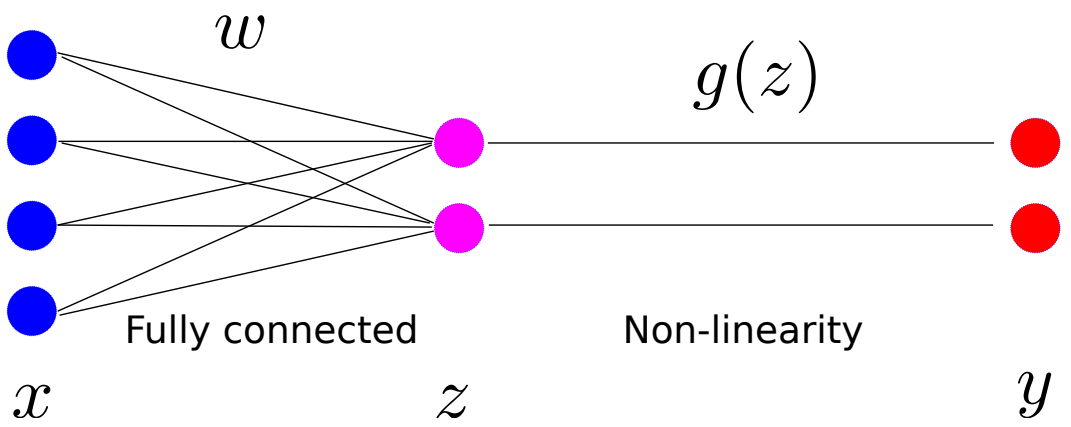
Introduction to Deep Learning
Convolutional Neural Networks

Loïc Le Folgoc
LTCI, Télécom Paris, IP Paris
loic.lefolgoc@telecom-paris.fr

Recall: neural networks

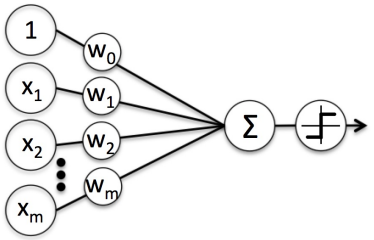
- Neural networks provide a highly flexible way to model complex dependencies and patterns in data

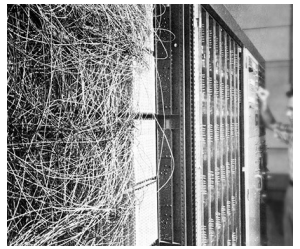
- In the previous lessons, we saw the following elements :
  - MLPs : fully connected layers, biases
  - Activation functions : sigmoid, softmax, ReLU
  - Optimisation : gradient descent, stochastic gradient descent
  - Regularisation : weight decay, dropout, batch normalisation



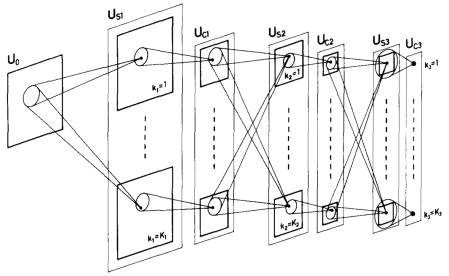
Some history of neural networks

- Perceptron [Rosenblatt, 1958]





- Neocognitron [Fukushima, 1980]:
  - notion of receptive field
  - based on work on animal perception [Hubert and Wiesel, 1968]



- Back-propagation [Rumelhart, 1985]

$$\Delta_p w_{ji} = \eta \delta_{pj} o_{pi}$$

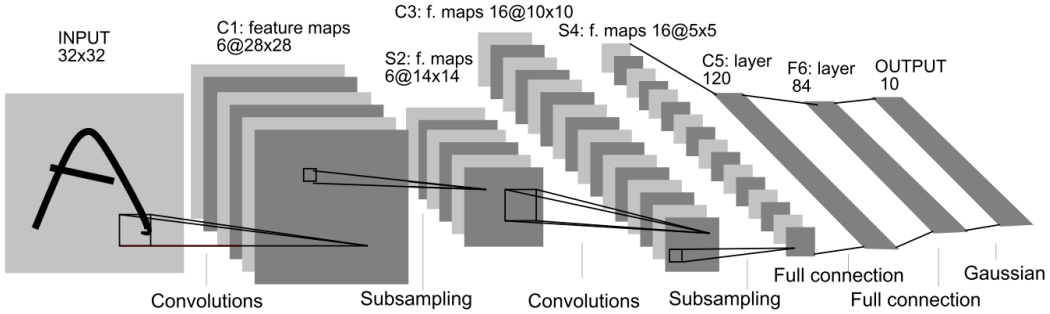
$$\delta_{pj} = (t_{pj} - o_{pj}) f'_j(net_{pj})$$

$$\delta_{pj} = f'_j(net_{pj}) \sum_k \delta_{pk} w_{kj}$$

[Rosenblatt, 1958] **The perceptron: a probabilistic model for information storage and organization in the brain**, Rosenblatt, F., *Psychological review, 1958*
[Fukushima, 1980] **Neocognitron: A Self-organizing Neural Network Model for a Mechanism of Pattern Recognition Unaffected by Shift in Position**, Fukushima, K., *Biological Cybernetics, 1980*
[Hubert and Wiesel, 1968] **Receptive fields and functional architecture of monkey striate cortex**, Hubel, D. H. and Wiesel, T. N., *1968*
[Rumelhart, 1985] **Learning representations by back-propagating errors**, Rumelhart, D. E., Hinton, G. E., & Williams, R. J., *Nature, 1986*

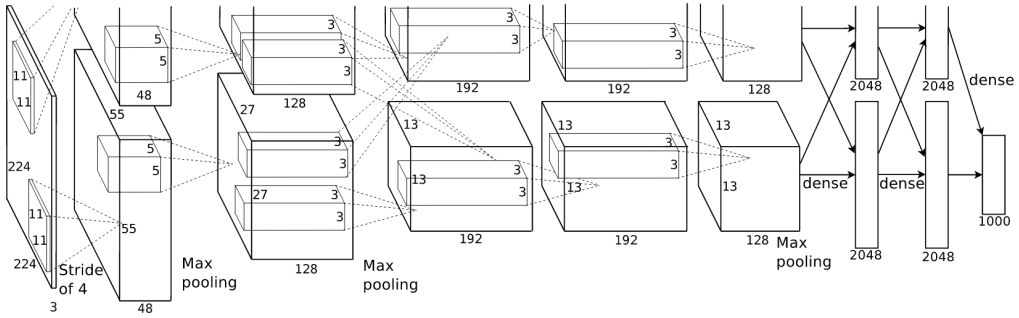
More recently: the deep learning revolution

- LeNet [LeCun, 1989]:
  - back-propagation for training
  - CNN for digit recognition
  - 3 days of training



- CUDA, 2007:
  - General purpose GPU
  - GPU for scientific computations

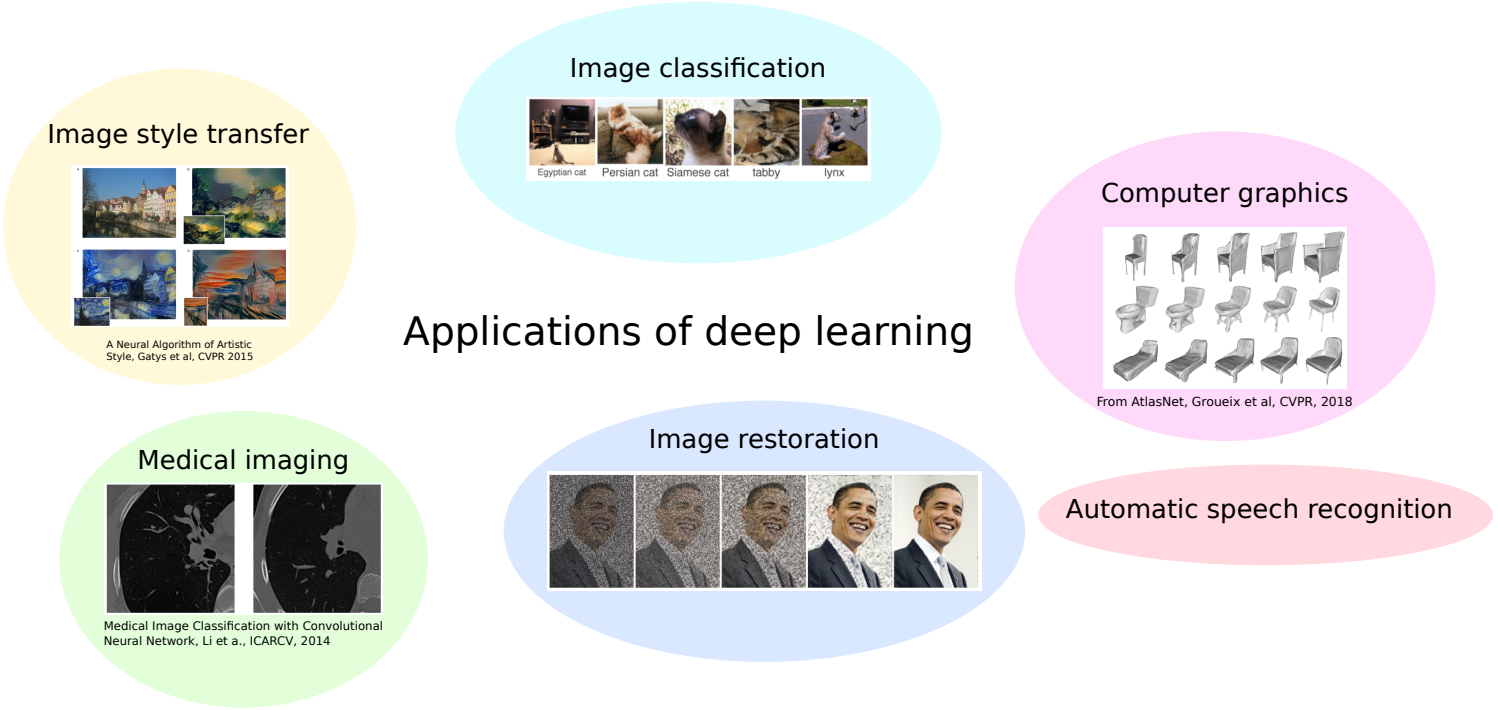
- AlexNet [Krizhevsky, 2012]:
  - GPU training of truly deep CNN
  - greatly outperformed traditional approaches in the ImageNet Large Scale Visual Recognition Challenge (ILSVRC)



[LeCun, 1989] **Backpropagation Applied to Handwritten Zip Code Recognition**, LeCun, Y. et al., AT&T Bell Laboratories, *Neural computation*, *1989*

More recently: the deep learning revolution

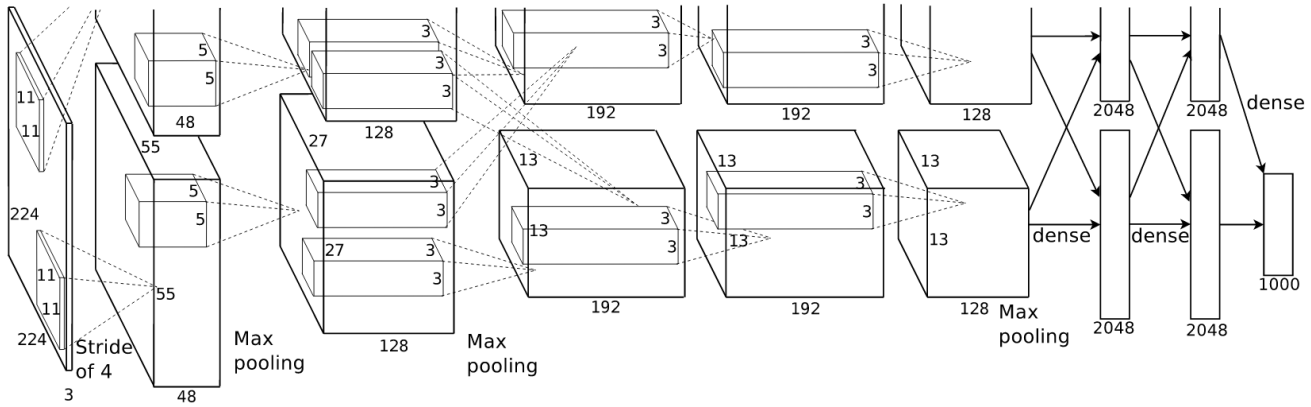
- Since 2012, CNNs have completely revolutionized many domains
- CNNs produce competitive/best results for most problems in image processing and computer vision



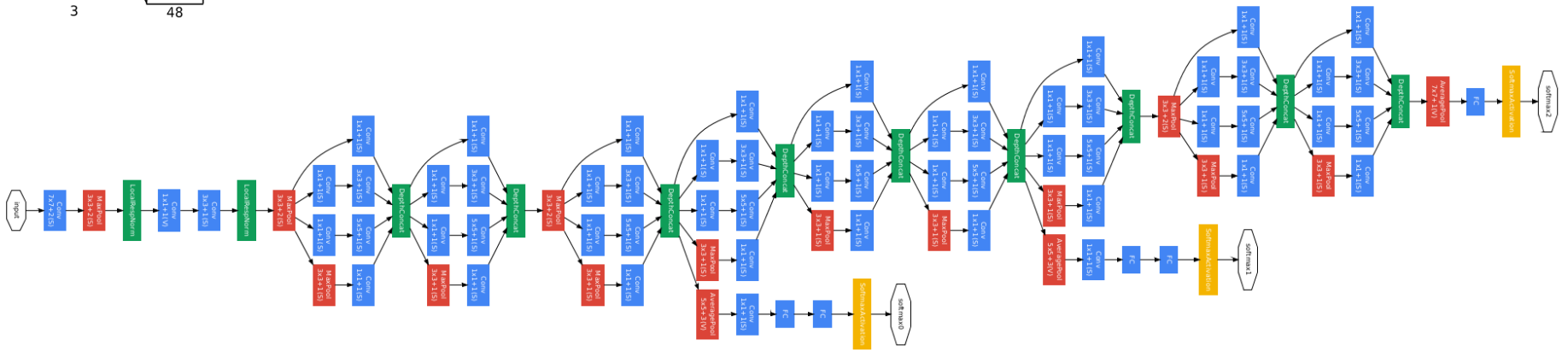
- Being applied to an ever-increasing number of problems

More recently: the deep learning revolution

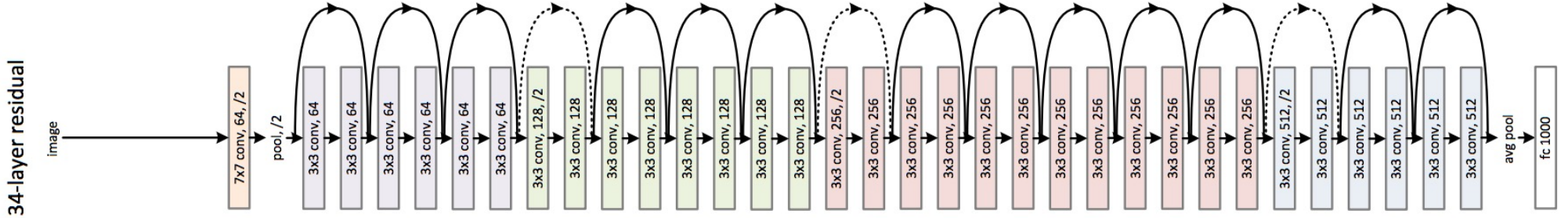
AlexNet



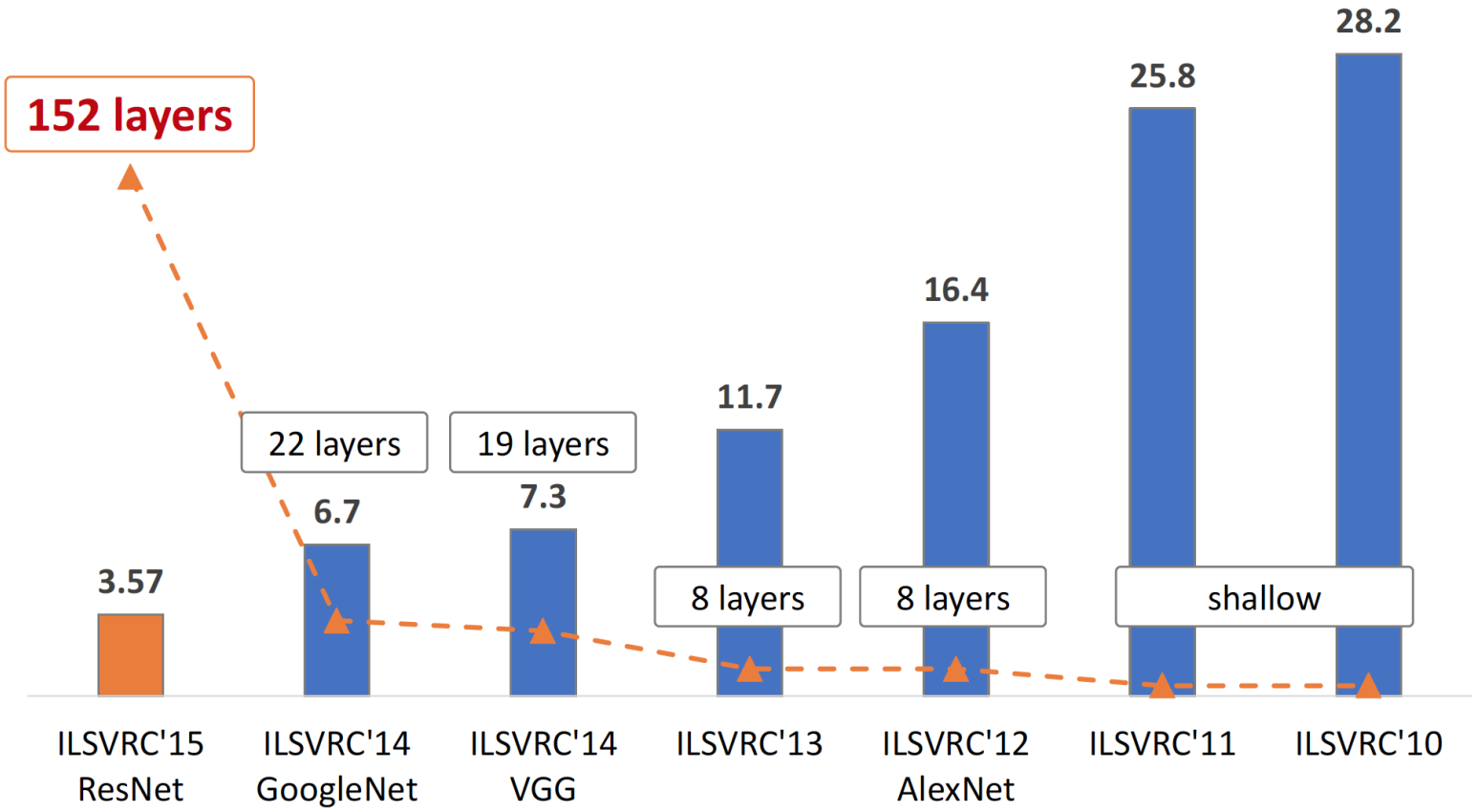
GoogLeNet



ResNet

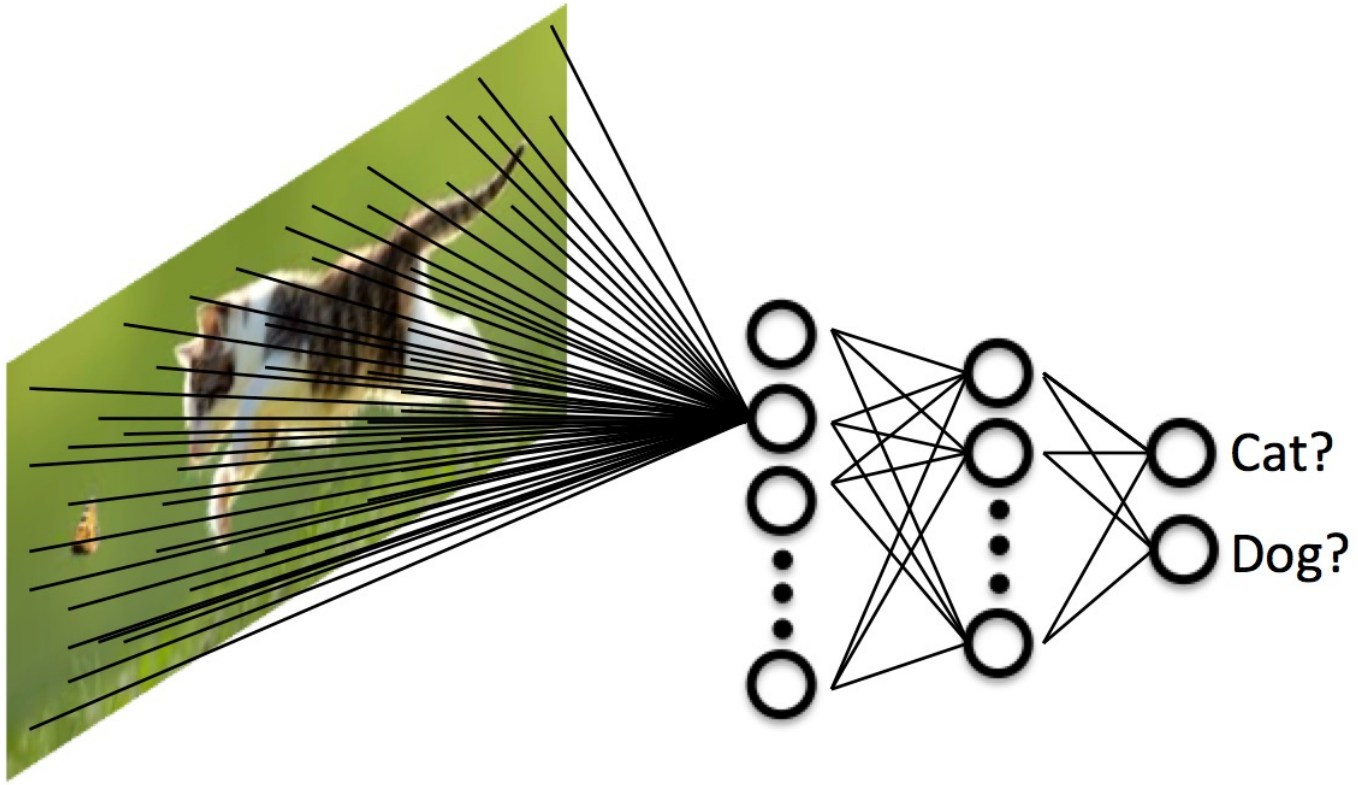


More recently: the deep learning revolution



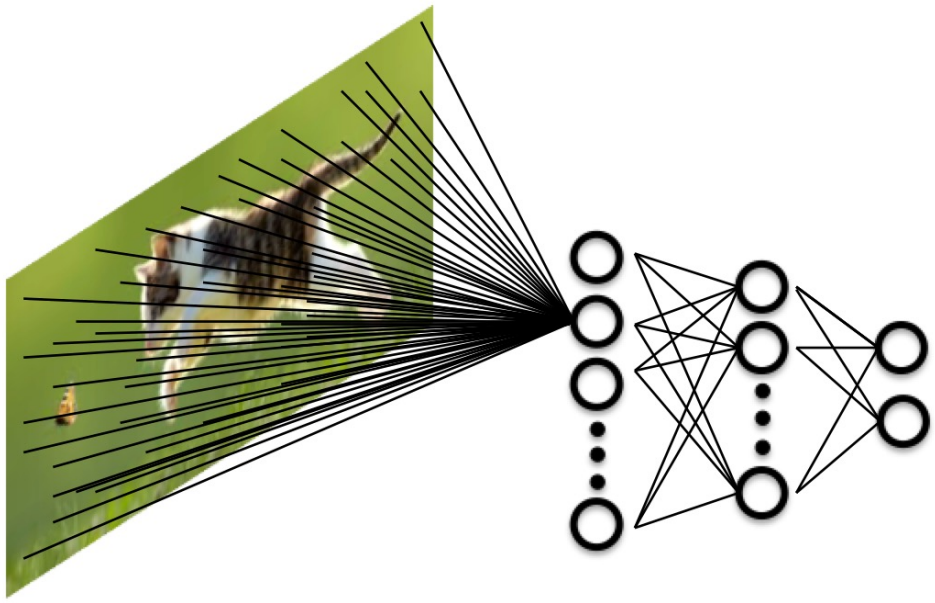
Top-5 error rate on ILSVRC

Neural networks for image classification



Neural networks: Fully-connected networks

- In MLPs each layer of the network was **fully connected**

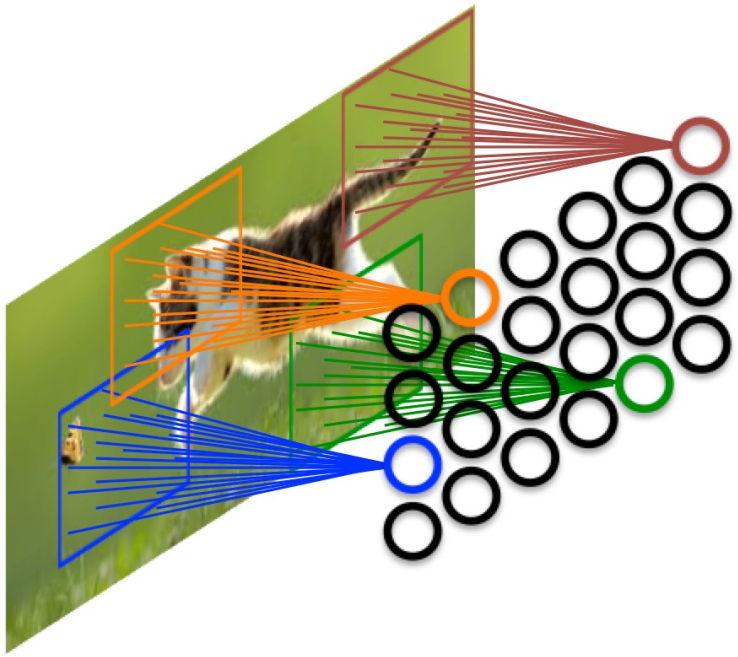


- Each neuron detects one feature/pattern
- Too many weights for each neuron/pattern

- Example: 1000×1000 image
  - 10^6 weights/neuron!

- Idea: we are sensitive to local features, only neighboring pixels are correlated

Neural networks: Locally-connected networks

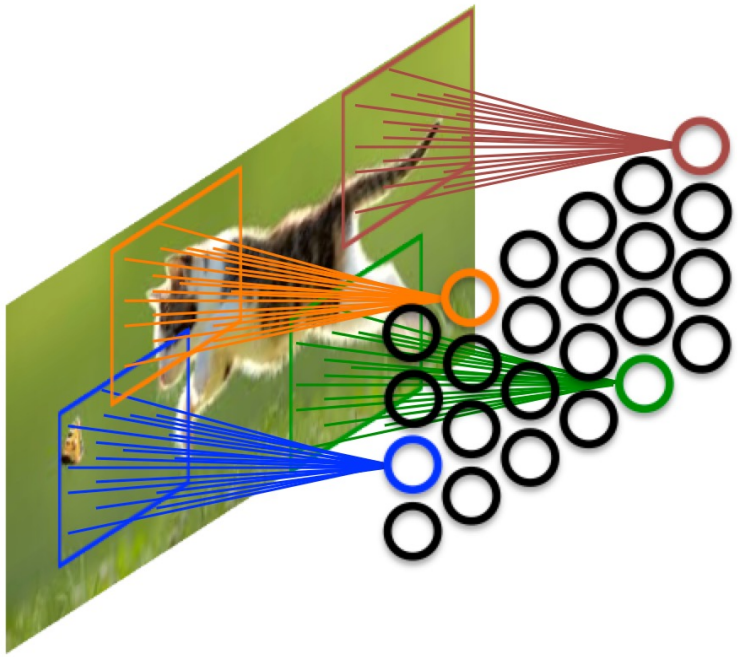


- Each neuron detects a different pattern at a certain location
- Kernel of a neuron: spatially-limited connections

- Example: 10×10 kernel
  - 10^2 weights/neuron

- If hidden layer with a grid of 1000×1000 neurons
  - $10^6\times10^2$ weights!

Neural networks: Locally-connected networks



- Each neuron detects a different pattern at a certain location
- Kernel of a neuron: spatially-limited connections

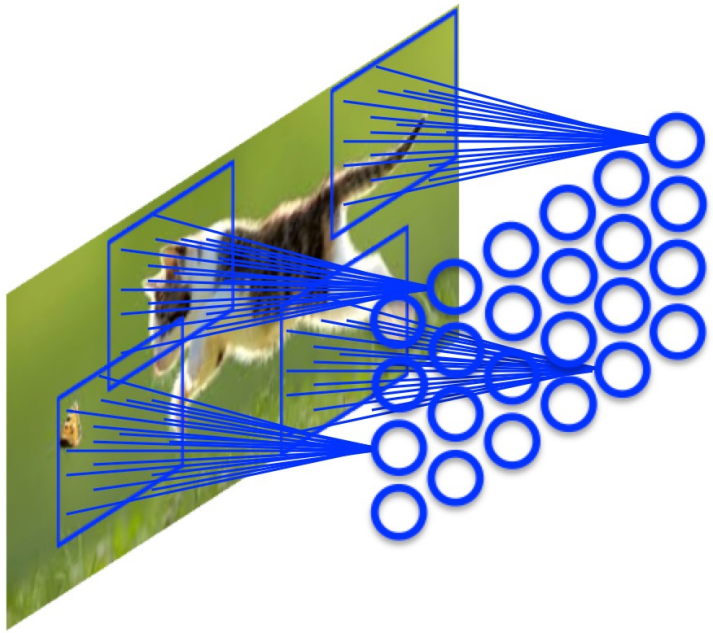
- Example: 10×10 kernel
  - 10^2 weights/neuron

- If hidden layer with a grid of 1000×1000 neurons
  - $10^6\times10^2$ weights!

- Idea: **Stationarity** in vision, the same feature may appear anywhere in the image

Neural networks: Convolutional neural networks

- In Convolutional Neural Networks each layer is **locally connected with shared weights**

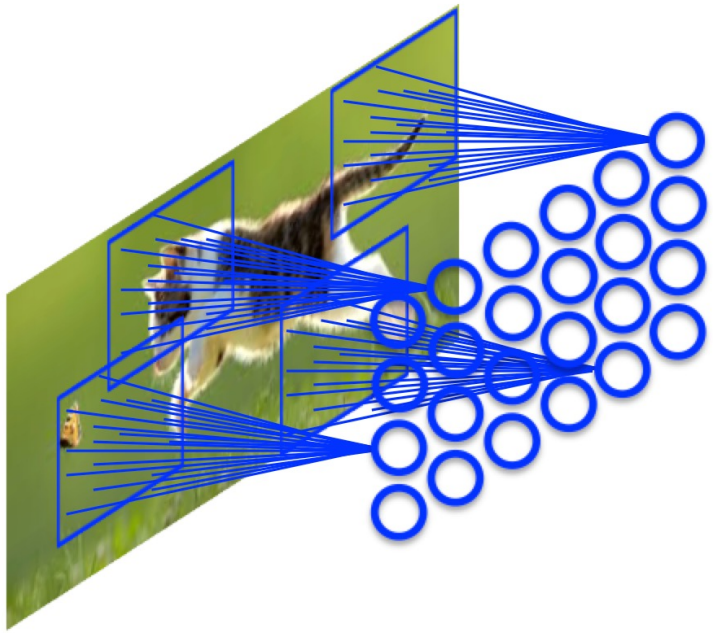


- Each neuron detects the same pattern at a certain location

- Example: 10×10 kernel
- If hidden layer with a grid of 1000×1000 neurons

Neural networks: Convolutional neural networks

- In Convolutional Neural Networks each layer is **locally connected with shared weights**

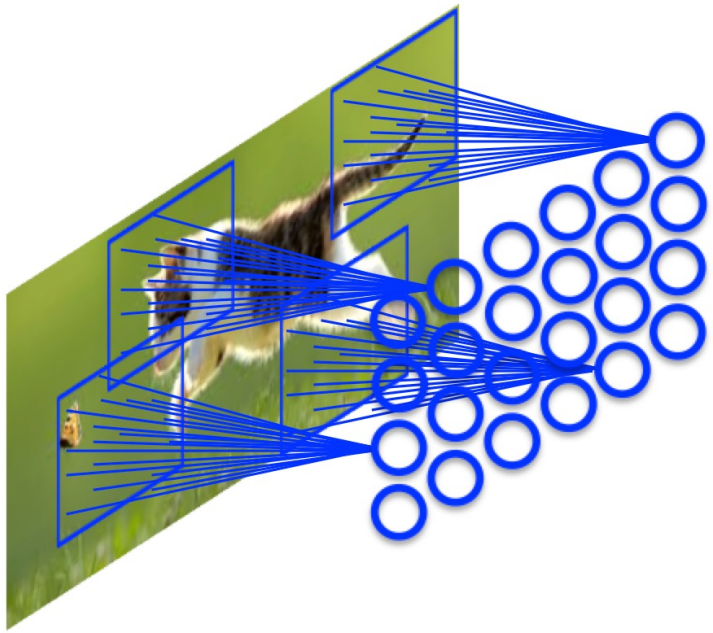


- Each neuron detects the same pattern at a certain location

- Example: 10×10 kernel
- If hidden layer with a grid of 1000×1000 neurons
  - … 10^2 weights!

Neural networks: Convolutional neural networks

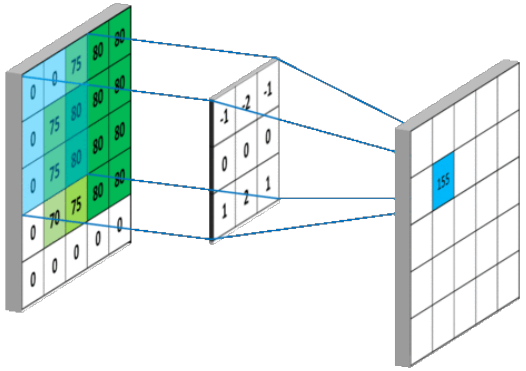
- In Convolutional Neural Networks each layer is **locally connected with shared weights**



- Each neuron detects the same pattern at a certain location

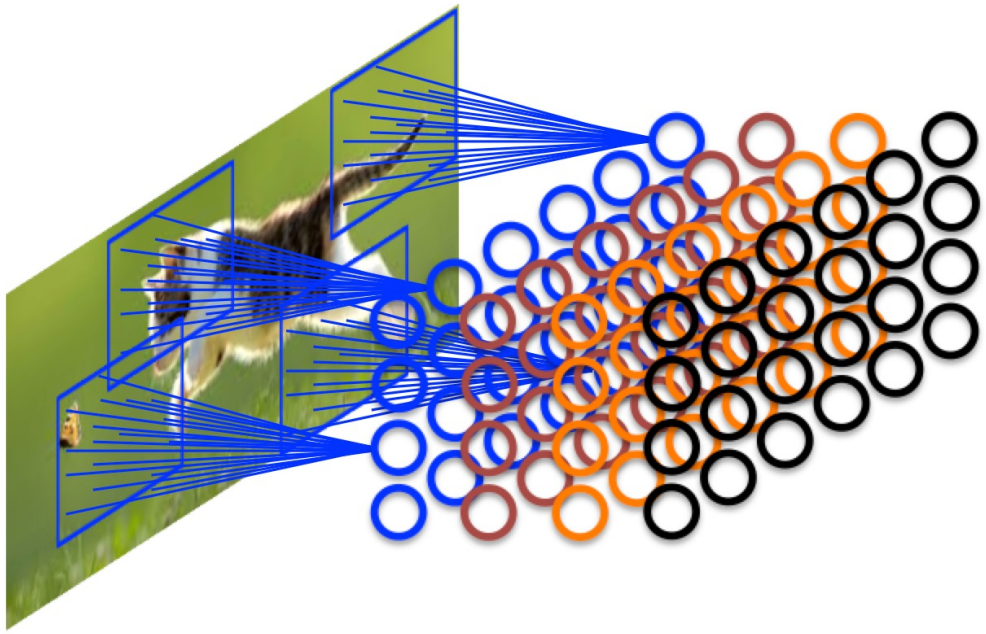
- Example: 10×10 kernel
- If hidden layer with a grid of 1000×1000 neurons
  - … 10^2 weights!

Convolution



Neural networks: Convolutional neural networks

- In Convolutional Neural Networks each layer contains **multiple feature maps**

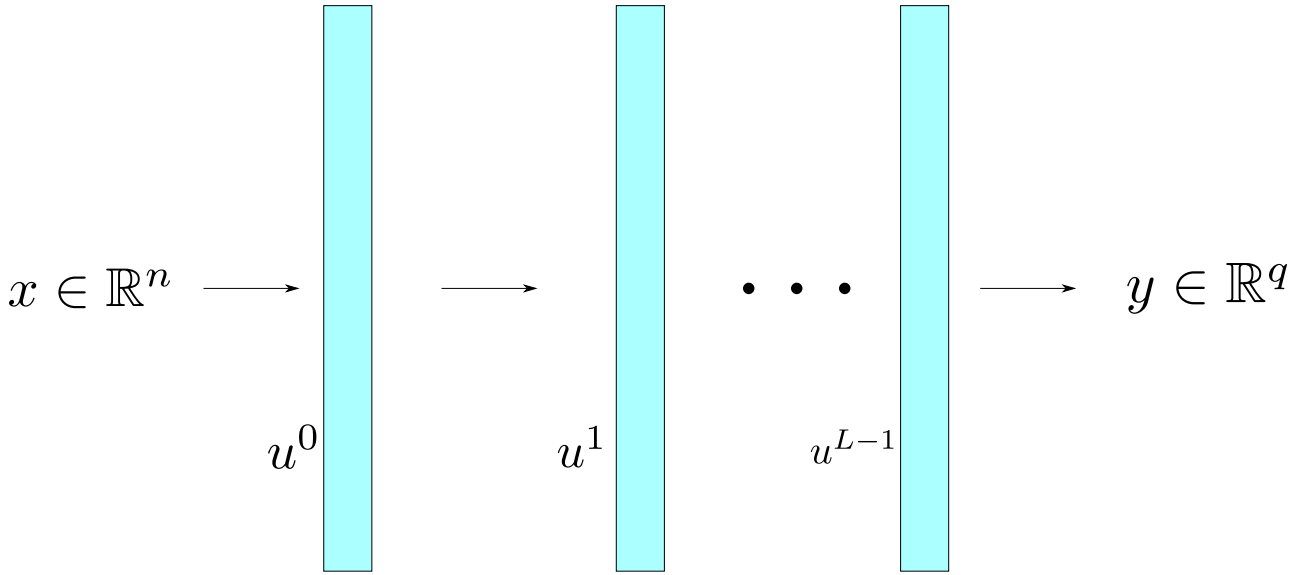


- Each neuron of a given feature map detects the same pattern at a certain location
- Different feature maps detect different patterns
- Example: $(10\times10)\times4$ stack of kernels

Some notation

Notations

- $x \in \mathbb{R}^n$: input vector
- $y \in \mathbb{R}^q$: output vector
- u_ℓ : feature vector at layer ℓ
- θ_ℓ : network parameters at layer ℓ



Neural network with L layers

CNNs

- A **Convolutional Neural Network** (CNN) is simply a concatenation of:
  1. Convolutions (filters)
  2. Additive biases
  3. Down-sampling (“Max-Pooling”)
  4. Non-linearities (activation functions)

- In this lesson, we will be mainly concentrating on **convolutional** and **down-sampling** layers

Convolutional Layers

Convolutional Layers

Convolution operator

Let f and g be two integrable functions. The **convolution operator** $*$ takes as its input two such functions, and outputs another function $h = f * g$, which is defined at any point $t \in \mathbb{R}$ as :

$$h(t) = (f * g)(t) = \int_{-\infty}^{+\infty} f(\tau) g(t - \tau) d\tau.$$

- Intuitively, the function h is defined as the inner product between f and a *shifted* version of g

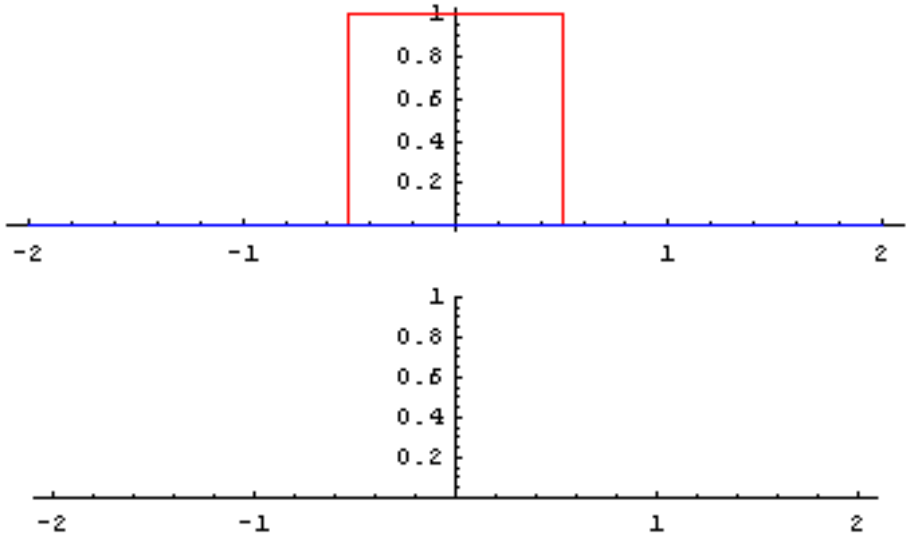
Many of the following slides are adapted from Alasdair Newson's, with his permission

Convolutional Layers

Convolution operator

Let f and g be two integrable functions. The **convolution operator** $*$ takes as its input two such functions, and outputs another function $h = f * g$, which is defined at any point $t \in \mathbb{R}$ as :

$$h(t) = (f * g)(t) = \int_{-\infty}^{+\infty} f(\tau) g(t - \tau) d\tau.$$



Animation: wikipedia

Convolutional Layers

- In many practical applications, in particular for CNNs, we use the **discrete convolution** operator, which acts on discretised functions;

Discrete convolution operator

Let f_n and g_n be two summable series, with $n \in \mathbb{Z}$. The discrete convolution operator is defined as :

$$(f * g)(n) = \sum_{i=-\infty}^{+\infty} f(i)g(n-i)$$

- Intuitively, the function h is defined as the inner product between f and a *shifted* version of g
- In practice, the filter is of small spatial support, around 3×3, or 5×5
- Therefore, only a **small number** of parameters need to be trained (9 or 25 for these filters)

Properties of convolution

1. Associativity : $(f * g) * h = f * (g * h)$
2. Commutativity : $f * g = g * f$
3. Bilinearity : $(\alpha f) * (\beta g) = \alpha\beta(f * g)$, for $(\alpha, \beta) \in \mathbb{R} \times \mathbb{R}$
4. Equivariance to translation : $(f * (g + \tau))(t) = (f * g)(t + \tau)$

Properties of convolution

Associativity, commutativity

- Associativity+commutativity implies that we can carry out convolution in any order
- There is no point in having two or more consecutive convolutions
  - True in fact for any linear map
- Or is there *really* no point… ?

Equivariance to translation

- Equivariance implies that the convolution of any shifted input $(f + \tau) * g$ contains the **same information** as $f * g$
- This is useful, since we want to detect objects **anywhere in the image**

2D Convolution

- Most often, we are going to be working with **images**
- Therefore, we require a 2D convolution operator : this is defined in a very similar manner to 1D convolution :

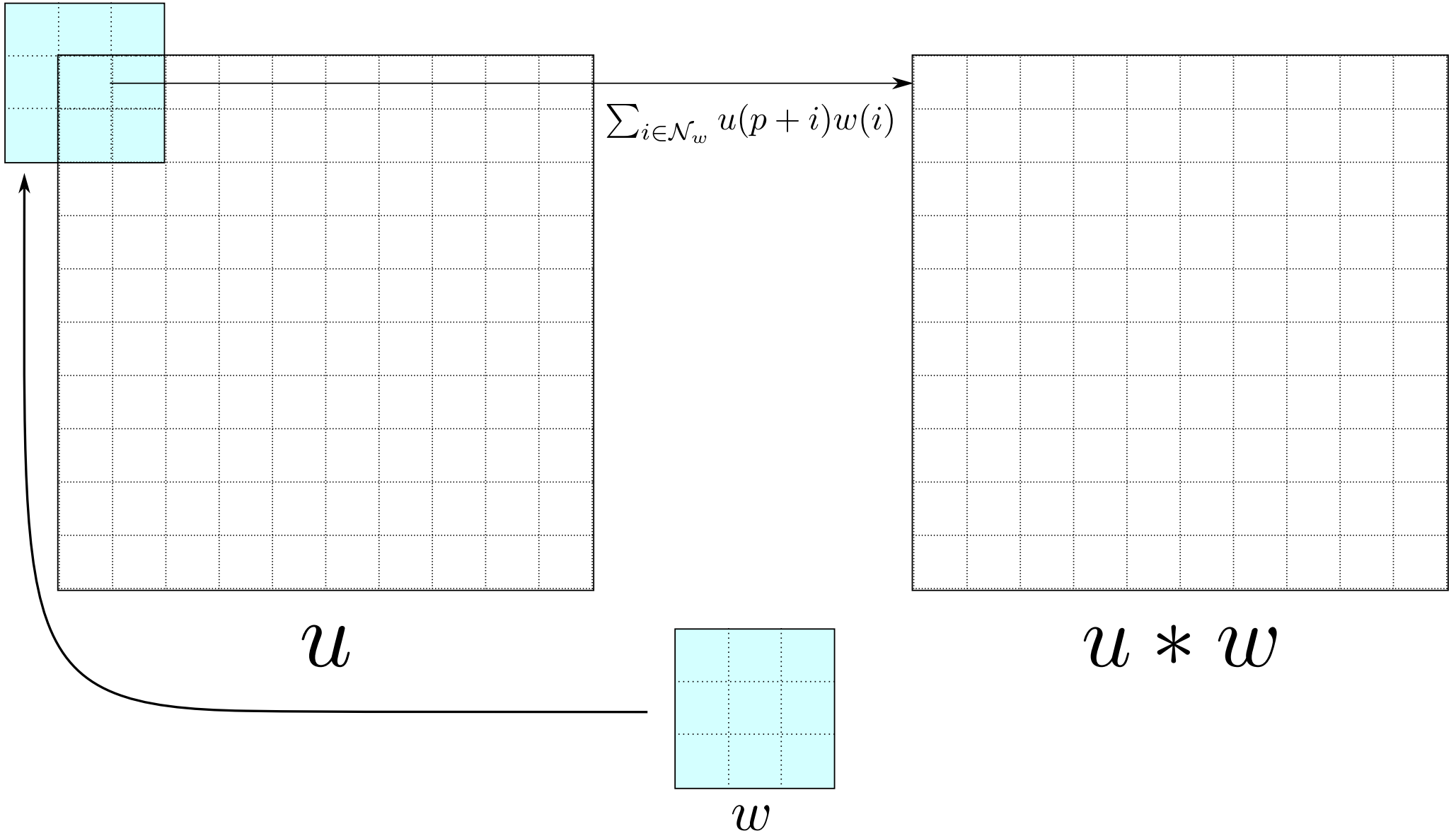
2D convolution operator

$$(f * g)(s,t) = \sum_{i=-\infty}^{+\infty} \sum_{j=-\infty}^{+\infty} f(i,j)g(s-i,t-j)$$

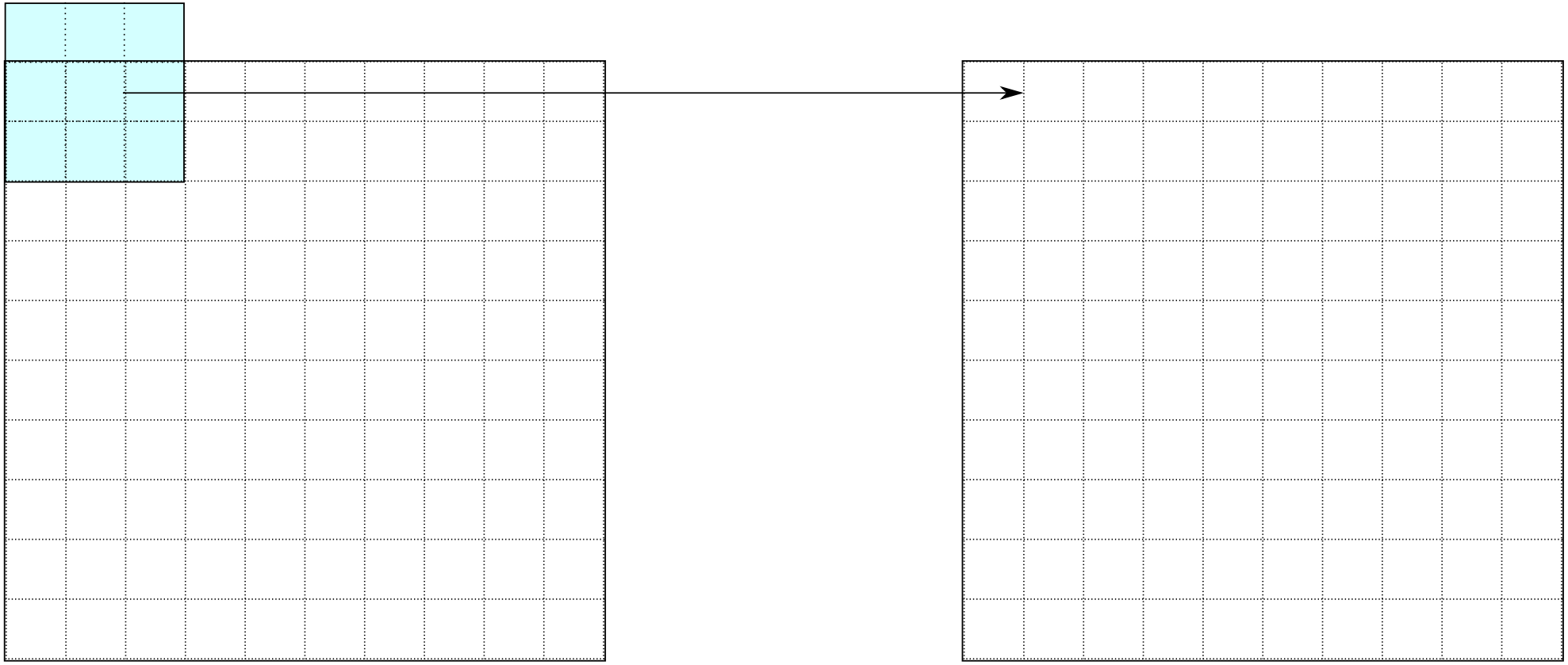
Important remarks for the rest of the lesson!

- We are going to denote the **filters with** w
- For lighter notation, we write $w(i) =: w_i$ (and the same for x_i etc.)

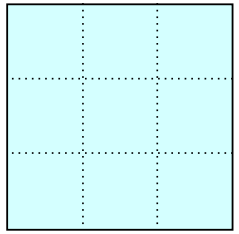
2D Convolution – Illustration



2D Convolution – Illustration



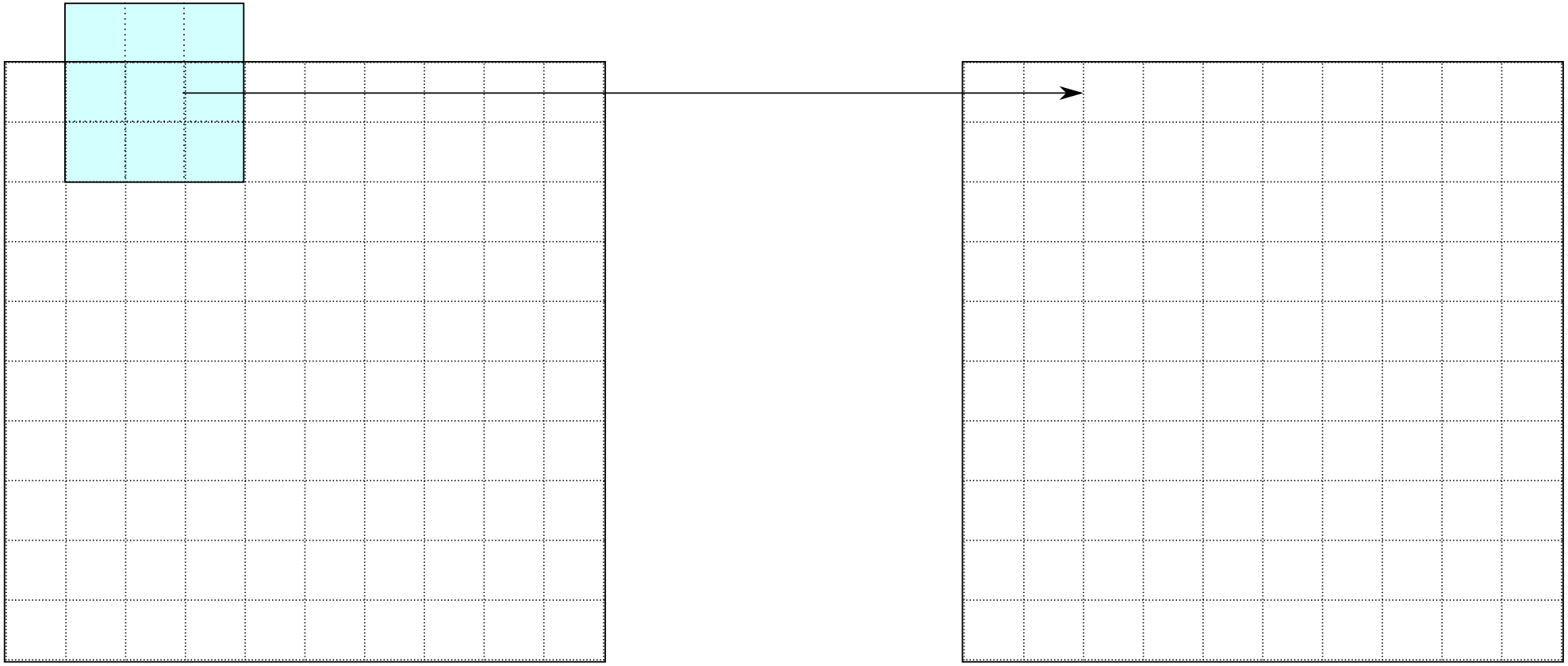
u



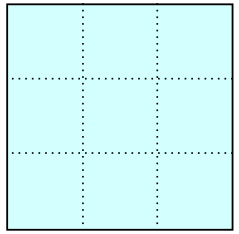
w

$u * w$

2D Convolution – Illustration



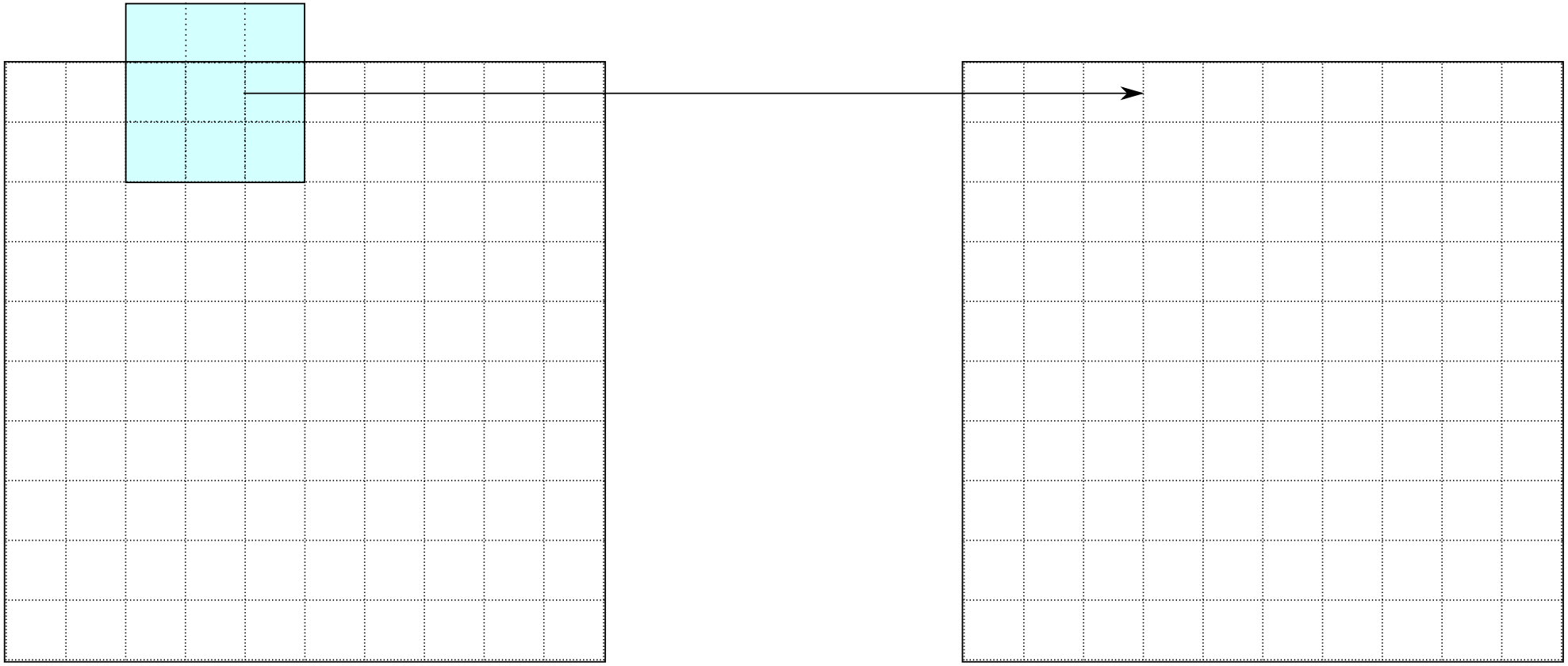
u



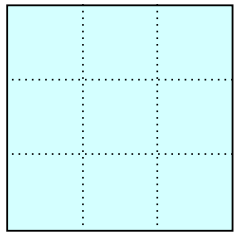
w

$u * w$

2D Convolution – Illustration



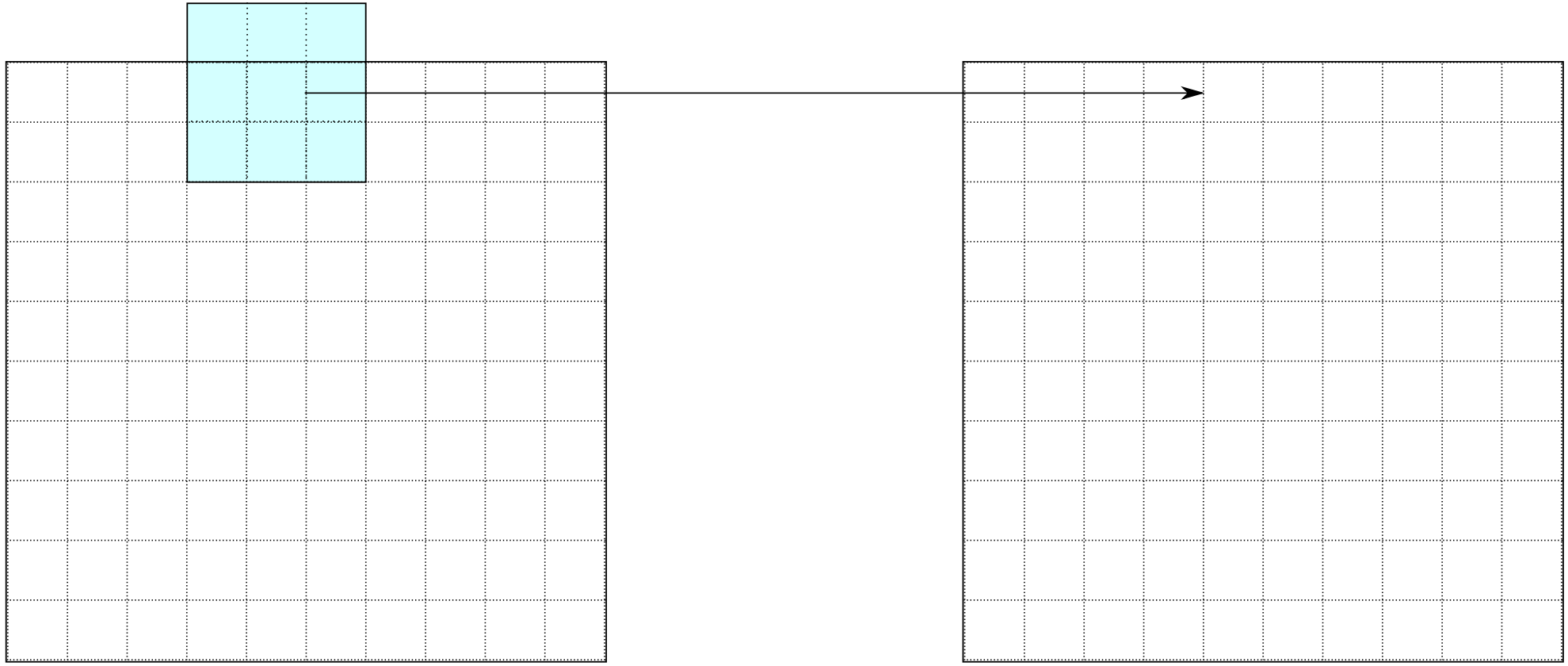
u



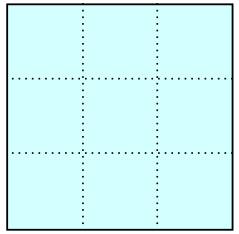
w

$u * w$

2D Convolution – Illustration



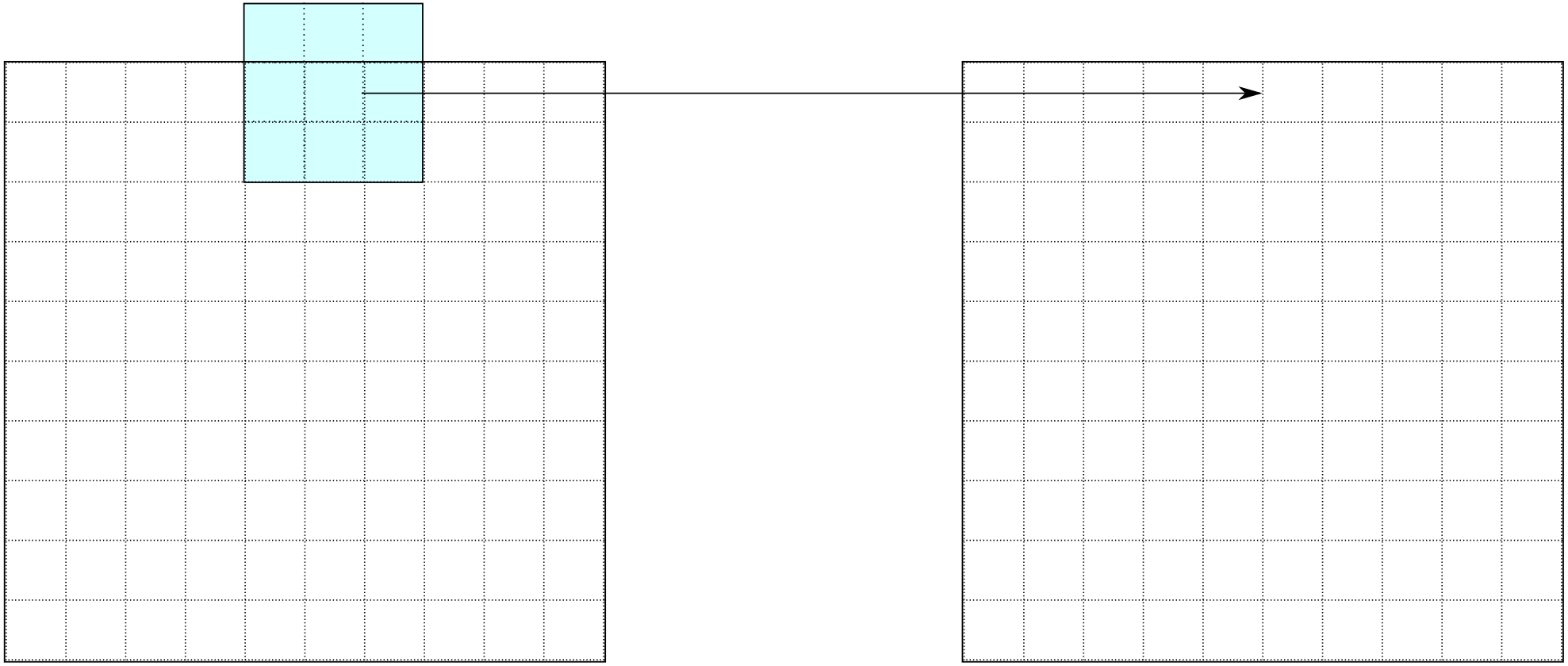
u



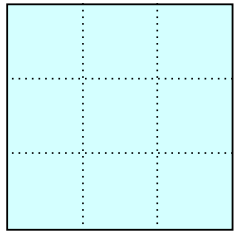
w

$u * w$

2D Convolution – Illustration



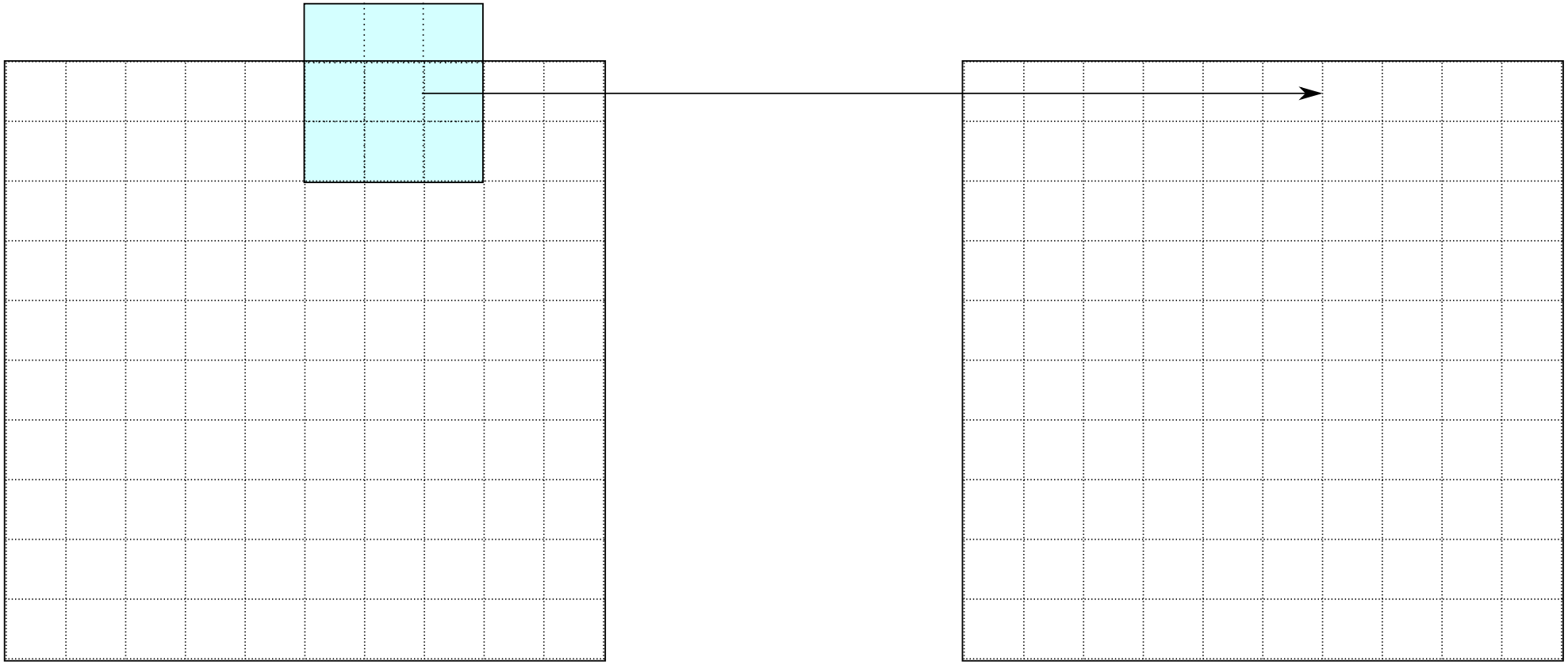
u



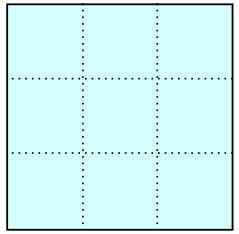
w

$u * w$

2D Convolution – Illustration



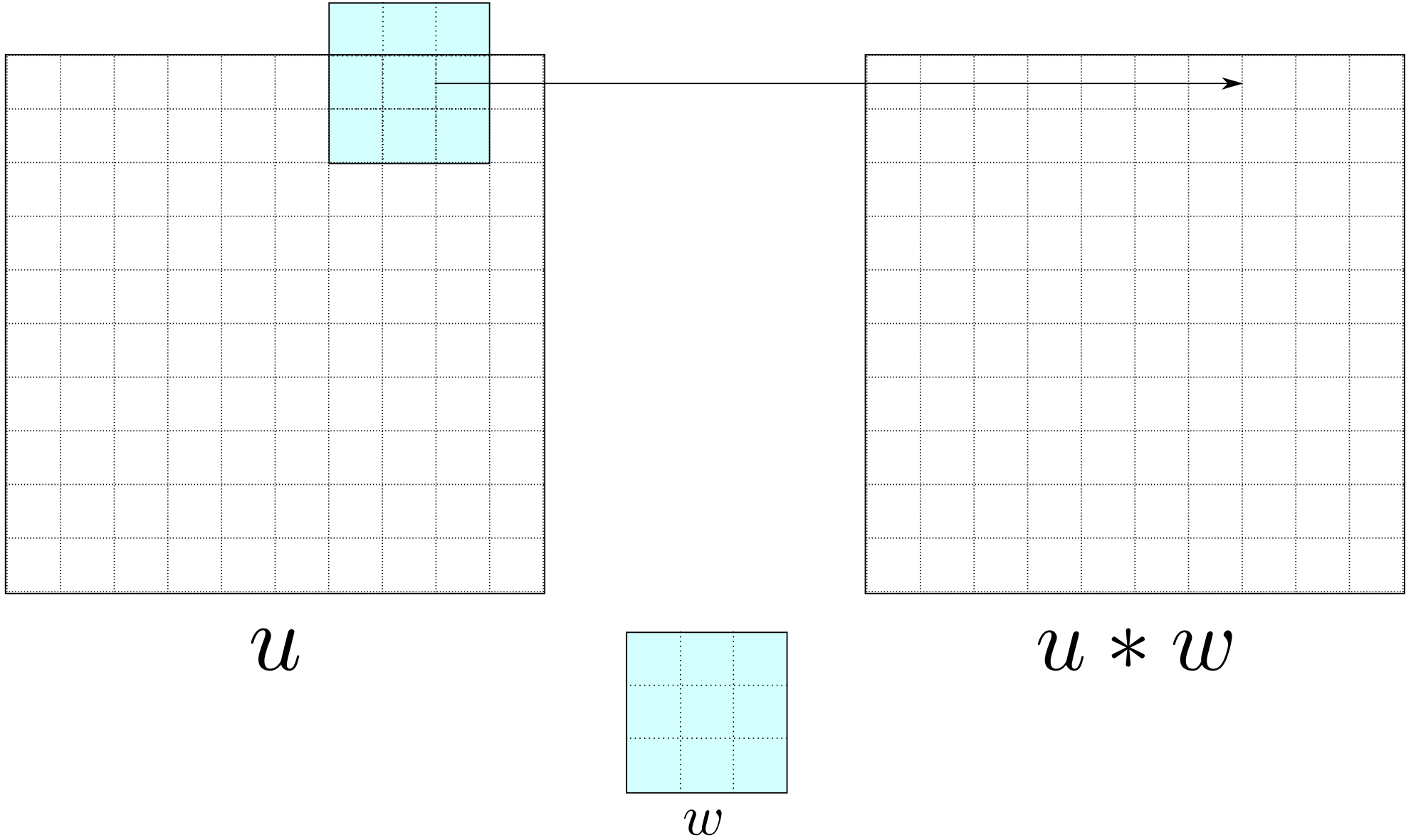
u



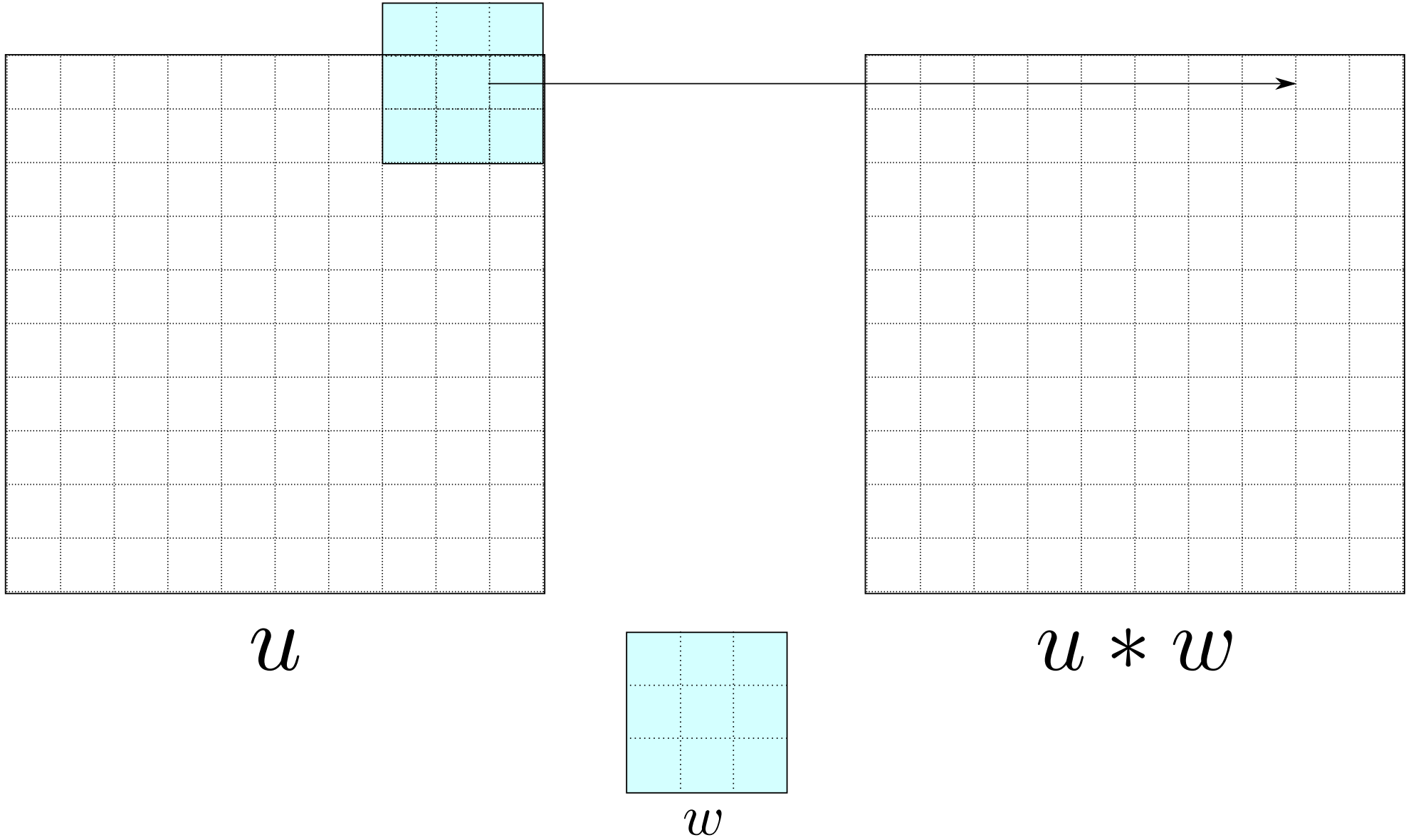
w

$u * w$

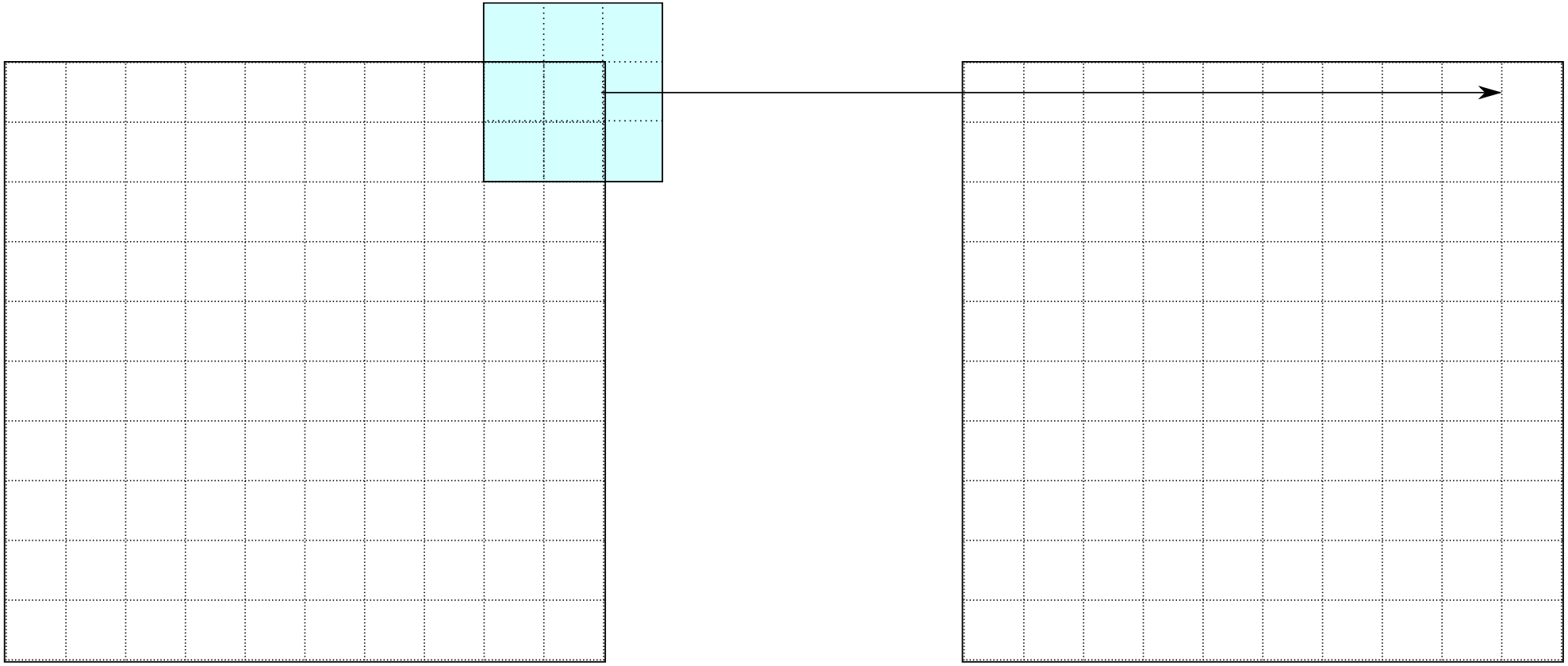
2D Convolution – Illustration



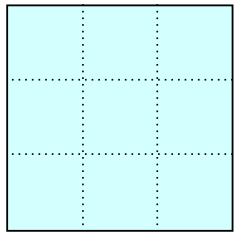
2D Convolution – Illustration



2D Convolution – Illustration



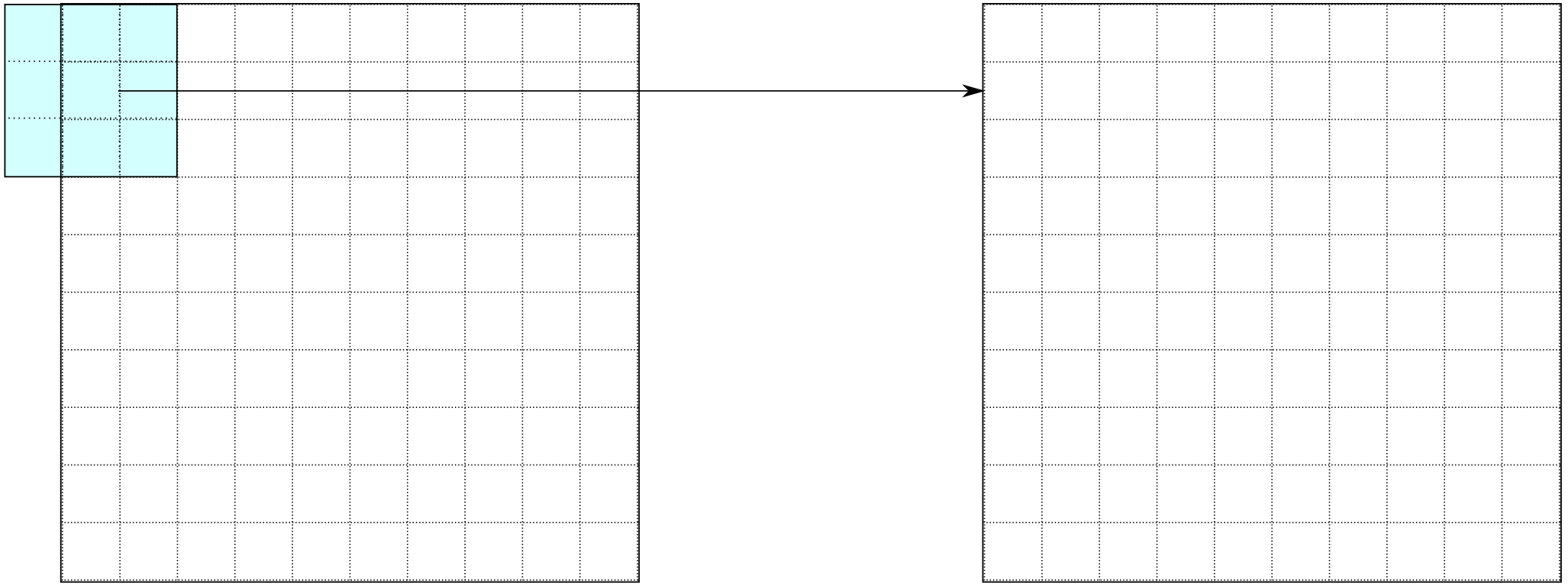
u



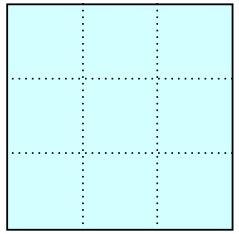
w

$u * w$

2D Convolution – Illustration



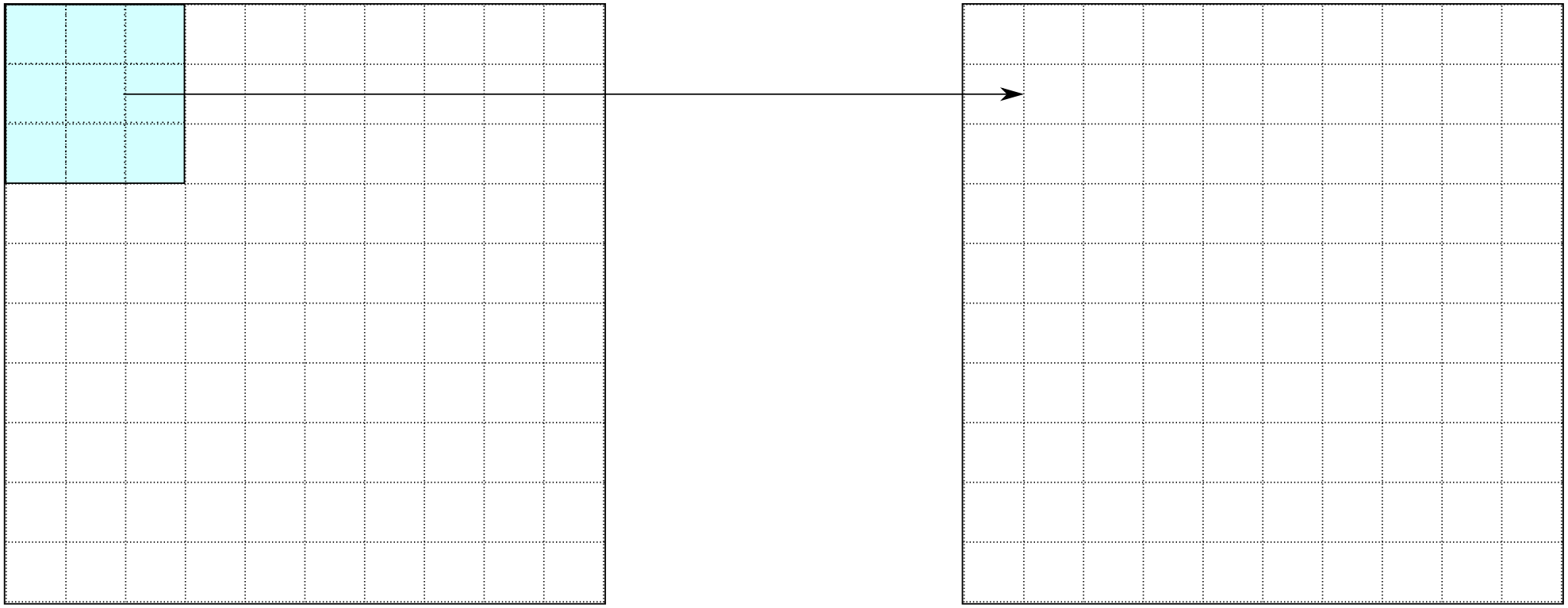
u



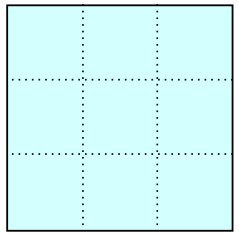
w

$u * w$

2D Convolution – Illustration



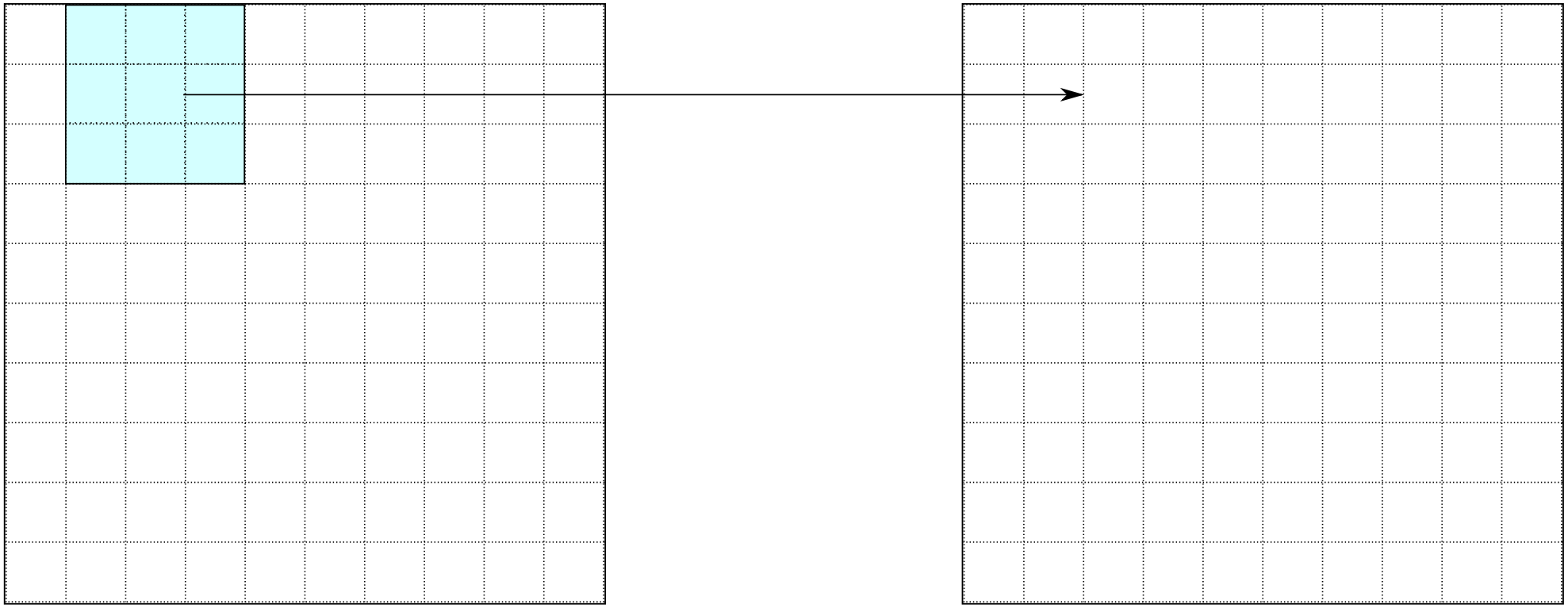
u



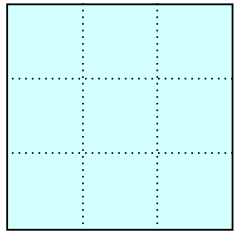
w

$u * w$

2D Convolution – Illustration



u



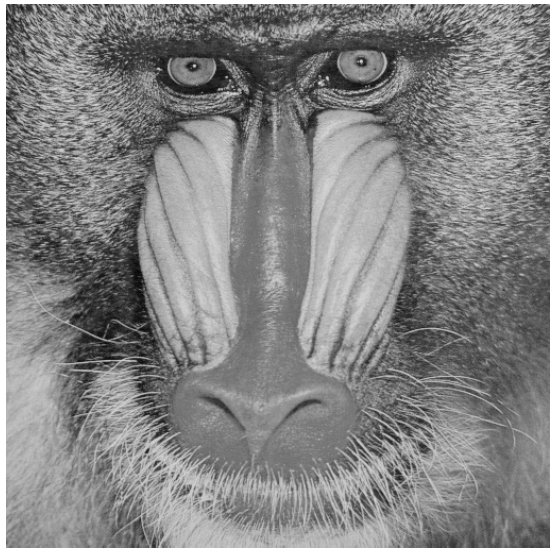
w

$u * w$

Examples

- The filter weights w_i determine what type of feature can be detected by convolutional layers
- Example, Sobel filters:

Input image

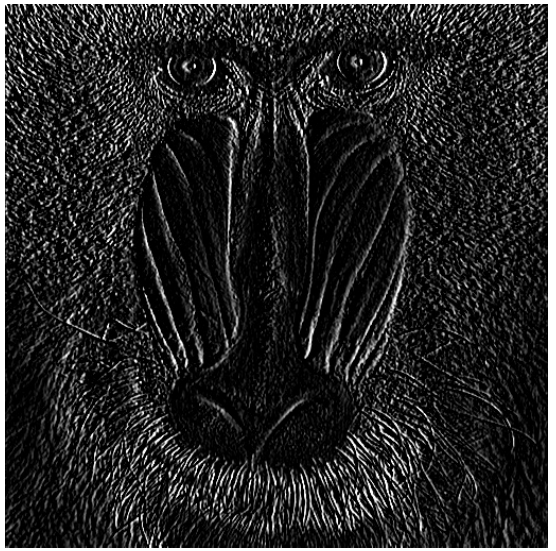


Horizontal edge

$$\begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}$$

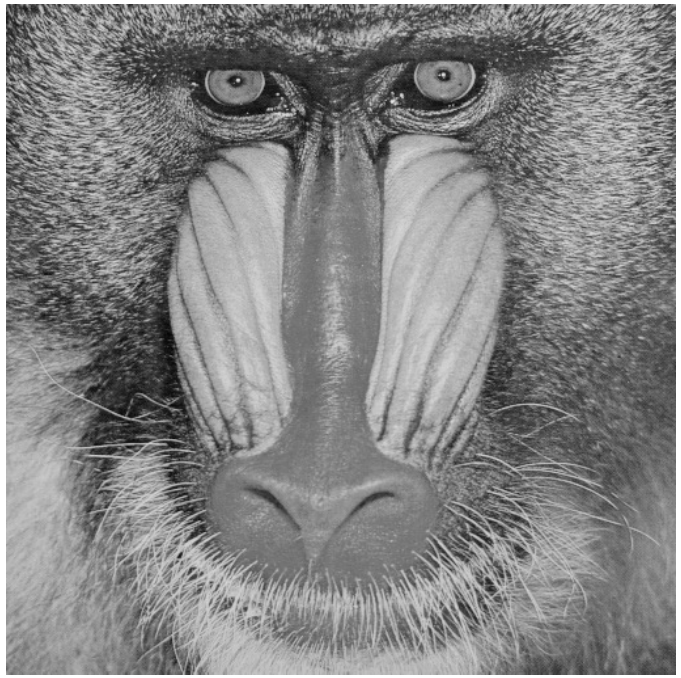
Vertical edge

$$\begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}$$

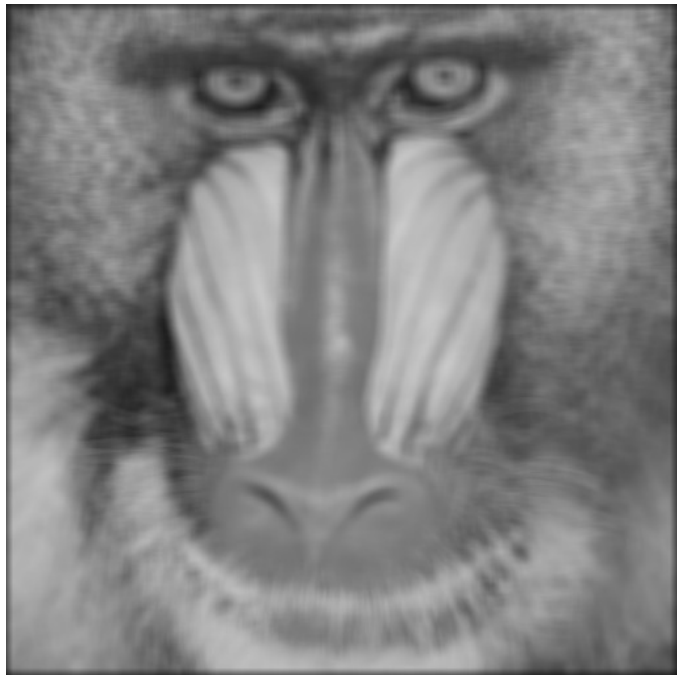


Examples

- Convolutional filters can also act as low-pass/smoothing filters



Input image



Smoothed image

Convolutions are linear operators

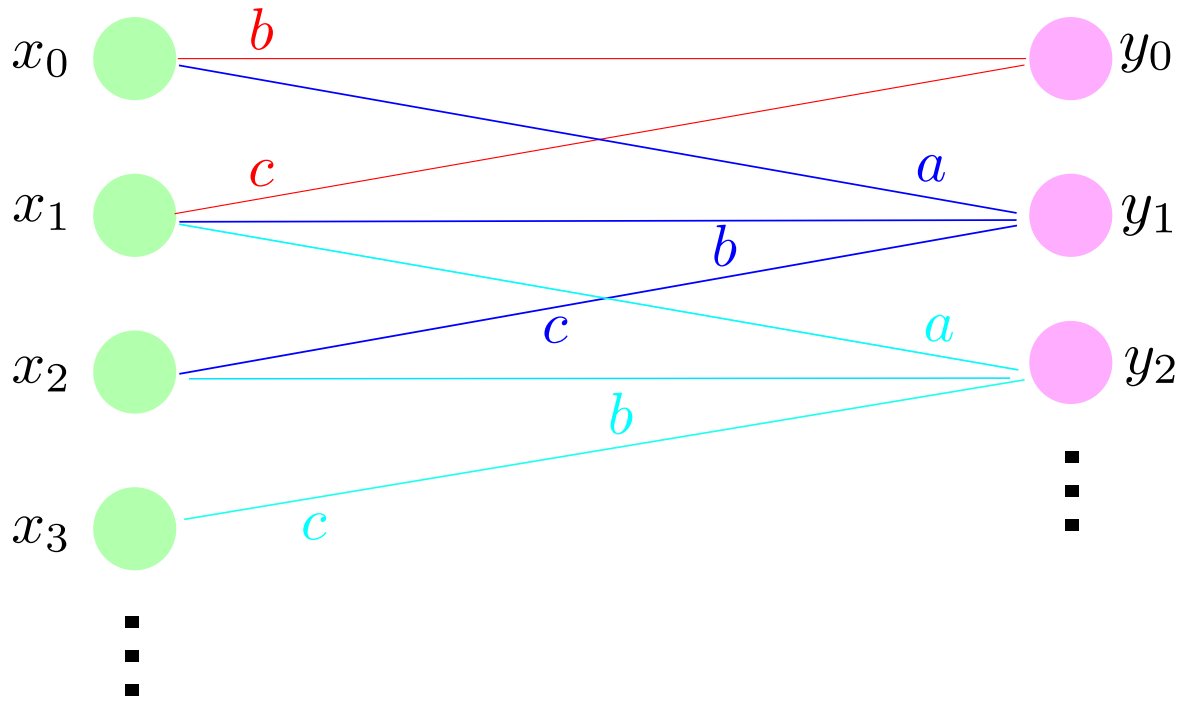
- We can also write convolution as a matrix/vector product, as in the case of fully connected layers

$$w = \begin{pmatrix} 0 & -1 & 0 \\ -1 & 4 & -1 \\ 0 & -1 & 0 \end{pmatrix} \rightarrow A_w = \begin{array}{c} \\ \\ \\ \\ \\ \\ \downarrow \\ \mathsf{K} \end{array} \overbrace{\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad}^{\mathsf{n}} \begin{pmatrix} 4 & -1 & \overset{0}{\cdots} & -1 & \overset{0}{\cdots} & & \\ -1 & 4 & -1 & \overset{0}{\cdots} & -1 & \overset{0}{\cdots} & \\ 0 & -1 & 4 & -1 & \overset{0}{\cdots} & -1 & \overset{0}{\cdots} \\ \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \\ -1 & & \overset{0}{\cdots} & -1 & \overset{0}{\cdots} & -1 & 4 \end{pmatrix}$$

- This further illustrated the drastic reduction in weight parameters (9 instead of Kn)
- Can be useful to view convolution in this manner (we will see this later)

Convolutional Layers

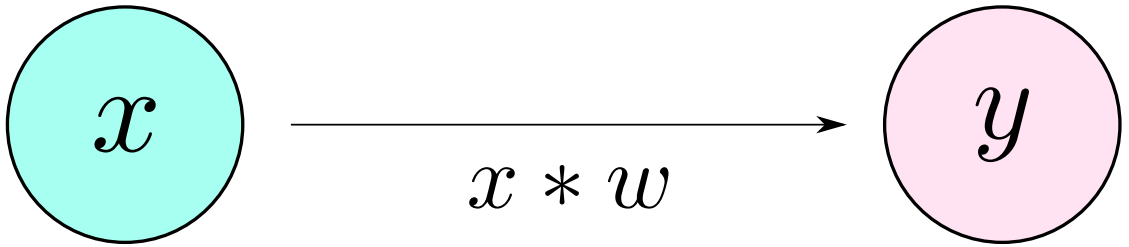
- At this point, it is good to have a more “neural network”-based illustration of convolutional layers



- We can see two of the main justifications for CNNs:
  1. **Sparse connectivity**
  2. **Weight sharing**

Convolutional Layers – Optimization

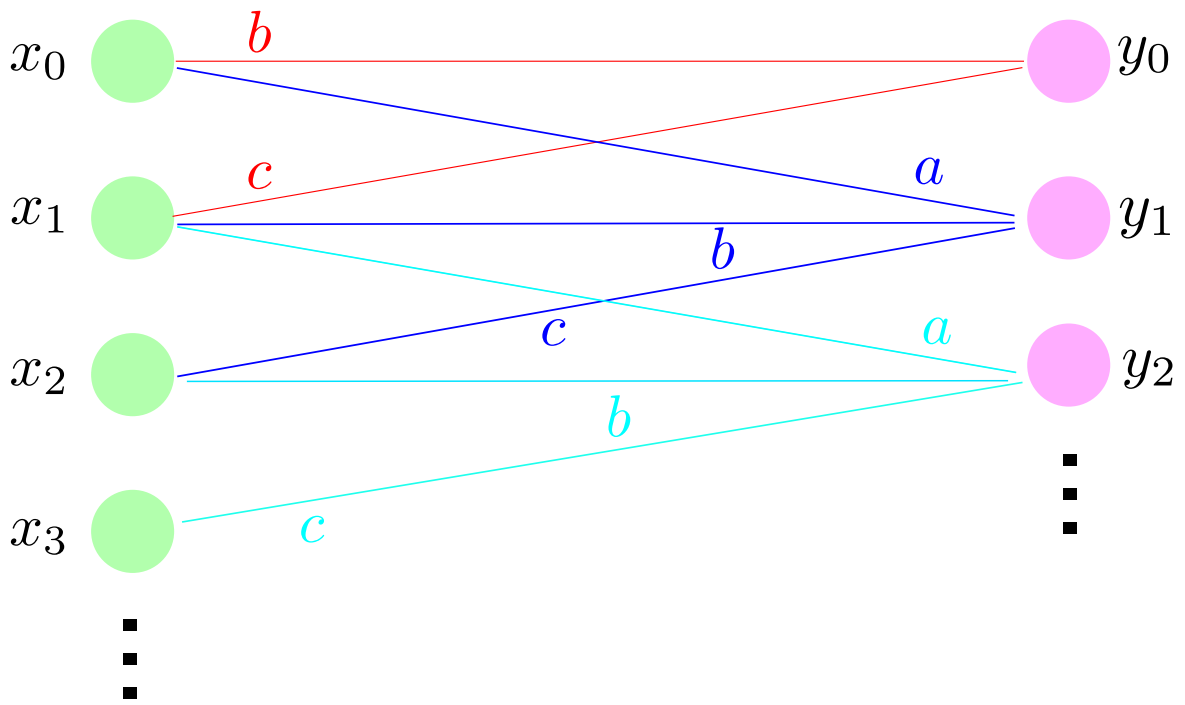
- Now that we understand convolution, how do we **optimize** a neural network with convolutional layers ? **Back-propagation**
- Consider a layer with just a convolution with w



- We have the derivatives $\frac{\partial \mathcal{L}}{\partial y_i}$ available
- We want to calculate the following quantities:
  - $\frac{\partial \mathcal{L}}{\partial x_j}$ (for further backpropagation) and
  - $\frac{\partial \mathcal{L}}{\partial w_k}$

- We shall use the abbreviation $\frac{\partial \mathcal{L}}{\partial y_i} =: dy_i$

Optimization – Example



- Say we want to calculate $dx_1 := \frac{\partial \mathcal{L}}{\partial x_1}$

Convolutional Layers – Optimization

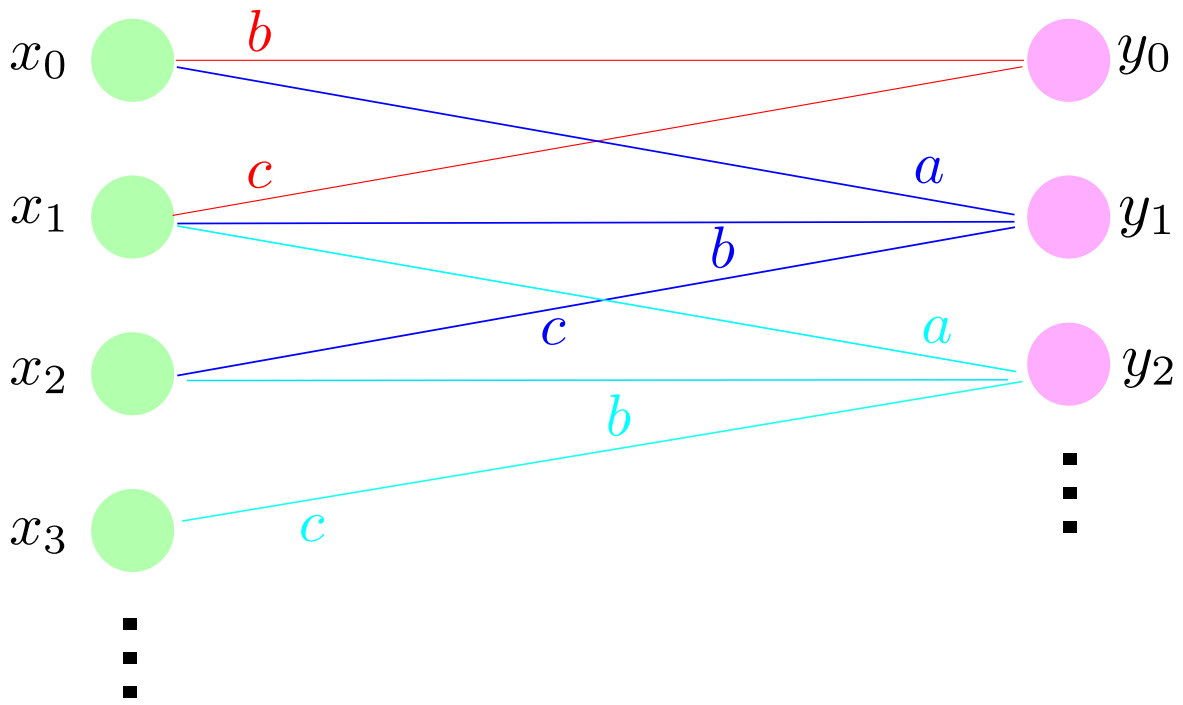
- Each element y_i depends on the inputs x_j and the weights w_k
- Therefore, the loss is a function of several variables (shortened as $\boldsymbol{x}$, $\boldsymbol{w}$)

$$\mathcal{L} := \mathcal{L}(y_1(\boldsymbol{x}, \boldsymbol{w}), \cdots, y_m(\boldsymbol{x}, \boldsymbol{w}))$$

- We use the multi-variate chain rule

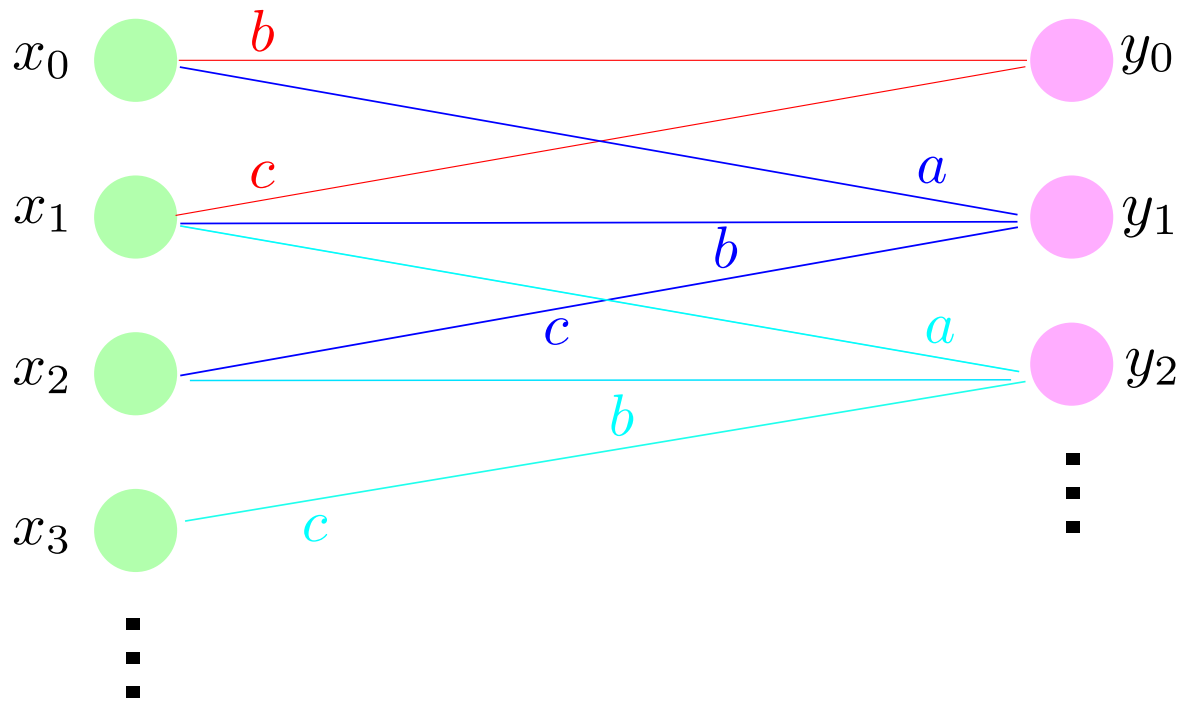
$$dx_1 = \sum_i \frac{\partial \mathcal{L}}{\partial y_i} \frac{\partial y_i}{\partial x_1}$$

Optimization – Example



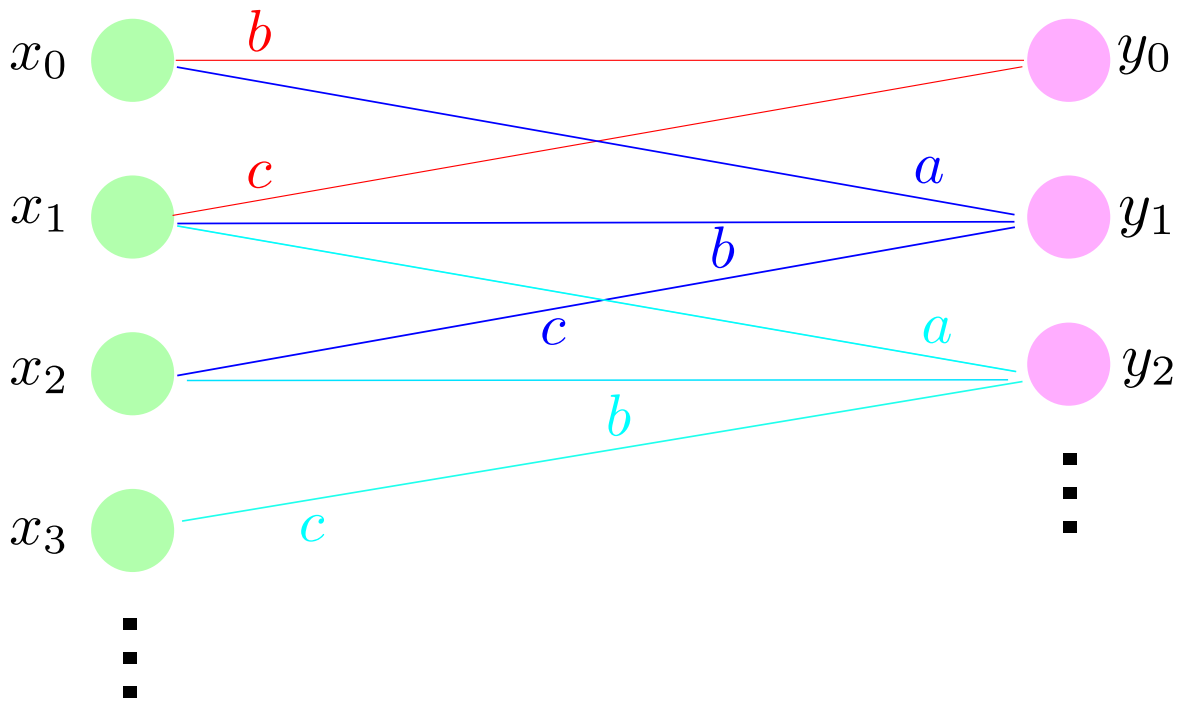
$dx_1 = ???$

Optimization – Example



$$dx_1 = dy_0 \frac{\partial y_0}{\partial x_1} + dy_1 \frac{\partial y_1}{\partial x_1} + dy_2 \frac{\partial y_2}{\partial x_1} = dy_0 \cdot c + dy_1 \cdot b + dy_2 \cdot a$$

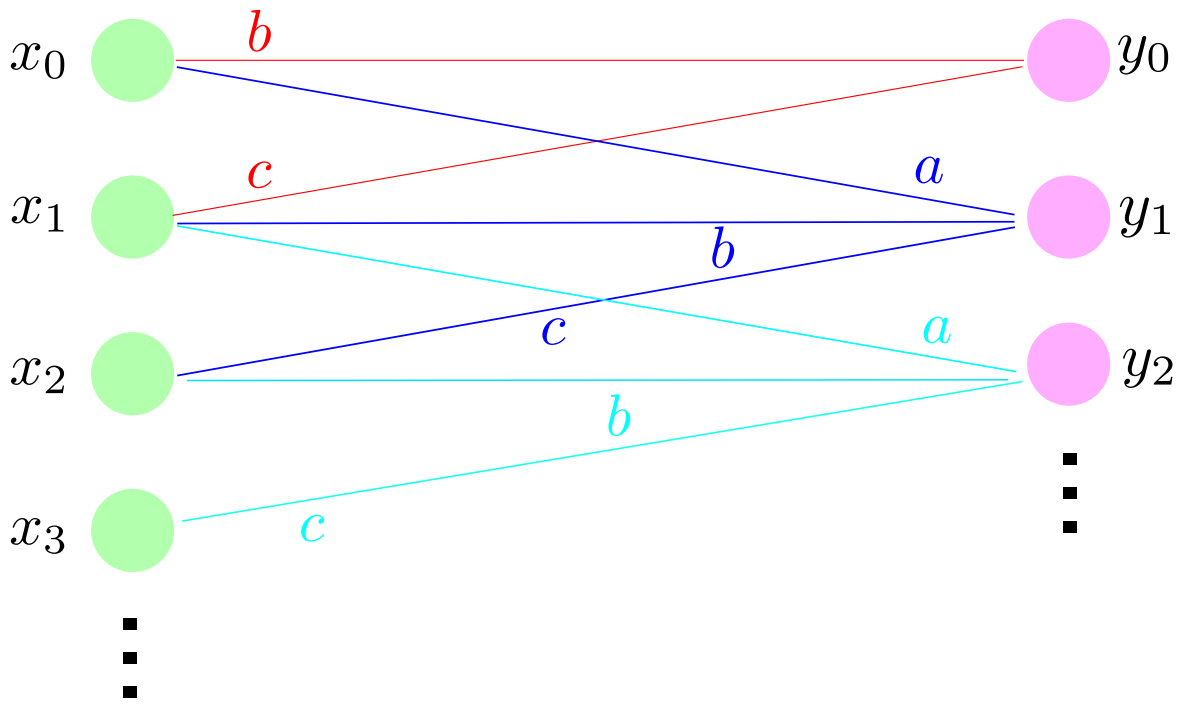
Optimization – Example



$$dx_1 = dy_0 \frac{\partial y_0}{\partial x_1} + dy_1 \frac{\partial y_1}{\partial x_1} + dy_2 \frac{\partial y_2}{\partial x_1} = dy_0 \cdot c + dy_1 \cdot b + dy_2 \cdot a$$

$$dx_2 = dy_1 \frac{\partial y_1}{\partial x_2} + dy_2 \frac{\partial y_2}{\partial x_2} + dy_3 \frac{\partial y_3}{\partial x_2} = dy_1 \cdot c + dy_2 \cdot b + dy_3 \cdot a$$

Optimization – Example



$$dx_1 = dy_0 \frac{\partial y_0}{\partial x_1} + dy_1 \frac{\partial y_1}{\partial x_1} + dy_2 \frac{\partial y_2}{\partial x_1} = dy_0 \cdot c + dy_1 \cdot b + dy_2 \cdot a$$

$$dx_2 = dy_1 \frac{\partial y_1}{\partial x_2} + dy_2 \frac{\partial y_2}{\partial x_2} + dy_3 \frac{\partial y_3}{\partial x_2} = dy_1 \cdot c + dy_2 \cdot b + dy_3 \cdot a$$

- As we can see, the order of the **weights is flipped**

Convolutional Layers – Optimization

- Now, let us calculate $dx_j := \frac{\partial \mathcal{L}}{\partial x_j}$ for any j

$$dx_j = \sum_i dy_i \frac{\partial y_i}{\partial x_j}$$

$$= \sum_i dy_i \frac{\partial (x * w)_i}{\partial x_j}$$

$$= \sum_i dy_i \frac{\partial (\sum_k x_k w_{i-k})}{\partial x_j}$$

$$= \sum_i dy_i \, w_{i-j} = \sum_i dy_i \, w_{-(j-i)}$$

- More compactly $dx_j = \big(dy * \mathrm{flip}(w)\big)_j$ for any j

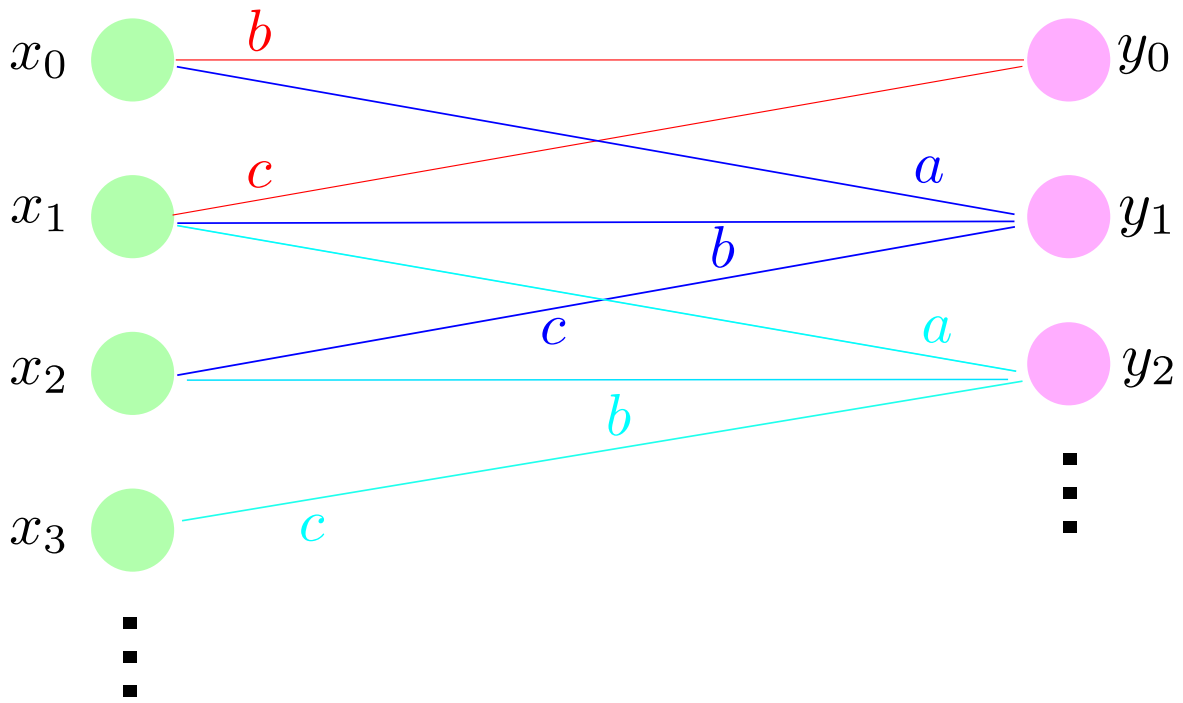
Convolutional Layers – Optimization

- Recall that the convolutional operator can be written $y = A_w x$ with A_w the convolution matrix
- The flipping of the weights corresponds to a transpose of A_w

$$dx = A_w^T \, dy$$

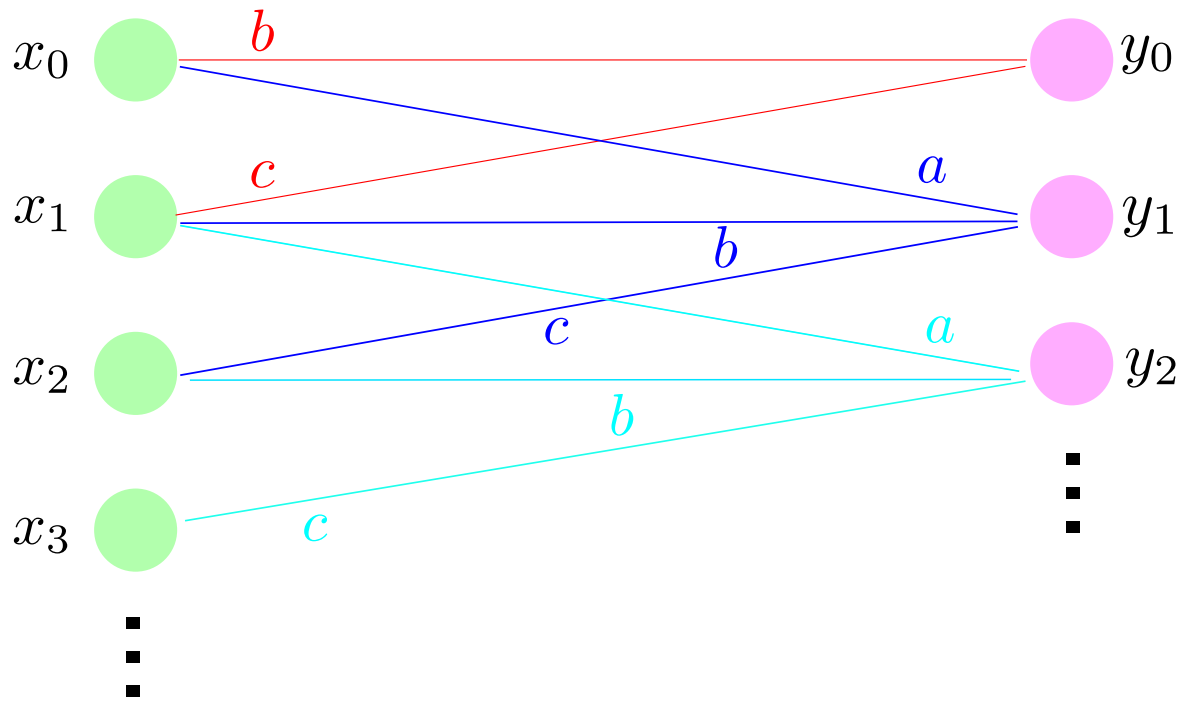
Optimization – Example

- Now for the second part: $dw_k := \frac{\partial \mathcal{L}}{\partial w_k}$



Optimization – Example

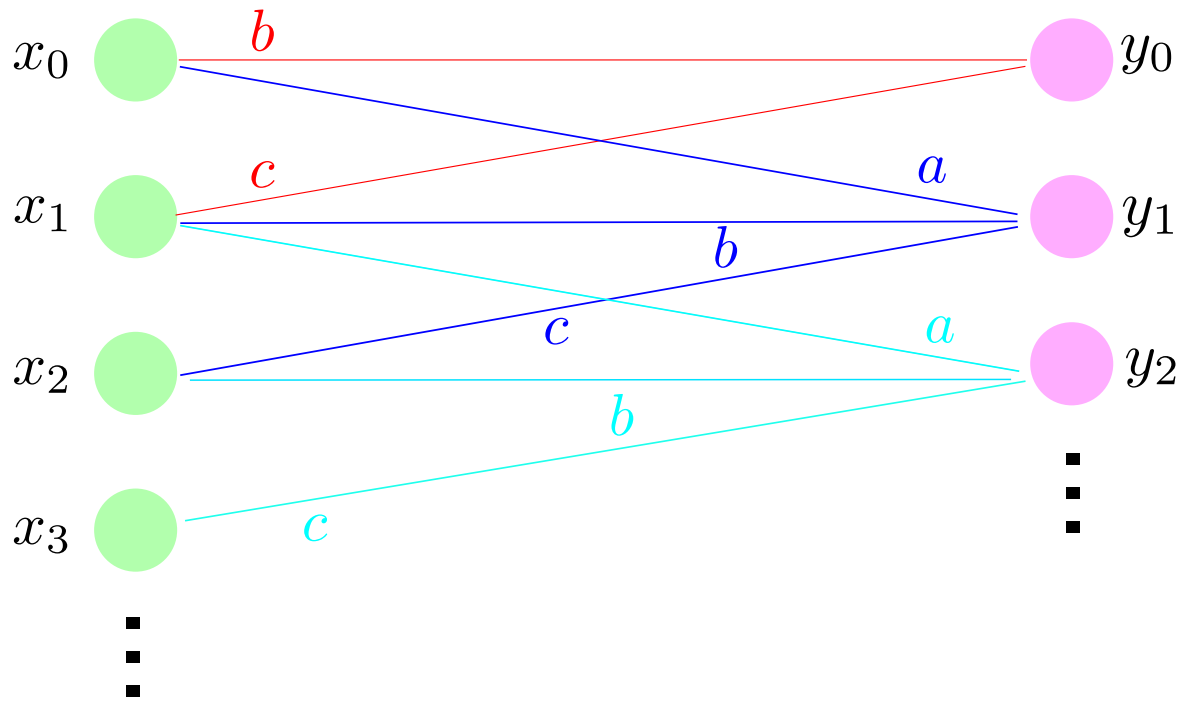
- Now for the second part: $dw_k := \frac{\partial \mathcal{L}}{\partial w_k}$



- Again, we use the chain rule. For example, $da = \sum_i \frac{\partial \mathcal{L}}{\partial y_i} \frac{\partial y_i}{\partial a}$

Optimization – Example

- Now for the second part: $dw_k := \frac{\partial \mathcal{L}}{\partial w_k}$



- We have $y_i = ax_{i-1} + bx_i + cx_{i+1}$

$$da = \sum_i dy_i x_{i-1}$$

Convolutional Layers – Optimization

- In the general case, we have:

$$dw_k = \sum_i \frac{\partial \mathcal{L}}{\partial y_i} \frac{\partial y_i}{\partial w_k}$$

$$= \sum_i dy_i \frac{\partial (x * w)_i}{\partial w_k}$$

$$= \sum_i dy_i \frac{\partial (\sum_j x_{i-j} w_j)}{\partial w_k}$$

$$= \sum_i dy_i \, x_{i-k} = \sum_i dy_i \, x_{-(k-i)}$$

- More compactly $dw_k = \big(dy * \mathrm{flip}(x)\big)_k$ for any k

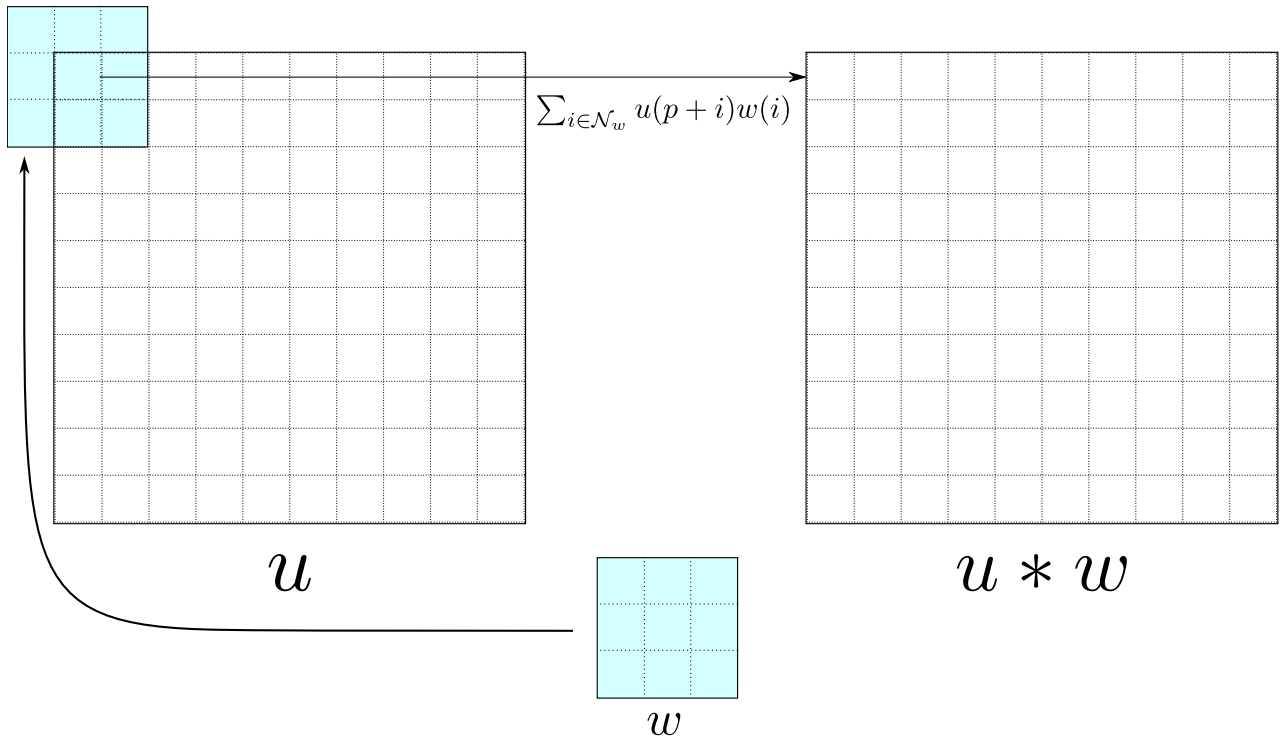
Convolutional Layers – Optimization

- Note: optimization of loss w.r.t. one parameter w_k **involves entire image**
- **Weights are "shared" across the entire image**
- This notion of **weight sharing** is one of the main justifications for using CNNs
- In practice, we do not calculate dw_k and dx_j ourselves, we use the **automatic differentiation** tools of TensorFlow, Pytorch, etc.

Convolutional Layers – border conditions

- The convolution operator poses a problem at the borders

- Theoretically, we consider functions defined over an infinite domain, but which have compact support
- In reality, we only have finite vectors/matrices to work on

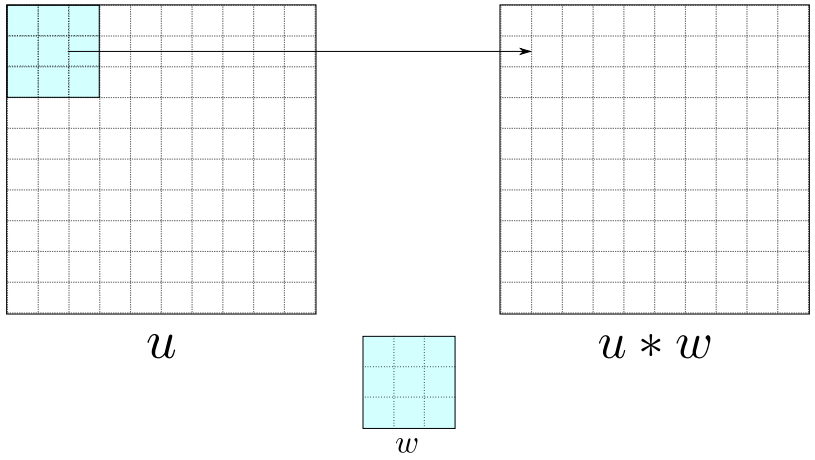


Convolutional Layers – border conditions

Two common approaches to border conditions

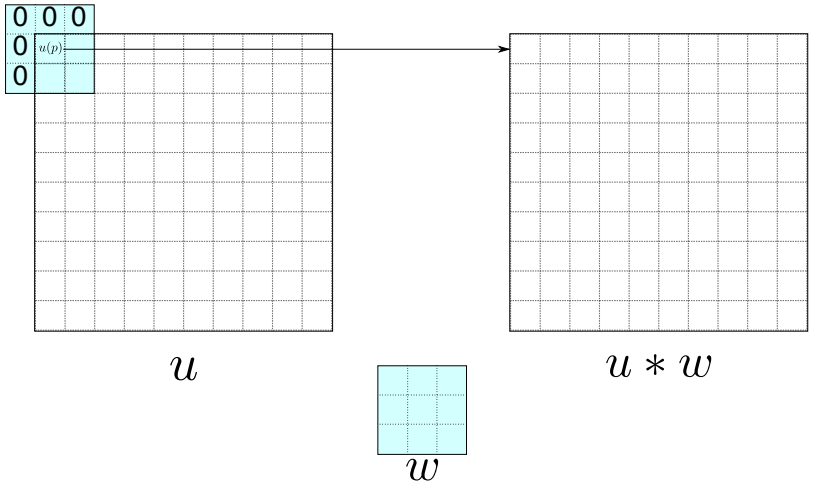
"VALID" approach

- Only take shift/dot products that do not extend beyond $\mathrm{Supp}(u)$
- Output size: $m - |w| + 1$



"SAME" approach

- Keep output size m
- Need to choose values outside of $\mathrm{Supp}(u)$: zero-padding



2D+feature convolution

- Several filters are used per layer, let us say K filters: $\{w^1, \cdots, w^K\}$

- The resulting vectors/images are then stacked together to produce the next layer's input $u^{l+1} \in \mathbb{R}^{m \times n \times K}$

$$u^{l+1} = [u * w^1, \cdots, u * w^K]$$

- Therefore, the next layer's weights must have a depth of K. The 2D convolution with an image of depth K is defined as

$$(u * w)_{y,x} = \sum_{i,j,k} u(i,j,k) w(y-i, x-j, k)$$

Useful explanation : https://towardsdatascience.com/a-comprehensive-introduction-to-different-types-of-convolutions-in-deep-learning-669281e58215

Illustration – multiple feature maps

- Several filters are learnt per layer, let us say K filters; the resulting K **feature maps are stacked**

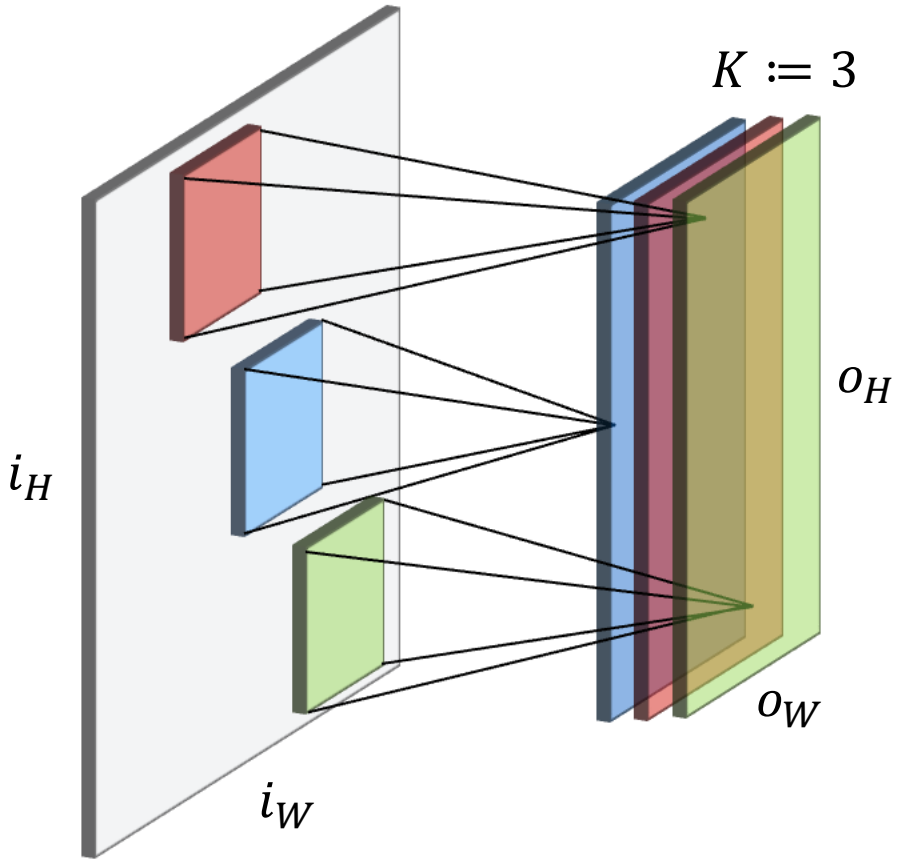


Illustration – multiple input channels, 1 feature map

- The input image can have a depth of K; **filters extend the full depth** of the input image
  - A single filter $= s_H \times s_W \times K$ tensor

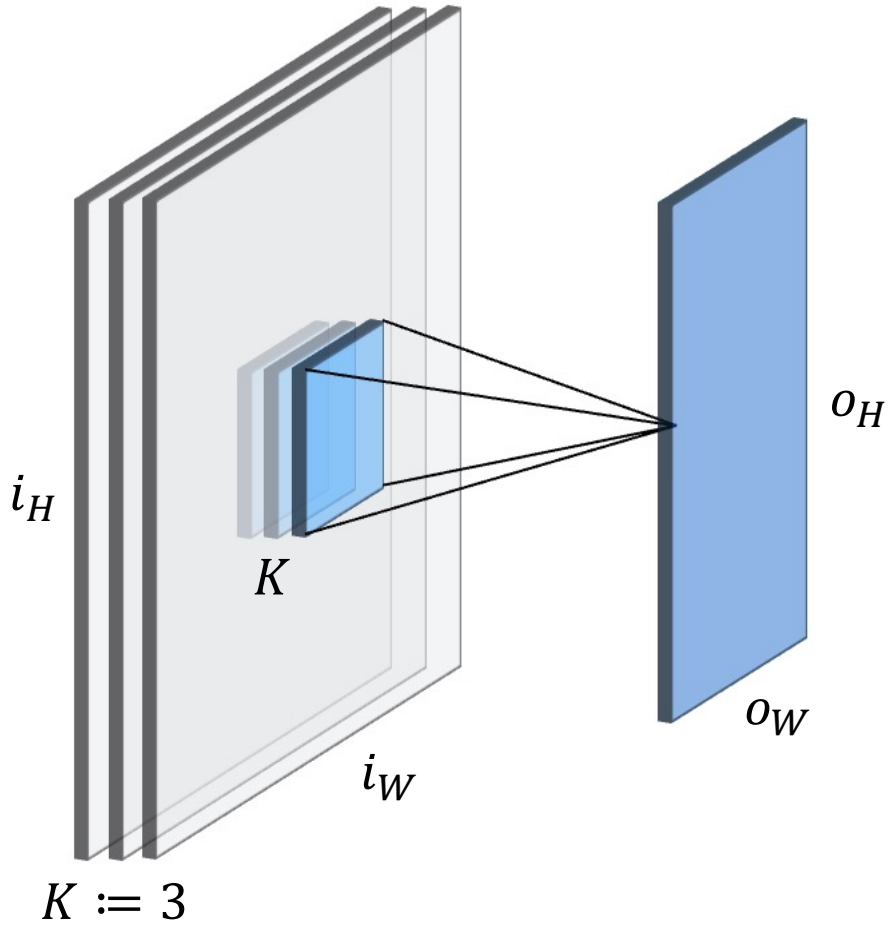
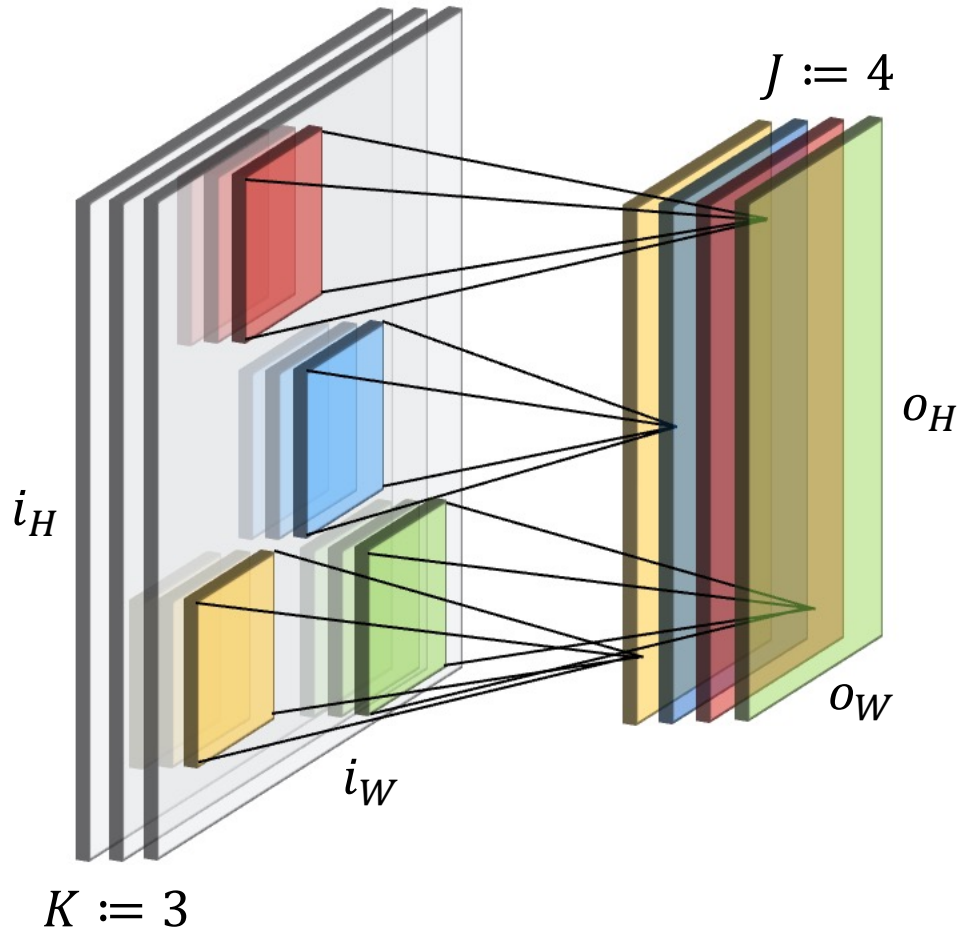


Illustration – multiple input channels, multiple feature maps

- In general the input has K channels and we learn J filters
  - A stack of filters $= s_H \times s_W \times K \times J$ tensor



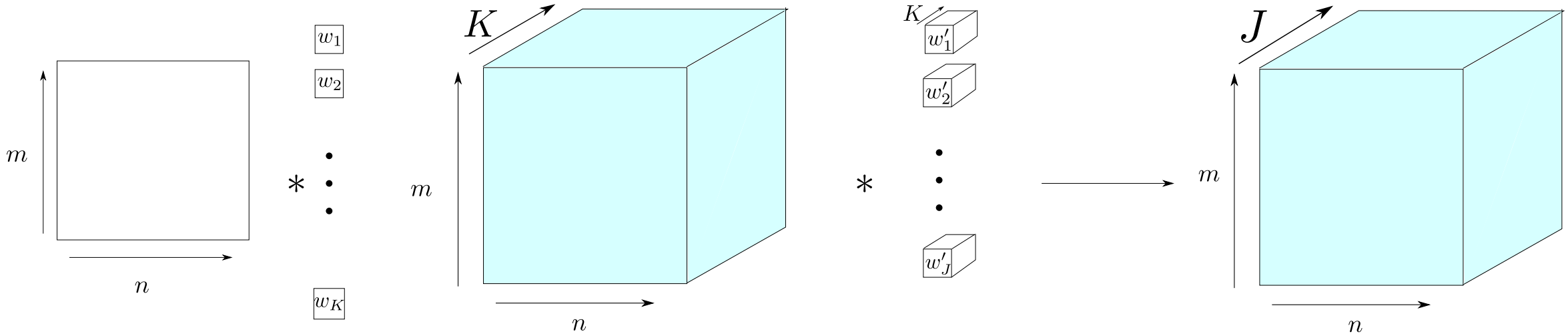
In this example, layer with

- $J := 4$ filters with
- $5 \times 5 \times 3$ kernel size

$\rightarrow 300$ weights

Convolutional Layers

- Illustration of several consecutive convolutional layers with different numbers of filters

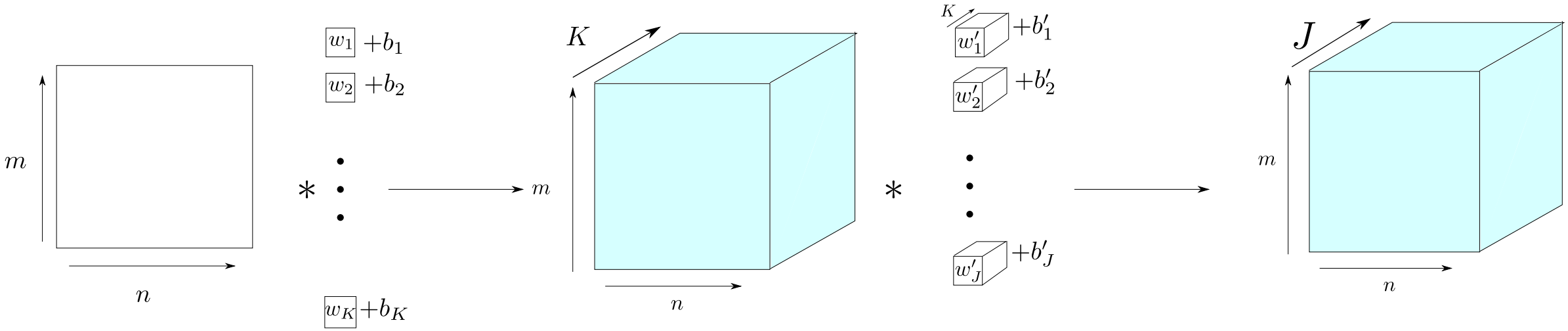


- Each layer contains “image” with a depth, where each channel corresponds to a different filter response
- Each layer is a concatenation of several features : rich information

Useful explanation : https://towardsdatascience.com/a-comprehensive-introduction-to-different-types-of-convolutions-in-deep-learning-669281e58215

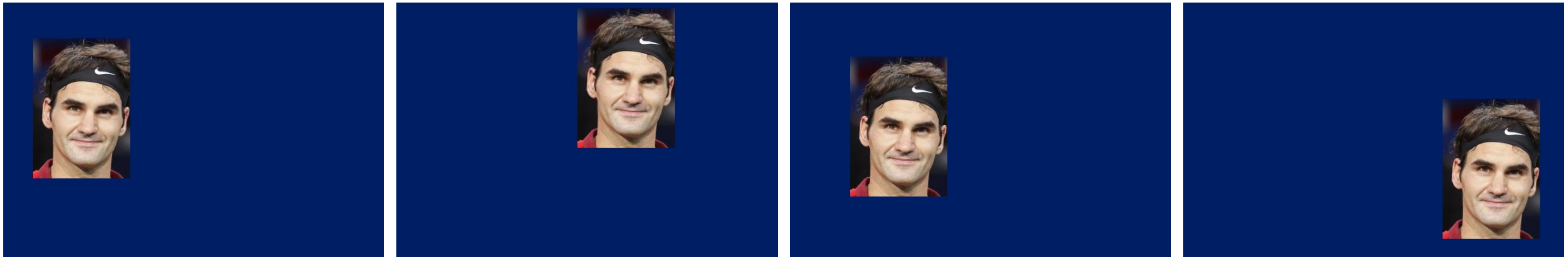
Convolutional Layers – a note on Biases

- A note on biases in neural networks : **each output layer is associated with one bias**
- There is **not** one bias per pixel
- This is coherent with the idea of **weight sharing** (bias sharing)



CNNs

- In many cases, we are primarily interested in **detection**;
- We would like to detect objects **wherever they are in the image**



- Formally, we would like to have some **shift invariance property**;
- This is done in CNNs by using **subsampling**, or some variant :
  - Strided convolutions
  - Max pooling

- We explain these now

Down-sampling and the receptive field

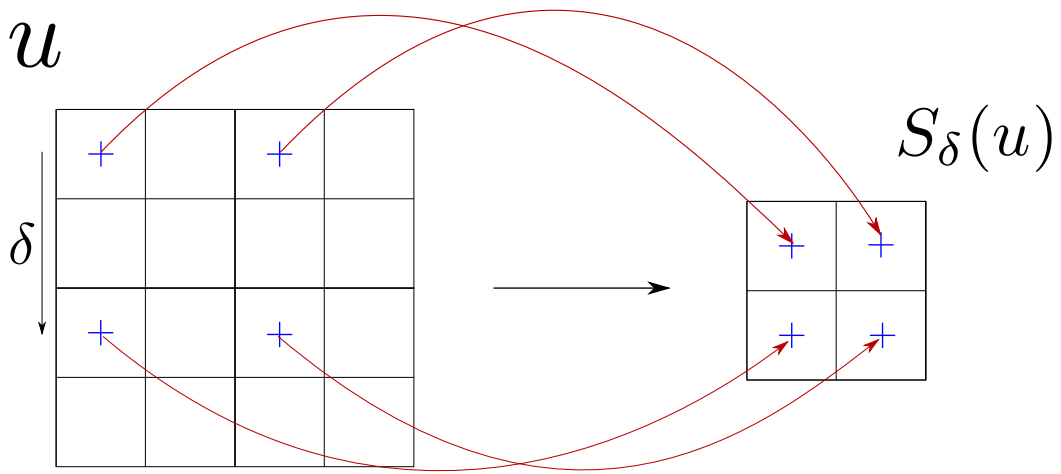
Strided Convolution

- **Strided convolution** is simply convolution, followed by **subsampling**

Subsampling operator (for 1D case)

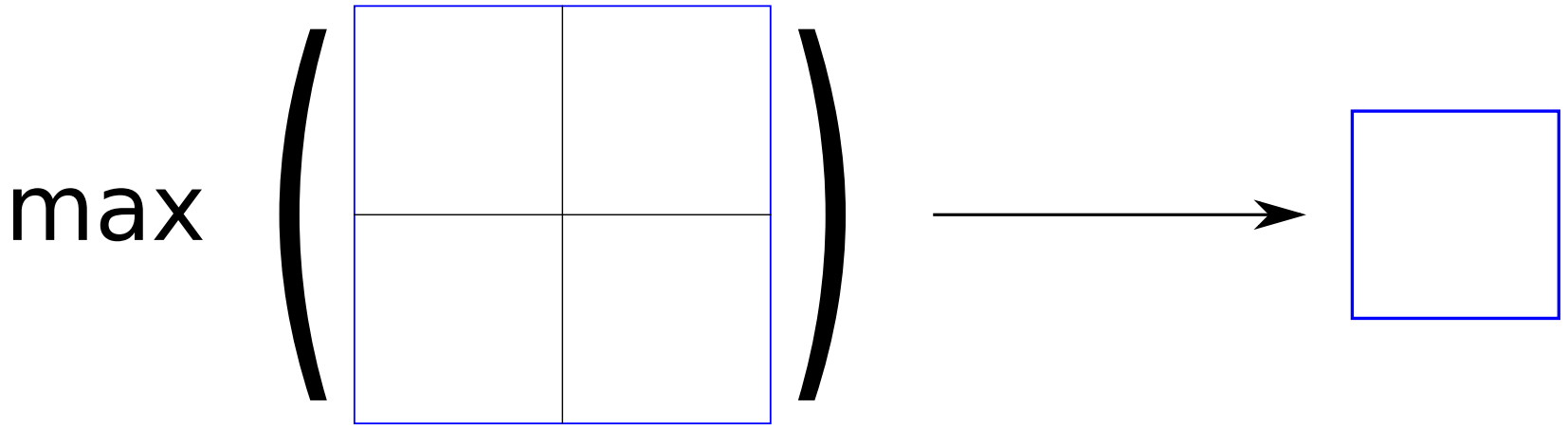
Let $x \in \mathbb{R}^n$. We define the subsampling step as $\delta > 1$, and the subsampling operator $S\delta : \mathbb{R}^n \to \mathbb{R}^{\frac{n}{\delta}}$, applied to x, as

$$S_\delta(x)\ (t) = x(\delta t), \quad \text{for } t = 0 \ldots \frac{n}{\delta} - 1$$



Max pooling

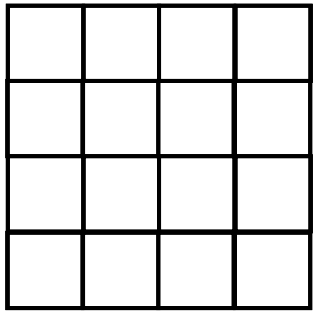
- **Max pooling** subsampling consists in taking the **maximum** value over a certain region
- This maximum value is the new subsampled value
- We will indicate the max pooling operator with S_m



Max pooling

Example:

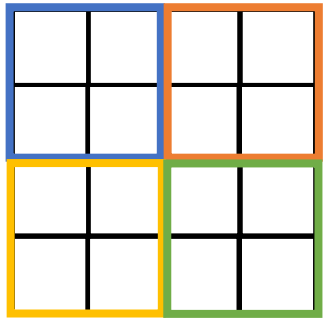
- Kernel size: 2×2
- Stride: 2



Max pooling

Example:

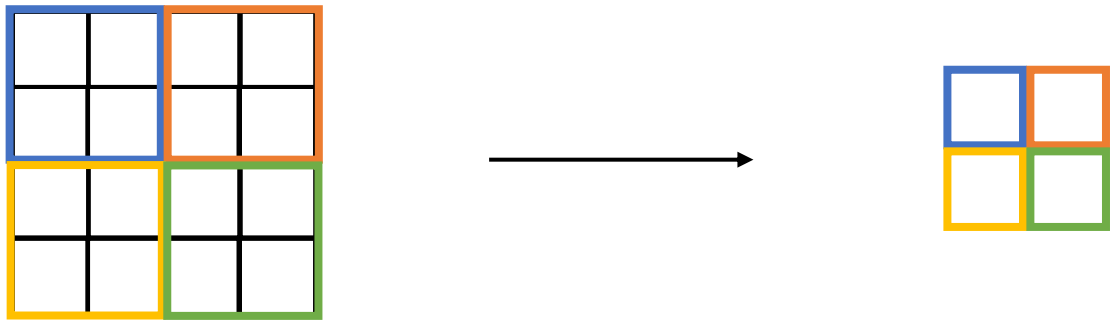
- Kernel size: 2×2
- Stride: 2



Max pooling

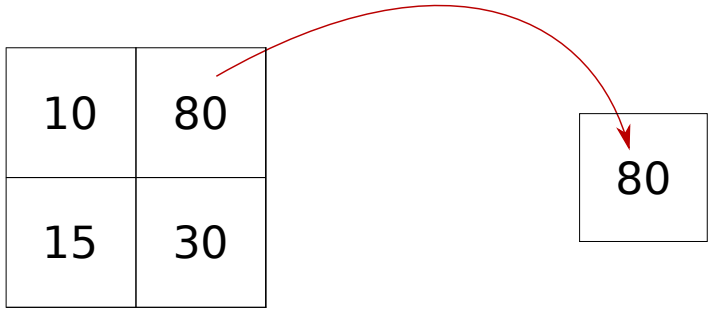
Example:

- Kernel size: 2×2
- Stride: 2

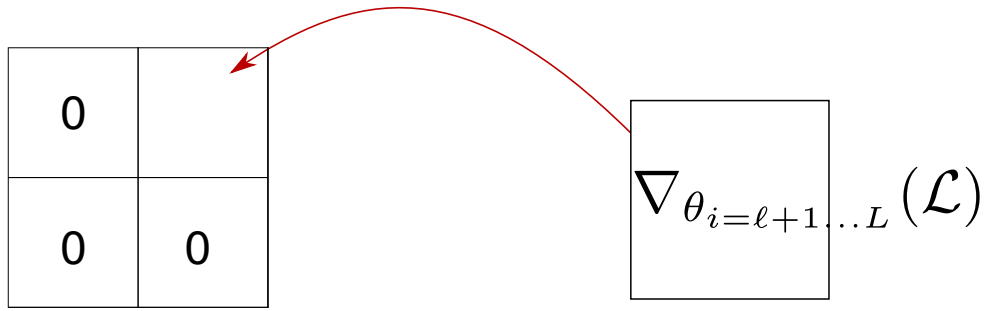


Max pooling

- Back-propagation of max pooling only passes the gradient through the **maximum**



Max pooling



Back propagation

Down-sampling

- Conclusion: cascade of convolution, non-linearities and subsampling produces **shift invariant** classification/detection

- We can detect Roger wherever he is in the image !

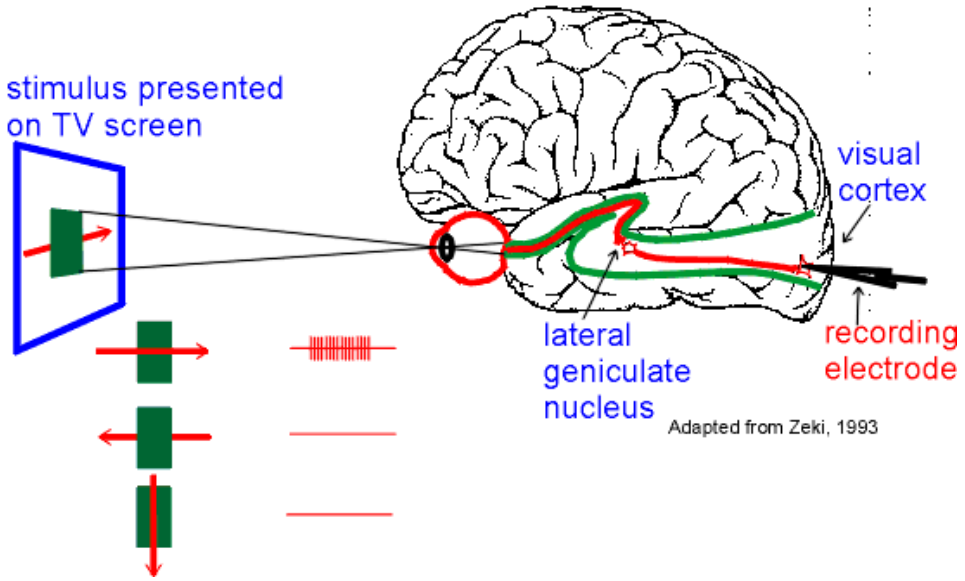
$$u * w$$

Convolution + non-linearity +max pooling

✓ ✓ ✓ ✓

The Receptive Field

- Neural networks were initially inspired by the brain's functioning
- Hubel and Wiesel [Hubert and Wiesel, 1968] showed that the visual cortex of cats and monkeys contained cells which individually responded to different small regions of the visual field
- The region which an individual cell responds to is known as the “receptive field” of that cell



[Hubert and Wiesel, 1968] **Receptive fields and functional architecture of monkey striate cortex**, Hubel, D. H. and Wiesel, T. N., *1968. Illustration from* http://www.yorku.ca/eye/cortfld.htm

The Receptive Field

- This idea was imitated in convolutional neural networks for **convolution** and **down-sampling** operations

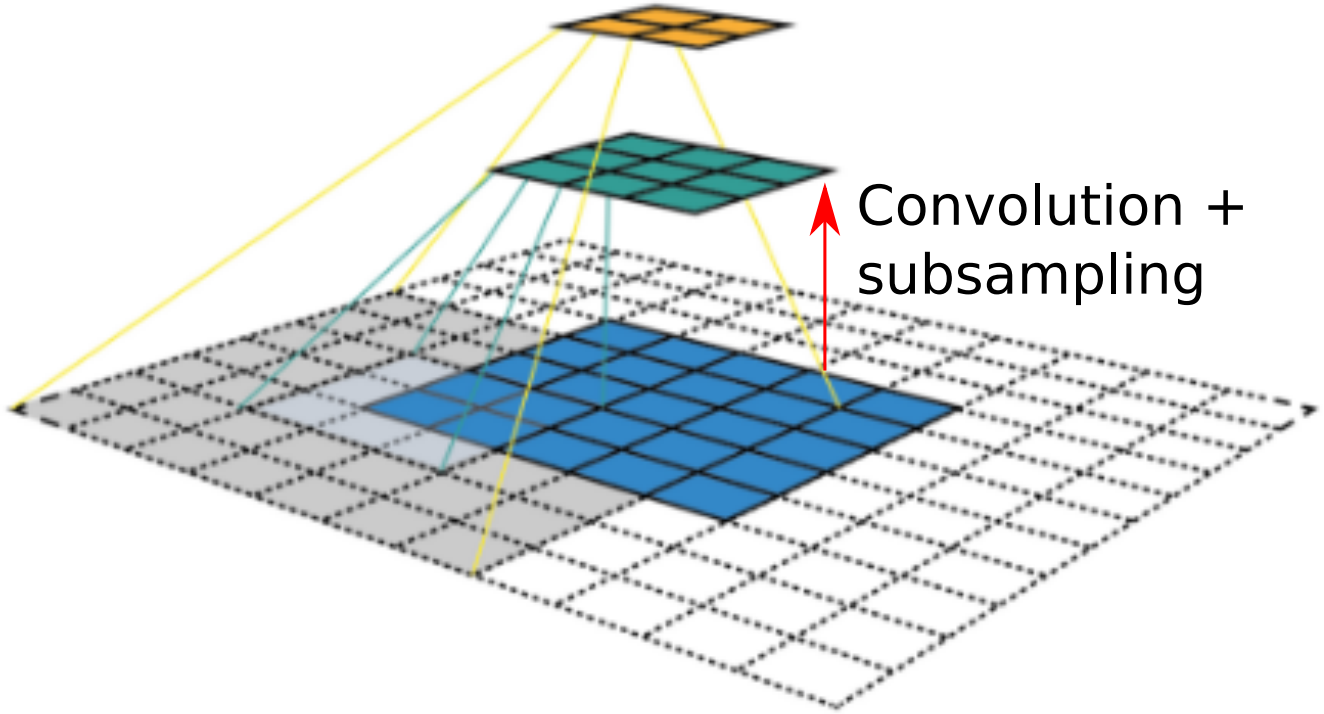
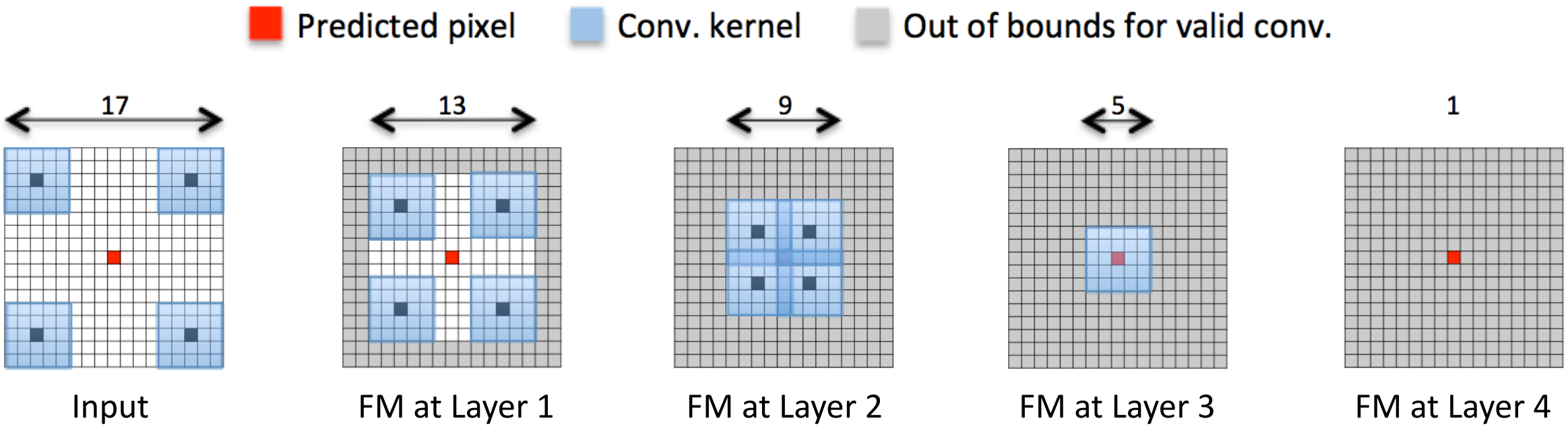


Illustration from: Applied Deep Learning, Andrei Bursuc, https://www.di.ens.fr/~lelarge/dldiy/slides/lecture_7/

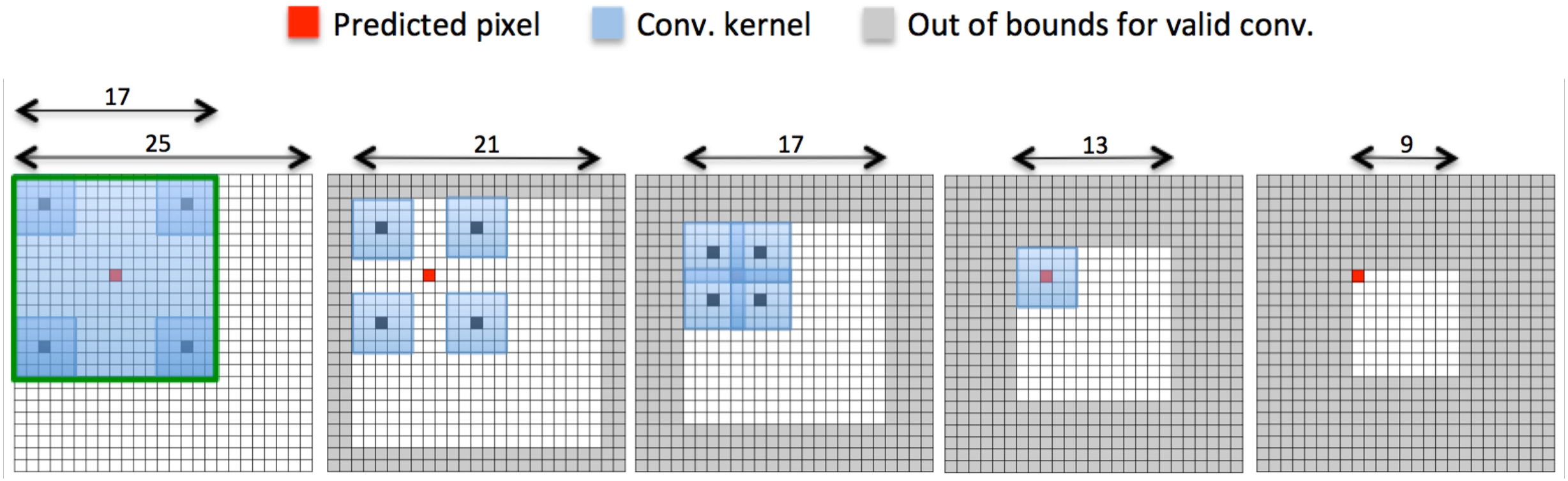
Receptive field: example of convolution layers



- The **receptive field increases with increasing depth**
  - Recursive formulae exist to compute the size of the receptive field, by backtracking through layers: resource here

- Feature Maps shrink in size, if no padding is used ("VALID" approach)

Receptive field: example of convolution layers



- The **size of feature maps expands with the input size**
- The **size of the receptive field remains the same**

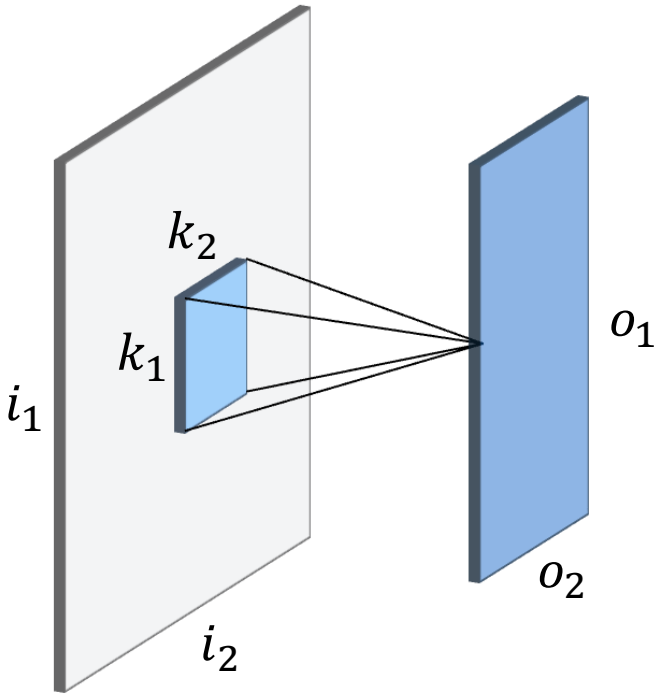
- Each prediction is **only influenced by its own receptive field** (not the whole input)

CNN details and variants

Convolutional Layer: input size → output size

- What is the **size of the output feature map** as a function of the input feature map size and convolution layer parameters?

- Assuming padding of size p_j, stride s_j along dimension j: output has size $o_j = \lfloor (i_j - k_j + 2p_j)/s_j \rfloor + 1$

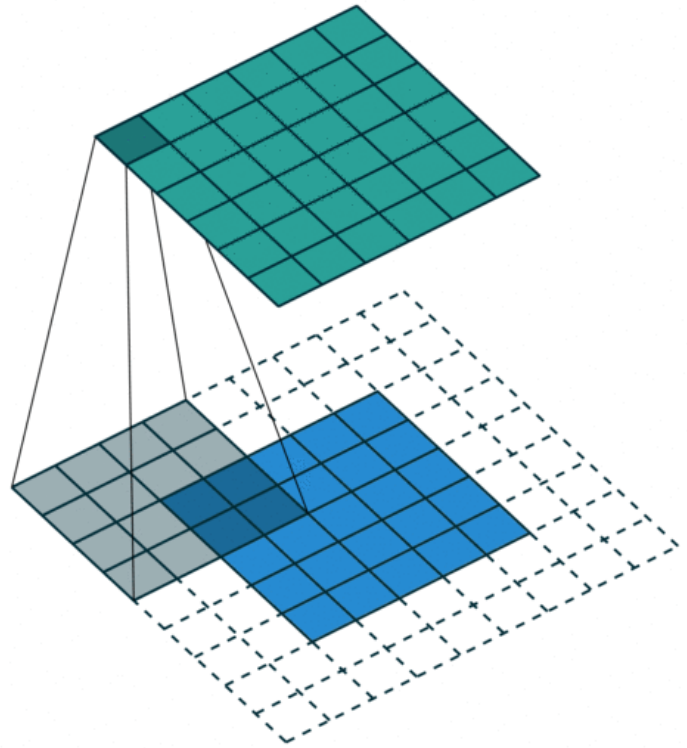


Example 1:

$i_j = 5, k_j = 4, p_j = 2, s_j = 1$

Then,

$$o_j = \left\lfloor \frac{5 - 4 + 4}{1} \right\rfloor + 1 = 6$$

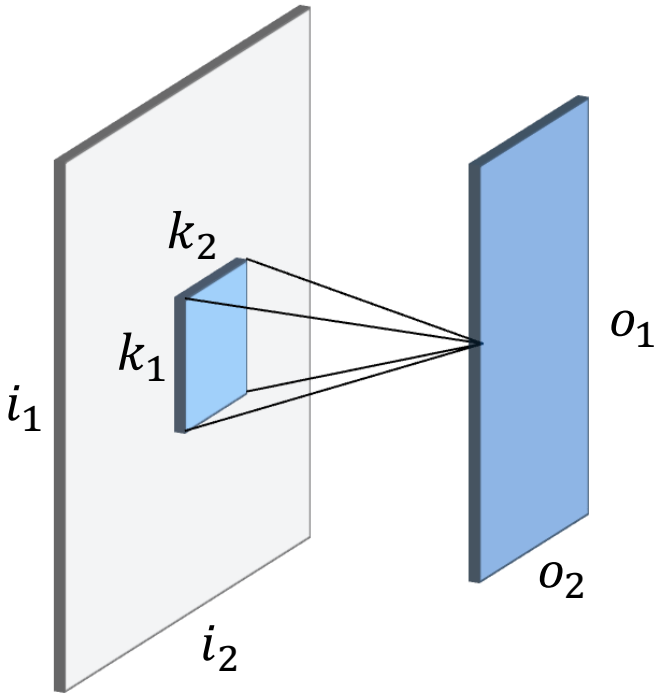


Animation: github.com/vdumoulin

Convolutional Layer: input size → output size

- What is the **size of the output feature map** as a function of the input feature map size and convolution layer parameters?

- Assuming padding of size p_j, stride s_j along dimension j: output has size $o_j = \lfloor (i_j - k_j + 2p_j)/s_j \rfloor + 1$

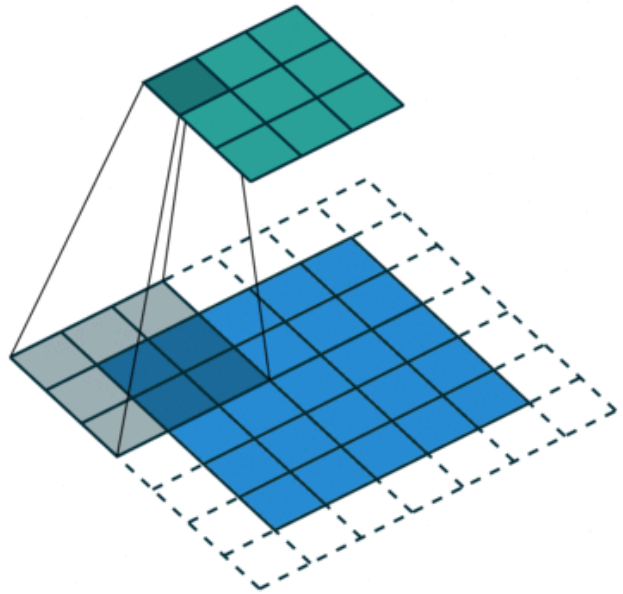


Example 2:

$i_j = 5, k_j = 3, p_j = 1, s_j = 2$

Then,

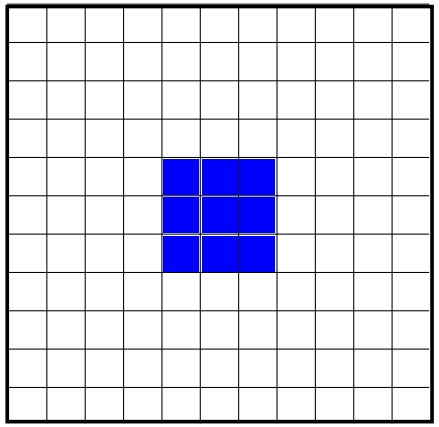
$$o_j = \left\lfloor \frac{5 - 3 + 2}{2} \right\rfloor + 1 = 3$$

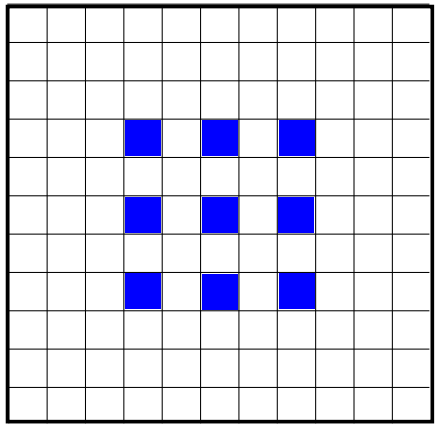


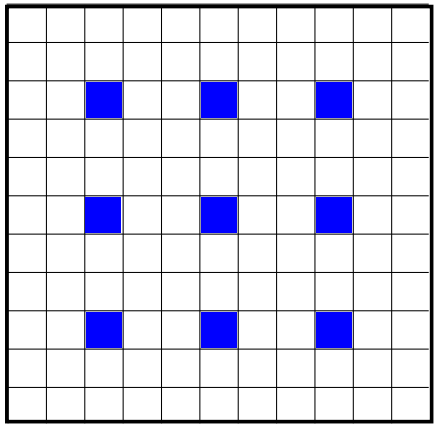
Animation: github.com/vdumoulin

Dilated Convolution

- There is a variant of convolution called *dilated convolution**
- Increase spatial extent of convolution without adding parameters
  - Space each point involved in the convolution by D


D = 1


D = 2

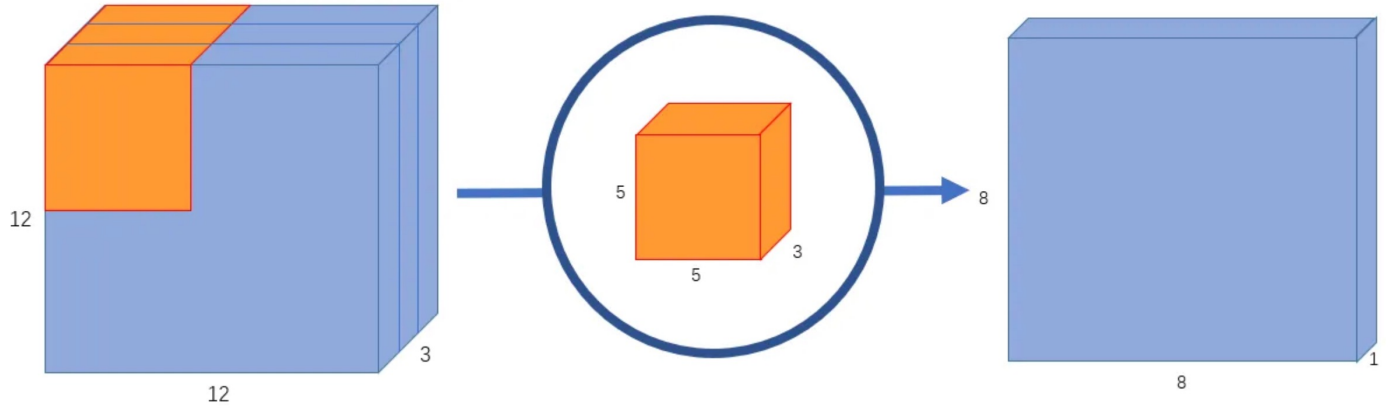

D = 3

$$(u * w)(y, x) = \sum_{i,j,k} u(y - Di, x - Dj, k)w(i, j, k)$$

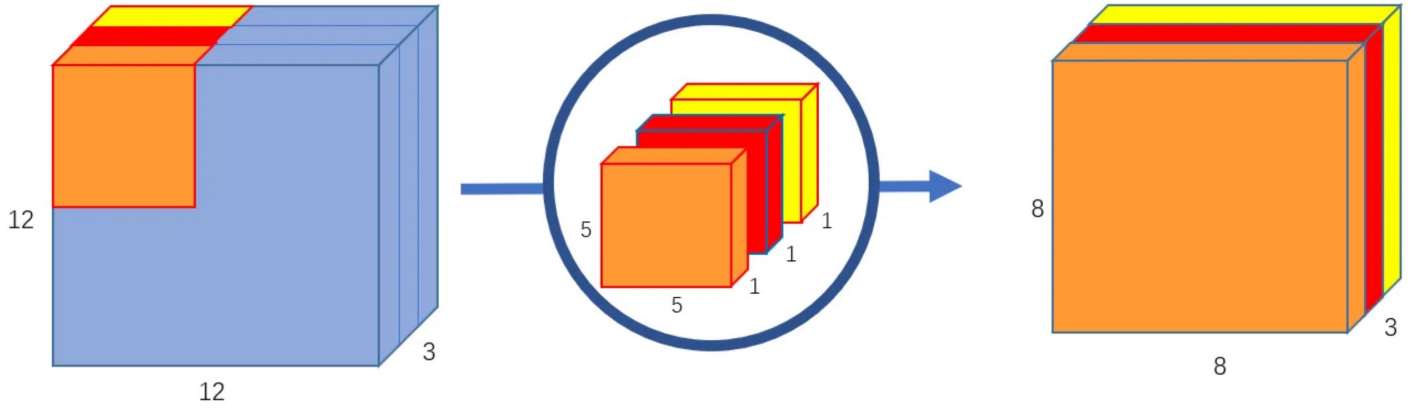
***Multi-Scale Context Aggregation by Dilated Convolution**, Yu, F, Kolten, V, *ICLR 2016*

Depthwise convolution

- Standard convolutions freely mix depth information (different channels) to generate each element in the output



- There is a variant called **depthwise convolution*** which **keeps each channel separate**
  - The number of output channels is the same as the number of input channels



Illustrations from: https://towardsdatascience.com/a-basic-introduction-to-separable-convolutions-b99ec3102728

*Xception: Deep learning with depthwise separable convolutions, Chollet, F, *CVPR 2017*

Pointwise convolution

- There is a variant of convolution called *pointwise convolution*, which **does not mix information from neighboring pixels**
  - Also called 1×1 convolution [Lin, 2013]

- It is equivalent to a linear transformation of the pixel multichannel data vector, with weights shared across all pixels

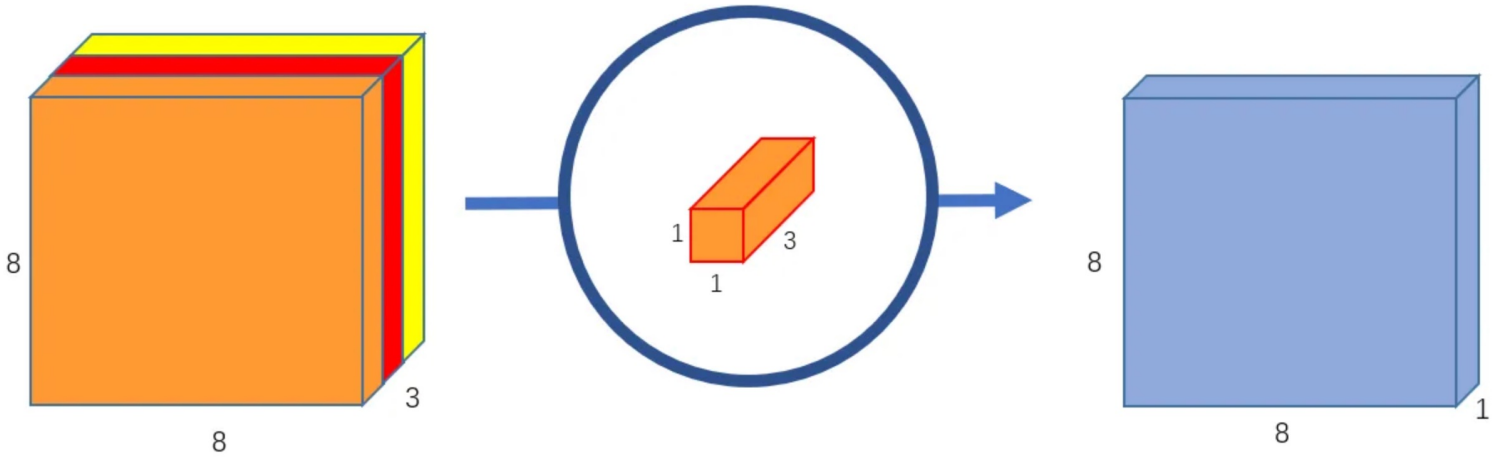


Illustration from: https://towardsdatascience.com/a-basic-introduction-to-separable-convolutions-b99ec3102728

[Lin, 2013] **Network in network**, Lin, M, et al, *arXiv 2013*

Depthwise separable convolution

- The *depthwise separable convolution** combines a depthwise convolution and a pointwise convolution

- This leads to a drastic **reduction in number of parameters** compared to the corresponding standard convolution, at little cost in expressivity

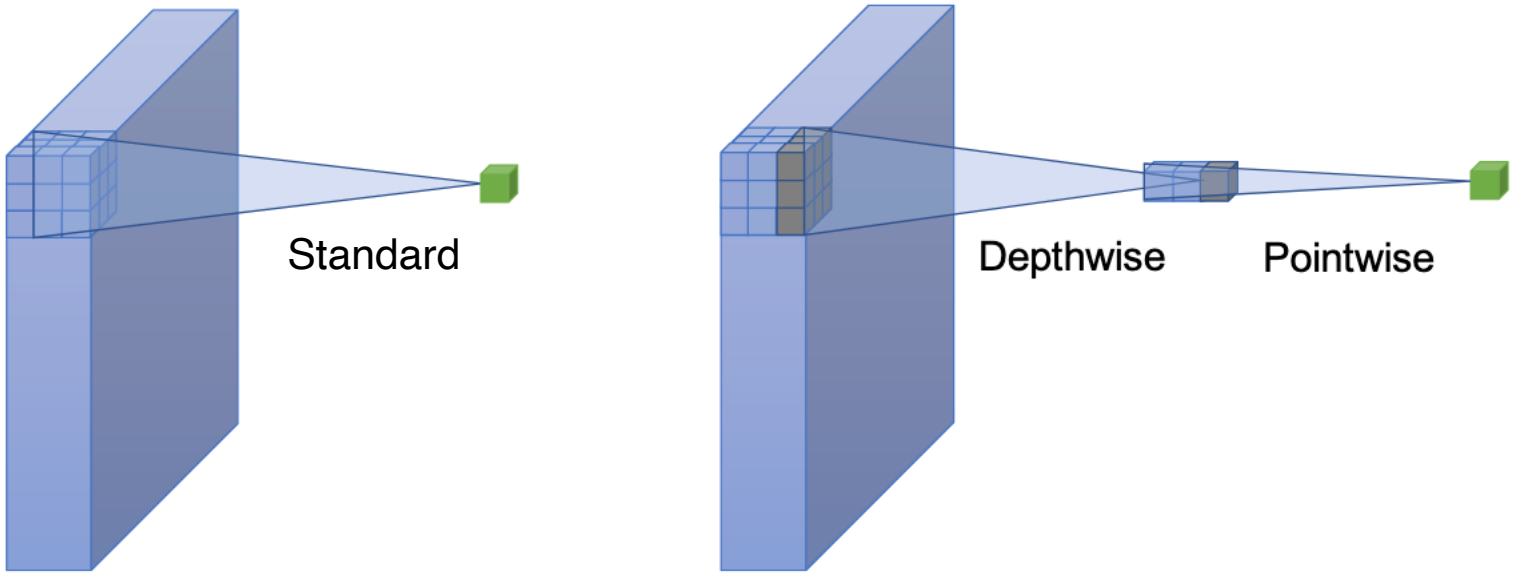


Illustration from: **Depthwise Convolution is All You Need for Learning Multiple Visual Domains**, Guo, Y, et al. *AAAI, 2019*

***Xception: Deep learning with depthwise separable convolutions**, Chollet, F, *CVPR 2017*

Depthwise separable vs. standard convolution - parameters

- Example (convolution only, ignoring the biases):
  - 64 input channels
  - 5×5 kernels
  - 128 output channels

- Standard convolution: $(64\times5\times5)\times128 = 204800$ parameters

- Depthwise separable convolution:
  $(5\times5)\times64$ (depthwise conv.) $+(64\times1\times1)\times128$ (pointwise conv.) $= 9792$ parameters!

CNNs in practice

How to build your CNN?

- We have looked at the following operations : convolutions, additive biases, non-linearities, max-pooling

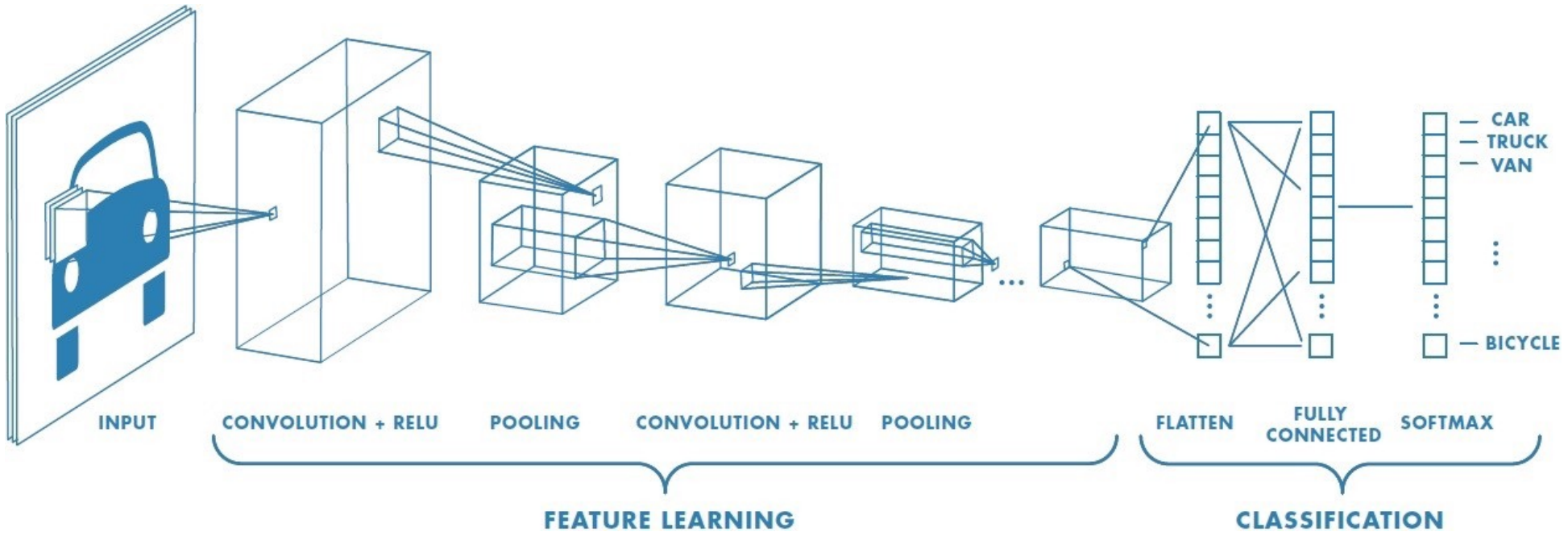
- All of these elements make up convolutional neural networks

- However, how do we put these together to create our own CNN ?
  - **Architecture**?
  - Programming tools?
  - Datasets?

Architecture: vanilla CNN for image classification

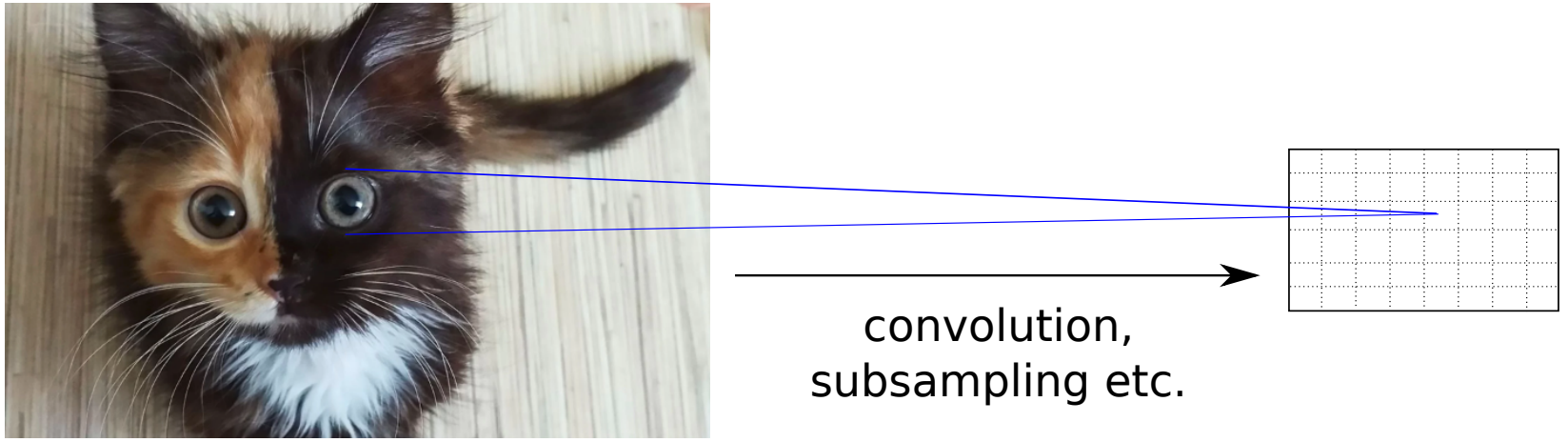
- Simple classification CNN architecture often consists of a **feature learning section**
  - Convolution + biases → non-linearities → downsampling
  - This continues until a fixed subsampling is achieved

- After this, a **classification section** is used
  - Fully connected layer → non-linearity



Architecture

- Central question: how to choose the number of layers?
- Complicated, very little theoretical understanding, a topic of research
- However, there are a few rules of thumb to follow
  - Receptive field of the **deepest layer should encompass what we consider to be a fundamental brick** of the objects we are analysing



- Set number of layers and subsampling factors according to the problem

Deeper models

- With increasing depth, features get **increasingly abstract**

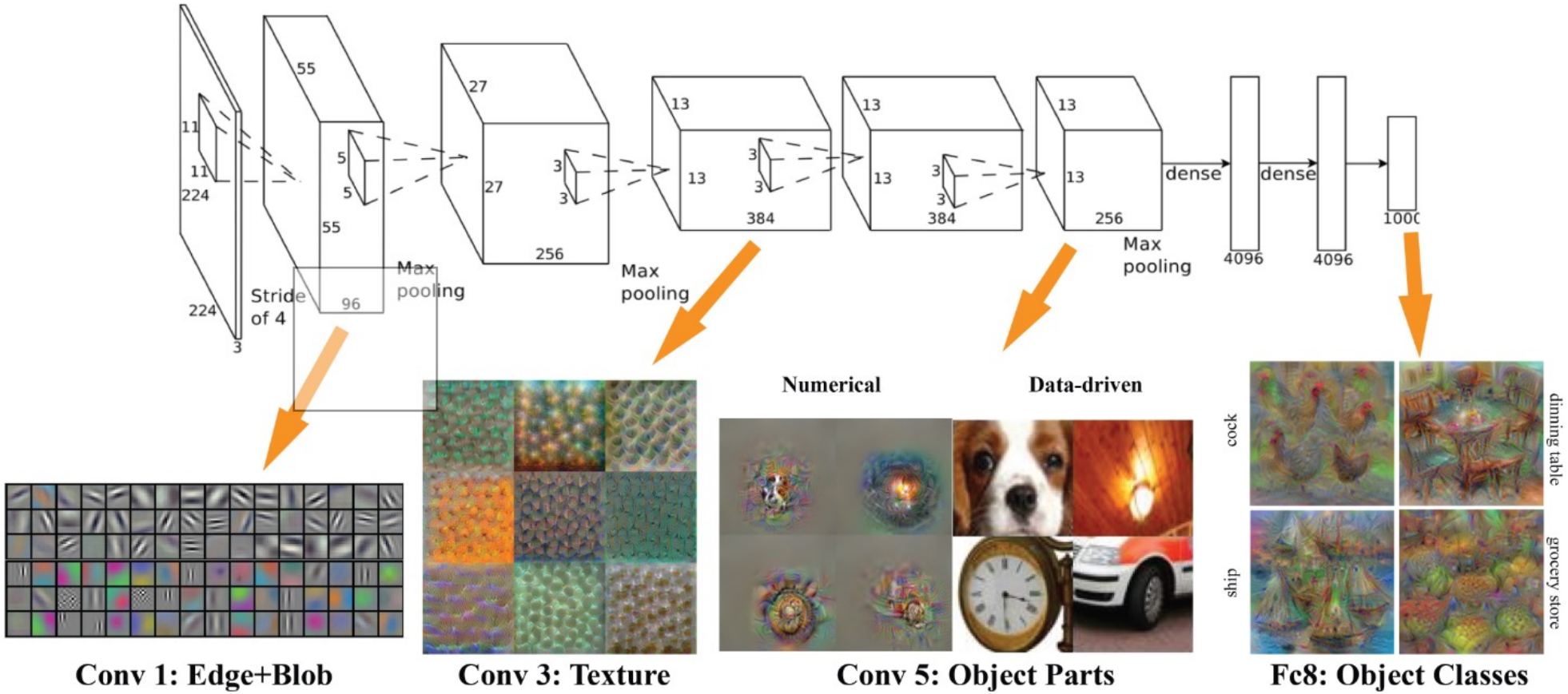


Illustration from: http://vision03.csail.mit.edu/cnn_art/

CNN programming frameworks

- Tensorflow
  - Open source, developed by Google
  - Implements a wide range of deep learning functionalities, widely used

- PyTorch
  - Open source, developed by Facebook/Meta
  - Implements a wide range of deep learning functionalities, widely used

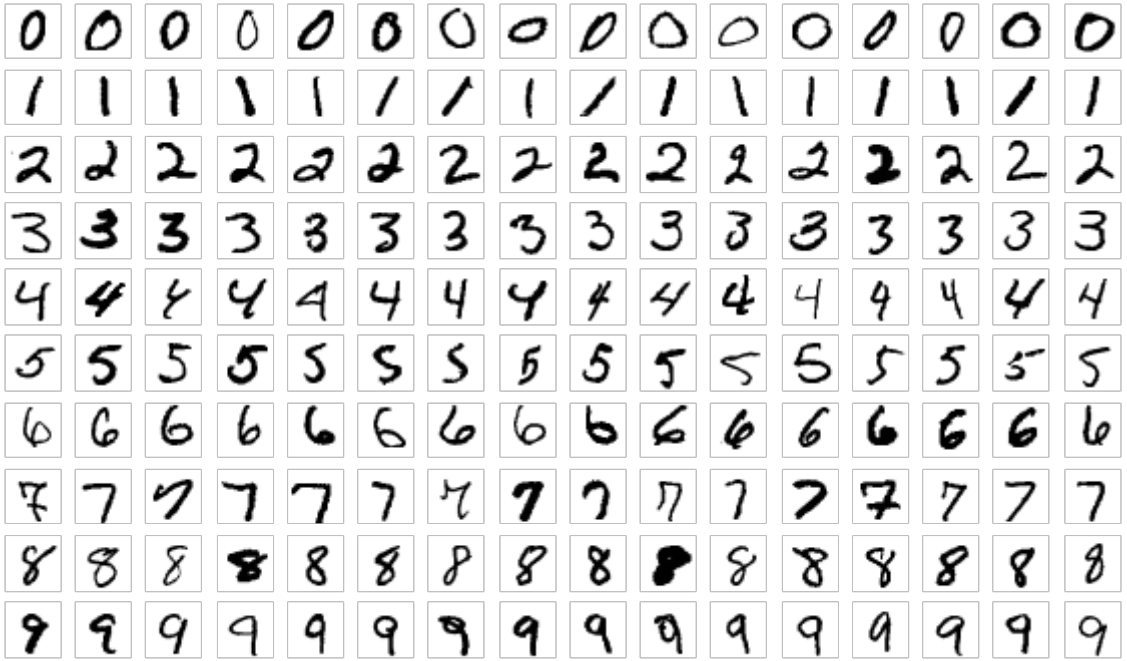
- JAX + Flax
  - Open source, developed by Google
  - Implements a wide range of deep learning functionalities, growing in popularity

- Keras
  - Developed from project ONEIROS, original author François Chollet
  - A high-level API for deep learning, with a low barrier of entry
  - Now supports Tensorflow, JAX and PyTorch as backend

Well-known image datasets

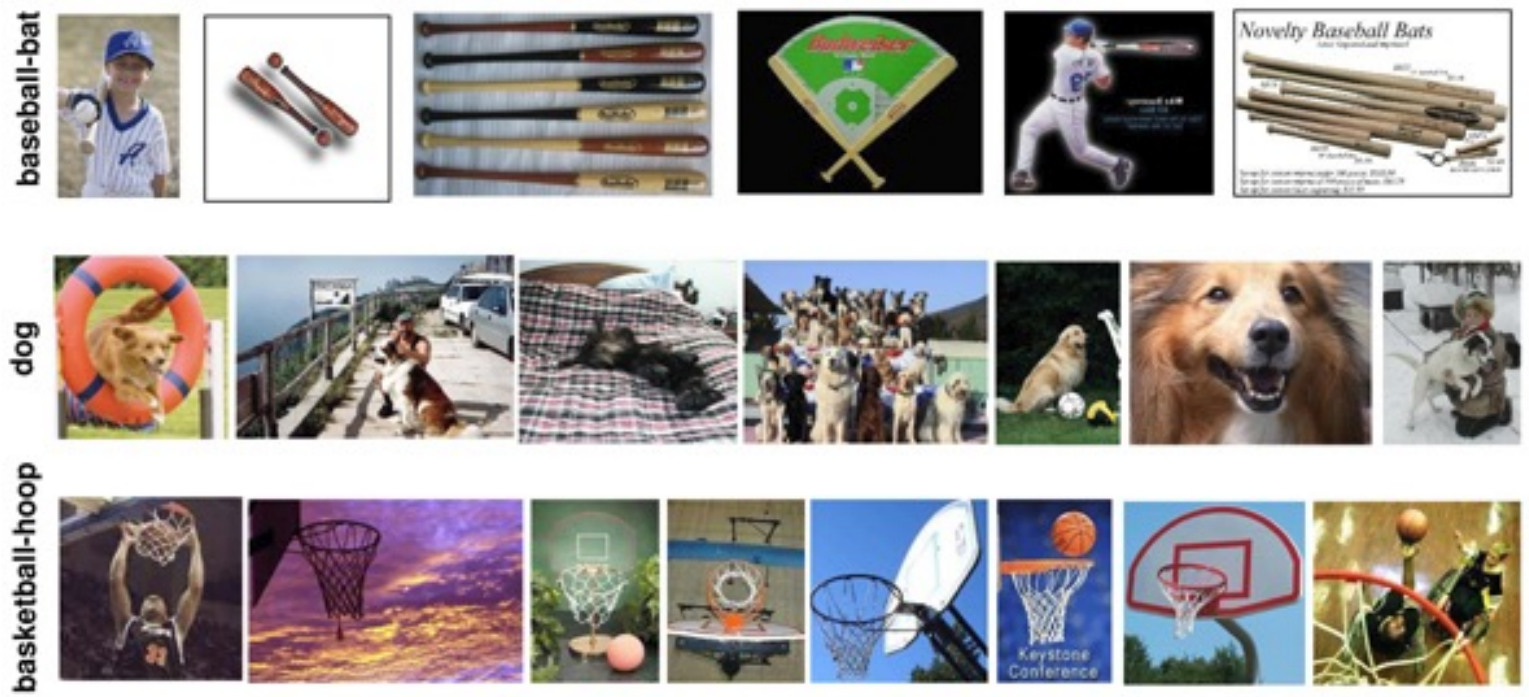
MNIST dataset

- MNIST is a dataset of 60,000 28×28 pixel grey-level images containing hand-written digits
- The digits are centered in the images and scaled to have roughly the same size
- A “simple” dataset, used for years to display the performance of CNNs, but not as much recently



Caltech 101

- Produced in 2003, first major object recognition dataset
- 9,146 images, 101 object categories, each category contains between 40 and 800 images
- Annotations exist for each image : bounding box for the object and a human-drawn outline



ImageNet dataset

- Dataset created in 2009 by researchers from Princeton university

- Very large dataset, hand-annotated
  - *ImageNet-1K*: 1,281,167 training images (50,000 validation images and 100,000 test images), 1000 mutually exclusive classes
  - *ImageNet-21K*: 14,197,122 images, divided into 21,841 (non mutually exclusive) classes

- ImageNet-1K used for the ImageNet Large Scale Visual Recognition Challenge (ILSVRC), an annual benchmark competition for object recognition algorithms until 2017

Well-known CNNs for image classification

LeNet (1989/1998)

- Created by Yann LeCun in 1989, goal : to recognise handwritten digits
- Able to classify digits with 98.9% accuracy, used by U.S. government to automatically read digits

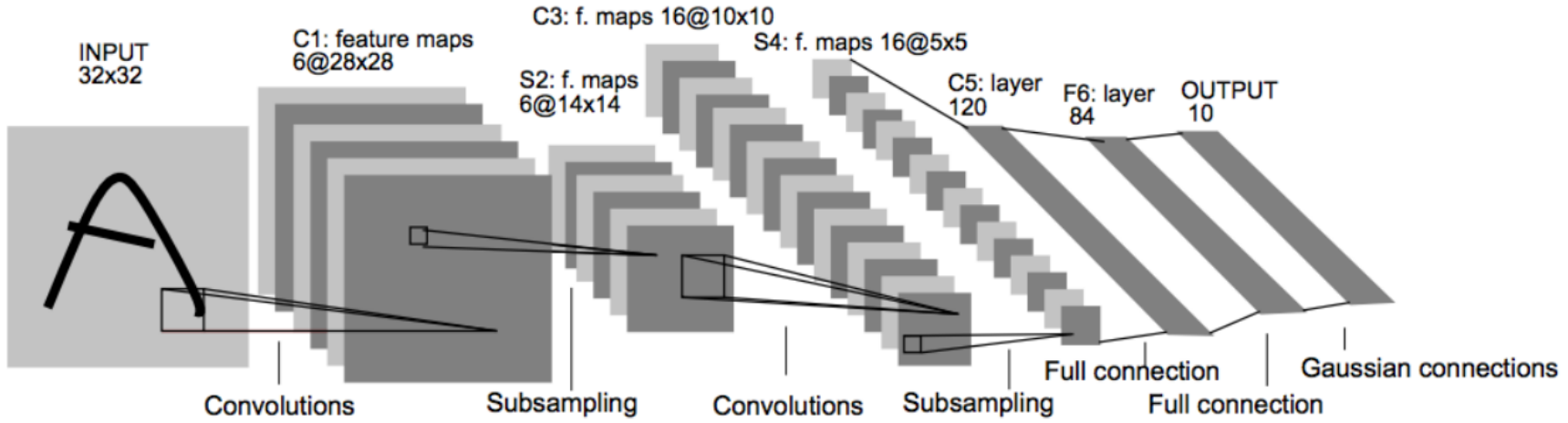


Fig. 2. Architecture of LeNet-5, a Convolutional Neural Network, here for digits recognition. Each plane is a feature map, i.e. a set of units whose weights are constrained to be identical.

- Used tanh activations, and average pooling for subsampling, had about $60K$ parameters

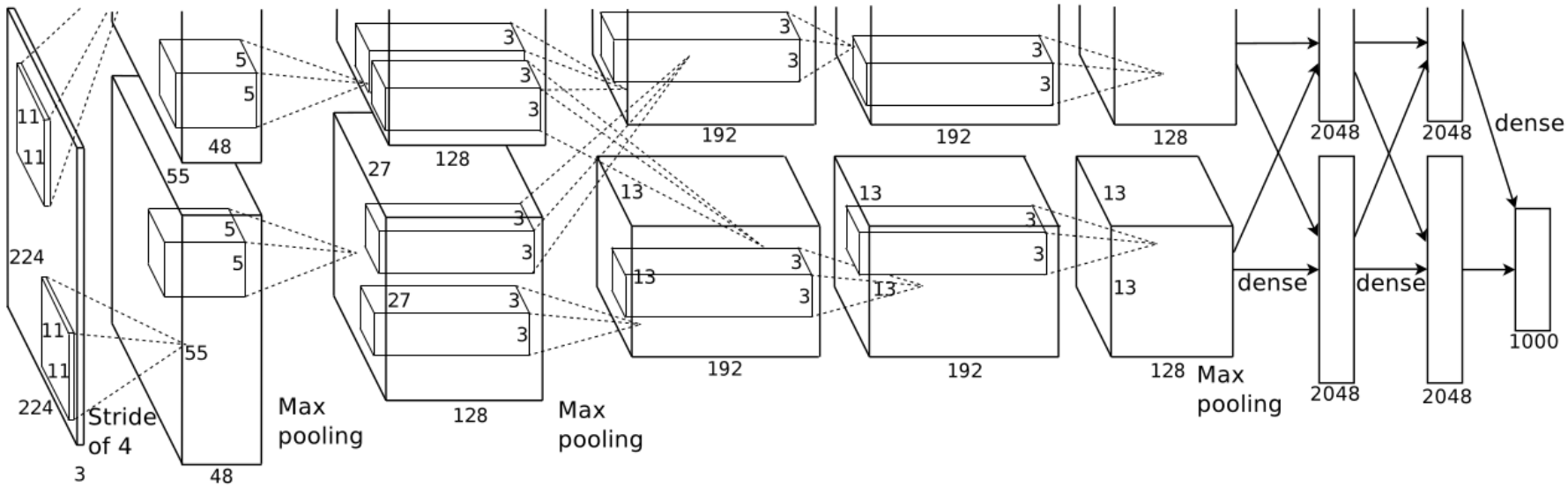
Illustration from: **Gradient-based Learning Applied to Document Recognition,** LeCun, Y. Bottou, L., Bengio, Y. and Haffner, *Proceedings of the IEEE, 1989*

AlexNet (2012)

- AlexNet : created by Alex Krizhevsky in 2012, first truly **deep neural network**

- Signaled beginning of dominance of deep learning in image processing and computer vision
- Improved accuracy of ImageNet Large Scale Visual Recognition Challenge by **10 percentage points** (16.4% top-5 error)

- Used ReLu activations, data augmentation and dropout regularization



(Krizhevsky et al., 2012)

Illustration from: **Imagenet classification with deep convolutional neural networks,** Krizhevsky, A., Sutskever, I. and Hinton, G. E, *NIPS, 2012*

AlexNet (2012)

- 96 convolutional kernels of size $11 \times 11 \times 3$ learnt by first convolutional layer on the $224 \times 224 \times 3$ input images.

AlexNet (2012)

AlexNet Network - Structural Details												
Input			Output			Layer	Stride	Pad	Kernel size		in	out	# of Param
227	227	3	55	55	96	conv1	4	0	11	11	3	96	
55	55	96	27	27	96	maxpool1	2	0	3	3	96	96	
27	27	96	27	27	256	conv2	1	2	5	5	96	256	
27	27	256	13	13	256	maxpool2	2	0	3	3	256	256	
13	13	256	13	13	384	conv3	1	1	3	3	256	384	
13	13	384	13	13	384	conv4	1	1	3	3	384	384	???
13	13	384	13	13	256	conv5	1	1	3	3	384	256	
13	13	256	6	6	256	maxpool5	2	0	3	3	256	256	
						fc6			1	1	9216	4096	
						fc7			1	1	4096	4096	
						fc8			1	1	4096	1000	
Total													62,378,344

Illustration adapted from: https://towardsdatascience.com/the-w3h-of-alexnet-vggnet-resnet-and-inception-7baaaecccc96

AlexNet (2012)

AlexNet Network - Structural Details												
Input			Output			Layer	Stride	Pad	Kernel size		in	out	# of Param
227	227	3	55	55	96	conv1	4	0	11	11	3	96	34944
55	55	96	27	27	96	maxpool1	2	0	3	3	96	96	0
27	27	96	27	27	256	conv2	1	2	5	5	96	256	614656
27	27	256	13	13	256	maxpool2	2	0	3	3	256	256	0
13	13	256	13	13	384	conv3	1	1	3	3	256	384	885120
13	13	384	13	13	384	conv4	1	1	3	3	384	384	1327488
13	13	384	13	13	256	conv5	1	1	3	3	384	256	884992
13	13	256	6	6	256	maxpool5	2	0	3	3	256	256	0
						fc6			1	1	9216	4096	37752832
						fc7			1	1	4096	4096	16781312
						fc8			1	1	4096	1000	4097000
Total													**62,378,344**

Illustration adapted from: https://towardsdatascience.com/the-w3h-of-alexnet-vggnet-resnet-and-inception-7baaaecccc96

VGG16 (2015)

- VGG16 is a 16-layer network
- All convolutional kernels are 3×3
  - Receptive field identical after two 3×3 convolutions as after one 5×5, but uses fewer parameters (18 vs. 25)

- Around 7.5% test error (top-5) on ILSVRC, runner-up of ILSVRC in 2014

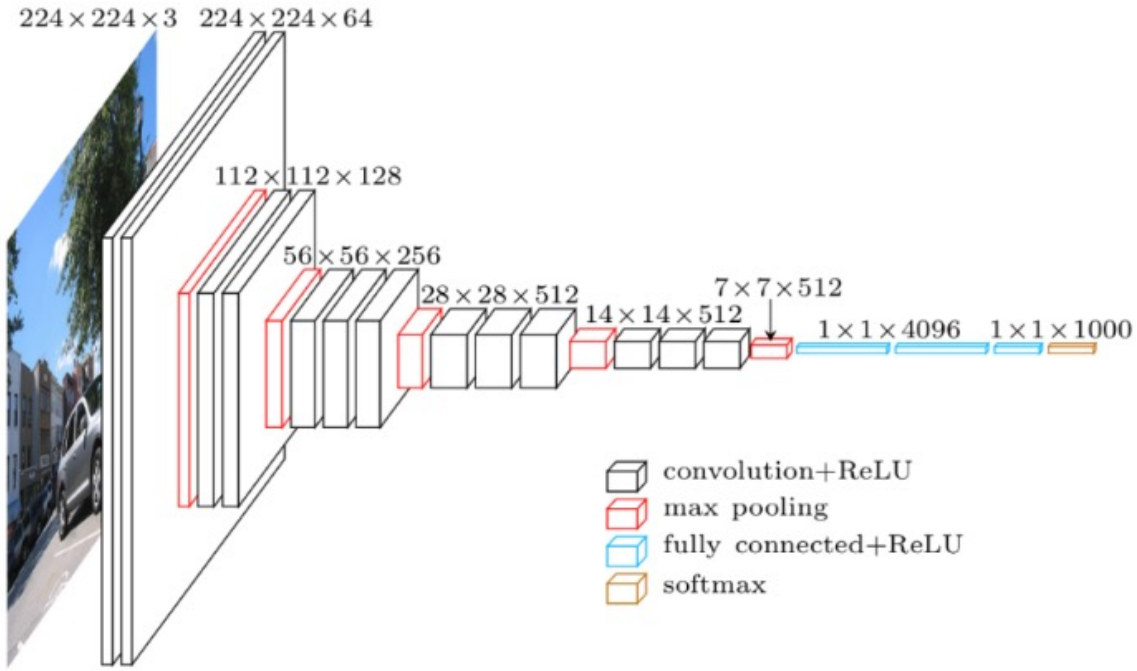


Illustration from: Mathieu Cord, https://blog.heuritech.com/2016/02/29/a-brief-report-of-the-heuritech-deep-learning-meetup-5/

***Very Deep Convolutional Networks for Large-Scale Image Recognition**, Simonyan, K. and Zisserman, A., *ICLR, 2015*

VGG16 (2015)

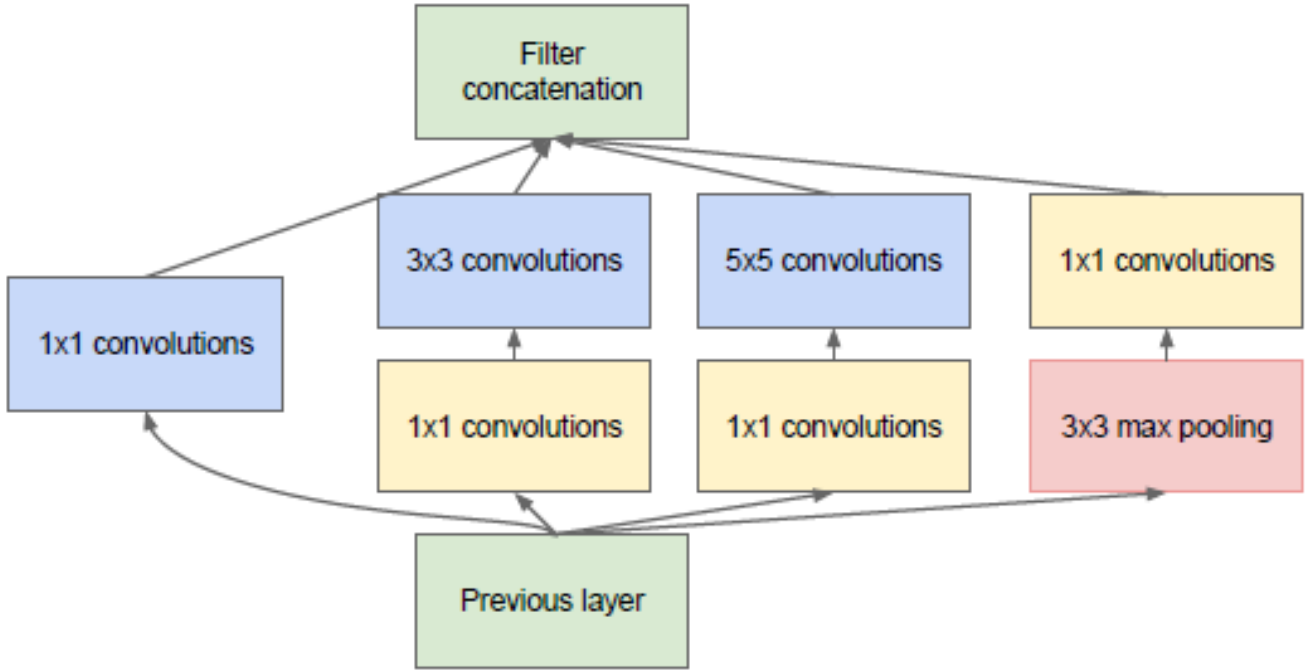
VGG16 - Structural Details													
#	Input Image			output			Layer	Stride	Kernel		in	out	Param
1	224	224	3	224	224	64	conv3-64	1	3	3	3	64	1792
2	224	224	64	224	224	64	conv3064	1	3	3	64	64	36928
	224	224	64	112	112	64	maxpool	2	2	2	64	64	0
3	112	112	64	112	112	128	conv3-128	1	3	3	64	128	73856
4	112	112	128	112	112	128	conv3-128	1	3	3	128	128	147584
	112	112	128	56	56	128	maxpool	2	2	2	128	128	65664
5	56	56	128	56	56	256	conv3-256	1	3	3	128	256	295168
6	56	56	256	56	56	256	conv3-256	1	3	3	256	256	590080
7	56	56	256	56	56	256	conv3-256	1	3	3	256	256	590080
	56	56	256	28	28	256	maxpool	2	2	2	256	256	0
8	28	28	256	28	28	512	conv3-512	1	3	3	256	512	1180160
9	28	28	512	28	28	512	conv3-512	1	3	3	512	512	2359808
10	28	28	512	28	28	512	conv3-512	1	3	3	512	512	2359808
	28	28	512	14	14	512	maxpool	2	2	2	512	512	0
11	14	14	512	14	14	512	conv3-512	1	3	3	512	512	2359808
12	14	14	512	14	14	512	conv3-512	1	3	3	512	512	2359808
13	14	14	512	14	14	512	conv3-512	1	3	3	512	512	2359808
	14	14	512	7	7	512	maxpool	2	2	2	512	512	0
14	1	1	25088	1	1	4096	fc		1	1	25088	4096	102764544
15	1	1	4096	1	1	4096	fc		1	1	4096	4096	16781312
16	1	1	4096	1	1	1000	fc		1	1	4096	1000	4097000
Total													138,423,208

GoogLeNet (2015)

- In 2014/2015, Google introduced* the “Inception” architecture/module

- Major reduction in the total number of parameters
  - $6M$ instead of $60M$ for AlexNet
  - Replace all fully-connected layers except the last by a simple global average pooling

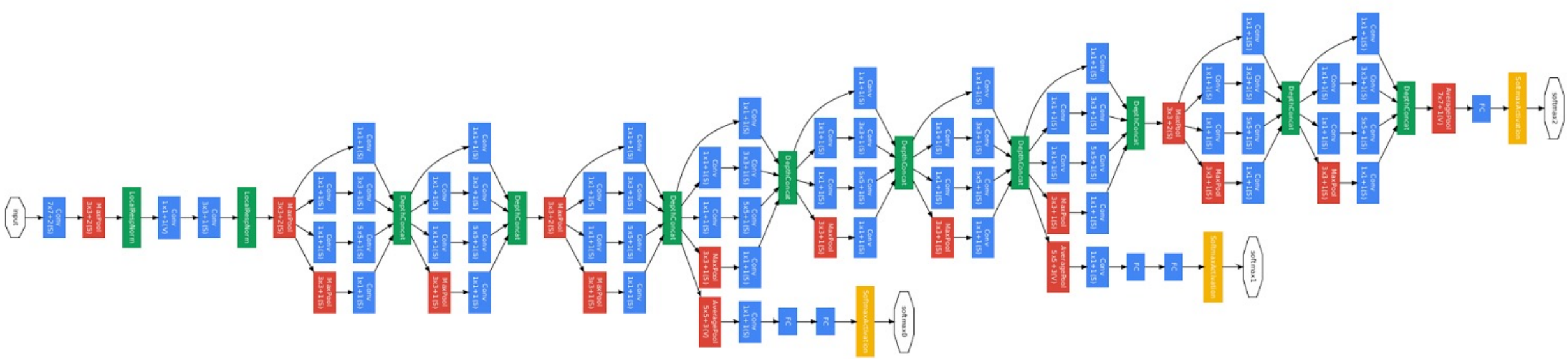
- Novel idea: have variable kernel sizes in one layer to increase representational power



*__Going deeper with convolutions__, Szegedy et al., *CVPR, 2015*

GoogLeNet (2015)

- Created by Google in 2014, GoogLeNet (a.k.a. Inception-v1) is a specific implementation of the "Inception" architecture with 22 layers
- Auxiliary/intermediate classifiers to ease training/convergence
- 6.6% top-5 test error rate on ImageNet (human error rate ~5%), winner of ILSVRC 2014



*__Going deeper with convolutions__, Szegedy et al., *CVPR, 2015*

GoogLeNet (2015)

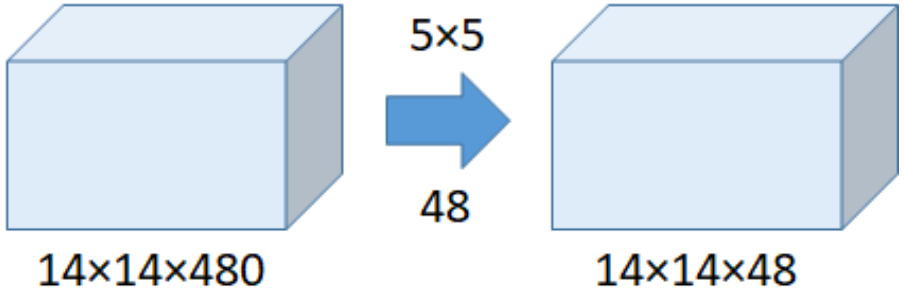
GoogLeNet - Structural Details													
Input Image			output			Layer	Input Layer	Stride	Pad	Kernel		in	out	Param
227	227	3	112	112	64	conv1	input	2	1	7	7	3	64	9472
112	112	64	56	56	64	maxpool1	conv1	2	0.5	3	3	64	64	0
56	56	64	56	56	64	conv1x1	maxpool1	1	0	1	1	64	64	4160
56	56	64	56	56	192	conv2-1		1	1	3	3	64	192	110784
56	56	192	28	28	192	maxpool2		2	0.5	3	3	192	192	0
⋮														
inception (5b)	7	7	832	7	7	192	conv1x1a	depth-concat	1	0	1	1	832	192	159936
	7	7	832	7	7	48	conv1x1b	depth-concat	1	0	1	1	832	48	39984
	7	7	832	7	7	832	maxpool-a	depth-concat	1	1	3	3	832	832	0
	7	7	832	7	7	384	conv1x1c	depth-concat	1	0	1	1	832	384	319872
	7	7	96	7	7	384	conv3-3	conv1x1a	1	1	3	3	192	384	663936
	7	7	16	7	7	128	conv5x5	conv1x1b	1	2	5	5	48	128	153728
	7	7	384	7	7	128	conv1x1d	maxpool-a	1	0	1	1	128	128	16512
				7	7	1024	depth-concat	conv1x1c, conv3x3, conv5x5, conv1x1d							
	7	7	1024	1	1	1024	avgpool	depth-concat	1	0	7	7	1024	1024	0
	1	1	1024	1	1	1000	fc	depth-concat	1	0	1	1	1024	1000	1025000
									Total						6,414,360

Illustration adapted from: https://towardsdatascience.com/the-w3h-of-alexnet-vggnet-resnet-and-inception-7baaaecccc96

GoogLeNet (2015)

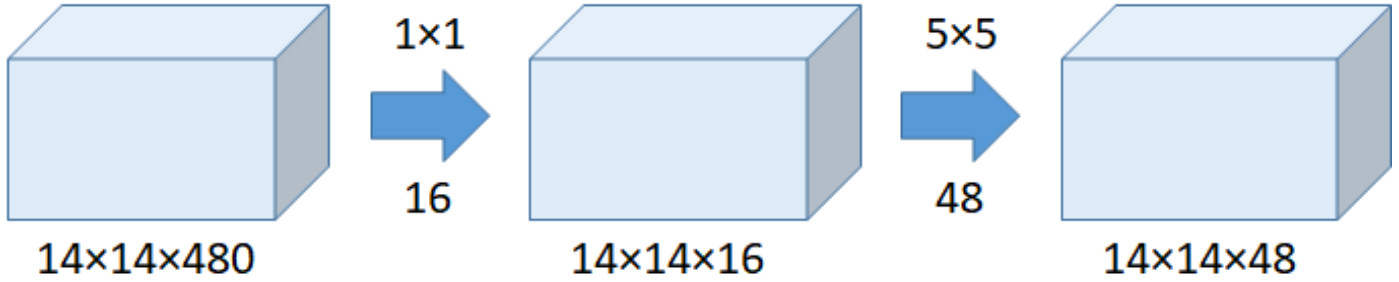
- Makes extensive use of 1×1 convolutional layers which serve as "bottleneck" layers
  - Reduces the number of parameters

- *Without* 1×1 convolutions:



Number of parameters:
$(480\times5\times5)\times48 = 576000$

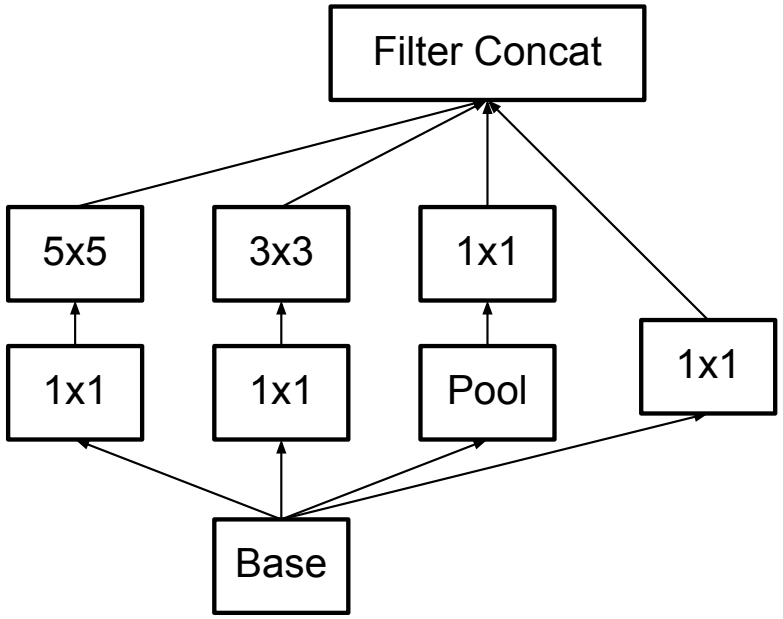
- *With* 1×1 convolutions:

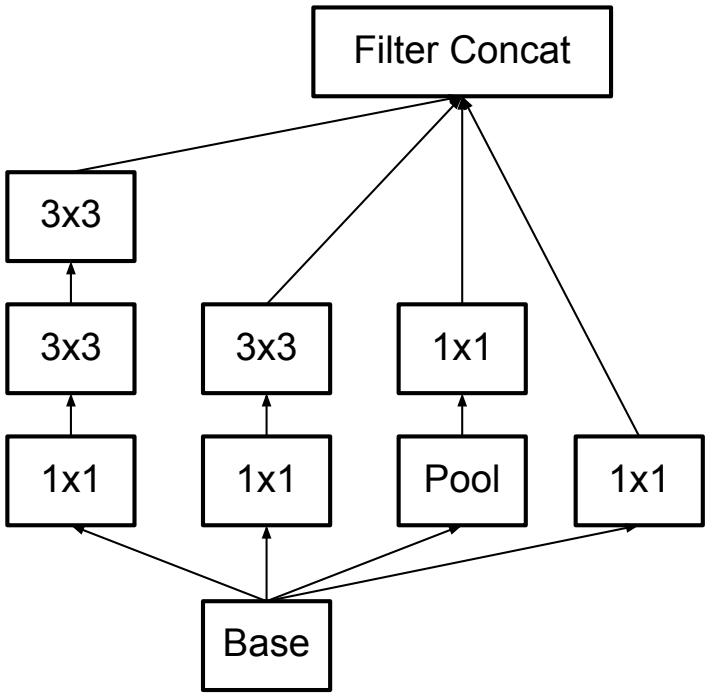


Number of parameters:
$(480\times1\times1)\times16$
$+(16\times5\times5)\times48$
$= 26880$

Inception-v2 [optional]

- Change the basic Inception module [Szegedy, 2015]
- Replace 7×7 and 5×5 filters with multiple 3×3 convolutions (as in VGG)

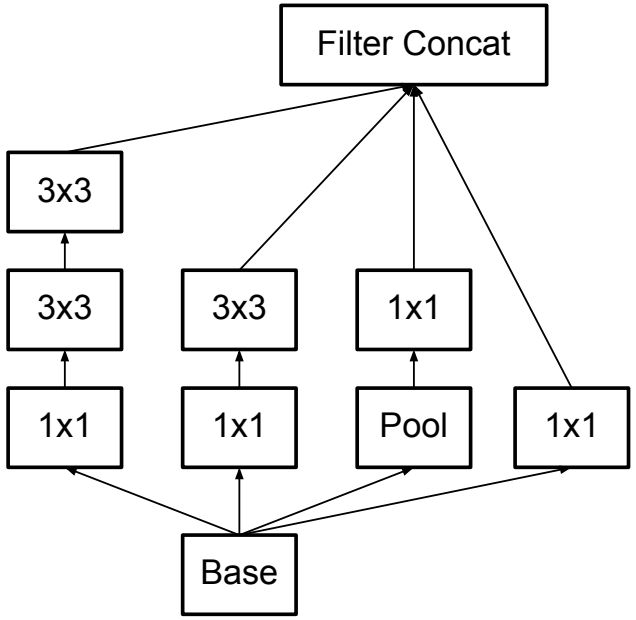




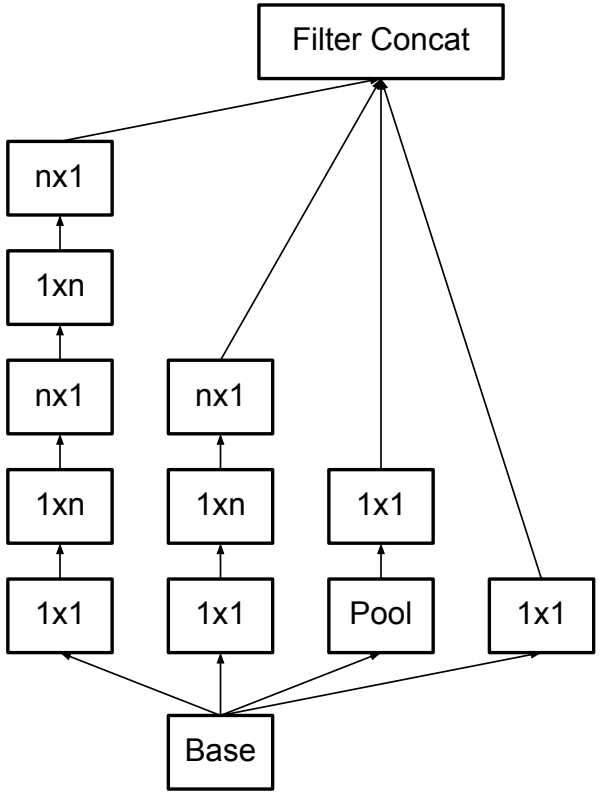
[Szegedy, 2015] **Rethinking the Inception Architecture for Computer Vision**, Szegedy et al., *arXiv, 2015*

Inception-v2 [optional]

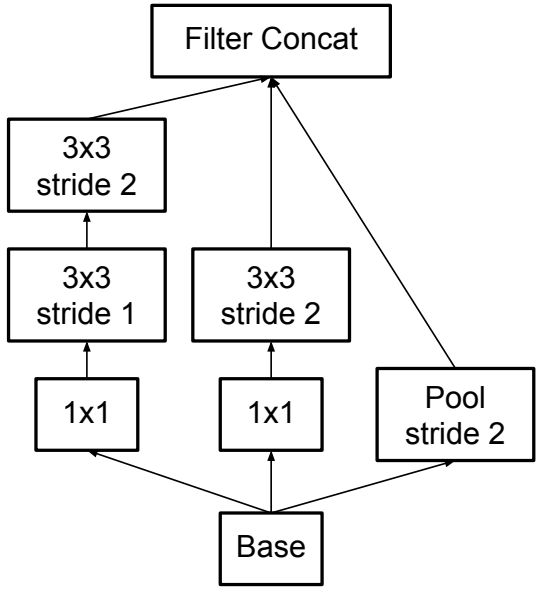
- Inception-v2 is a 48-layer network based on the modified Inception modules
  - More parameter-efficient and FLOPs-efficient



Module used early in the network



Module used deeper in the network



Module used for subsampling

Inception-v3 [optional]

- Also presented in [Szegedy, 2015]
- Same architecture as Inception-v2 but…

- RMSProp optimizer
- BatchNorm in the auxillary classifiers
- **Label Smoothing**
  - A type of regularization preventing the network to become unduly overconfident about a class
  - Replaces hard label e.g., $(0,1,0,\cdots,0)$, by "soft" label $(\frac{\epsilon}{K-1}, 1-\epsilon, \frac{\epsilon}{K-1}, \cdots, \frac{\epsilon}{K-1})$ when training

- There is also an Inception-v4 architecture [Szegedy, 2017] with a modified stem
  - The stem is the first part of the network directly processing the input, before the Inception modules (generally consisting of a few conv. and subsampling layers)

[Szegedy, 2015] **Rethinking the Inception Architecture for Computer Vision**, Szegedy et al., *arXiv, 2015*
[Szegedy, 2017] **Inception-v4, inception-resnet and the impact of residual connections on learning**, Szegedy et al., *AAAI, 2017*

Going even deeper

- Deeper models tend to have higher training and test error than shallower models. This is not just overfitting.

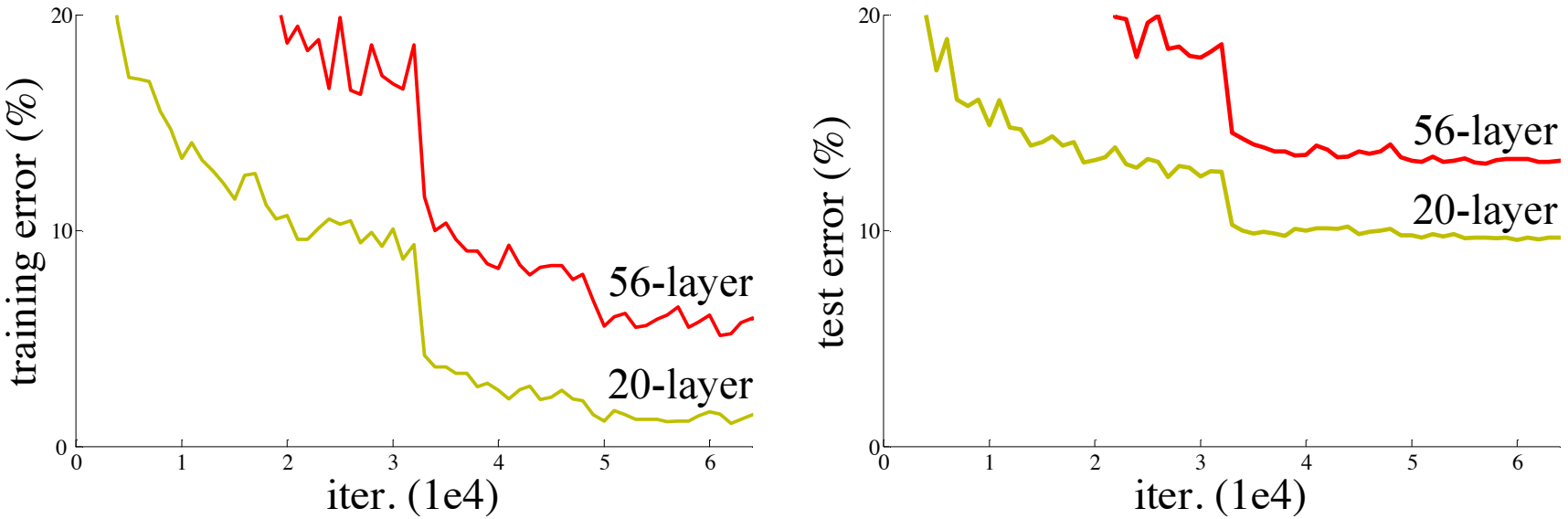


Illustration from [He, 2016]

- Possible reasons → solutions:
  - Vanishing gradients → ReLu, good initialization
  - Internal covariate shift → batch normalization
  - More difficult optimization with complex loss landscape → **ResNet**

ResNet (2016)

- ResNets [He, 2016] use **residual connections** to mitigate the **vanishing gradient** problem and ease optimization
  - Variants with 18-layers, 34-layers, 50-layers, 100-layers, 152-layers!

- Winner of ILSVRC 2015 with 3.7% top-5 test error (for an ensemble of ResNets)
- Residual mechanism used in many subsequent architectures

- Denoting the desired underlying mapping as $\mathcal{H}(x)$, stacked nonlinear layers fit:
  - $\mathcal{H}(x)$ in standard networks
  - $\mathcal{F}(x) \coloneqq \mathcal{H}(x) - x$ in ResNets; the original mapping is recast as $\mathcal{F}(x) + x$

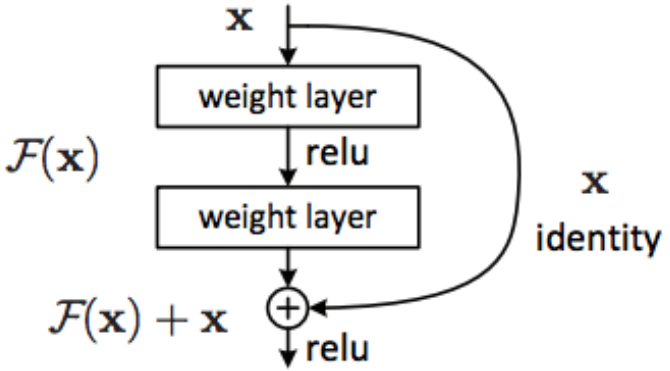


Figure 2. Residual learning: a building block.

- Recently, [Kolesnikov, 2020] obtained 87.54% top-1 accuracy with a ResNet-152x4 with weights transferred from the larger JFT-300M dataset

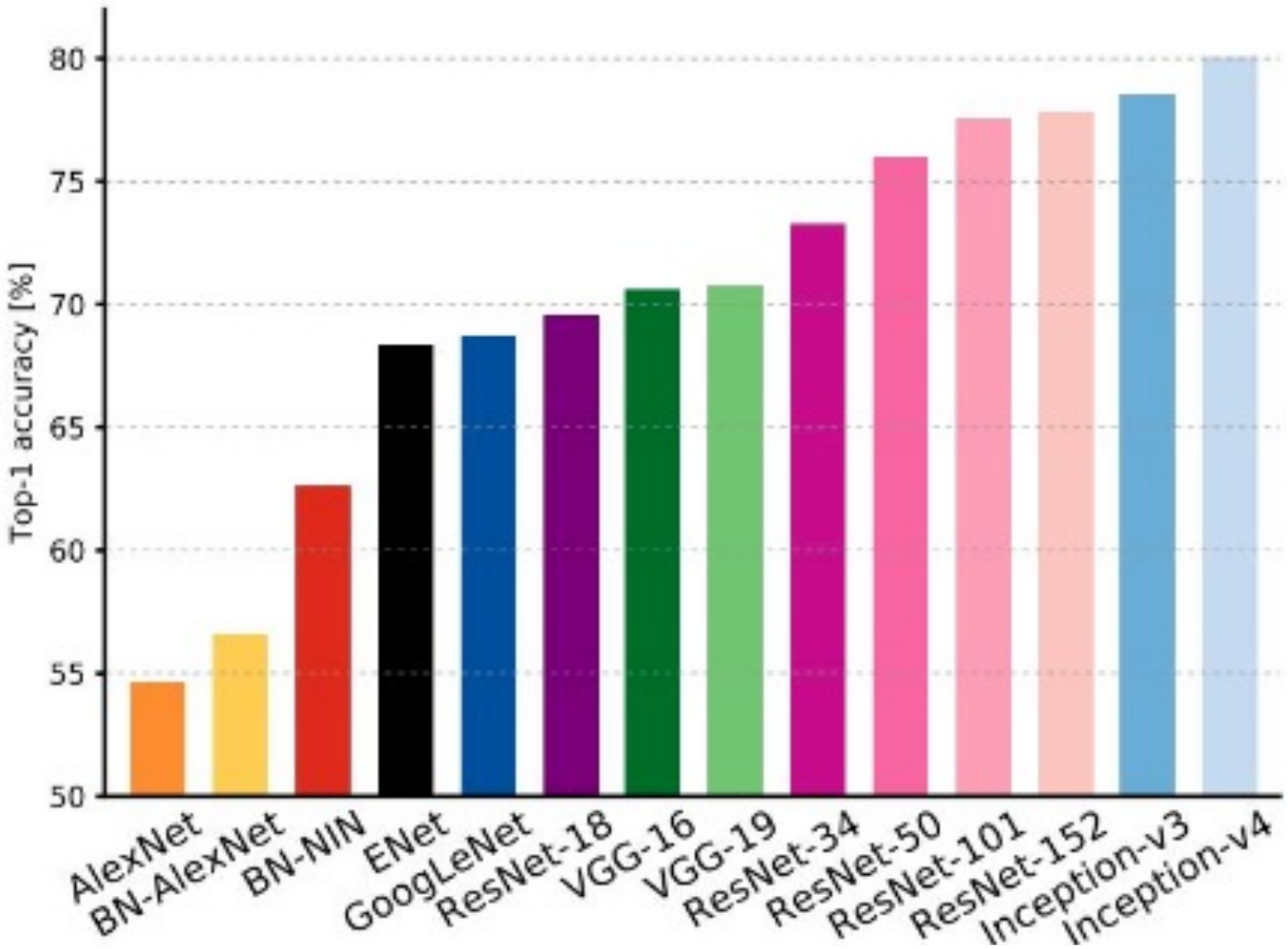
[He, 2016] **Deep residual learning for image recognition**, He et al., *CVPR, 2016*
[Kolesnikov, 2020] **Big transfer (bit): General visual representation learning**, Kolesnikov et al., *ECCV, 2020*

ResNet (2016)

ResNet18 - Structural Details														
#	Input Image			output			Layer	Stride	Pad	Kernel		in	out	Param
1	227	227	3	112	112	64	conv1	2	1	7	7	3	64	9472
	112	112	64	56	56	64	maxpool	2	0.5	3	3	64	64	0
2	56	56	64	56	56	64	conv2-1	1	1	3	3	64	64	36928
3	56	56	64	56	56	64	conv2-2	1	1	3	3	64	64	36928
4	56	56	64	56	56	64	conv2-3	1	1	3	3	64	64	36928
5	56	56	64	56	56	64	conv2-4	1	1	3	3	64	64	36928
6	56	56	64	28	28	128	conv3-1	2	0.5	3	3	64	128	73856
7	28	28	128	28	28	128	conv3-2	1	1	3	3	128	128	147584
8	28	28	128	28	28	128	conv3-3	1	1	3	3	128	128	147584
9	28	28	128	28	28	128	conv3-4	1	1	3	3	128	128	147584
10	28	28	128	14	14	256	conv4-1	2	0.5	3	3	128	256	295168
11	14	14	256	14	14	256	conv4-2	1	1	3	3	256	256	590080
12	14	14	256	14	14	256	conv4-3	1	1	3	3	256	256	590080
13	14	14	256	14	14	256	conv4-4	1	1	3	3	256	256	590080
14	14	14	256	7	7	512	conv5-1	2	0.5	3	3	256	512	1180160
15	7	7	512	7	7	512	conv5-2	1	1	3	3	512	512	2359808
16	7	7	512	7	7	512	conv5-3	1	1	3	3	512	512	2359808
17	7	7	512	7	7	512	conv5-4	1	1	3	3	512	512	2359808
	7	7	512	1	1	512	avg pool	7	0	7	7	512	512	0
18	1	1	512	1	1	1000	fc					512	1000	513000
Total														11,511,784

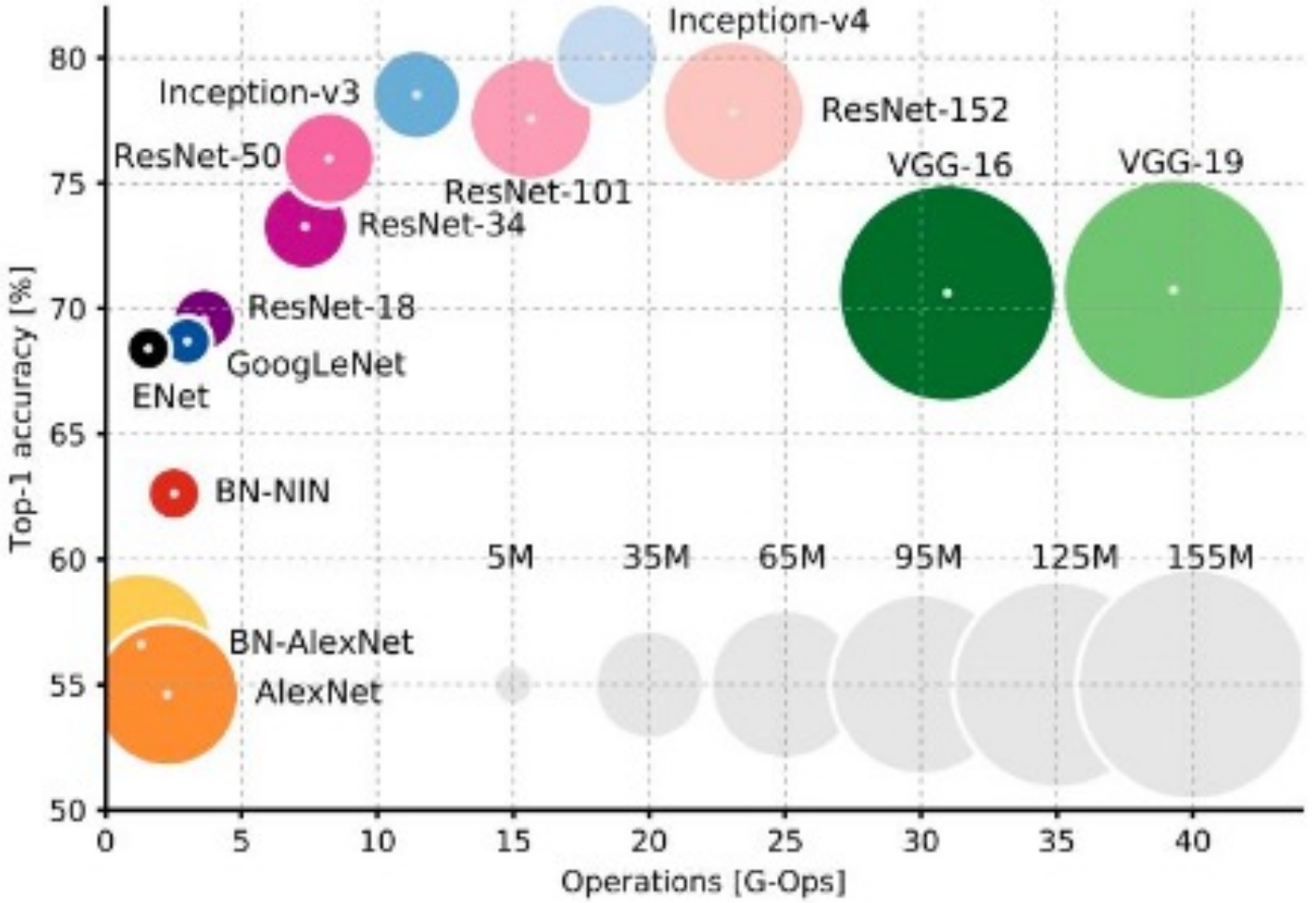
Illustration from: https://towardsdatascience.com/the-w3h-of-alexnet-vggnet-resnet-and-inception-7baaaecccc96

State-of-the-art as of 2017



From https://medium.com/@sidereal/cnns-architectures-lenet-alexnet-vgg-googlenet-resnet-and-more-666091488df5

State-of-the-art as of 2017



From https://culurciello.github.io/tech/2016/06/04/nets.html

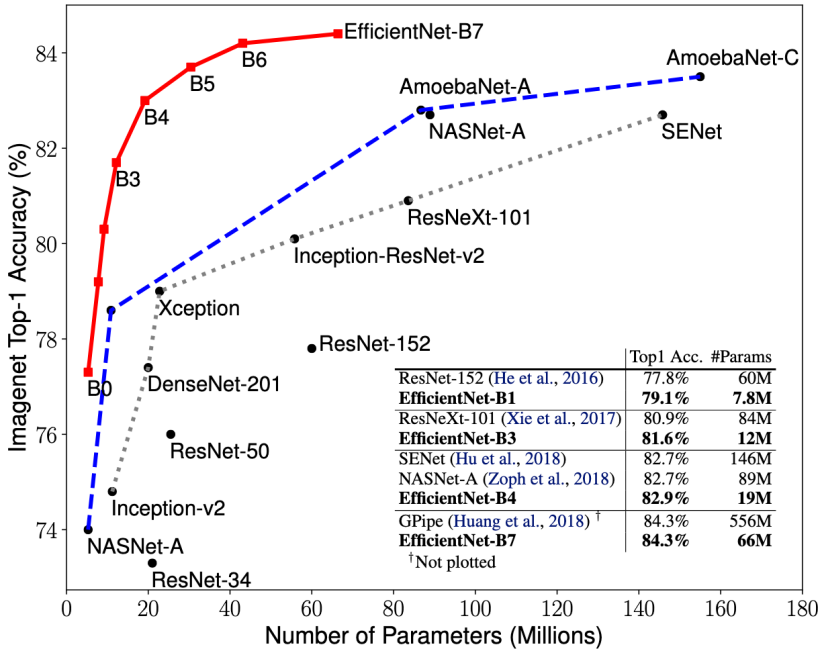
Recent convolutional architectures [optional]

- ResNeXt [Xie, 2017] is a modified ResNet architecture that uses **group convolutions** for higher parameter efficiency and introduces the concept of cardinality

- MobileNets [Howard, 2017] are lightweight architectures suited to **mobile and embedded vision applications**, thanks to the use of depthwise separable convolutions

- EfficientNets [Tan, 2019] are resource-efficient architectures via an **optimal compound scaling** of network depth, width, resolution
  - State-of-the-art convolutional architecture for a while

- **ConvNeXt** [Liu, 2022] brings many incremental improvements to ResNets to make them state-of-the-art again against new vision transformer architectures



	Top1 Acc.	#Params
ResNet-152 (He et al., 2016)	77.8%	60M
EfficientNet-B1	**79.1%**	**7.8M**
ResNeXt-101 (Xie et al., 2017)	80.9%	84M
EfficientNet-B3	**81.6%**	**12M**
SENet (Hu et al., 2018)	82.7%	146M
NASNet-A (Zoph et al., 2018)	82.7%	89M
EfficientNet-B4	**82.9%**	**19M**
GPipe (Huang et al., 2018) †	84.3%	556M
EfficientNet-B7	**84.3%**	**66M**

†Not plotted

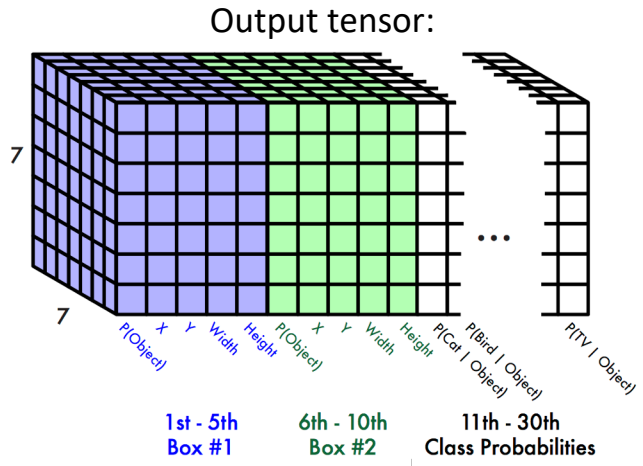
From [Tan, 2019]

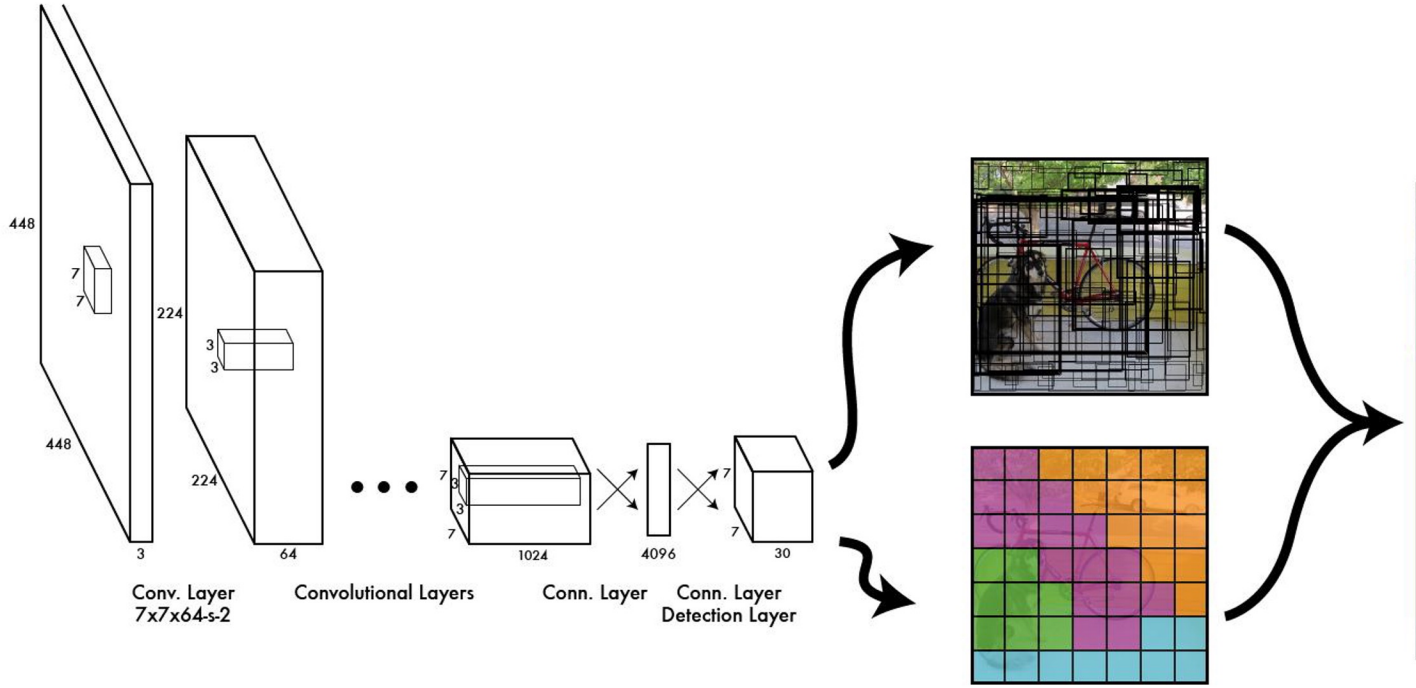
[Xie, 2017] **Aggregated residual transformations for deep neural networks**, Xie, S. et al. (Facebook), *CVPR, 2017*
[Howard, 2017] **Mobilenets: Efficient convolutional neural networks for mobile vision applications**, Howard, A.G. et al. (Google), *arXiv, 2017*
[Tan, 2019] **Efficientnet: Rethinking model scaling for convolutional neural networks**, Tan, M., et Le, Q. (Google), *ICML, 2019*
[Liu, 2022] **A convnet for the 2020s**, Liu, Z. et al. (Facebook), *CVPR, 2022*

Applications of ConvNets

Object detection

- We can also **detect the position and class** of **all** objects in images
- YOLO [Redmon, 2016] finds bounding boxes and assigns classes in a single forward pass of the network
  - The image is processed into a $7\times7\times30$ output tensor predicting box locations and class probabilities
  - YOLOv8 as of 2023





[Redmon, 2016] **You only look once: Unified, real-time object detection**, Redmon, J. et al., *CVPR, 2016*

Image segmentation

- Goal: **contour relevant structures** in images
  - Assign a label to each pixel

- Example in medical imaging: brain tumor segmentation

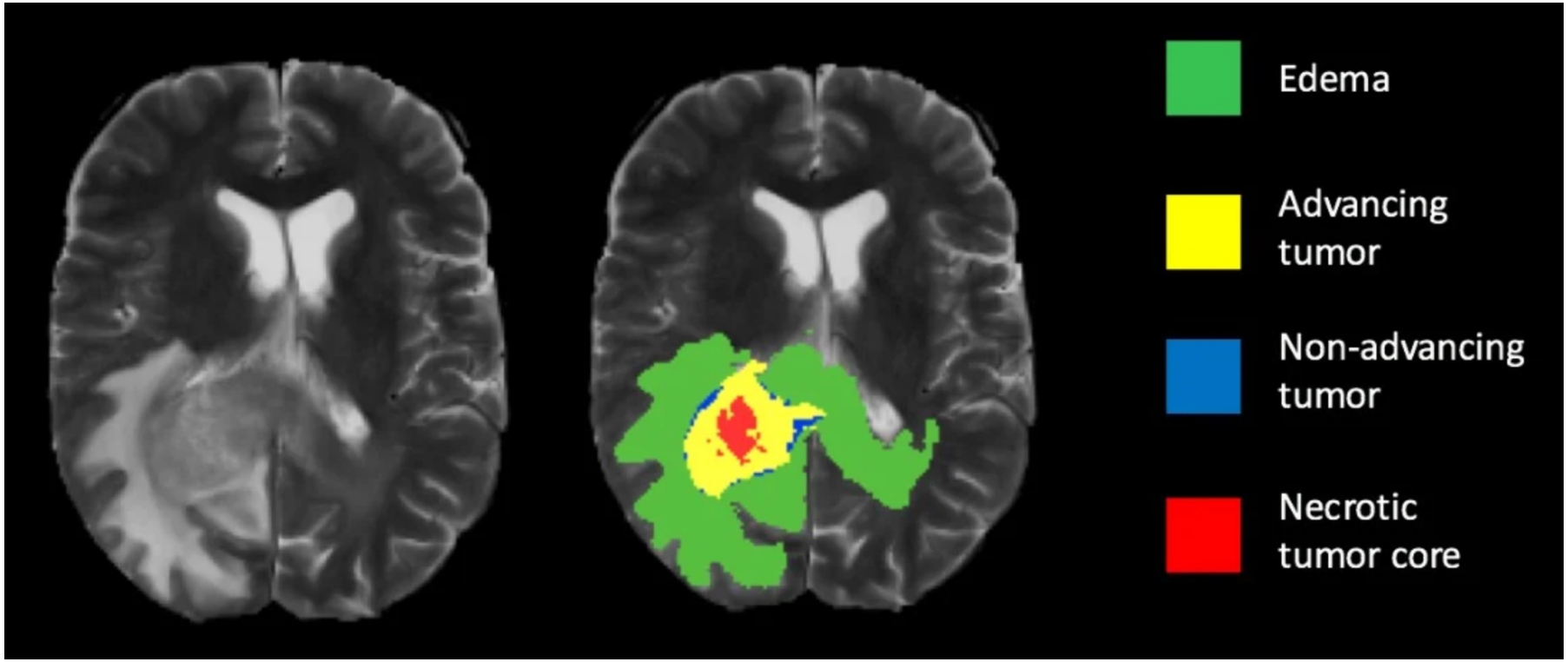
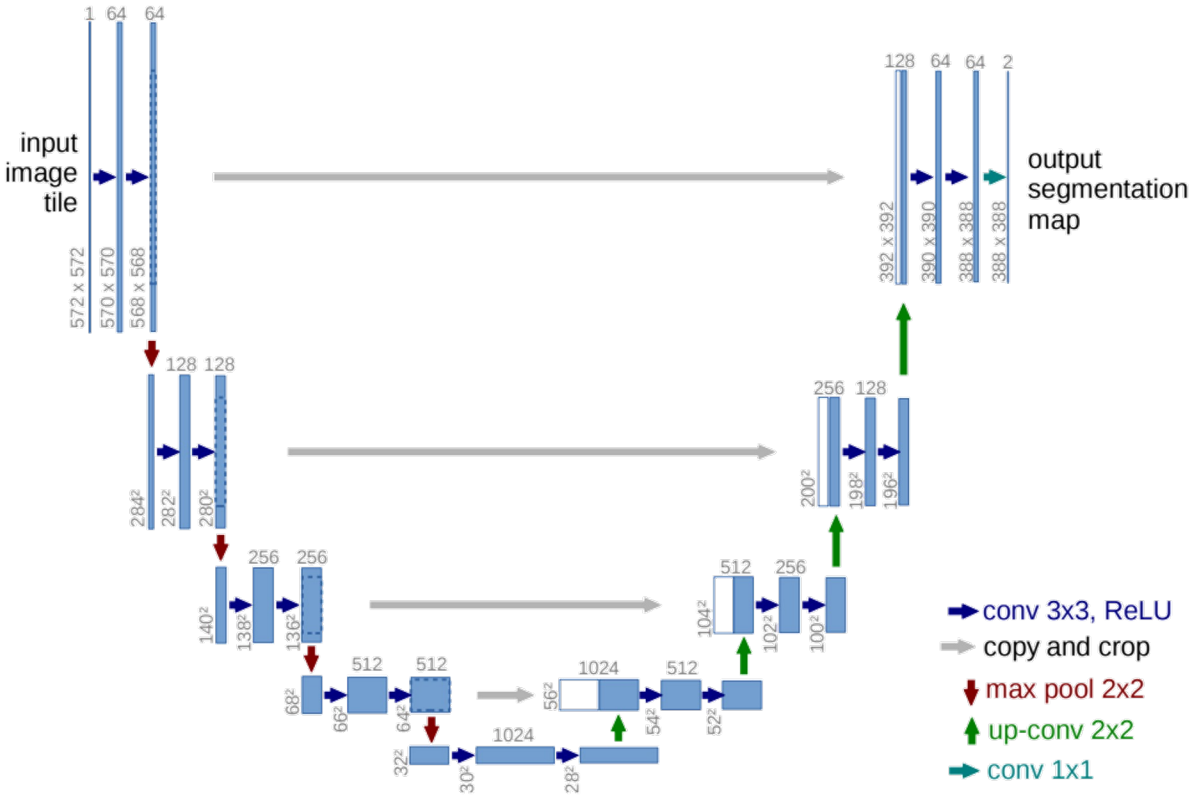


Illustration from data-flair.training

Image segmentation with CNNs

- The U-net [Ronneberger, 2015] is a network architecture that maps an input image to a segmentation mask
  - A **down-sampling branch** captures local and global context while the **upsampling branch** allows for pixel label predictions at the same resolution as the input image



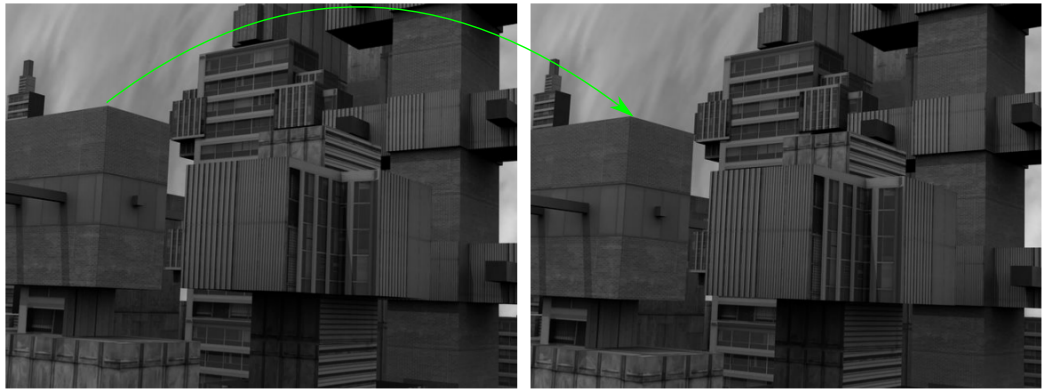
[Ronneberger, 2015] **U-net: Convolutional networks for biomedical image segmentation**, Ronneberger, O. et al., *MICCAI, 2015*

Motion estimation

- **Motion estimation** is a central task for many image processing and computer vision problems: tracking, video editing

- **Optical flow** involves estimating a vector field (u, v): $\mathbb{R}^2 \rightarrow \mathbb{R}^2$ where each vector is the displacement of pixel (x, y) from an image I_1 to I_2

$$I_1(x, y) \approx I_2(x + u(x, y), y + v(x, y))$$



(a) Frame 1

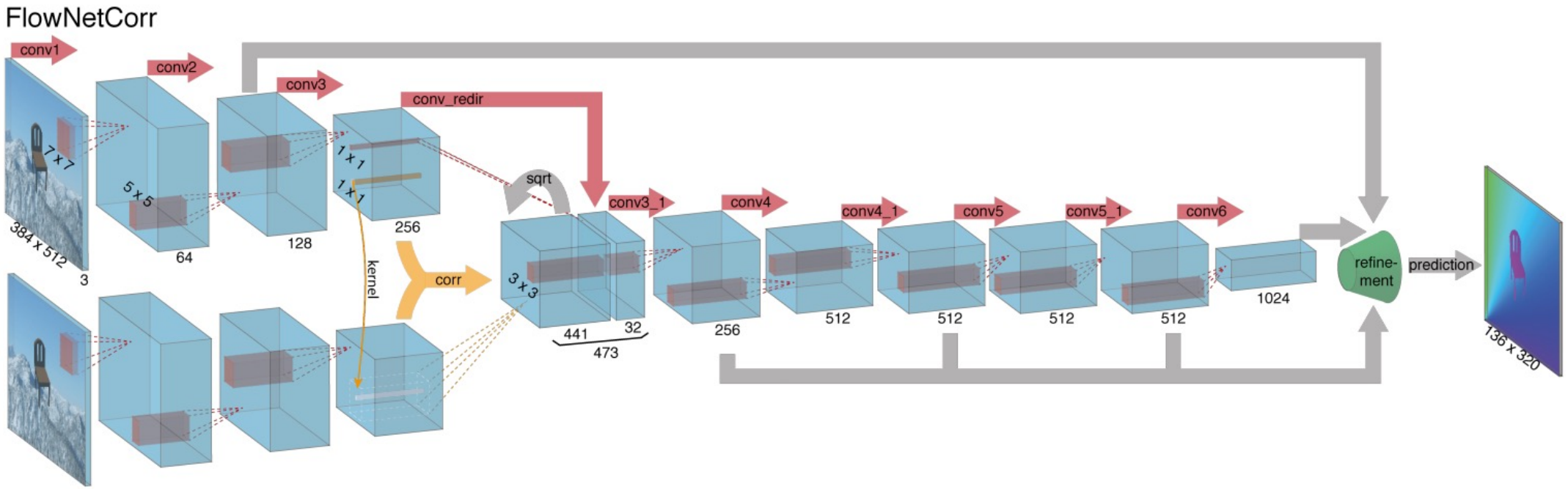
(b) Frame 2

Optical flow

Illustration from: **BriefMatch: Dense binary feature matching for real-time optical flow estimation**, Eilertsen, G, Forssén, P-E, Unger, J., *Scandinavian Conference on Image Analysis, 2017*

Motion estimation with CNNs

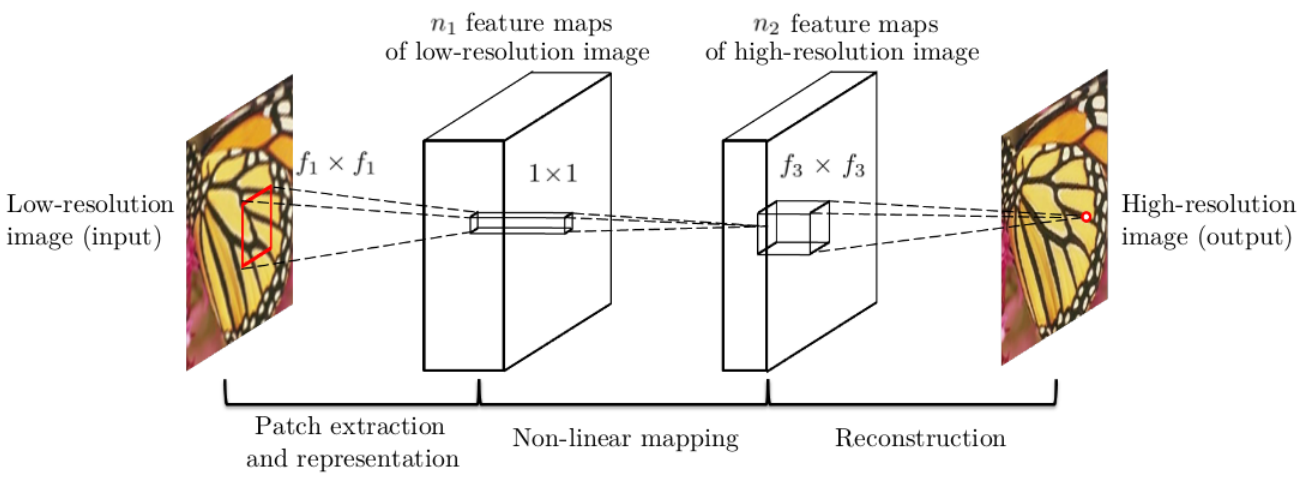
- Example: FlowNet [Fischer, 2015] first extracts meaningful features from the pair of images (in parallel) and then combines them to compute the optical flow

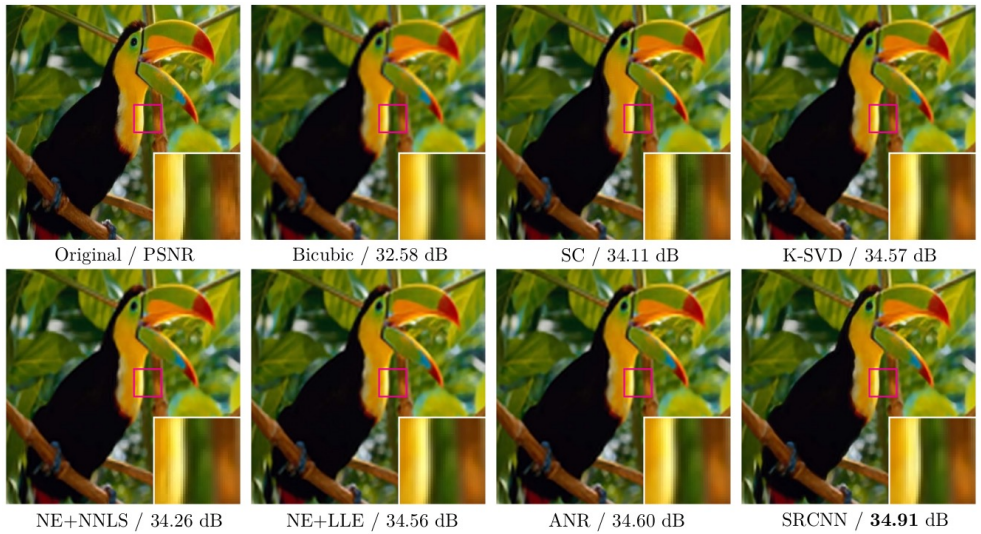


[Fischer, 2015] **FlowNet: Learning optical flow with convolutional networks**, Fischer et al., *ICCV, 2015*

Super-resolution

- Image super-resolution: go from a low-resolution image to a higher resolution one
- Relatively straightforward approach with a CNN: [Chao, 2014]





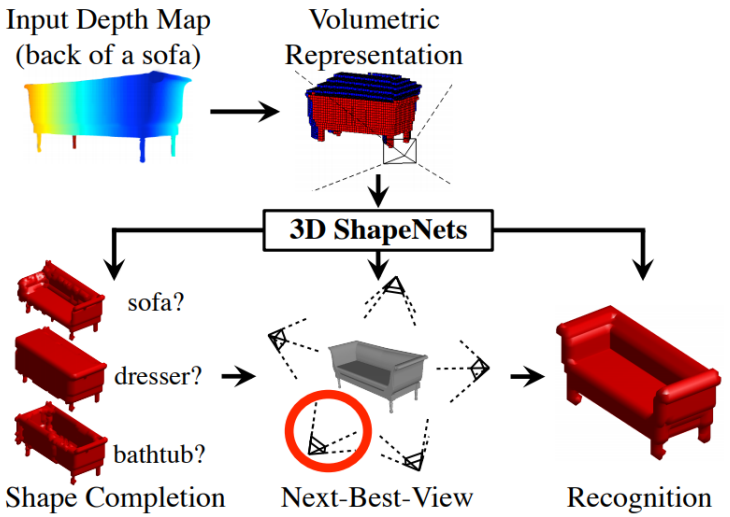
- Drawback, highly dependent on degradation used in lower resolution images in database

[Chao, 2014] **Learning a deep convolutional network for image super-resolution**, Chao et al., *ECCV, 2014*

Shapes and point clouds

- CNNs require regular grids. Point cloud data are not in this format
- Nevertheless, ways have been found to deal with this

- ShapeNet* rasterizes the shapes on regular 3D image grids that are processed by CNNs
- Each voxel (3D pixel) contains a Bernoulli random variable representing the probability of this voxel belonging to the shape
- This general approach (using voxels) is followed in many other approaches



*3d shapenets: A deep representation for volumetric shapes, Zhirong, W. et al., *CVPR, 2015*

CNNs for style transfer

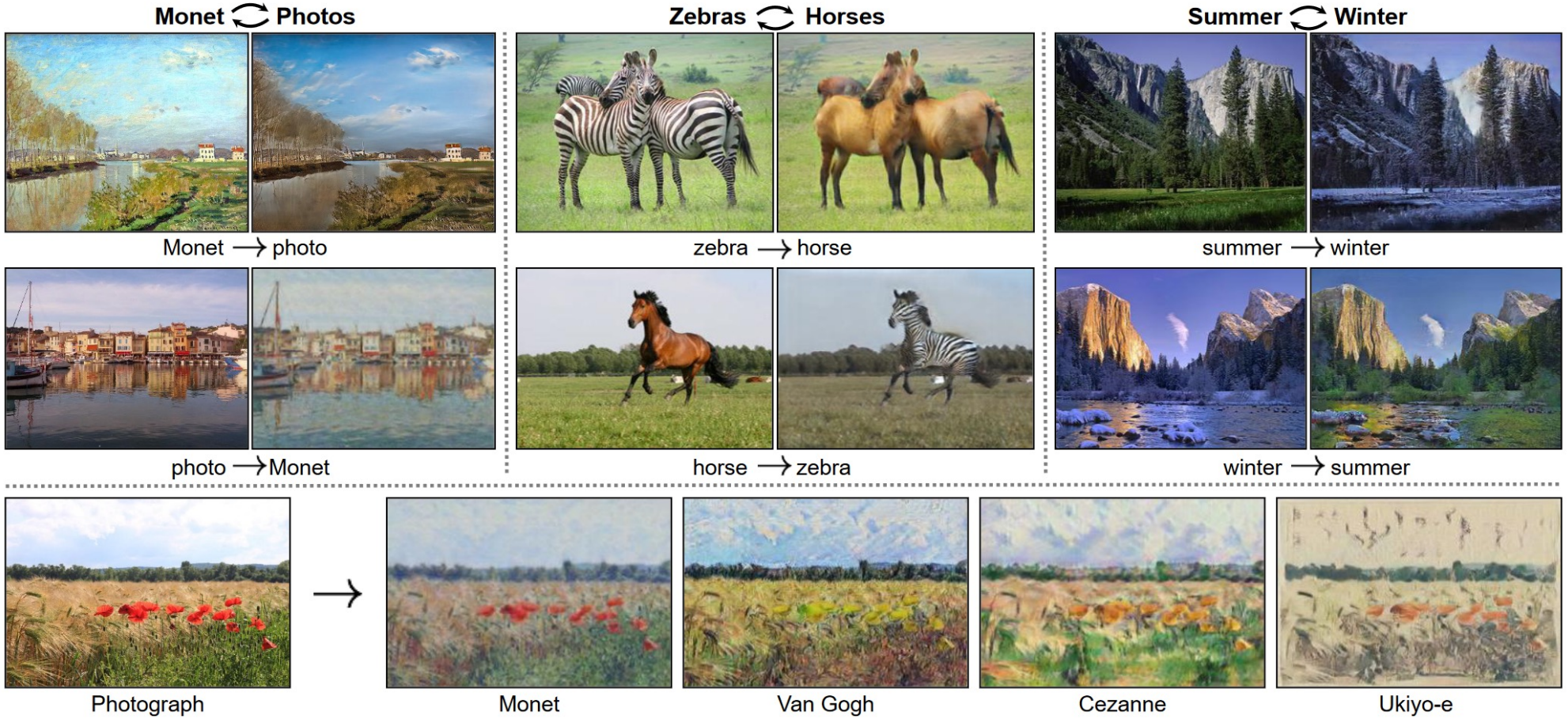
Input image

Target style

Output image

CNNs for style transfer

- Example: CycleGANs for unpaired **image-to-image translation** [Zhu, 2017]
  - Map between source and target domains with generator networks G_{ST} and G_{TS} s.t. $G_{ST}\left(G_{TS}(x)\right) \simeq x$.



[Zhu, 2017] **Unpaired image-to-image translation using cycle-consistent adversarial networks**, Zhu, J. Y. et al., *ICCV, 2017*

Visualizing CNNs & adversarial examples

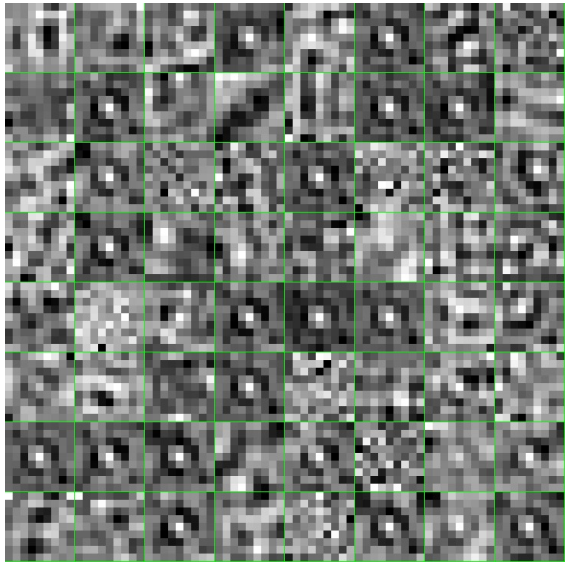
Interpreting CNNs

- As is often the case in deep learning, it is very difficult to understand what is going on in CNNs
- Much research is being dedicated to understanding these networks
  - Explainable AI
- We discuss two topics related to interpretability
  - Visualizing CNNs
  - Adversarial examples

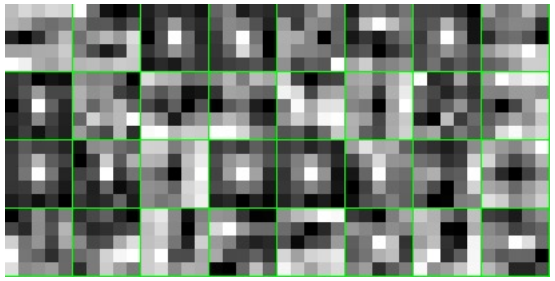
Following slides adapted from Alasdair Newson's

Visualizing CNNs

- We would like to **understand what CNNs are learning**
  - Unfortunately filters are difficult to interpret (especially deeper layers)



Layer 1 filters



Layer 3 filters

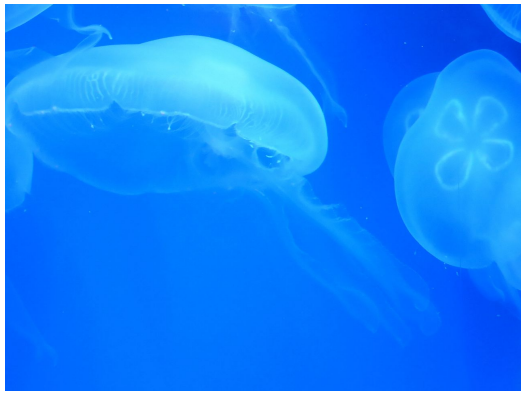
- Therefore, much research has been dedicated to **visualizing** CNNs

“Deep dream”

- Idea : “invert” CNN, find x to maximise the output of a certain layer
  - Understand what this layer is “seeing”
- This is **possible due to backpropagation**

Basic CNN visualisation algorithm

- Choose a layer ℓ to visualise
  - $x_0 \sim \mathcal{N}(0, 1)$
  - For $i = 1 \dots N$
    - $x^i = x^{i-1} + \lambda \nabla_x \|u^\ell(x^{i-1})\|$ Gradient ascent
- Return x^N



$$x^i = x^{i-1} + \lambda \nabla_x \|u^\ell(x^{i-1})\| \xrightarrow[\text{Gradient ascent}]{}$$

Visualizing features

- Generalization: **maximize response to a given filter response**

- Choose layer l, filter k and pixel (i, j)
- Random initialization x_0, constrain norm of solution x

$$\hat{x} = \underset{x}{\operatorname{argmax}}\, u^l_{i,j,k}$$

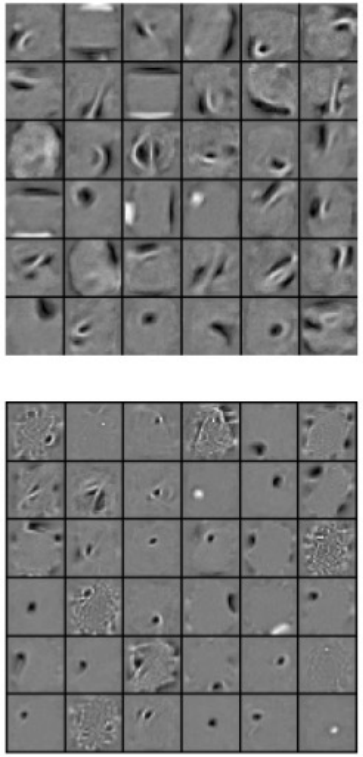
$$\text{with } \|x\| = \rho$$

- Optimization: **gradient ascent**

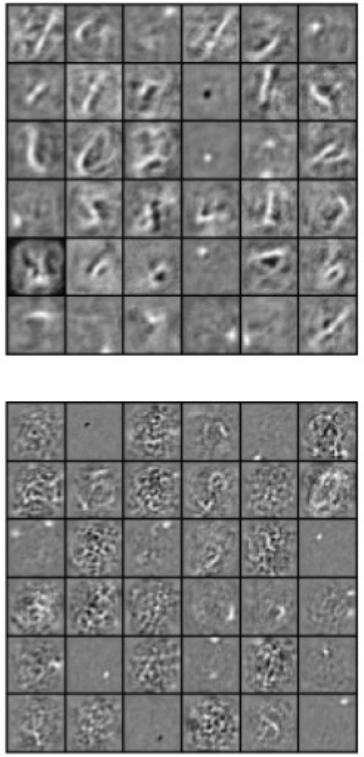
Visualizing Higher-Layer Features of a Deep Network, Erhan, Bengio, Courville, Vincent, *University of Montreal, 2009*

Visualizing CNNs

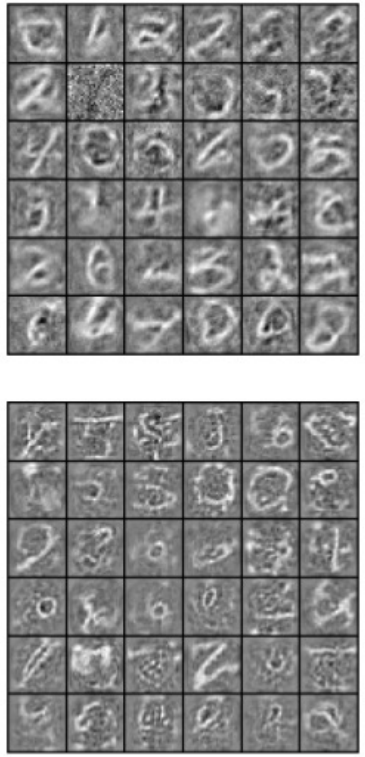
- Maximization of different activations applied to MNIST dataset



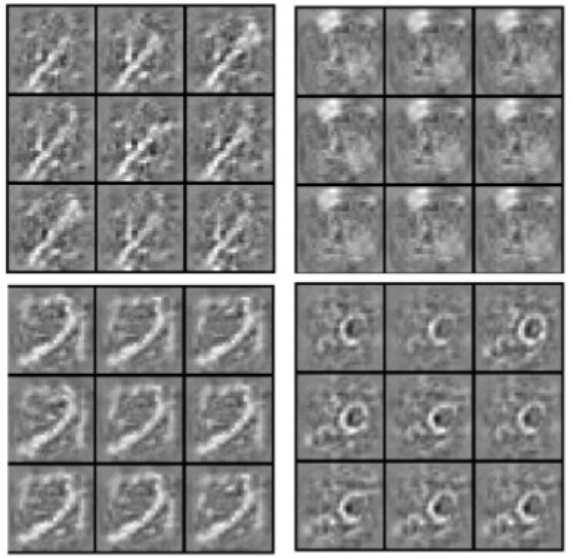
Layer 1



Layer 2



Layer 3



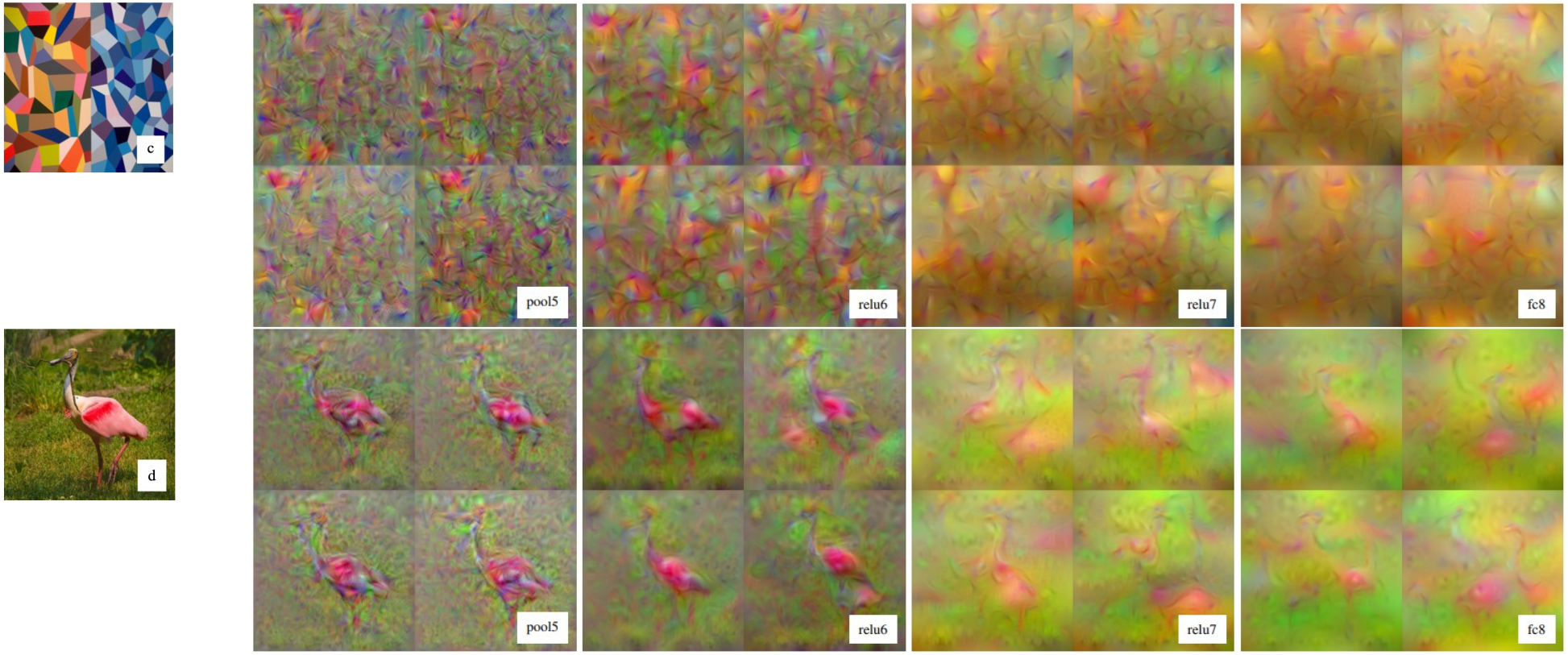
4 units of layer 3, 9 random initializations each

Visualizing Higher-Layer Features of a Deep Network, Erhan, Bengio, Courville, Vincent, *University of Montreal, 2009*

Visualizing CNNs

- More sophisticated approach: standard inverse problem with regularization

$$\hat{x} = \underset{x}{\operatorname{argmax}} \; \|f(x) - f_0\|_2^2 + \lambda \|x\|_2^2 + \mu \|\nabla x\|_2^2$$



Understanding Deep Image Representations by Inverting Them, Mahendran and Vedaldi, *CVPR, 2014*

Visualizing CNNs

- Another approach [Simonyan, 2013] proposes to see **what images correspond to what classes**

- Choose a class c, maximize the response of this class

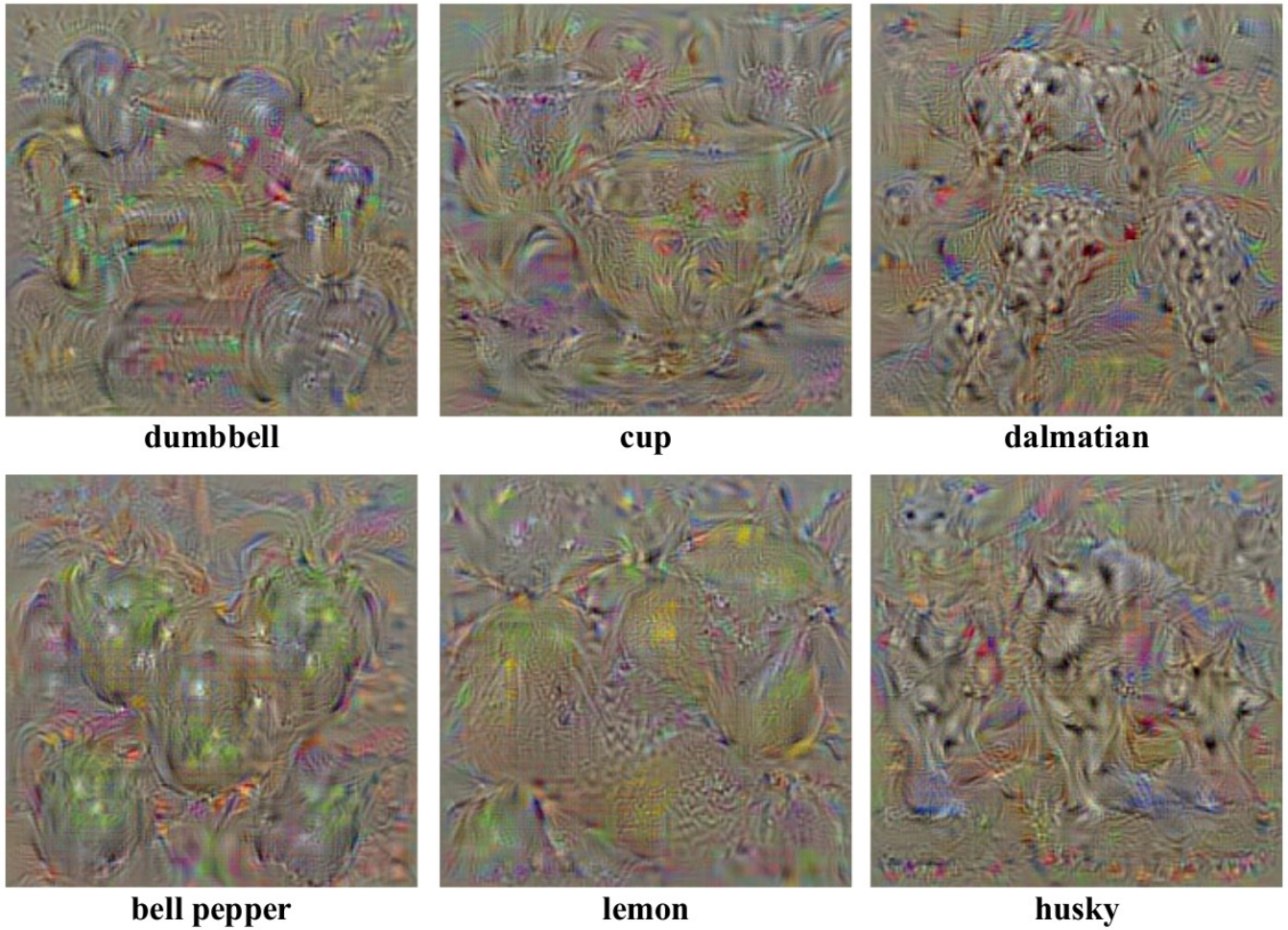
$$\hat{x} = \underset{x}{\operatorname{argmax}} f(x)_c - \lambda \|x\|_2^2$$

- Find an L_2-regularized image which maximizes the score for a given class c
- Initialize with random input image x_0

Deep Inside Convolutional Networks: Visualising Image Classification Models and Saliency Maps, Simonyan, Vedaldi, Zisserman, *arXiv preprint arXiv:1312.6034, 2013*

Visualizing CNNs

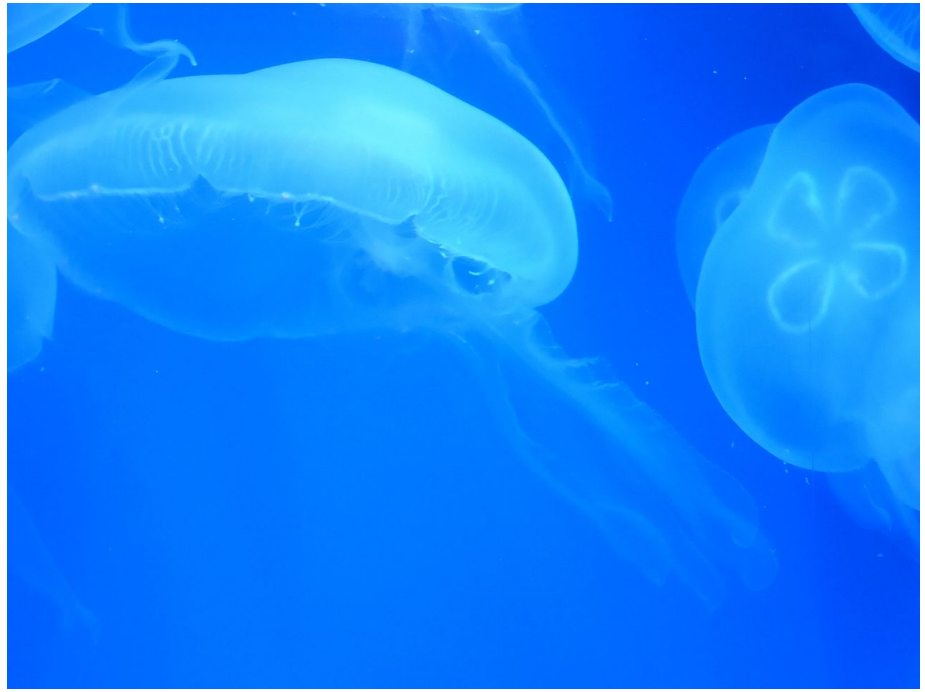
- Class-model visualization



Deep Inside Convolutional Networks: Visualising Image Classification Models and Saliency Maps, Simonyan, Vedaldi, Zisserman, *arXiv preprint arXiv:1312.6034, 2013*

Visualizing CNNs

- Similar idea with Inception architecture of Google: “Deep Dream”
- Maximize a class from input image



Input image

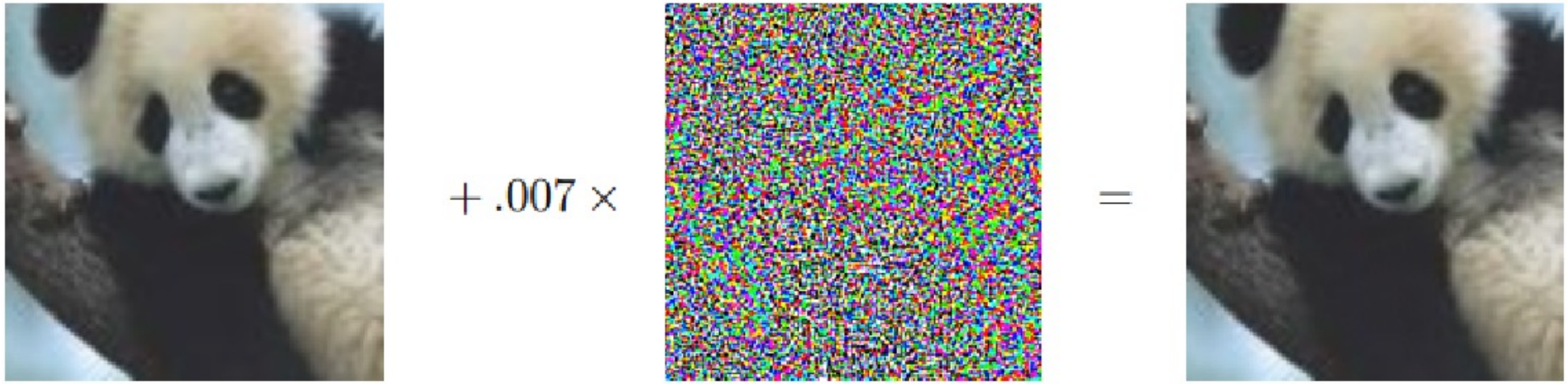
Maximizing dog category

Deepdream - a code example for visualizing neural networks, Mordvintsev, A., Olah, C. and Tyka, M. (Google Research), *2015*

Adversarial examples

- We often get the impression that CNNs are the end all be all of AI
- Consistently produces state-of-the-art results on images

- However, CNNs **are not infallible**: adversarial examples!



- How was this image created?

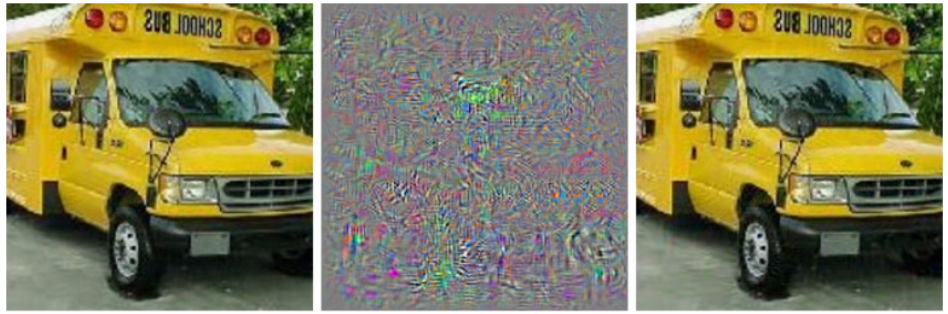
[Szegedy, 2013] **Intriguing properties of neural networks**, Szegedy, C. et al, *arXiv preprint arXiv:1312.6199, 2013*

Adversarial examples

- [Szegedy, 2013] proposes to add a small perturbation r that fools the classifier network f into choosing the wrong class c for $\hat{x} = x + r$

$$\underset{r}{\operatorname{argmin}} |r|^2 , \qquad \text{s.t. } f(x + r) = c, \qquad x + r \in [0,1]^n$$

- $\hat{x}$ is the closest example to x s.t. $\hat{x}$ is classified as class c
- Minimization with box-constrained L-BFGS algorithm

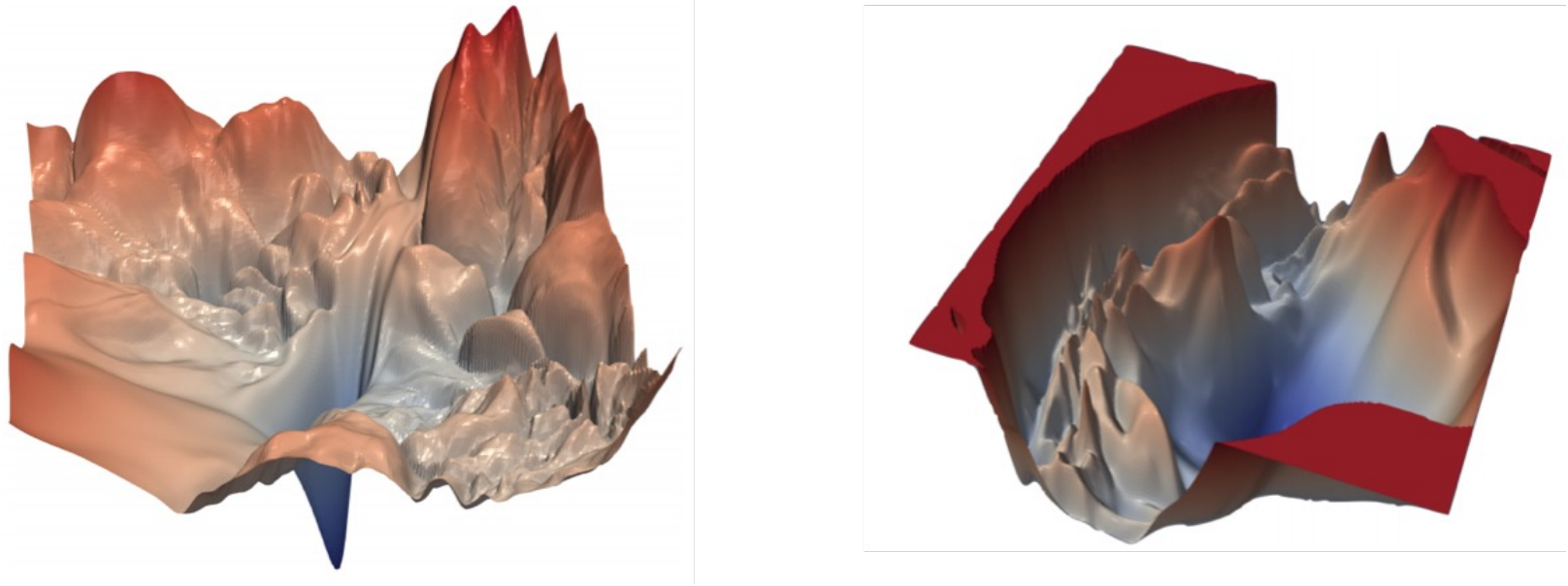


x r $\hat{x}$

[Szegedy, 2013] **Intriguing properties of neural networks**, Szegedy, C. et al, *arXiv preprint arXiv:1312.6199, 2013*

Adversarial examples

- Common explanation: the space of images is very high-dimensional, and contains many areas that are unexplored during training time

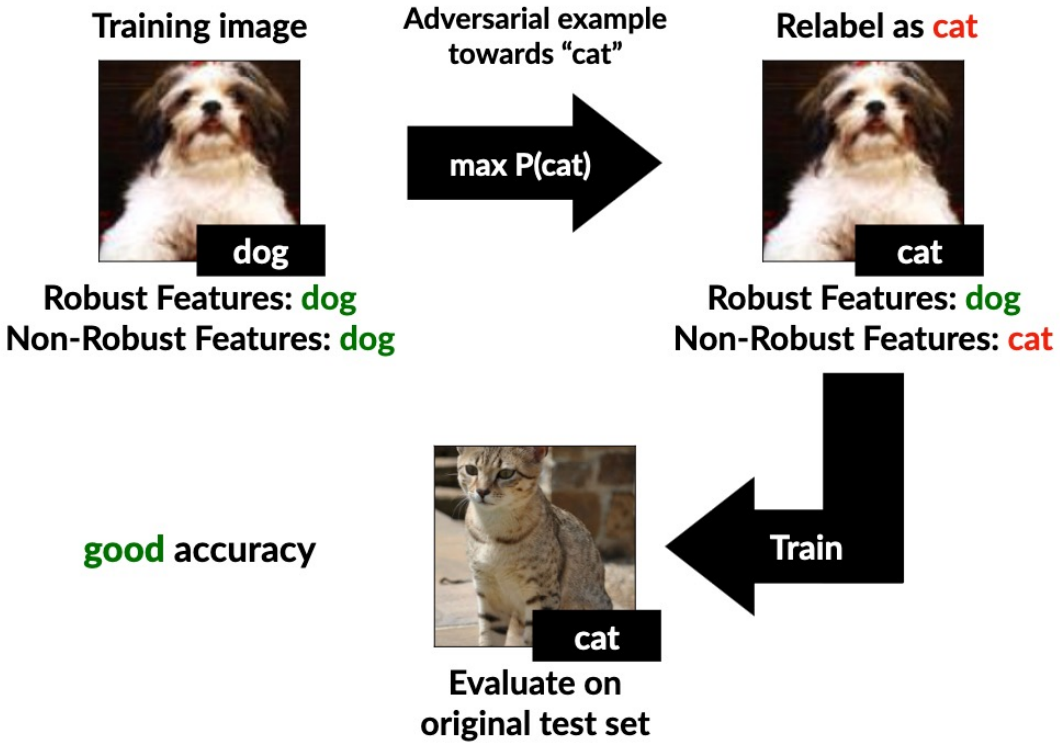


Example of loss surfaces in commonly used networks (Res-Nets)

Illustration from: **Visualizing the Loss Landscape of Neural Nets**, Li, H et al., *NIPS, 2018*

Adversarial examples

- Some features learnt from data do **not correspond to human-meaningful** explanations but do generalize well on test data [Ilyas, 2019]
  - These features are generally not robust to adversarial perturbations
  - Adversarial examples contain types of features not present in natural images



[Ilyas, 2019] **Adversarial examples are not bugs, they are features**, Ilyas et al., *NeuRIPS 2019*

Adversarial examples

- Many approaches to adversarial examples exist. [Goodfellow, 2015] proposes a principled way of creating these

- Consider the output of a fully connected layer $\langle w, \hat{x} \rangle = \langle w, x \rangle + \langle w, r \rangle$

- Let us set $r = \text{sign}(w)$. What happens to $\langle w, \hat{x} \rangle$?
  - Increase by nm as dimension n increases (m is average value of w)
  - However, $|r|_\infty$ does not increase with n

- Conclusion: we can add a small vector r that increases the output response $\langle w, \hat{x} \rangle$

[Goodfellow, 2015] **Explaining and Harnessing Adversarial Examples**, Goodfellow, I.J, Shlens, J. and Szegedy, C., *ICLR 2015*

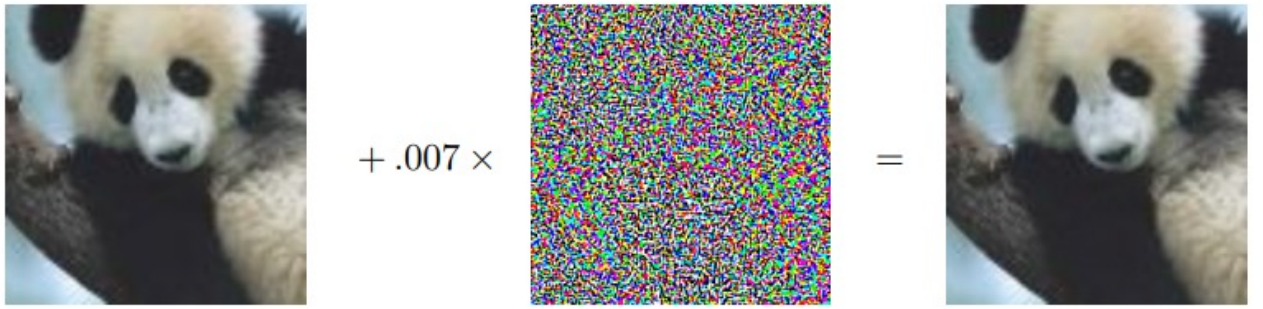
Adversarial examples

- [Goodfellow, 2015] considers a local linearization of the network's loss around x_0

$$\mathcal{L}(x) \approx \mathcal{L}(x_0) + \langle \nabla_x \mathcal{L}(\theta, x_0, y_0), x - x_0 \rangle$$

- Thus, the perturbation image $\hat{x}$ is set to

$$\hat{x} = x + \epsilon \cdot \text{sign}(\nabla_x \mathcal{L}(\theta, x, y))$$



[Goodfellow, 2015] **Explaining and Harnessing Adversarial Examples**, Goodfellow, I.J, Shlens, J. and Szegedy, C., *ICLR 2015*

Universal adversarial examples

- Even worse, it is possible to create **universal adversarial examples***
- Perturbations that fool a network for any given image class: $f(x+r) \neq f(x)$ for any x
  - $80-90\%$ fooling rate


common newt


carousel


grey fox


macaw

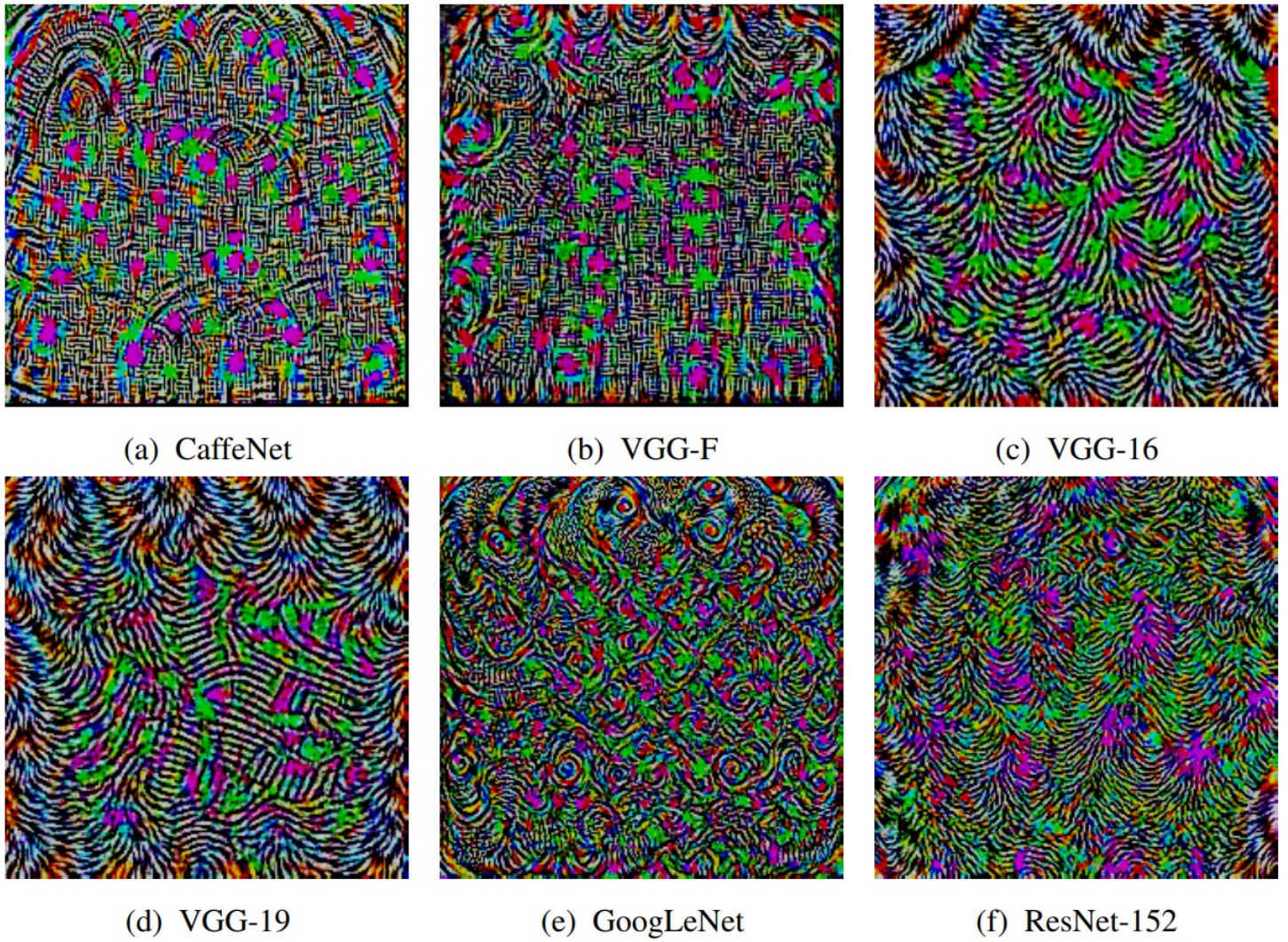

three-toed sloth


macaw

- Simple algorithm: initialize perturbation r, go through database adding specific perturbations to r, project onto set $\{r,\ \|r\| < \epsilon\}$
- What do these perturbations look like?

* **Universal adversarial perturbations**, Moosavi-Dezfooli, S-M, et al., *arXiv preprint, 2017*

Universal adversarial examples



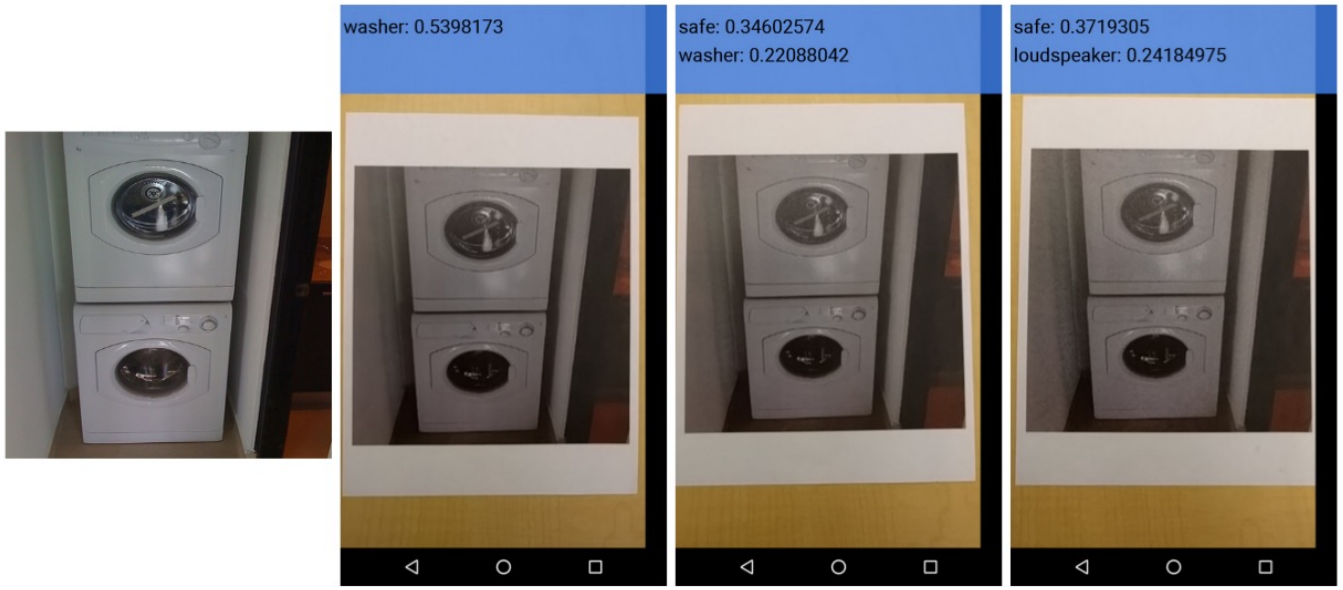
(a) CaffeNet (b) VGG-F (c) VGG-16

(d) VGG-19 (e) GoogLeNet (f) ResNet-152

* **Universal adversarial perturbations**, Moosavi-Dezfooli, S-M, et al., *arXiv preprint, 2017*

Adversarial examples

- Conclusion: CNNs are not necessarily robust
- Adversarial examples are a significant problem:
  - Even **printed photos** of adversarial examples work*



(a) Image from dataset (b) Clean image (c) Adv. image, $\epsilon = 4$ (d) Adv. image, $\epsilon = 8$

- Explaining and resisting adversarial examples is an active research topic

* **Adversarial Examples in the Physical World**, Kurakin, A., Goodfellow, I. J, Bengio, S. et al., *ICLR workshop, 2017*

Summary and perspectives

- CNNs represent the state-of-the-art in many different domains/problems
  - Nowadays, this is challenged by the rise of vision transformers

- **However: theoretical understanding is still limited**
  - This leads to problems such as adversarial examples
  - It is not clear whether CNNs are truly robust/generalizable
  - Important topic if CNNs are to be used in industrial applications

- Beyond architectures, much of recent progress has to do with how neural networks are trained
  - We have only discussed supervised learning
  - Many other tools are available: self-supervised learning, transfer learning, foundation models…
    → **[DS-telecom-23] Representation Learning** course