UNIVERSITY
OF OSLO

IN4310 Convolutional Neural Networks

Ali Ramezani-Kebrya

ali@uio.no

Key Learning Goals [1, 2]

Convolution in deep learning, i.e, a sliding window of inner products

Local pattern detection and translation invariance

Reduction of # parameters compared to fully connected networks

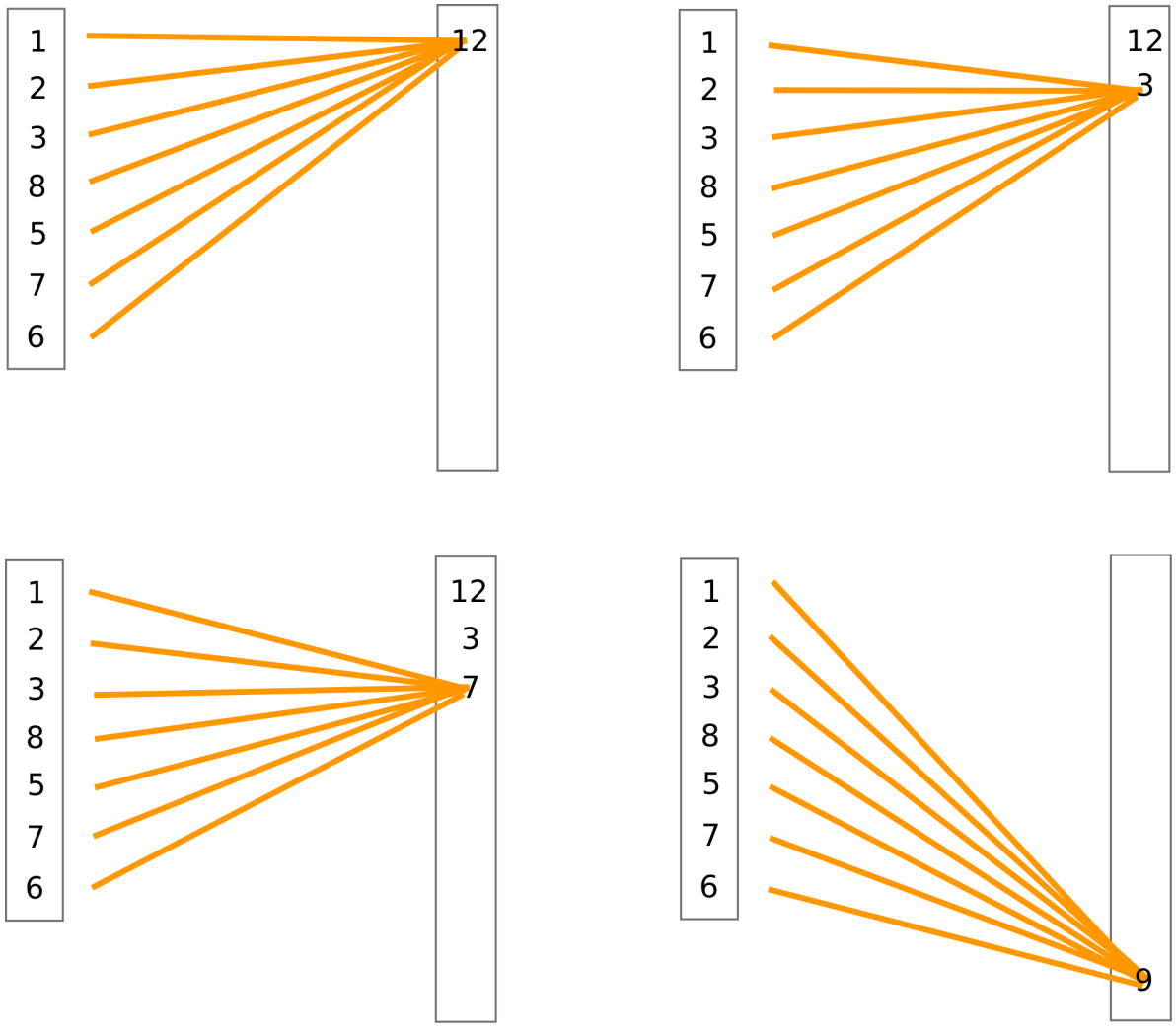
Hierarchical pattern recognition as layers go deeper

Effect of kernel size, stride, padding, and pooling on output shape

Receptive field size

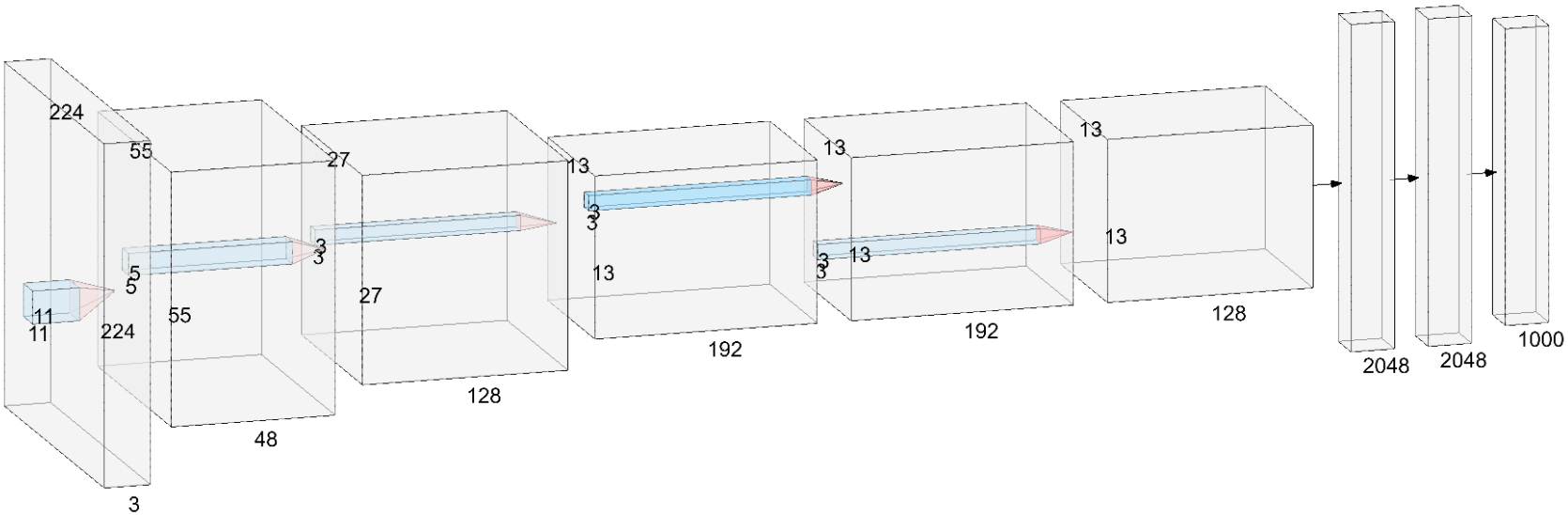
Outline

Fully Connected Layer with One Input Channel



parameters grows with the size of input and output

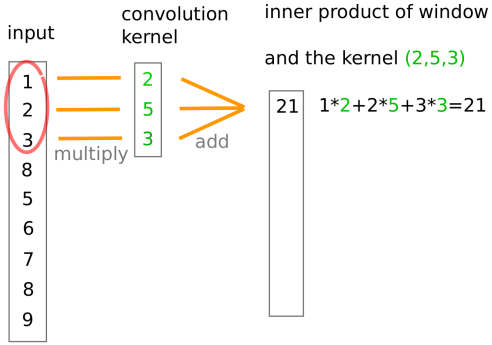
Focus on Local Patterns [3]

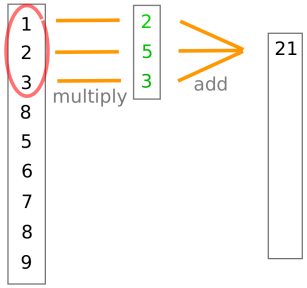


How to connect neurons sparsely when stacking layers?

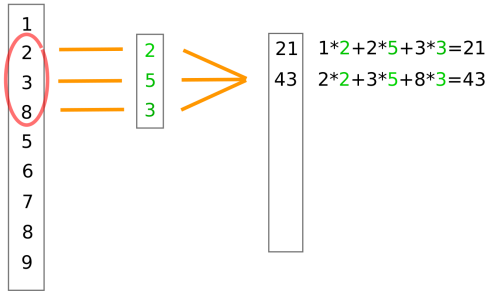
Neighboring pixels tend to be related

Connect only neighboring neurons in the input

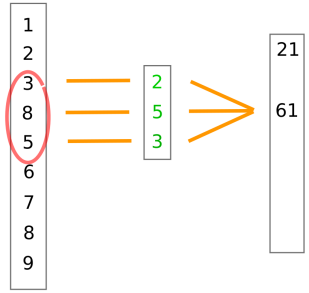




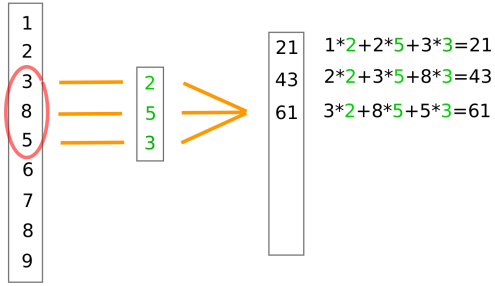
stride 1 = move kernel by 1 element



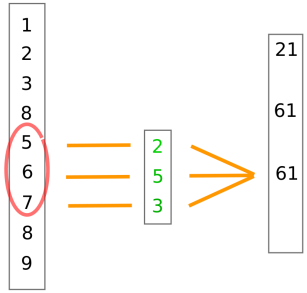
stride 2 = move kernel by 2 elements



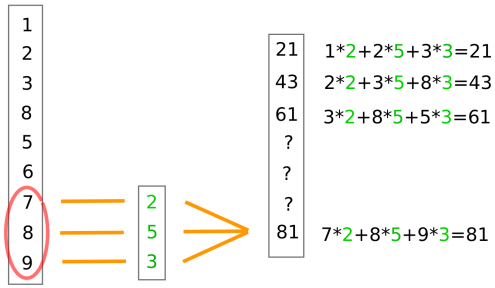
stride 1 = move kernel by 1 element



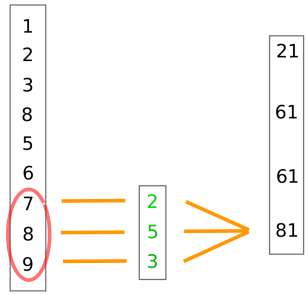
stride 2 = move kernel by 2 elements



stride 1 = move kernel by 1 element



stride 2 = move kernel by 2 elements



2-d Convolution Example with One Input Channel

1/9	1/9	1/9
1/9	1/9	1/9
1/9	1/9	1/9

3x3 Mean Filter

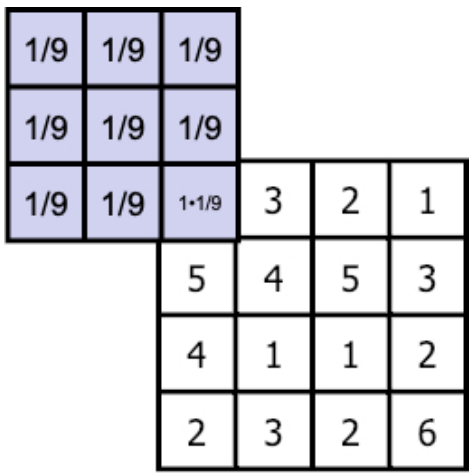
1	3	2	1
5	4	5	3
4	1	1	2
2	3	2	6

5x5 Image

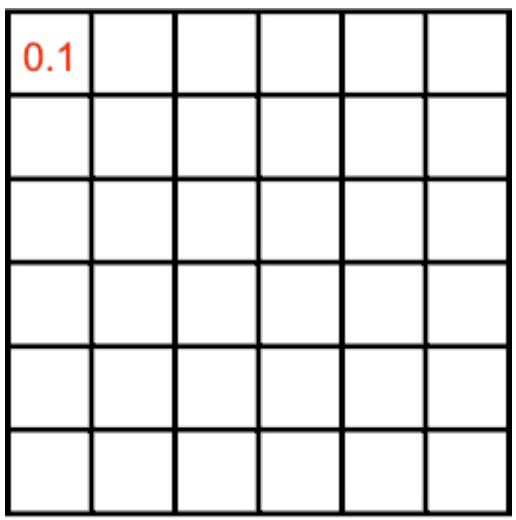
2-d Convolution Example with One Input Channel

Place the filter at the first position where the filter and image overlap

Compute the sum of the products



Image

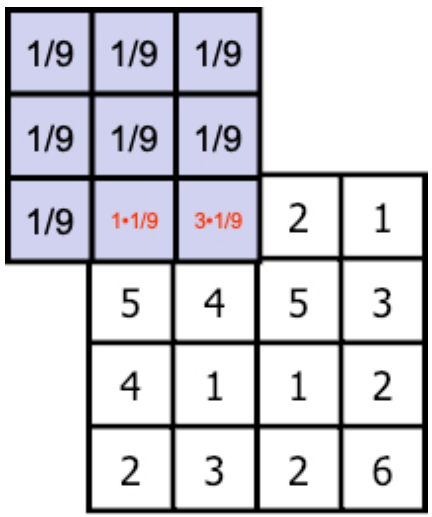


Partial Output

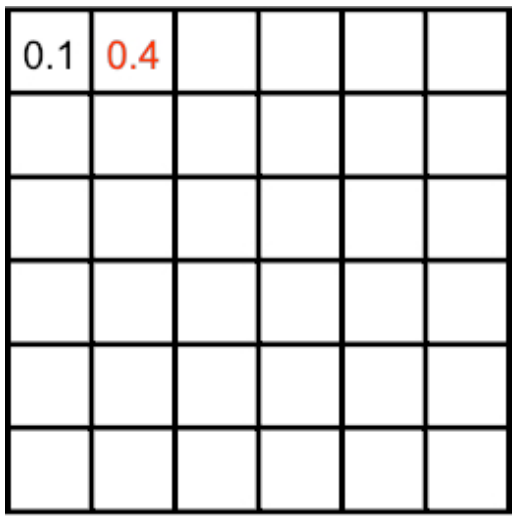
2-d Convolution Example with One Input Channel

Place the filter at the first position where the filter and image overlap

Compute the sum of the products



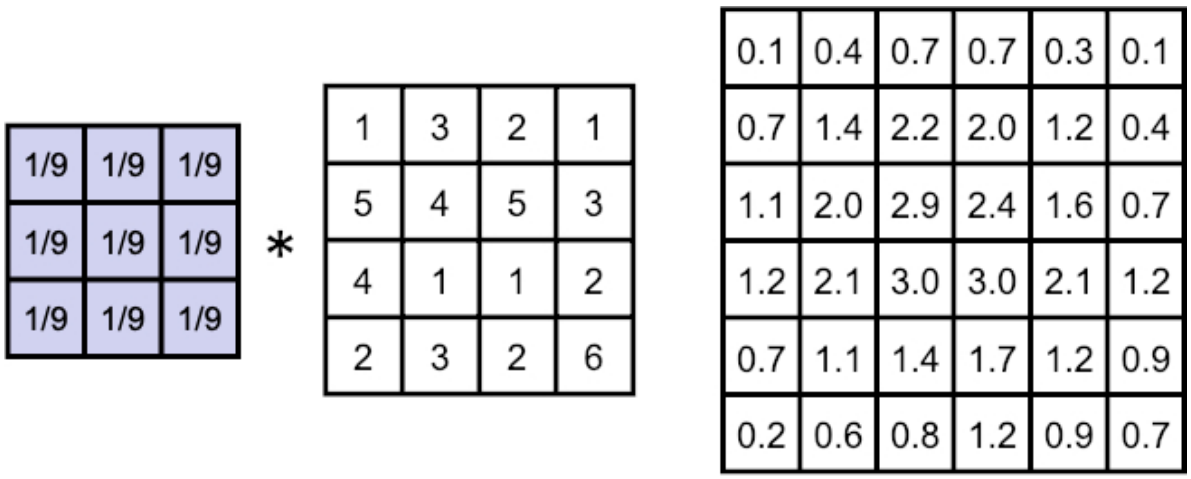
Image



Partial Output

2-d Convolution Example with One Input Channel

After 36 computations, no more overlaps so we are done.



Algebraic Properties of (Discrete) Convolution

Convolution defines a product on a vector (linear) space

Such space is a commutative associative algebra without identity

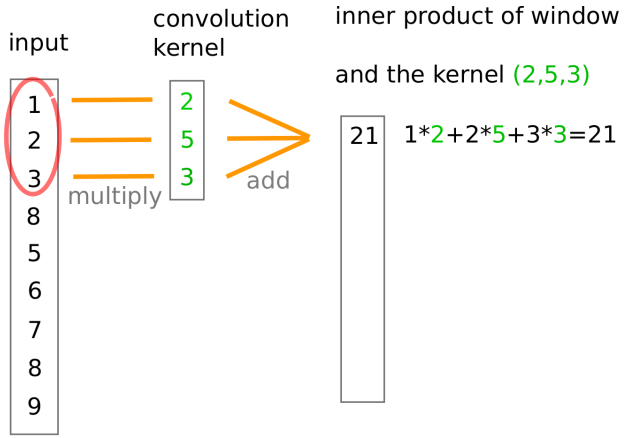
Commutativity $f * g = g * f$

Associativity $(f * g) * h = f * (g * h)$

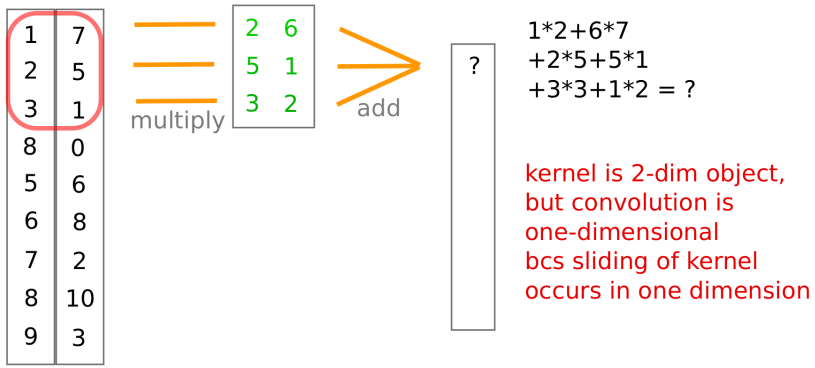
Distributivity $f * (g + h) = (f * g) + (f * h)$

Associativity with scalar multiplication $\alpha(f * g) = (\alpha f) * g = f * (\alpha g)$

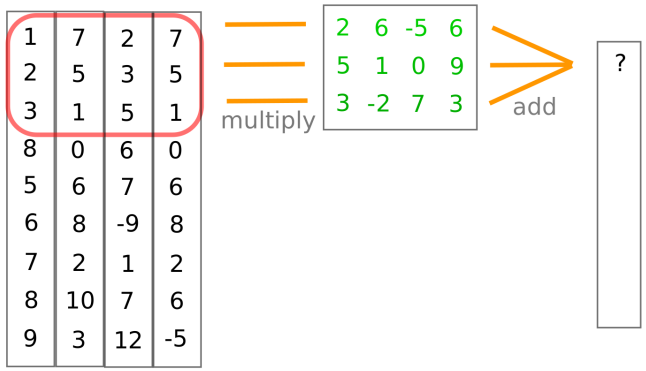
1-d Convolution Example with Multiple Input Channels



2 input channels, kernel is of size (nchannels, kernel size) = (2,3)



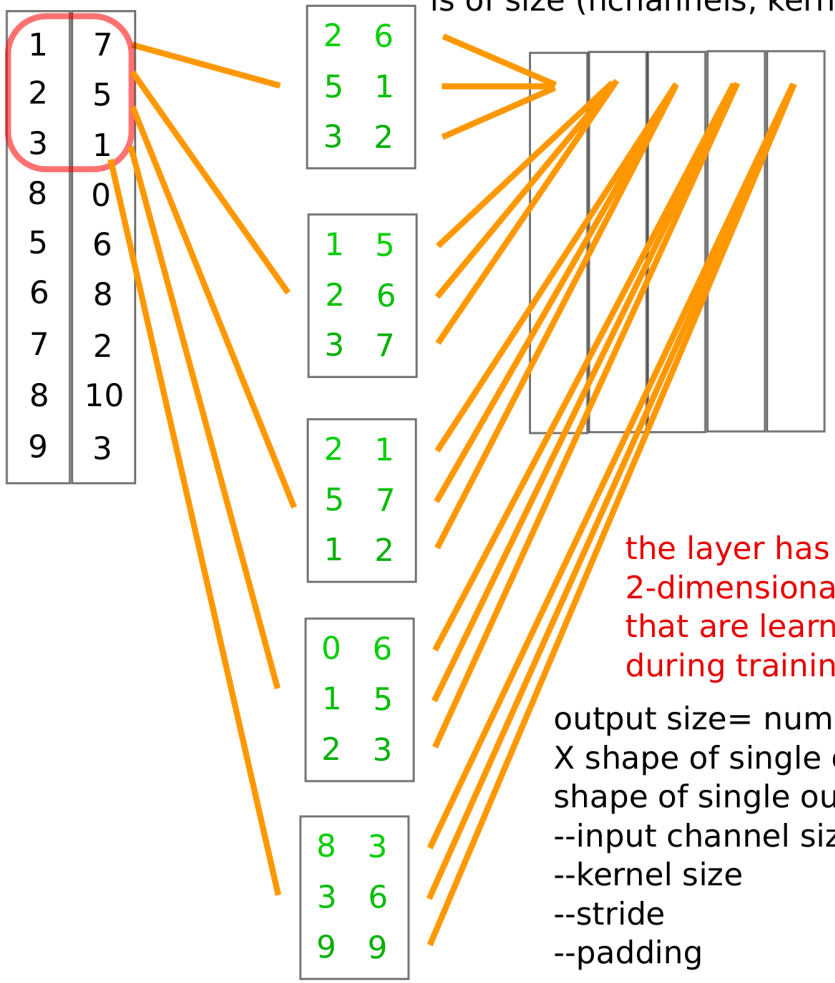
4 input channels, kernel is of size (nchannels, kernel size) = (4,3)



1-d Convolution with Multiple Input and Output Channels

5 output channels -- by 5 independent kernels

2 input channels, each of the five kernels
is of size (nchannels, kernel size) = (2,3)



the layer has now 5
2-dimensional kernels
that are learnt/updated
during training

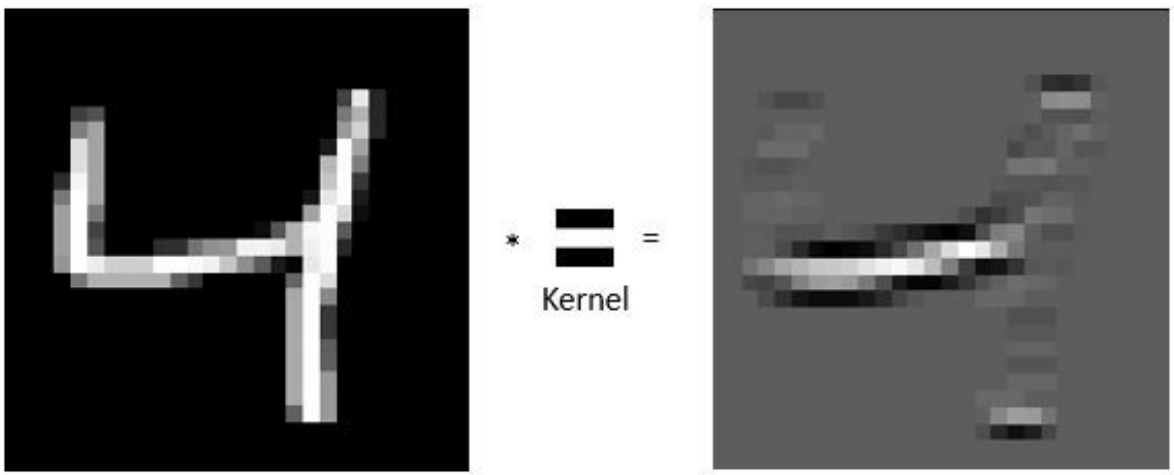
output size= number of output channels
X shape of single output channel
shape of single output channel from:
--input channel size
--kernel size
--stride
--padding

Horizontal Edges: Output after Convolution and ReLU

1	2	1
0	0	0
-1	-2	-1

Filter for Extracting Horizontal Edges

Highlight changes in intensity (gradients) in the vertical direction



Horizontal filter example

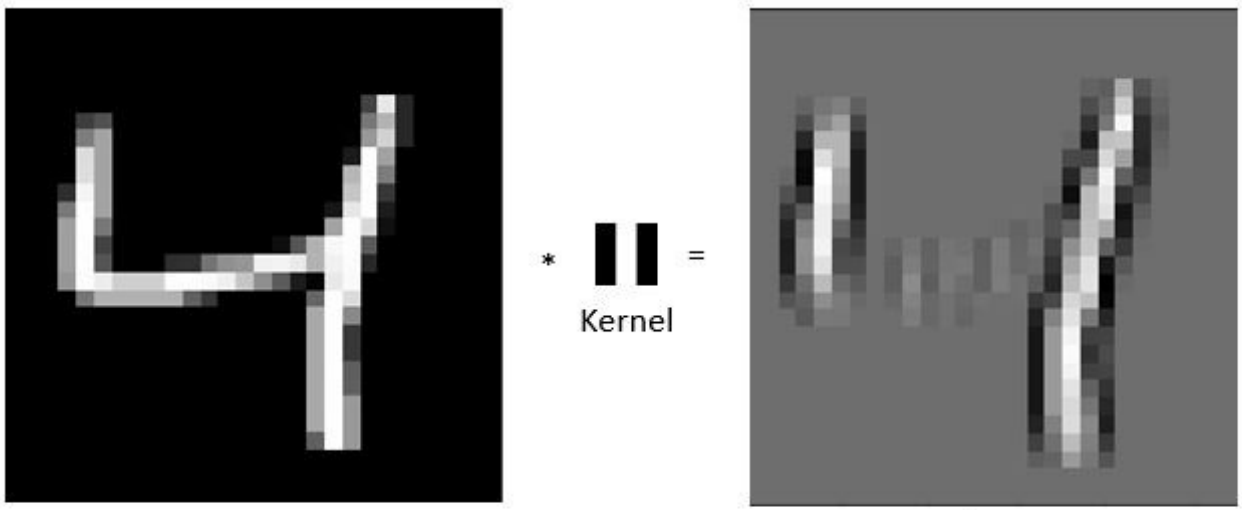
pylessons.com/CNN-tutorial-introduction

Vertical Edges: Output after Convolution and ReLU

1	0	-1
2	0	-2
1	0	-1

Filter for Extracting Vertical Edges

Highlight changes in intensity (gradients) in the horizontal direction



Vertical filter example

pylessons.com/CNN-tutorial-introduction

2-d Discrte Convolution and Cross-correlation

Convolution of a 2-d image $\mathbf{x}$ and a 2-d kernel (filter) K:

$$z_{i,j} = \sum_m \sum_n K_{m,n} x_{i-m,j-n}$$

Several neural network libraries implement cross-correlation equivalent formulation [2]:

$$z_{i,j} = \sum_m \sum_n K_{m,n} x_{i+m,j+n}$$

For simplicity, we call both operations convolution

Learning algorithm learns kernel values through ERM

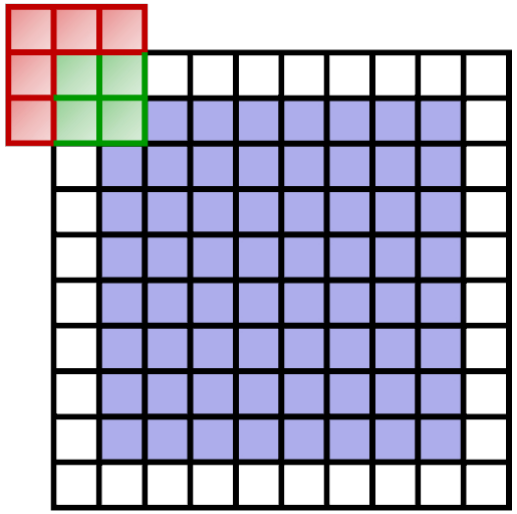
Inner Product View

Convolution is a series of inner products

Inner product between filter and a window-view of input tensor

Window is sliding, for 1- and 2-d conv. along 1 and 2 dimensions

Padding



To keep output size unchanged compared to input, one can pad the image

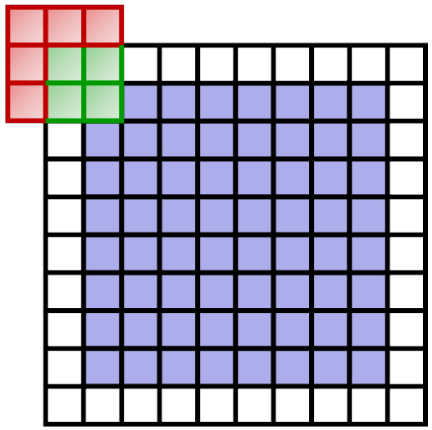
Most common in deep learning: Pad with 0s (zero padding)

Expand input image with another fixed value, e.g. mean pixel value

Use value of nearest pixel (replicate padding)

Use mirror-reflected indexing (reflect padding)

Output Size # of Inner Product Computations



For $M \times N$ input and $k \times k$ filter

Valid (or reduced): Only positions where the filter fits inside input

Output will have smaller size than input $(M - k + 1) \times (N - k + 1)$

Same: Positions where the filter centre is inside input

Output will be the same size as input

Full: All positions where the filter and input overlap

Output will have larger size than input $(M + k - 1) \times (N + k - 1)$

Convolution with Multiple Channels

Convolution kernel has additional dimensions for input & output channels

C_i # input channels in the feature map to be processed

C_o # output channels

We use tensor notation with brackets

Let $r = 1, \ldots, C_o$

$$z[r, i, j] = \sum_m \sum_n \sum_{c=1}^{C_i} K[c, m, n, r] x[c, i + m, j + n]$$

Outline

Why Convolutions? Less Parameters and Parameter Sharing

Consider layer l of a neural network with N_l neurons in layer l and N_{l+1} neurons in layer $l+1$

Fully connected: How many parameters for layer l?

Locally connected: Suppose each output neuron takes input from a patch of 5 neighbors: How many parameters for layer l?

1-d convolution with kernel size 5: only 5 parameters, no matter how many inputs or outputs (parameter sharing)

Convolutions have a small # parameter, independent of input and output size in the sliding dimension (# output channels (filters) matters)

Convolution Intuition

Control # parameters limited relative to training set size to generalize well

Better off stacking simple functions in depth and many layers than learning complex functions in one layer

Convolution output at one fixed input region is an inner product between kernel weights and the region

Slide the kernel over the input tensor and apply the inner product over many windows

Apply the same kernel across all locations for computing inner products

Share the parameters to be learned

Why Convolutions? Localized Pattern Detection

Each convolution layer detects localized patterns over all input tensor

An object

Can appear anywhere

Why Conv.? Localized Pattern Detect. & Translation Invariance

Kernel matrix is some pattern detector

Convolution $y = \text{kernel}^\top \text{input window} + b$

Inner product is a similarity measure

Positive for inputs parallel to kernel, zero for inputs orthogonal, negative for inputs antiparallel to kernel

Apply an activation function $g(\text{kernel}^\top \text{input window} + b)$, detector for patterns in input channels similar to the kernel

Detection is performed across all sliding space (1-d,2-d,3-d,n-d)

Why Conv.? Localized Pattern Detect. & Translation Invariance

Detection is global for fully connected, localized for convolutions

Convolution detects patterns in a translation-invariant manner!

Detect similar patterns in different sizes/scales

Detect similar patterns from different view angles

Detect deformations of the same pattern/object

Detect different types in the same class

Stacking Layers for CNNs

At each convolutional layer, apply a set of local filters

Stack multiple small filters instead of using few large filters

Remember that convolution is associative

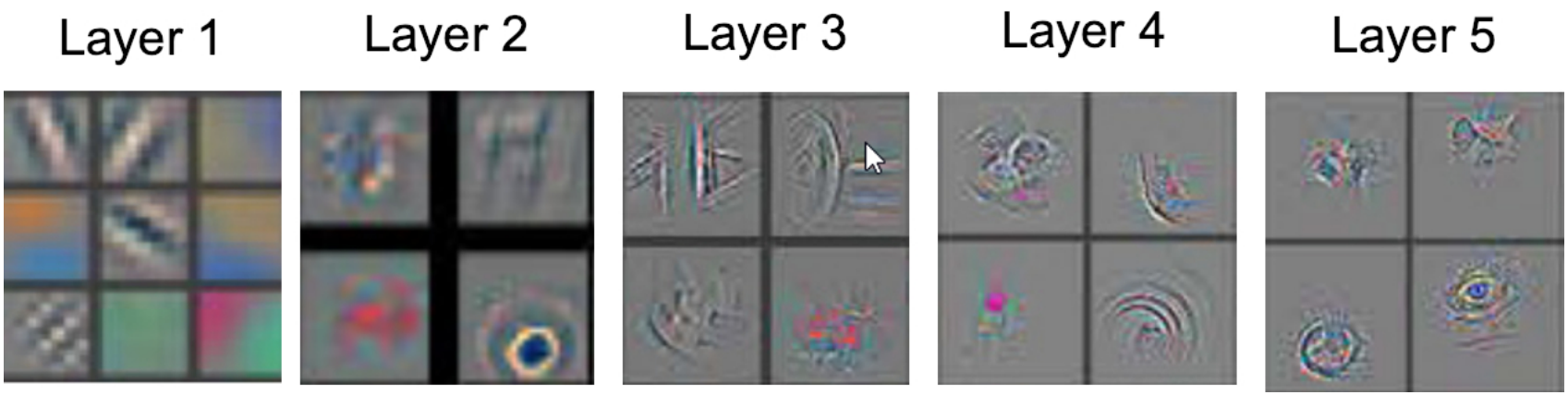
Requires much less parameters

- Two 3x3 filters has 18 parameters, one 5x5 filter has 25 parameters
- Three 3x3 filters has 27 parameters, one 7x7 filter has 49 parameters
- Four 3x3 filters has 36 parameters, one 9x9 filter has 81 parameters

Focus on very local image attributes in the first conv. layers

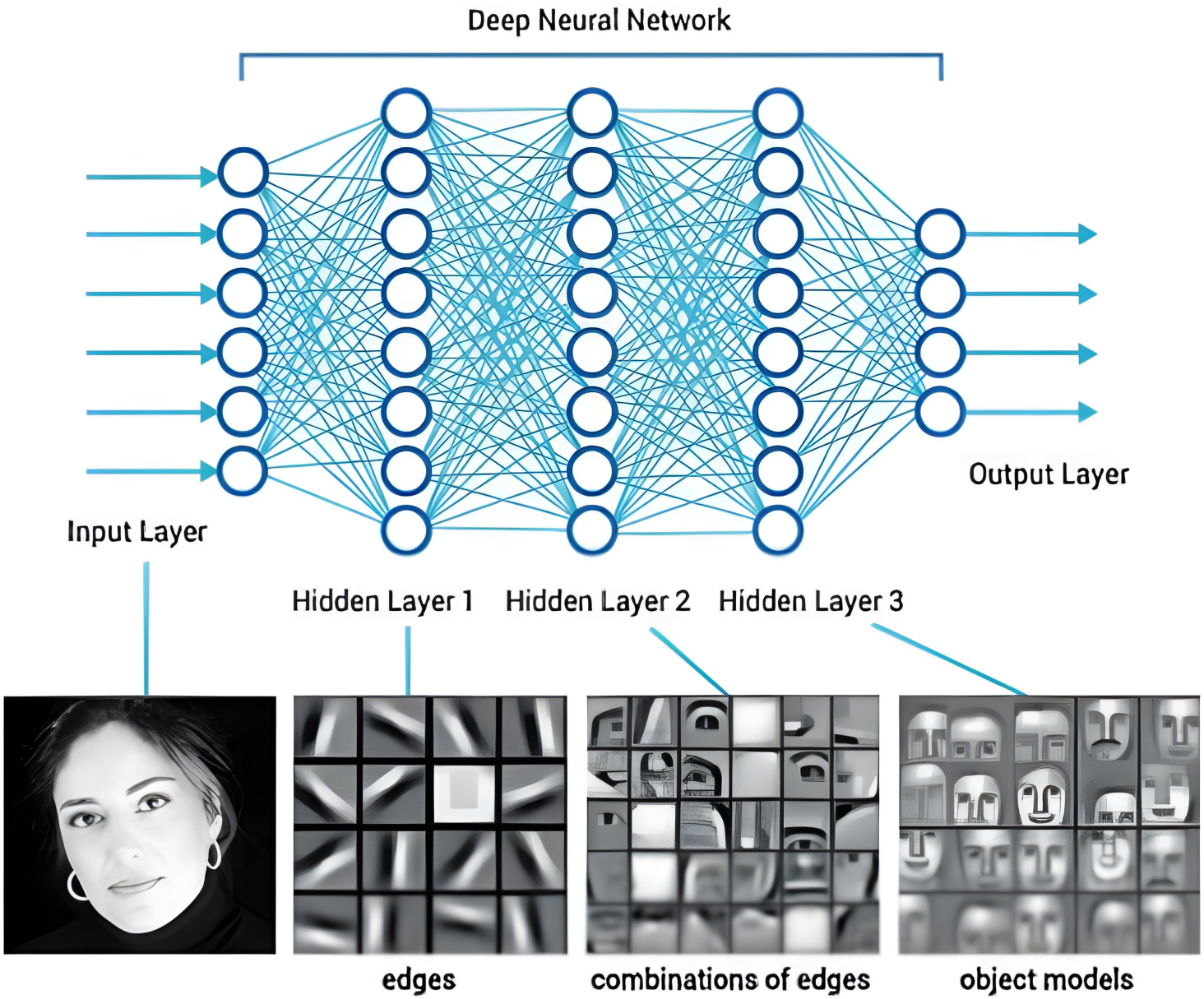
Look for gradually more complex patterns in deeper layers

Filters Detect More Complex Patterns in Deeper Layers



Zeiler M.D., Fergus R. (2014) Visualizing and Understanding Convolutional Networks

Filters Detect More Complex Patterns in Deeper Layers



https://iq.opengenus.org/hidden-layers/

Interactive Time-out

Join at menti.com | use code 5761 5667

Fully Convolutional Neural Nets (ConvNets and CNNs)

After a series of convolutional layers, obtain a 3-d activation maps tensor

For a classification task, we want c outputs (e.g. number of classes)

Apply c filters in the last convolutional layer

Calculate the average over the spatial dimensions, and finally softmax

Values interpreted as class probabilities

CNNs with Dense Final Layer

Alternatively, we can flatten 3-d tensor from final convolutional layer

Stack one or more fully-connected layers on top

The last layer has c neurons

Apply the softmax function

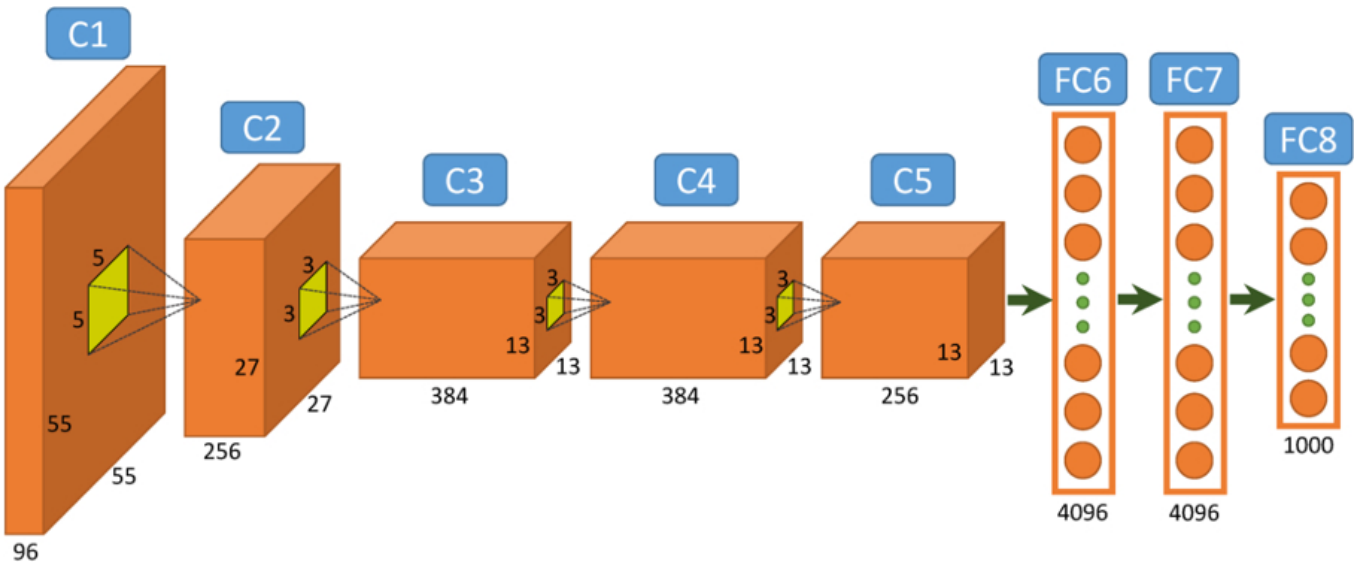
Classical CNN Architecture

Stack multiple convolutional layers each with multiple filters

Add bias for each layer and apply a non-linear activation function

Reduce spatial dimensions with more filters as we get deeper

Apply dense layers at the end



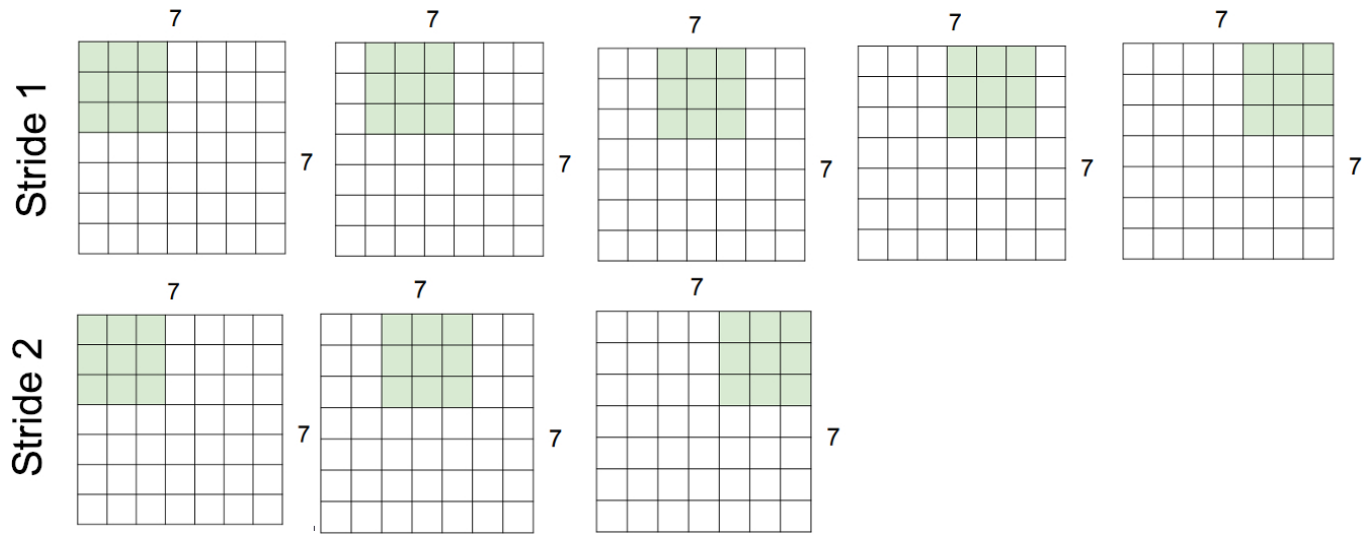
Outline

Stride

Stride is number of input elements/pixels which filter moves in every step

Stride is spacial step length in the discrete convolution

It causes downsampling with a factor equal to the stride



Output Spatial Size

$$N^{[l+1]} = \left\lfloor \frac{N^{[l]} + 2p - k}{s} + 1 \right\rfloor$$

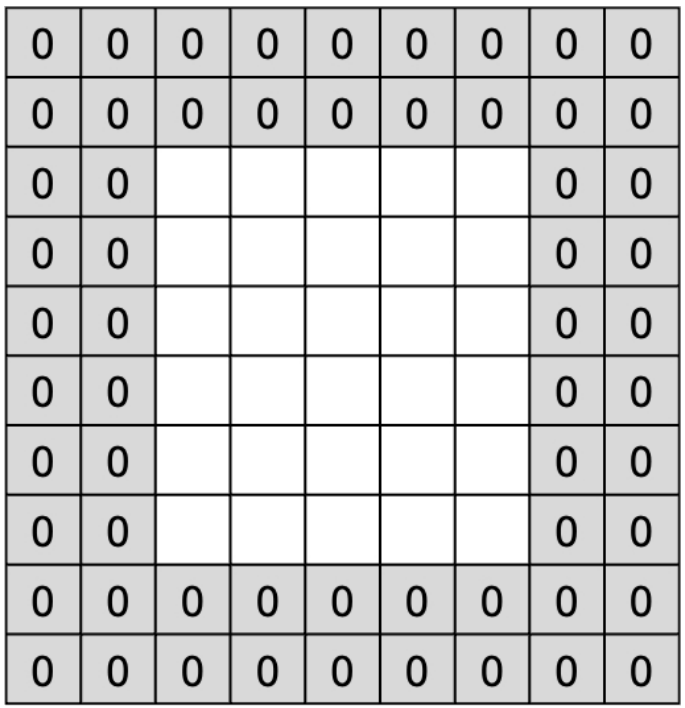
Output size $N^{[l+1]} \times N^{[l+1]}$

Input size $N^{[l]} \times N^{[l]}$

Padding p: changes input shape $N^{[l]} \times N^{[l]} \rightarrow (N^{[l]} + 2p) \times (N^{[l]} + 2p)$

Filter size $k \times k$

Stride s



Standard Padding

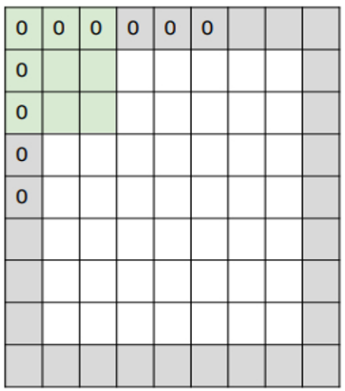
$$N^{[l+1]} = \left\lfloor \frac{N^{[l]} + 2p - k}{s} + 1 \right\rfloor$$

Standard Padding $p = \frac{k-1}{2}$

$$N^{[l+1]} = \left\lfloor \frac{N^{[l]} - 1}{s} + 1 \right\rfloor$$

Preserves spatial size of input $N^{[l+1]} = N^{[l]}$ under $s = 1$

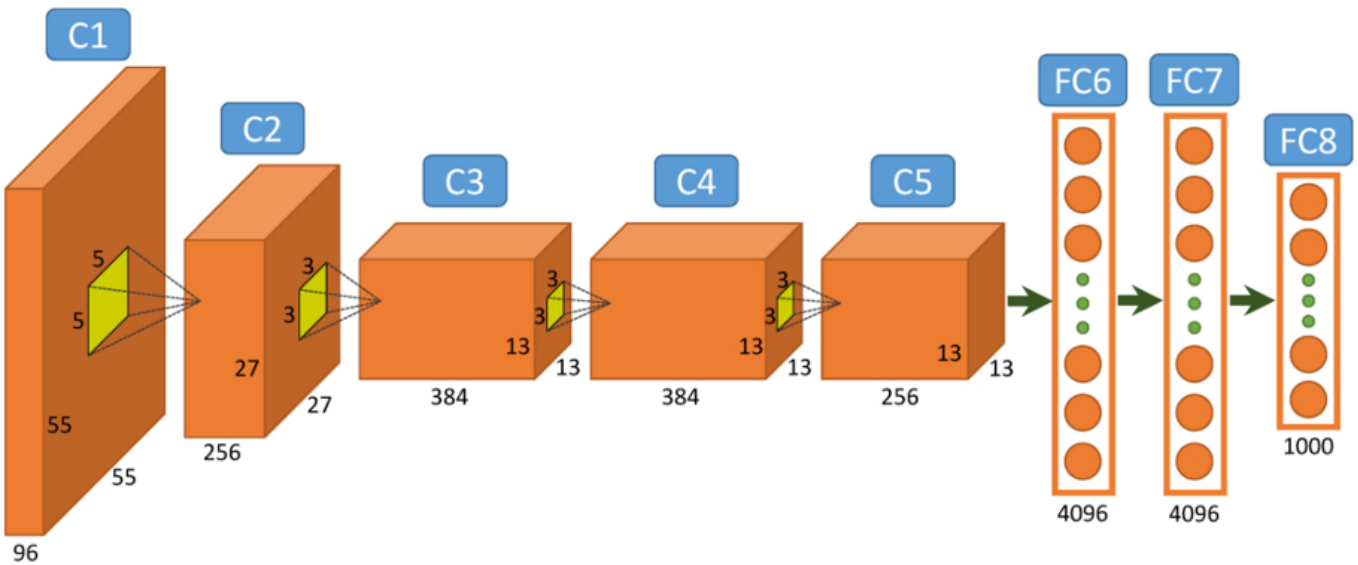
Filter size k has no influence on the output spatial size



Output Spatial Size

Stride $s > 1$ decreases the spatial size

Often combined with an increase in number of channels

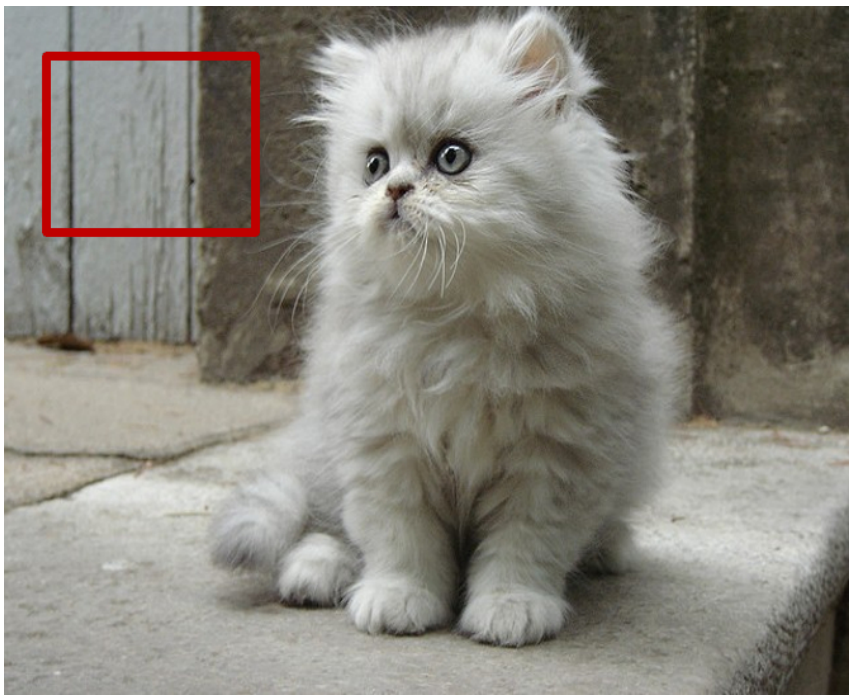


Outline

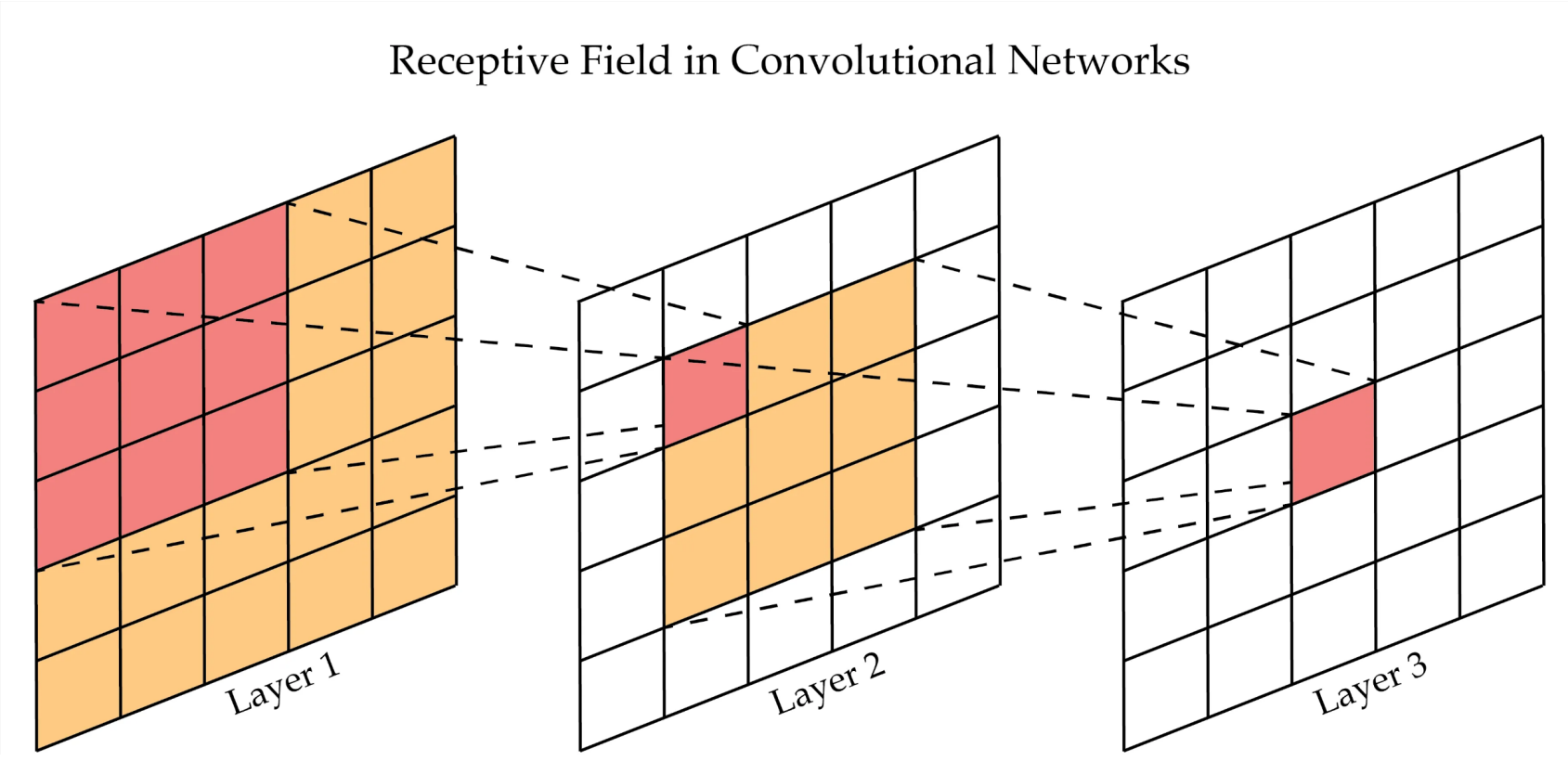
Receptive Field (Field of View)

CNN is built by stacking convolutional layers

How much of input image is available to a particular neuron?



Receptive Field (Field of View)

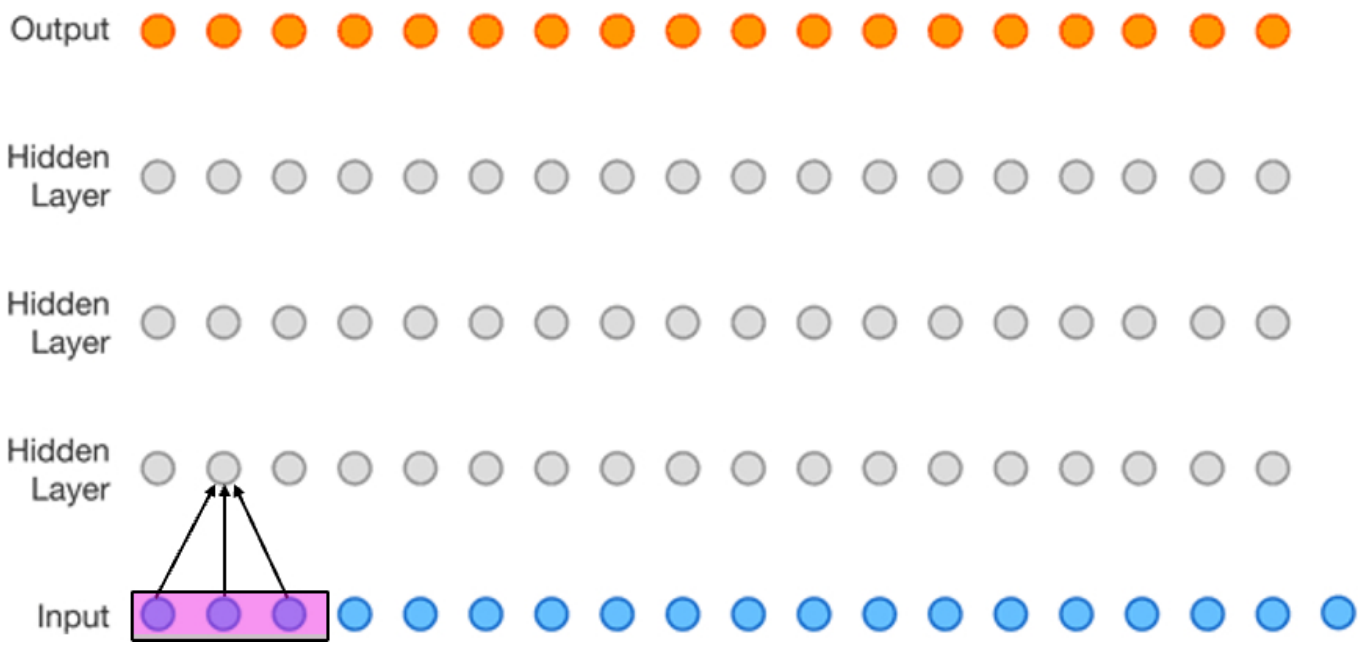


https://medium.com/@rekalantar/receptive-fields-in-deep-convolutional-networks-43871d2ef2e9

Receptive Field (Field of View)

Receptive field of neurons increases with each layer

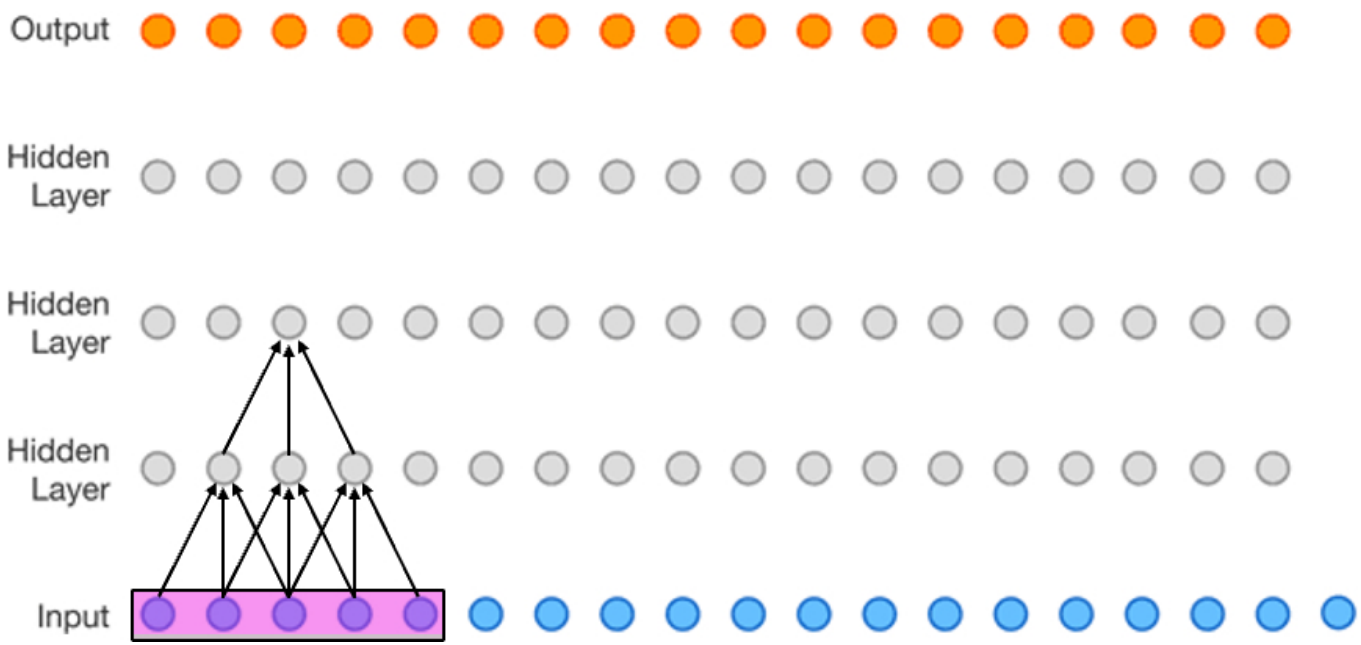
3 inputs influence each neuron in the first hidden layer



Receptive Field (Field of View)

Receptive field of neurons increases with each layer

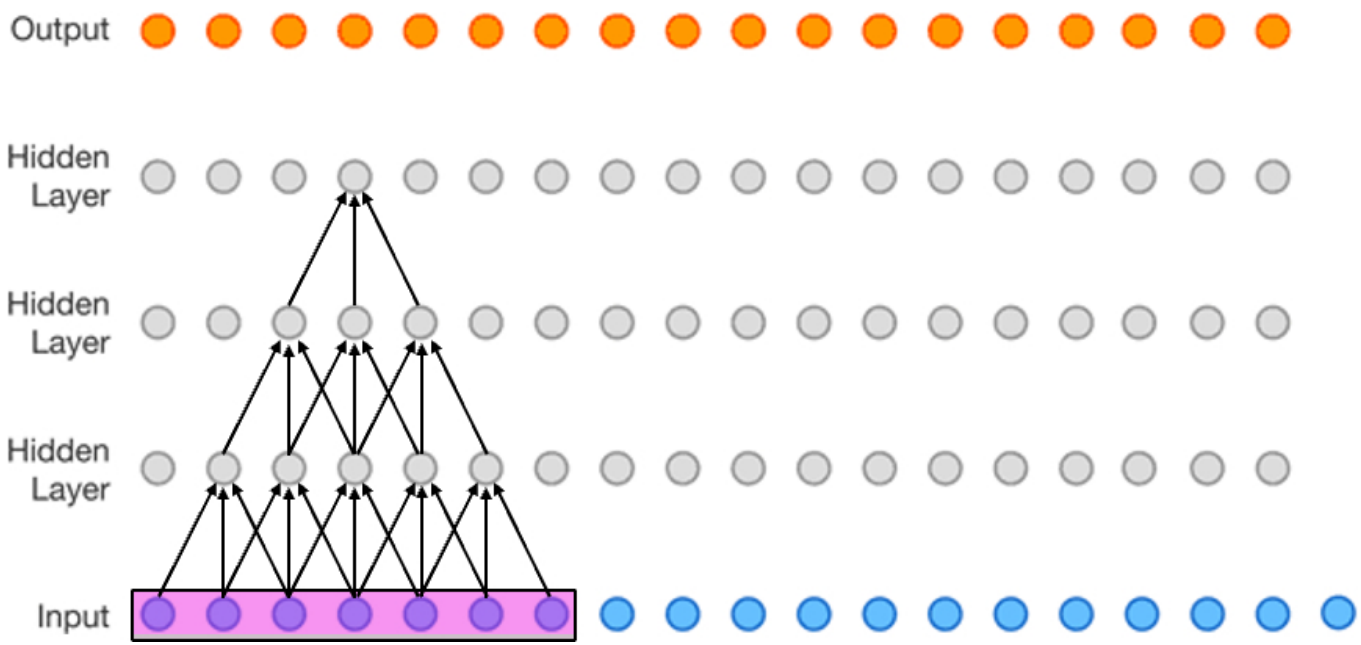
5 inputs influence each neuron in the second hidden layer



Receptive Field (Field of View)

Receptive field of neurons increases with each layer

7 inputs influence each neuron in the third hidden layer

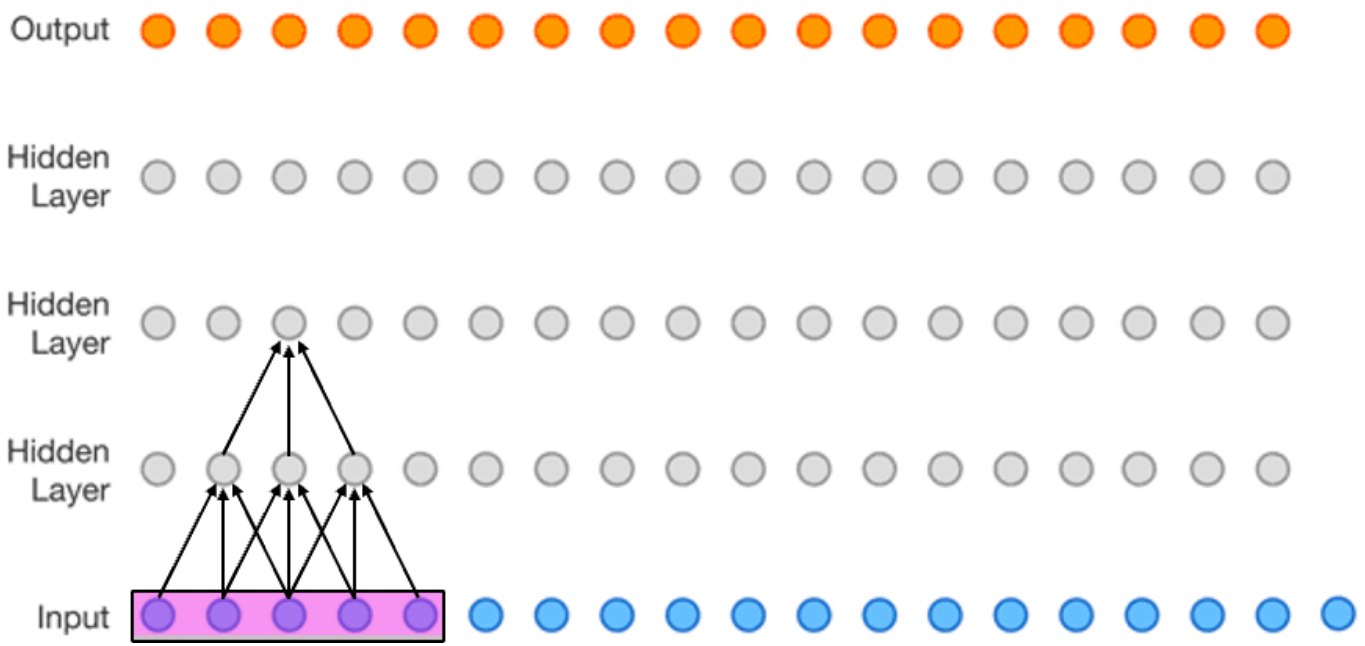


Receptive Field (Field of View)

Under $s = 1$, receptive field grows with $k - 1$ for each layer

Larger receptive field per parameter is generally good

Deeper with more layers and smaller filters is more parameter efficient



Receptive Field (Field of View)

Layer index $l = 1, \ldots, L$

Filter size $k^{[l]} \times k^{[l]}$

Stride $s^{[l]} \times s^{[l]}$

Recursion for receptive field $R^{[l]}$ with $R^{[0]} = 1$:

$$R^{[l]} = R^{[l-1]} + \left(k^{[l]} - 1\right) \prod_{i=1}^{l-1} s^{[i]}$$

Unravelling recursion

$$R^{[l]} = \sum_{j=1}^{l} \left(k^{[j]} - 1\right) \prod_{i=1}^{j-1} s^{[i]}$$

Outline

Pooling

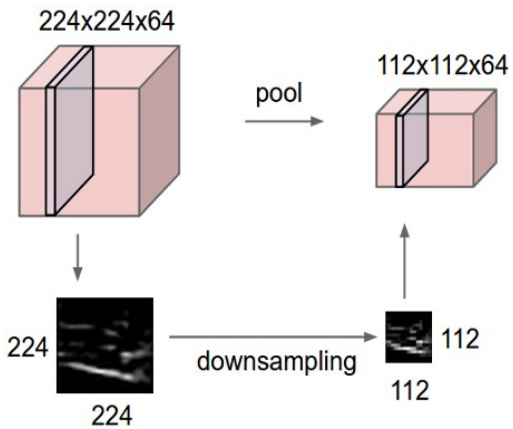
Spatial reduction and forcing invariance

For multiple filters, operates over each channel independently as before

No parameters to be learned

Two common methods:

- Max pooling
- Average pooling



Max Pooling

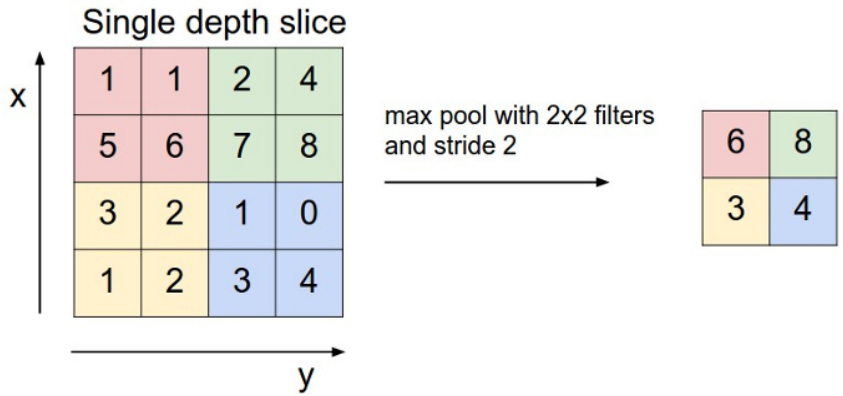
A strided maximum filtering

Maximum filter is a non-linear filter

Choosing the maximum value inside the filter neighborhood

The highest pattern detector response over a field of view is selected

Explicitly remove some spatial information



References

[1] Y. S. Abu-Mostafa, M. Magdon-Ismail, and H.-T. Lin. *Learning from data*. AMLBook New York, 2012.

[2] I. Goodfellow, Y. Bengio, and A. Courville. *Deep learning*. MIT press, 2016.

[3] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2012.