

Convolutional Neural Networks

Davide Bacciu

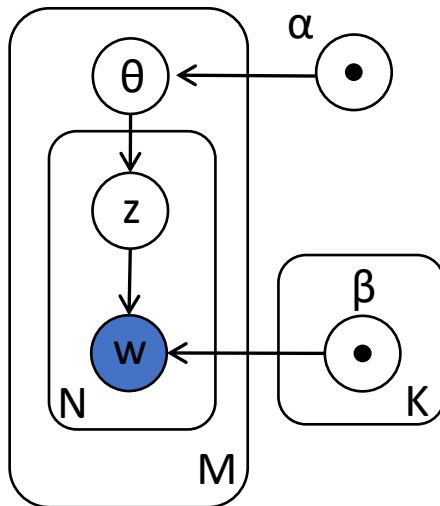
Dipartimento di Informatica
Università di Pisa

Intelligent Systems for Pattern Recognition (ISPR)



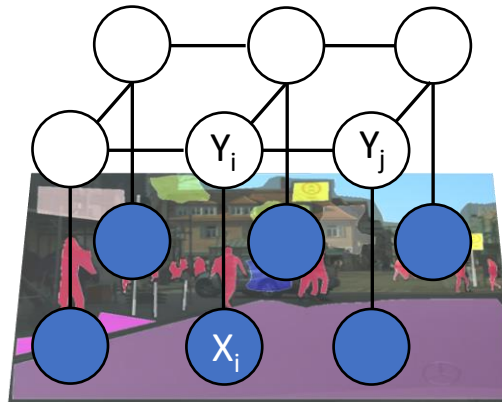
Generative Graphical Models

Bayesian Models/Nets



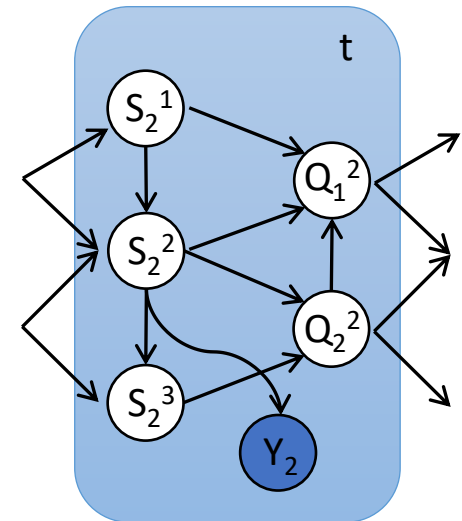
- Unsupervised data understanding
- Interpretability
- Weak on supervised performance

Markov Random Fields



- Knowledge and constraints through feature functions
- CRF: the supervised way to generative
- Computationally heavy

Dynamic Models

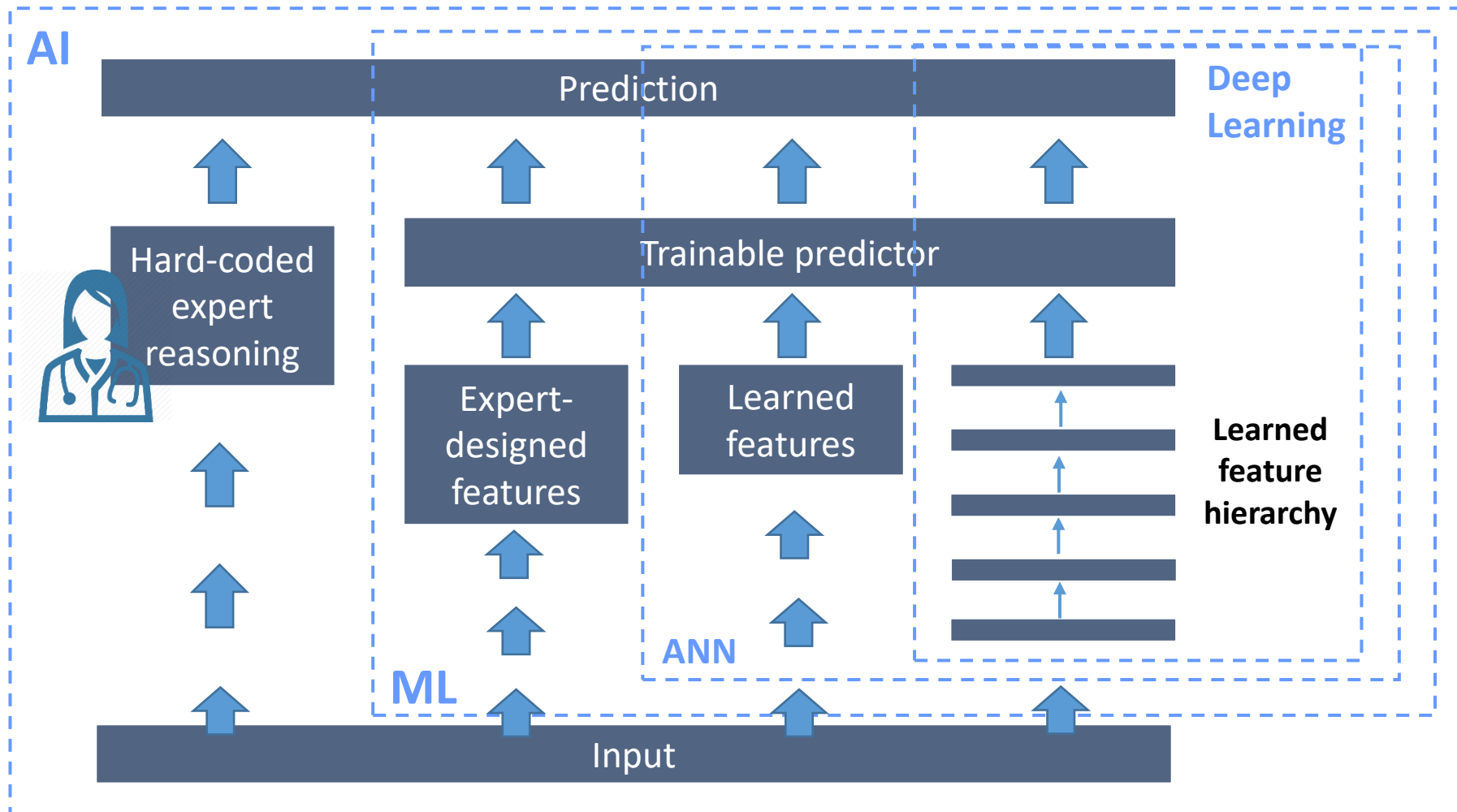


- Topology unfolds on data structure
- Structured data processing
- Complex causal relationships

Module's Take Home Messages

- Consider **using generative models** when
 - Need interpretability
 - Need to incorporate prior knowledge
 - Unsupervised learning or learning with partially observable supervision
 - Need reusable/portable learned knowledge
- Consider **avoiding generative models** when
 - Having tight computational constraints
 - Dealing with raw, noisy low-level data
- **Variational** inference and **sampling**
 - Efficient ways to learn an approximation to intractable distributions
- Neural networks can be **used as variational functions** or to **implement sampling** processes

Deep Learning

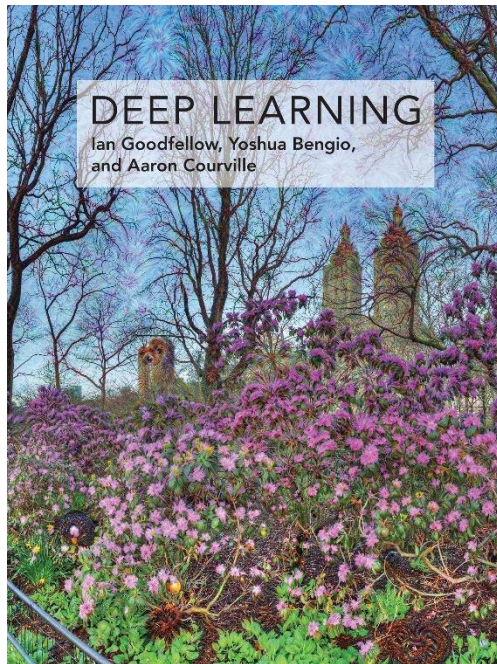


Module Outline

- Foundational models
 - Convolutional Neural Networks
 - Deep Autoencoders and RBM
 - Gated Recurrent Networks (LSTM, GRU, ...)
- Recursive and contextual (by A. Micheli)
 - Tree and graph processing
- Deep Randomized Neural Networks (by C. Gallicchio)
- Advanced topics
 - Memory networks, attention, Neural Turing machines
 - Variational deep learning and generative adversarial learning
 - Continual learning (TBC, by V. Lomonaco)

API and research seminars (by Postdoc/Ph.D.)

Reference Book

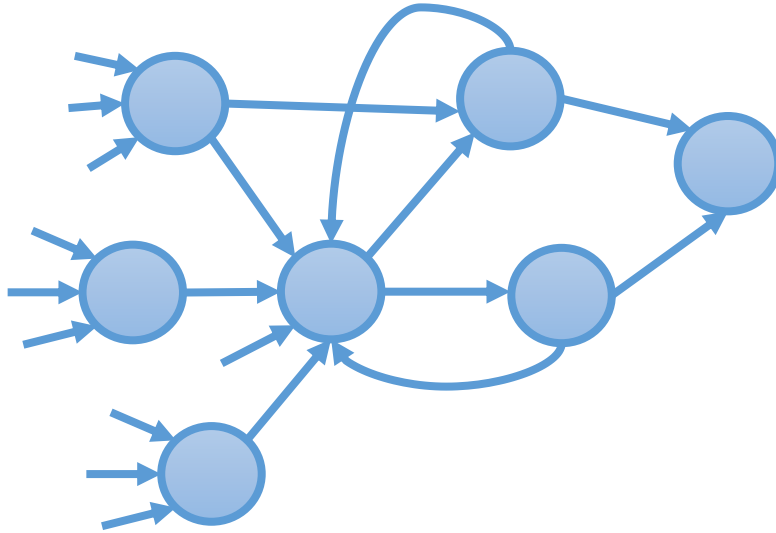


*Ian Goodfellow and Yoshua Bengio and
Aaron Courville, Deep Learning, MIT Press*

- Chapters 6-10, 14,20
- Integrated by course slides and additional readings

Freely available online

Module's Prerequisites



- Formal model of neuron
- Neural network
 - Feed-forward
 - Recurrent
- Cost function optimization
 - Backpropagation/SGD
 - Regularization
- Neural network hyper-parameters and model selection

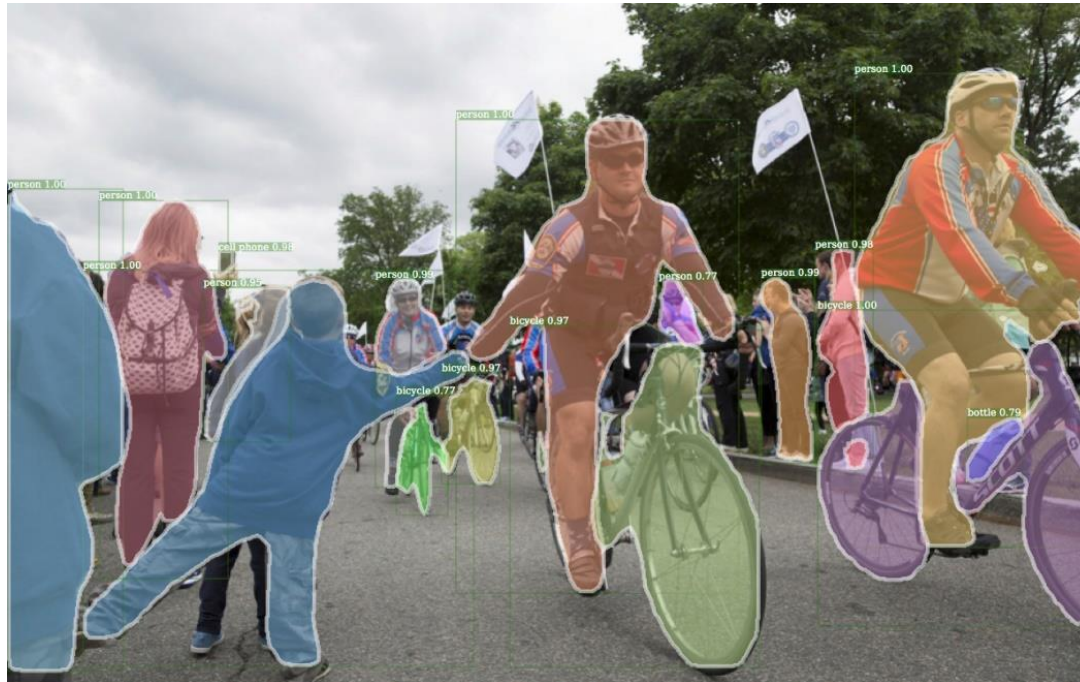
Lecture Outline

- Introduction and historical perspective
- Dissecting the **components** of a CNN
 - Convolution, stride, pooling
- CNN **architectures** for machine vision
 - Putting components back together
 - From LeNet to ResNet
- Advanced topics
 - Interpreting convolutions
 - Advanced **models and applications**

Split in two
lectures

Introduction

Convolutional Neural Networks



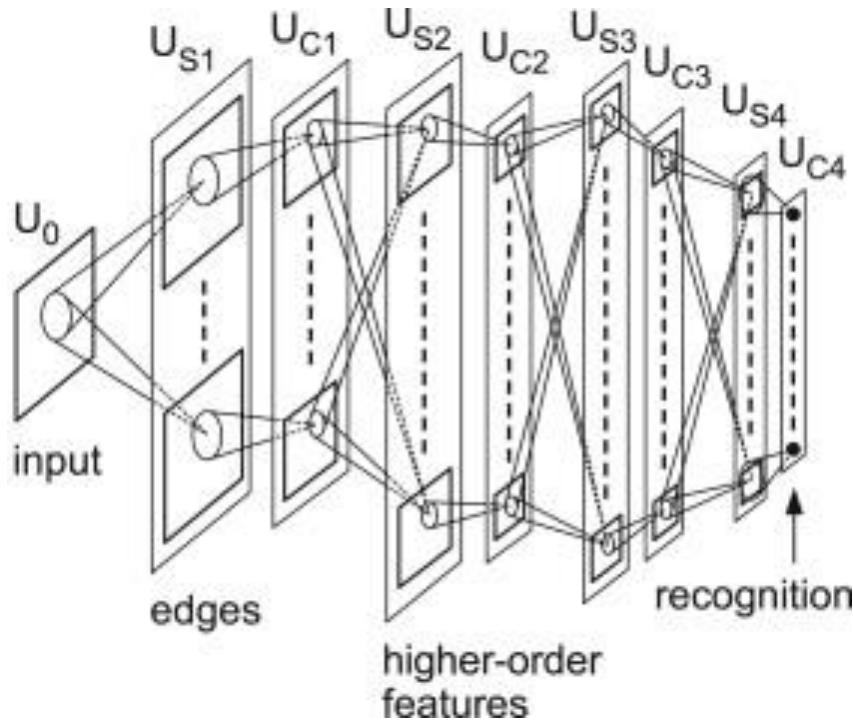
Introduction

Convolutional Neural Networks



Destroying Machine Vision research
since 2012

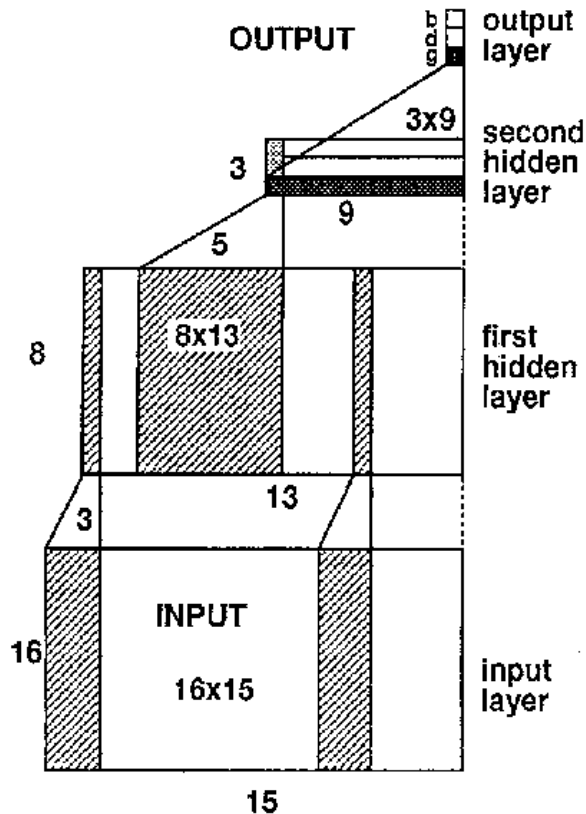
Neocognitron



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



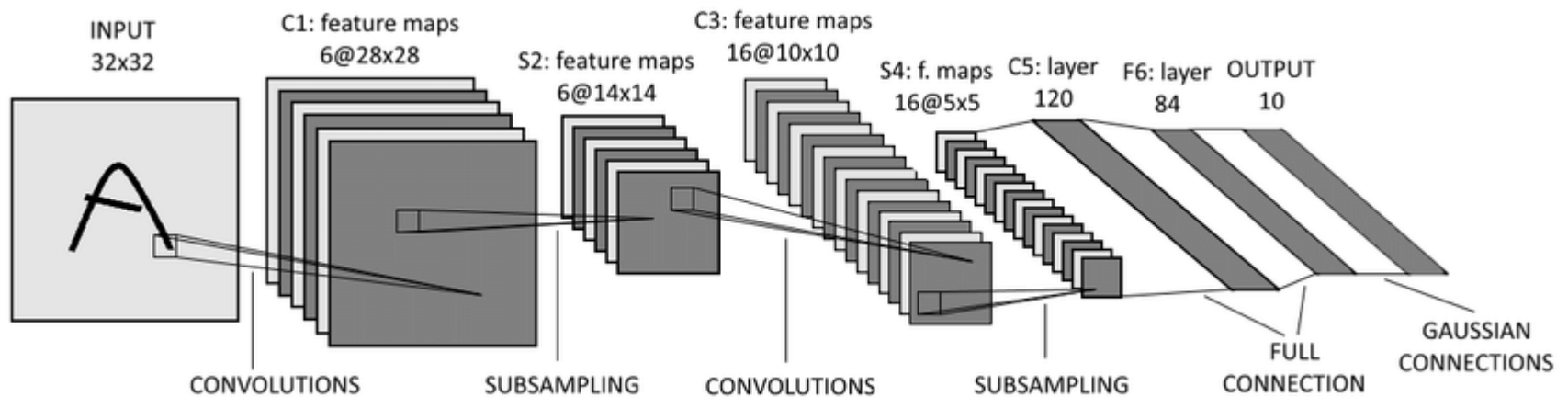
- Apply a bank of 16 convolution kernels to sequences (windows of 15 elements)
- Trained by **backpropagation** with **parameter sharing**
- Guess who introduced it?

...yeah, HIM!



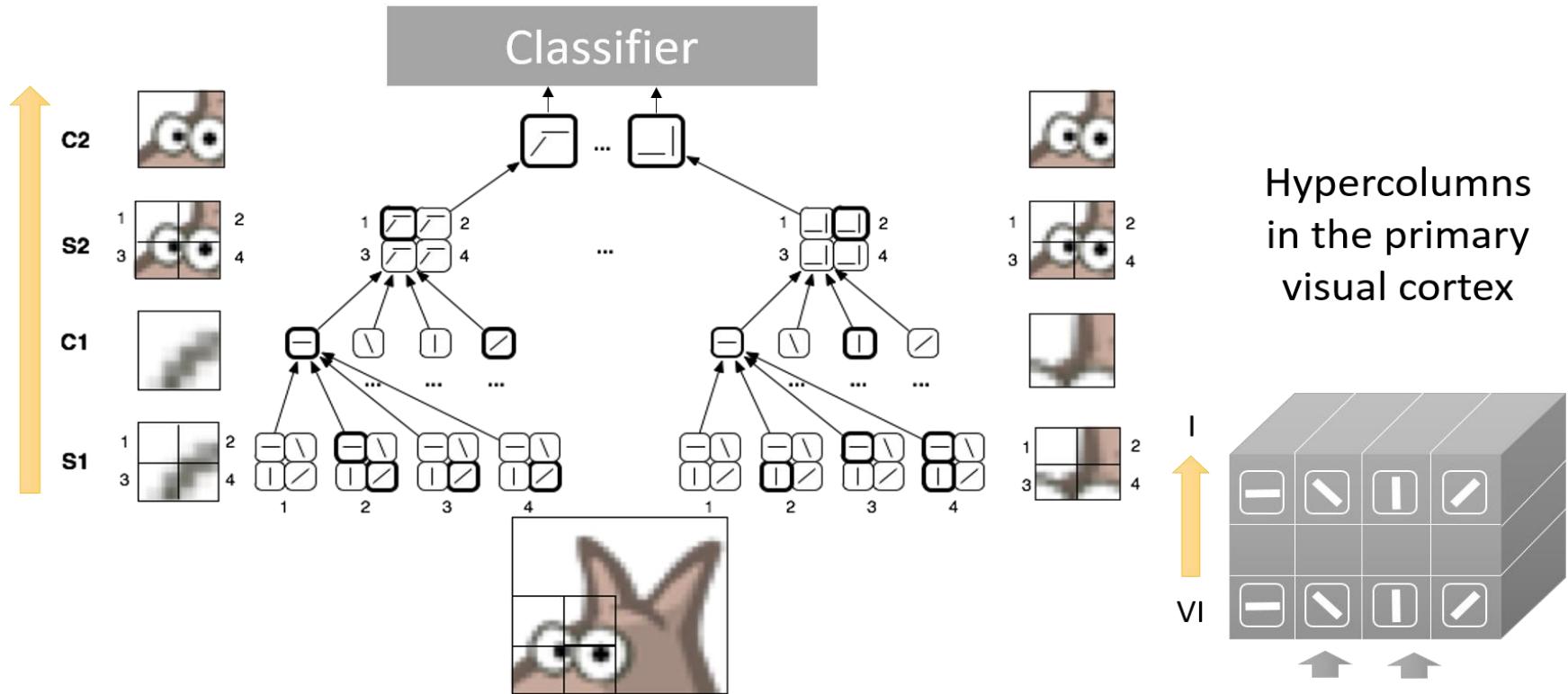
Time delay neural network
(Waibel & Hinton, 1987)

CNN for Images



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

A Revised Bio-Inspired Model (HMAX)



Learning hierarchical representation of objects with the Hubel-Wiesel model (Riesenhuber&Poggio, 1999)

Dense Vector Multiplication

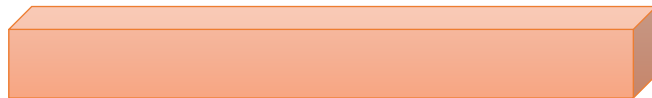
Processing images: the **dense** way

32x32x3 image



Reshape it into
a vector

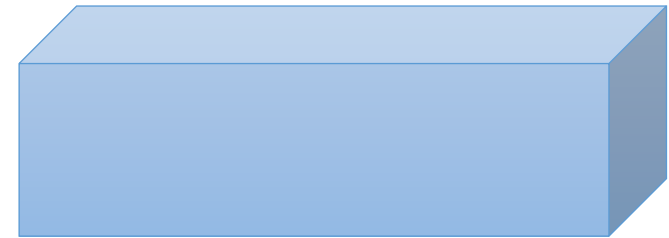
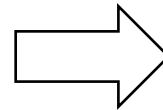
x



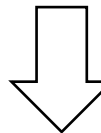
3072

An input-sized weight
vector for each
hidden neuron

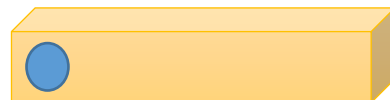
100x3072



W

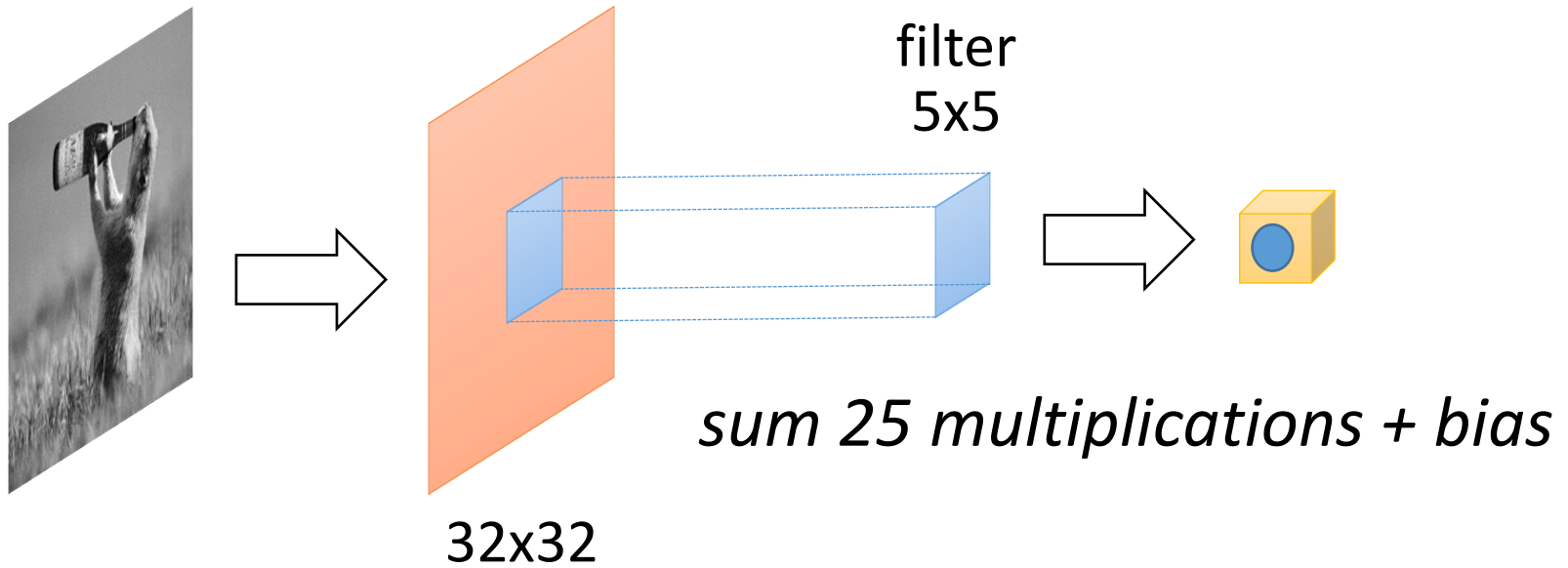
$$Wx^T$$


Each element contains the
activation of 1 neuron



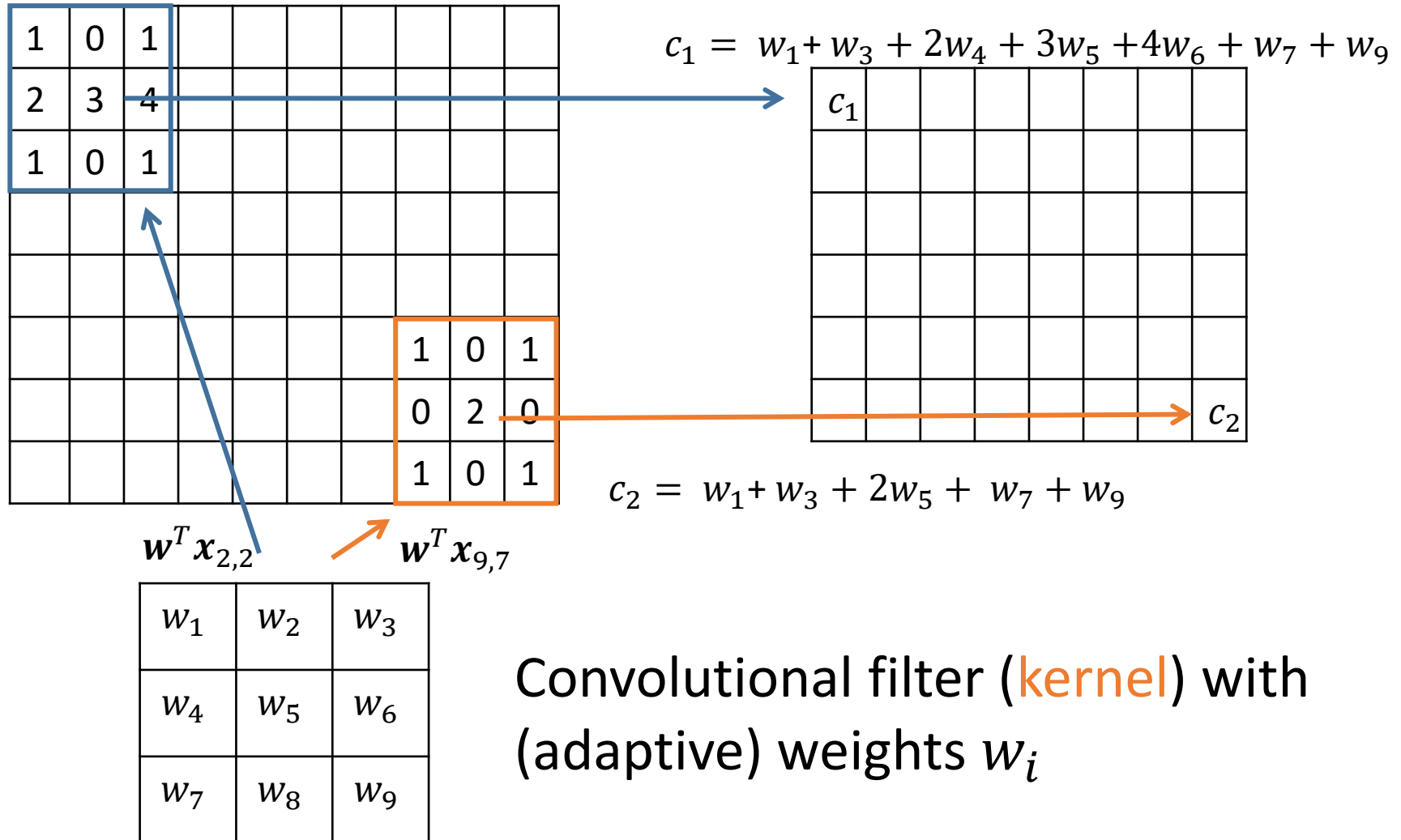
100

Convolution (Refresher)

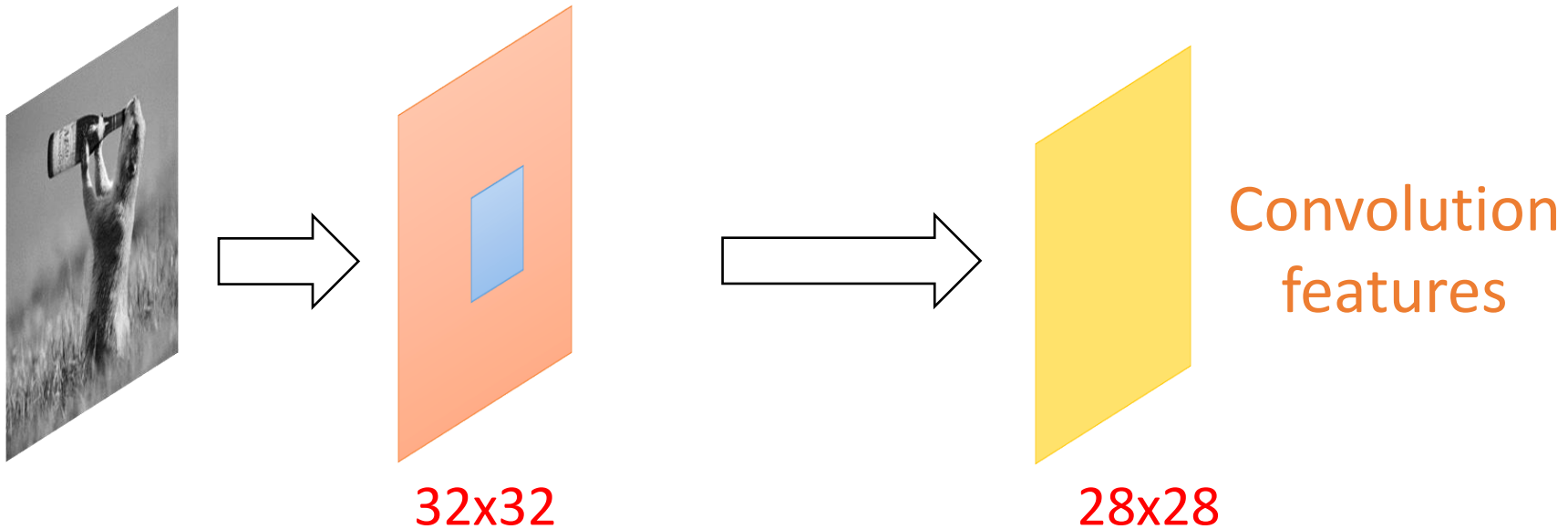


Matrix input preserving
spatial structure

Adaptive Convolution

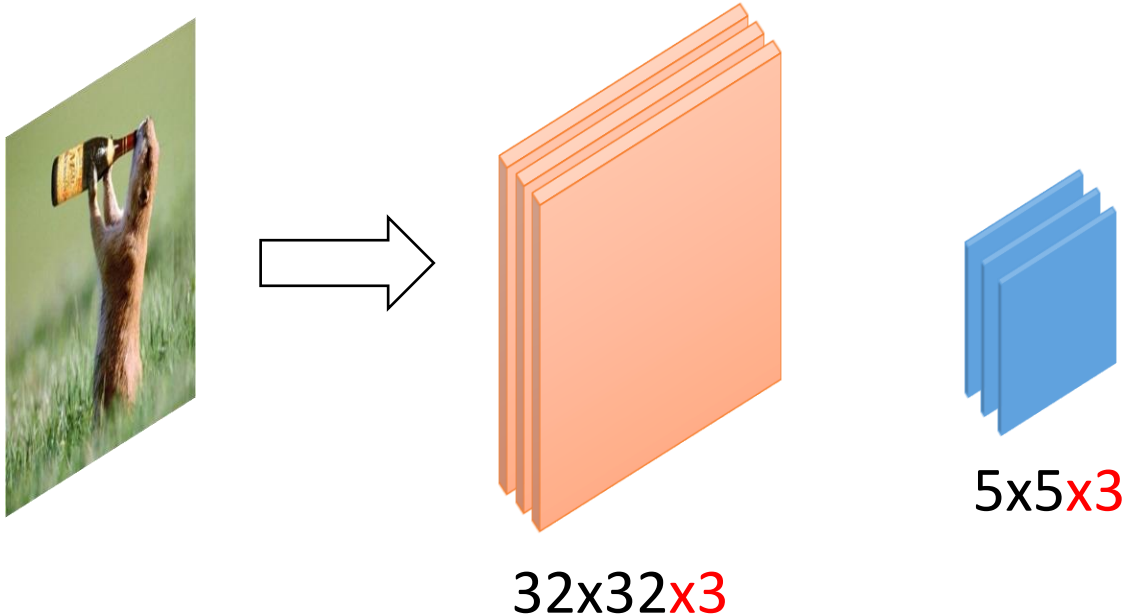


Convolutional Features



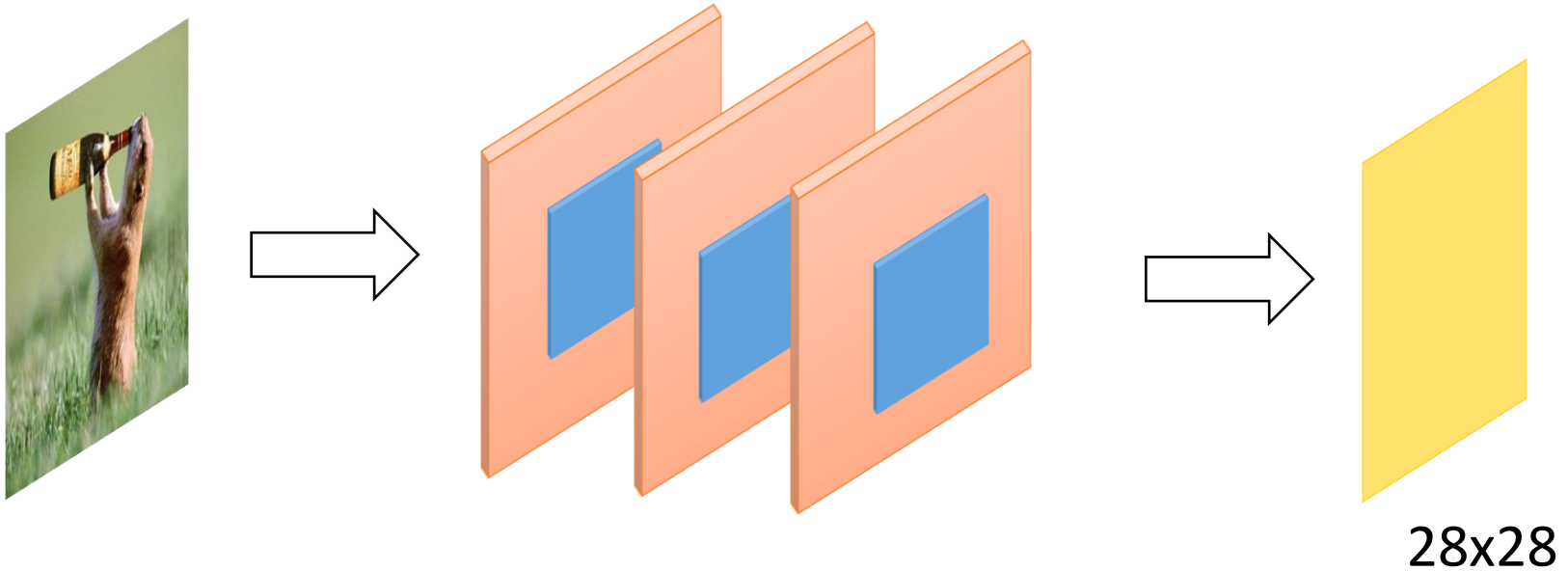
Slide the filter on the image
computing elementwise products
and summing up

Multi-Channel Convolution



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

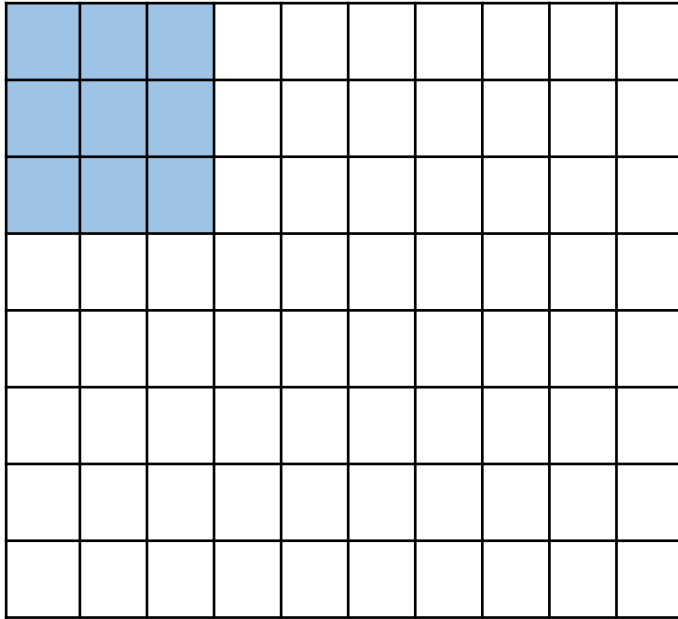
Multi-Channel Convolution



All channels are typically **convolved together**

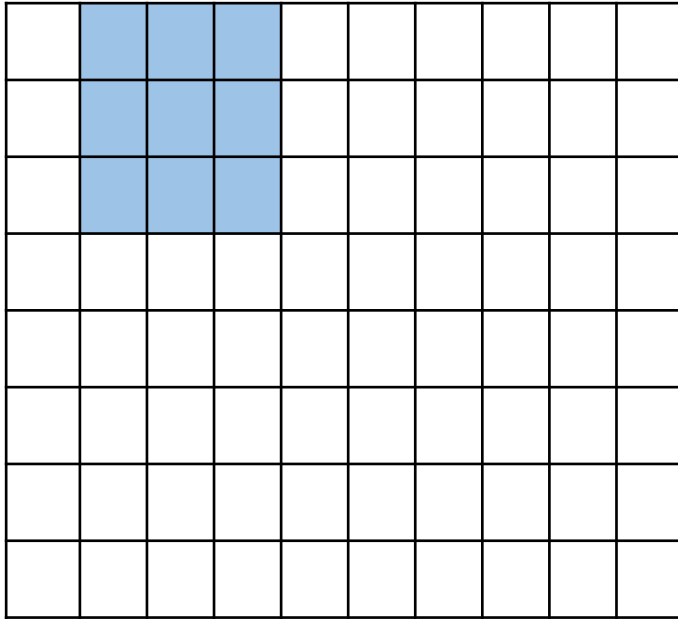
- They are summed-up in the convolution
- The **convolution map stays bi-dimensional**

Stride



- Basic convolution **slides the filter** on the image one pixel at a time
 - Stride = 1

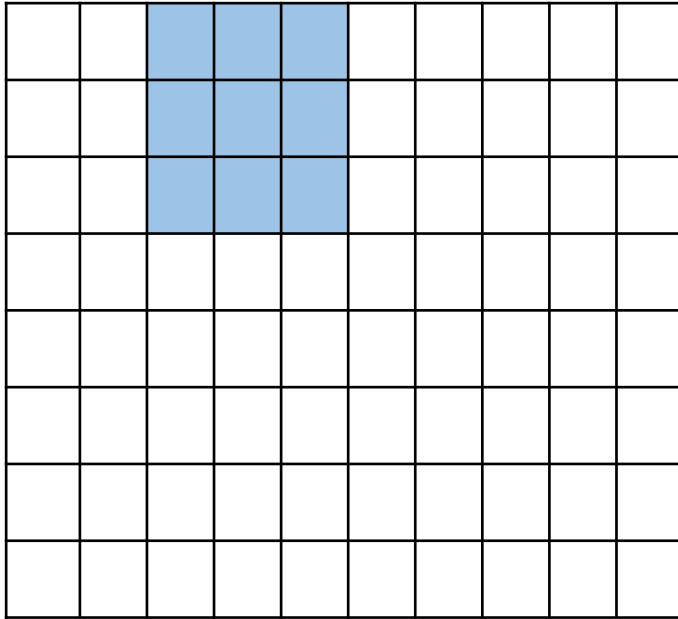
Stride



stride = 1

- Basic convolution **slides the filter** on the image one pixel at a time
 - Stride = 1

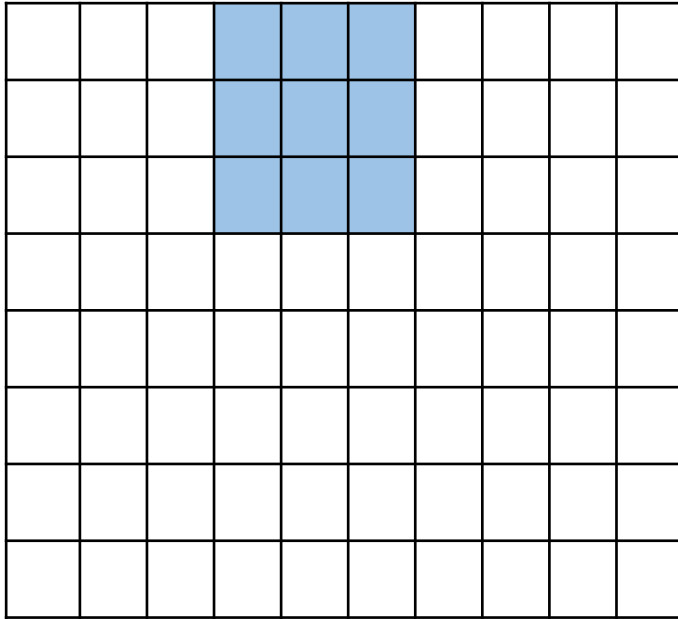
Stride



stride = 1

- Basic convolution **slides the filter** on the image one pixel at a time
 - Stride = 1

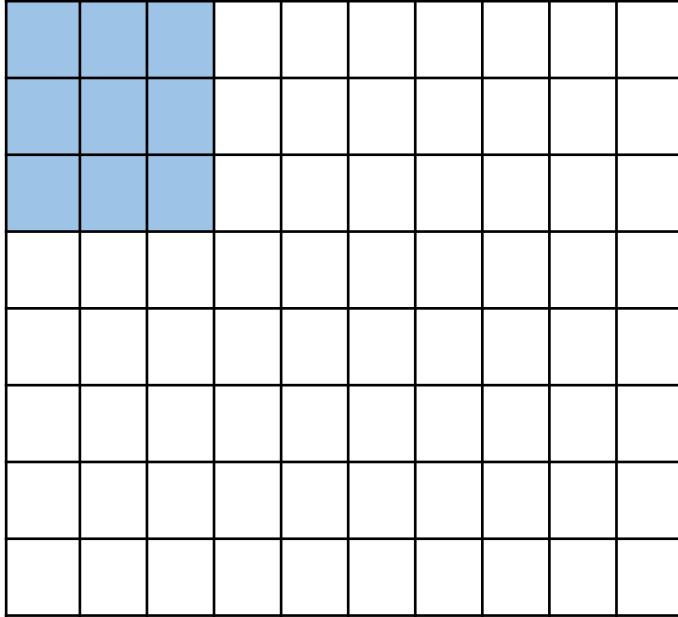
Stride



stride = 1

- Basic convolution **slides the filter** on the image one pixel at a time
 - Stride = 1

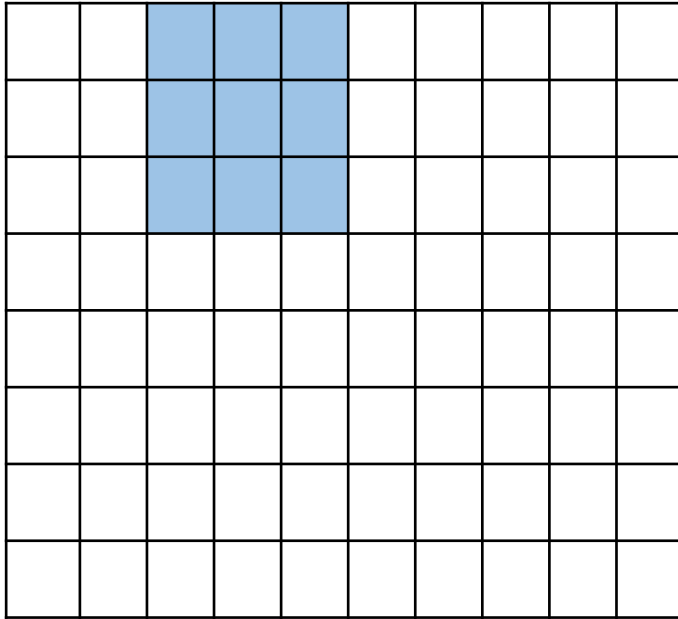
Stride



stride = 2

- Basic convolution **slides the filter** on the image one pixel at a time
 - Stride = 1
- Can define a different stride
 - Hyperparameter

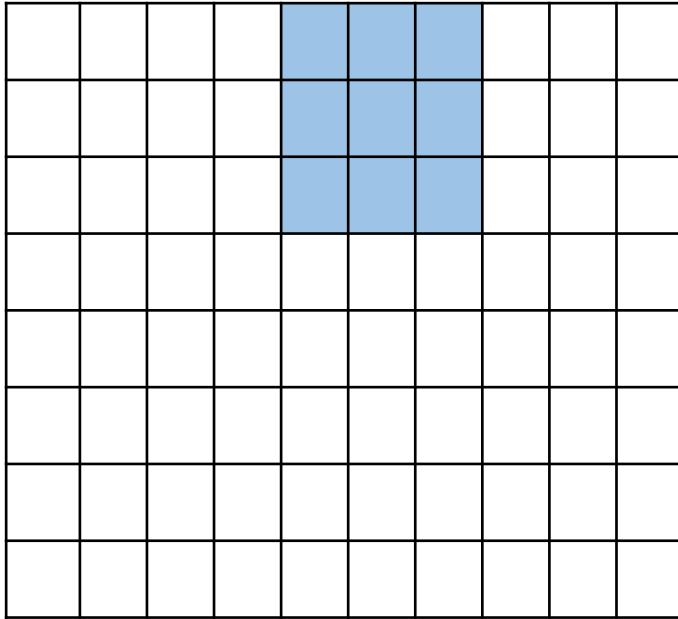
Stride



stride = 2

- Basic convolution **slides the filter** on the image one pixel at a time
 - Stride = 1
- Can define a different stride
 - Hyperparameter

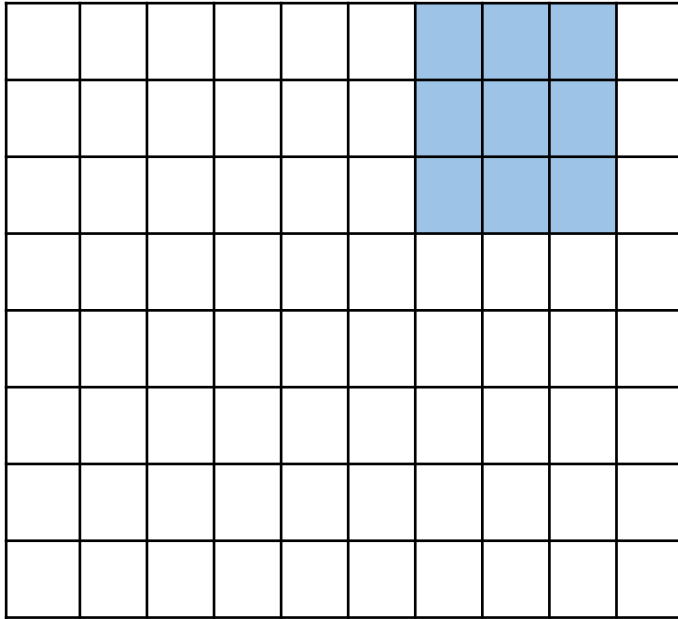
Stride



stride = 2

- Basic convolution **slides the filter** on the image one pixel at a time
 - Stride = 1
- Can define a different stride
 - Hyperparameter

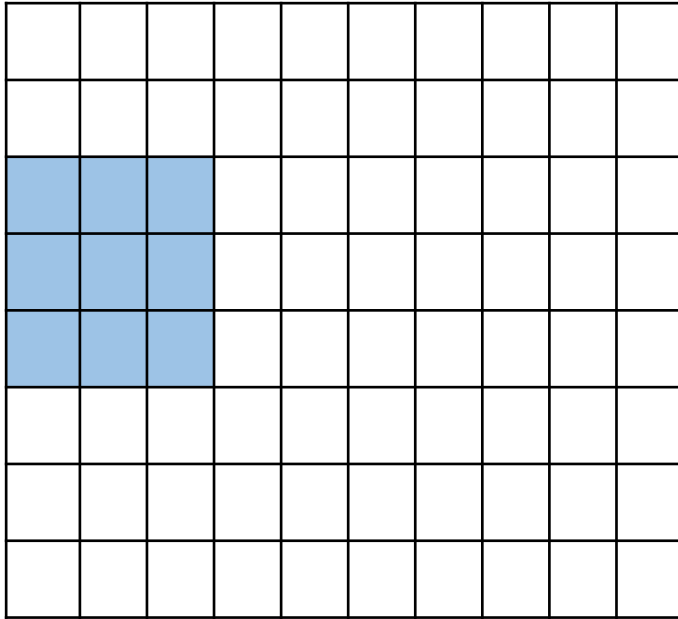
Stride



stride = 2

- Basic convolution **slides the filter** on the image one pixel at a time
 - Stride = 1
- Can define a different stride
 - Hyperparameter

Stride

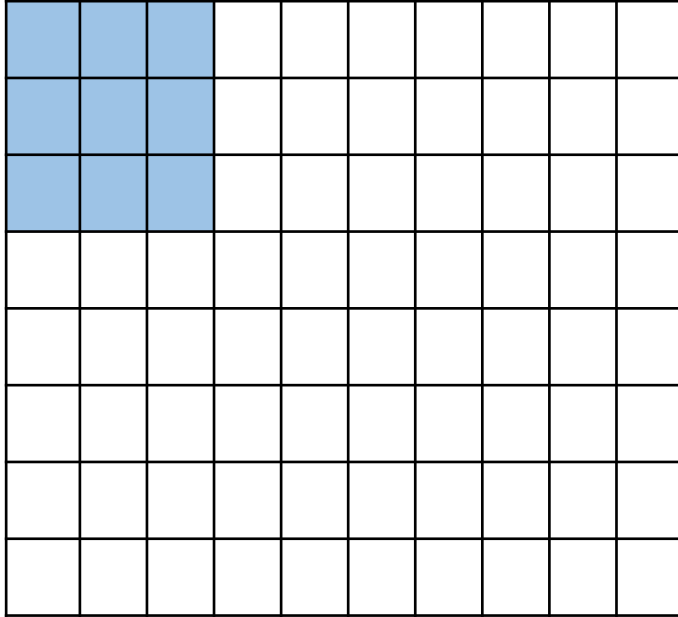


stride = 2

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

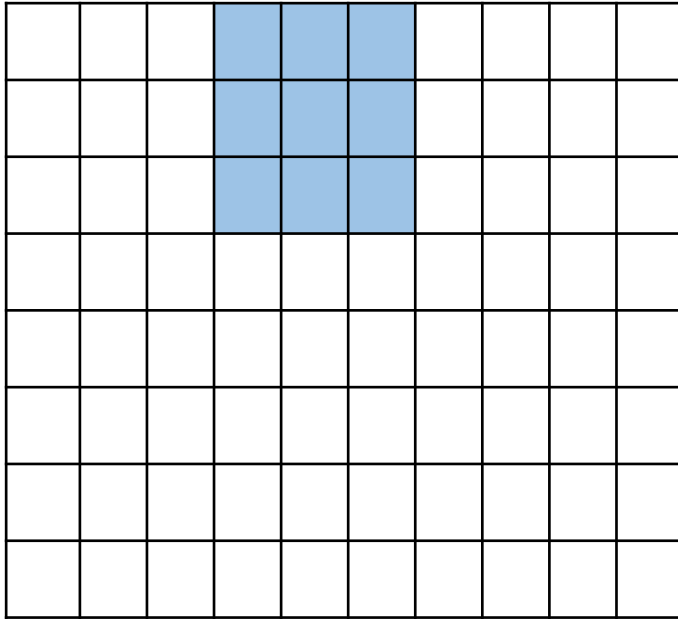
Stride



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

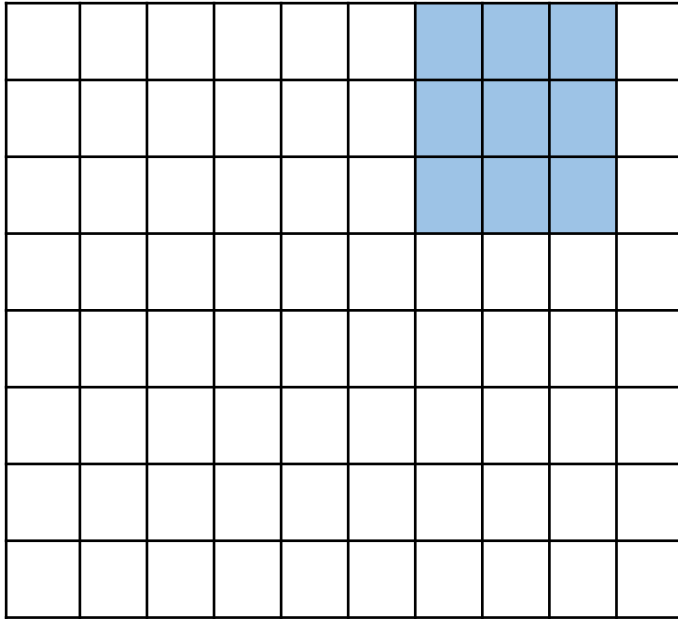
Stride



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



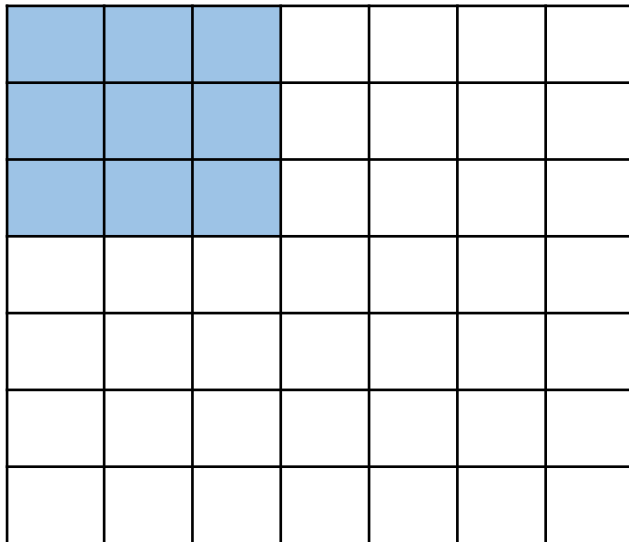
stride = 3

- 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**?

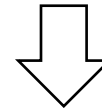
$W=7$



$H=7$

Take a 3x3 filter with stride 1

$K=3, S=1$

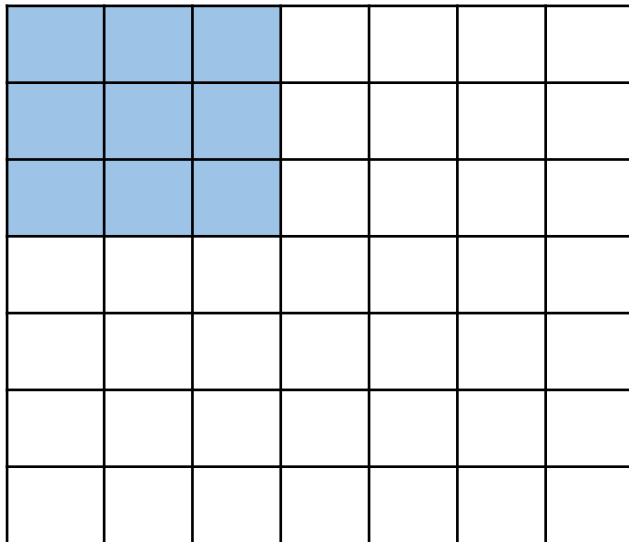


Output image is: **5x5**

Activation Map Size

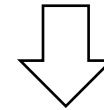
What is the **size of the image** after application of a **filter** with a given **size** and **stride**?

$W=7$



Take a 3x3 filter with stride 2

$K=3, S=2$



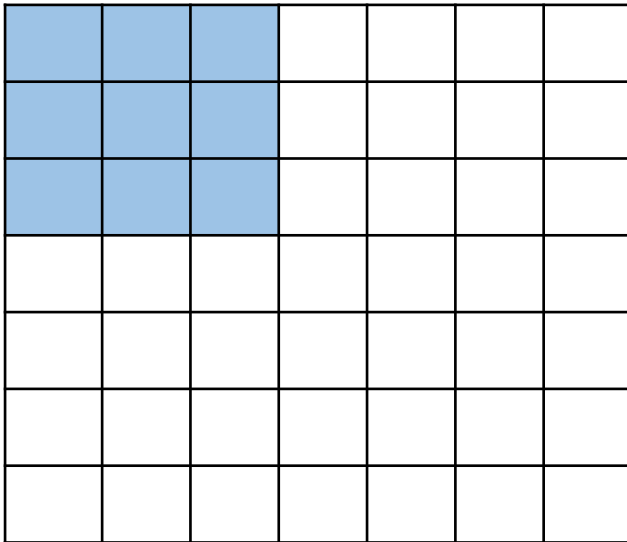
Output image is: **3x3**

Activation Map Size

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

$W=7$

$H=7$



General rule

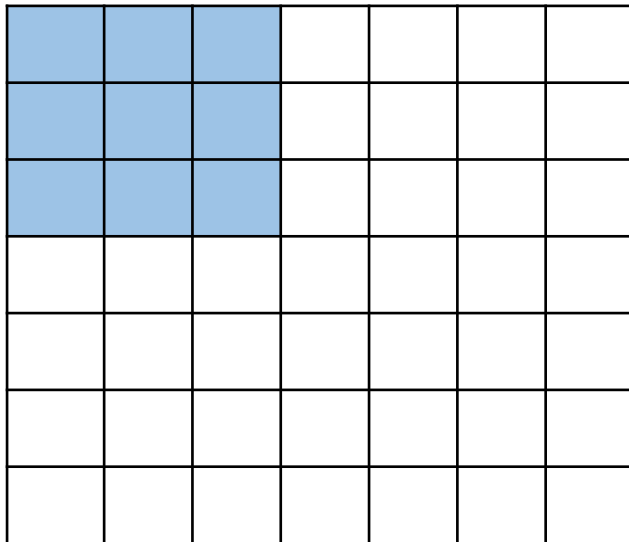
$$W' = \frac{W - K}{S} + 1$$

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

Activation Map Size

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

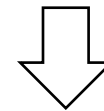
$W=7$



$H=7$

Take a 3x3 filter with stride 3

$K=3, S=3$



Output image is:

Doesn't fit! Cannot scan the whole image

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

$W=7$

[illegible]

Zero Padding

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

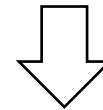
$W=7$ ($P=1$)

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

$H=7$

($P = 1$)

$K=3, S=1$



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

$W=7$ ($P=1$)

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

$H=7$

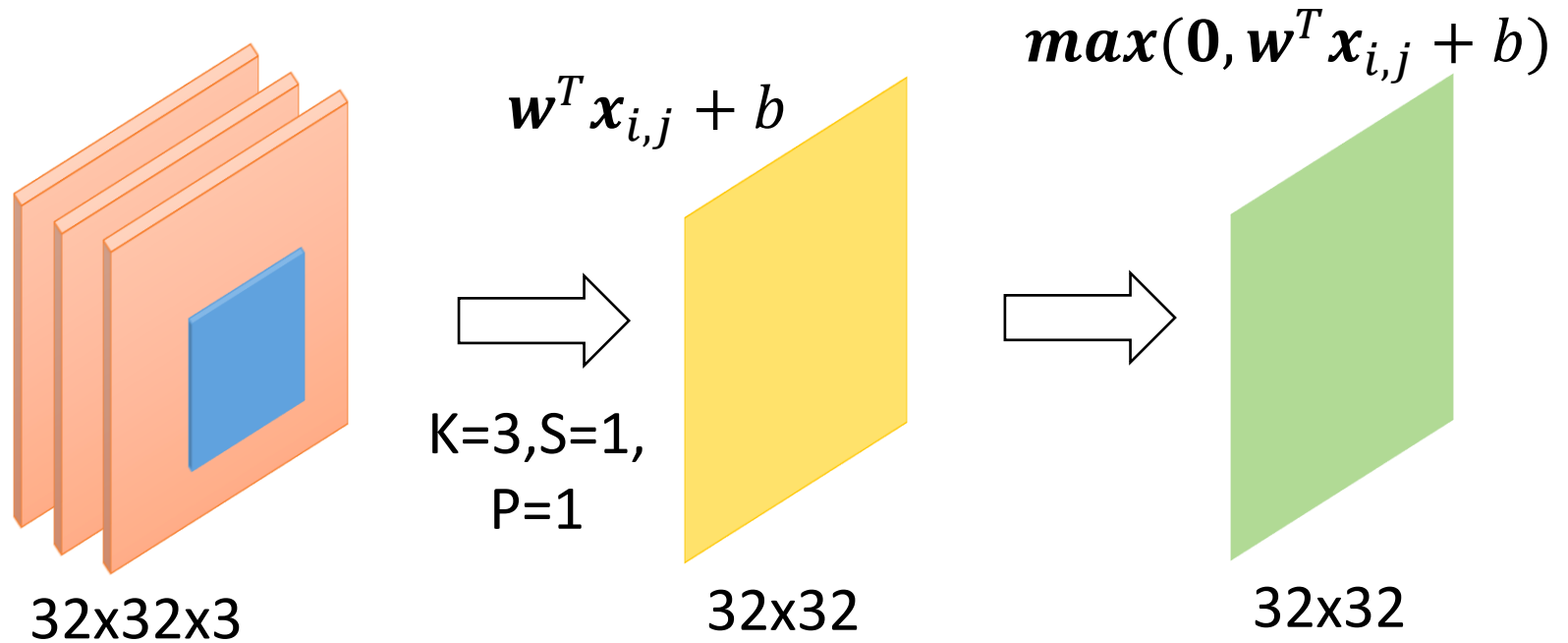
($P = 1$)

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**

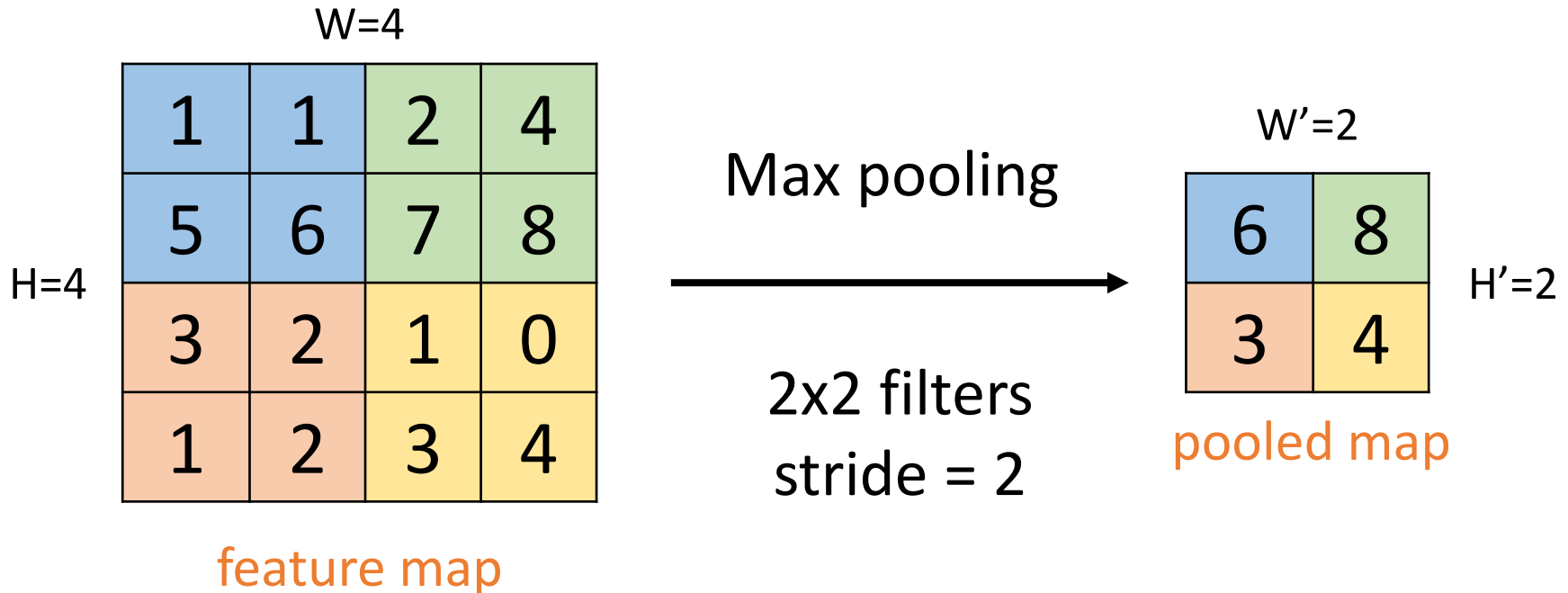
Feature Map Transformation



- 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

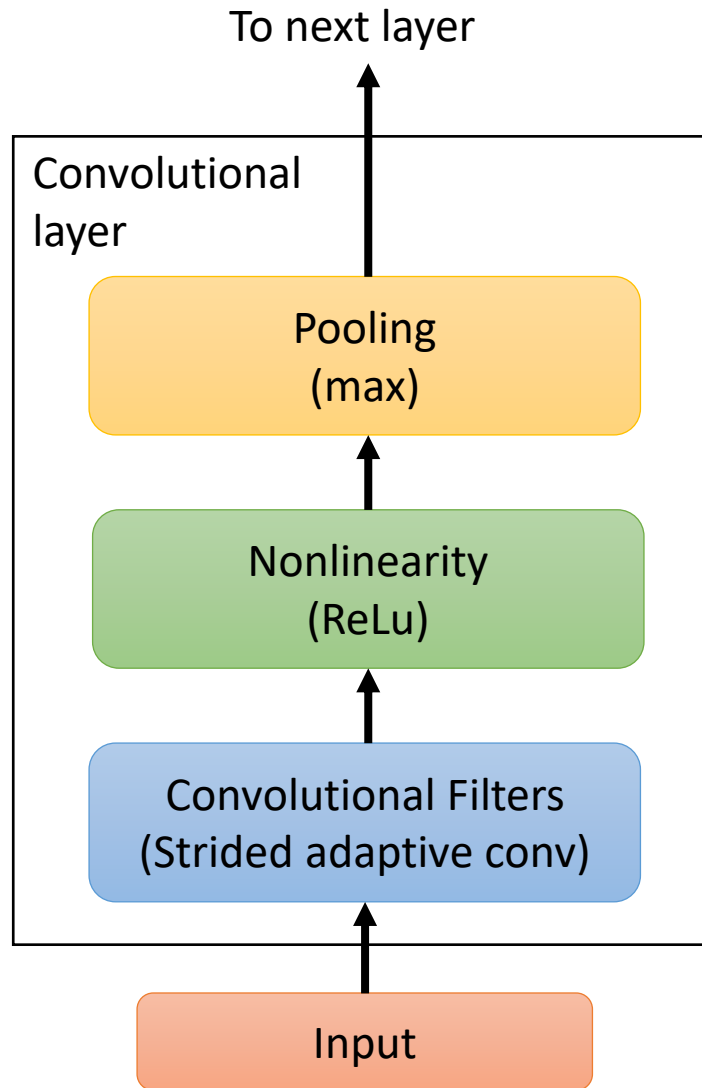


Pooling Facts

- 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

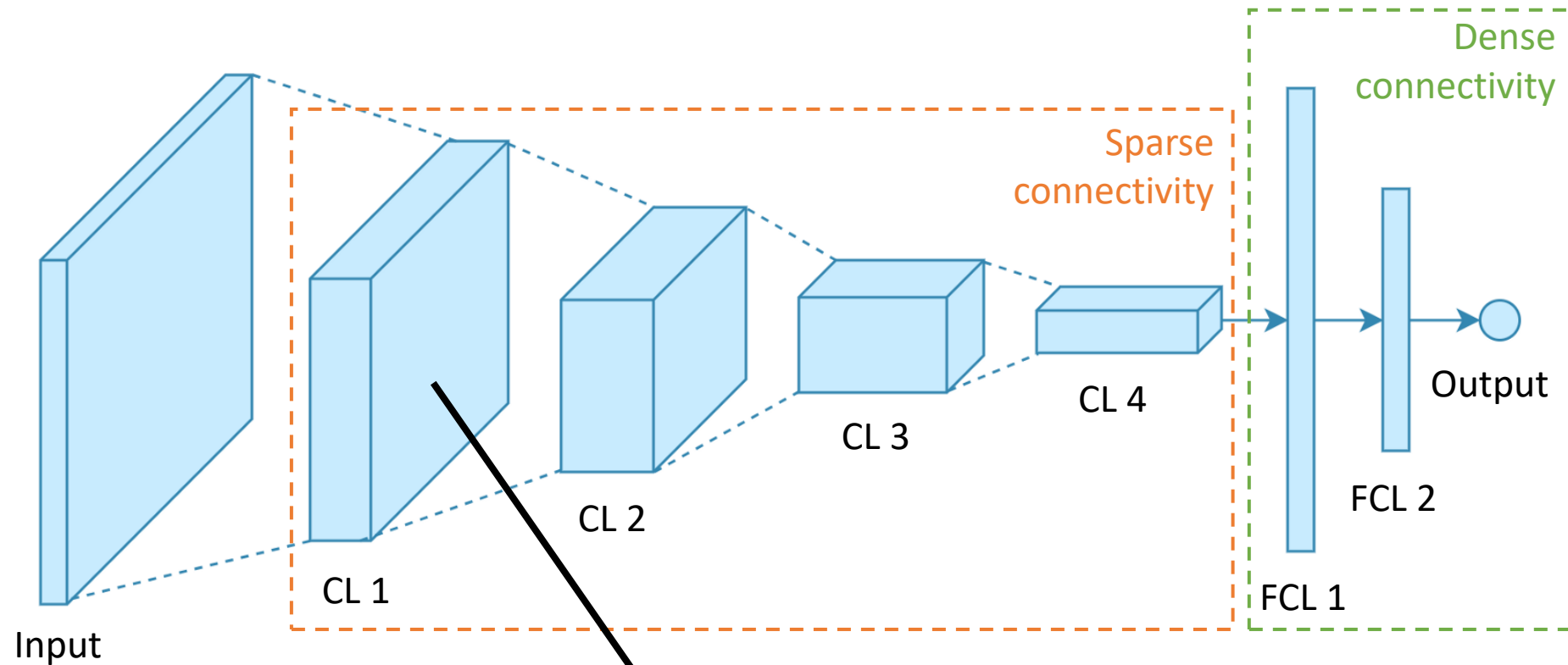
$$W' = \frac{W - K}{S} + 1$$

The Convolutional Architecture



- An architecture 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

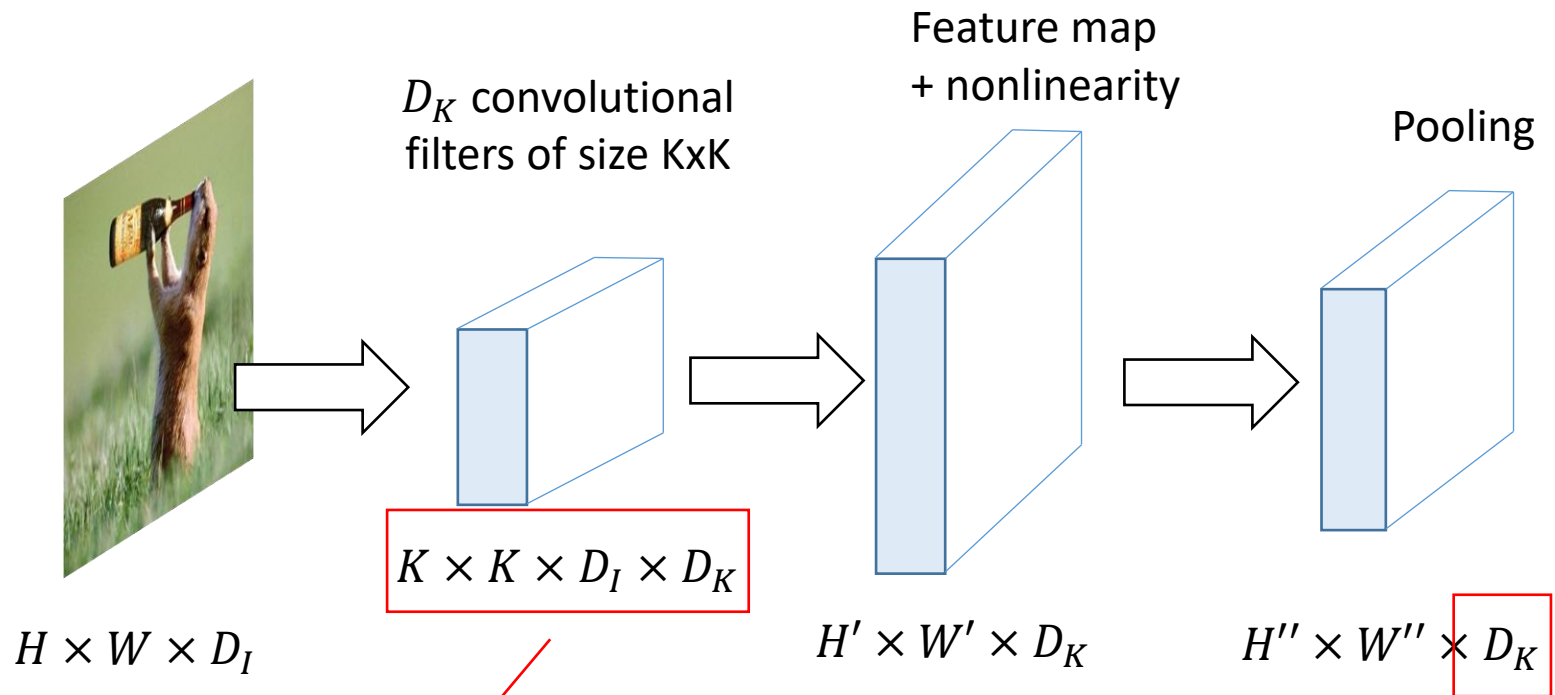
A Bigger Picture



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 (in most frameworks, you have to supply it)

A Note on Convolution

- We know that **discrete convolution** between image I and a filter/kernel K is

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

and it is **commutative**.

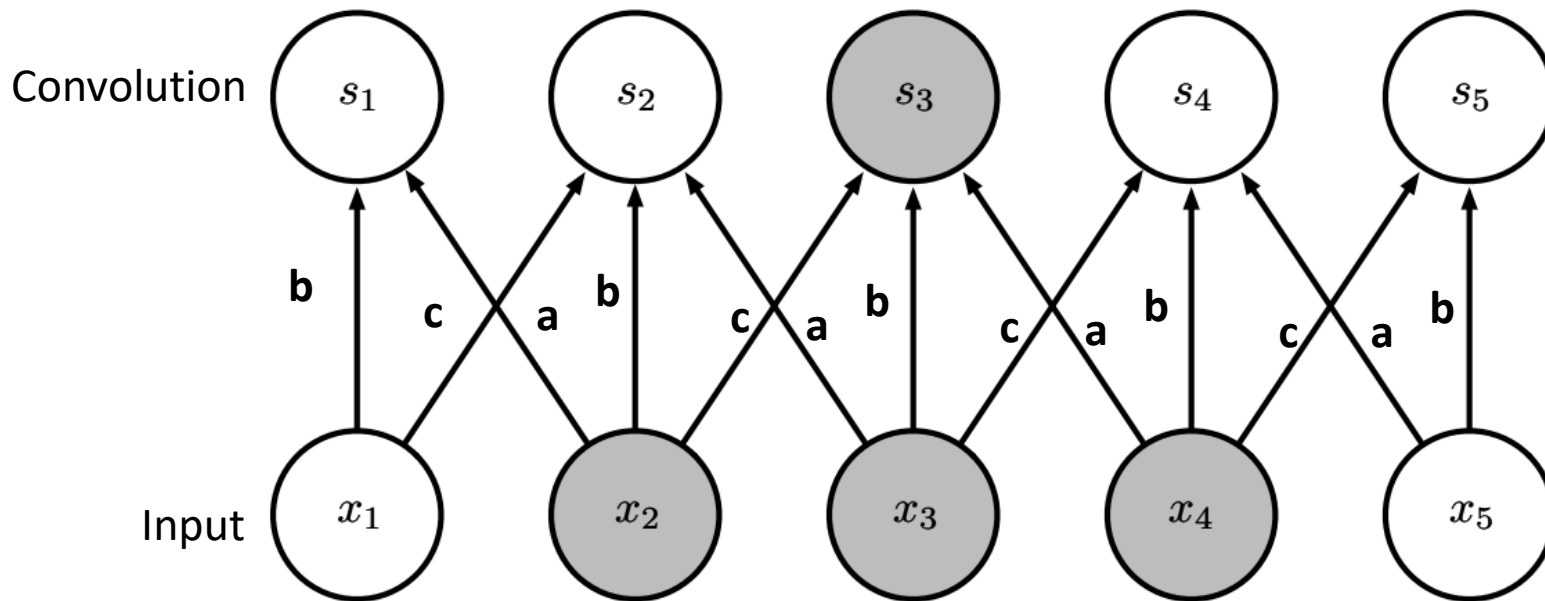
- In practice, convolution **implementation in DL libraries** does not flip the kernel

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

Which is **cross-correlation** and it is not commutative.

CNN as a Sparse Neural Network

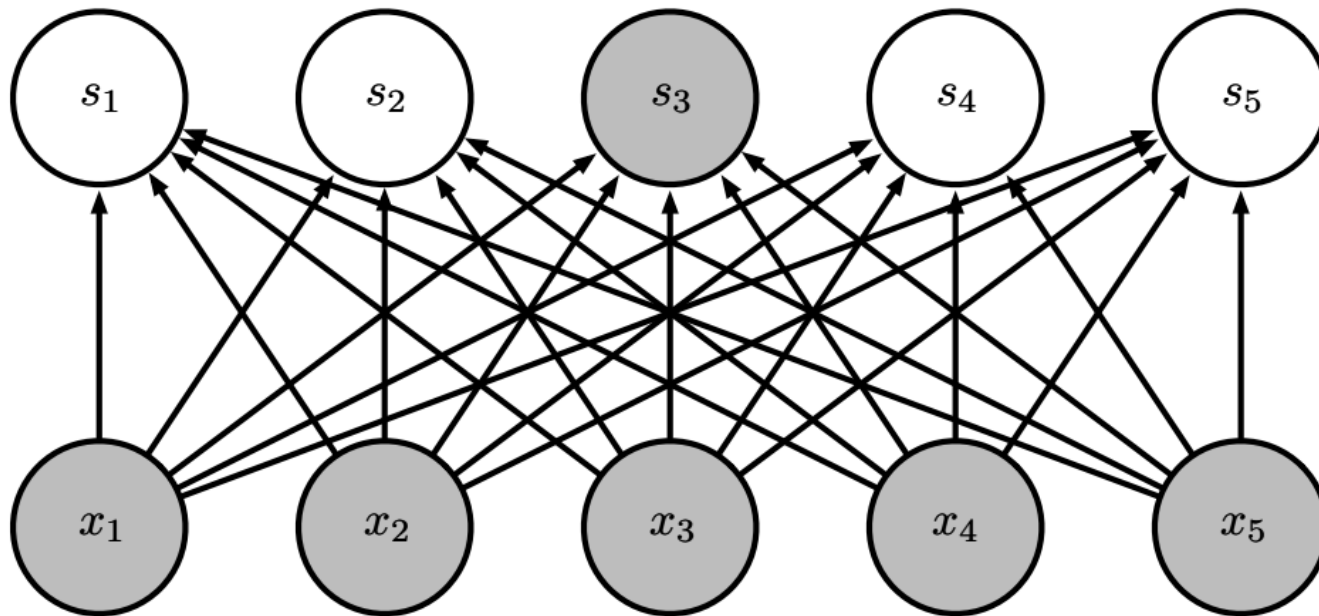
Let us take a 1-D input (sequence) to ease graphics



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

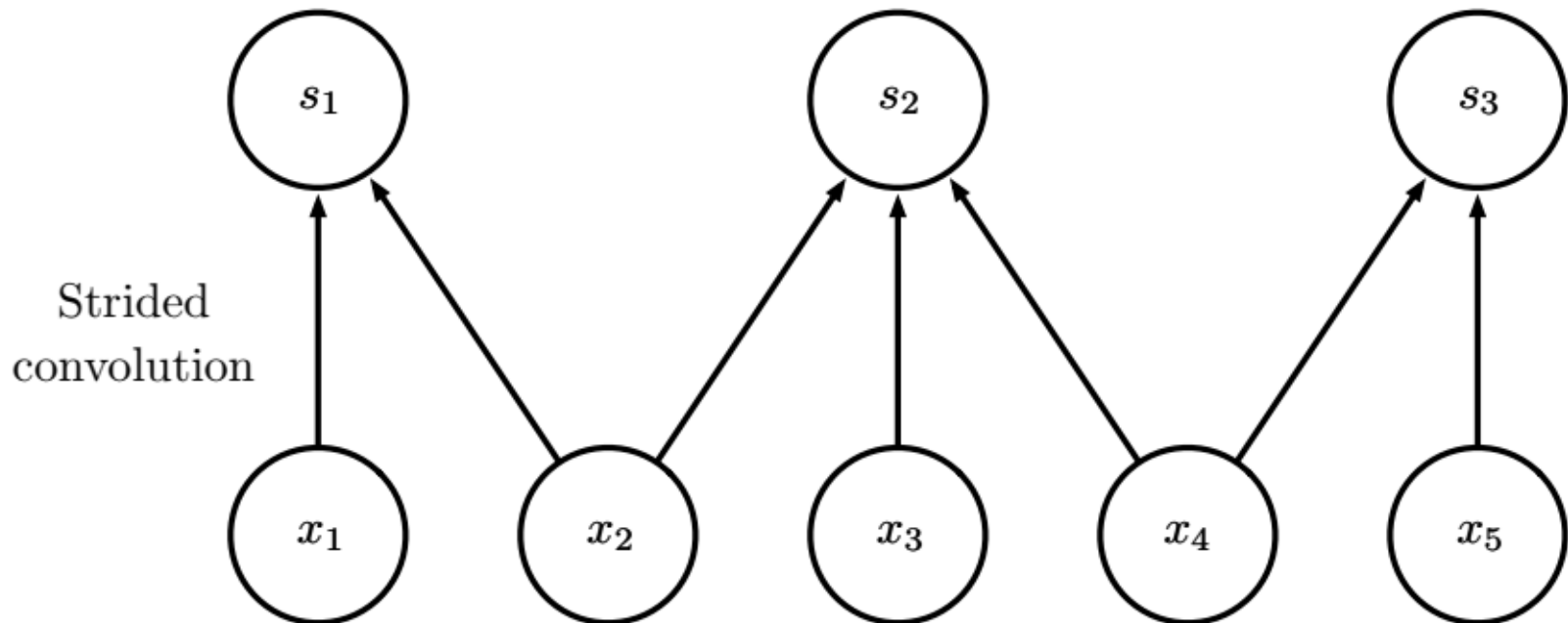
Dense Network

The dense counterpart would look like this



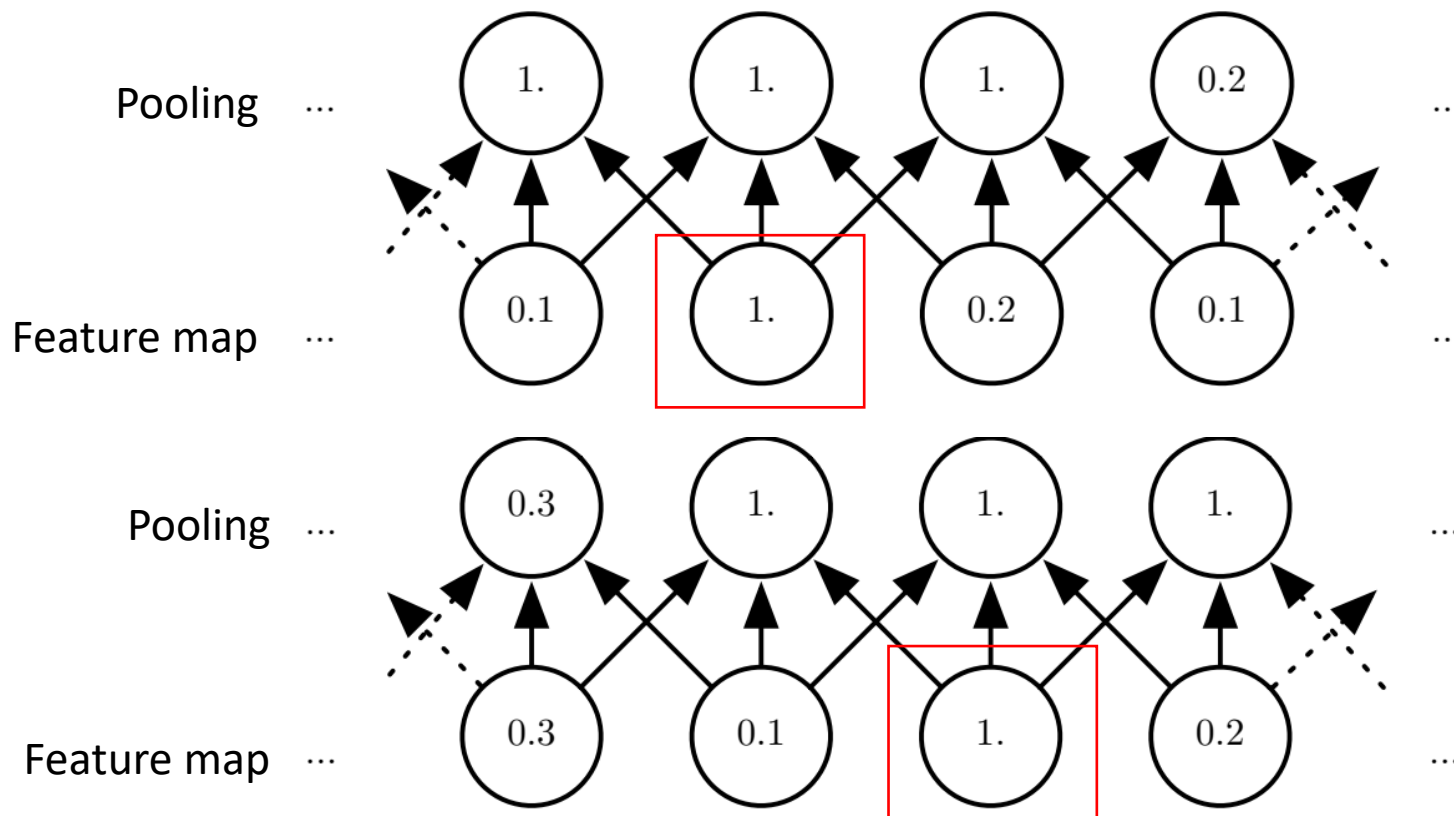
Strided Convolution

Make connectivity sparser

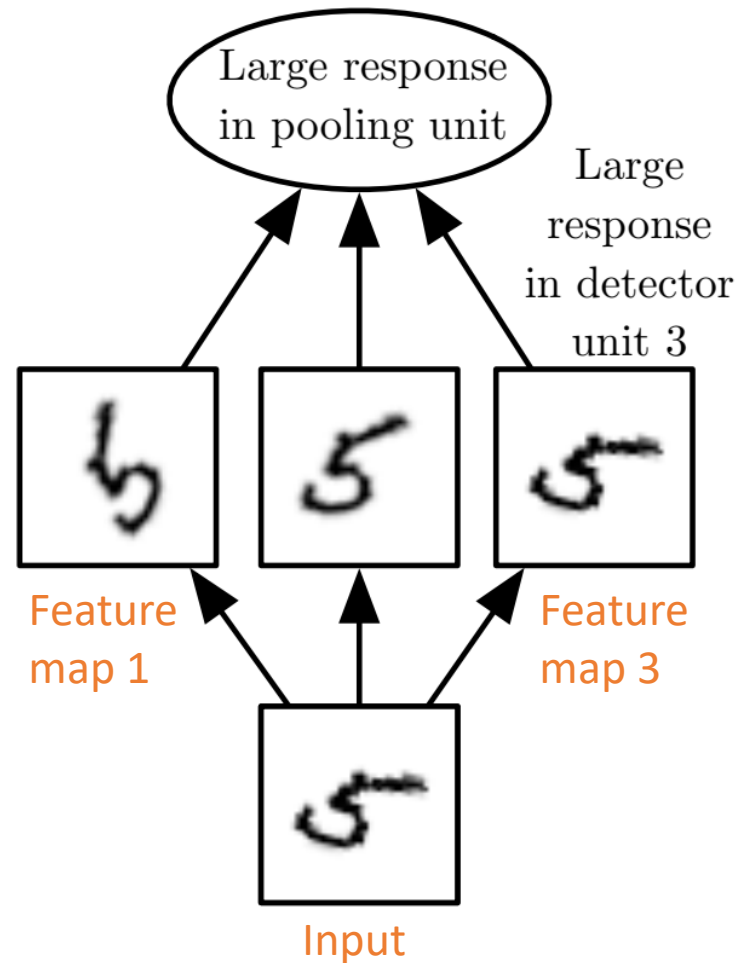
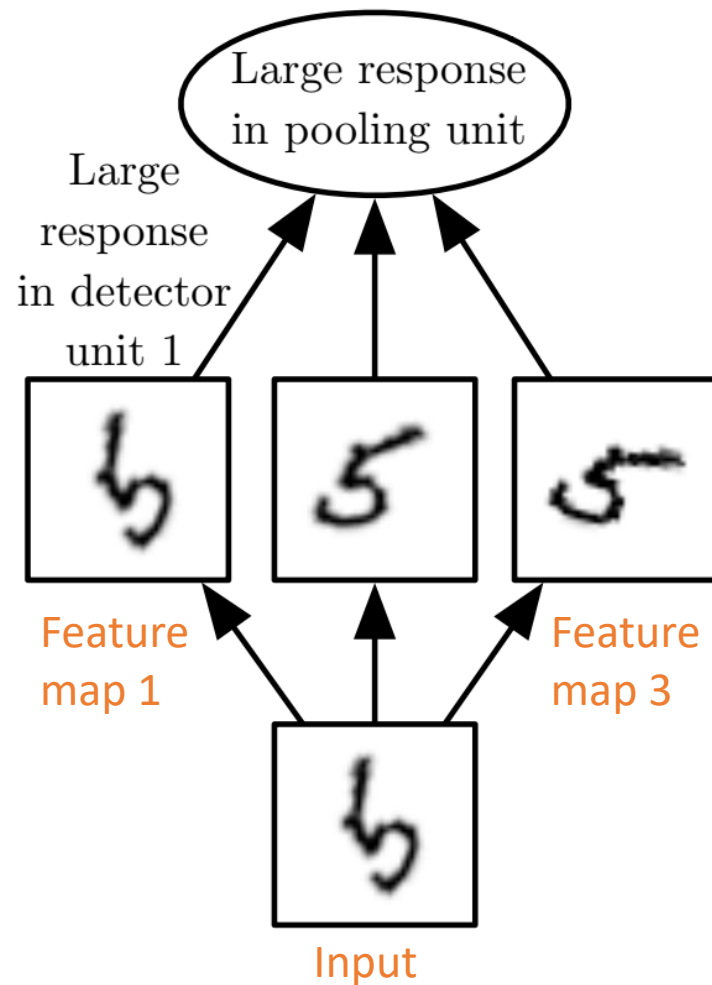


Max-Pooling and Spatial Invariance

A feature is detected even if it is spatially translated

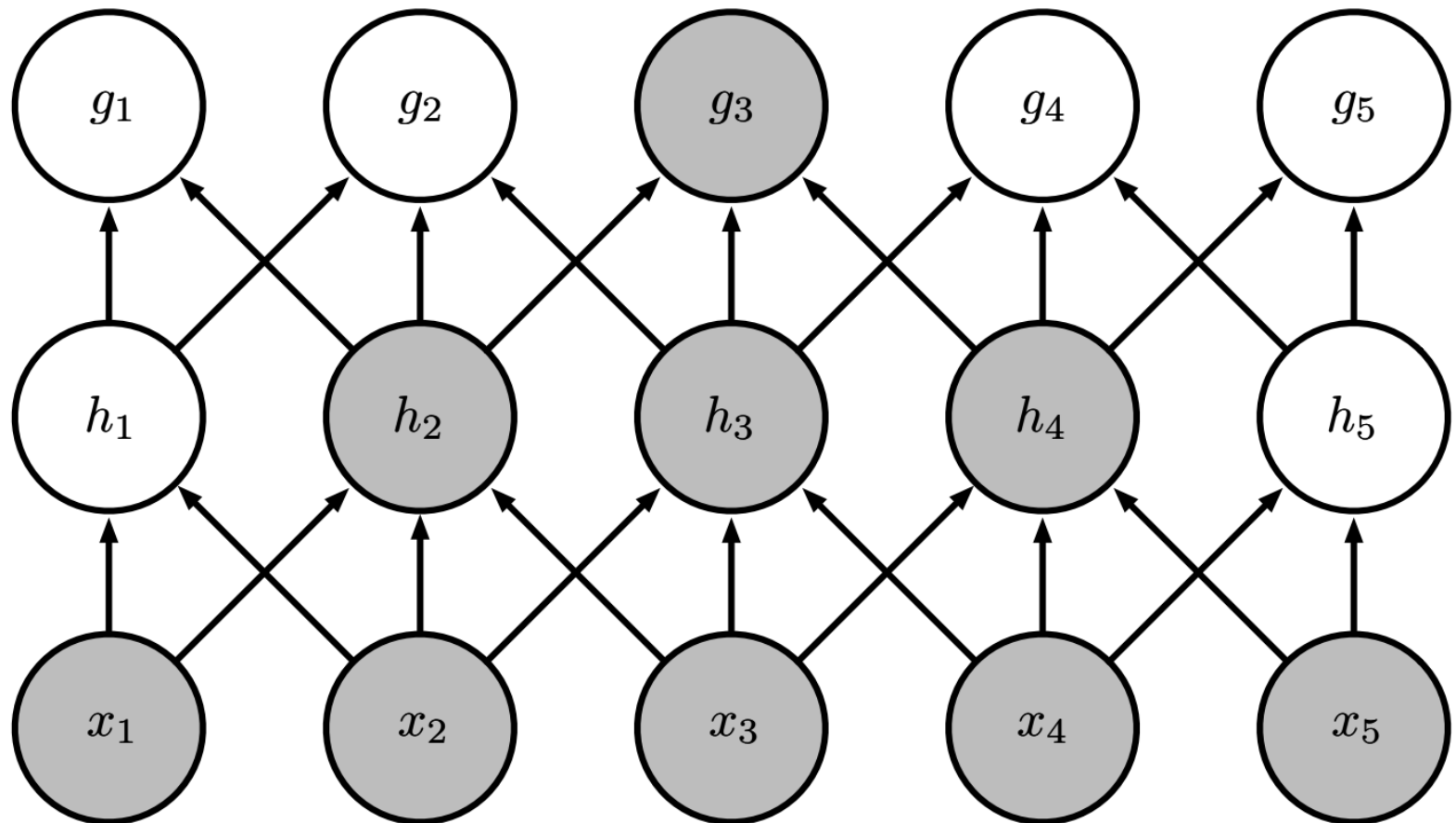


Cross Channel Pooling and Spatial Invariance

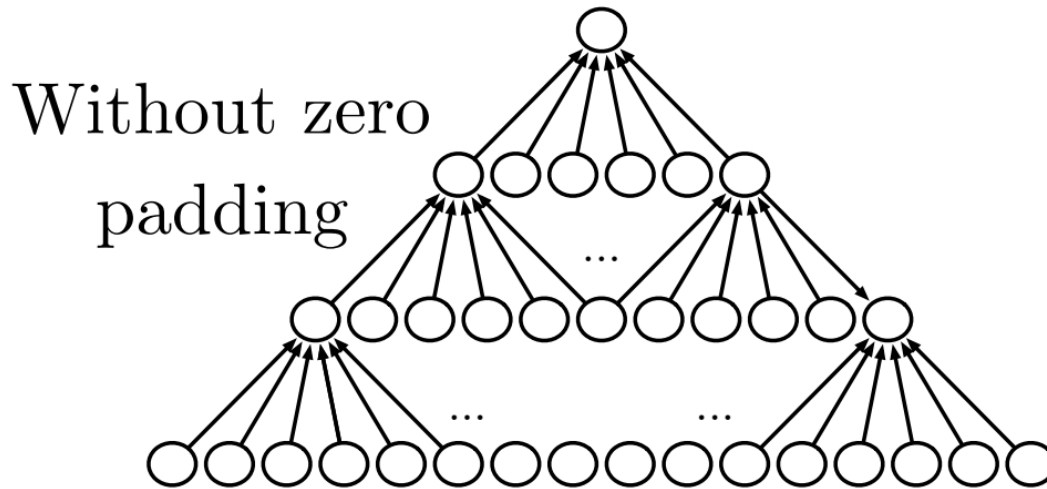


Hierarchical Feature Organization

The deeper the larger the receptive field of a unit

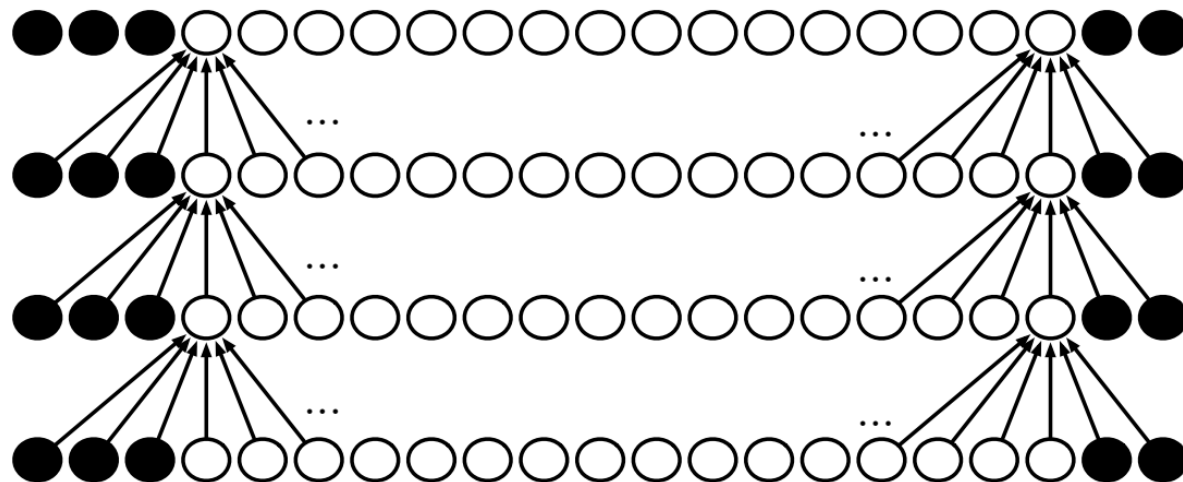


Zero-Padding Effect



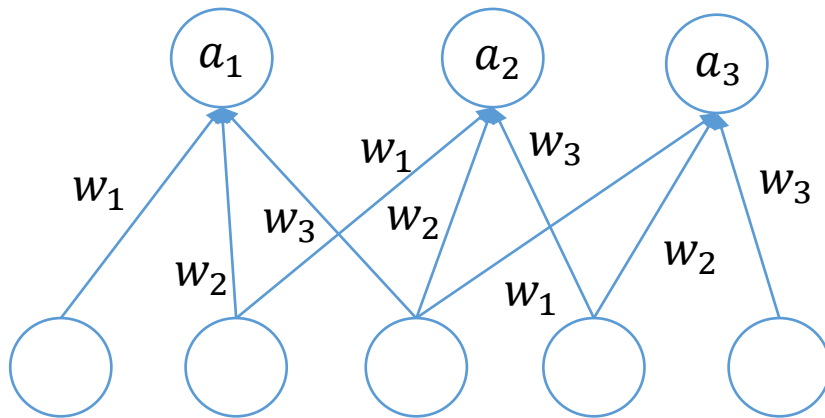
Assuming no
pooling

With zero
padding



CNN Training

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

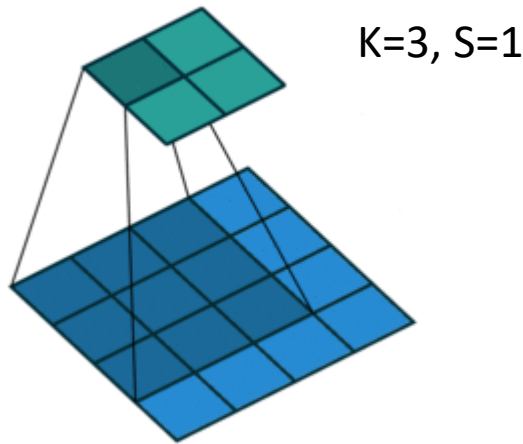


The gradient Δw_i is obtained by **summing the contributions from all connections** sharing the weight

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



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

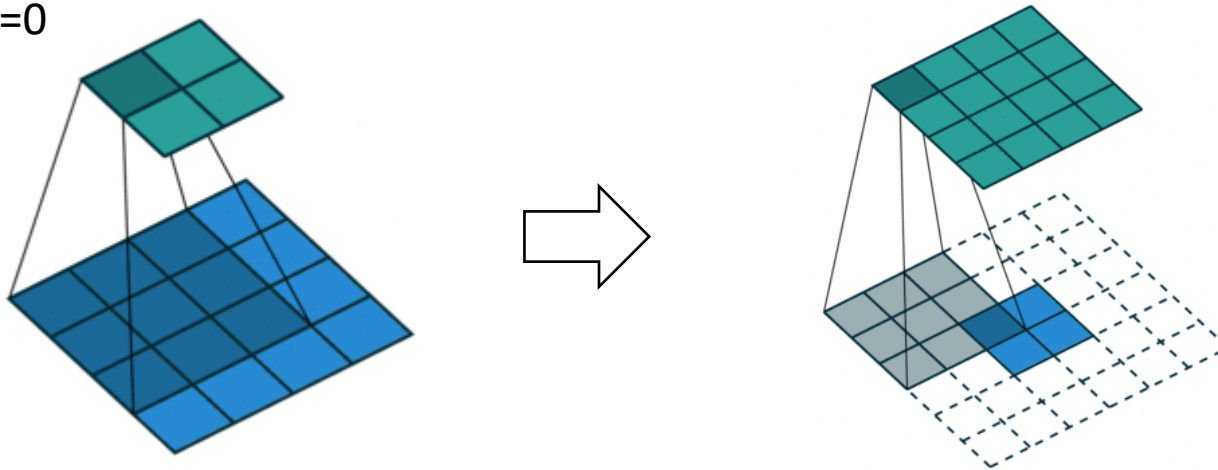
$$\begin{pmatrix} w_{0,0} & w_{0,1} & w_{0,2} & 0 & w_{1,0} & w_{1,1} & w_{1,2} & 0 & w_{2,0} & w_{2,1} & w_{2,2} & 0 & 0 & 0 & 0 & 0 \\ 0 & w_{0,0} & w_{0,1} & w_{0,2} & 0 & w_{1,0} & w_{1,1} & w_{1,2} & 0 & w_{2,0} & w_{2,1} & w_{2,2} & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & w_{0,0} & w_{0,1} & w_{0,2} &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

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

$K=3, S=1, P=0$

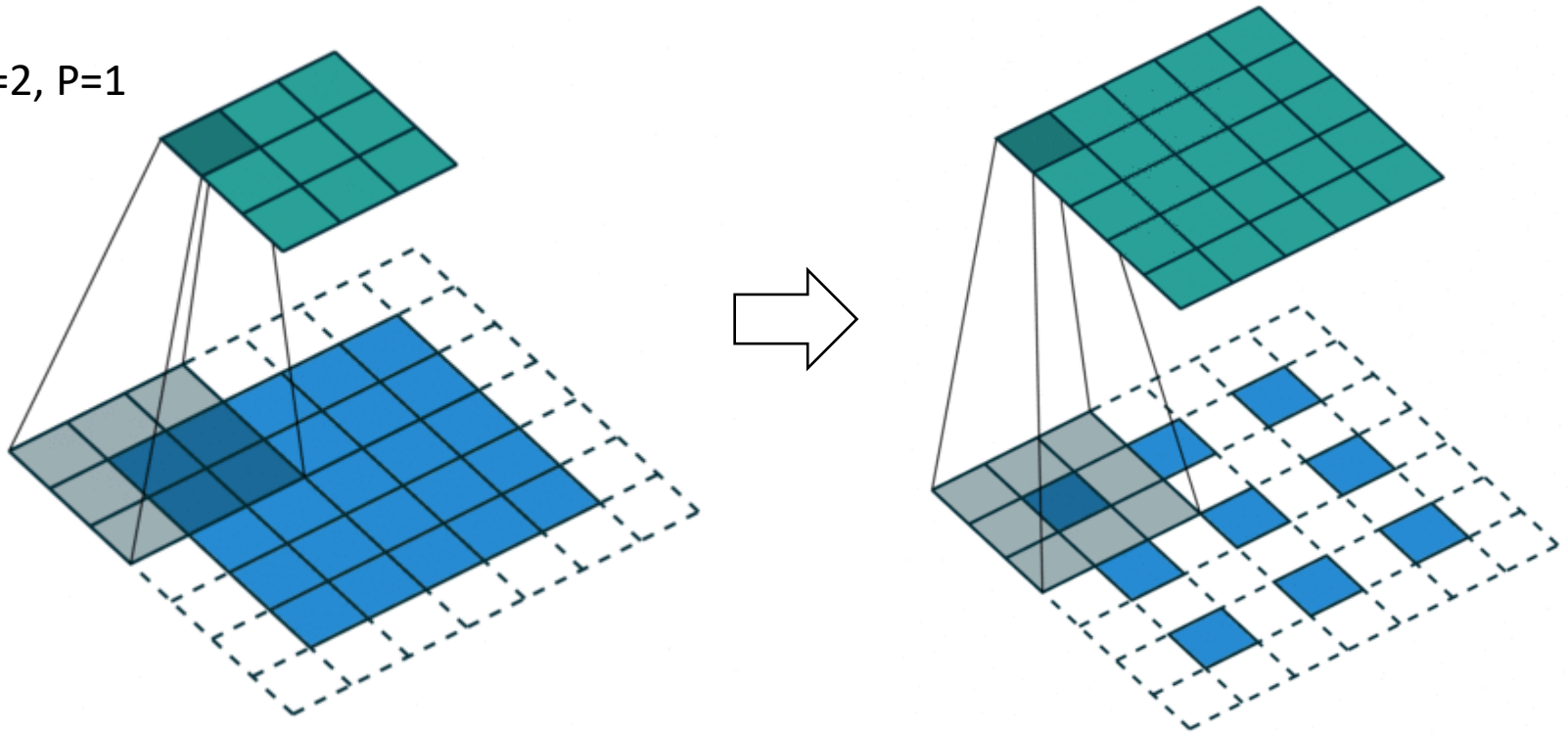


If you had **no padding in the forward** convolution, you need to **pad much when performing transposed** convolution

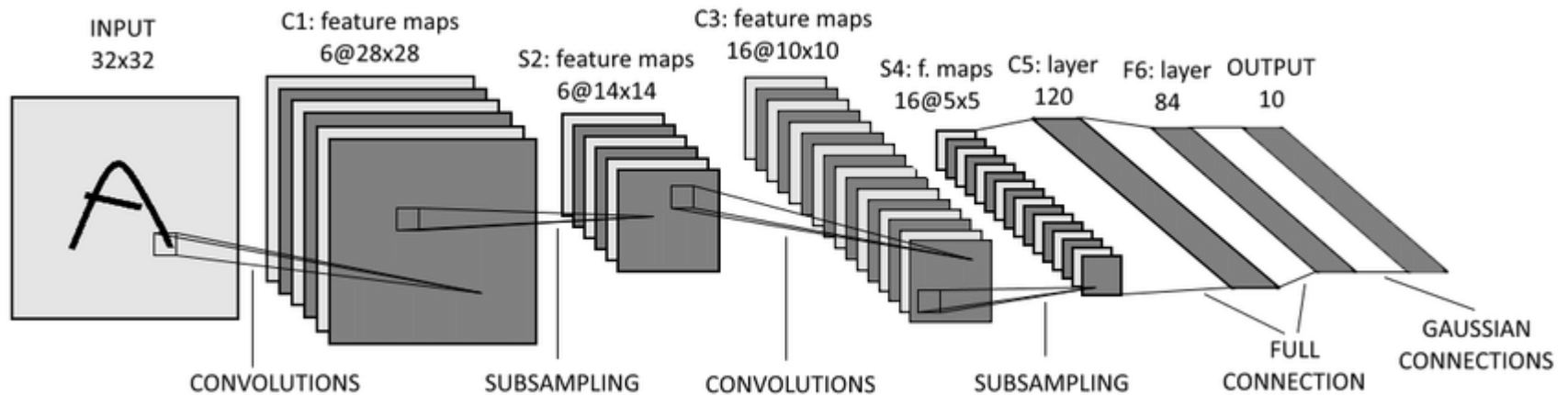
Deconvolution (Transposed Convolution)

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

$K=3, S=2, P=1$



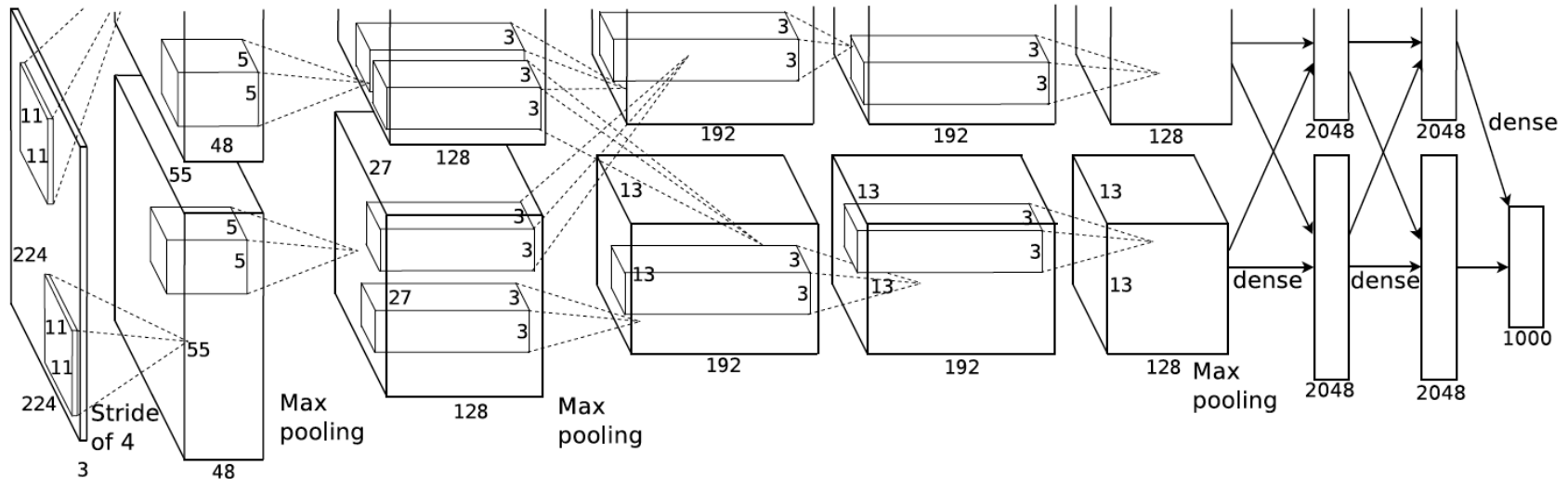
LeNet-5 (1989)



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

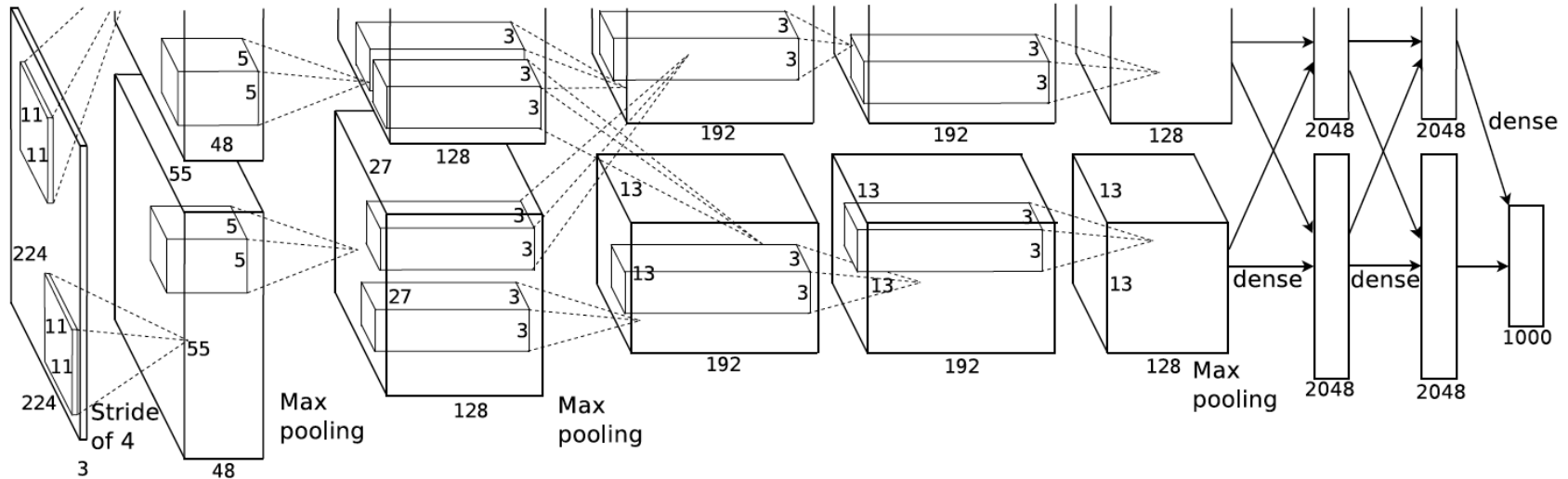
AlexNet (2012) - Architecture

ImageNet Top-5 : 15.4%



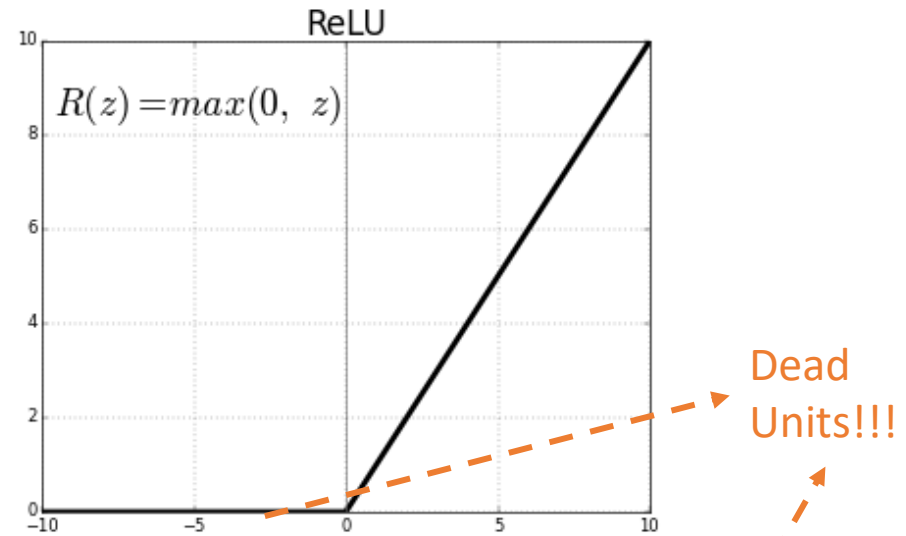
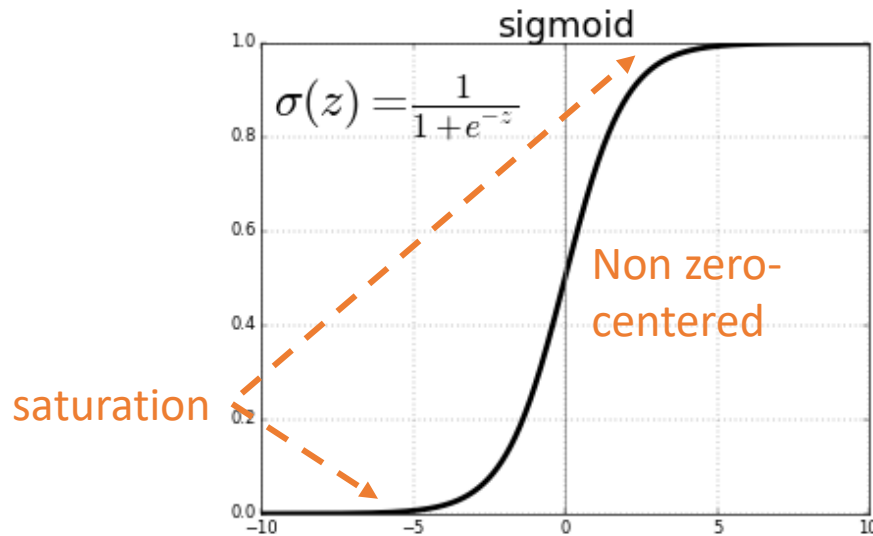
- RGB images **227x227x3**
- 5 convolutional layers + 3 fully connected layers
- Split into **two parts** (top/bottom) each on 1 GPU

AlexNet - Innovations



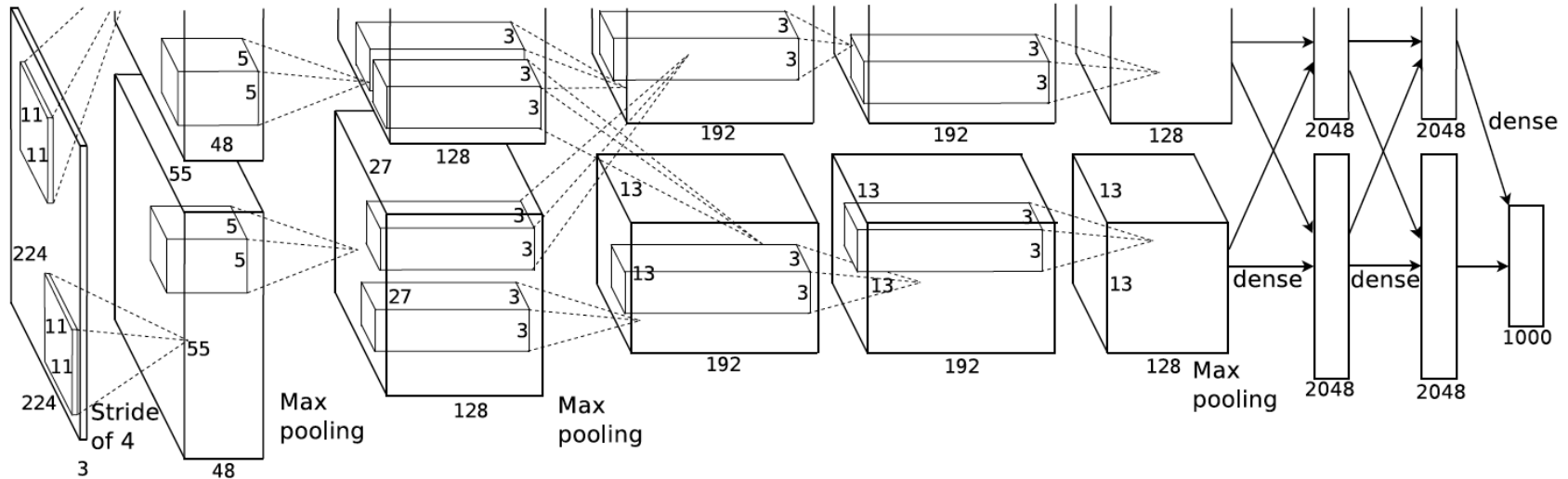
- Use heavy data **augmentation** (rotations, random crops, etc.)
- Introduced the use of **ReLU**
- Dense layers regularized by **dropout**

ReLU Nonlinearity



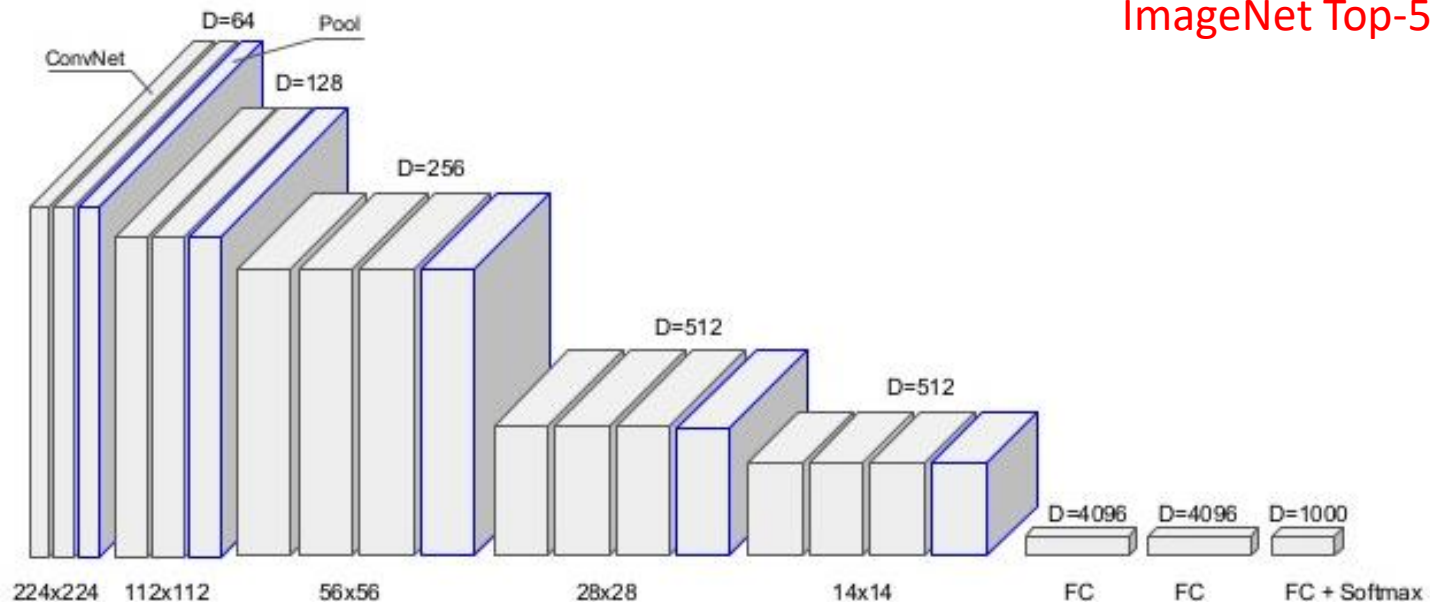
- ReLu help counteract **gradient vanish**
 - Sigmoid first derivative vanish as we increase or decrease z
 - ReLu first derivative is 1 when unit is active and 0 elsewhere
 - ReLu second derivative is 0 (no second order effects)
- Easy to **compute** (zero thresholding)
- Favors **sparsity**

AlexNet - Parameters



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

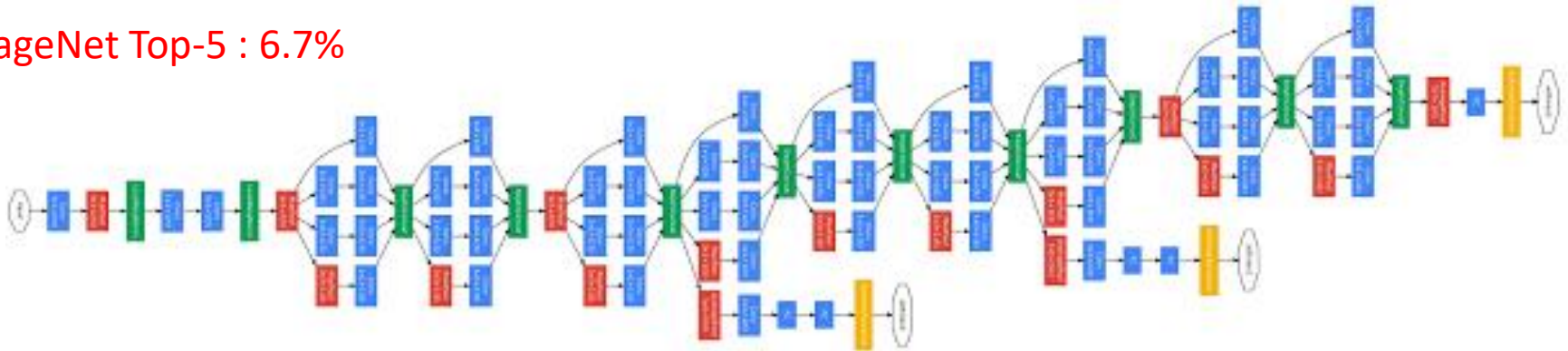
VGGNet – VGG16 (2014)



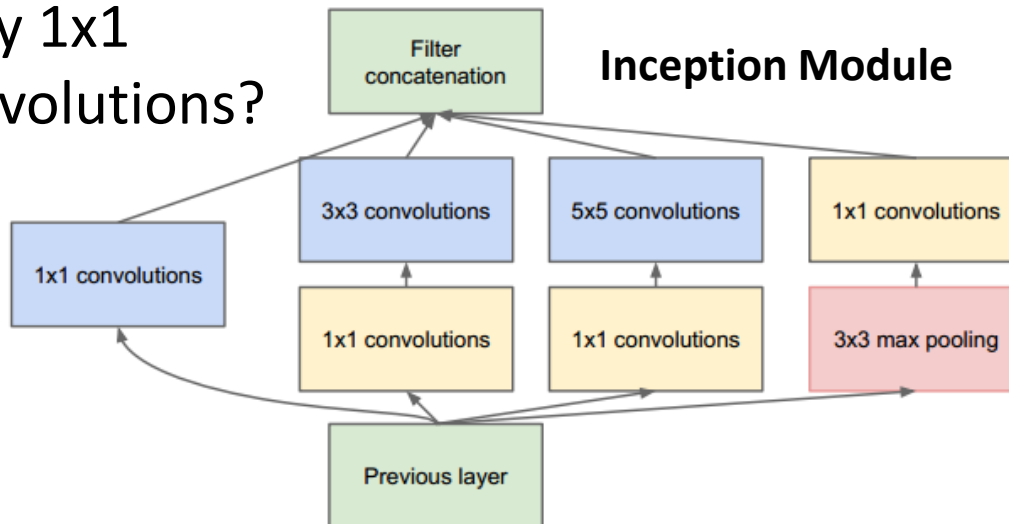
- Standardized convolutional layer
 - 3x3 convolutions with stride 1
 - 2x2 max pooling with stride 2 (not after every convolution)
- Various configuration analysed, but best has
 - 16 Convolutional + 3 Fully Connected layers
 - About 140 millions parameters (85% in FC)

GoogLeNet (2015)

ImageNet Top-5 : 6.7%

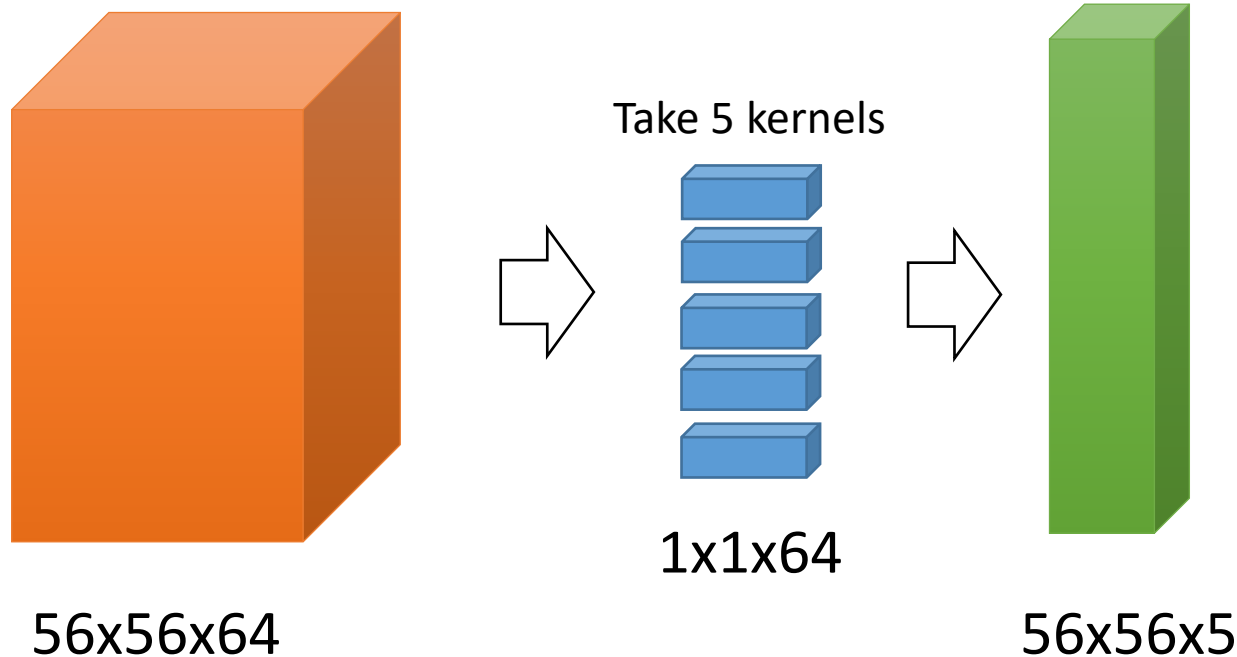


Why 1x1
convolutions?



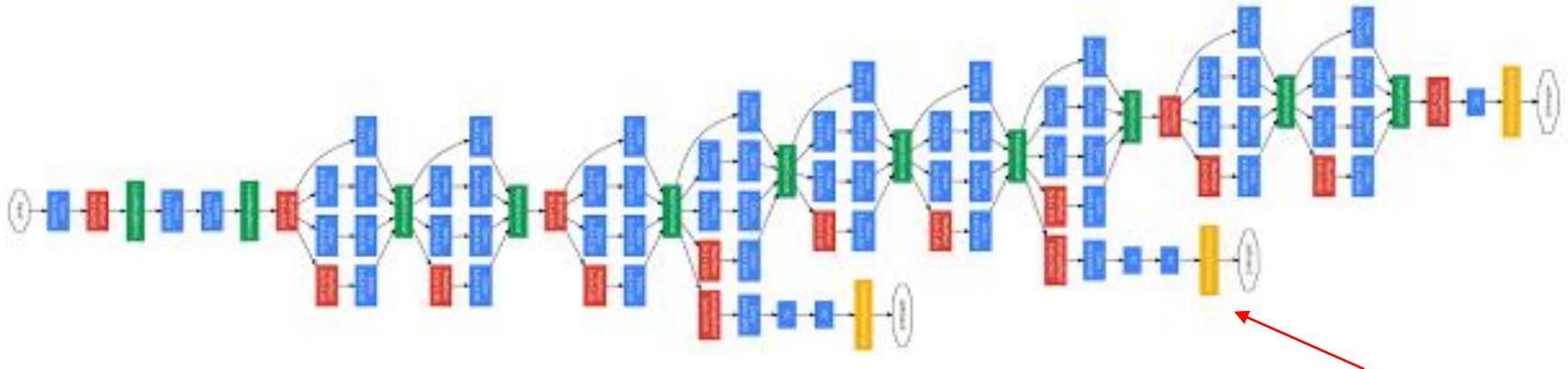
- Kernels of **different size** to capture details at varied scale
- Aggregated before sending to next layer
- Average **pooling**
- No fully connected layers

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



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

Auxiliary outputs
to inject gradients
at deeper layers

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)

$$\mu_b = \frac{1}{N_b} \sum_{i=1}^{N_b} x_i$$
$$\sigma_b^2 = \frac{1}{N_b} \sum_{i=1}^{N_b} (x_i - \mu_b)^2$$

$$\hat{x}_i = \frac{x_i - \mu_b}{\sqrt{\sigma_b^2 + \epsilon}}$$

Normalization

$$y = \gamma \hat{x}_i + \beta$$

Scale and shift

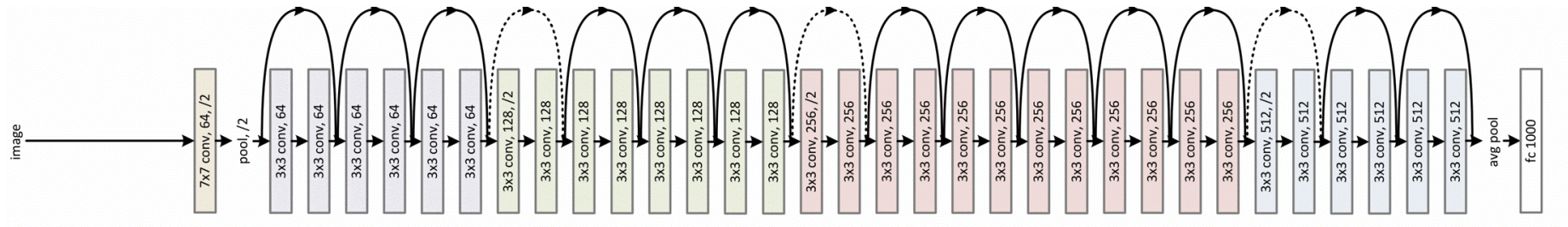
Trainable linear transform potentially allowing to cancel unwanted zero-centering effects (e.g. sigmoid)

Need to backpropagate through this!

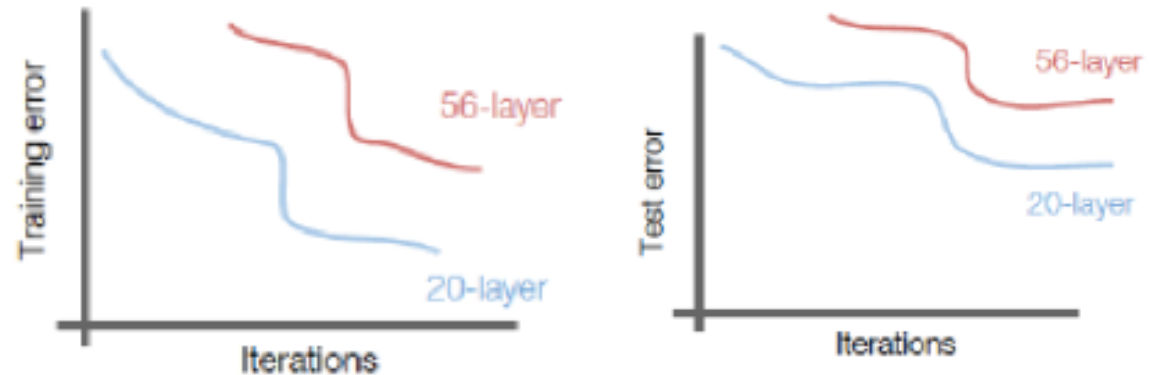
ResNet (2015)

ImageNet Top-5 : 3.57%

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

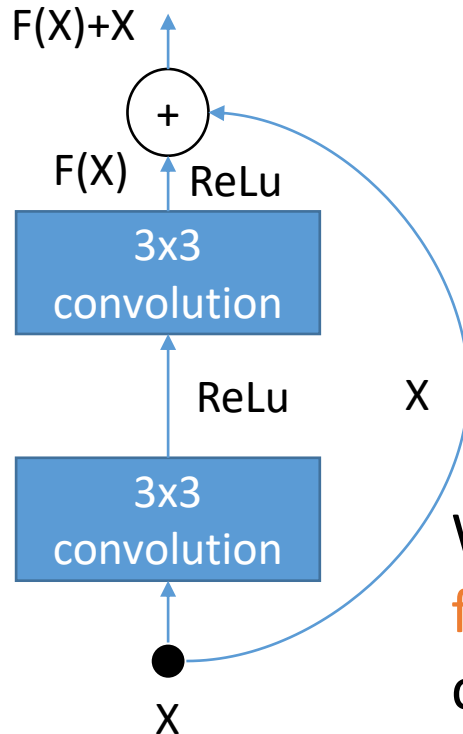
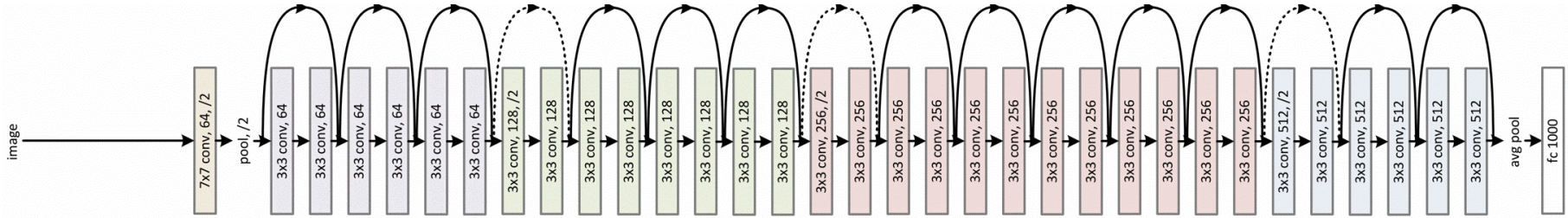


Why wasn't this working before?



Gradient vanishes when backpropagating too deep!

ResNet Trick

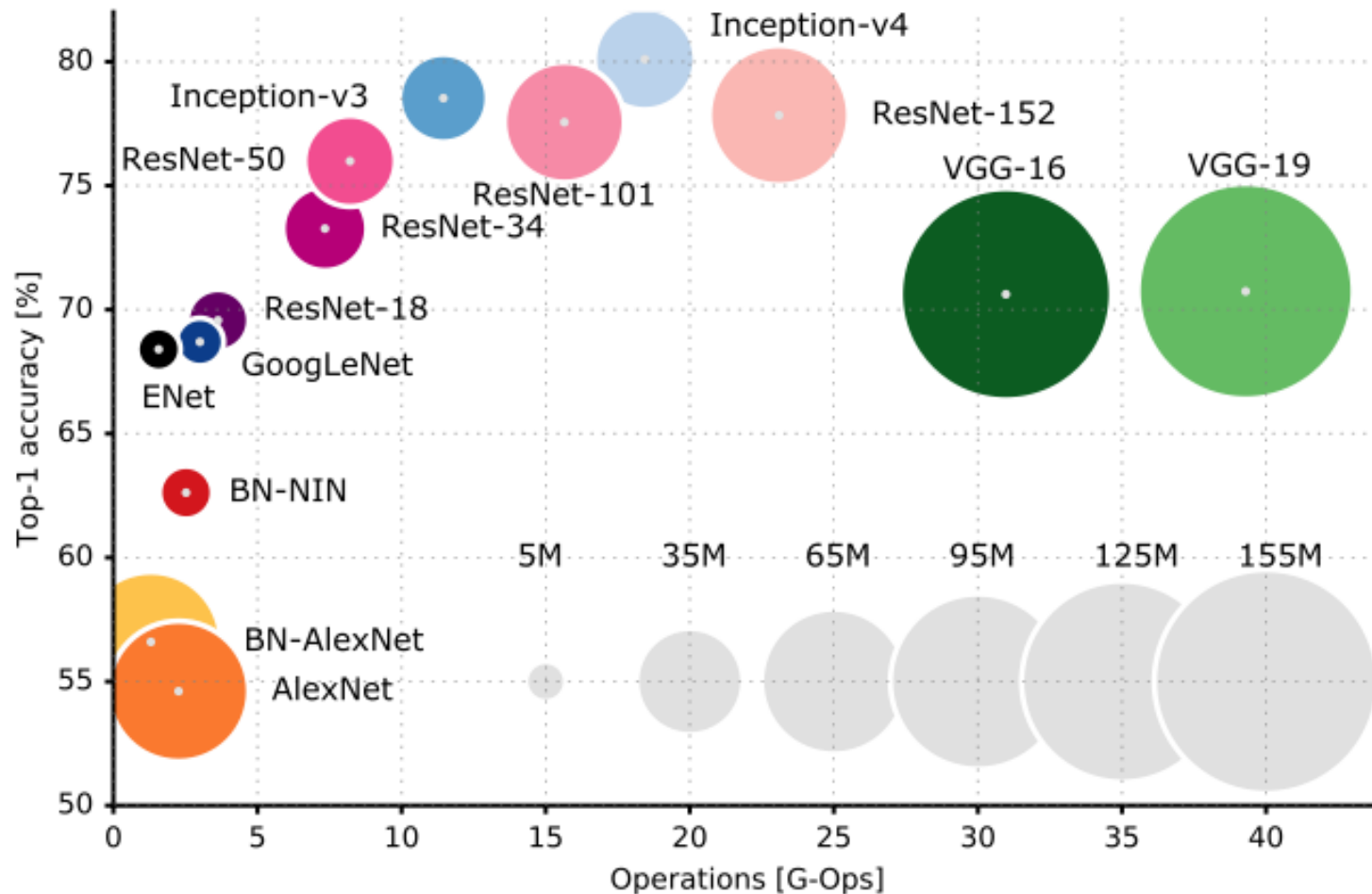


Residual
block

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

CNN Architecture Evolution



Understanding CNN Embedding



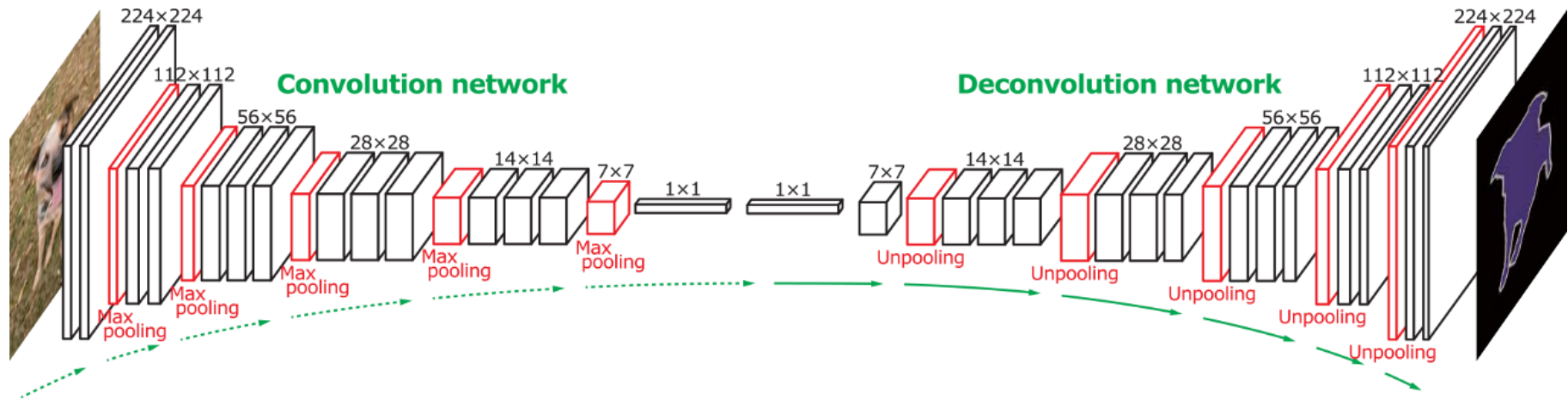
tSNE projection of
AlexNet last
hidden dense
layer

<https://cs.stanford.edu/people/karpathy/cnnembed/>

Interpreting Intermediate Levels

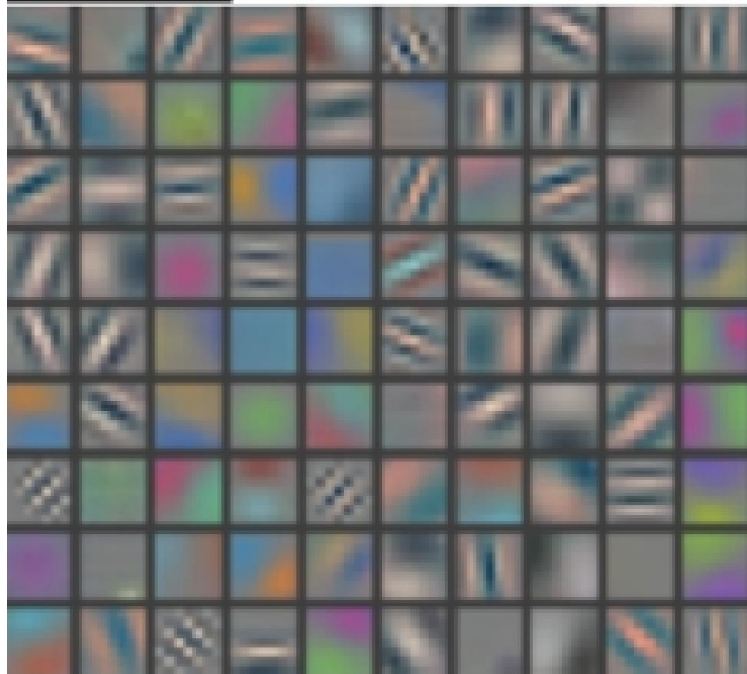
- What about the information captured in convolutional layers?
- Visualize kernel weights (filters)
 - Naïve approach
 - Works only for early convolutional layers
- Map the **activation of the convolutional kernel back in pixel space**
 - Requires to reverse convolution
 - **Deconvolution**

Deconvolutional Network (DeConvNet)



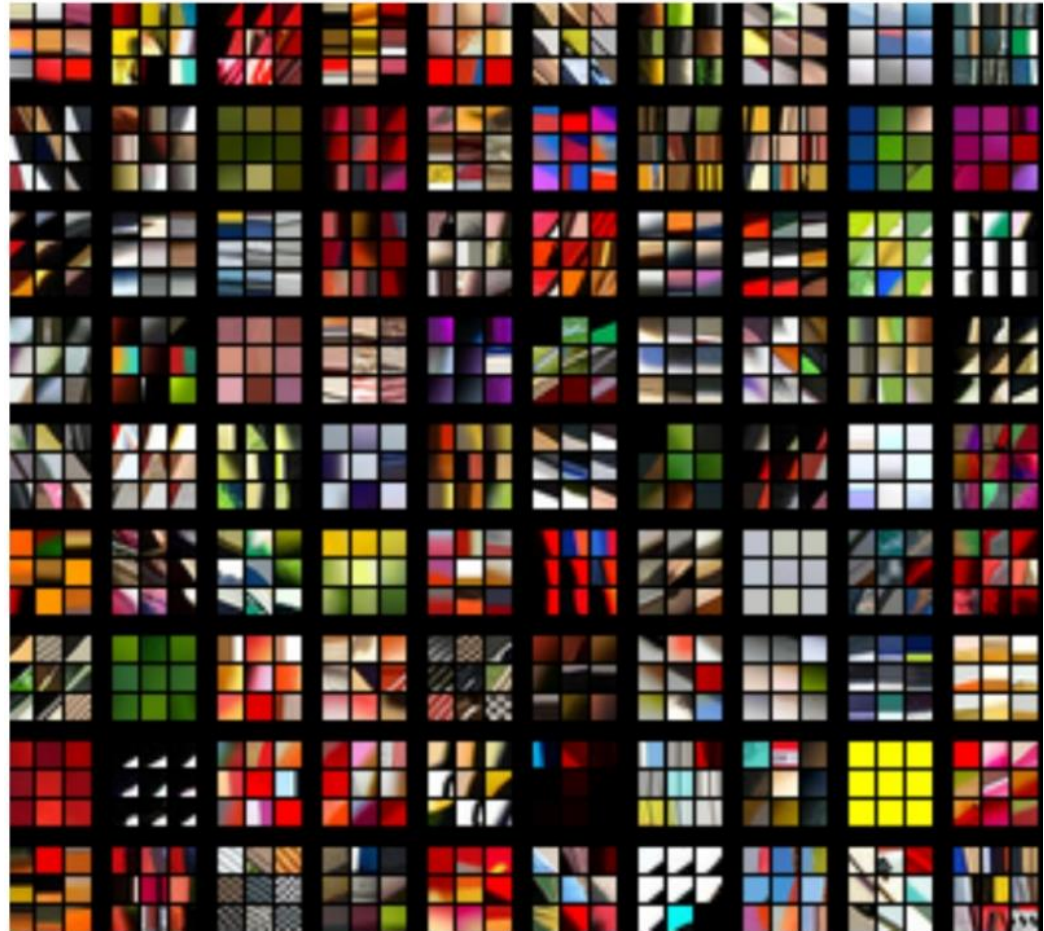
- Attach a DeConvNet to a **target layer**
- Plug an input and forward propagate activations until layer
- Zero activations of **target neuron**
- Backpropagate on the DeConvNet and see what parts of the **reconstructed image are affected**

Filters & Patches – Layer 1

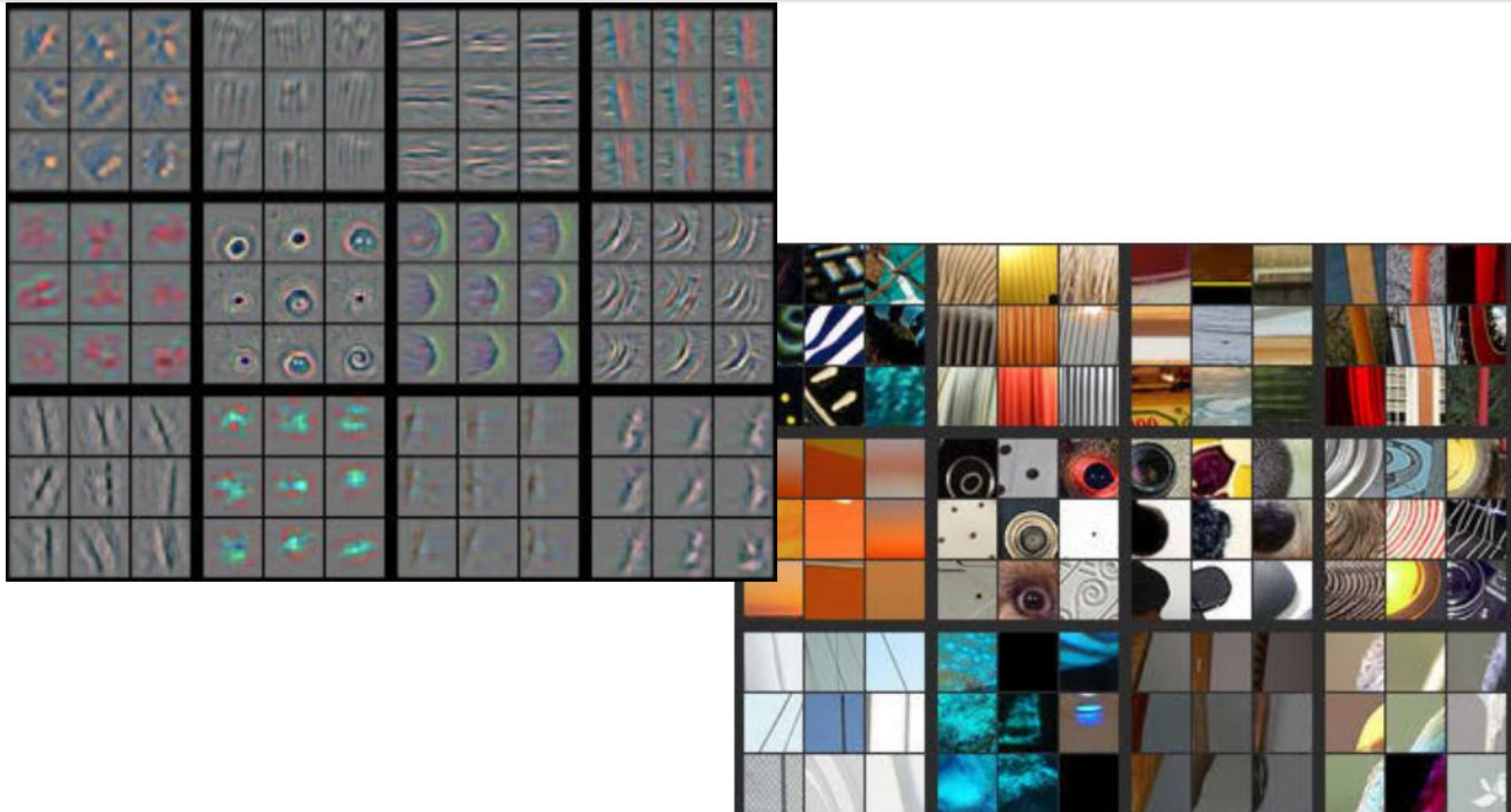


Reconstructed filters in pixel space

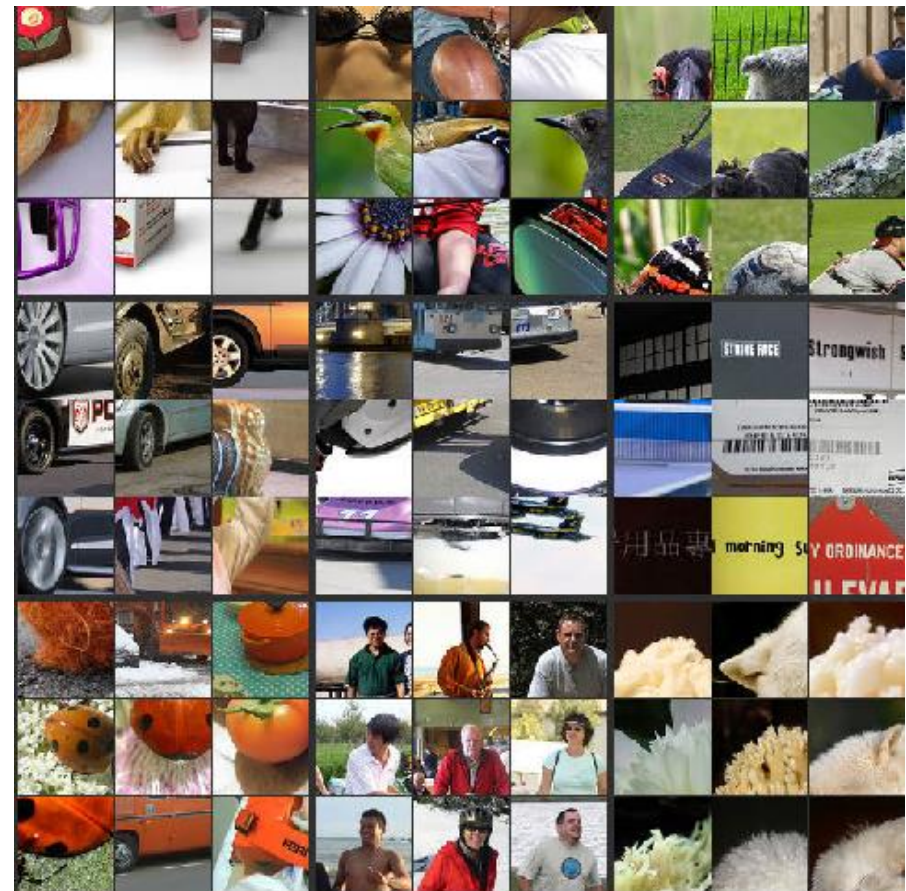
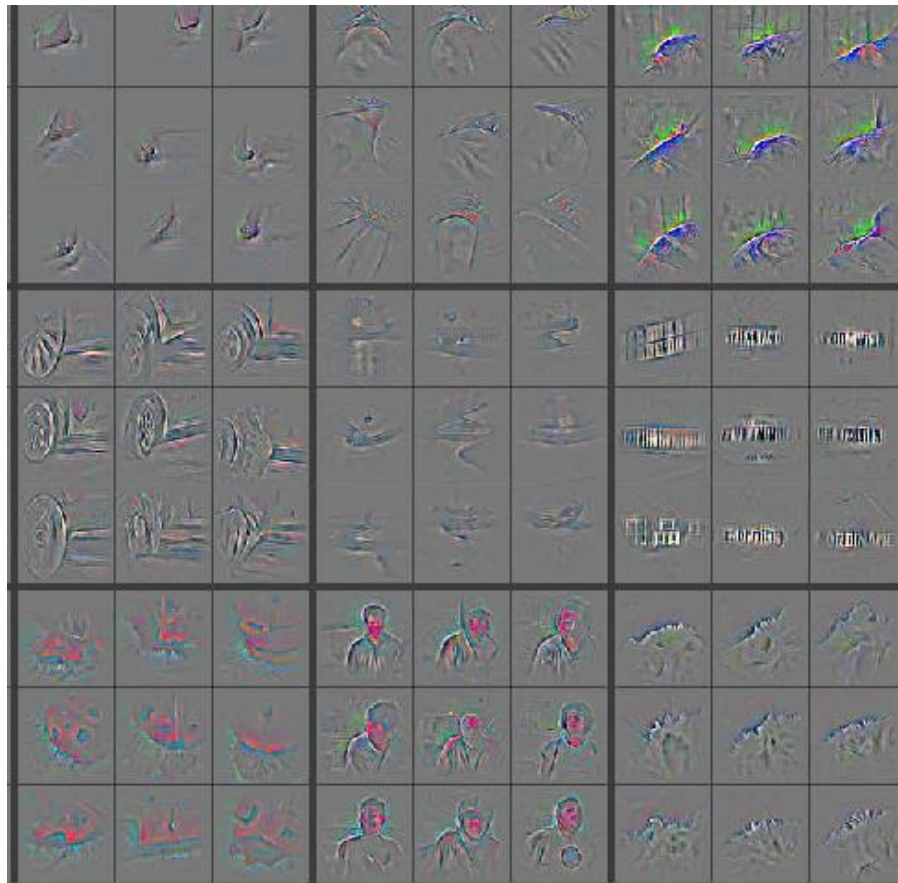
Corresponding top-9 image patches



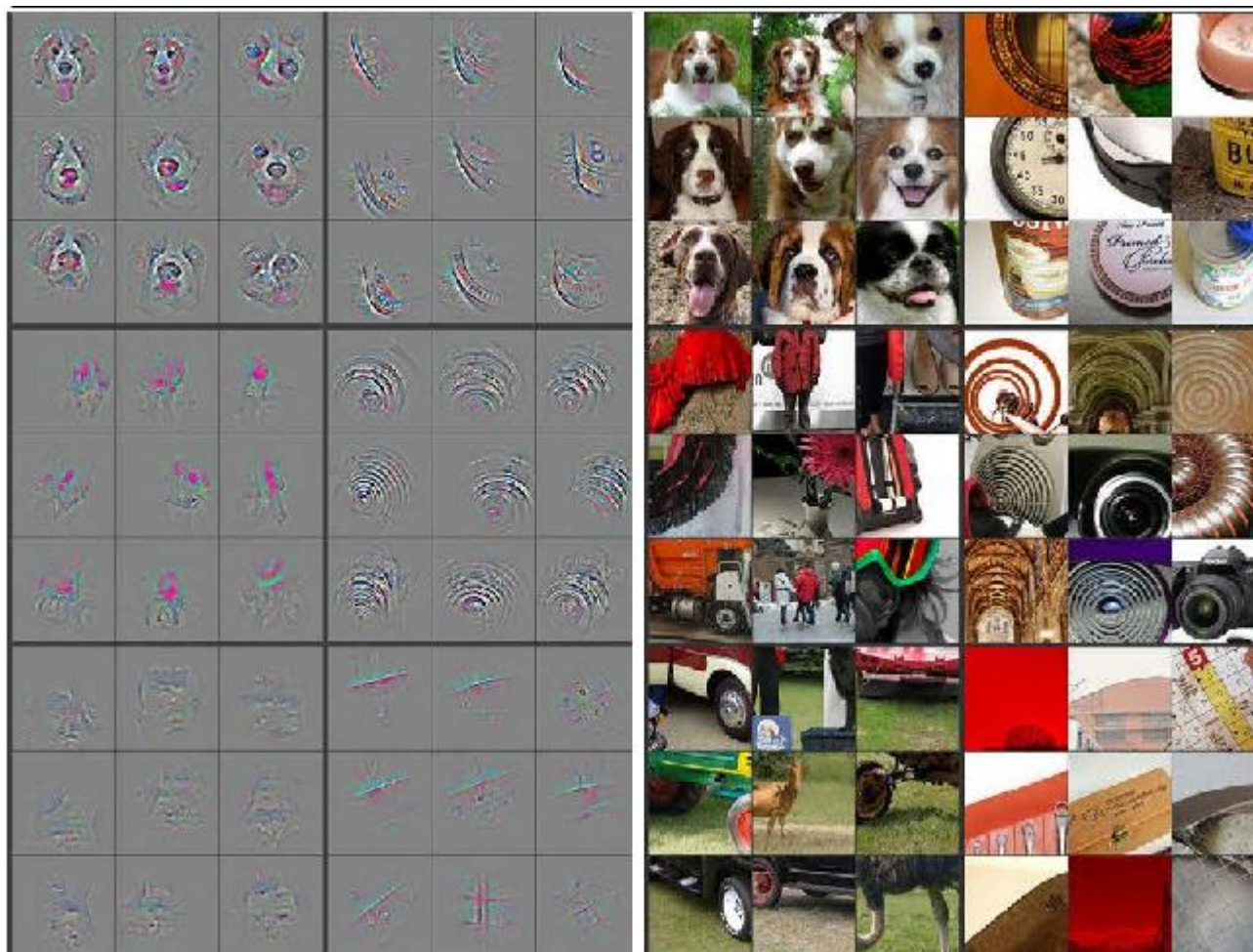
Filters & Patches – Layer 2



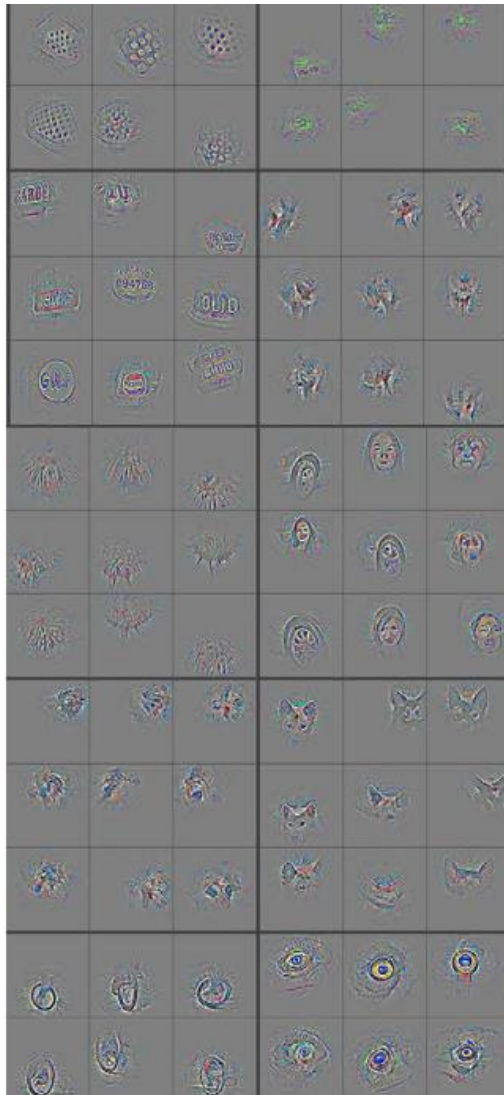
Filters & Patches – Layer 3



Filters & Patches – Layer 4



Filters & Patches – Layer 5

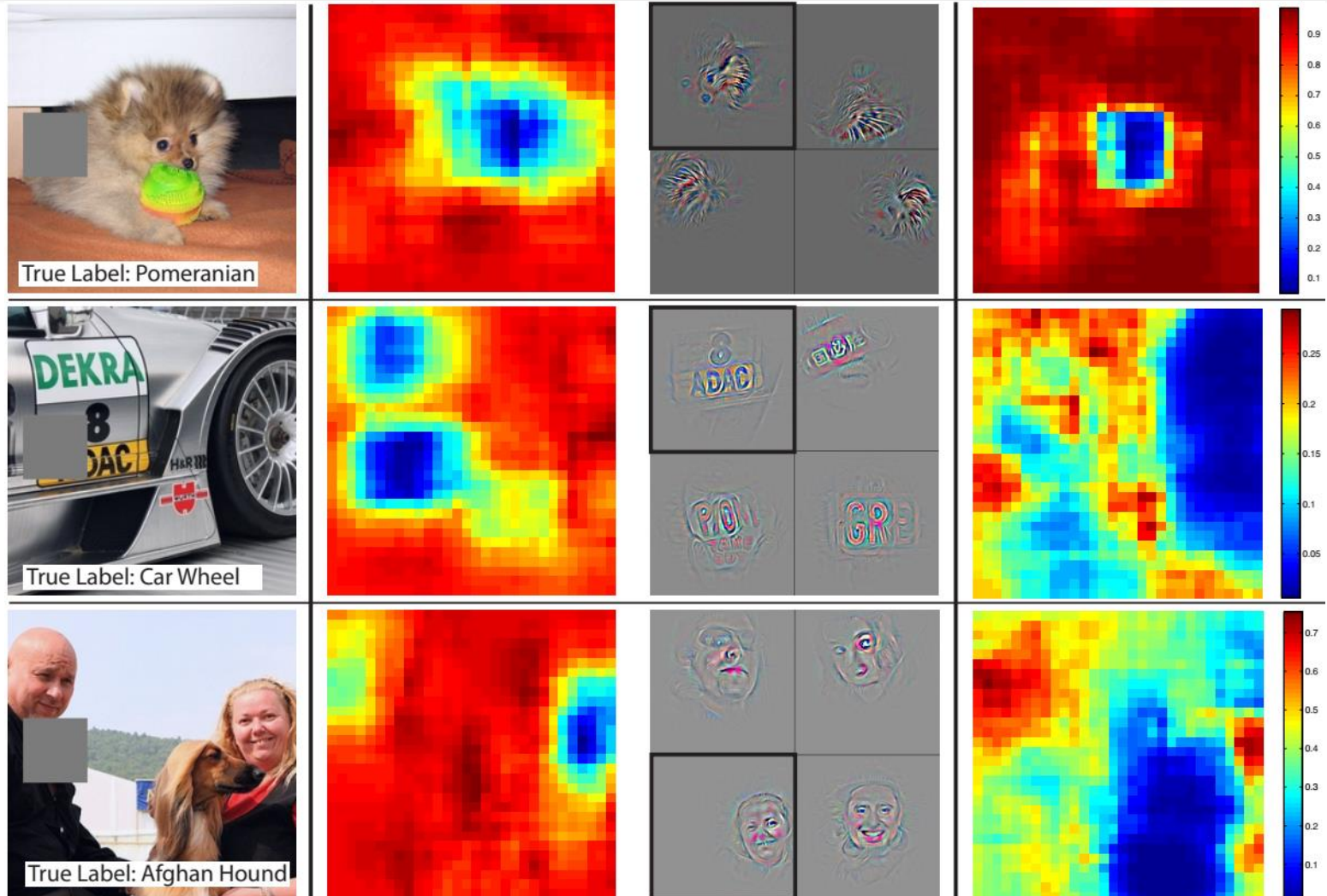


Zeiler&Fergus, Visualizing and Understanding
Convolutional Networks, ICML 2013

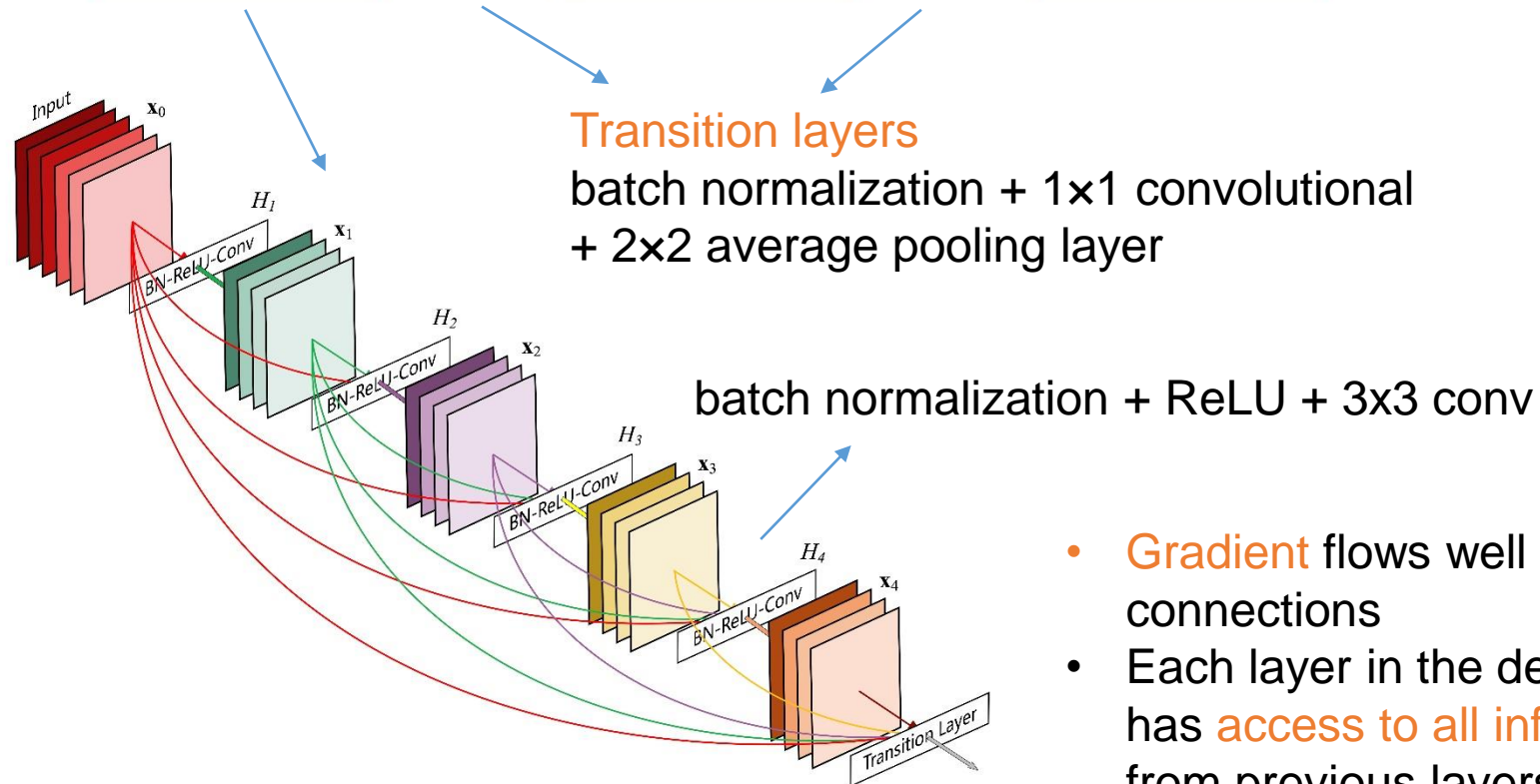
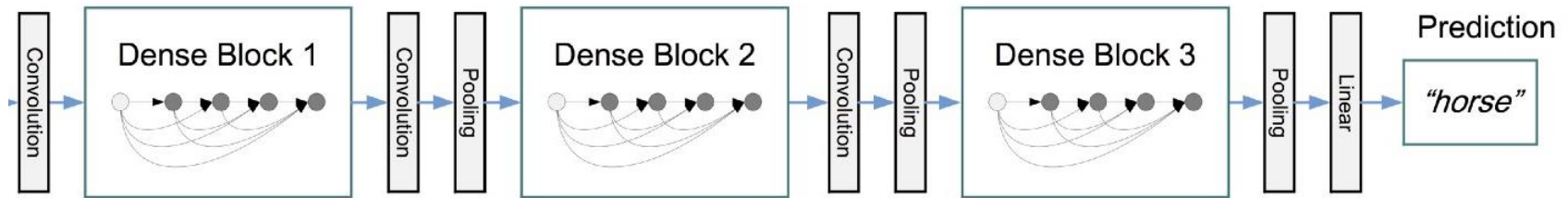
Occlusions

- Measure what happens to feature maps and object classification if we occlude part of the image
- Slide a grey mask on the image and project back the response of the best filters using deconvolution

Occlusions



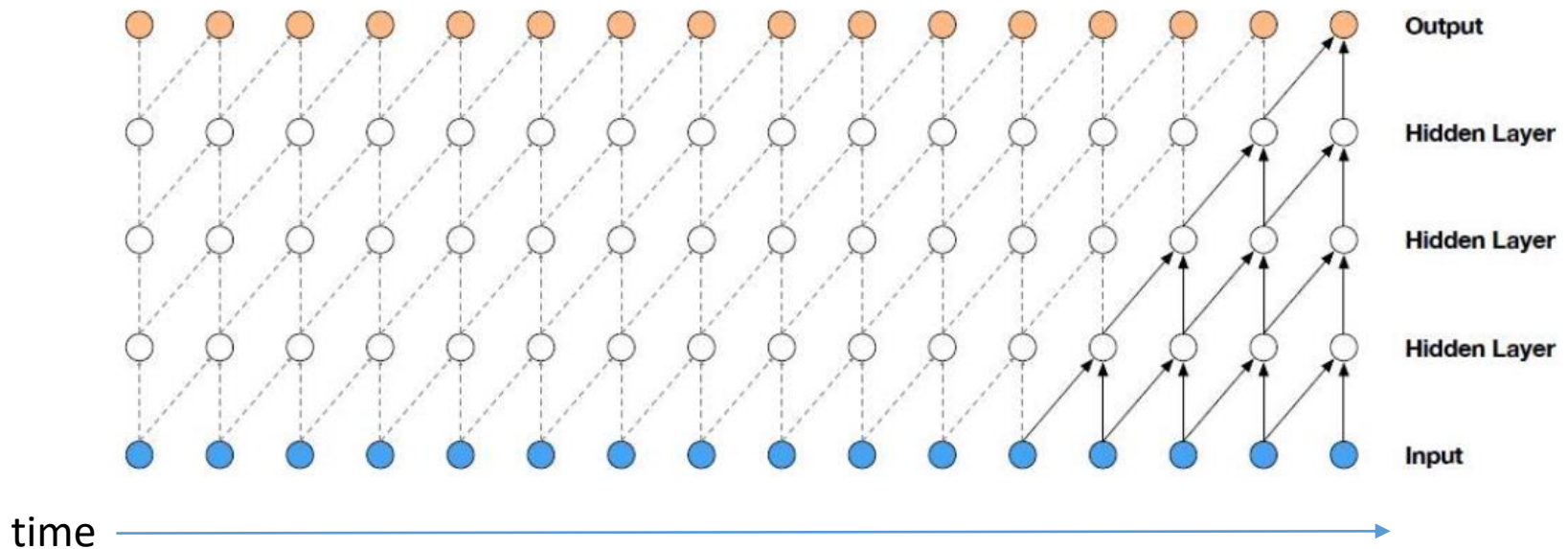
Dense CNN



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

Causal Convolutions

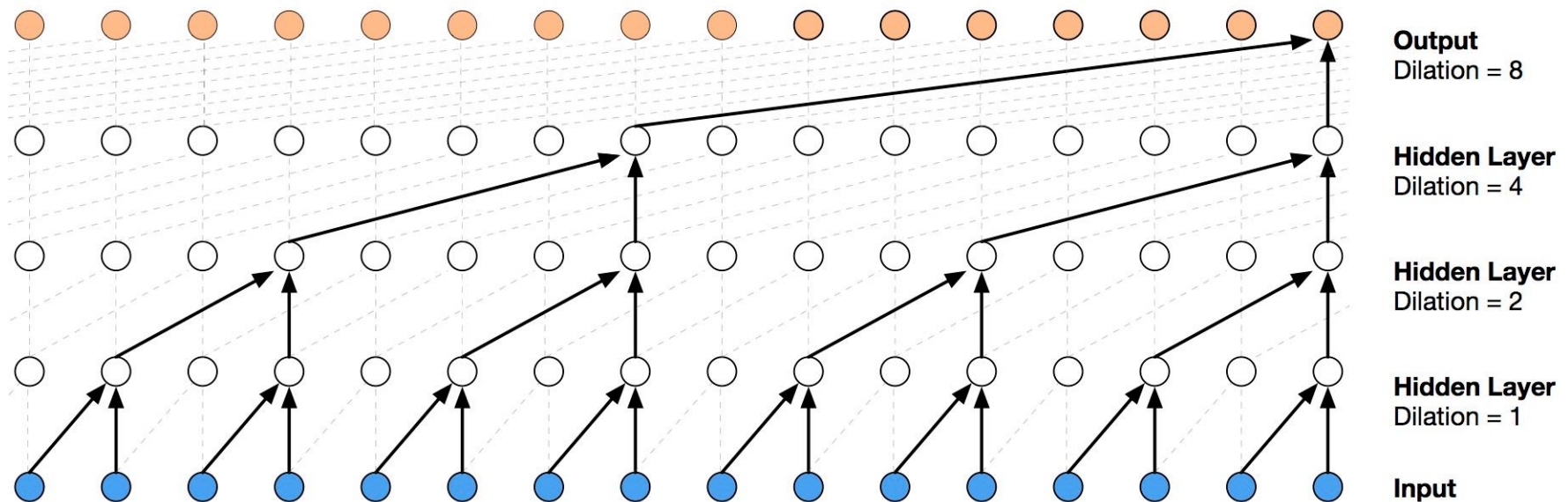
Preventing a convolution from allowing to see into the future...



Problem is the **context size grows slow** with depth

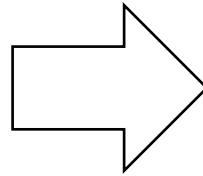
Causal & Dilated Convolutions

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



Similar to striding, but size is preserved

Semantic Segmentation

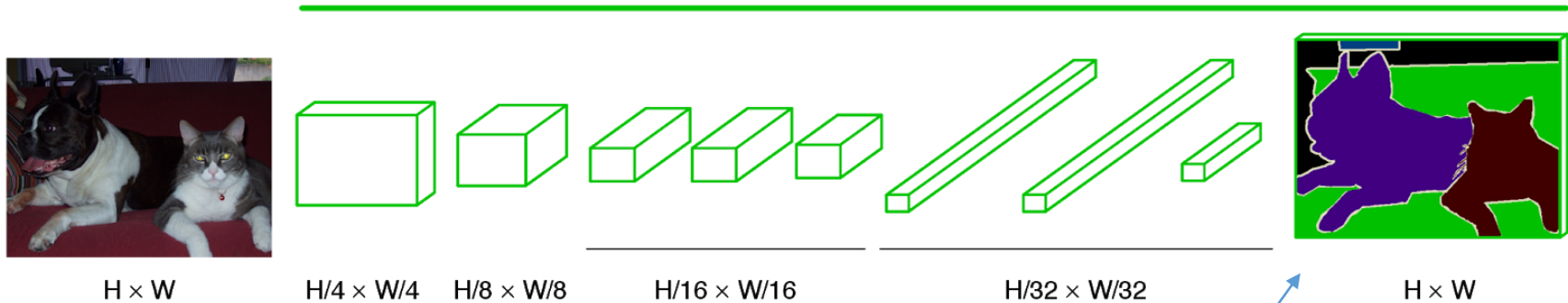


Traditional CNN cannot be used for this task due to the downsampling of the striding and pooling operations

Fully Convolutional Networks (FCN)

Shelhamer et al, Fully Convolutional Networks for Semantic Segmentation, PAMI 2016

convolution

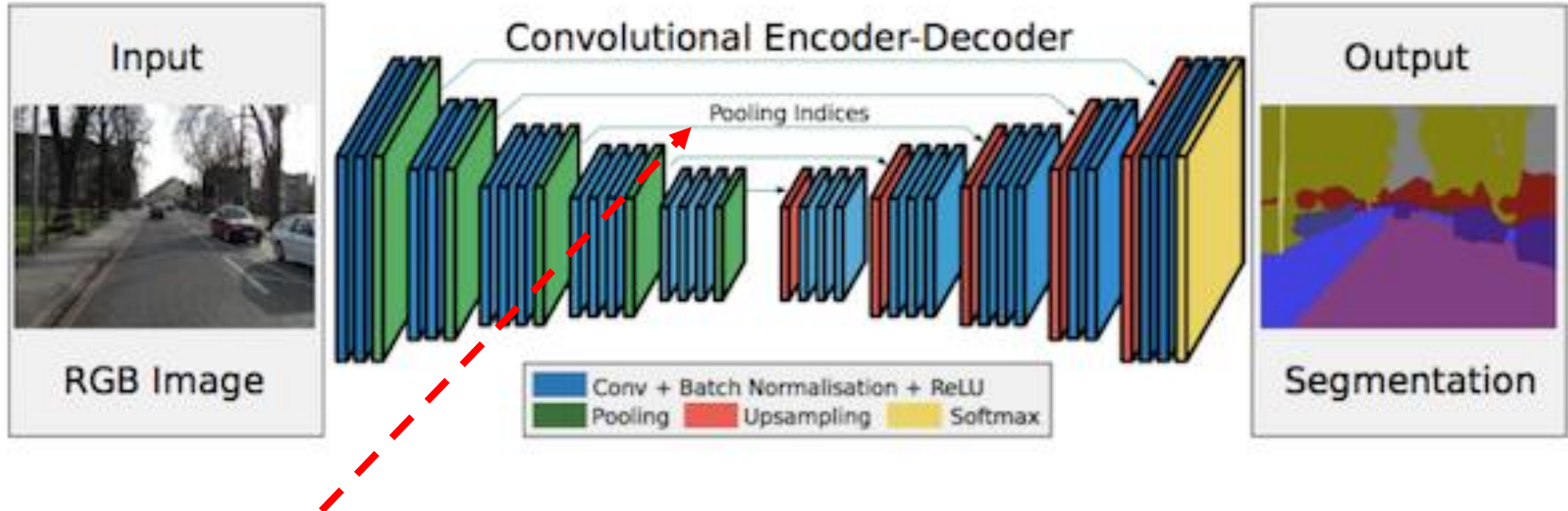


Convolutional part to
extract interesting
features at various scales

Learn an upsampling function of
the fused map to generate the
semantic segmentation map

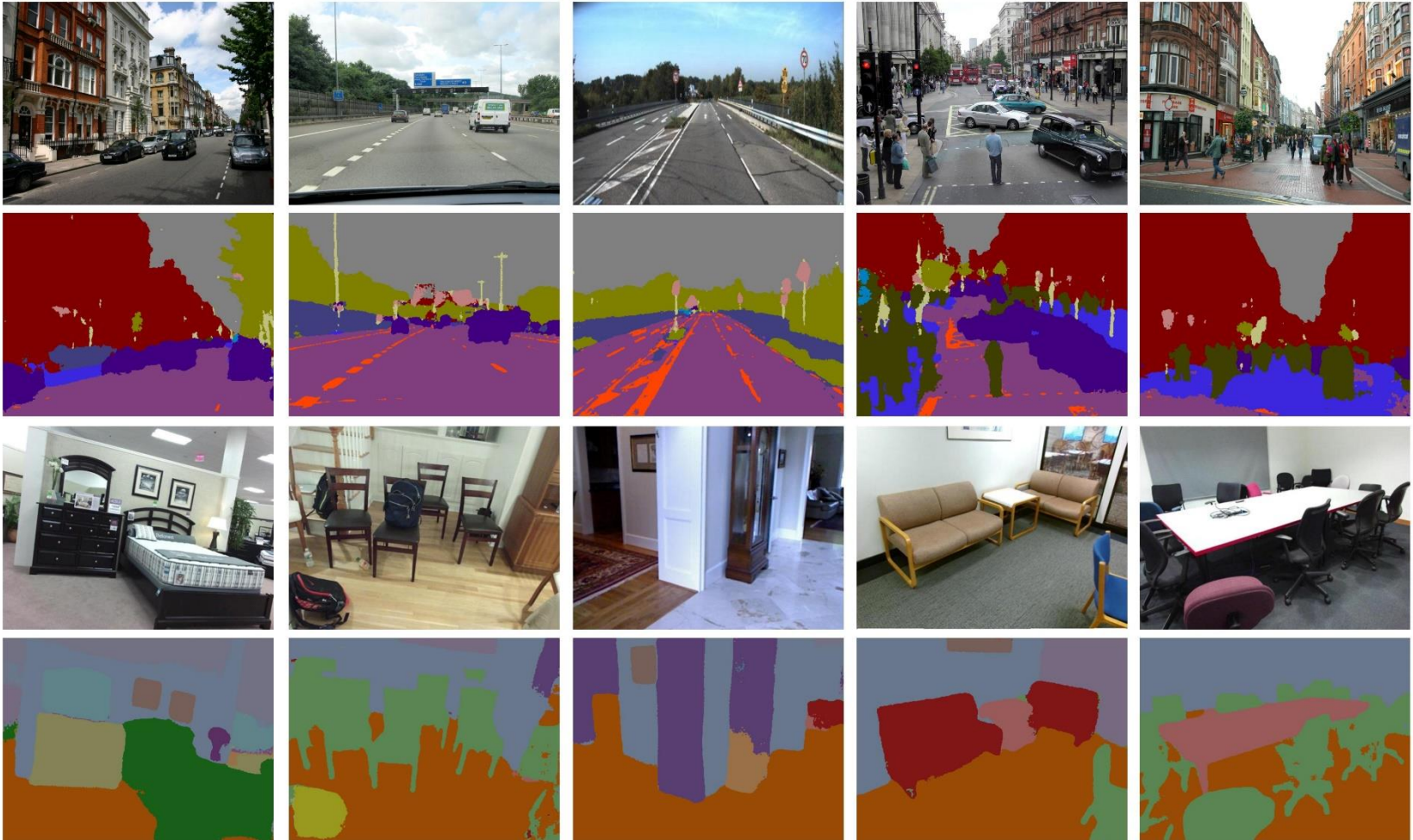
Fuse information from feature maps
of different scale

Deconvolution Architecture



Maxpooling indices transferred to decoder to improve the segmentation resolution.

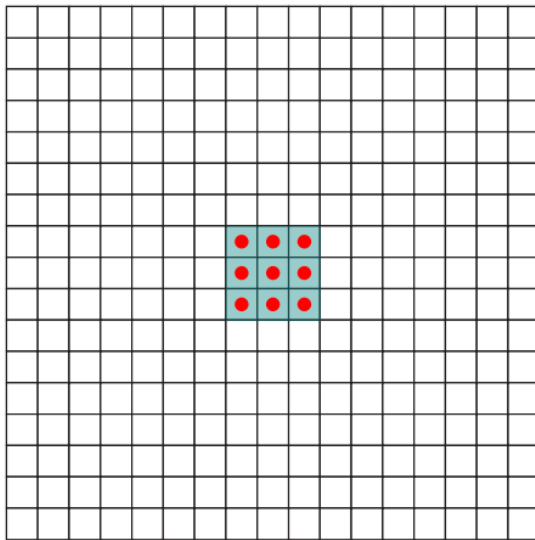
SegNet Segmentation



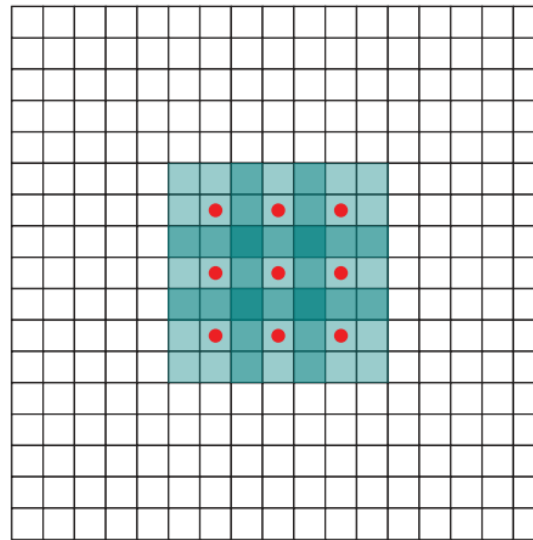
Demo here: <http://mi.eng.cam.ac.uk/projects/segnet/>

Use Dilated Convolutions

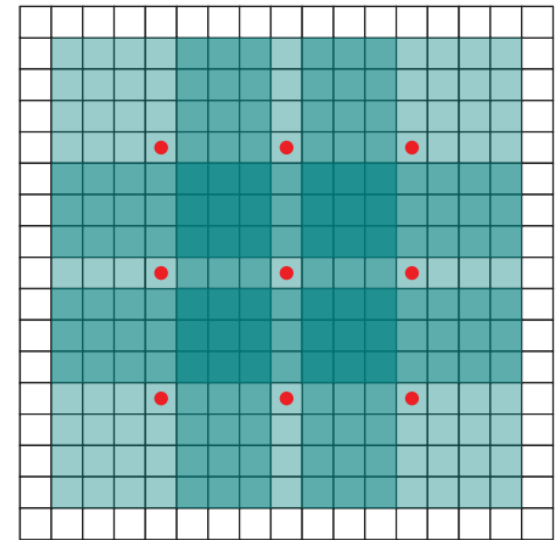
Always perform 3x3 convolutions with no pooling at each level



Level 1



Level 2



Level 3

Context increases without

- Pooling (changes map size)
- Increasing computational complexity

Segmentation by Dilated CNN

Dilated CNN

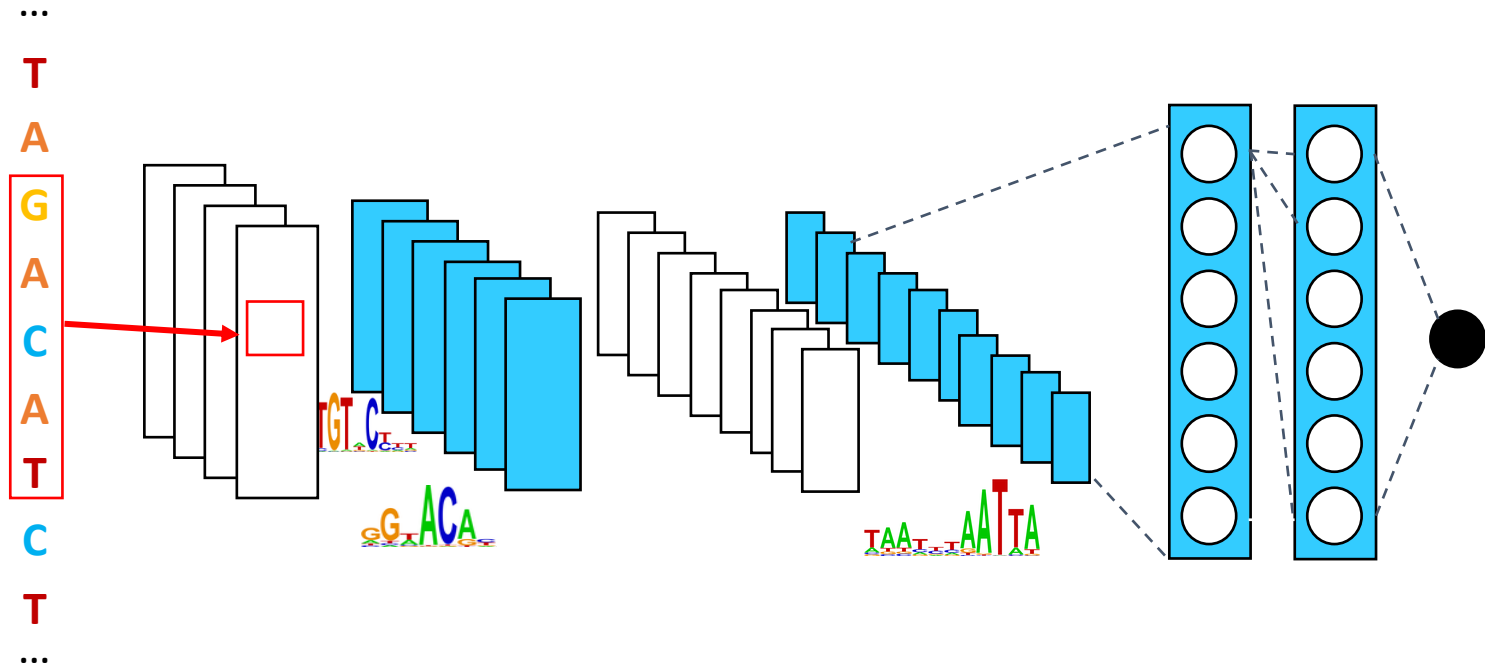
GT

Dilated CNN

GT



CNN & Genomic Sequences

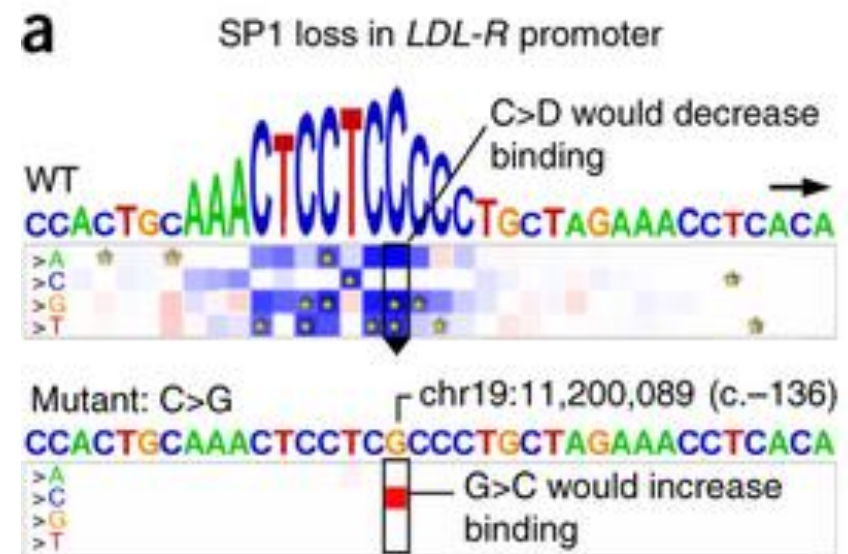


1D convolutions throughout the input sequence

- Trained to respond to task-specific motifs
- Applied to small sequence regions

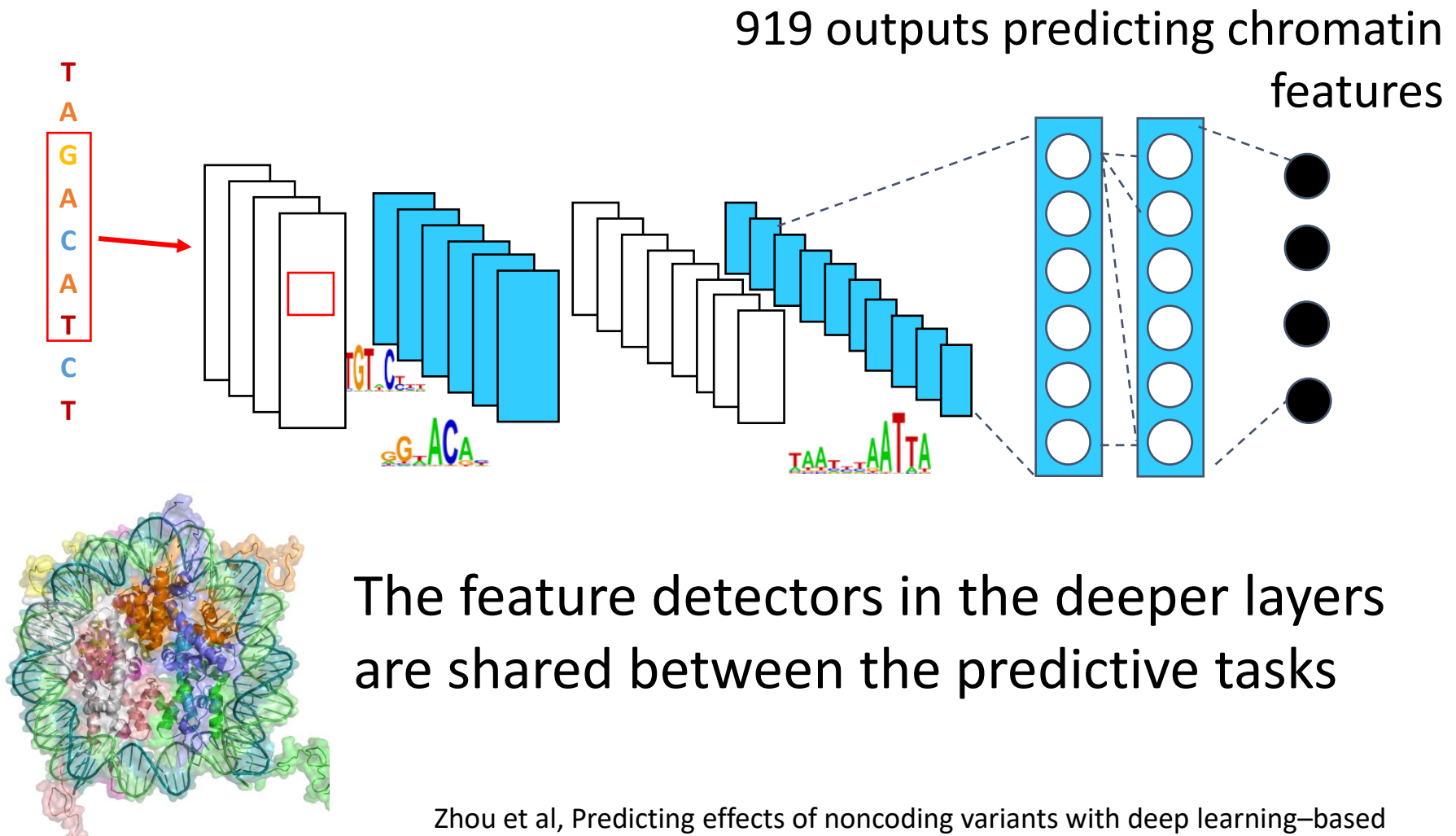
DeepBind

- 927 CNN models predicting a **binding score** for **transcription factors** and **RNA-binding proteins**
 - Score new sequences
 - Assess mutations that deplete/increase binding score
- Use **convolution visualization** to interpret results of CNN training



Mutation Maps

DeepSea



Zhou et al, Predicting effects of noncoding variants with deep learning-based sequence model. *Nature methods*. 2015 - <http://deepsea.princeton.edu>

Software

- CNN are supported by any deep learning framework (TF, Torch, Pytorch, MS Cognitive TK,...)
- Caffe was one of the **initiators** and basically built around CNN
 - Introduced **protobuffer** network specification
 - ModelZoo of **pretrained models** (LeNet, AlexNet, ...)
 - Support for **GPU**

Caffe Protobuffer

```
name: "LeNet"
layer {
  name: "data"
  type: "Input"
  ...
  input_param { shape: { dim: 64 dim: 1 dim: 28 dim: 28 } }
}
layer {
  name: "conv1"
  type: "Convolution"
  bottom: "data"
  ...
  convolution_param {
    num_output: 20
    kernel_size: 5
    stride: 1
    weight_filler {
      type: "xavier"
    }
  }
}
```

Software

- CNN are supported by any deep learning framework (TF, Torch, Pytorch, MS Cognitive TK, Intel OpenVino)
- Caffe was one of the **initiators** and basically built around CNN
 - Introduced **protobuf** network specification
 - ModelZoo of **pretrained models** (LeNet, AlexNet, ...)
 - Support for **GPU**
- Caffe2 is Facebook's extensions to Caffe
 - Less CNN oriented
 - Support from **large scale to mobile nets**
 - More **production oriented** than other frameworks

Other Software

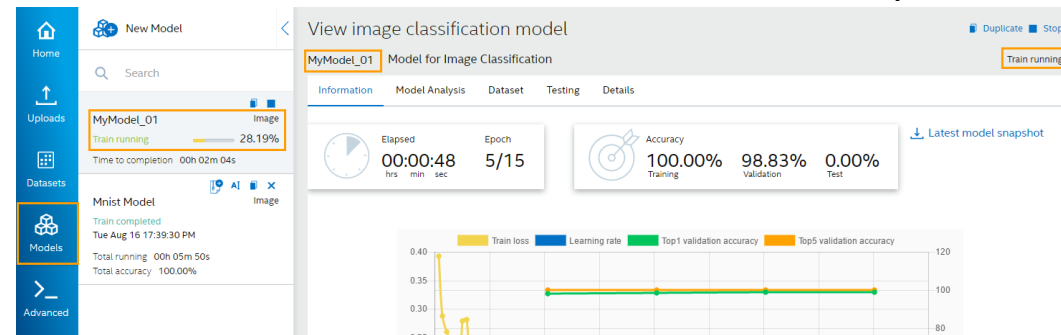
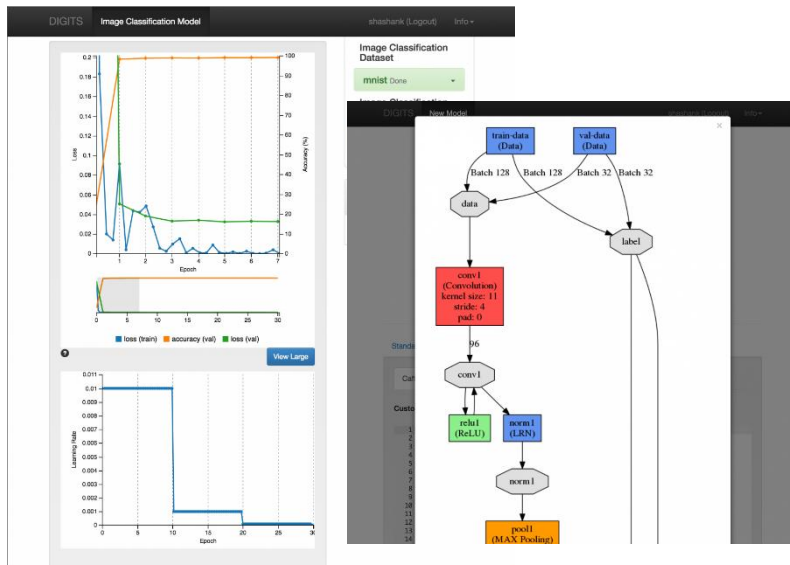
- Matlab distributes its **Neural Network Toolbox** which allows importing pretrained models from Caffe and Keras-TF
- Matconvnet is an unofficial Matlab library **specialized for CNN development** (GPU, modelzoo, ...)
- Want to have a **CNN in your browser?**
 - Try ConvNetJS
(<https://cs.stanford.edu/people/karpathy/convnetjs/>)

GUIs

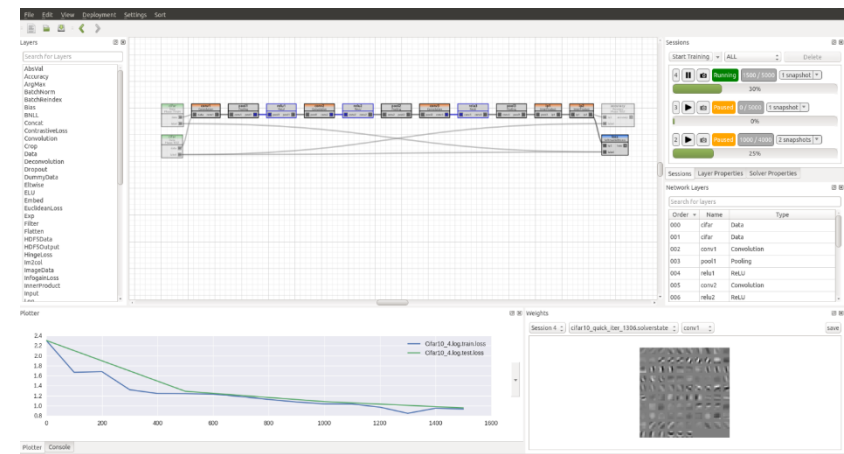
Major hardware producers have GUI and toolkits wrapping Caffe, Keras and TF to play with CNNs

Intel OpenVino

NVIDIA Digits



Barista



Plus
others...

Take Home Messages

- Key things
 - **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**
 - **Bypass connections** allow to go deeper
- Dilated (**à trous**) convolutions
- You can **use CNN outside** of machine vision

Next Lecture

Deep Autoencoders

- Autoencoders and dimensionality reduction
- Neural autoencoders (sparse, denoising contractive)
- Deep neural autoencoders and pretraining
- Deep generative-based autoencoders
- Visualization and multi-modal data fusion with autoencoders