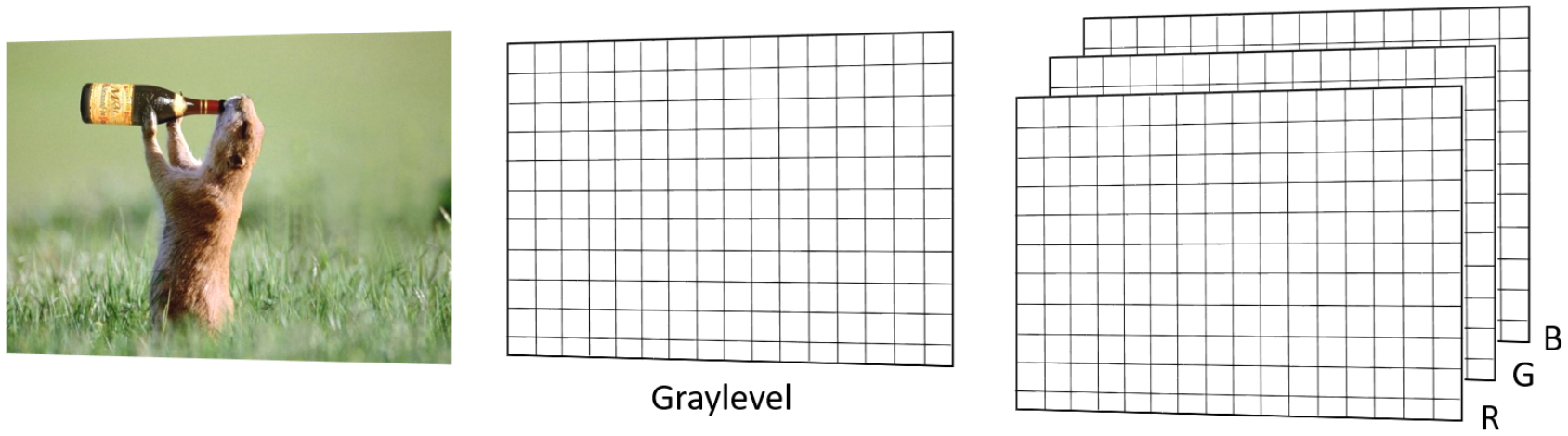
CNN for Images



First convolutional neural network for images dates back to 1989 (LeCun)

Image Data

Images are matrices of pixel intensities or color values (RGB)



- Other representations exist, but not of interest for the course
- You will see more on this in the Computer Vision course

Dense Vector Multiplication

Processing images: the dense way

32x32x3 image

Reshape it into a vector

x

3072

An input-sized weight vector for each hidden neuron

W

100x3072

Wx^T

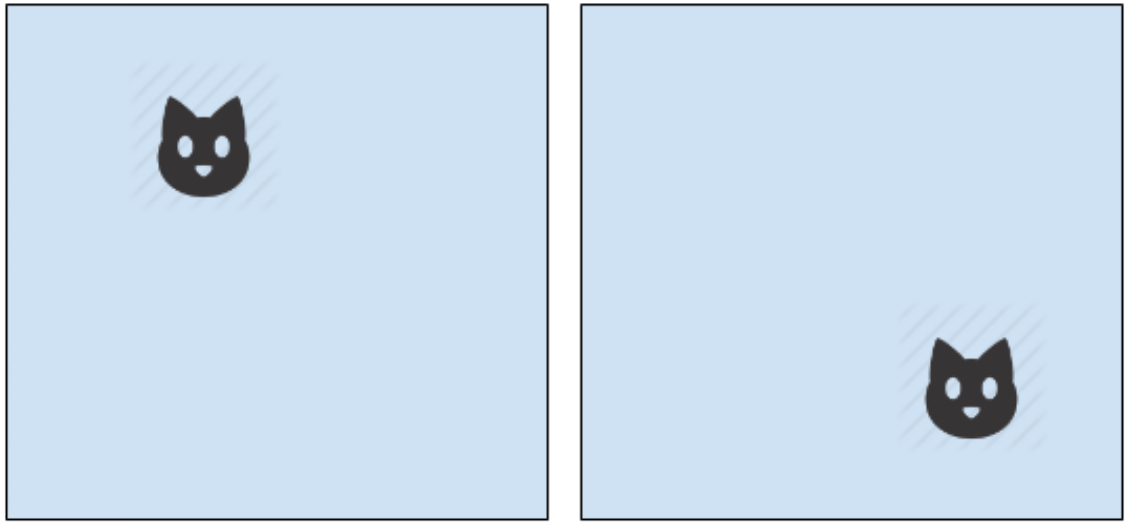
100

Each element contains the activation of 1 neuron

About invariances

MLPs are positional

We (most likely) need translation invariance!



- If we unfold the two images into two vectors, the features identifying the cat will be in different positions
- But this still remains a picture of a cat, which we would like to classify as such irrespectively of its position in the image

An inductive bias to keep in mind

Nearby pixels are more correlated than far away ones

The input representation should not destroy pixel relationships (like vectorization does)

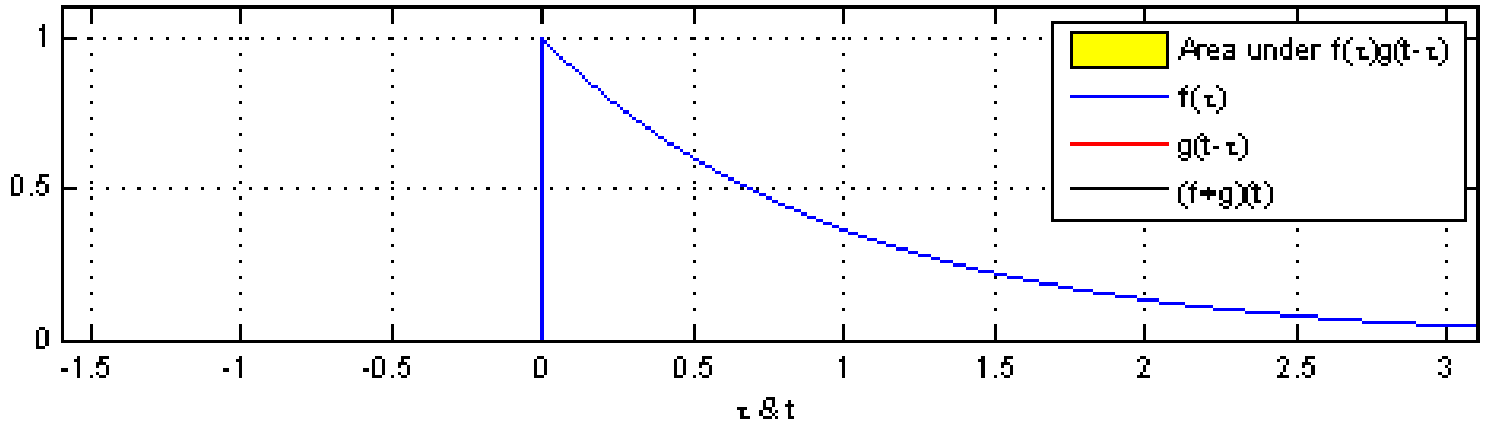
So what?

We need a new type of neuron that can scan the pixels of an image sequentially and apply the same set of weights to the different image parts

Translated in mathematical terms: we need convolution!

Convolution

- A commutative smoothing operator between two signals: one is kept fixed and the other one slides computing a weighted-average signal

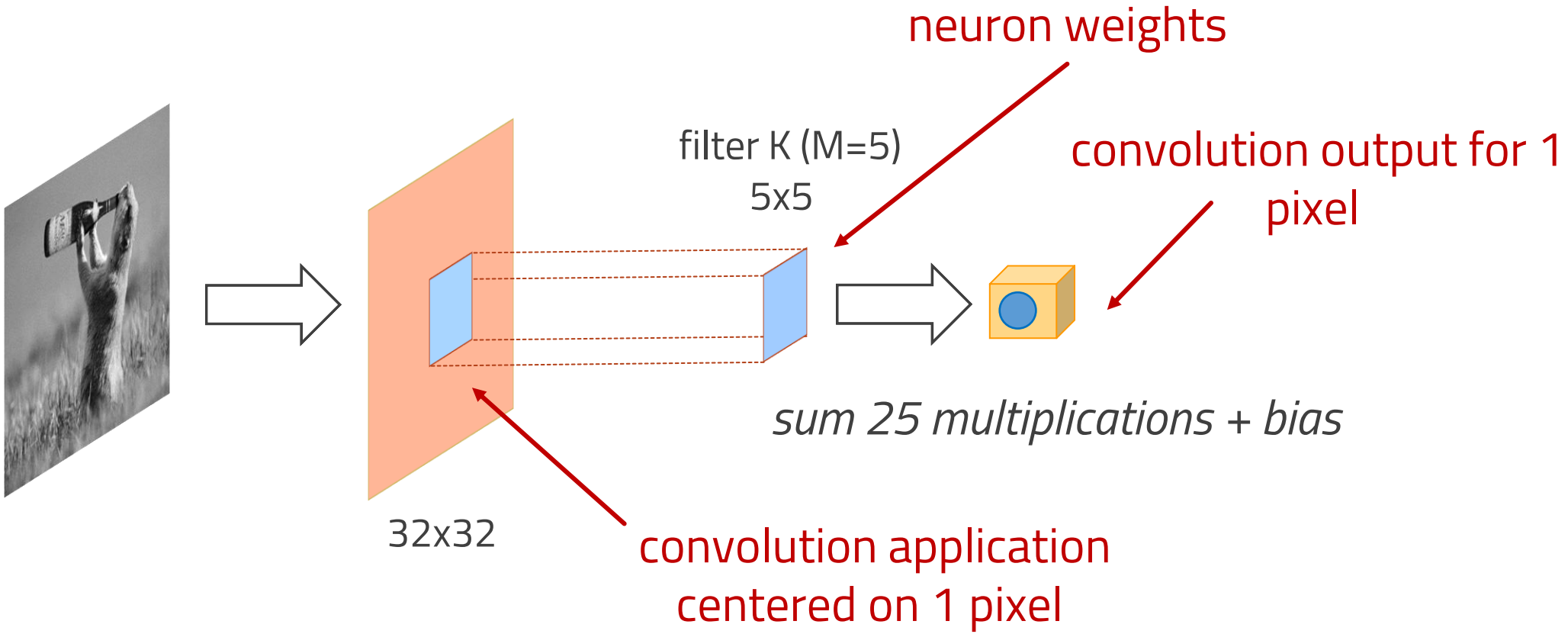


- For our purposes we will focus on discrete convolution between an image I and a filter/kernel K on finite support $[-M, +M]$. That is

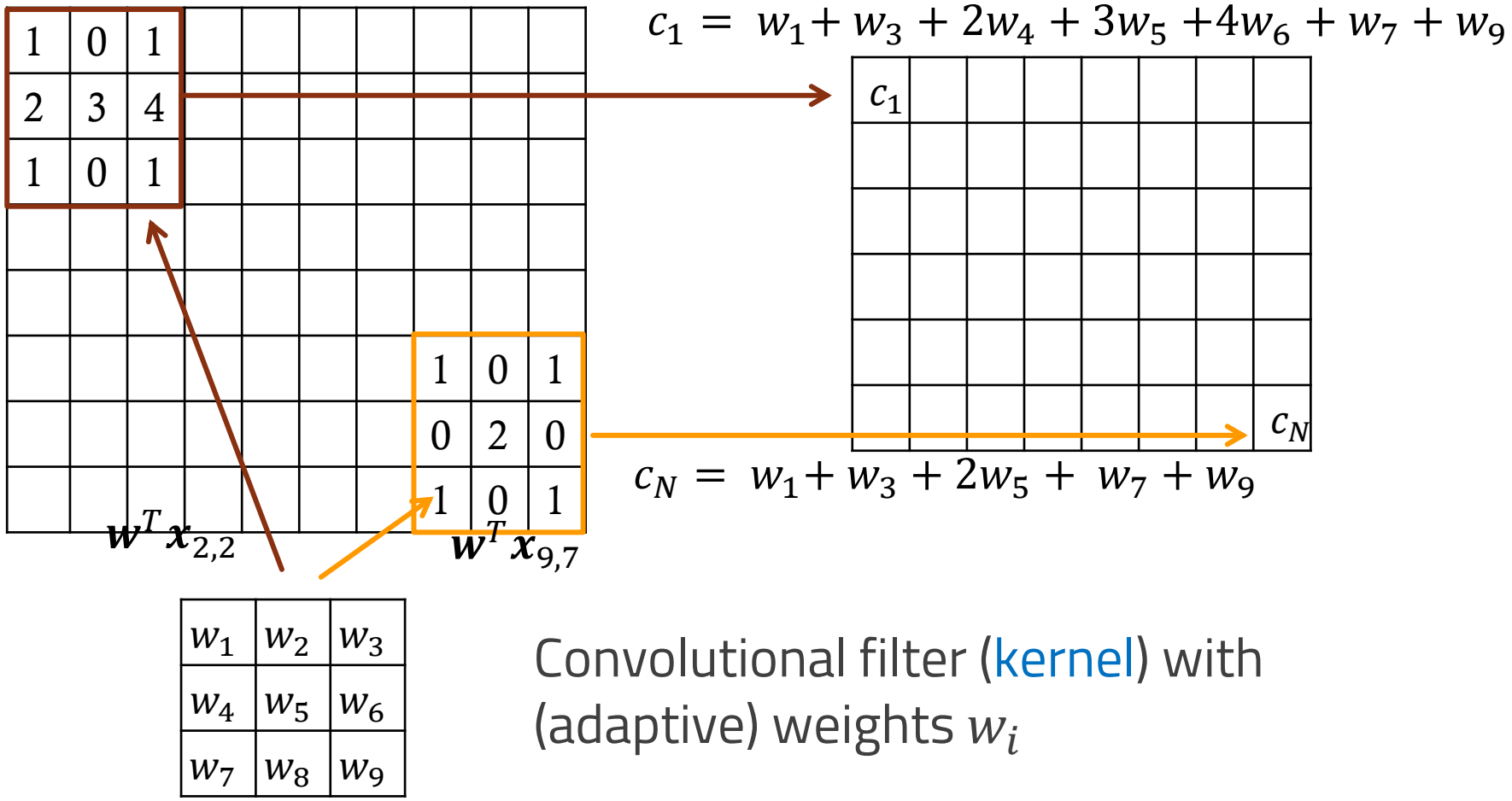
$$(I * K)(i,j) = \sum_m \sum_n I(i-m, j-n) K(m,n)$$

CNNs building blocks

2D Convolution

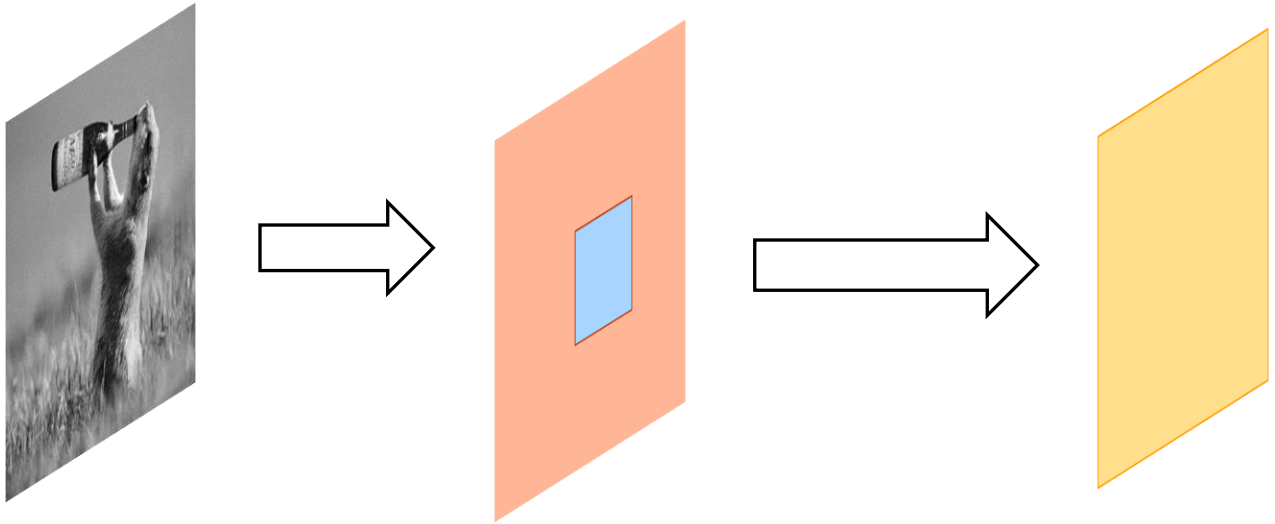


Adaptive Convolution



Convolutional filter (kernel) with (adaptive) weights w_i

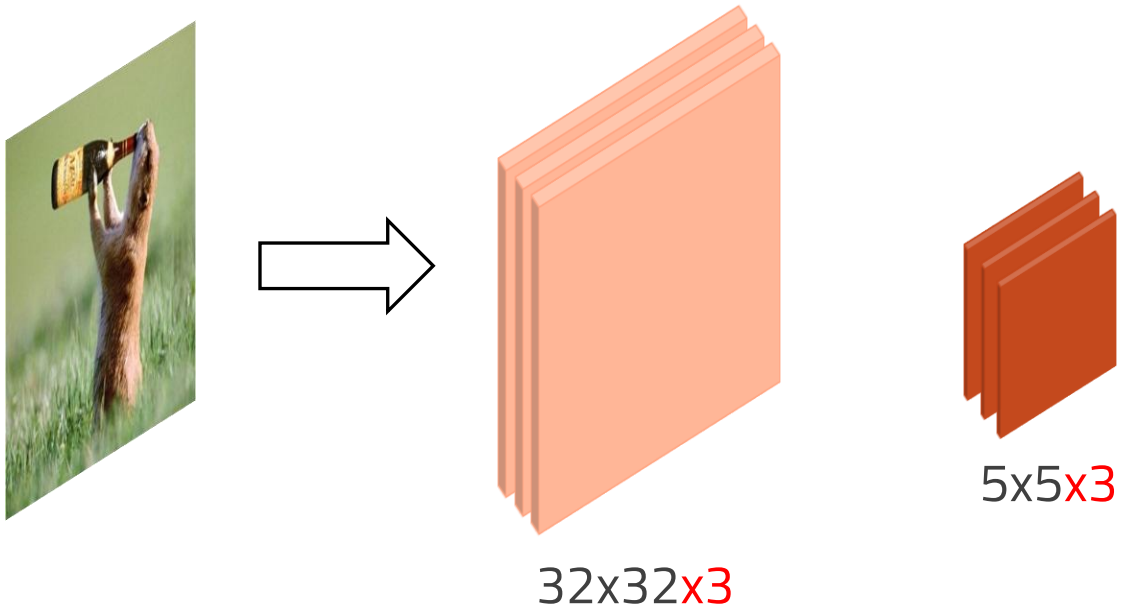
Convolutional Features



Convolution features

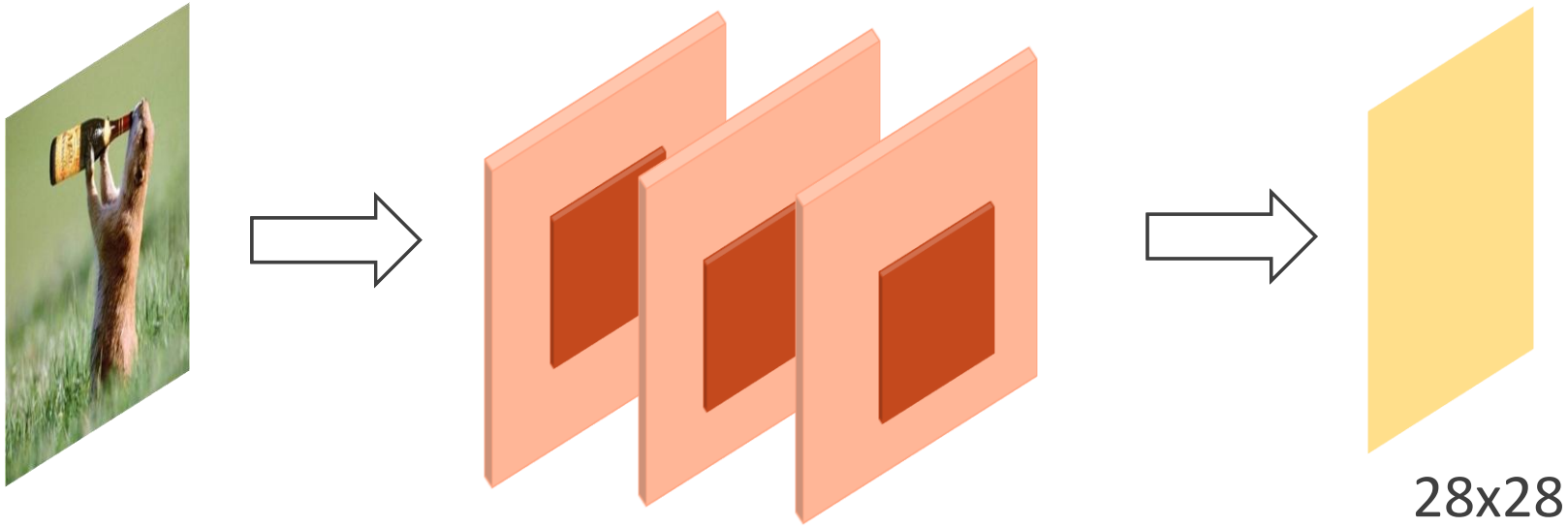
Slide the filter on the image computing elementwise products and summing up

Multi-Channel Convolution



Convolution filter has a number of slices equal to the number of image channels

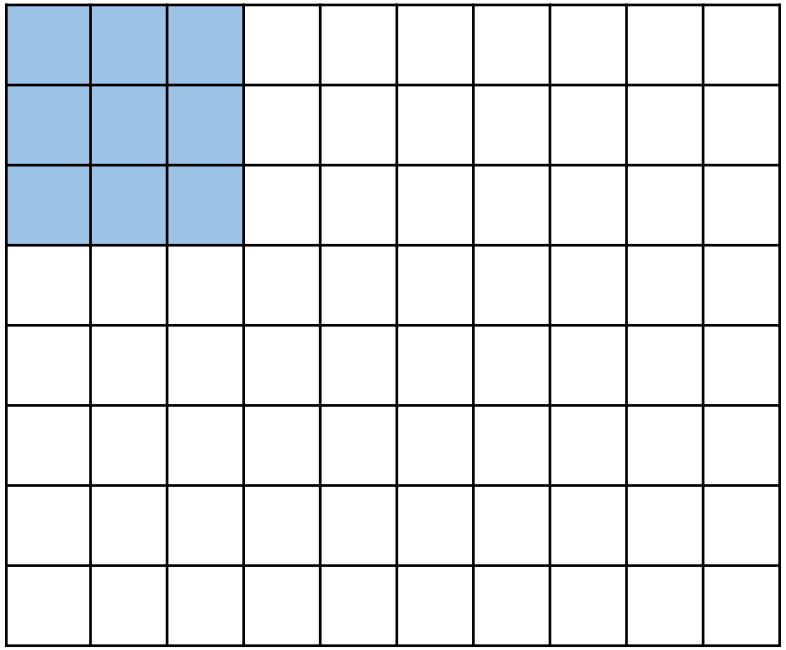
Multi-Channel Convolution



All channels are typically convolved together

- They are summed-up in the convolution
- The convolution map stays bi-dimensional

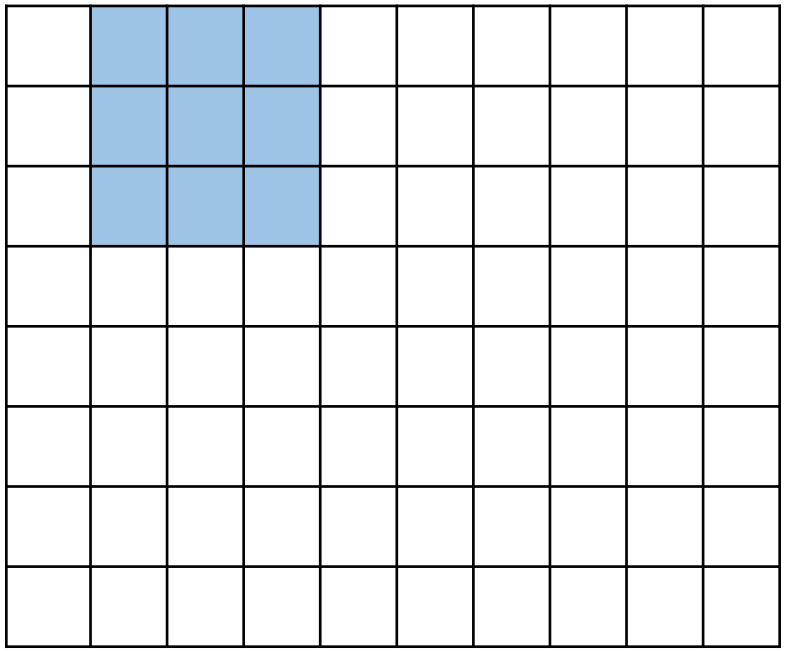
Stride



Basic convolution slides the filter on the image one pixel at a time

- Stride = 1

Stride

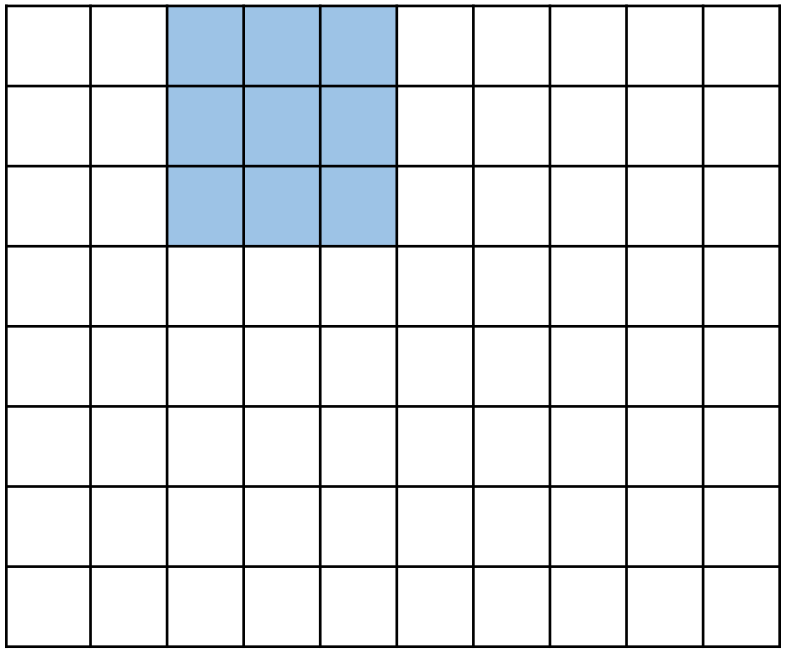


stride = 1

Basic convolution slides the filter on the image one pixel at a time

- Stride = 1

Stride

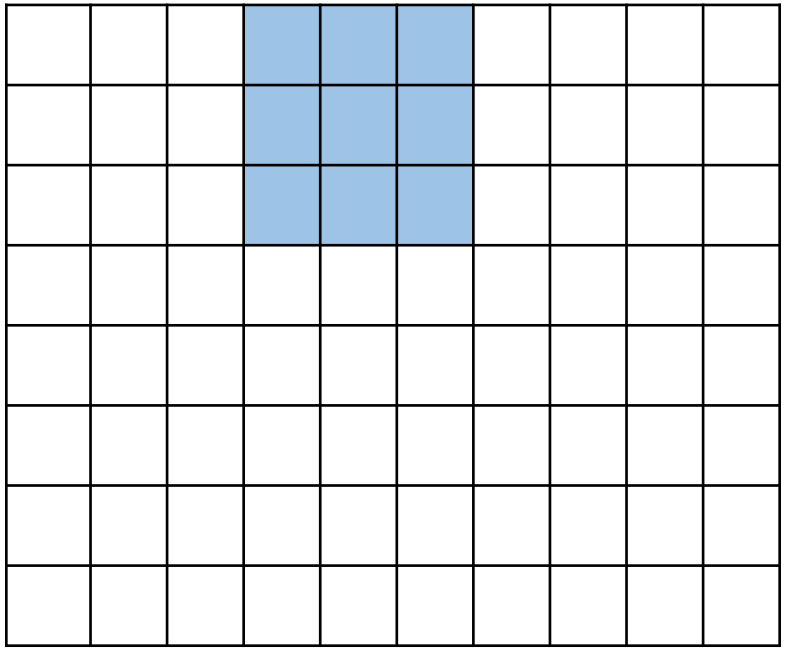


stride = 1

Basic convolution slides the filter on the image one pixel at a time

- Stride = 1

Stride

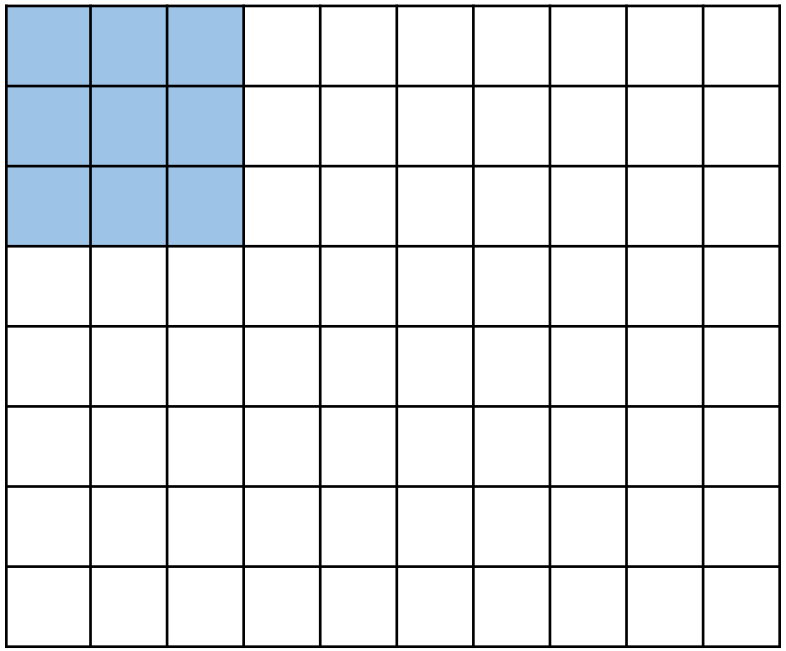


stride = 1

Basic convolution slides the filter on the image one pixel at a time

- Stride = 1

Stride



stride = 2

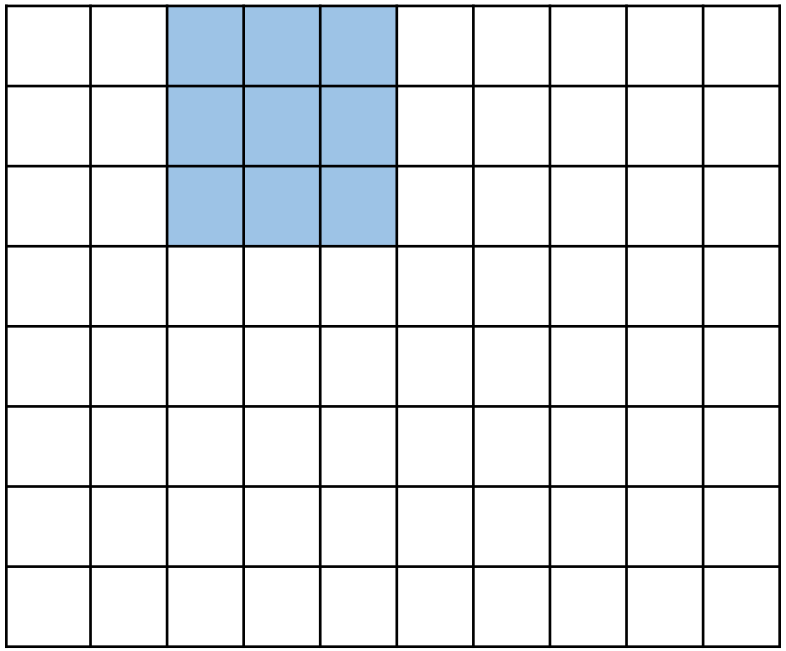
Basic convolution slides the filter on the image one pixel at a time

- Stride = 1

Can define a different stride

- Hyperparameter

Stride



stride = 2

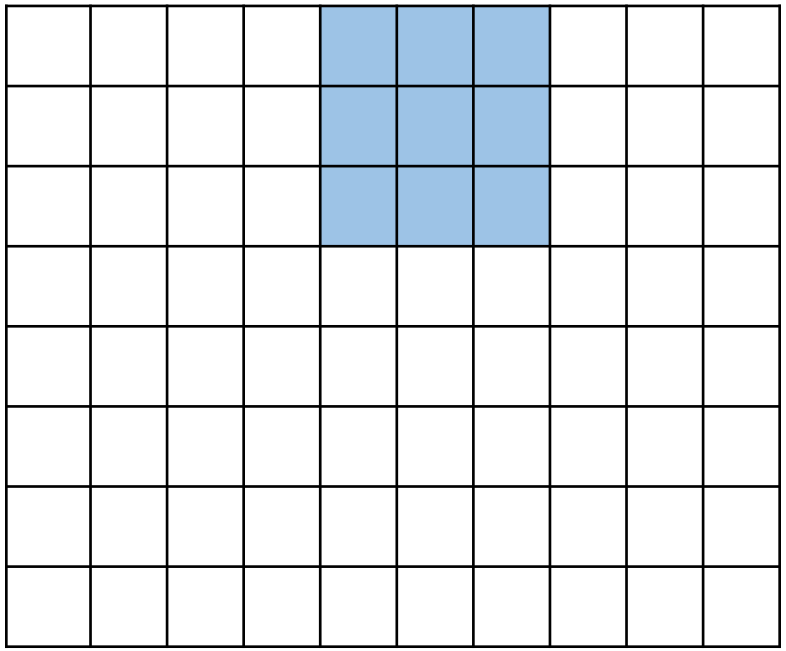
Basic convolution slides the filter on the image one pixel at a time

- Stride = 1

Can define a different stride

- Hyperparameter

Stride



stride = 2

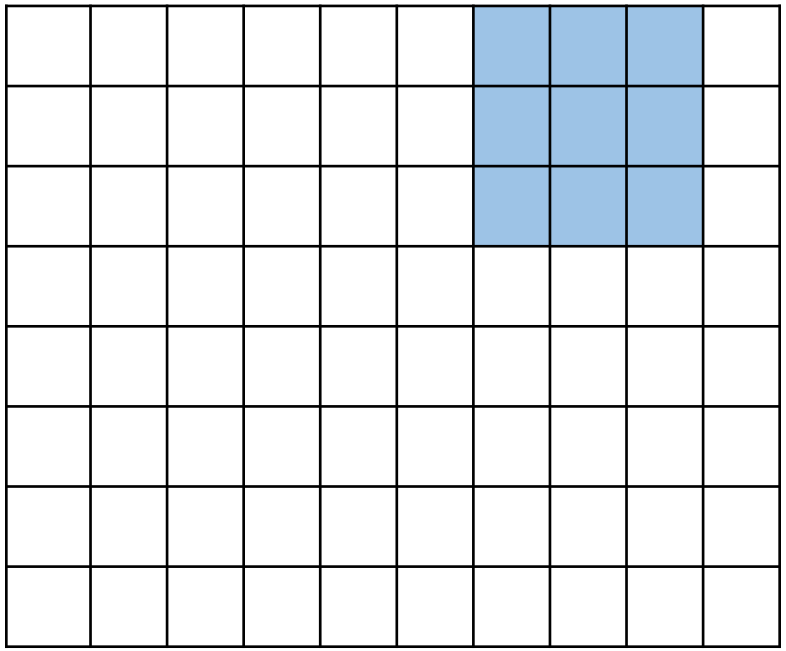
Basic convolution slides the filter on the image one pixel at a time

- Stride = 1

Can define a different stride

- Hyperparameter

Stride



stride = 2

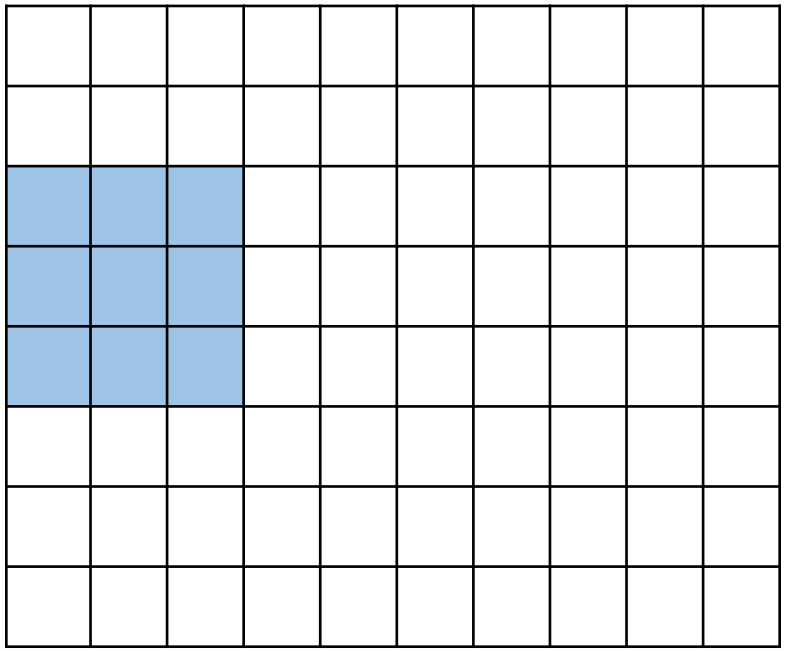
Basic convolution slides the filter on the image one pixel at a time

- Stride = 1

Can define a different stride

- Hyperparameter

Stride



stride = 2

Works in both directions!

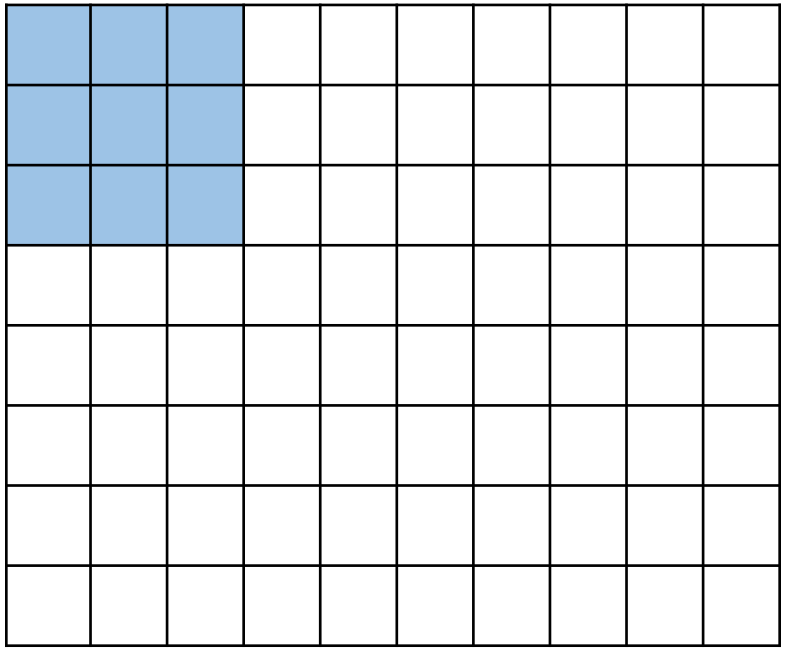
Basic convolution slides the filter on the image one pixel at a time

- Stride = 1

Can define a different stride

- Hyperparameter

Stride



stride = 3

Basic convolution slides the filter on the image one pixel at a time

- Stride = 1

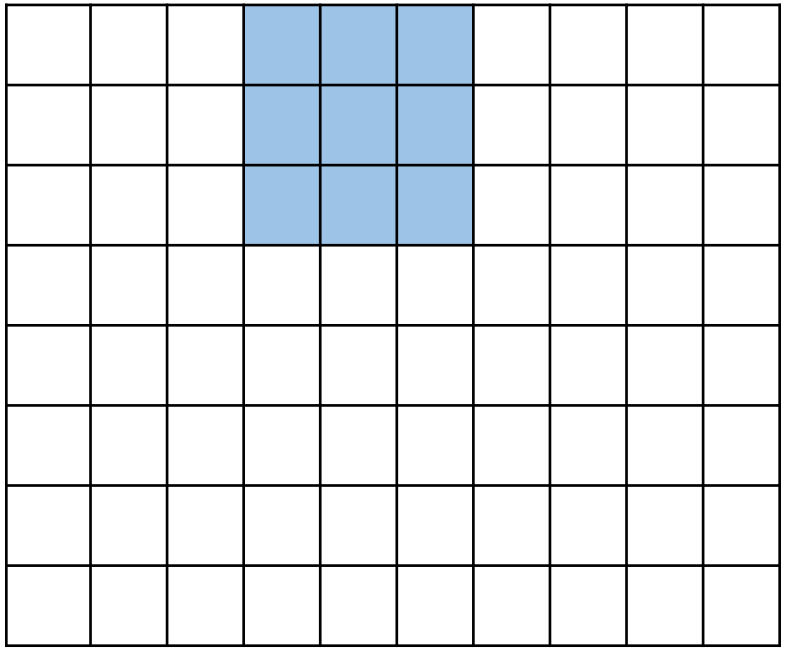
Can define a different stride

- Hyperparameter

Stride reduces the number of multiplications

- Subsamples the image

Stride



stride = 3

Basic convolution slides the filter on the image one pixel at a time

- Stride = 1

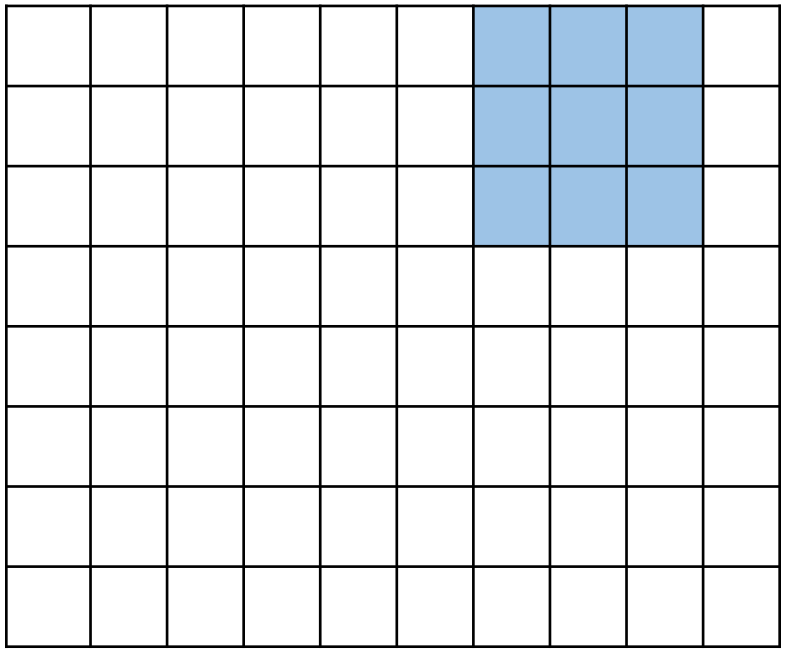
Can define a different stride

- Hyperparameter

Stride reduces the number of multiplications

- Subsamples the image

Stride



stride = 3

Basic convolution slides the filter on the image one pixel at a time

- Stride = 1

Can define a different stride

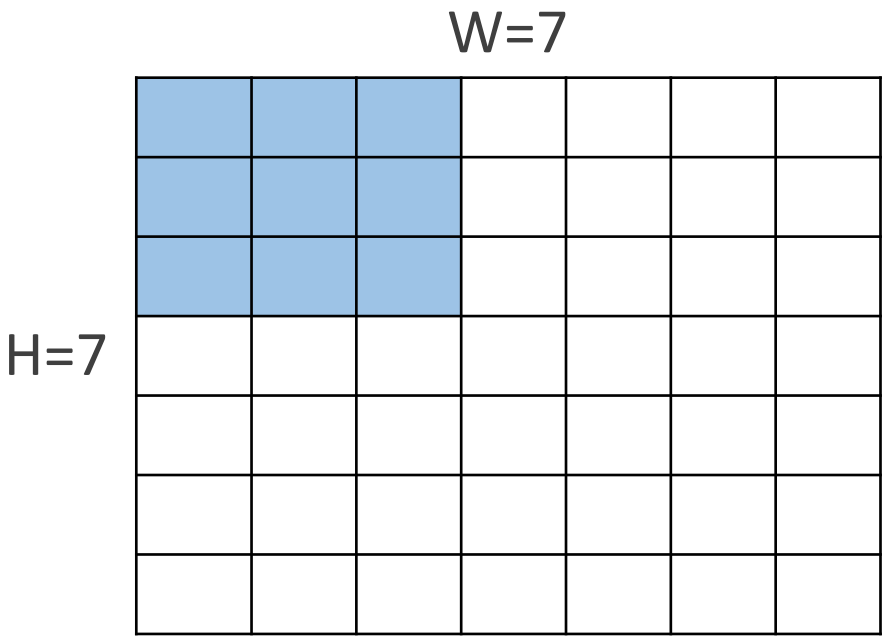
- Hyperparameter

Stride reduces the number of multiplications

- Subsamples the image

Activation Map Size

What is the size of the image after application of a filter with a given size and stride?



Take a 3x3 filter with stride 1

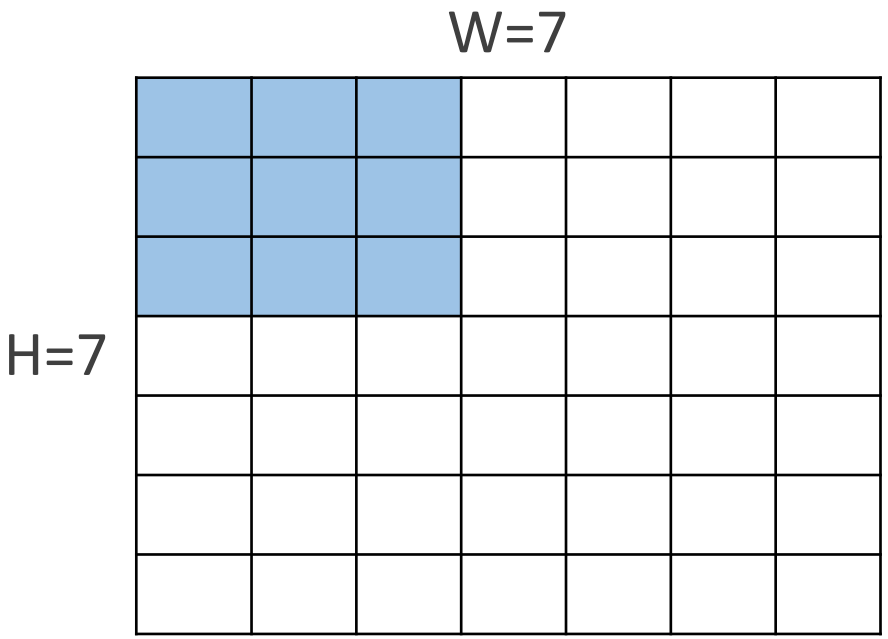
K=3, S=1

⇩

Output image is: 5x5

Activation Map Size

What is the size of the image after application of a filter with a given size and stride?



Take a 3x3 filter with stride 2

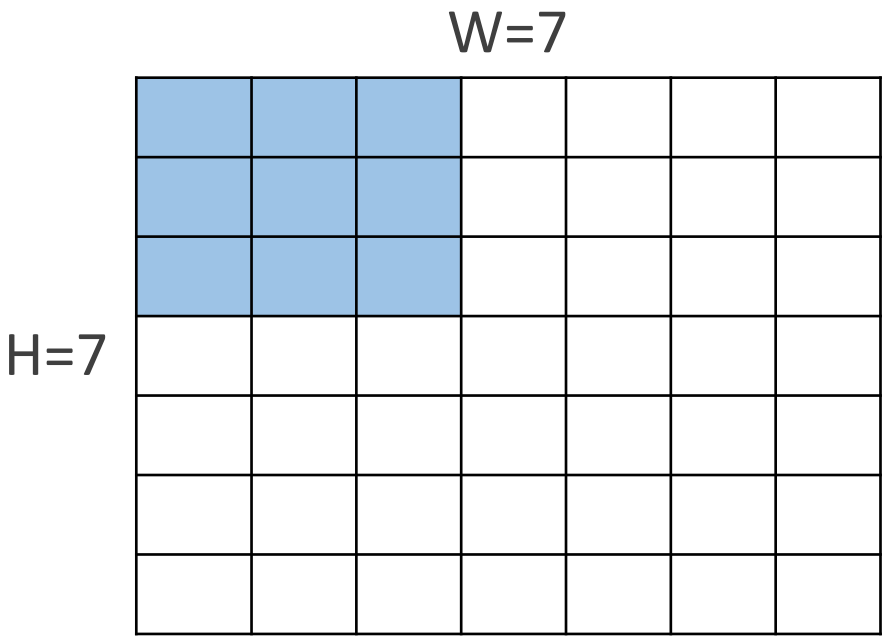
K=3, S=2

⇩

Output image is: 3x3

Activation Map Size

What is the size of the image after application of a filter with a given size and stride?



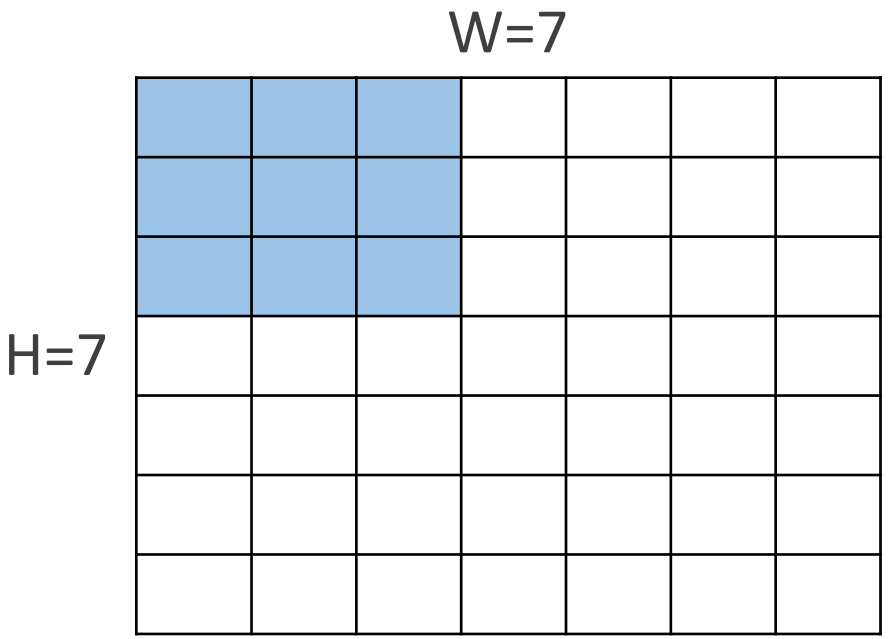
General rule

$$W' = \frac{W - K}{S} + 1$$

$$H' = \frac{H - K}{S} + 1$$

Activation Map Size

What is the size of the image after application of a filter with a given size and stride?



Take a 3x3 filter with stride 3

K=3, S=3

⇩

Output image is:

not really an image!

Zero Padding

Add columns and rows of zeros to the border of the image

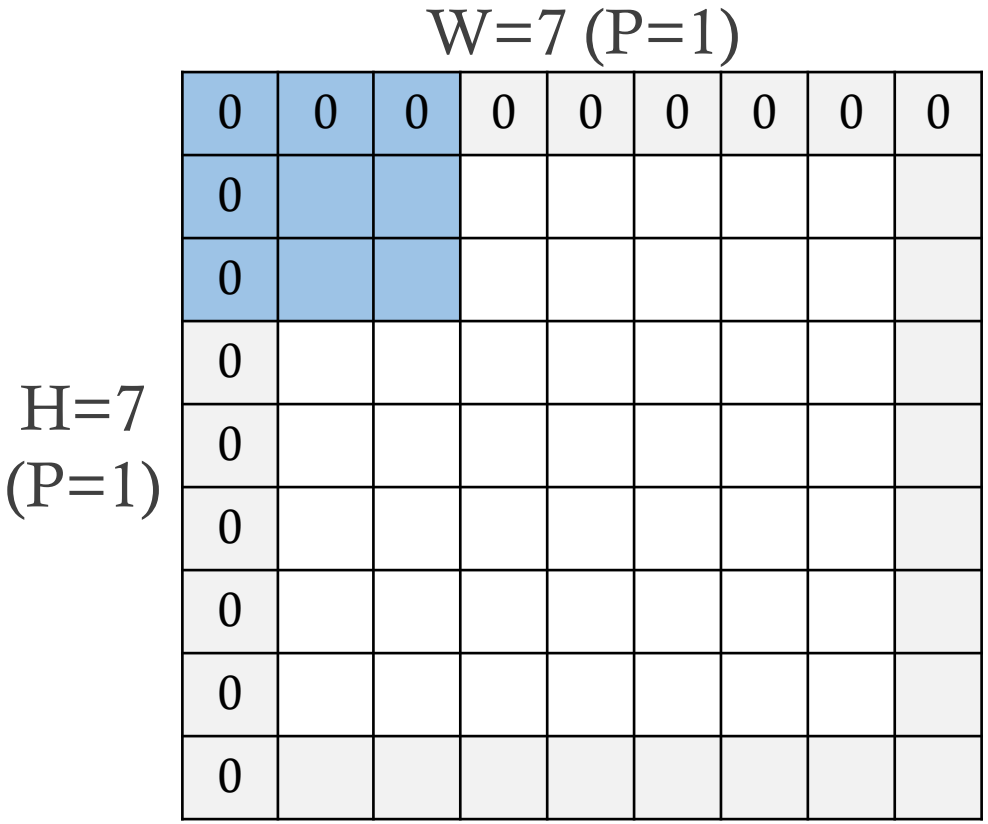
W=7

H=7

0	0	0	0	0	0	0	0	0
0								
0								
0								
0								
0								
0								
0								
0								

Zero Padding

Add columns and rows of zeros to the border of the image



K=3, S=1

⇩

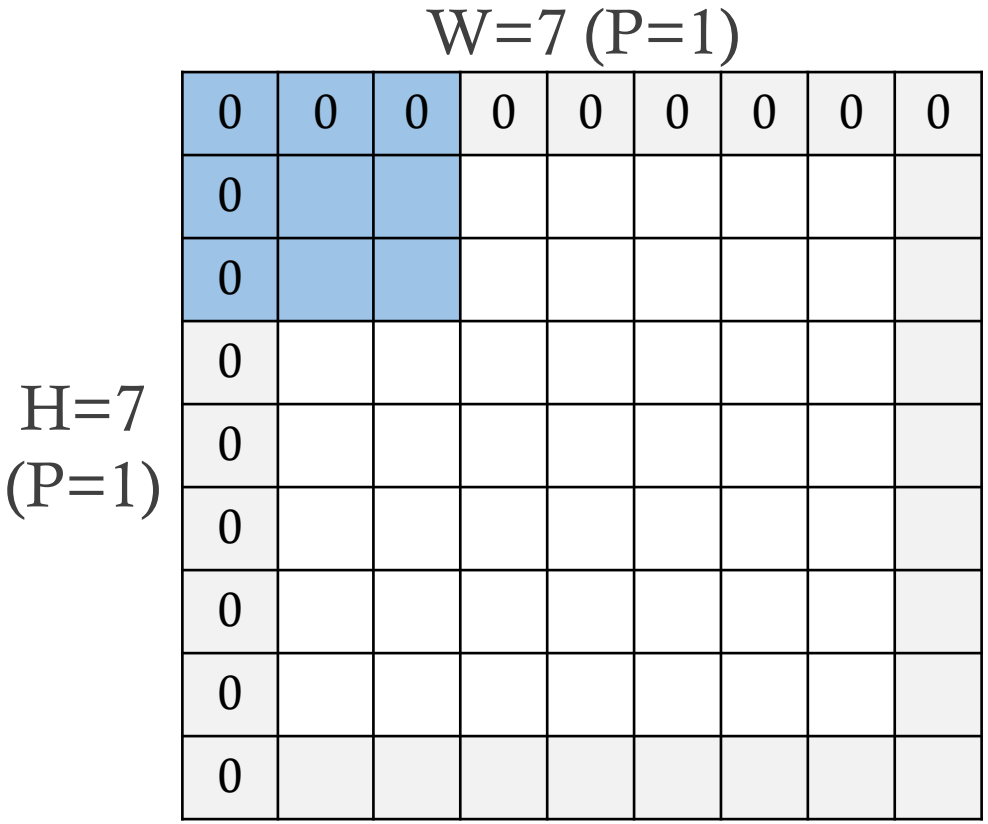
Output image is?

$$W' = \frac{W - K + 2P}{S} + 1$$

7x7

Zero Padding

Add columns and rows of zeros to the border of the image

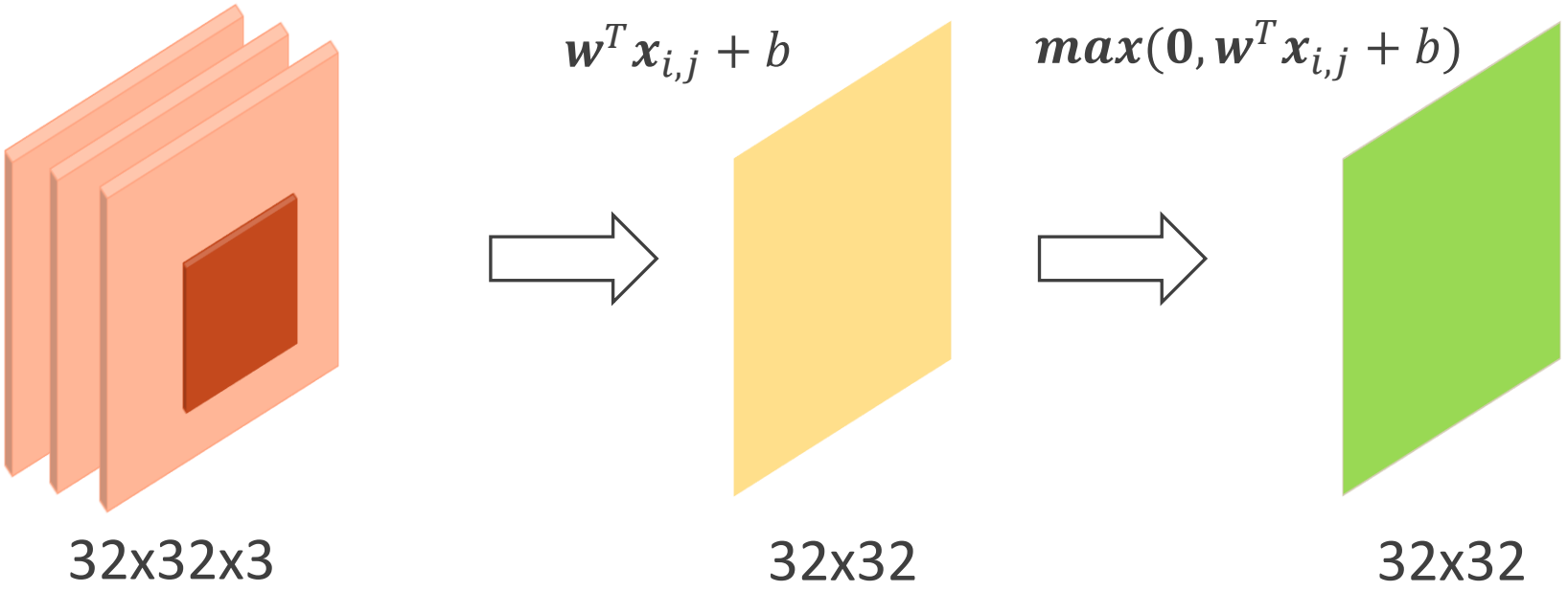


Zero padding serves to retain the original size of image

$$P = \frac{K - 1}{2}$$

Pad as necessary to perform convolutions with a given stride S

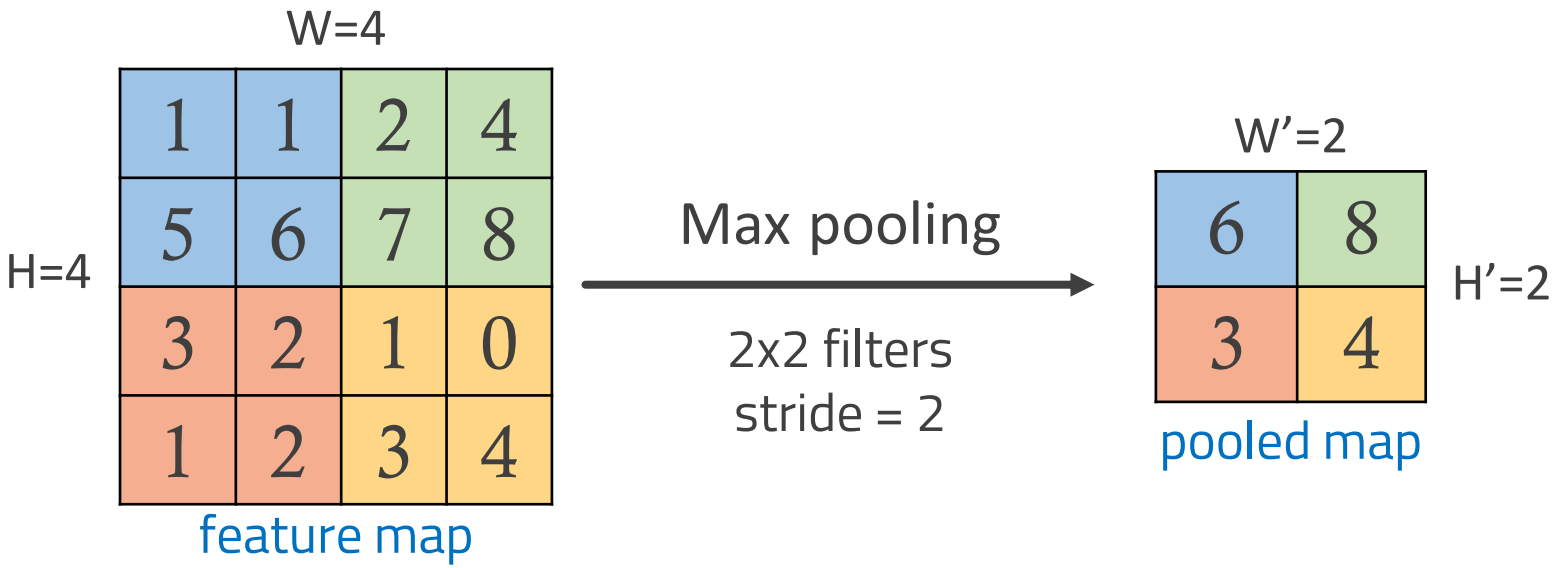
Feature Map Transformation



- Convolution is a linear operator
- Apply an element-wise nonlinearity to obtain a transformed feature map

Pooling

- Operates on the feature map to make the representation
  - Smaller (subsampling)
  - Robust to (some) transformations



Pooling Facts

- Max pooling is the one used more frequently, but other forms are possible
  - Average pooling
  - L2-norm pooling
  - Random pooling
- It is uncommon to use zero padding with pooling

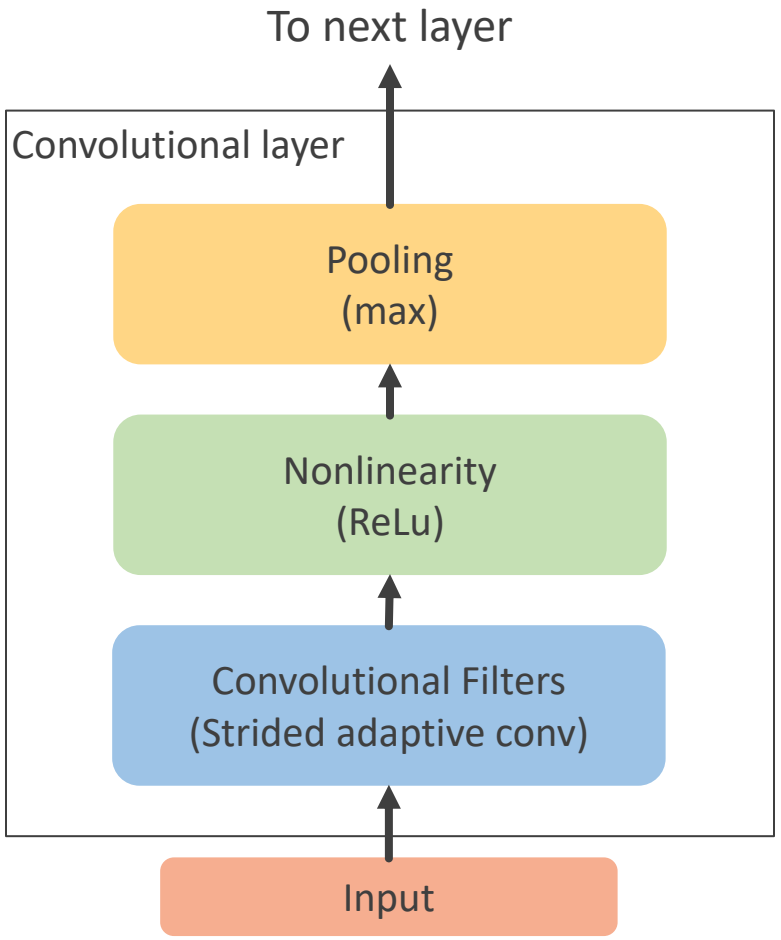
$$W' = \frac{W - K}{S} + 1$$

Convolutions as neural layers

Lecture I

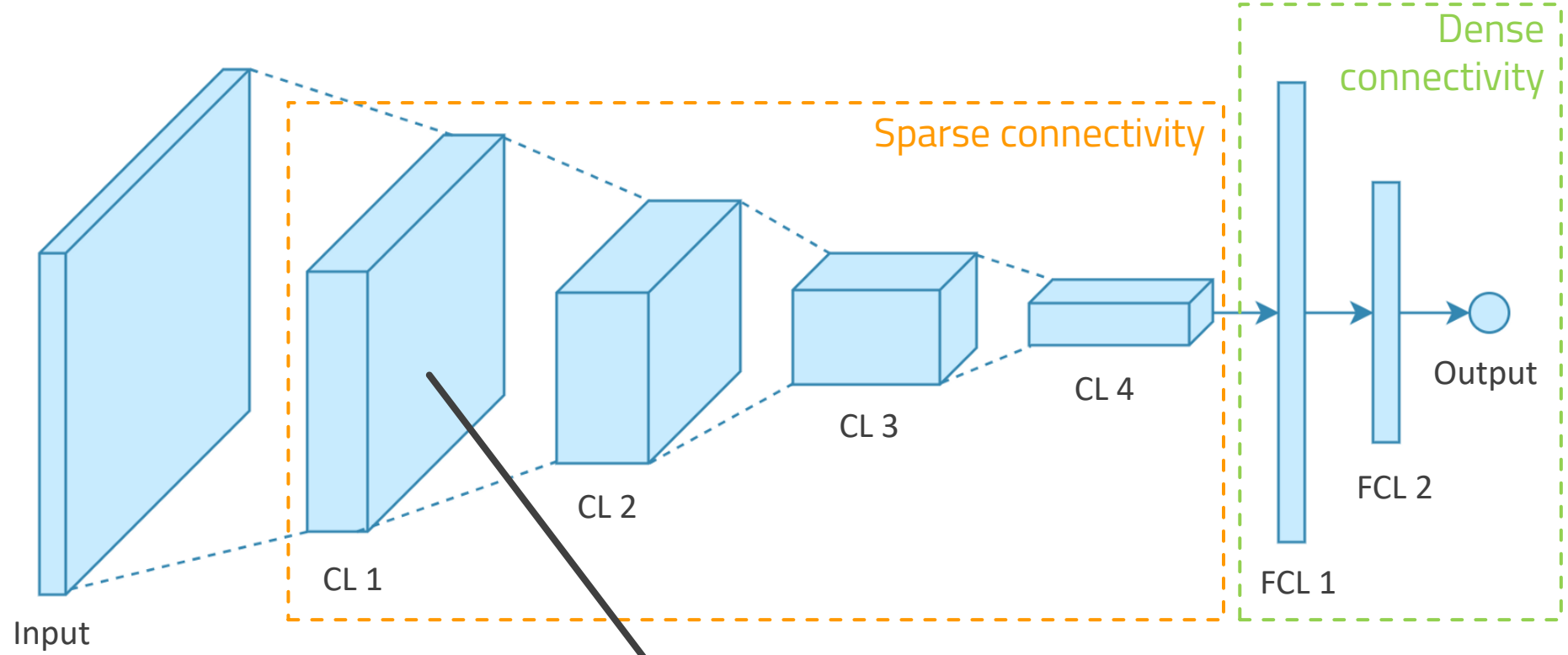
Università di Pisa

The Convolutional Layer



- An module made by a hierarchical composition of the basic elements
- Convolution layer is an abstraction for the composition of the 3 basic operations
- Network parameters are in the convolutional component

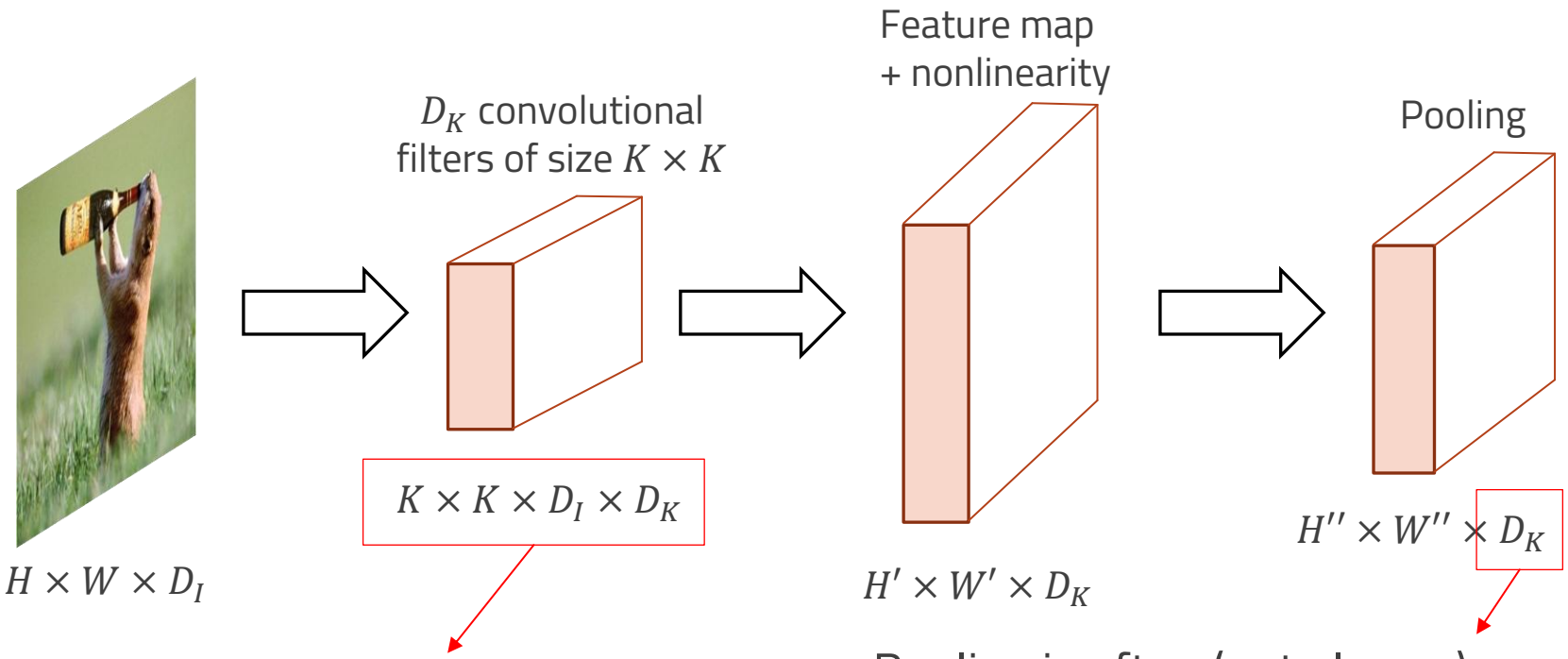
A Bigger Picture



CL -> Convolutional Layer
FCL -> Fully Connected Layer

Contains several convolutional filters with different size and stride

Convolutional Filter Banks



Number of model parameters due to this convolution element (add D_K bias terms)

Pooling is often (not always) applied independently on the D_K convolutions

Specifying CNN in Code (Keras)

Number of convolution filters D_k

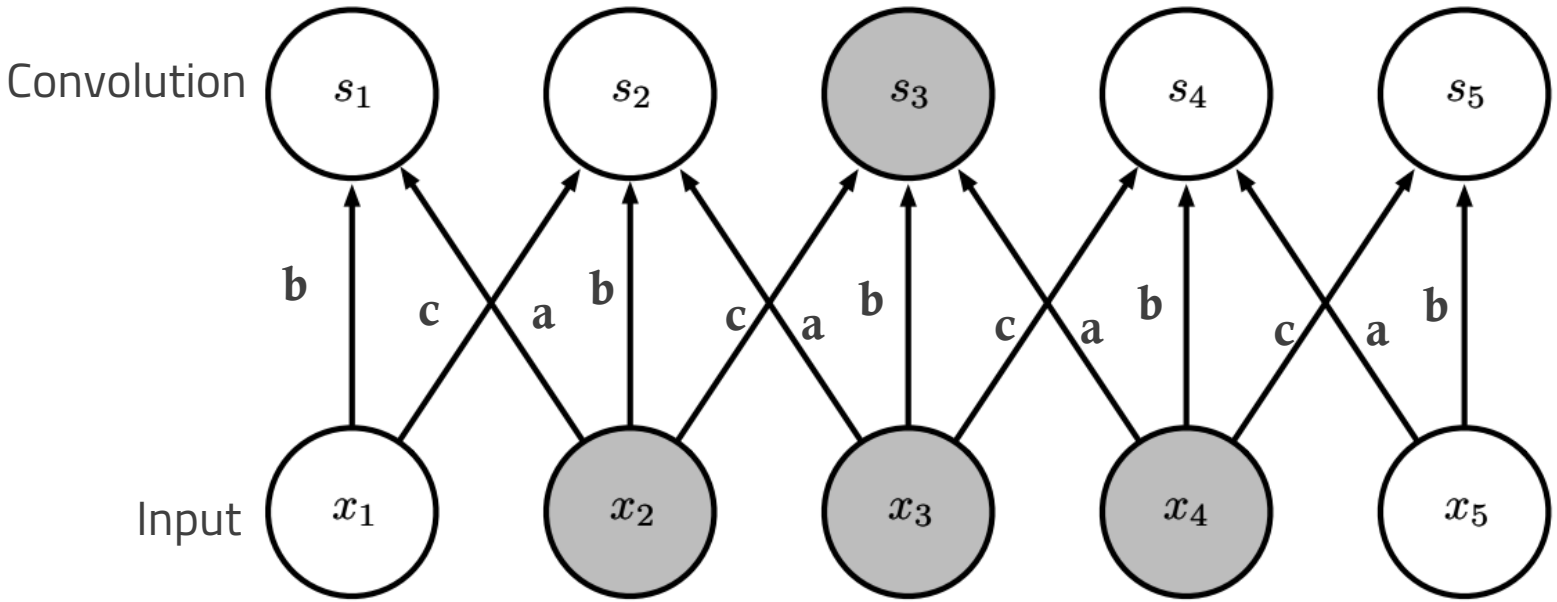
Define input size (only first hidden layer)

```
model = Sequential()
model.add(Conv2D(32, kernel_size=(5, 5), strides=(1, 1),
                 activation='relu',
                 input_shape=input_shape))
model.add(MaxPooling2D(pool_size=(2, 2), strides=(2, 2)))
model.add(Conv2D(64, (5, 5))
model.add(Activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Flatten())
model.add(Dense(1000, activation='relu'))
model.add(Dense(num_classes, activation='softmax'))
```

Does for you all the calculations to determine the final size to the dense layer

CNN as a Sparse Neural Network

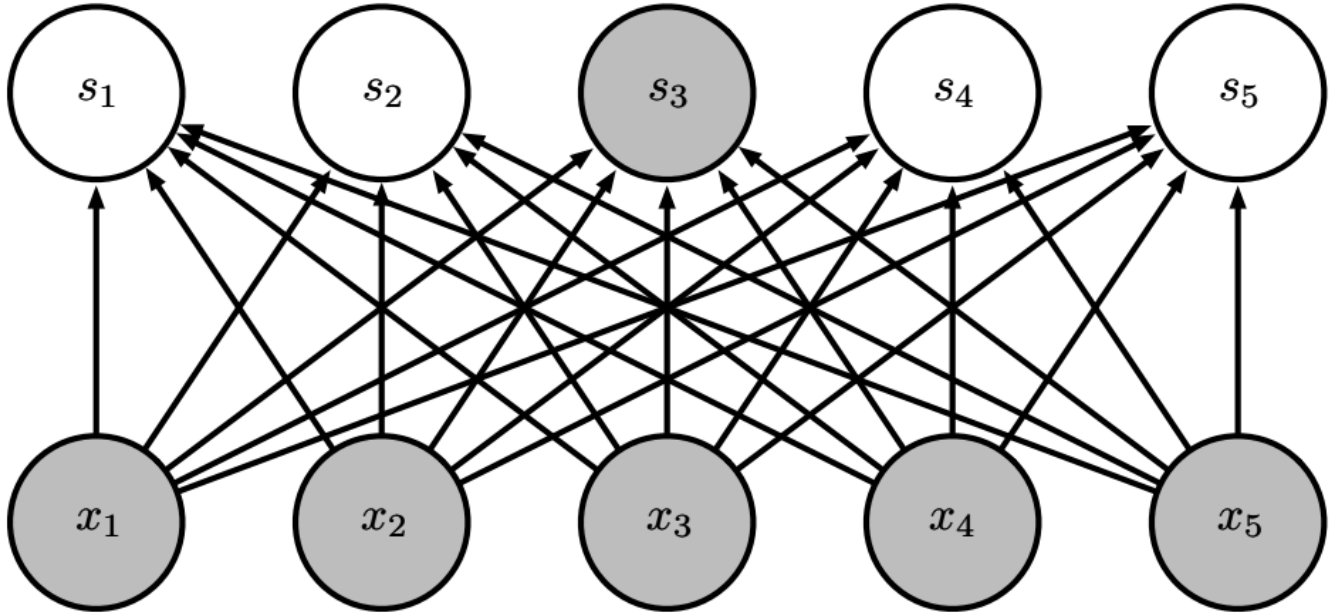
Let us take a 1-D input (sequence) to ease graphics



Convolution amounts to sparse connectivity (reduce parameters) with parameter sharing (enforces invariance)

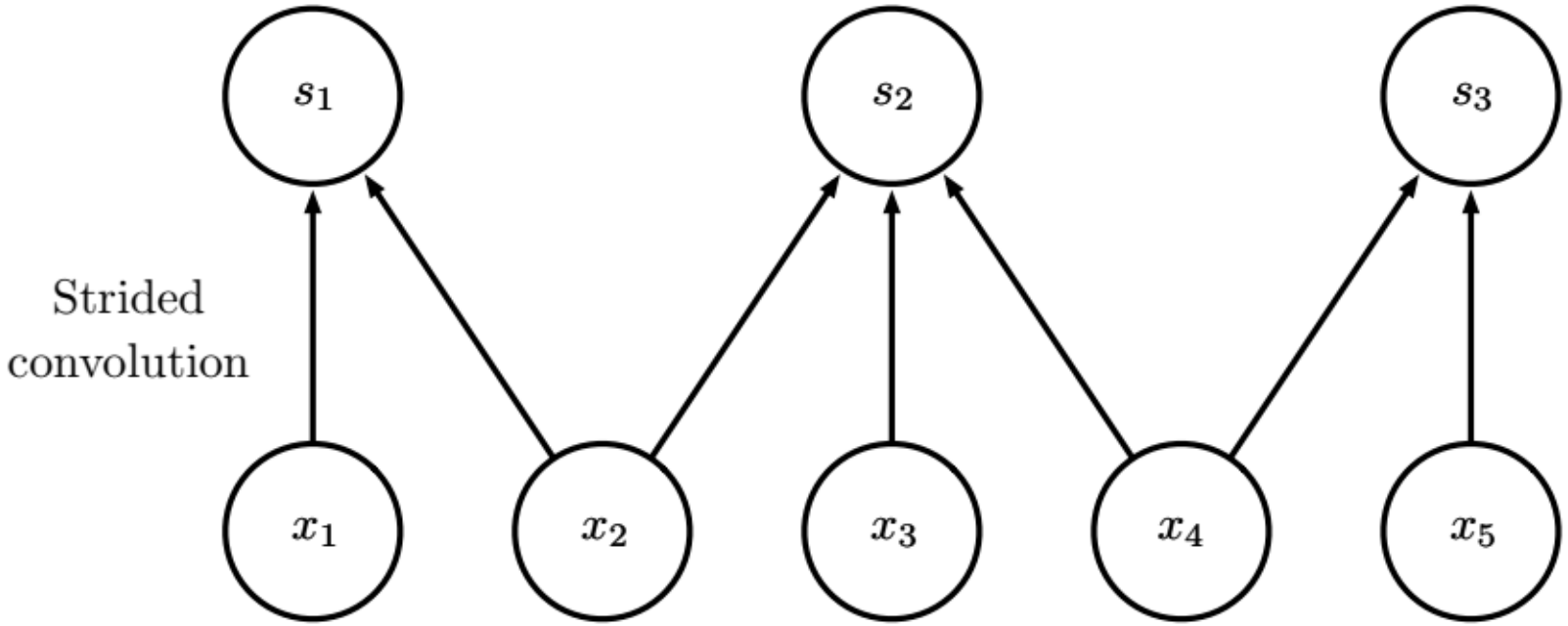
Dense Network

The dense counterpart would look like this



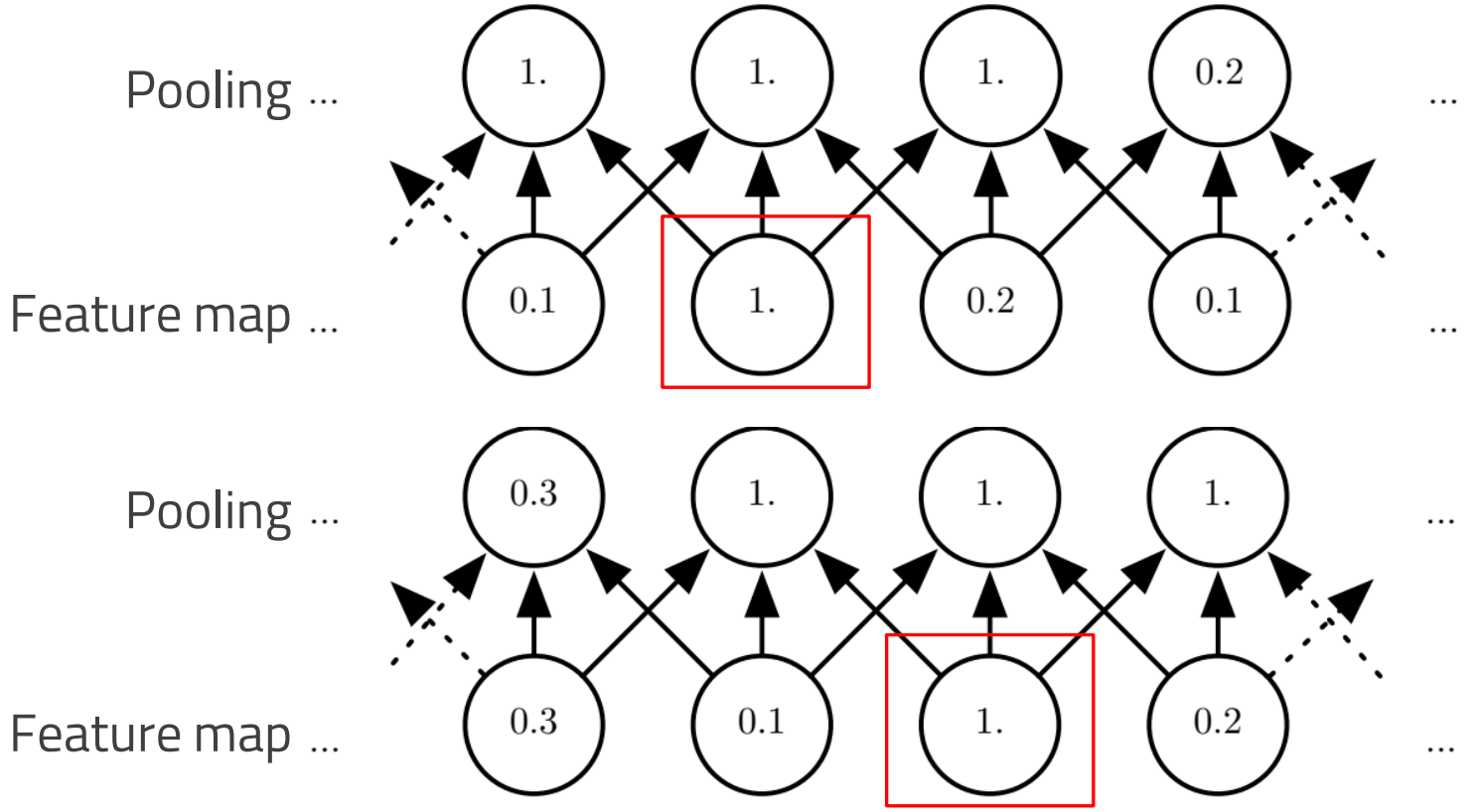
Strided Convolution

Make connectivity sparser

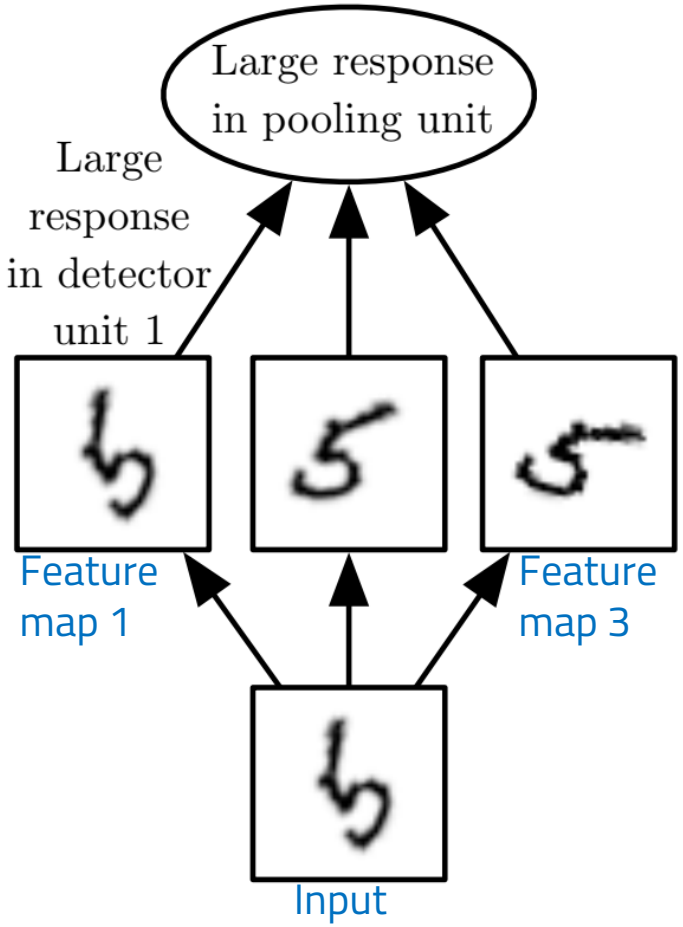


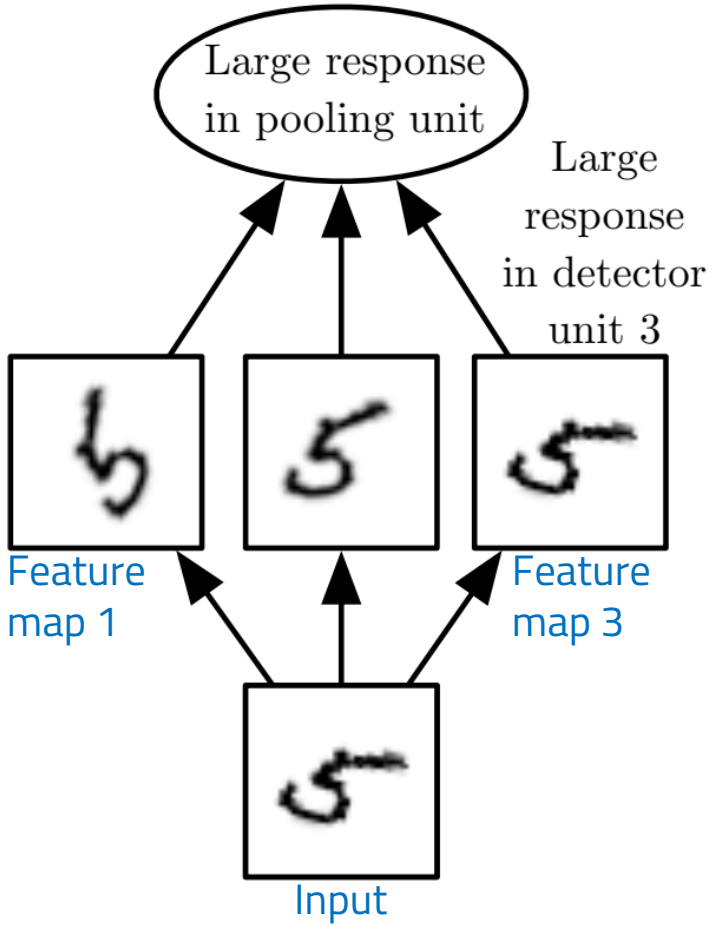
Max-Pooling and Spatial Invariance

A feature is detected even if it is spatially translated



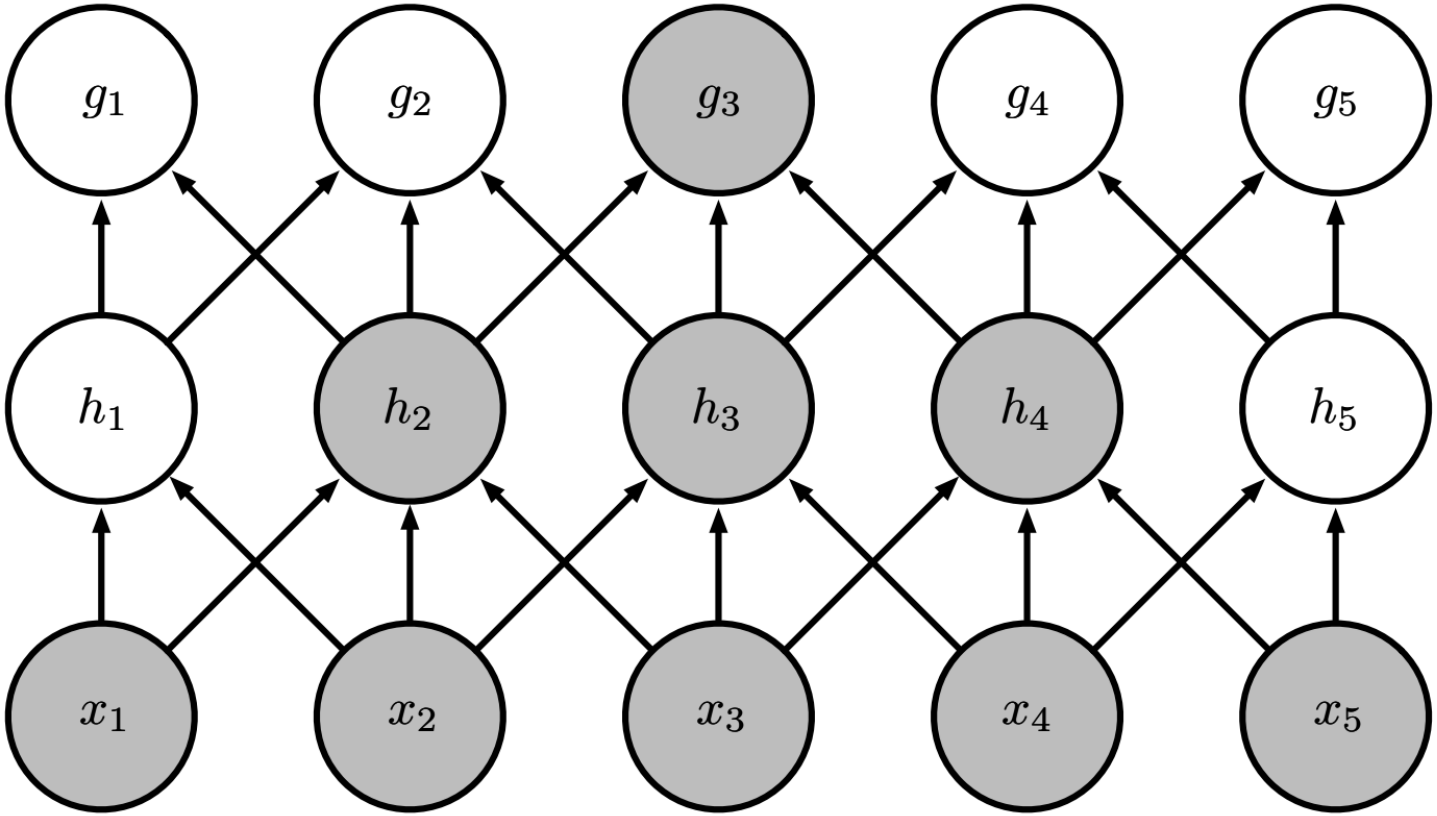
Cross Channel Pooling and Spatial Invariance



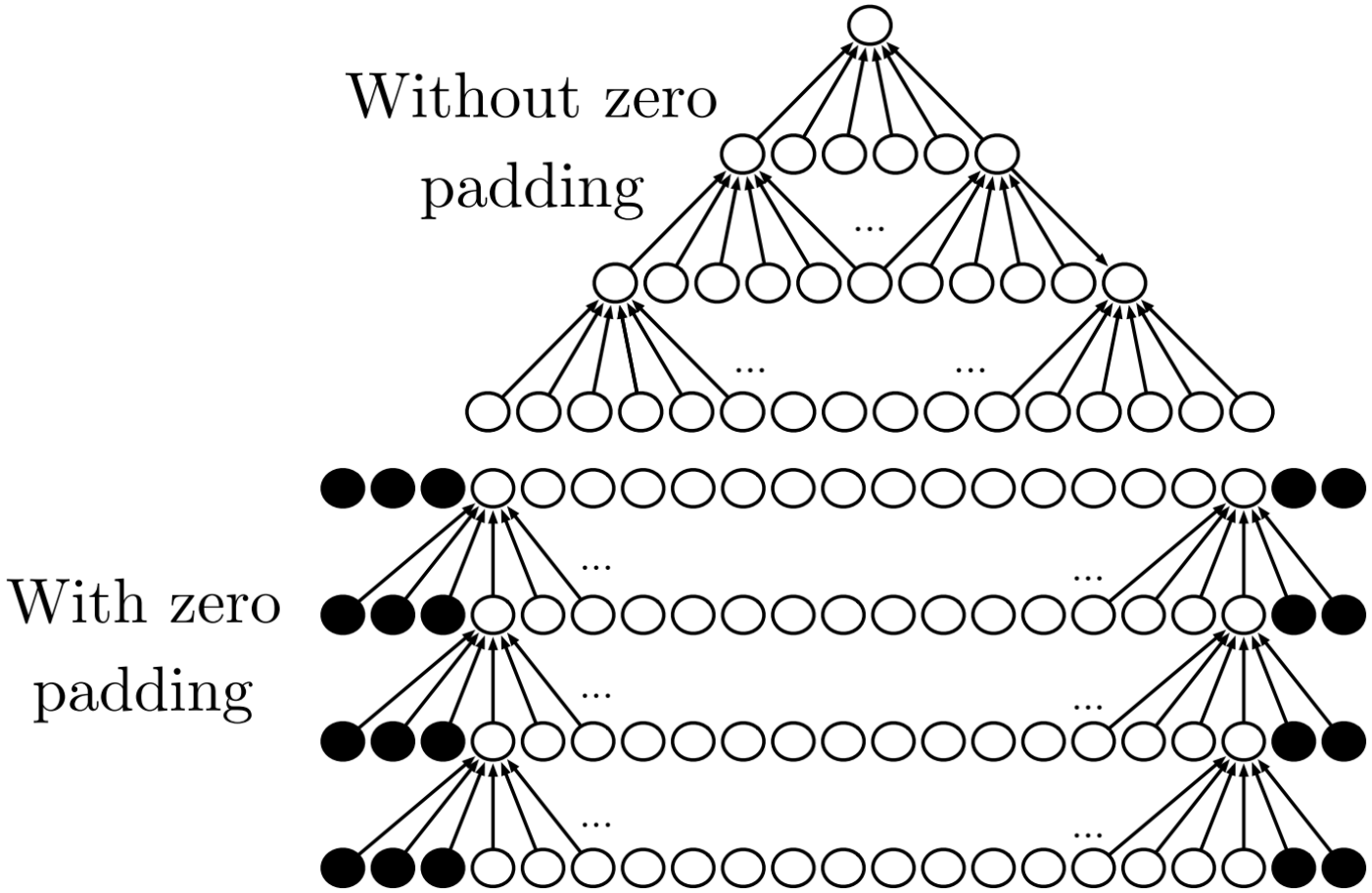


Hierarchical Feature Organization

The deeper the larger the receptive field of a unit



Zero-Padding Effect



Assuming no pooling

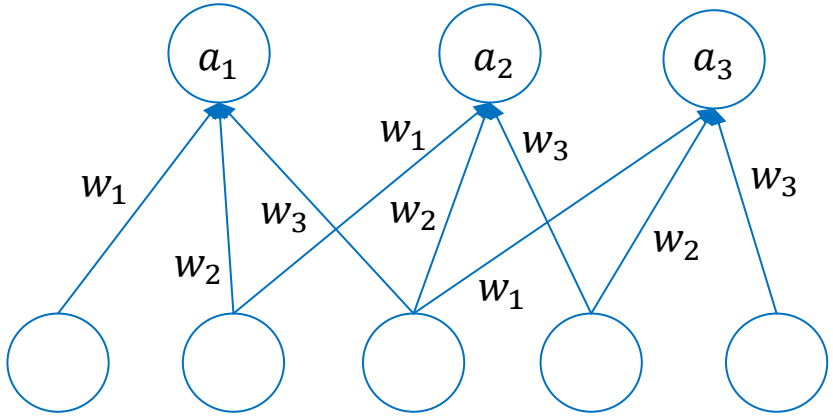
Training CNNs

Lecture II

Università di Pisa

CNN Training

Variants of the standard backpropagation that account for the fact that connections share weights (convolution parameters)

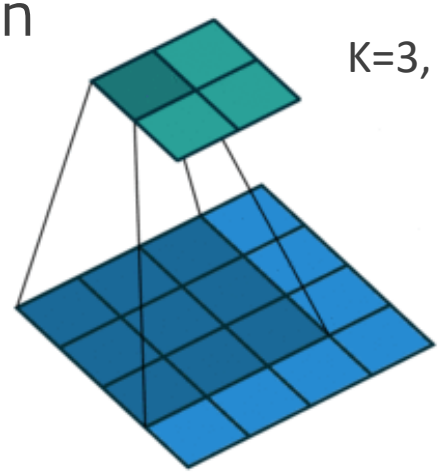


The gradient Δw_i is obtained by summing the contributions from all connections sharing the weight

Backpropagating gradients from convolutional layer N to N-1 is not as simple as transposing the weight matrix (need deconvolution with zero padding)

Backpropagating on Convolution

Convolution



K=3, S=1

Input is a 4x4 image
Output is a 2x2 image

Backpropagation step requires going back from the 2x2 to the 4x4 representation

Can write convolution as dense multiplication with shared weights

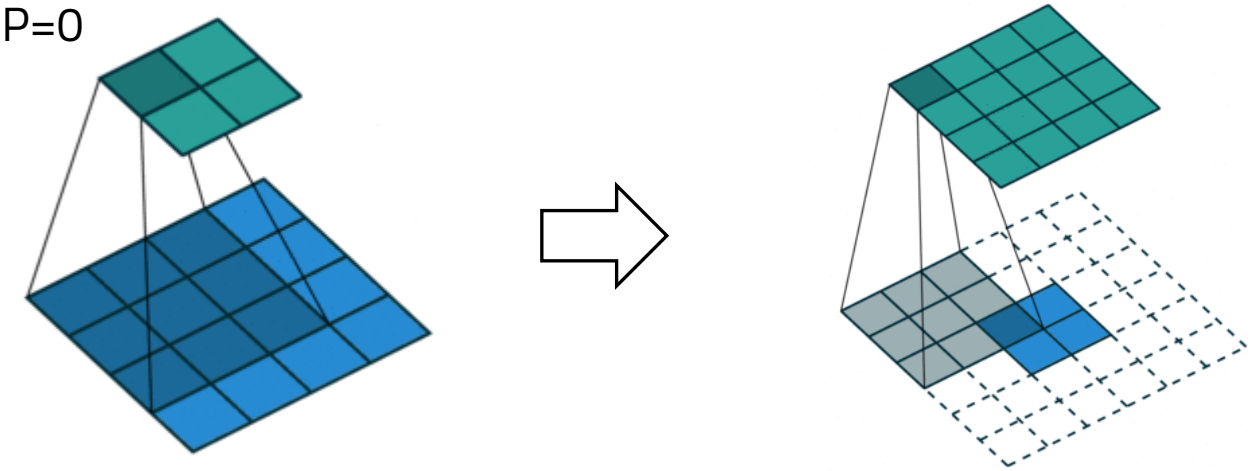
$$\begin{pmatrix} w_{0,0} & w_{0,1} & w_{0,2} & 0 & w_{1,0} & w_{1,1} & w_{1,2} & 0 & w_{2,0} & w_{2,1} & w_{2,2} & 0 & 0 & 0 & 0 & 0 \\ 0 & w_{0,0} & w_{0,1} & w_{0,2} & 0 & w_{1,0} & w_{1,1} & w_{1,2} & 0 & w_{2,0} & w_{2,1} & w_{2,2} & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & w_{0,0} & w_{0,1} & w_{0,2} & 0 & w_{1,0} & w_{1,1} & w_{1,2} & 0 & w_{2,0} & w_{2,1} & w_{2,2} & 0 \\ 0 & 0 & 0 & 0 & 0 & w_{0,0} & w_{0,1} & w_{0,2} & 0 & w_{1,0} & w_{1,1} & w_{1,2} & 0 & w_{2,0} & w_{2,1} & w_{2,2} \end{pmatrix}$$

Backpropagation is performed by multiplying the 4x1 representation to the transpose of this matrix

Deconvolution (Transposed Convolution)

We can obtain the transposed convolution using the same logic of the forward convolution

K=3, S=1, P=0

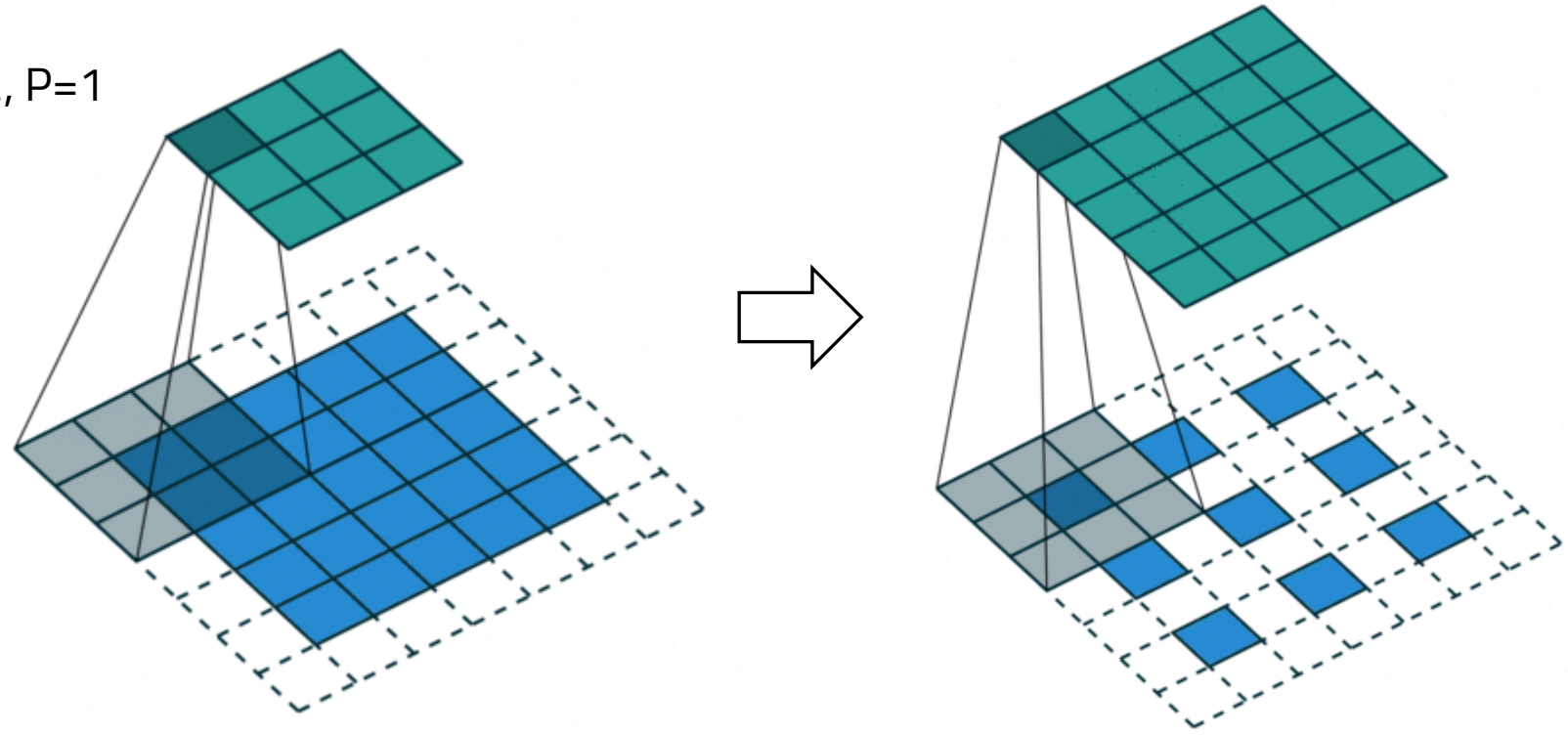


If you had no padding in the forward convolution, you need to pad much when performing transposed convolution

Deconvolution (Transposed Convolution)

If you have striding, you need to fill in the convolution map with zeroes to obtain a correctly sized deconvolution

K=3, S=2, P=1



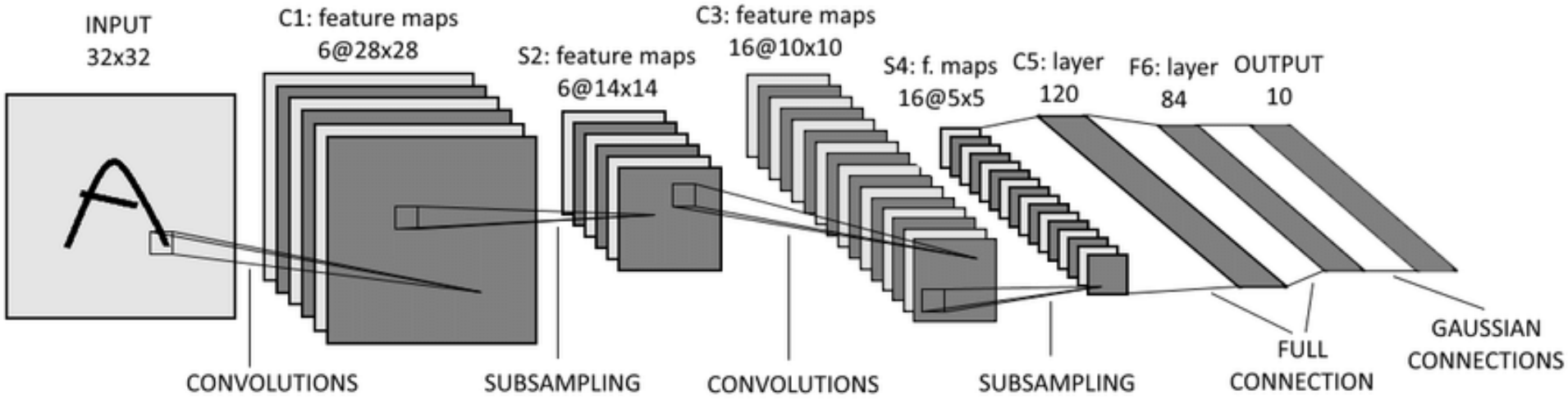
https://github.com/vdumoulin/conv_arithmetic

Notable Architectures

Lecture II

Università di Pisa

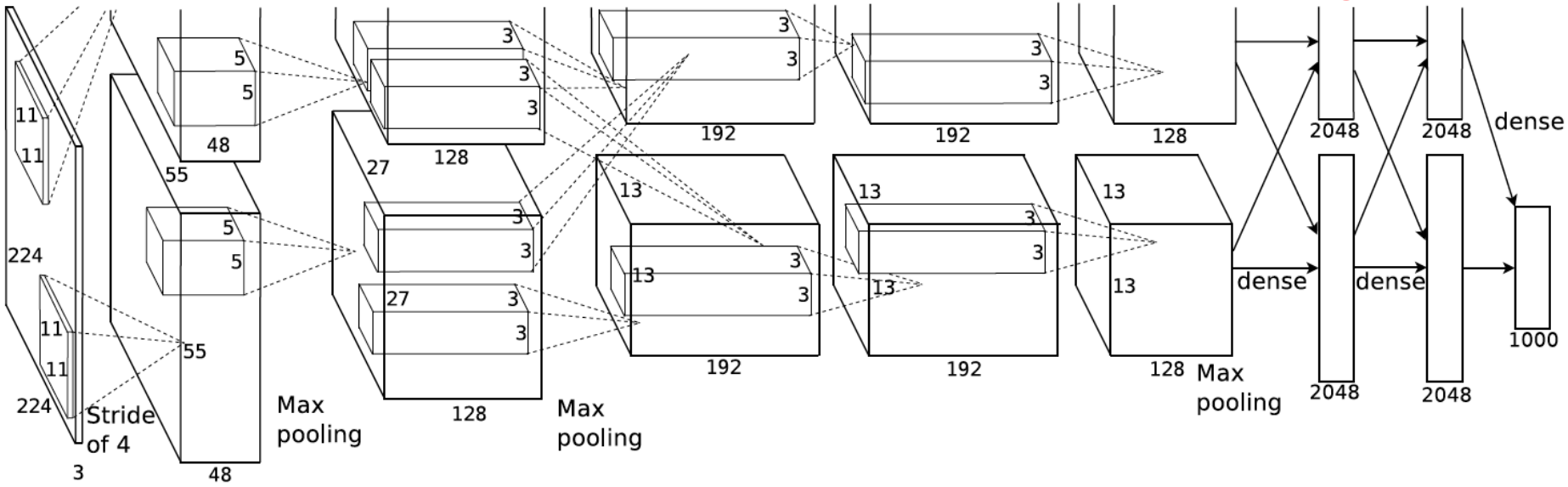
LeNet-5 (1989)



- Grayscale images
- Filters are 5x5 with stride 1 (sigmoid nonlinearity)
- Pooling is 2x2 with stride 2
- No zero padding

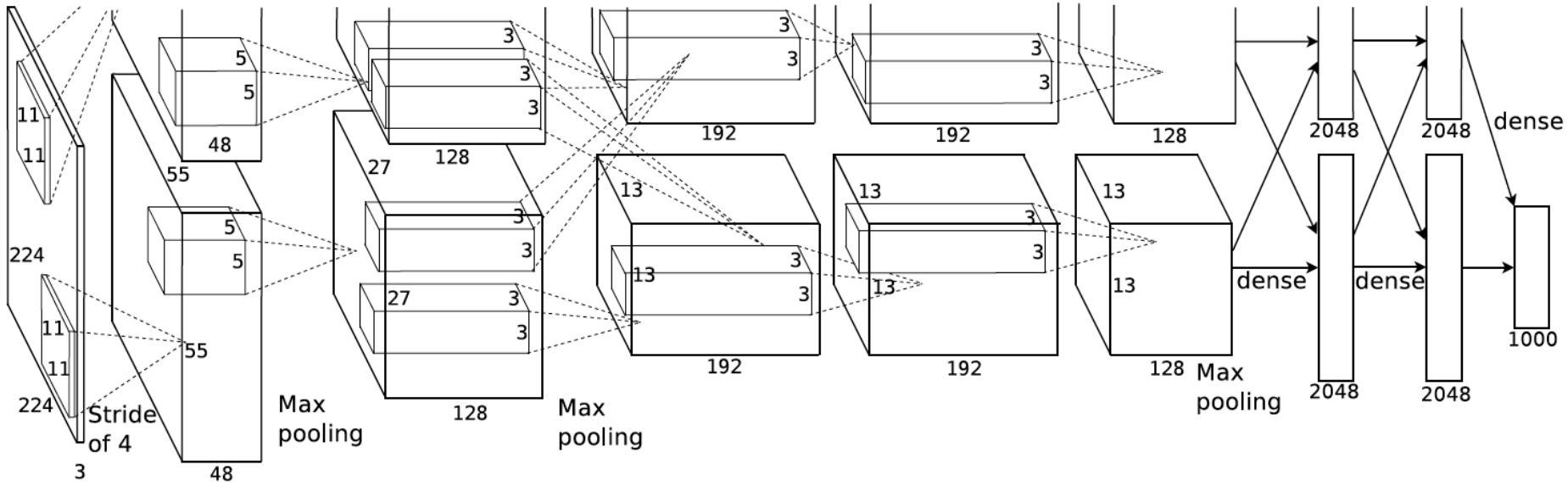
AlexNet (2012) - Architecture

ImageNet Top-5 : 15.4%



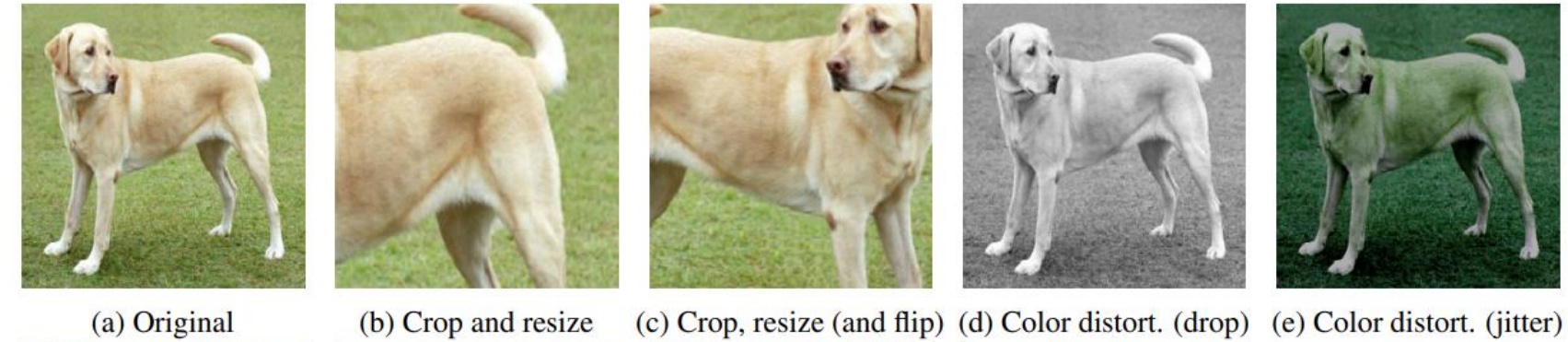
- RGB images 227x227x3
- 5 convolutional layers + 3 fully connected layers
- Split into two parts (top/bottom) each on 1 GPU

AlexNet - Innovations



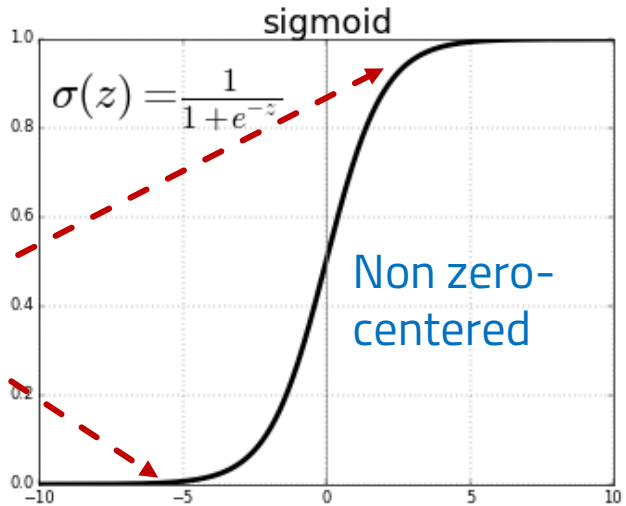
- Use heavy data augmentation (rotations, random crops, etc.)
- Introduced the use of ReLu
- Dense layers regularized by dropout

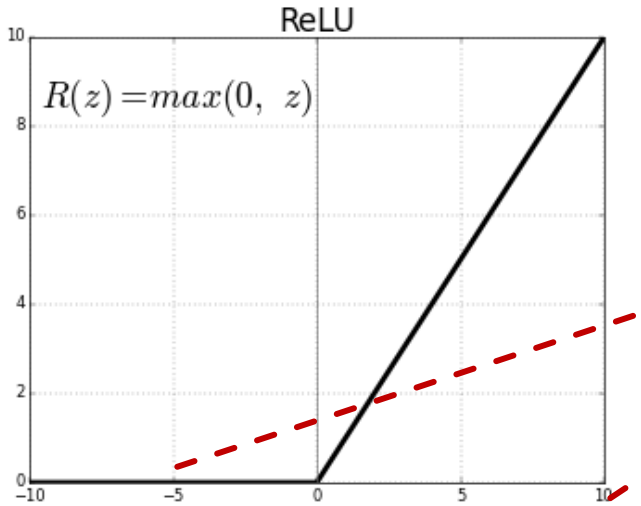
Data Augmentation



Key intuition - If I have an image with a given label, I can transform it (by flipping, rotation, etc) and the resulting image will still have the same label

ReLU Nonlinearity





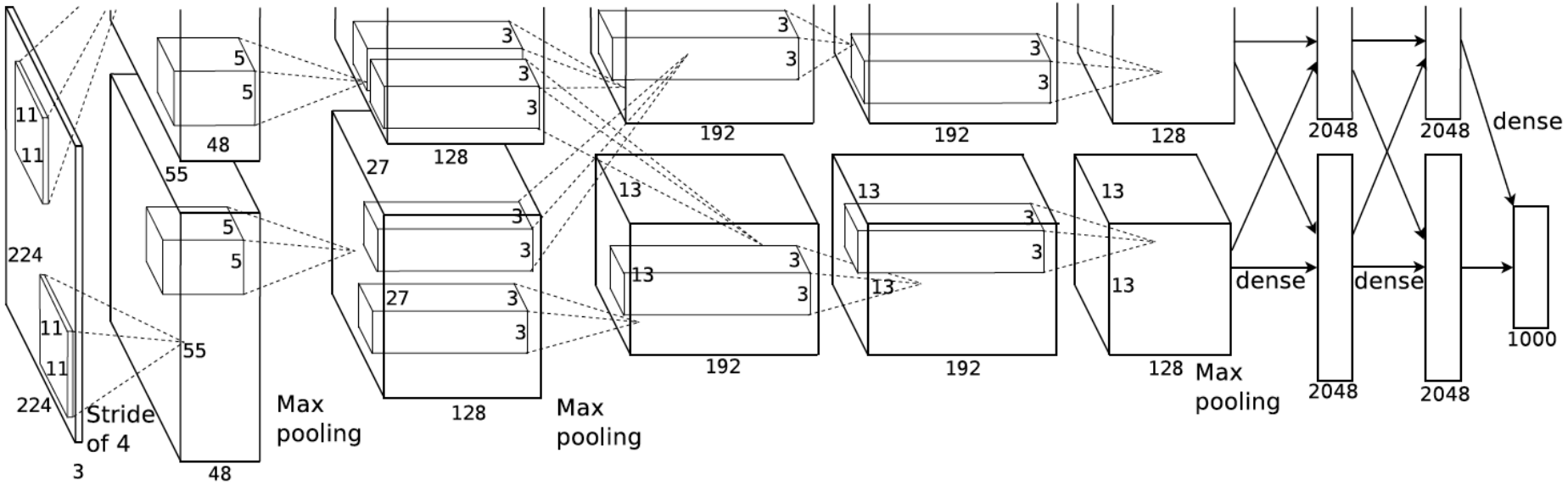
Saturation

Non zero-centered

Dead Units!!!

- ReLu help counteract gradient vanish
  - Sigmod first derivative vanishes as we increase or decrease z
  - ReLu first derivative is 1 when unit is active and 0 elsewhere
  - ReLu second derivative is 0 (no second order effects)
- Easy to compute (zero thresholding)
- Favors sparsity

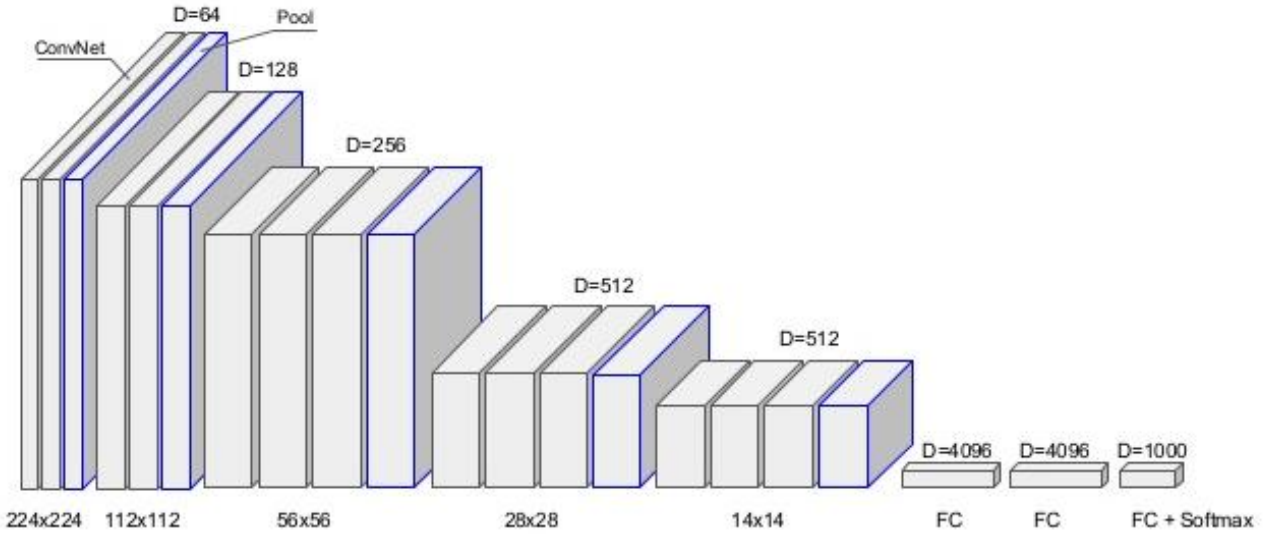
AlexNet - Parameters



- 62.3 millions of parameters (6% in convolutions)
- 5-6 days to train on two GTX 580 GPUs (95% time in convolutions)

VGGNet – VGG16 (2014)

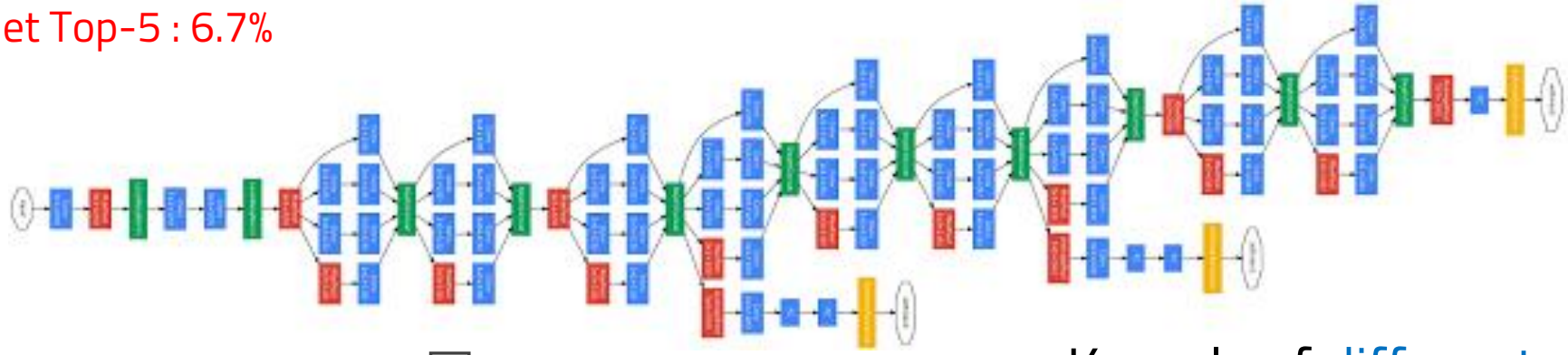
ImageNet Top-5 : 7.3%



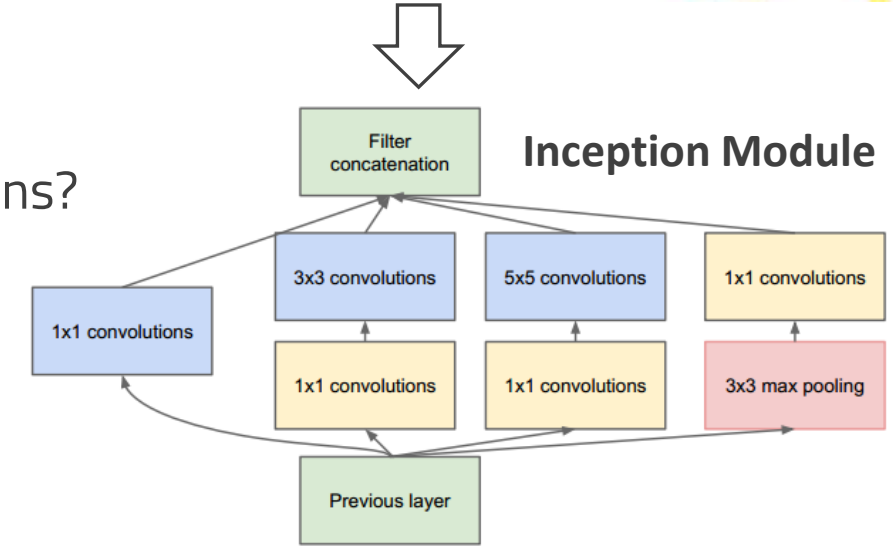
- Standardized convolutional layer
  - 3x3 convolutions with stride 1
  - 2x2 max pooling with stride 2 (not after every convolution)
- Various configuration analysed, but best has
  - 16 Convolutional + 3 Fully Connected layers
  - About 140 millions parameters (85% in FC)

GoogLeNet (2015)

ImageNet Top-5 : 6.7%

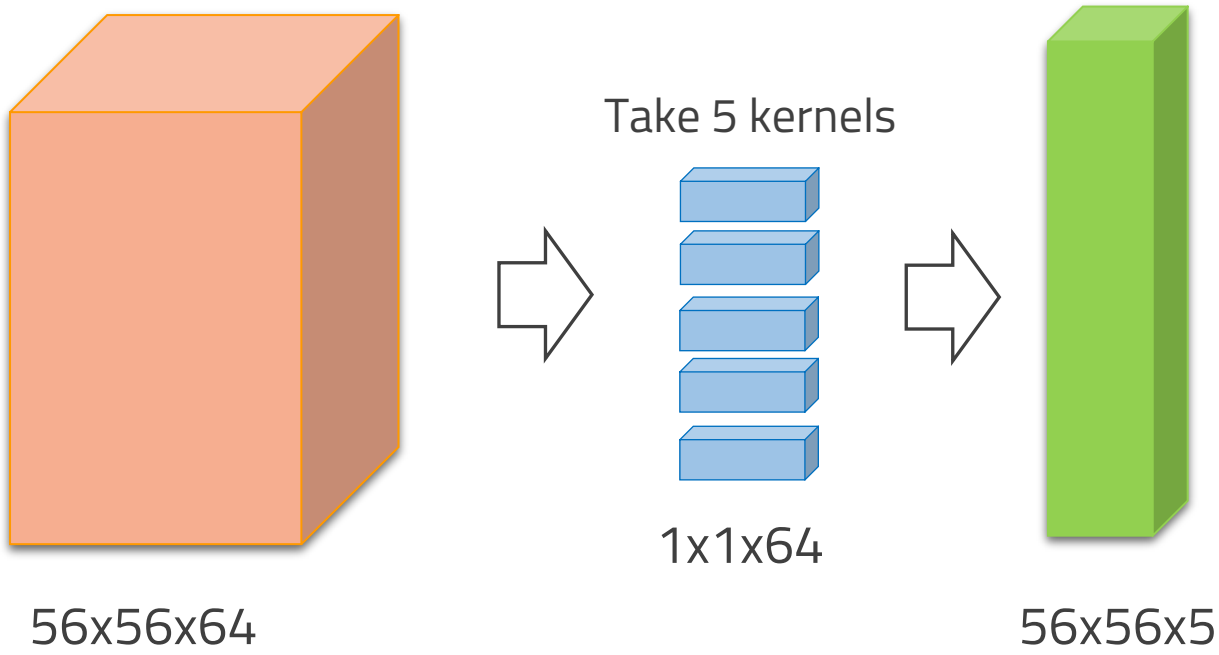


Why 1x1 convolutions?



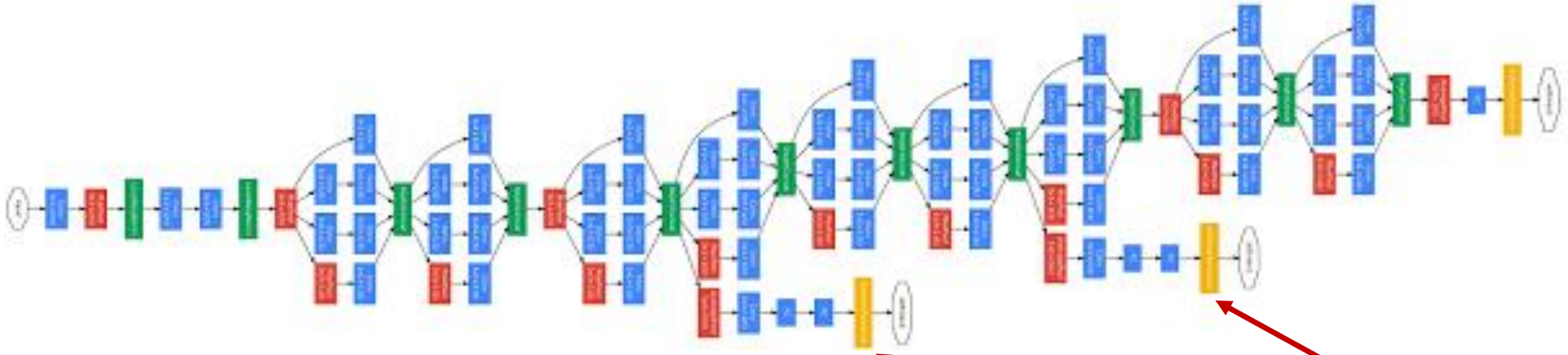
- Kernels of different size to capture details at varied scale
- Aggregated before sending to next layer
- Average pooling
- No fully connected layers

1x1 Convolutions are Helpful



By placing 1x1 convolutions before larger kernels in the Inception module, the number of input channels is reduced, saving computations and parameters

Back on GoogLeNet



Auxiliary outputs to inject gradients at deeper layers

- Only 5 millions of parameters
- 12X less parameters than AlexNet
- Followed by v2, v3, v4 … of the Inception module
  - More filter factorization
  - Introduce heavy use of Batch Normalization

Batch Normalization

- Very deep neural network are subject to internal covariate shift
  - Distribution of inputs to a layer N might vary (shift) with different minibatches (due to adjustments of layer N-1)
  - Layer N can get confused by this
  - Solution is to normalize for mean and variance in each minibatch (bit more articulated than this actually)

Batch Normalization (BN)

For a given mini-batch B, BN normalizes activations:

1) Compute mean and variance across the batch

$$\mu_B^l = \frac{1}{N_B}\sum_{i=1}^{N_B} h_i^l \quad \text{and} \quad \sigma_B^l = \frac{1}{N_B}\sum_{i=1}^{N_B}\left(h_i^l - \mu_B^l\right)^2$$

Activation of one neuron in layer l for minibatch sample i

2) Normalize activations

$$\widehat{h_i^l} = \frac{h_i^l - \mu_B^l}{\sqrt{\left(\sigma_B^l\right)^2 + \epsilon}}$$

Need to backpropagate through these

3) Scale and shift through learnable parameters (γ, β)

$$\widehat{h_i^{l\prime}} = \gamma\widehat{h_i^l} + \beta$$

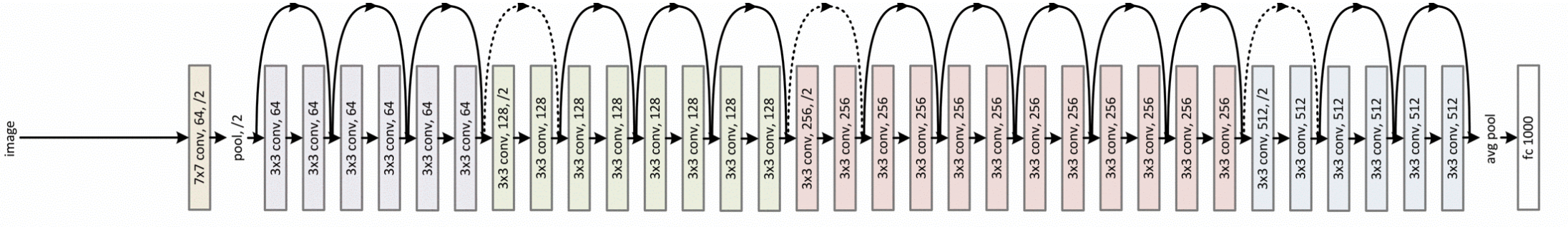
Trainable linear transform potentially allowing to cancel unwanted zero-centering effects (e.g. sigmoid)

Becomes the input to layer $l+1$

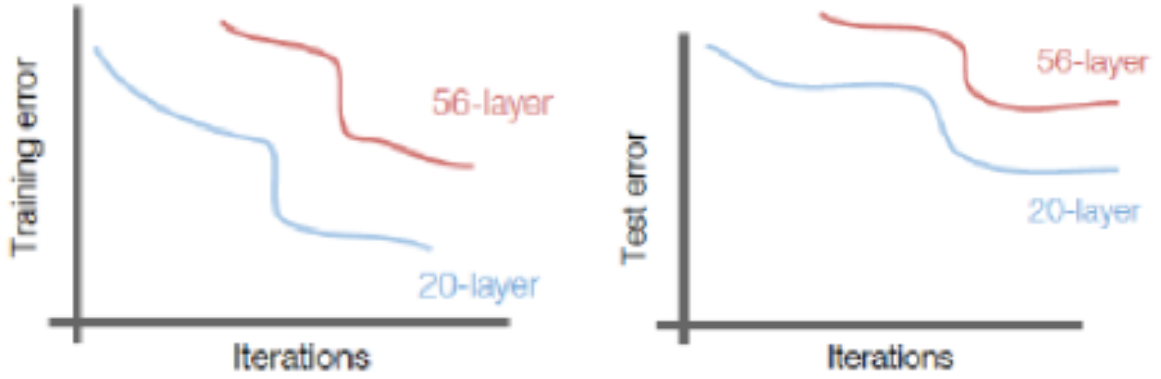
ResNet (2015)

Begin of the Ultra-Deep Network Era (152 Layers)

ImageNet Top-5 : 3.57%

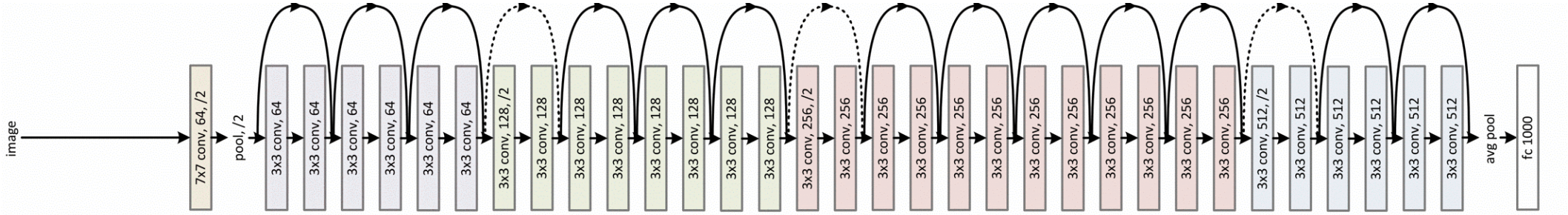


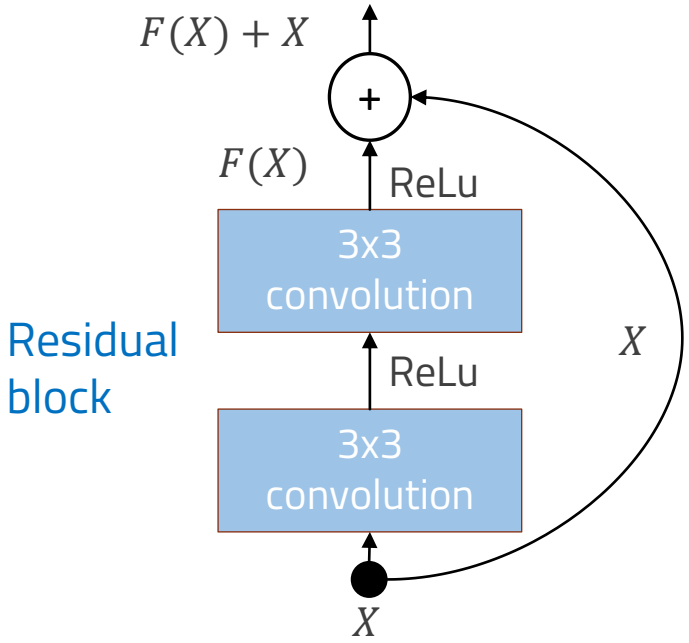
Why wasn't this working before?



Gradient vanishes when backpropagating too deep!

ResNet Trick



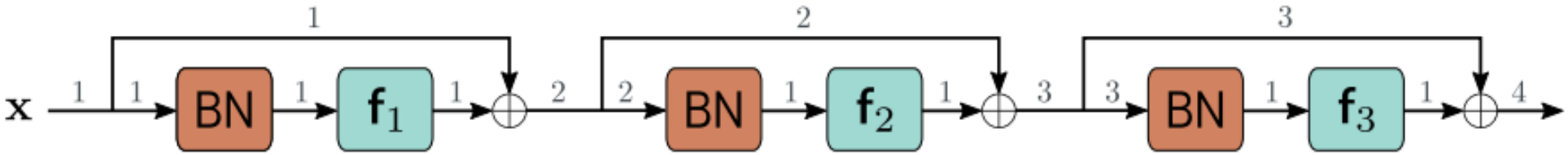


Residual block

The input to the block X bypasses the convolution and is then combined with its residual $F(X)$ resulting from the convolutions

When backpropagating the gradient flows in full through these bypass connections (more on this in the upcoming lectures)

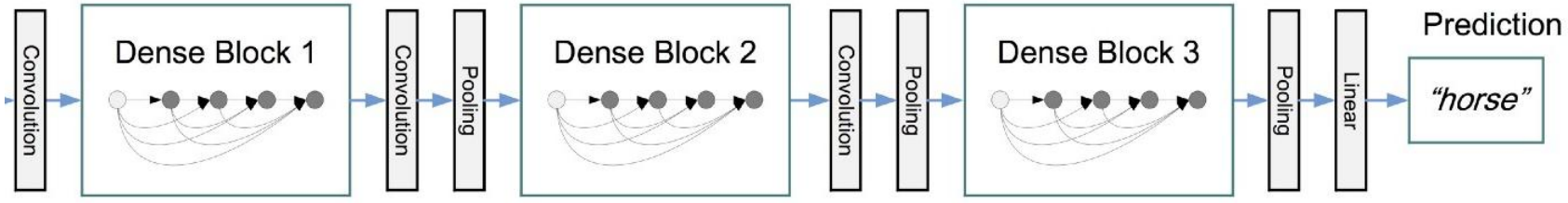
ResNet & Batch Norm

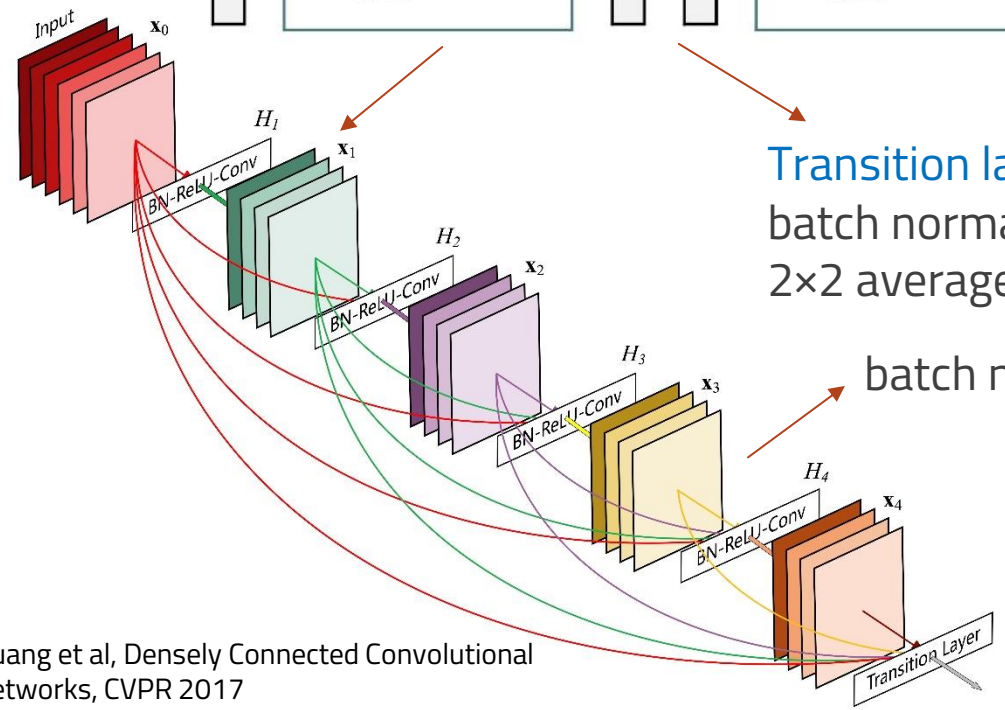


When connecting several Residual Blocks in series, one need to be careful about amplification/compounding of variance due to the residual connectivity

- Batch norm can alleviate this effect

Dense CNN





Huang et al, Densely Connected Convolutional Networks, CVPR 2017

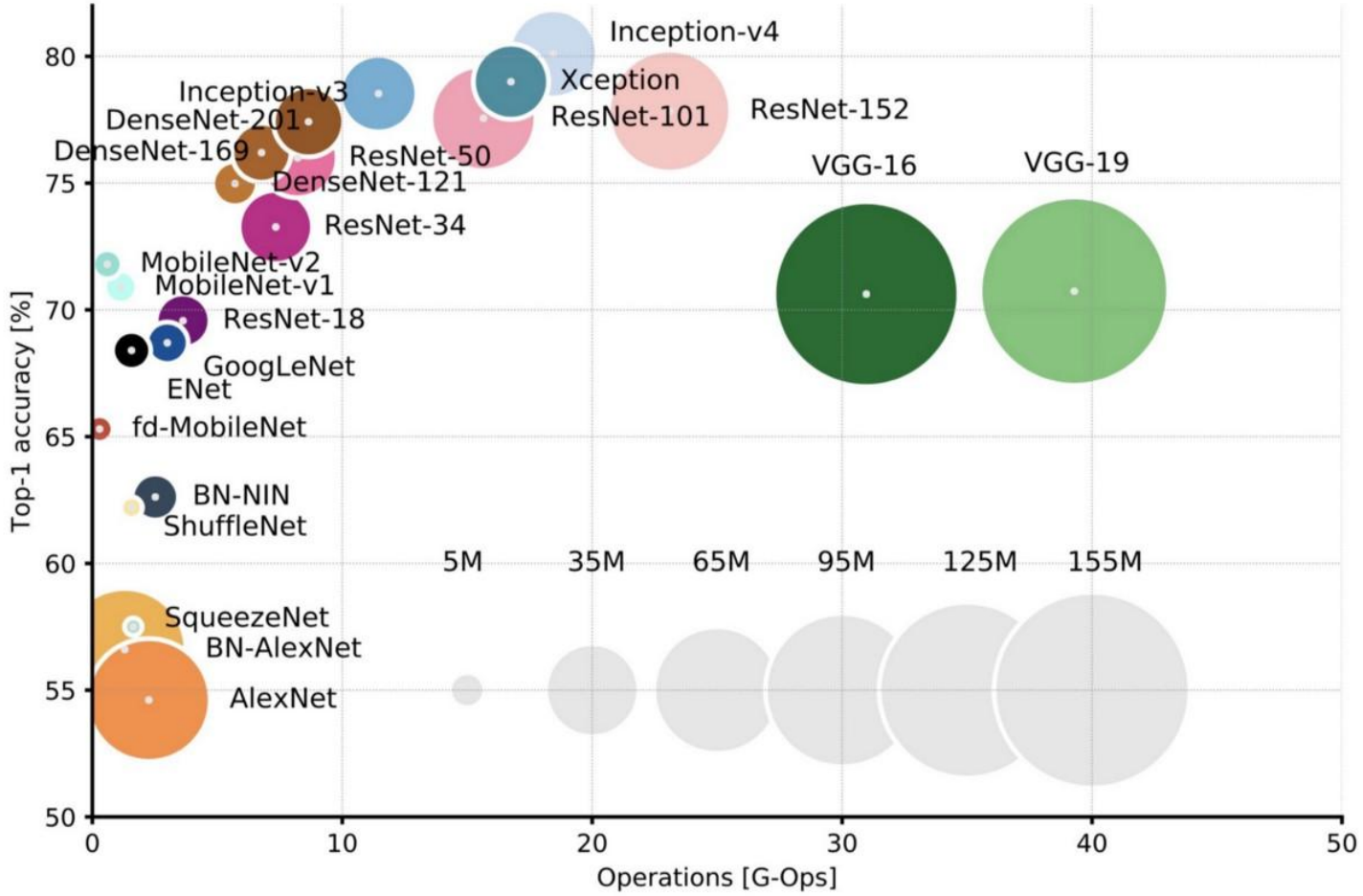
Transition layers

batch normalization + 1×1 convolutional + 2×2 average pooling layer

batch normalization + ReLU + 3x3 conv

- Gradient flows well in bypass connections
- Each layer in the dense block has access to all information from previous layers

CNN Architecture Evolution



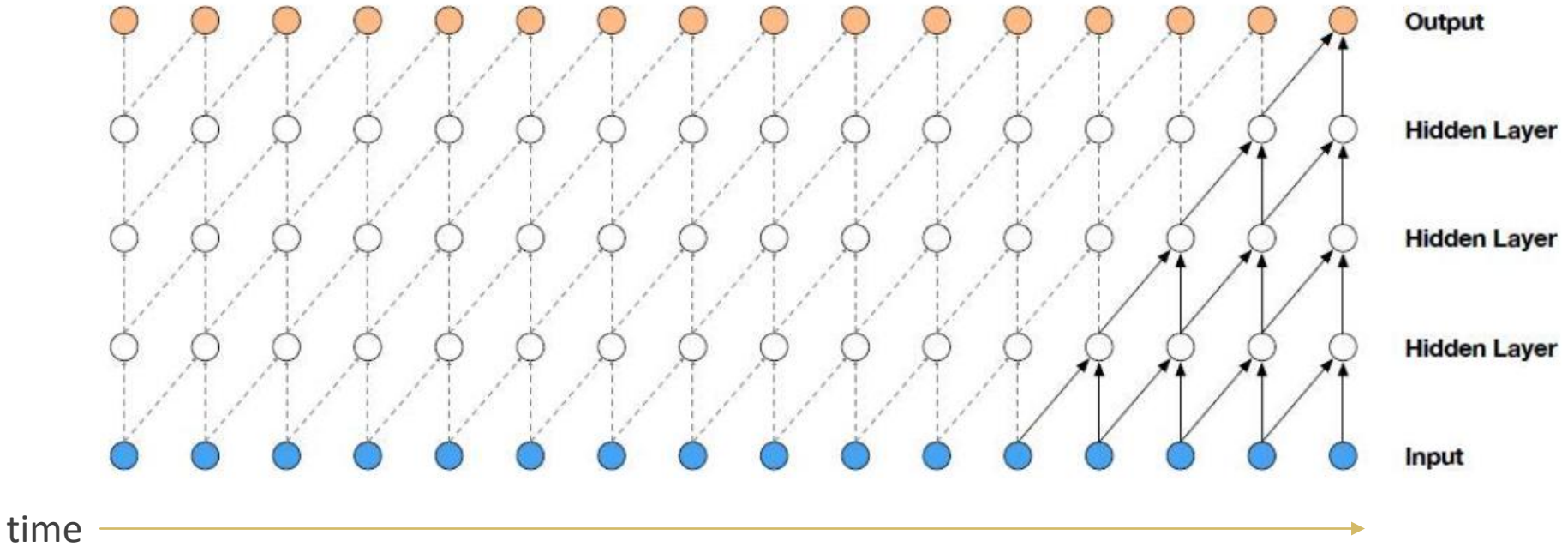
Advanced concepts

Lecture II

Università di Pisa

Causal Convolutions

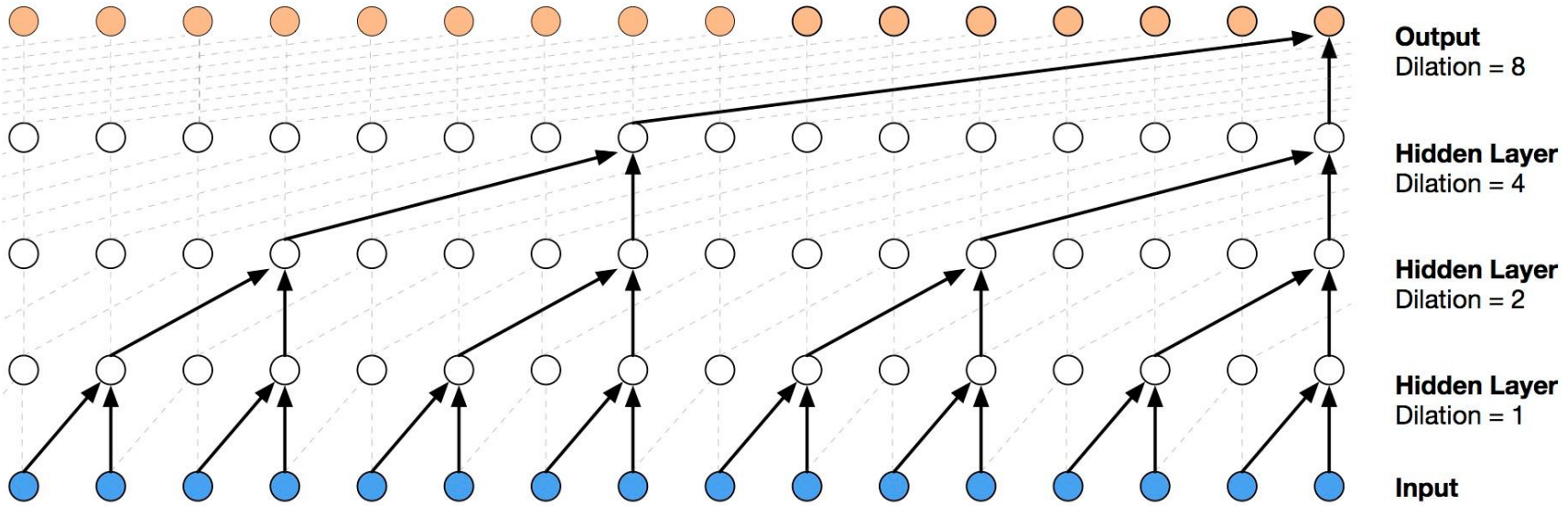
Preventing a convolution from allowing to see into the future…



Problem is the context size grows slow with depth

Causal & Dilated Convolutions

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



Similar to striding, but size is preserved

Oord et al, WaveNet: A Generative Model for Raw Audio, ICLR 2016

Vision Tasks

You will see plenty of these in the Computer Vision course

Image classification

- Identify a scene, the presence of a predominant object or visual concept
- Predict the identity of a person
- Architectures: InceptionNet, ResNet, DenseNet, …

Image regression

- Predict geometric/photometric information (size, distance, …)
- Predict biometric/biomedical markers
- Architectures: same as above

Object detection

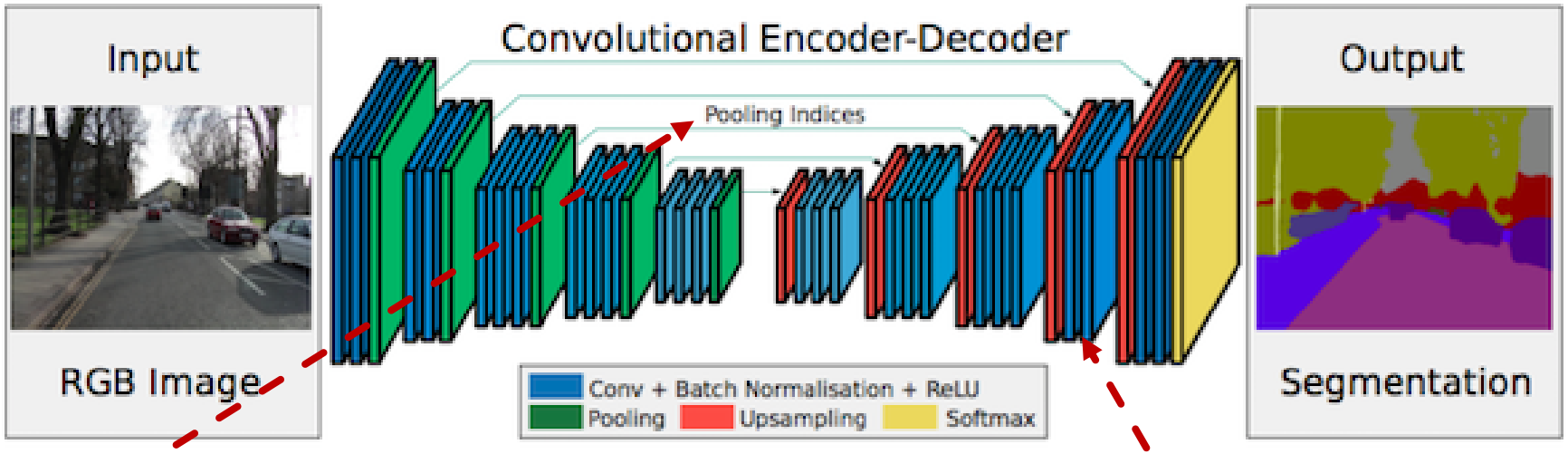
- Put bounding boxes around (multiple) objects and classify them

Semantic segmentation

- Delineate structures/objects at pixel level

We cannot really reuse previous architectures for these tasks

Semantic Segmentation - Deconvolution

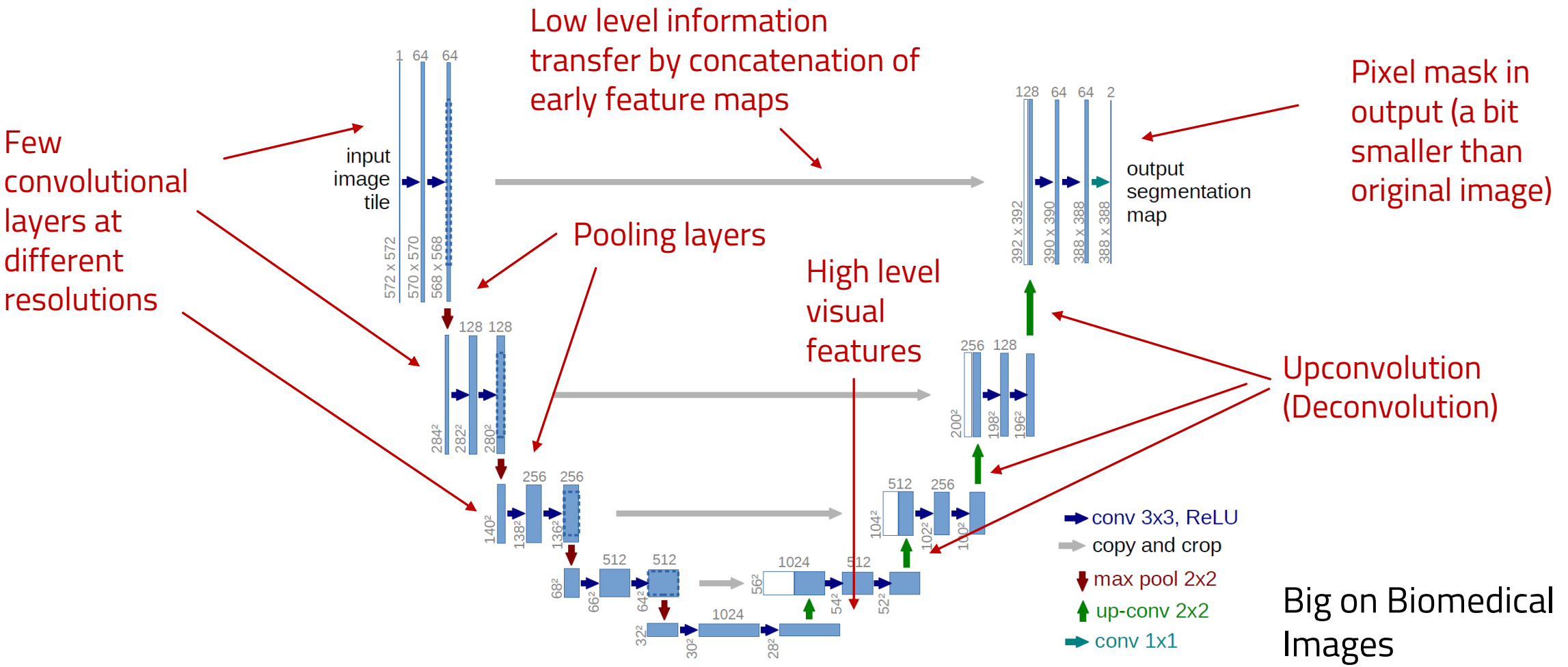


Maxpooling indices transferred to decoder to improve the segmentation resolution.

Deconvolutional layers to go back to image space

SegNet demo here:
http://mi.eng.cam.ac.uk/projects/segnet/

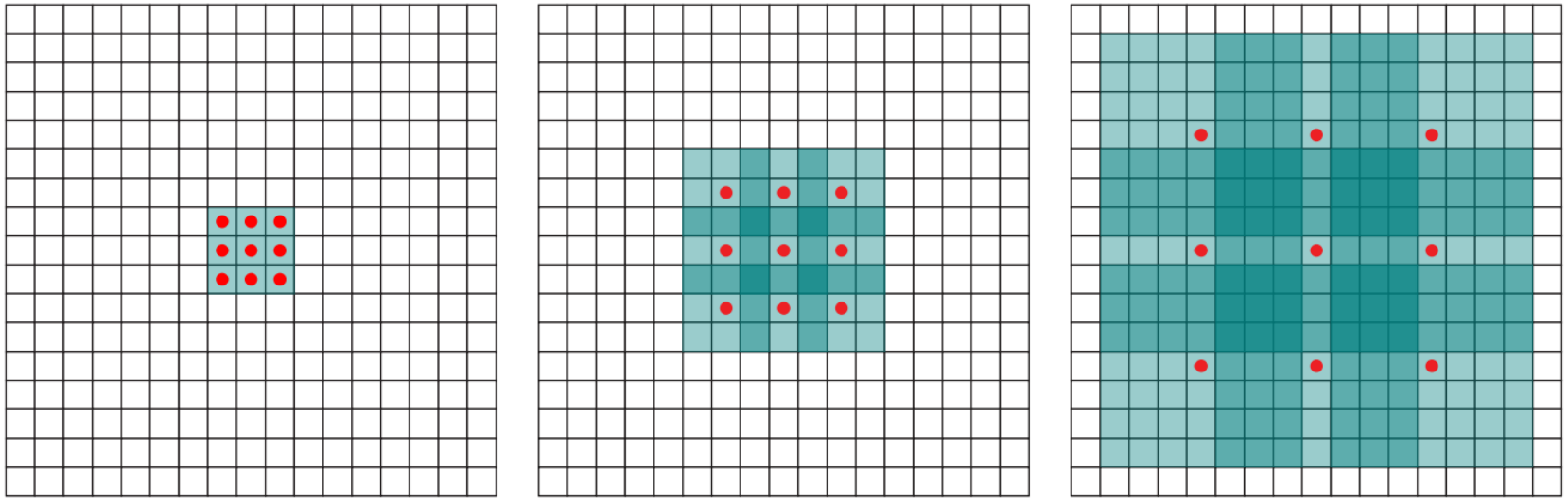
Semantic Segmentation - U-Nets



Big on Biomedical Images

Semantic Segmentation - Use Dilated Convolutions

Always perform 3x3 convolutions with no pooling at each level

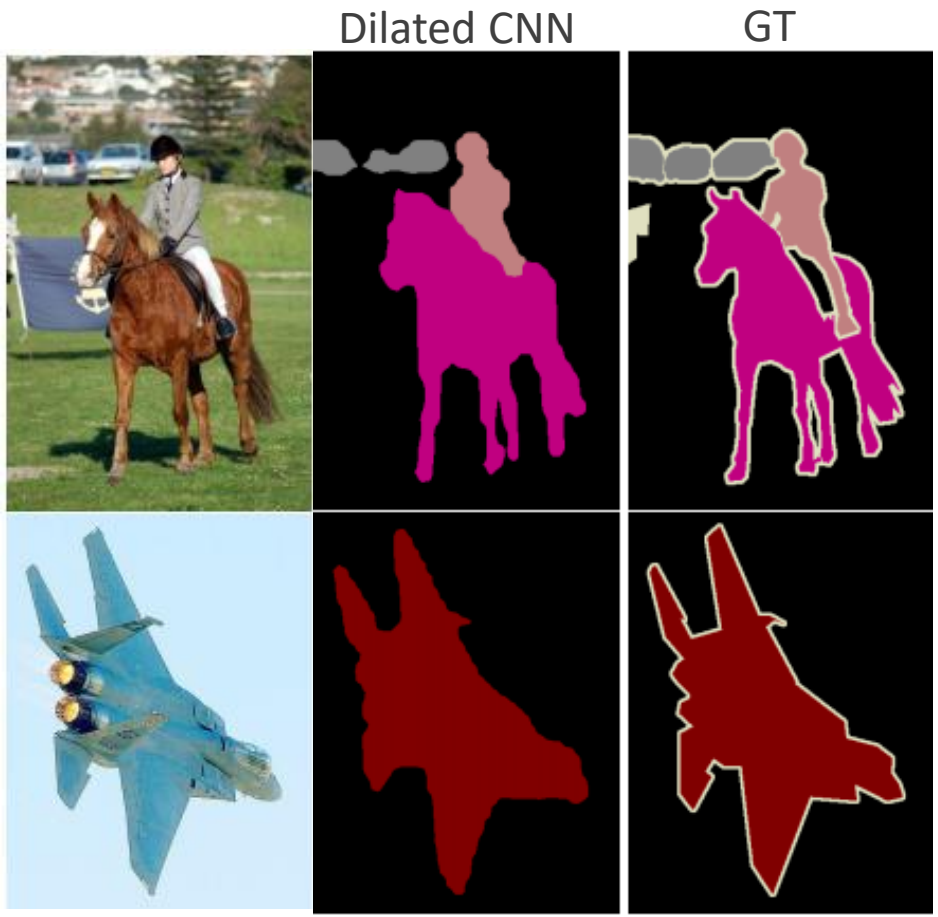


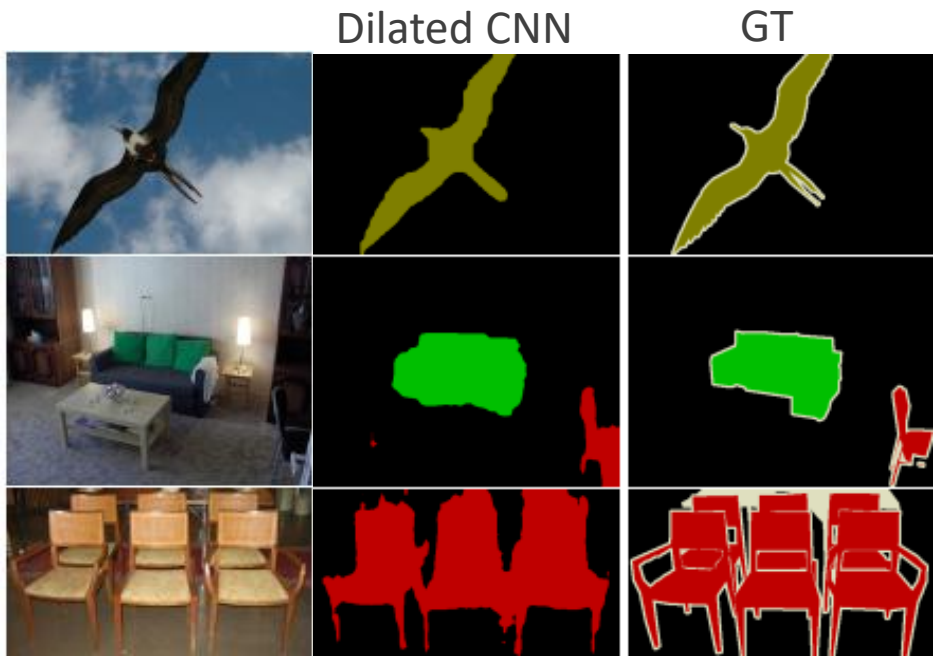
Context increases without

- Pooling (changes map size)
- Increasing computational complexity

Yu et al, Multi-Scale Context Aggregation by Dilated Convolutions, ICLR 2016

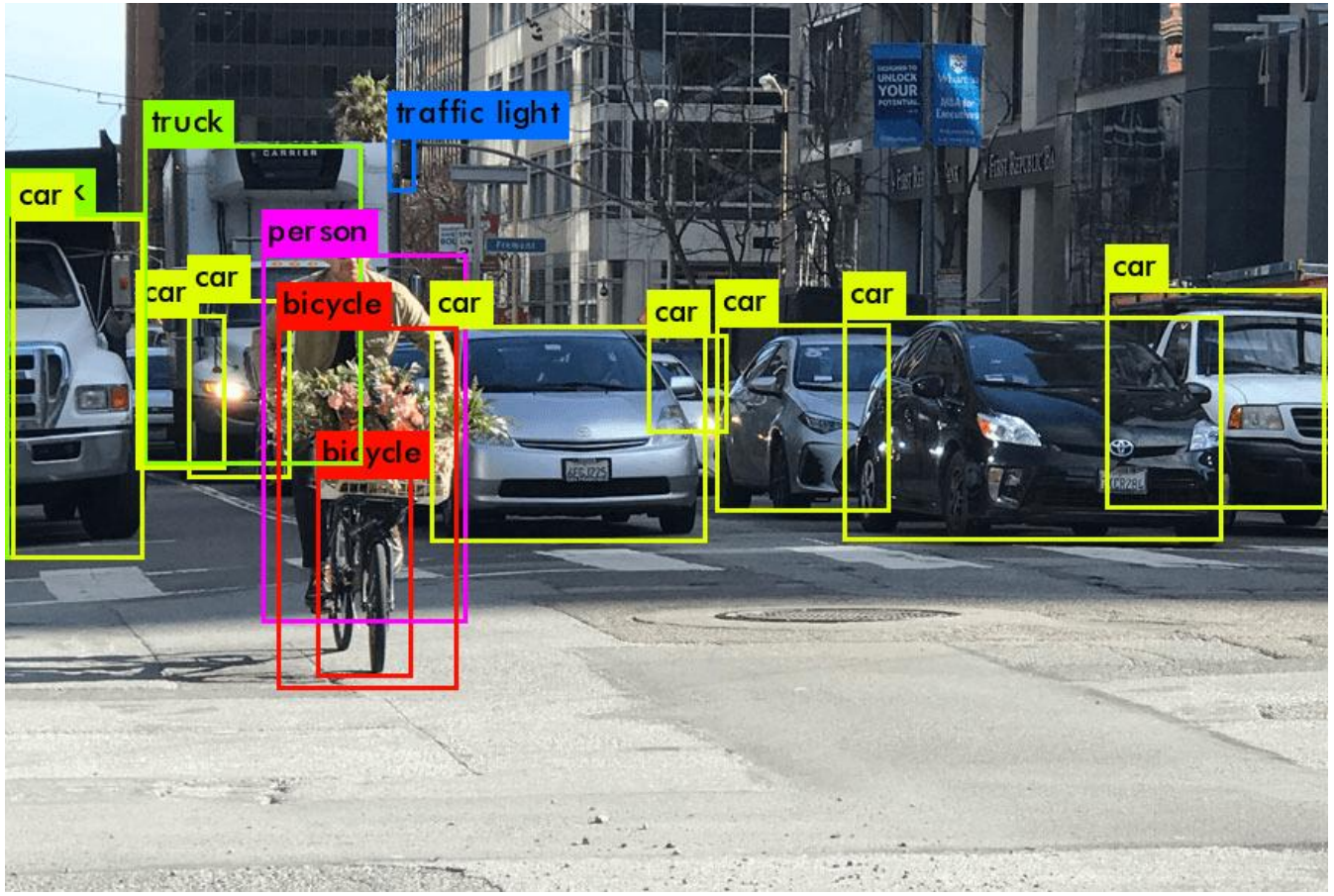
Segmentation by Dilated CNN



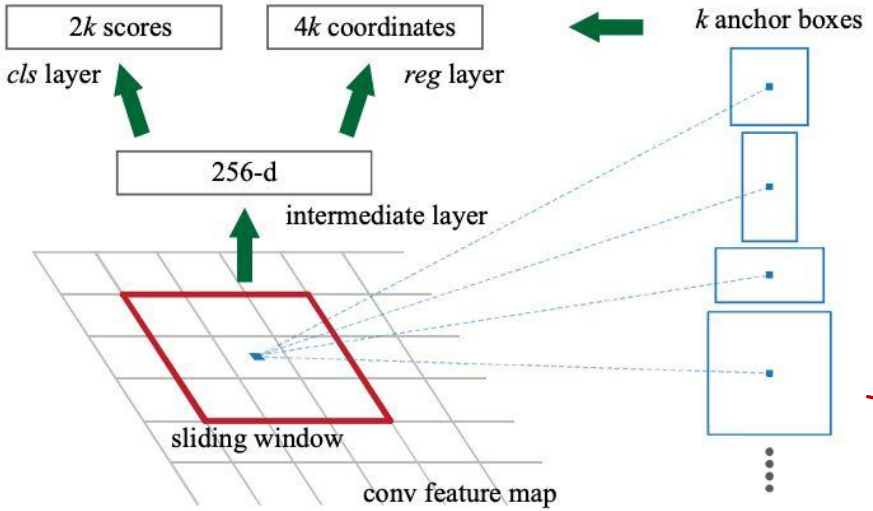


Yu et al, Multi-Scale Context Aggregation by Dilated Convolutions, ICLR 2016

Object Detection

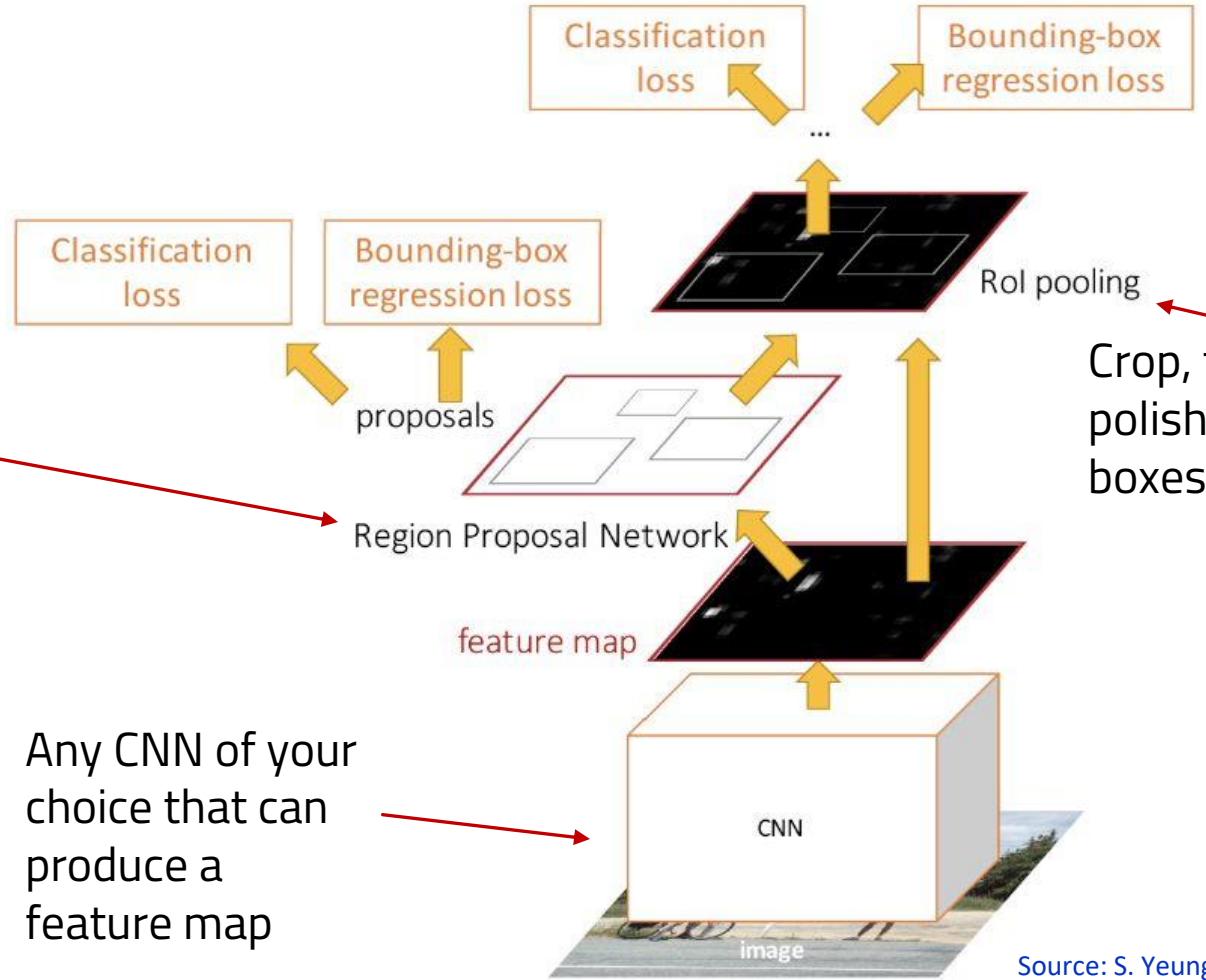


Object Detection: Faster R-CNN



Generate bounding boxes proposals

- x,y position
- size
- confidence



Crop, fuse and polish bounding boxes proposals

Any CNN of your choice that can produce a feature map

Source: S. Yeung, BIODS 220

Wrap-up

Take Home Messages

- Key aspects
  - Convolutions in place of dense multiplications allow sparse connectivity and weight sharing
  - Pooling enforces invariance and allows to change resolution but shrinks data size
  - Full connectivity compress information from all convolutions but accounts for 90% of model complexity
- Lessons learned
  - ReLU are efficient and counteract gradient vanish
  - 1x1 convolutions are useful
  - Need batch normalization
  - Residual connections allow to go deeper
- Dilated (à trous) convolutions are highly effective in item-by-item tasks

Next Lecture

Propagation issues in deep networks

- Understanding the dynamics of information propagation in deep learning models
- Focus on learning with sequential data
- Gradient issues in recurrent neural networks