



Lecture 4: Convolutional neural networks



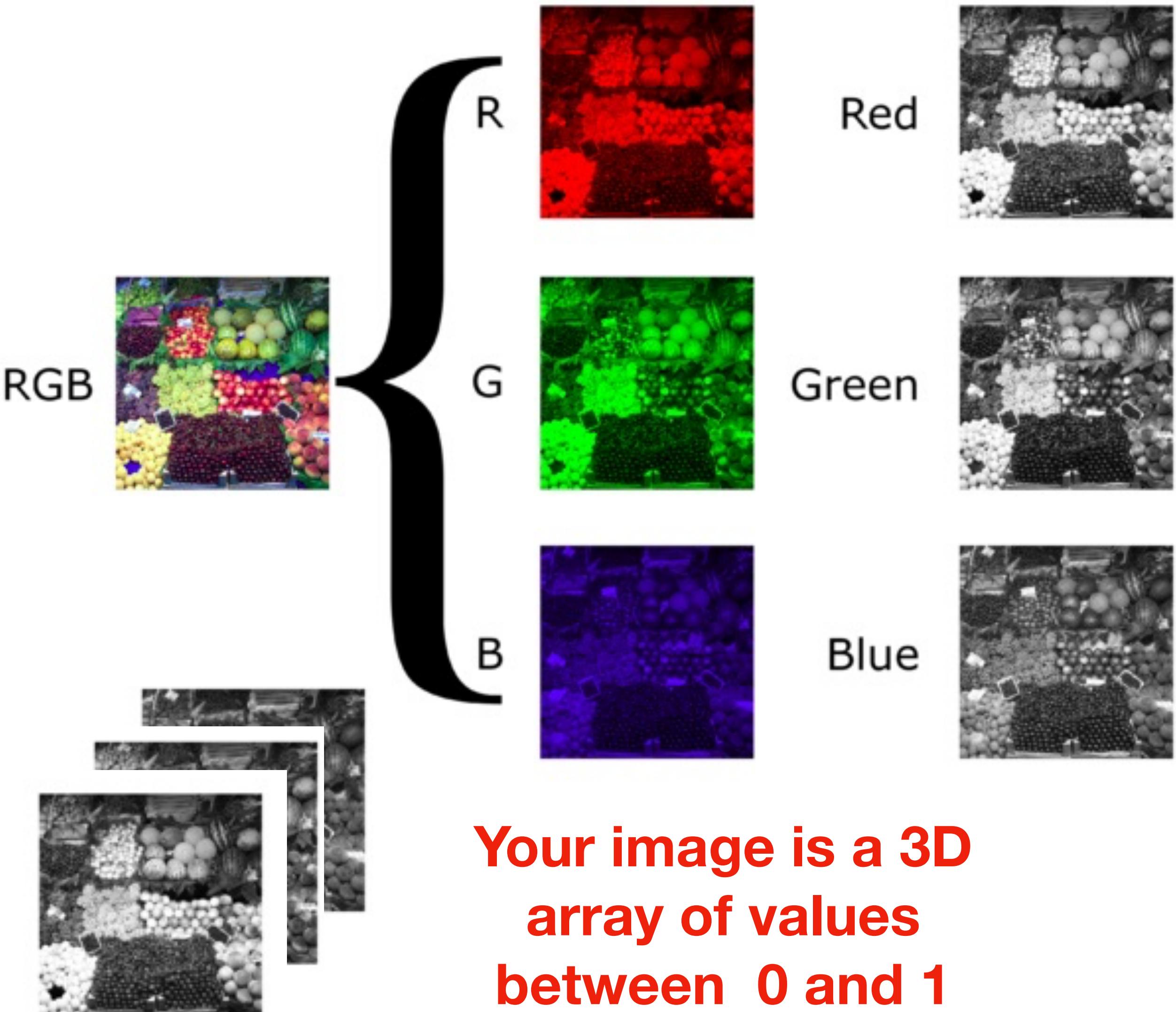
Program for Today

- [Table of Contents](#)
- [Acknowledgements](#)
- [Notation](#)
- ✓ • [1 Introduction](#)
- [Part I: Applied Math and Machine Learning Basics](#)
 - ✓ ○ [2 Linear Algebra](#)
 - ✓ ○ [3 Probability and Information Theory](#)
 - ✓ ○ [4 Numerical Computation](#)
 - ✓ ○ [5 Machine Learning Basics](#)
- [Part II: Modern Practical Deep Networks](#)
 - ✓ ○ [6 Deep Feedforward Networks](#)
 - [7 Regularization for Deep Learning](#)
 - [8 Optimization for Training Deep Models](#)
 - [9 Convolutional Networks](#)
 - [10 Sequence Modeling: Recurrent and Recursive Nets](#)
 - [11 Practical Methodology](#)
 - [12 Applications](#)
- [Part III: Deep Learning Research](#)
 - [13 Linear Factor Models](#)
 - [14 Autoencoders](#)
 - [15 Representation Learning](#)
 - [16 Structured Probabilistic Models for Deep Learning](#)
 - [17 Monte Carlo Methods](#)
 - [18 Confronting the Partition Function](#)
 - [19 Approximate Inference](#)
 - [20 Deep Generative Models](#)
- [Bibliography](#)
- [Index](#)

Date	Topic	Tutorial
Sep 17	Intro & class description	Linear Algebra in a nutshell + prob and stat
Sep 24	Basic of machine learning + Dense NN	Basic jupyter + DNN on mnist (give jet dnn as homework)
Oct 1	Convolutional NN	Convolutional NNs with MNIST
Oct 8	Training in practice: regularization, optimization, etc	Practical methodology
Oct 15		Tensor Flow tutorial (tbc)
Oct 22	Recurrent NN	Tutorial on RNNs
Oct 29	Graph NNs	Tutorial on Graph NNs
Nov 5	Unsupervised learning and anomaly detection	Autoencoders with MNIST
Nov 12	Generative models: GANs, VAEs, etc	Normalizing flows
Nov 19	Spiking Neural Network	Tutorial on neuromorphic chips & spiking NNs (tbc)
Nov 26	Network compression (pruning, quantization, Knowledge Distillation)	
Dec 3		Tutorial on hls4ml/qkeras
Dec 10		Transformers
Dec 17		tbd

Digital Image Processing

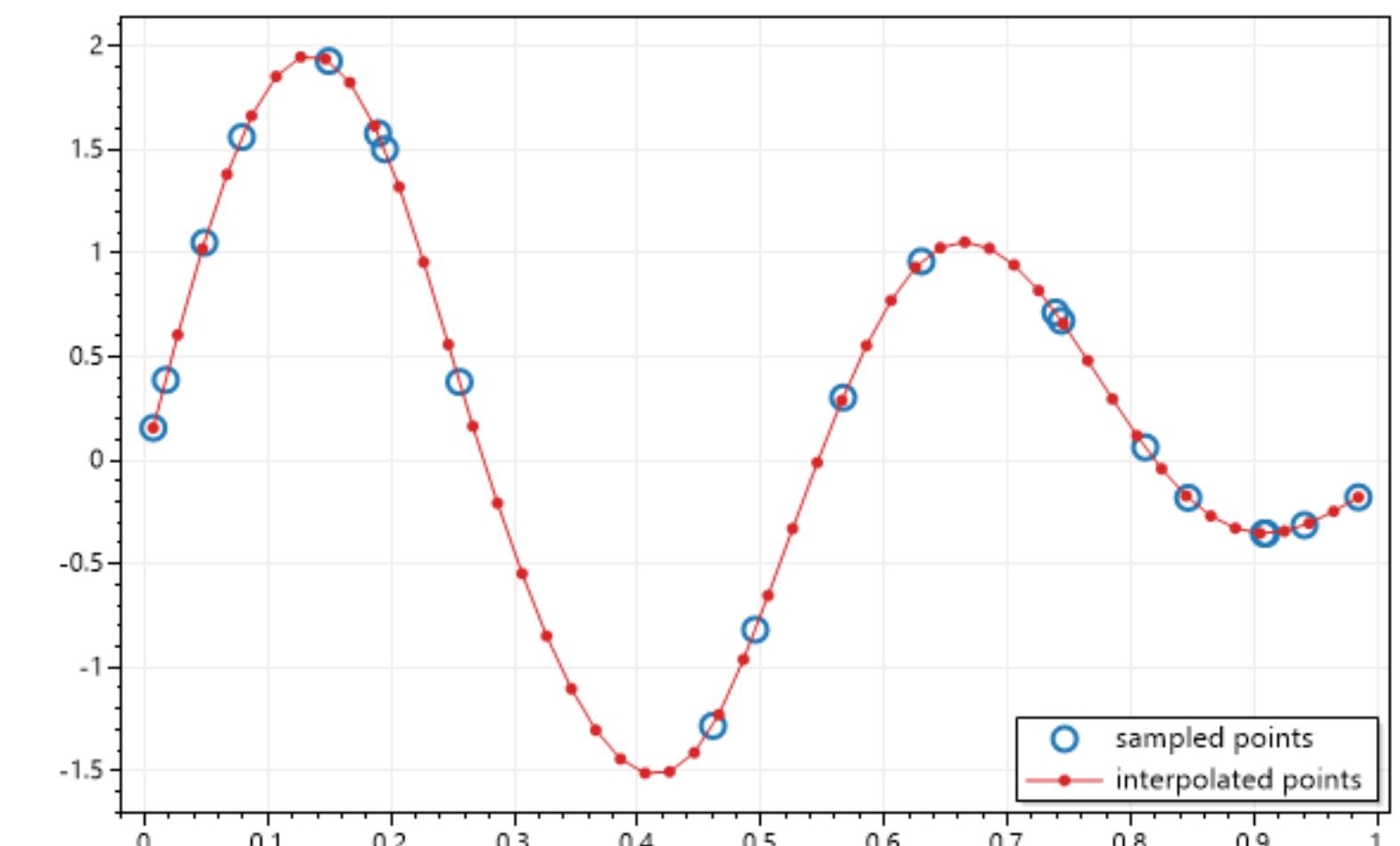
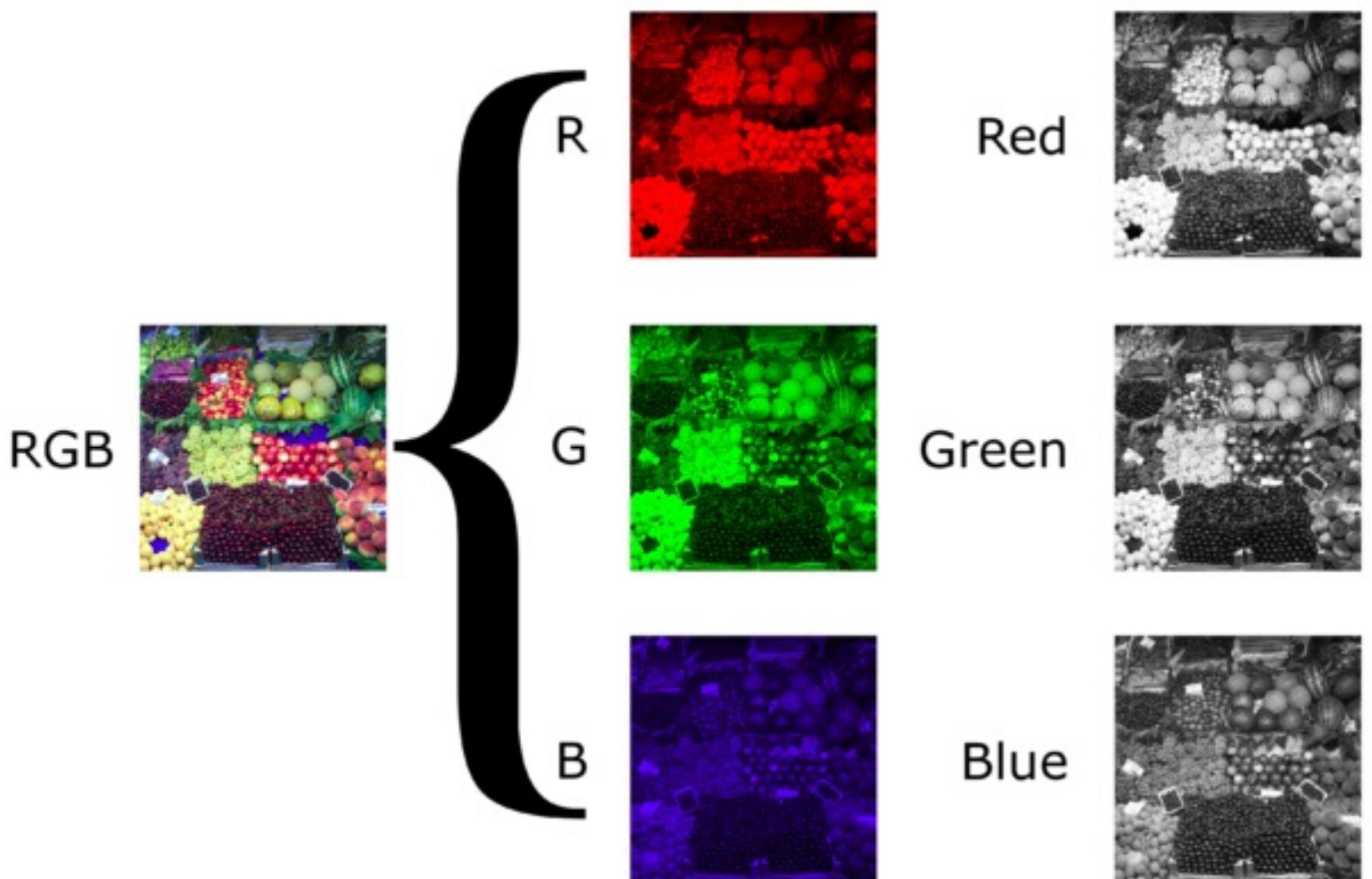
- *Each image is a matrix of pixels*
- *Each pixel comes with a color*
- *In RGB scheme, these are three colour values defined in $[0,1]$*
- *Each image becomes a 3D tensor*
- *3 channels of 2D pixelated images*
- *Digital images can be processed with Convolutional Neural Networks*



Your image is a 3D array of values between 0 and 1

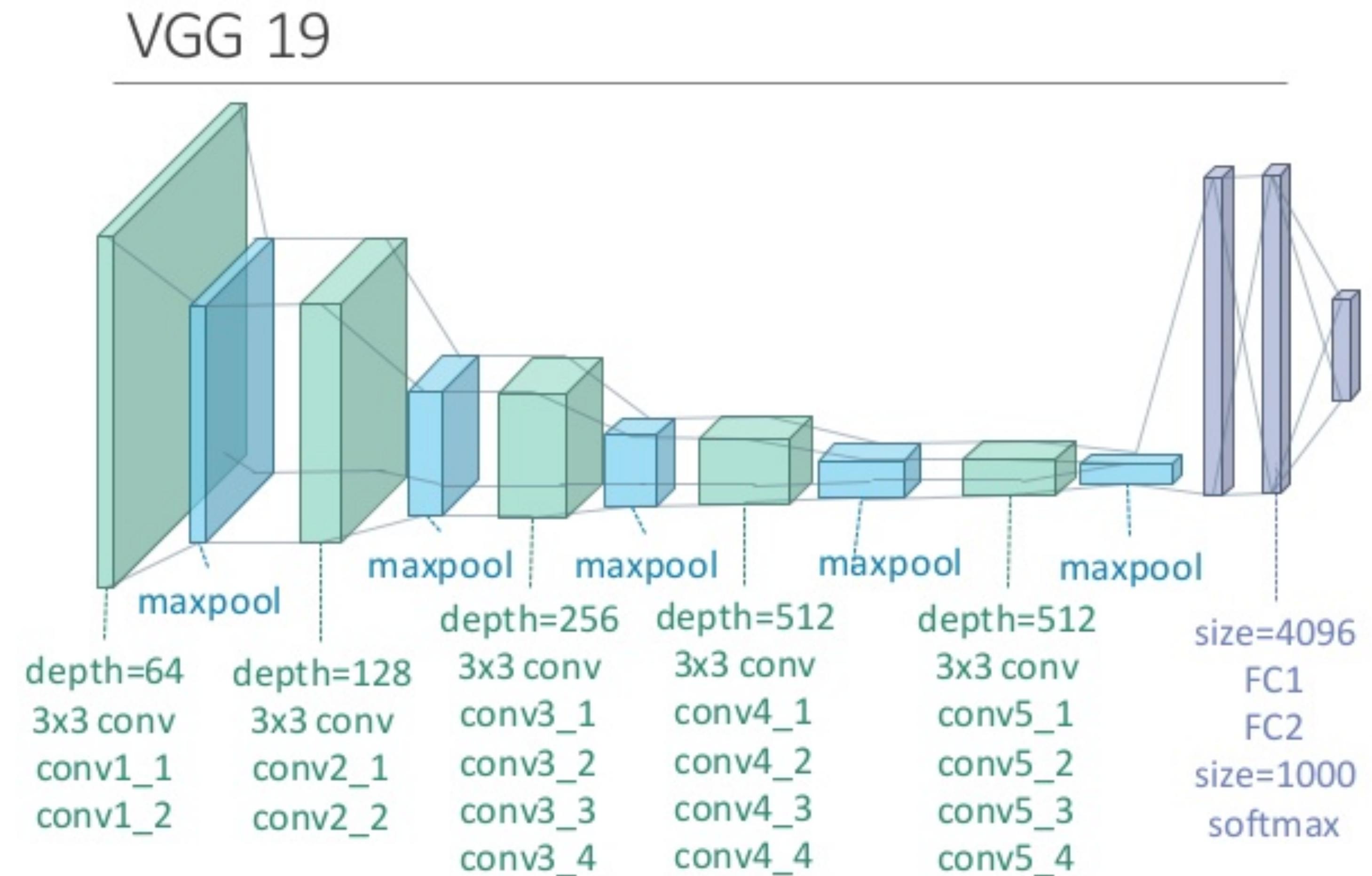
What is a CNN

- A special kind of NN designed process data coming in the form of a regular array
- An image, coming in RGB format
- A time series, i.e., the recording of a given readout at fixed intervals of time
- ...
- The name of the network comes from the mathematical operation that the network applies to the data: **Convolutional networks use convolution in place of general matrix multiplication in at least one of their layers.**



A three-step process

- A CNN processing comes in three steps. You could see it as a sequence of three layers
- Convolution: the actual image processing (inspired in structure by pre-Deep-Learning image processing)
- Activation: same as DNN
- Pooling: an array manipulation to reduce the image dimensionality, typically compensated by channel increase



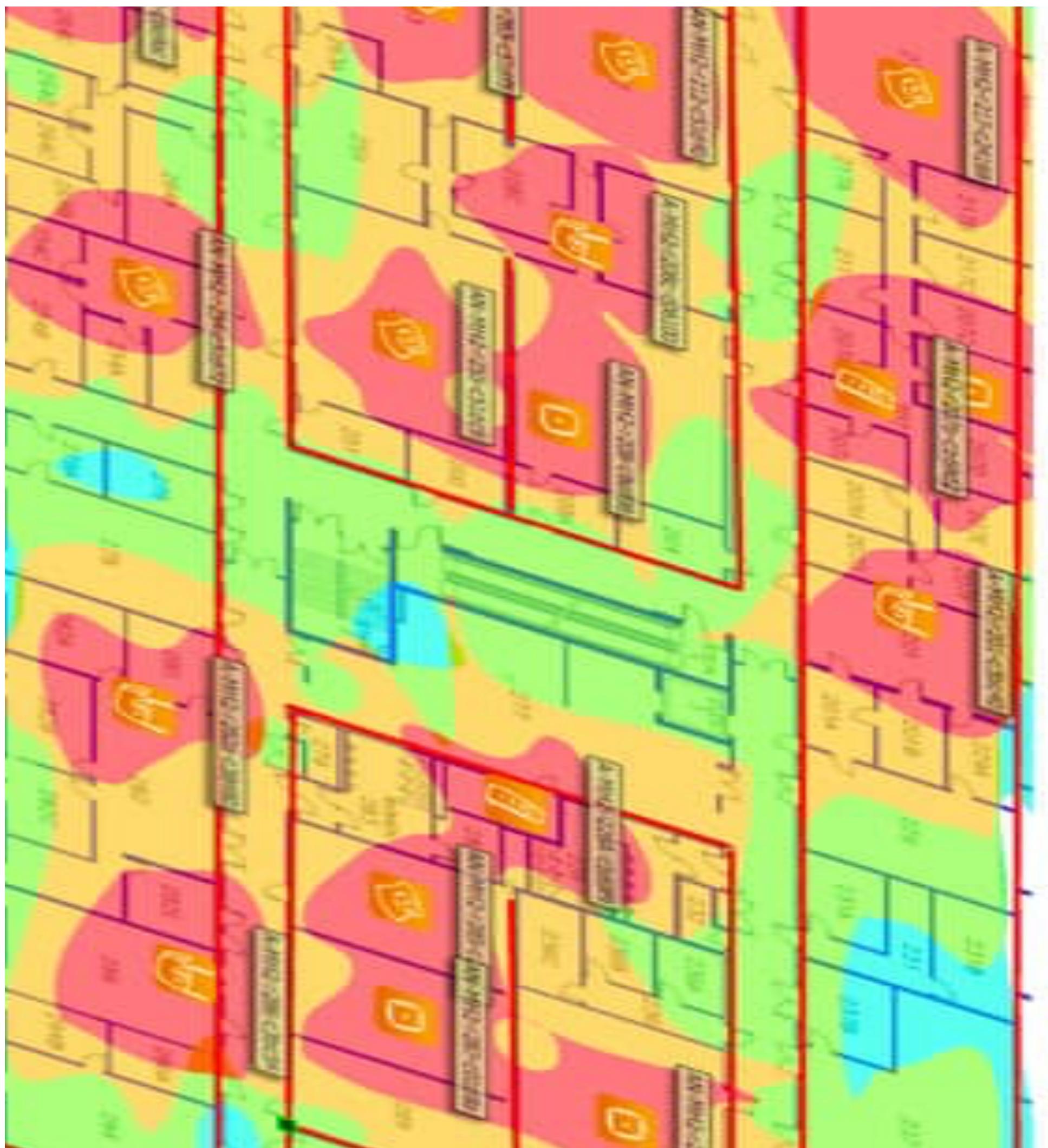
What is a convolution

- Example: you want to know how good is the signal on your cell phone
- You know the signal strength as a function of position $f(x)$
- You know the phone position through the coordinates returned by the GPS
- But the GPS has a Gaussian resolution uncertainty σ . For a given value of x , the GPS returns some x' , distributed according to $G(x - x' | \sigma)$
- To know the signal strength, you need to average across all the values of x , weighted by the probability that for that given value of x the GPS returns x'

The input **The convolution kernel**

$$S(x') = \int dx f(x) G(x' - x) = (f * G)(x')$$

*This is the convolution of f with G
 G has to be a positive defined function (a pdf)*





A discrete convolution

- Our data is a discrete (1D, 2D, ..., nD) array of values
- For a 1D array, need to define the discrete equivalent of the convolutional integral, which trivially is

$$(f * g)(t_i) = \sum_j f(a_j)g(t_i - a_j)$$

- At 2D, for an image I of pixel $I(i, j)$ and a kernel given by some function $K(i, j)$ of the same pixel coordinates (i, j) ,

$$S(i, j) = (I * K)(i, j) = \sum_k \sum_l I(k, l)K(i - k, j - l) = \sum_k \sum_l I(i - k, j - l)K(k, l)$$

You can convince yourself of the last identity defining $k = i - k'$, $l = j - l'$, and then redefining $k' \rightarrow k$ and $l' \rightarrow l$. Notice that for a given point (i, j) the convolution moves towards lower coordinates while one moves fwd in the kernel coordinates

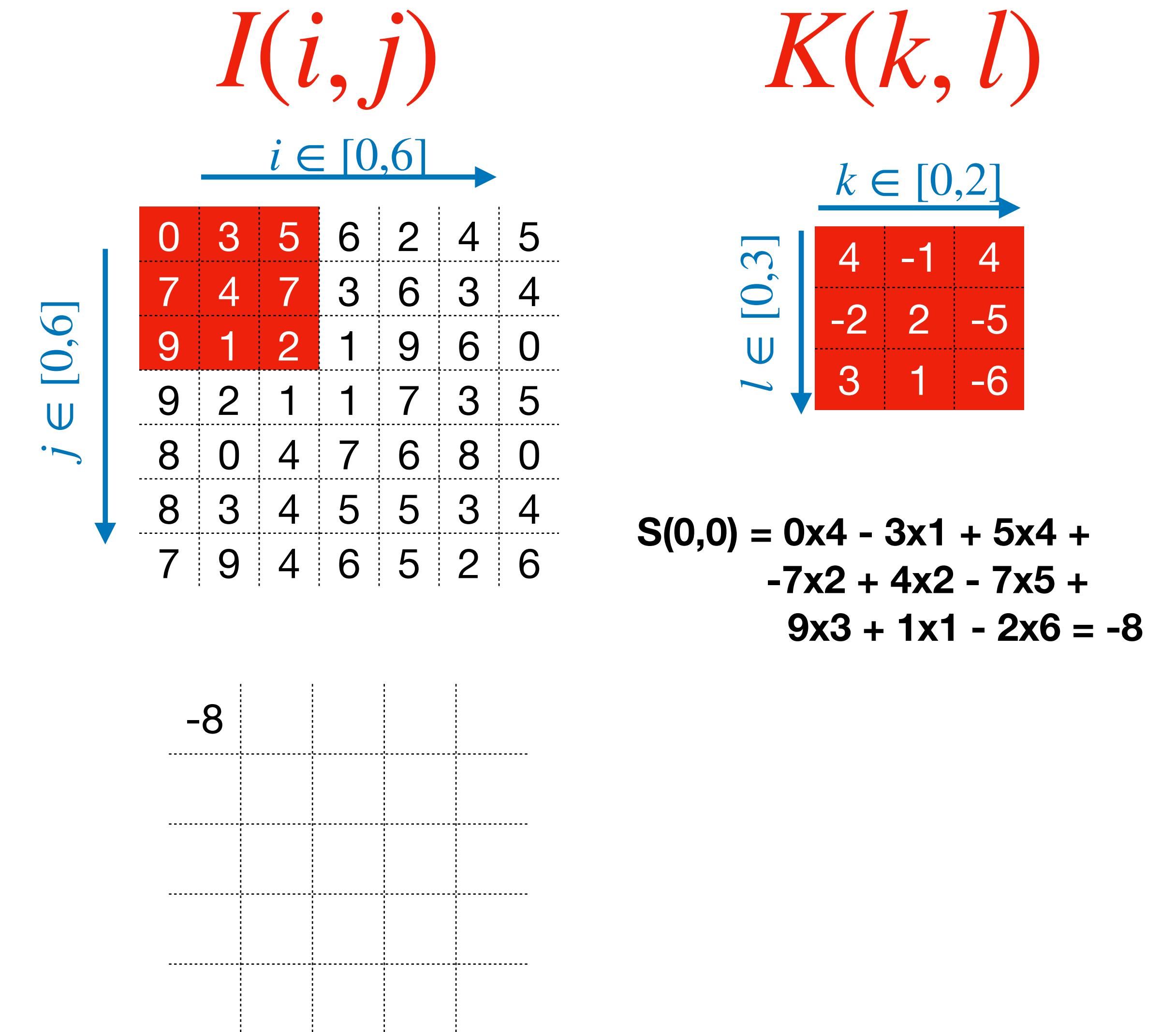
cnn use Cross correlation

- Typically, CNNs don't apply the convolution we just described

- Instead, they apply a similar expression, called cross correlation

$$S(i,j) = (I * K)(i,j) = \sum_k \sum_l I(i+k, j+l)K(k, l)$$

- In this case, one moves coherently across the input image and the kernel



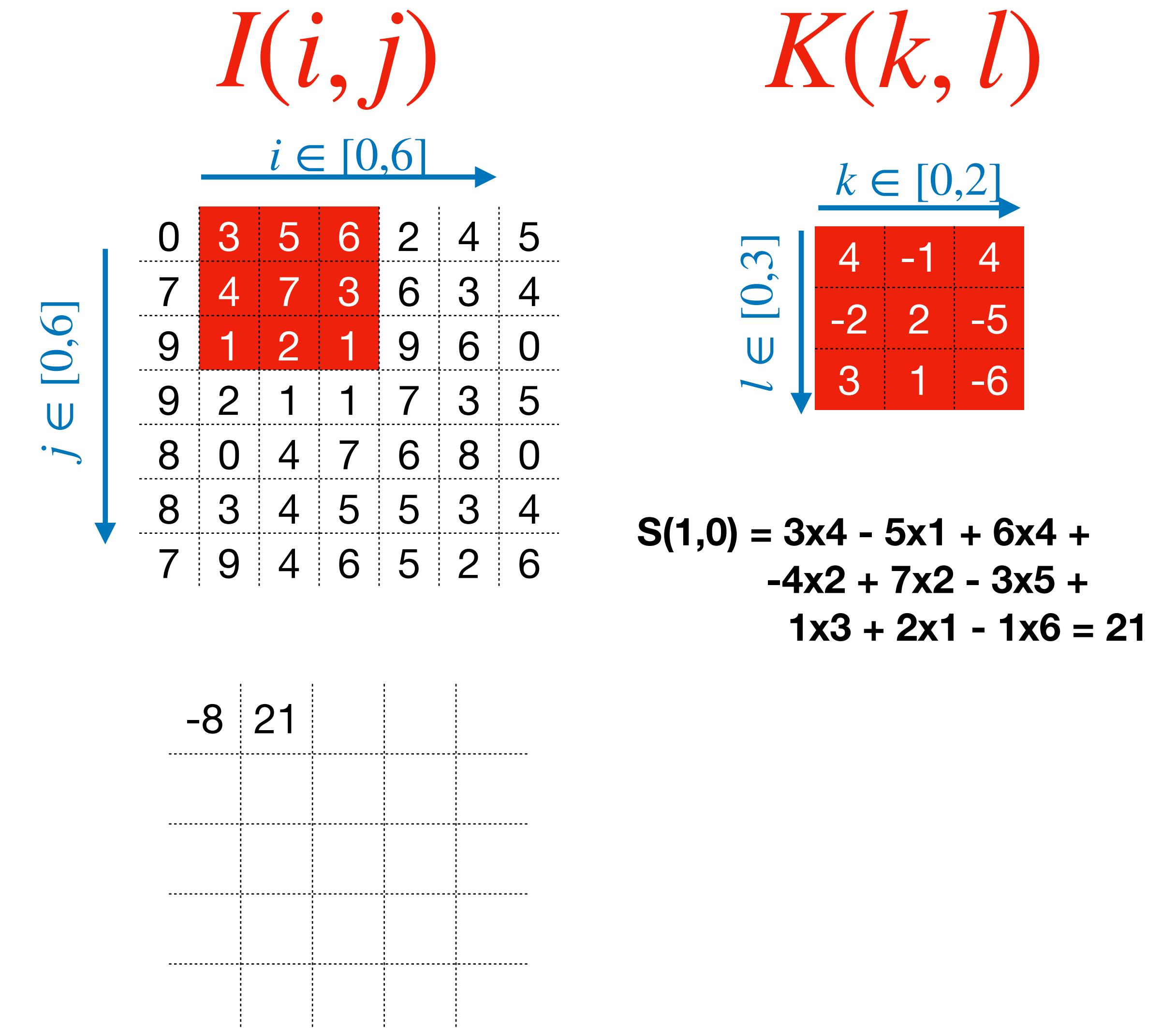
CNN USE CROSS correlation

- Typically, CNNs don't apply the convolution we just described

- Instead, they apply a similar expression, called cross correlation

$$S(i,j) = (I * K)(i,j) = \sum_k \sum_l I(i+k, j+l)K(k, l)$$

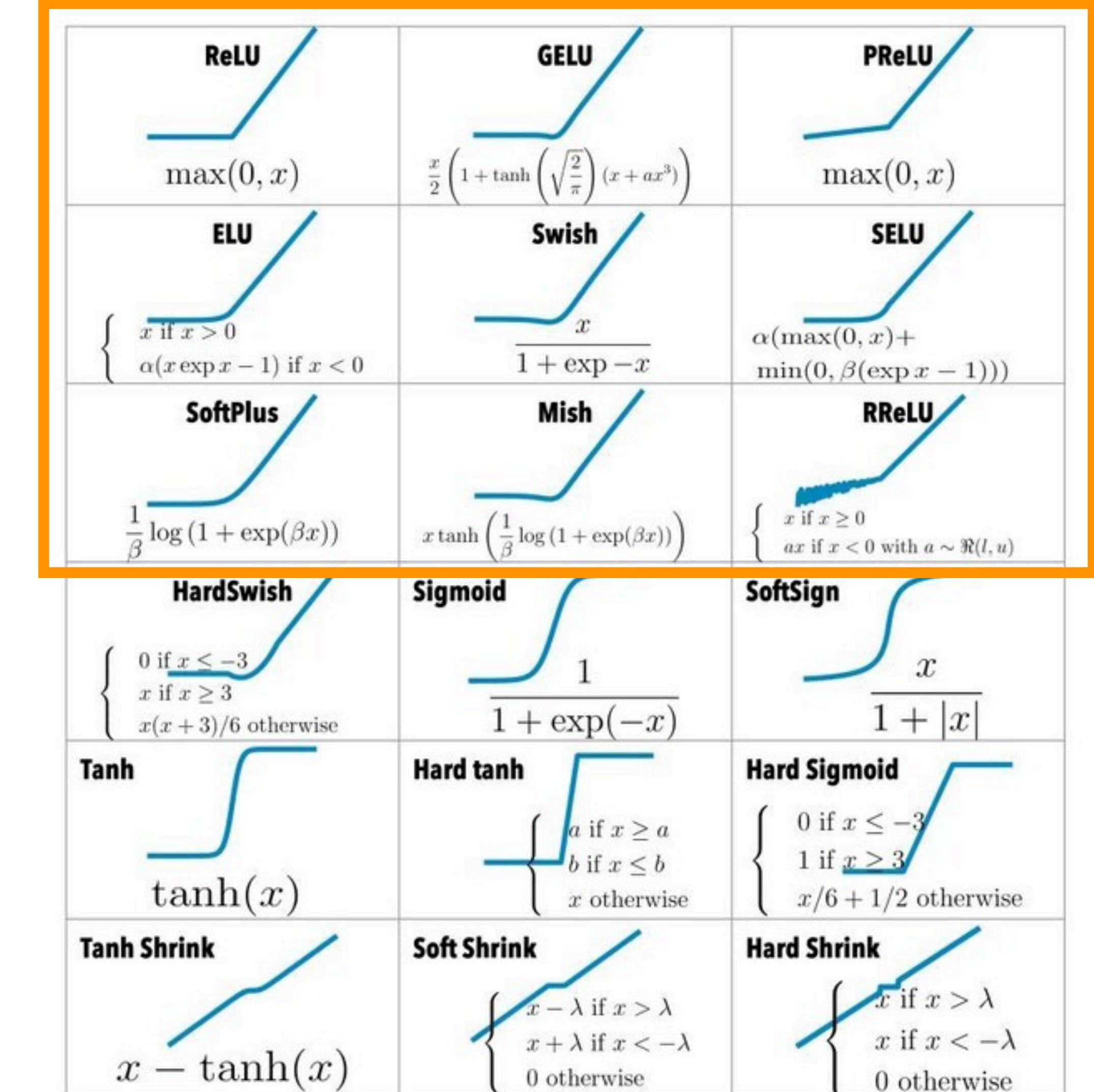
- In this case, one moves coherently across the input image and the kernel



After convolution: activation

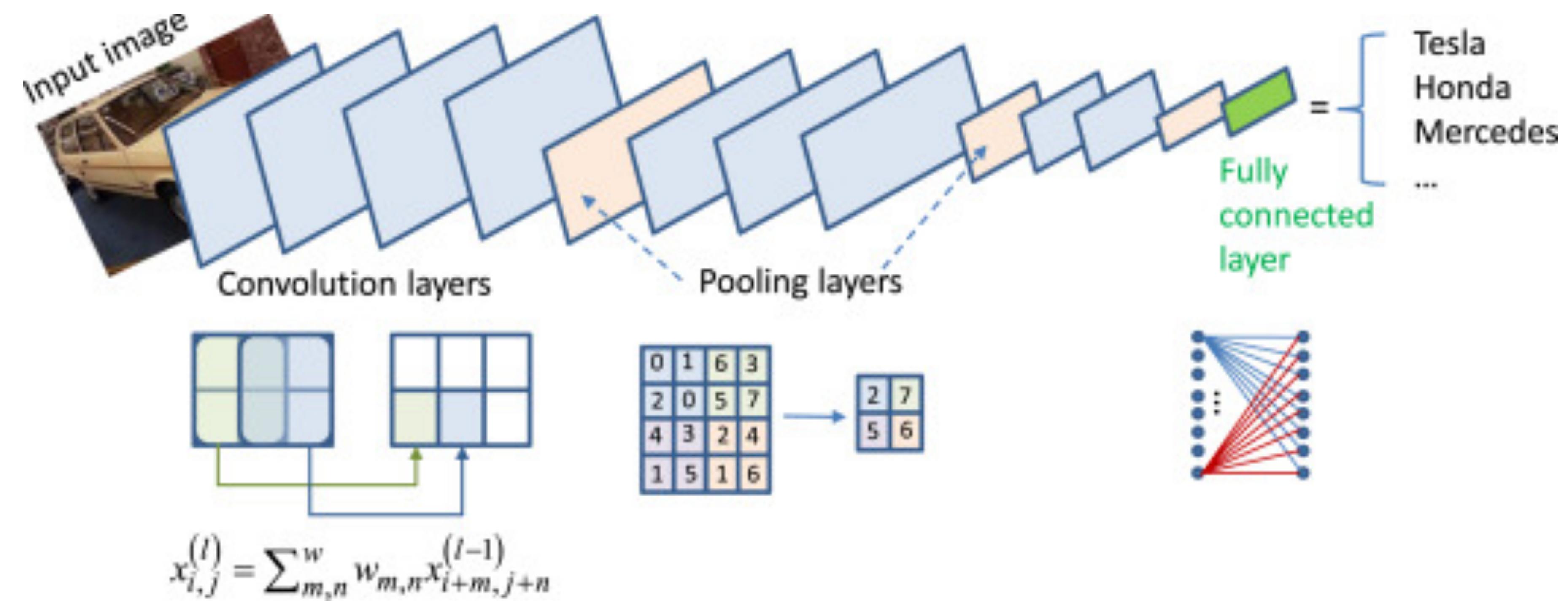
- Activation functions are the essential ingredient to NNs complexity
- Bring non-linearity in NN processing. Crucial to increase complexity
- The choice of the activation function is a hyperparameter
- In practice, the fastest is the function, the more efficient is the learning
- Recent development on functions with fast gradient calculation

Neural Network Activation Functions: a small subset!



Pooling

- A pooling function is a manipulation of an array, in which the content of a pixel is replaced by a function of the nearby pixels
- As a result, the image size is reduced.



Pooling

- *MaxPooling: Given an image and a filter of size $k \times k'$, scans the image and replaces each $k \times k'$ patch with its maximum*

0	3	5	6	2	4	5
7	4	7	3	6	3	4
9	1	2	1	9	6	0
9	2	1	1	7	3	5
8	0	4	7	6	8	0
8	3	4	5	5	3	4
7	9	4	6	5	2	6



9	7	9	9	9
9	7	9	9	9
9	7	7	9	9
9	7	7	8	8
9	9	7	8	8

- *AveragePooling: Given an image and a filter of size $k \times k'$, scans the image and replaces each $k \times k'$ patch with its average*

0	3	5	6	2	4	5
7	4	7	3	6	3	4
9	1	2	1	9	6	0
9	2	1	1	7	3	5
8	0	4	7	6	8	0
8	3	4	5	5	3	4
7	9	4	6	5	2	6



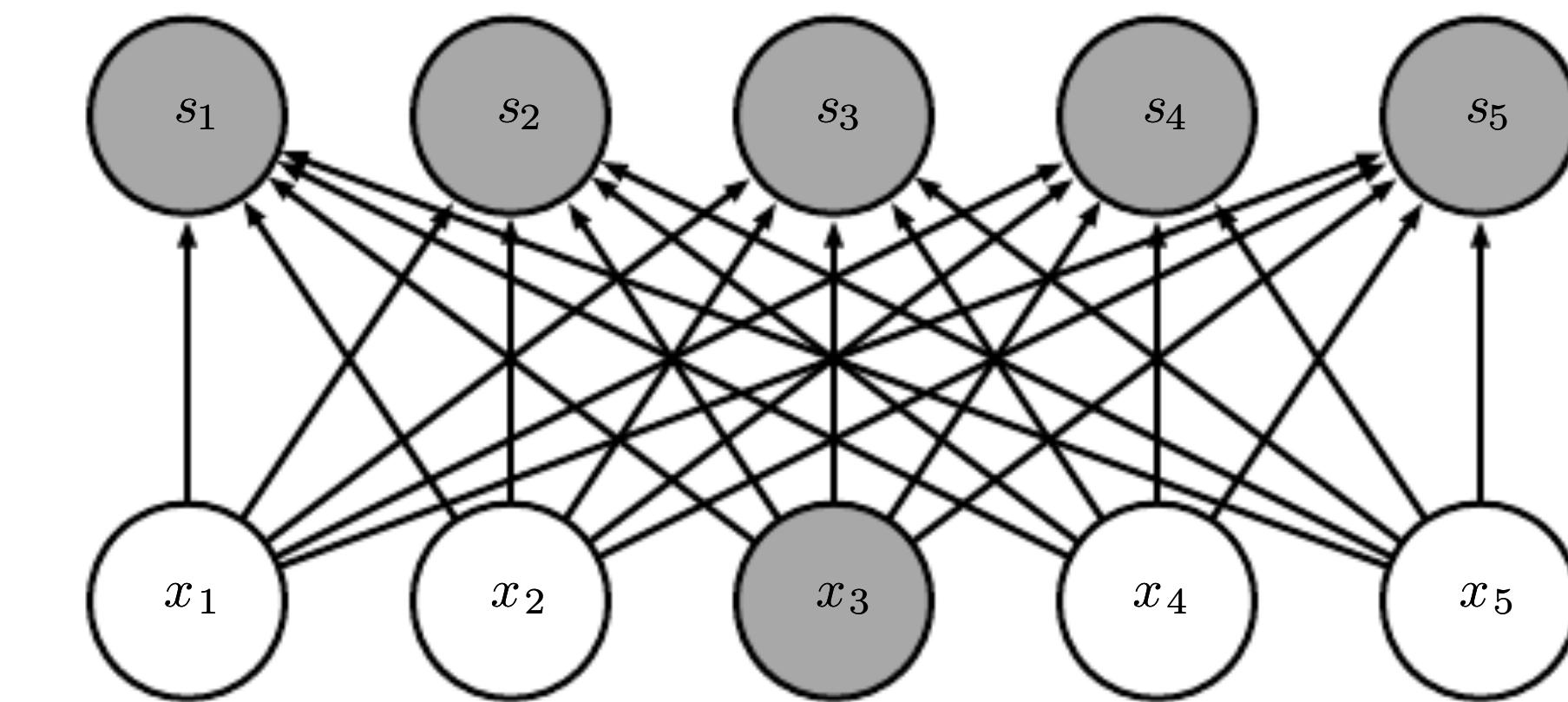
4.2				

			5.0	

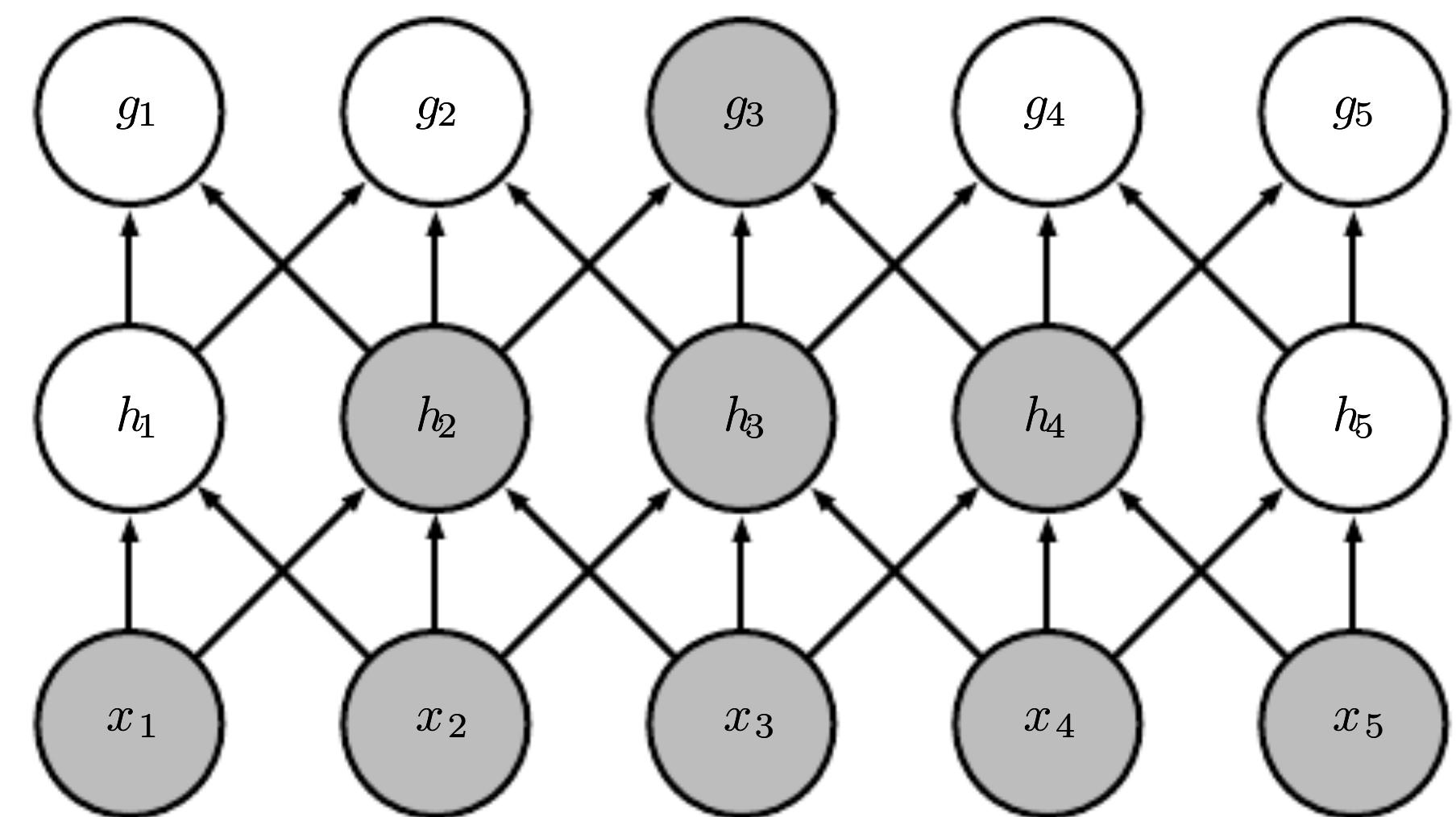
- ...

Why do CNN work?

- Sparse interactions: in a DNN, each output unit interacts with every input unit, increasing the number of parameters. In a CNN instead, the kernel is smaller than the image. A kernel with a few parameters might identify a given feature (an edge) over millions of input pixels.
- less parameters to reach large complexity
- better computational efficiency (e.g., during training)

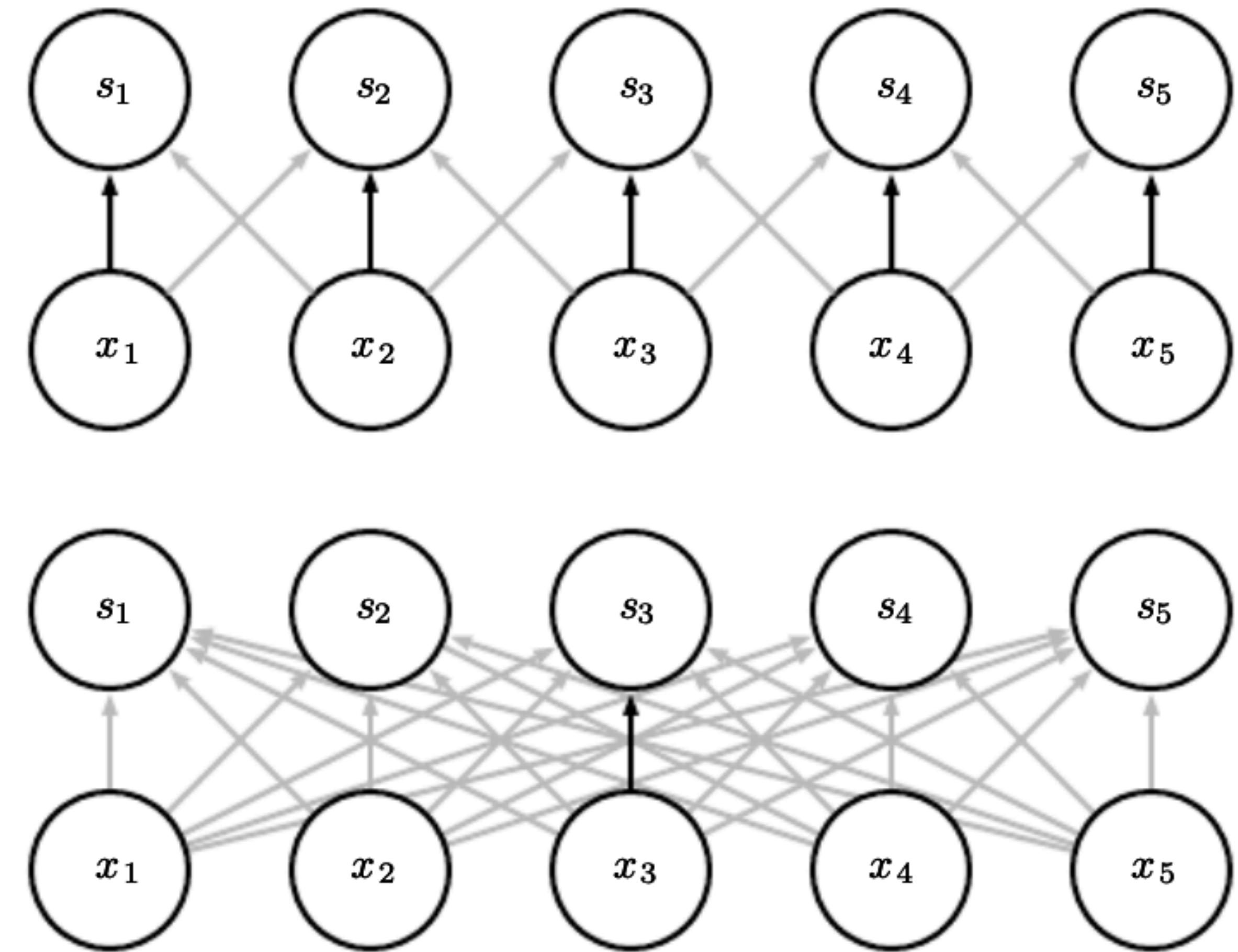


One extra layer but much less connections (i.e., parameters) reach the same receptive field



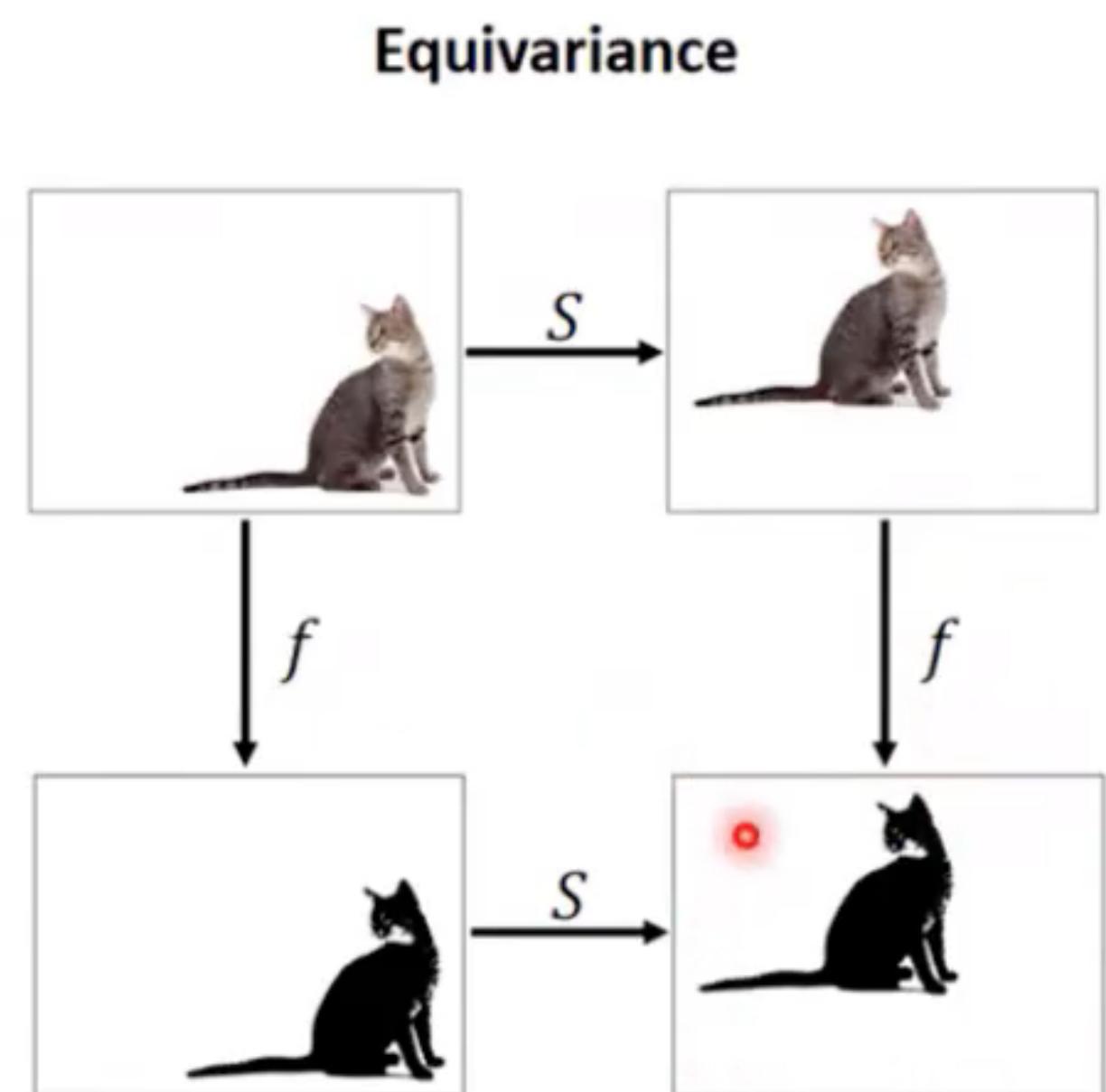
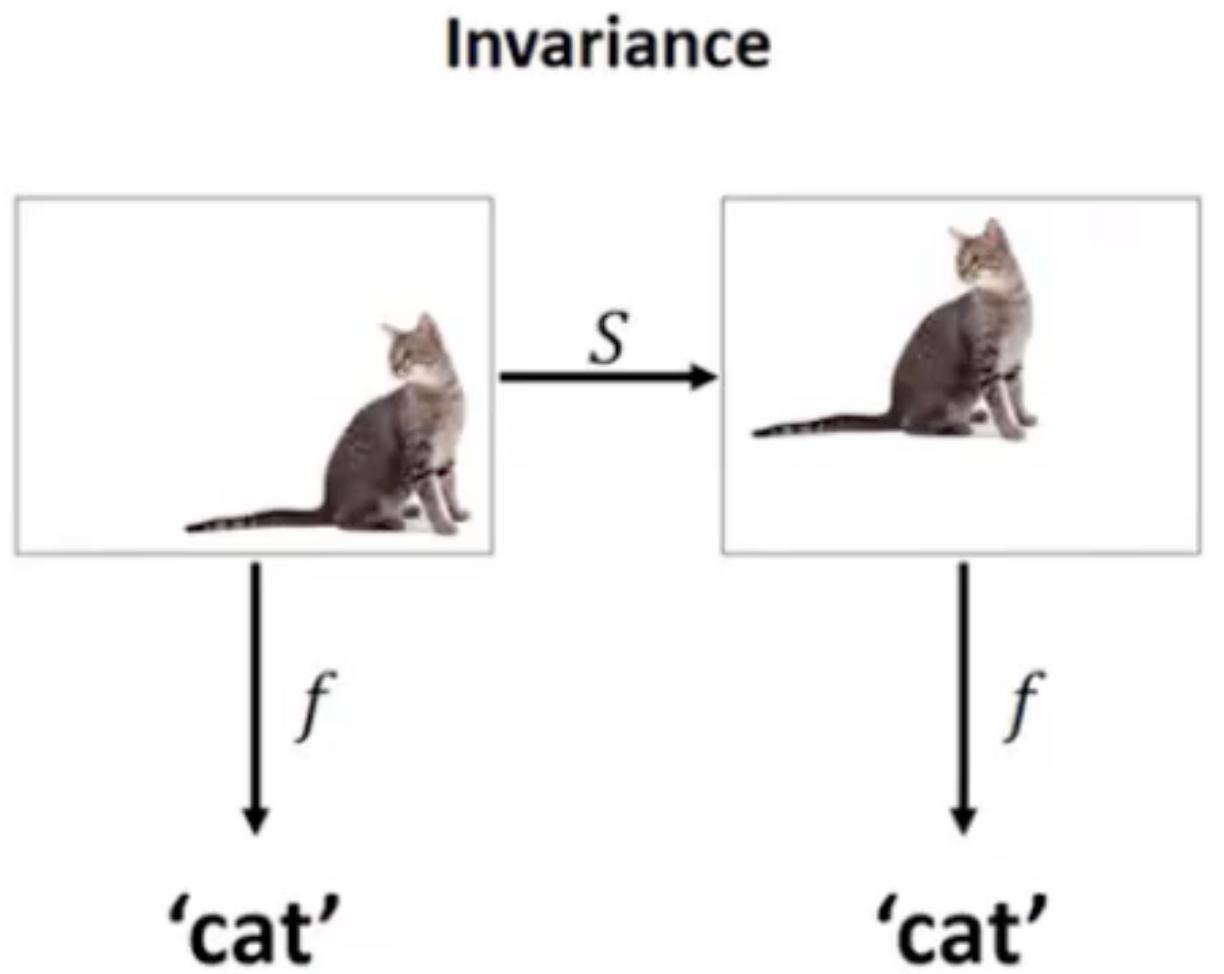
Why do CNN work?

- **Parameter sharing:** In a CNN, the same parameter acts on various parts of the network (dark arrows). In a DNN, each parameter is used once. While DNN parameters learn tasks related to specific inputs, a CNN parameter learns across the full image
- Sharing weights has computational advantages, e.g., less required memory



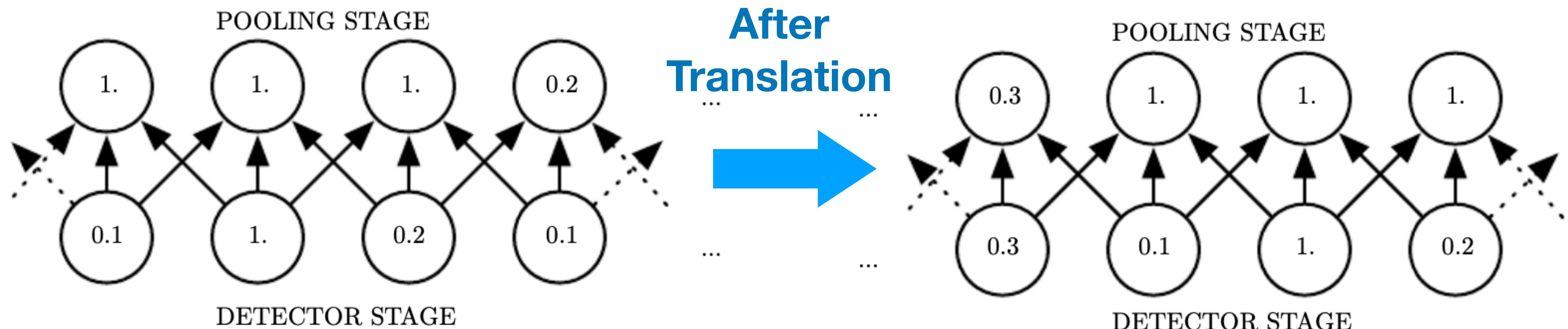
Why do CNN work?

- Equivariant representation: thanks to parameter sharing, CNN are equivariant to translation: if the input is shifted, so is the output
- A function $f(x)$ is equivariant to a function $g(x)$ if $f(g(x)) = g(f(x))$
- For CNN
 - Consider the operation $S(i) = i - 1$ that shifts the each pixel to the right (it moves the pixel $i - 1$ to position i)
 - (Modulo border effects), for an image I it is true that $S(f(I)) = f(S(I))$
- Having architectures equivariant to the problem symmetries make the learning much faster
 - If a DNN has to learn that to give the same answer for I and $f(I)$, it needs a much larger dataset (a dataset augmented adding all the $S(i)$ images), which implies a longer training process
 - This is built in the CNN architecture, which then arrives faster to the loss minimum



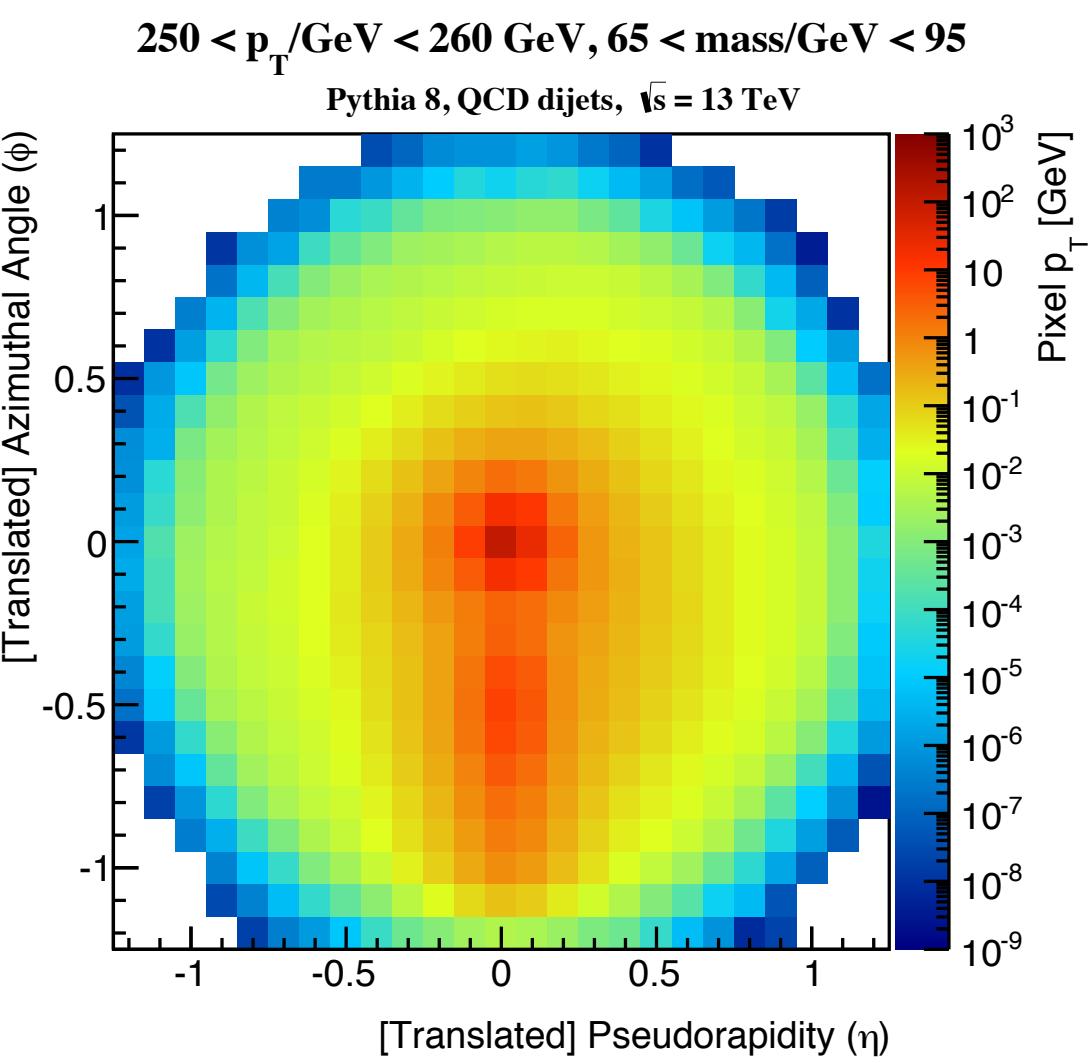
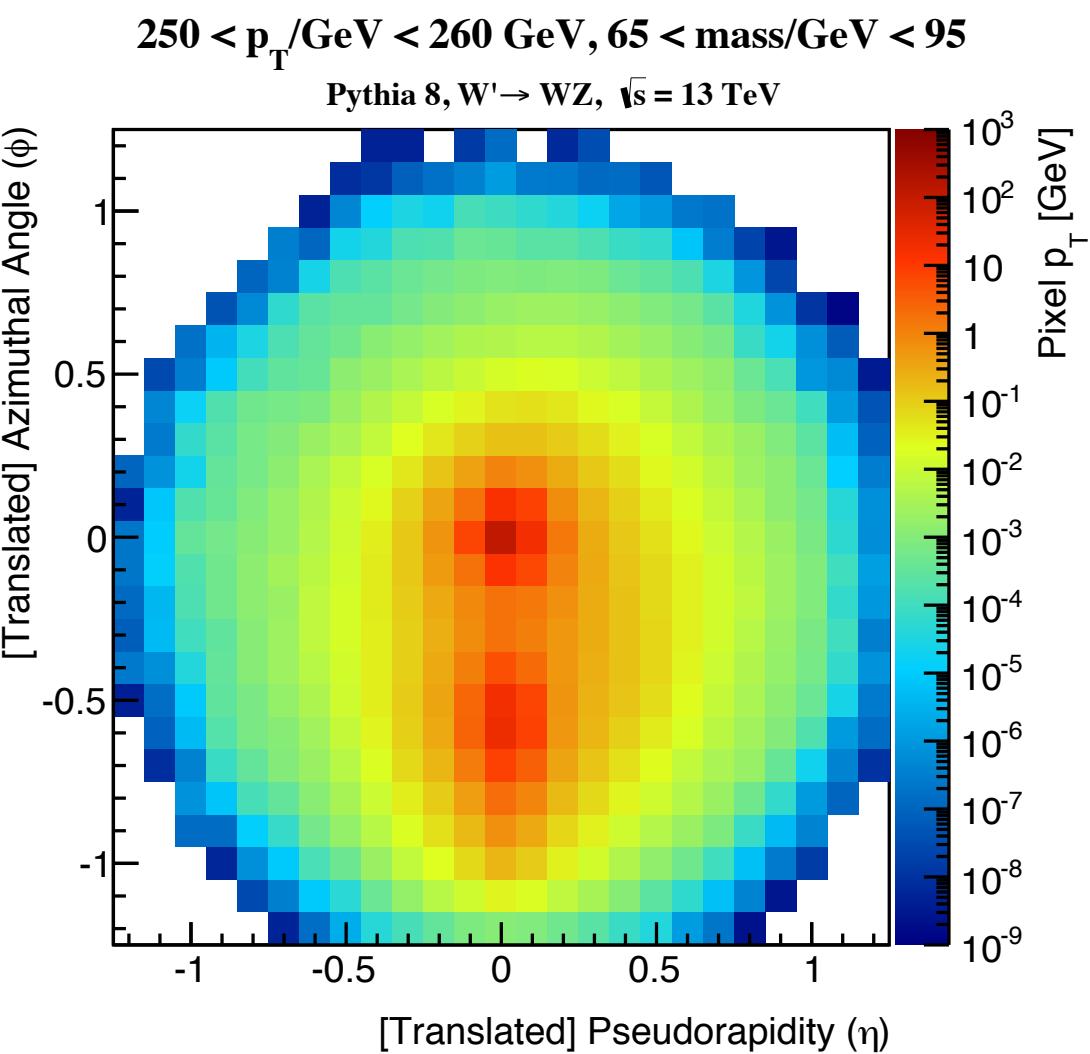
Approximate Invariance

- With pooling, network equivariant is promoted to approximate invariance: the output is about the same for small translations
- See the example of MaxPooling
- This is imposing a strong prior via architecture choice to the learning process



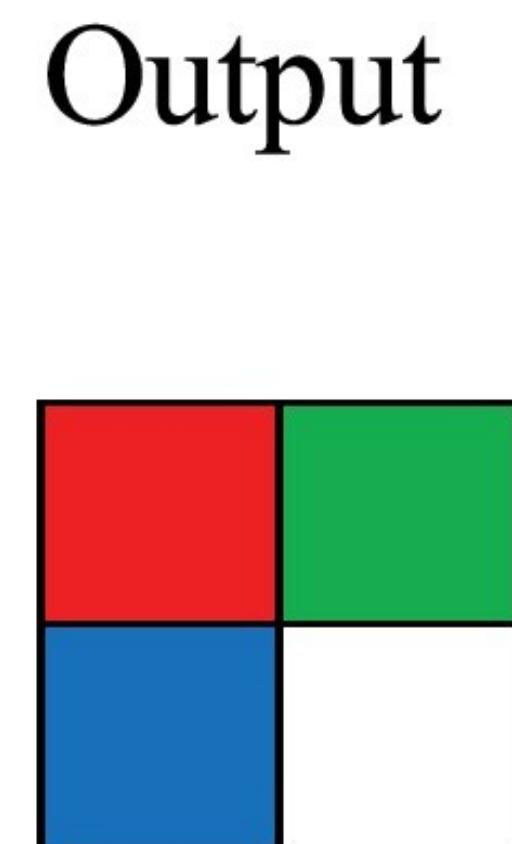
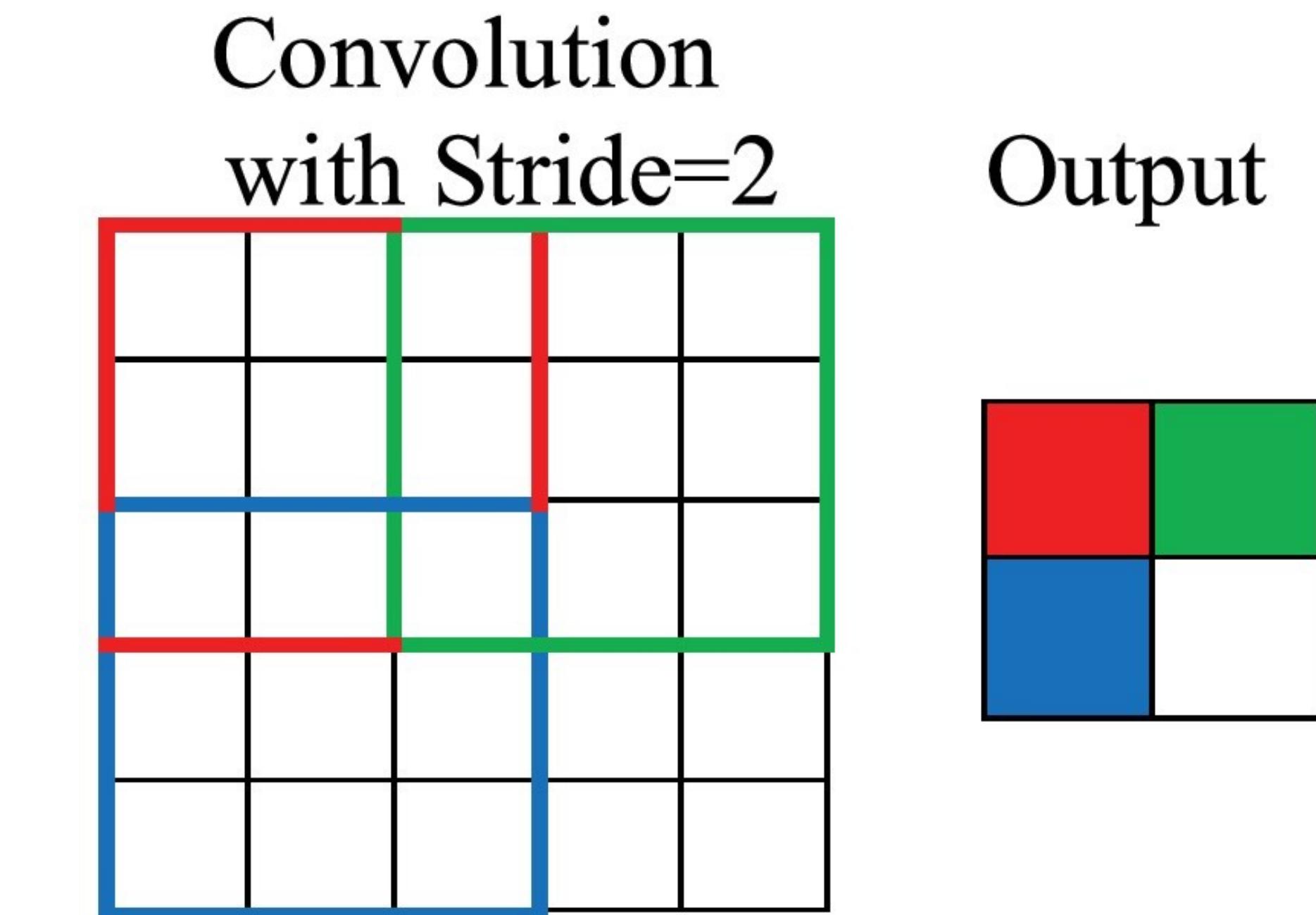
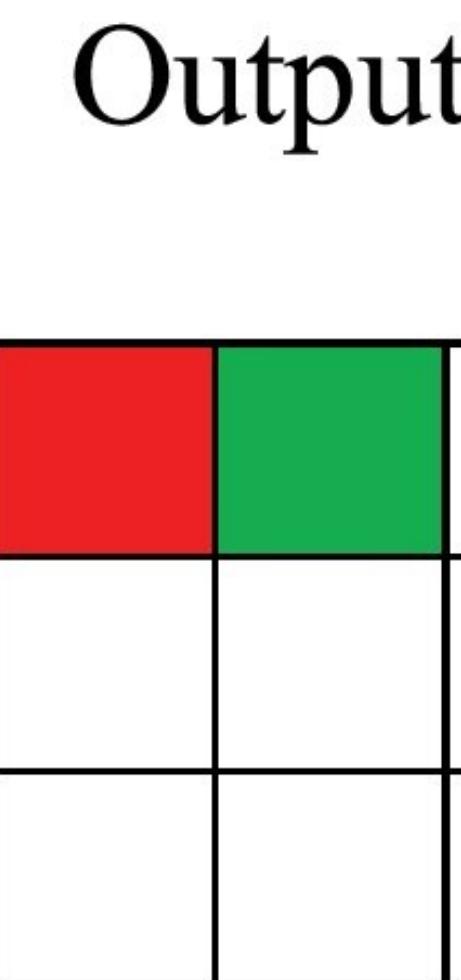
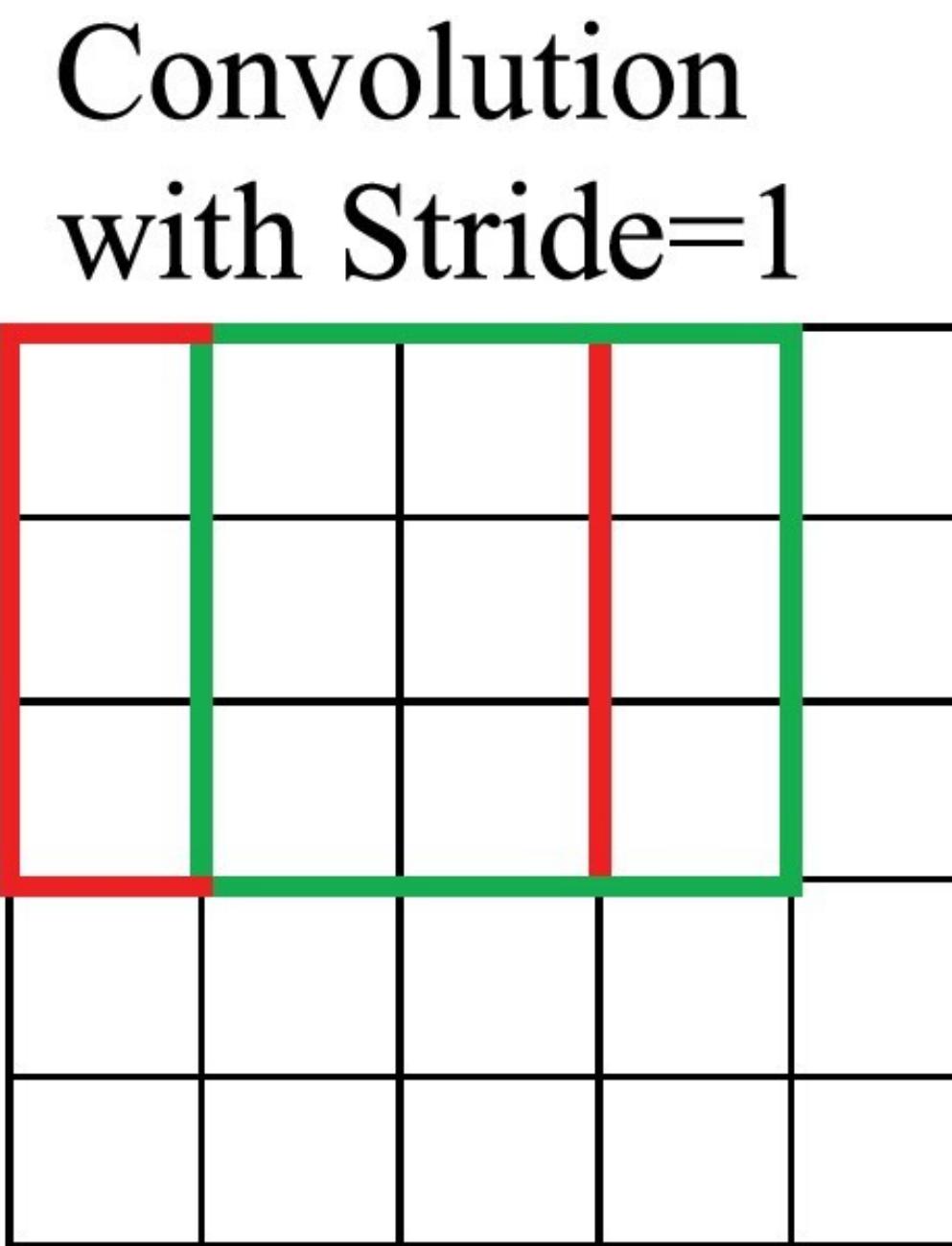
Pooling Choice

- A strong prior could be bad if it goes against the problem nature
- Example 1: try to estimate the energy of a jet from its image
- Max pooling implies informations loss
- Average pooling makes more sense
- Example 2: try to count how many sub-jets in the jet
- Max pooling might work better than average pooling



Other parameters: Stride

- When applying convolution or padding, one can specify a *stride*
- by how many pixels the kernel moves



Other parameters: Padding

- When the filter arrived at the edge, it might exceeds it (if n/k is not an integer)
- In this case, a padding rule needs to be specified
- Valid:** stop at the image boundary
- Same:** repeat the values at the boundary
- Zero:** fill the extra columns with zeros

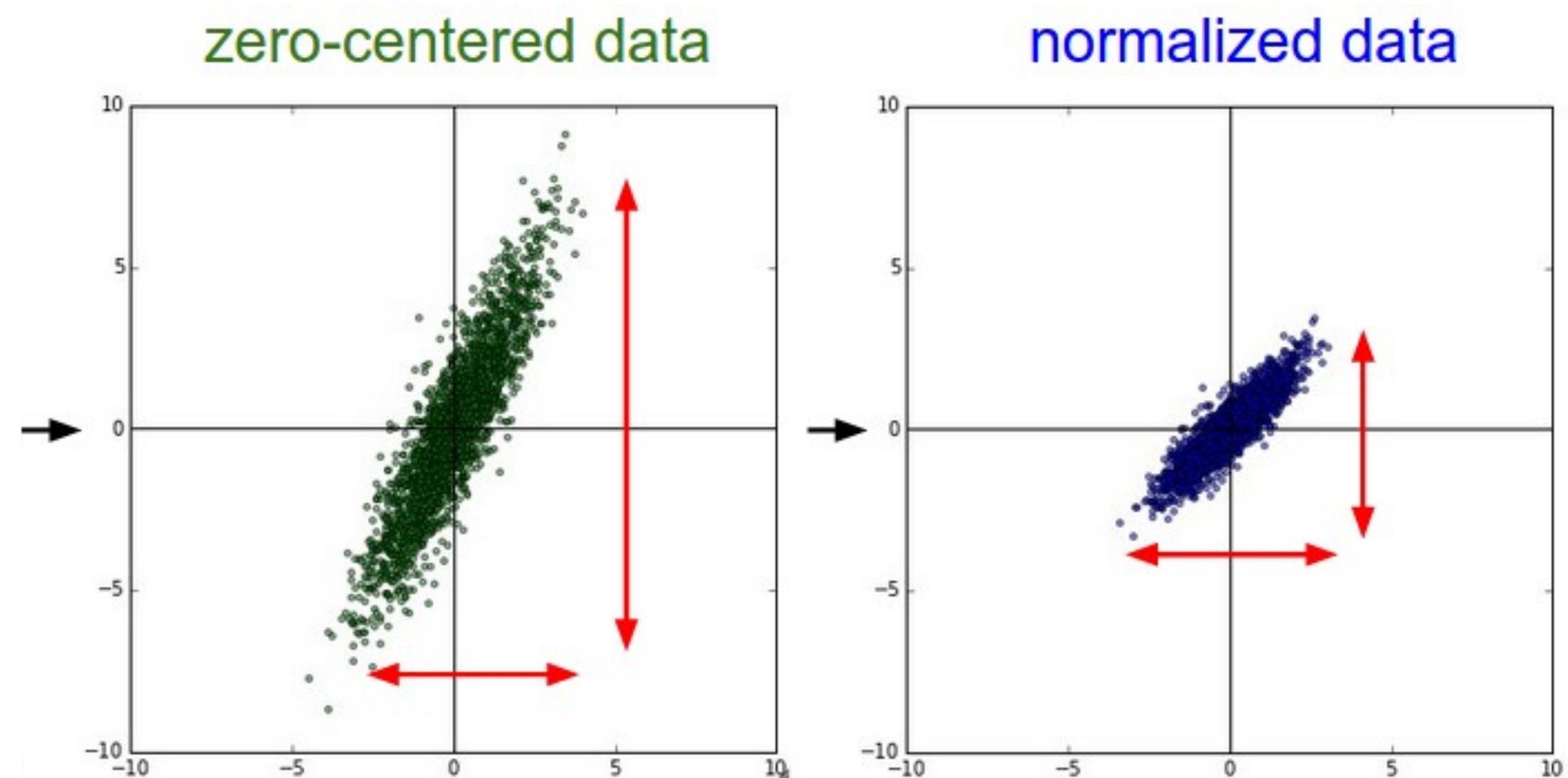
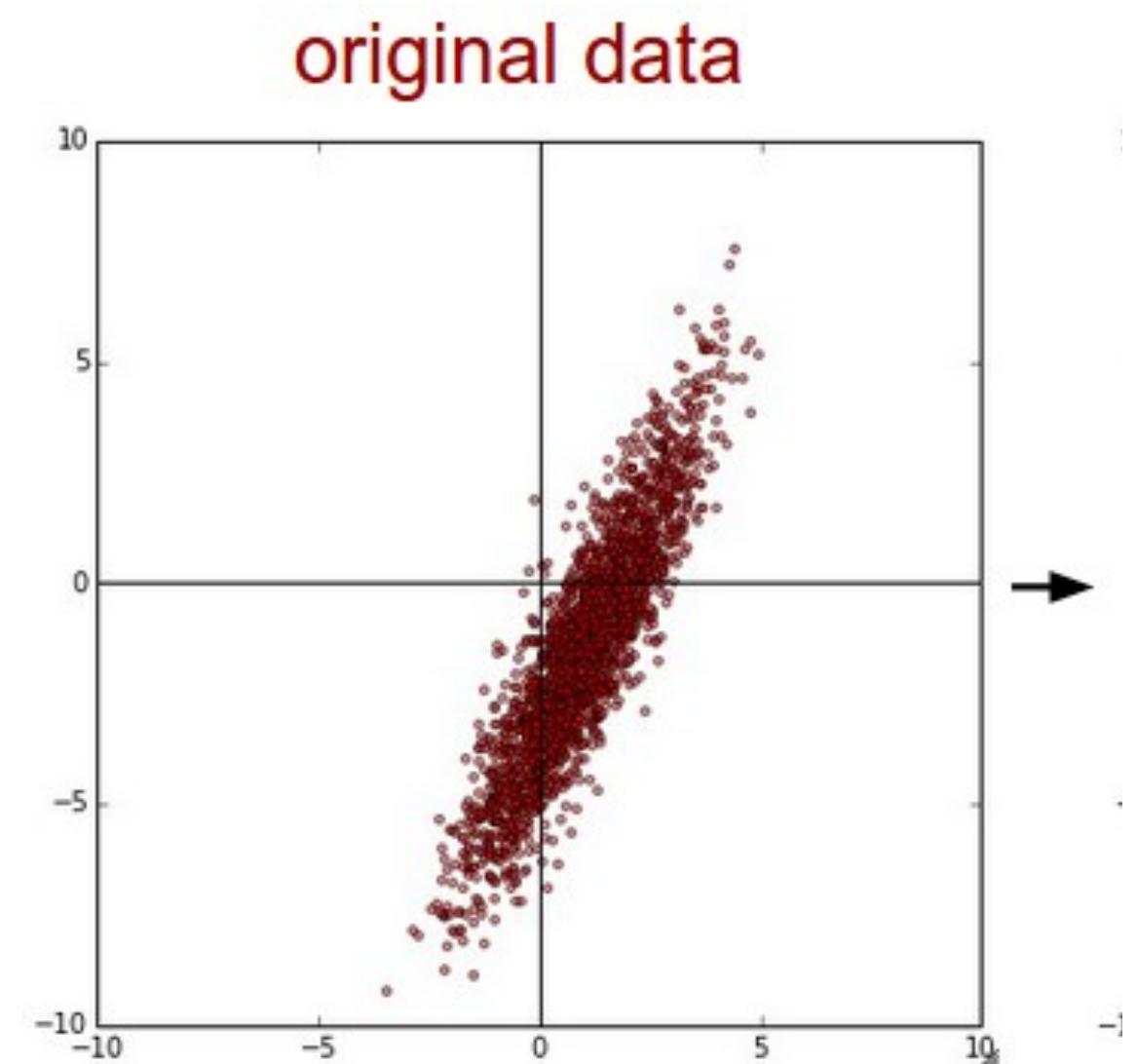
0	3	5	6	2	4	4
7	4	7	3	6	3	3
9	1	2	1	9	6	6
9	2	1	1	7	3	3
8	0	4	7	6	8	8
8	3	4	5	5	3	3
8	3	4	5	5	3	3

0	3	5	6	2	4
7	4	7	3	6	3
9	1	2	1	9	6
9	2	1	1	7	3
8	0	4	7	6	8
8	3	4	5	5	3

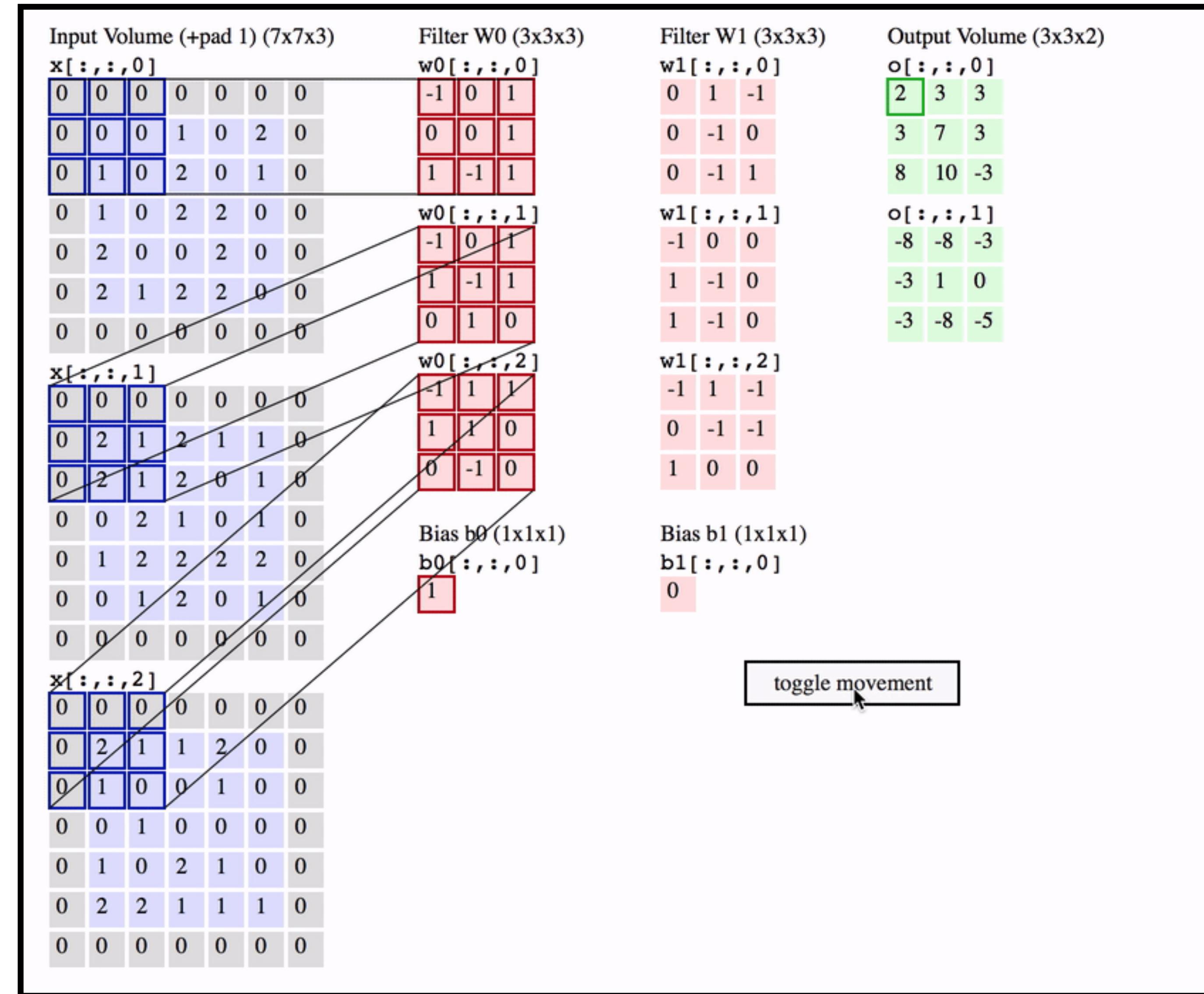
0	3	5	6	2	4	0
7	4	7	3	6	3	0
9	1	2	1	9	6	0
9	2	1	1	7	3	0
8	0	4	7	6	8	0
8	3	4	5	5	3	0
0	0	0	0	0	0	0

Batch Normalization

- It is good practice to give normalized inputs to a layer
- With all inputs having the same order of magnitude, all weights are equally important in the gradient
- Prevents explosion of the loss function
- This can be done automatically with BatchNormalization
- non-learnable shift and scale parameters, adjusted batch by batch

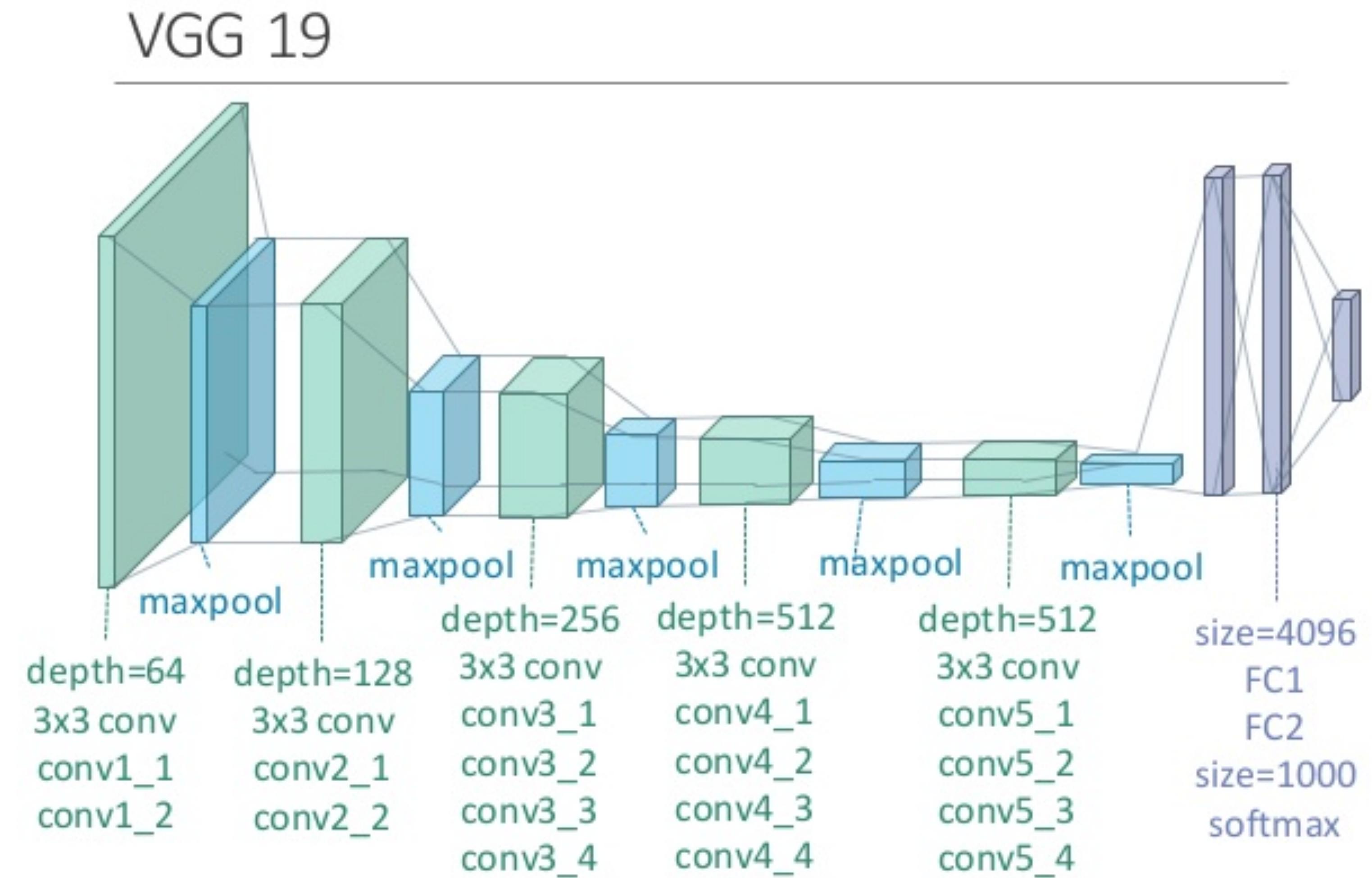


What it looks like...



The full network

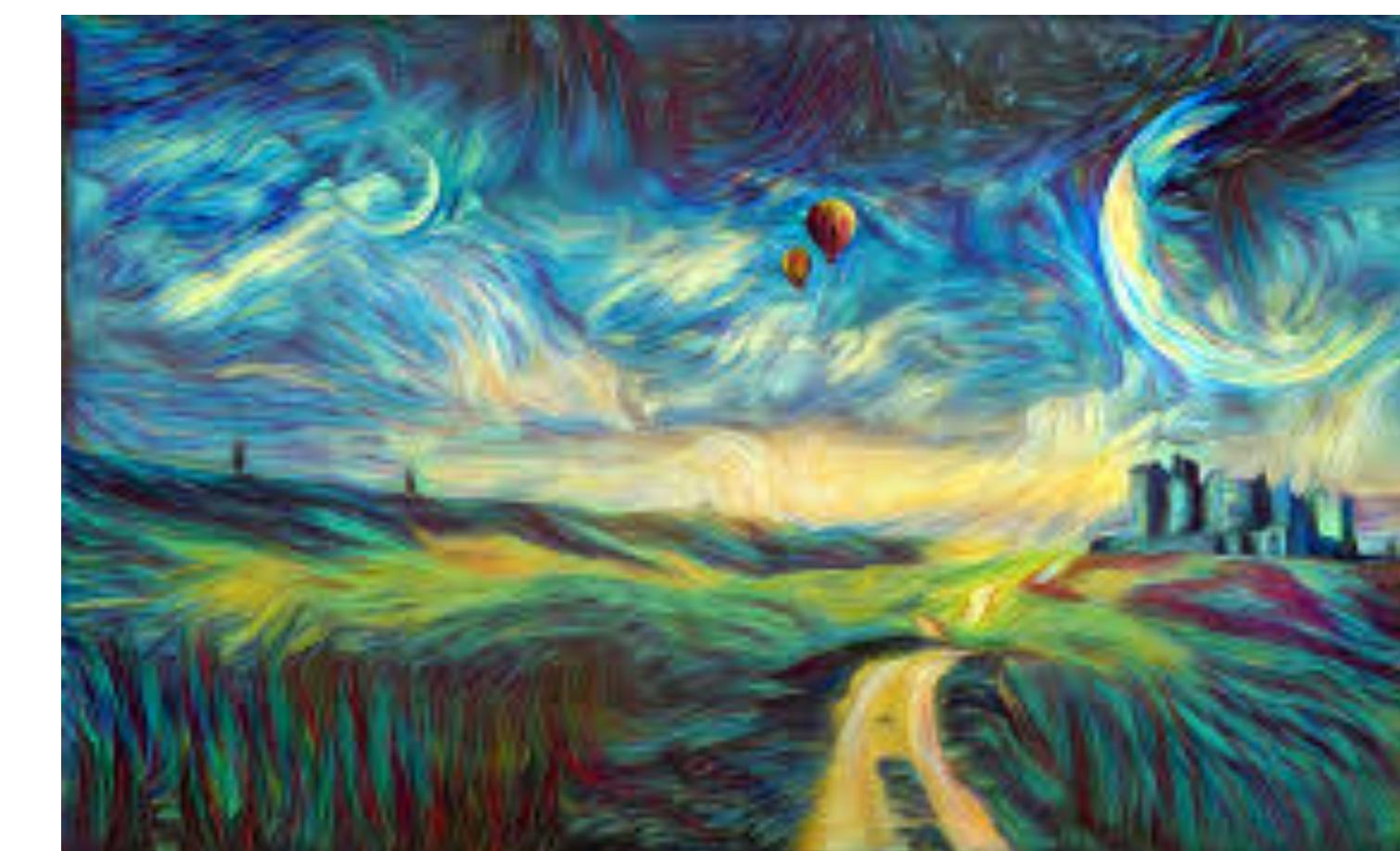
- A full ConvNN is a sequence of Con2D+Pooling (+BatchNormalization+Dropout) layers
- The Conv+Pooling layer reduces the 2D image representation
- The use of multiple filters on the image make the output grow on a third dimension
- Eventually, flattening occurs and the result is given to a dense layer



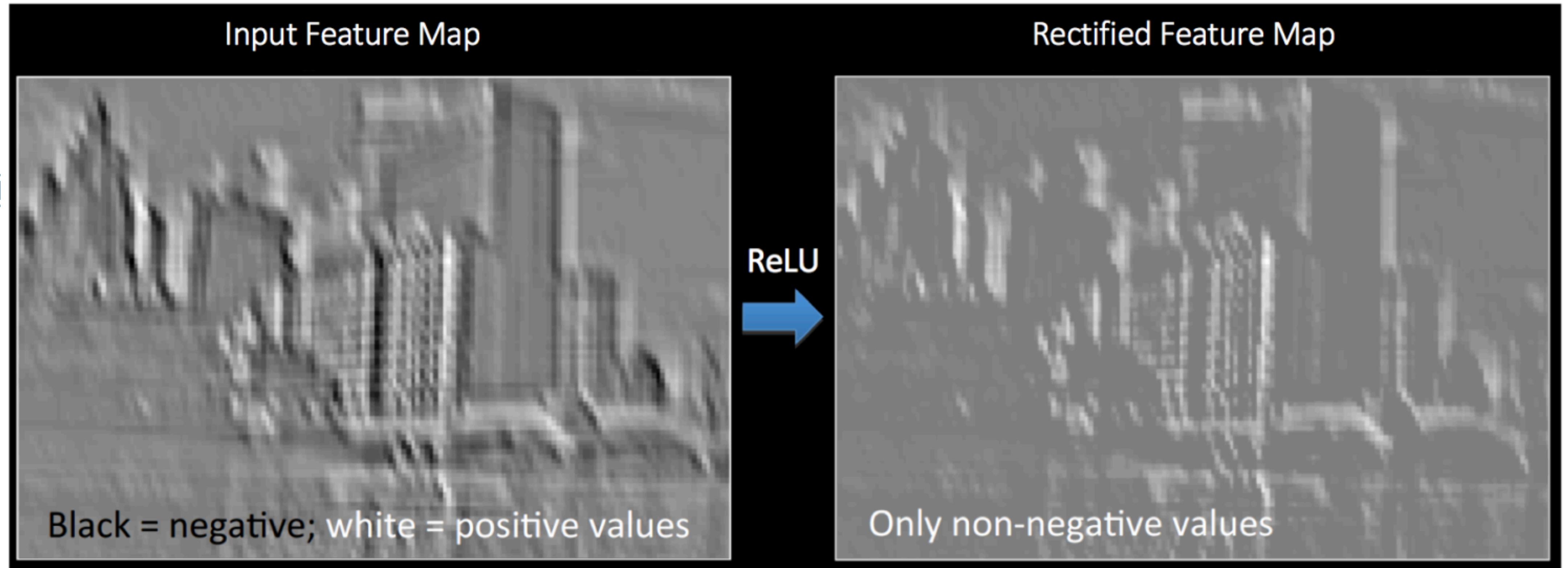
What does a ConvNN learn?

- Each filter alters the image in a different way, picking up different aspects of the image
- edges oriented in various ways
- enhancing / blur of certain features
- It is interesting to check what each filter is doing (and to produce DeepDreams)

Operation	Filter	Convolved Image	Operation	Filter	Convolved Image
Identity	$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$		Sharpen	$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$	
	$\begin{bmatrix} 1 & 0 & -1 \\ 0 & 0 & 0 \\ -1 & 0 & 1 \end{bmatrix}$			$\frac{1}{9} \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$	
	$\begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$			$\frac{1}{16} \begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$	
	$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$				

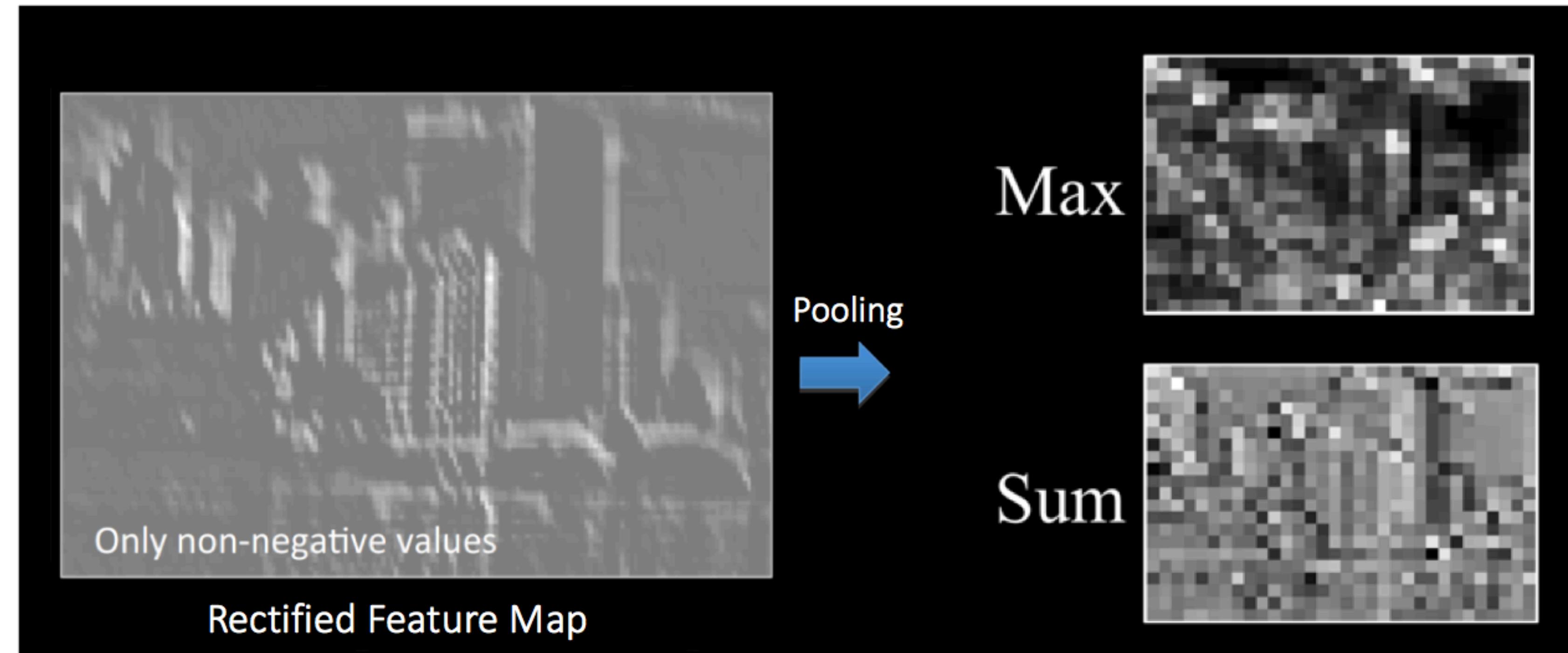


What does a ConvNN learn?



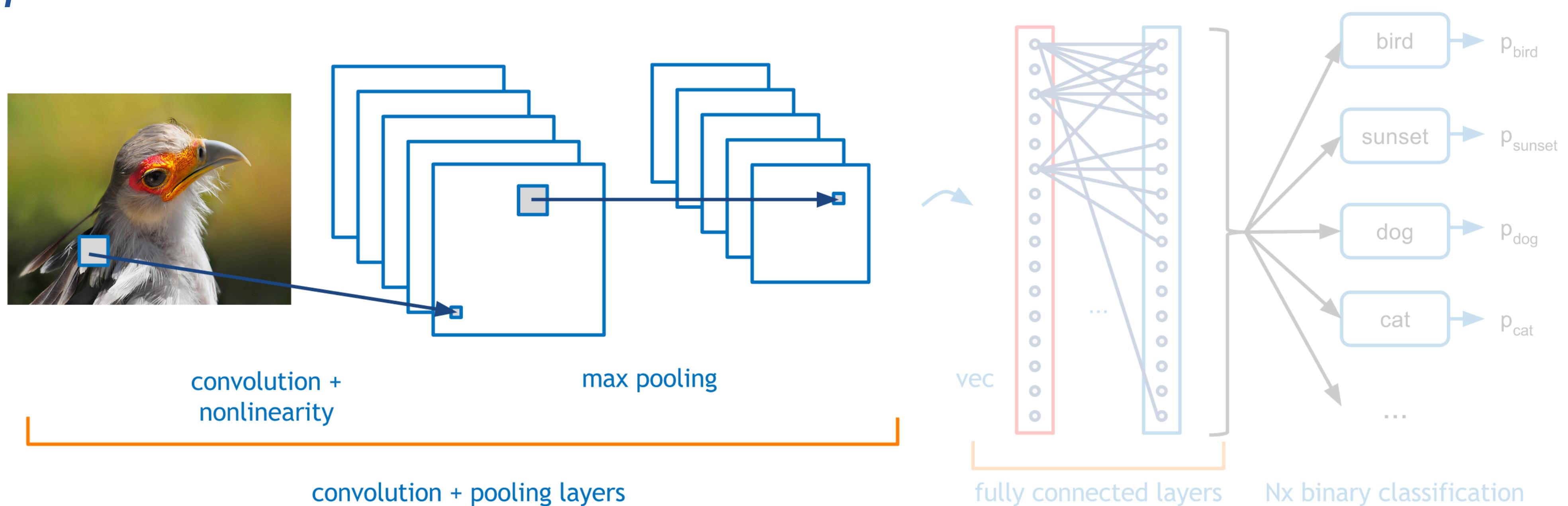
What does a ConvNN learn?

- *Pooling is important in smoothing out the image*
- *reduce parameters downstream (and prevent overfitting)*
- *makes processing independent to local features (distortion, translation)*
- *yields a scale invariant representation of the image (which could be good or bad)*



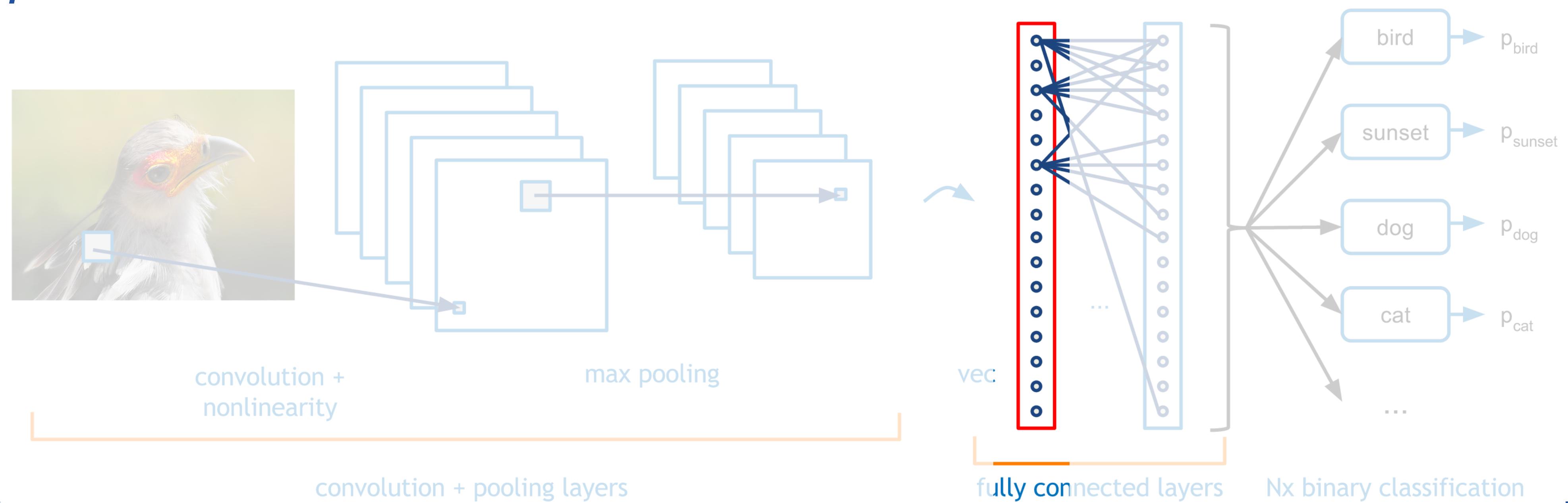
Two tasks in one network

- *The Conv layer starts from RAW data and defines interesting quantities (high level features)*
- *The HLFs replace the physics-motivated inputs of a DNN*
- *The DNN at the end exploits the engineered features to accomplish the task*



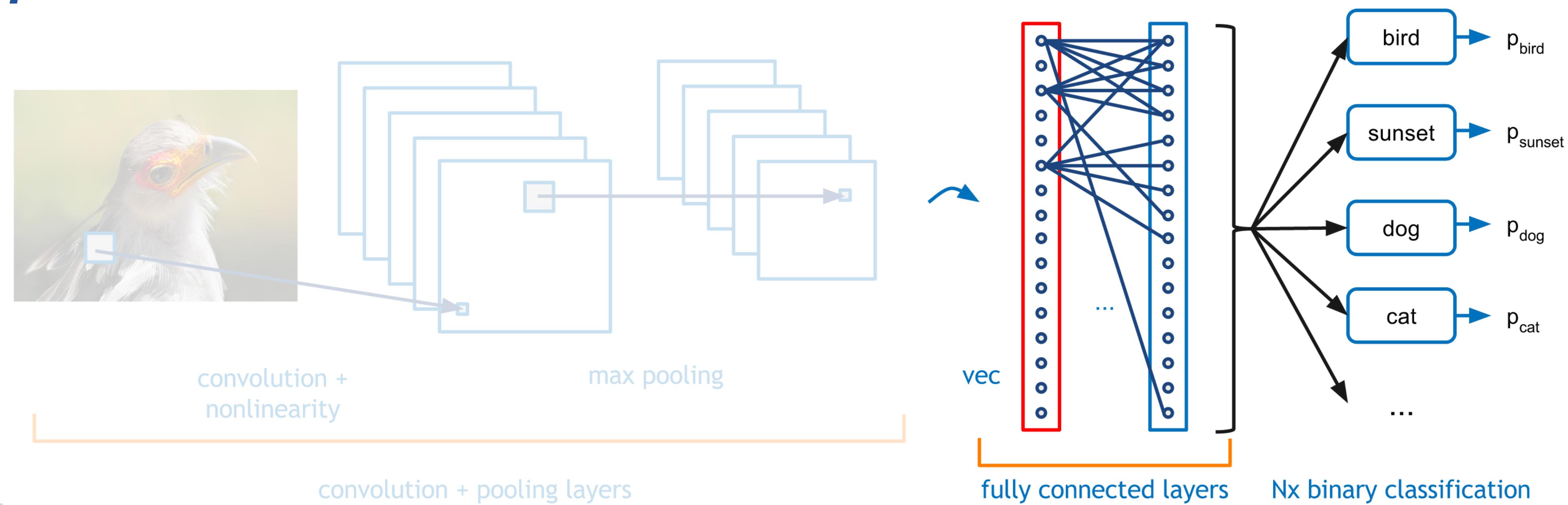
Two tasks in one network

- The Conv layer starts from RAW data and defines interesting quantities (high level features)
- The HLFs replace the physics-motivated inputs of a DNN
- The DNN at the end exploits the engineered features to accomplish the task



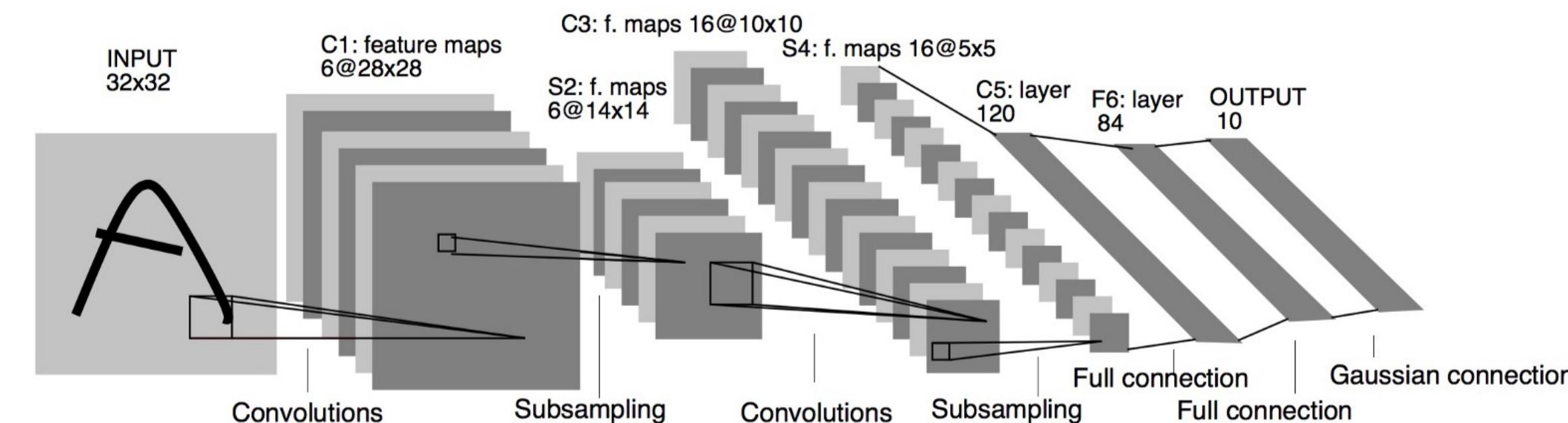
Two tasks in one network

- The Conv layer starts from RAW data and defines interesting quantities (high level features)
- The HLFs replace the physics-motivated inputs of a DNN
- The DNN at the end exploits the engineered features to accomplish the task



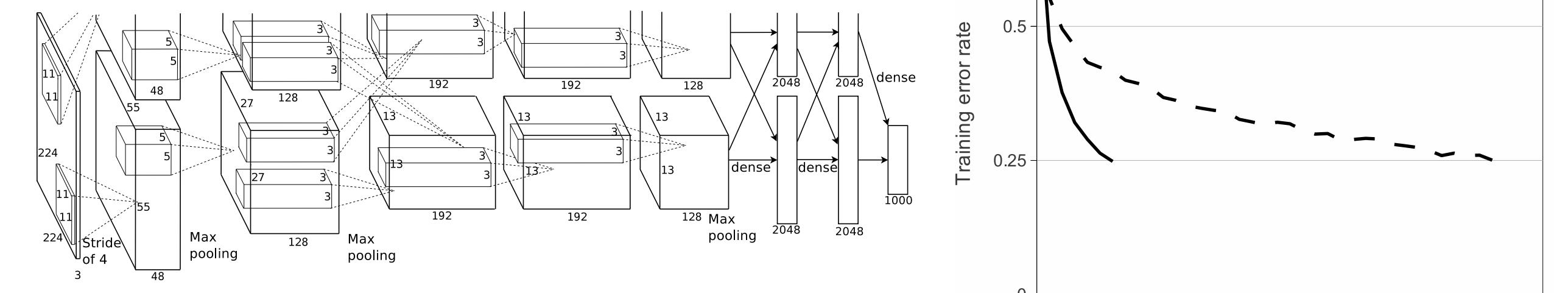
A history of ConvNns

- Neocognitron (1980): translation-invariant image processing with NNs

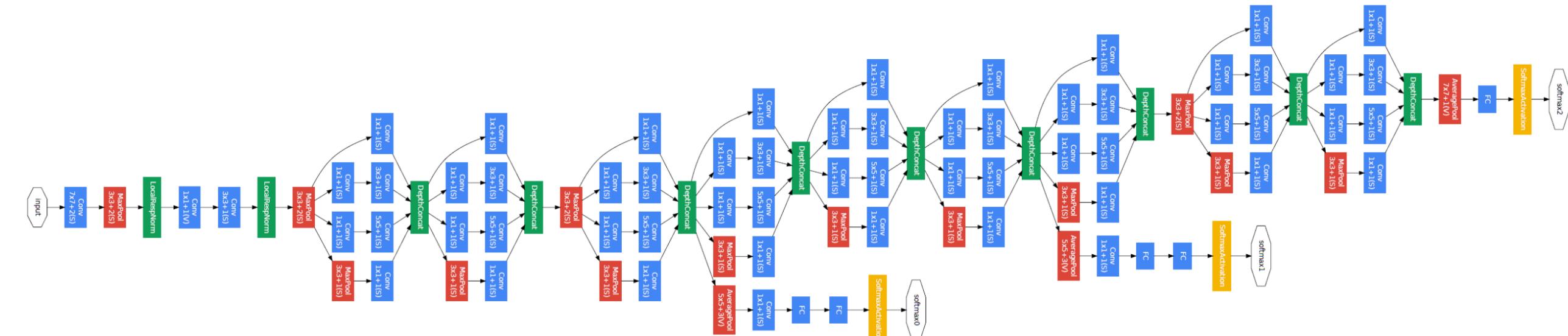


- LeNet (1989): considered the very first ConvNN, designer for digit recognition (ZIP codes)

- AlexNet (2012): the first big ConvNN (60M parameters, 650K nodes), setting the state of the art: trained on GPUs, using ReLU and Dropout

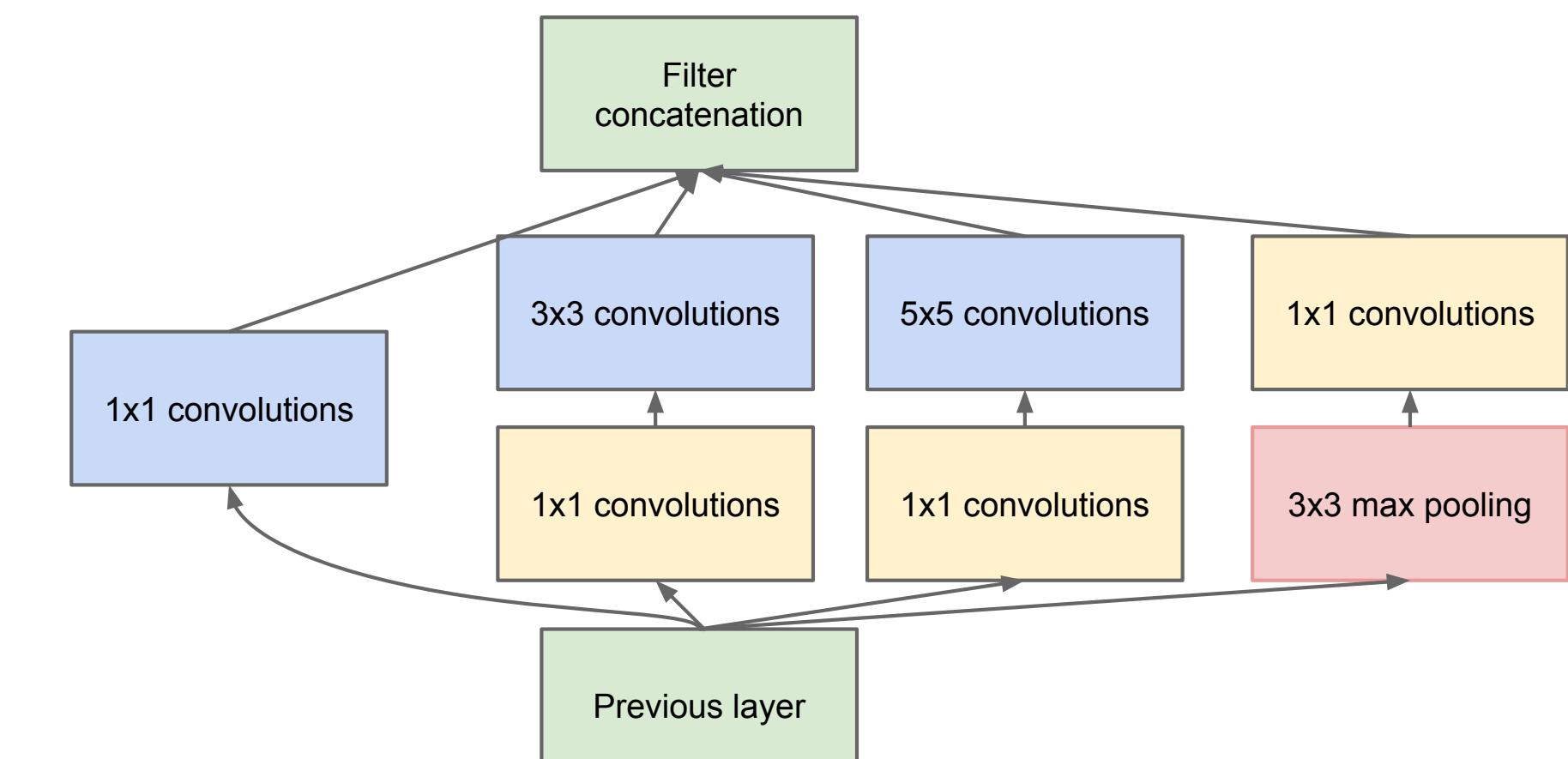
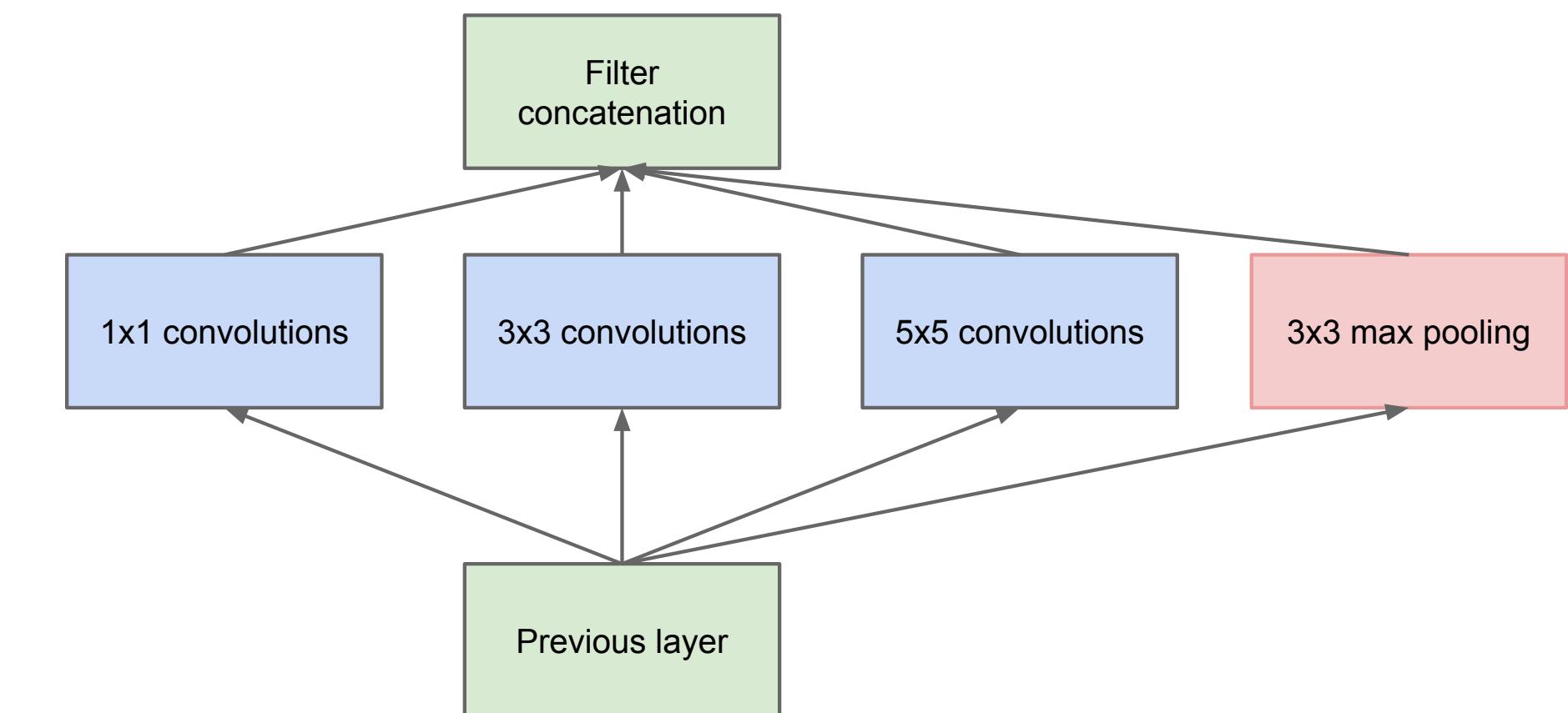
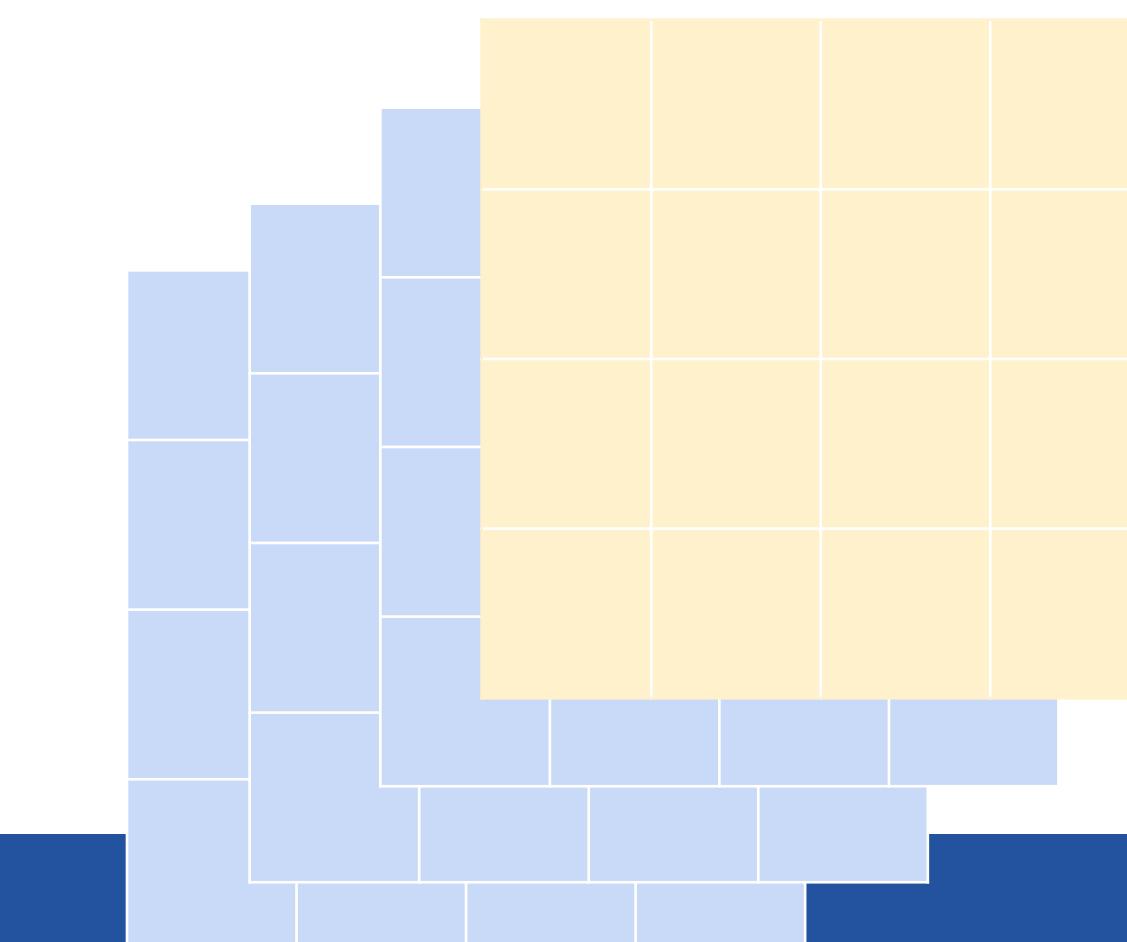


- GoogleNet (2014): built on AlexNet, introduced an inception model to reduce the number of parameters



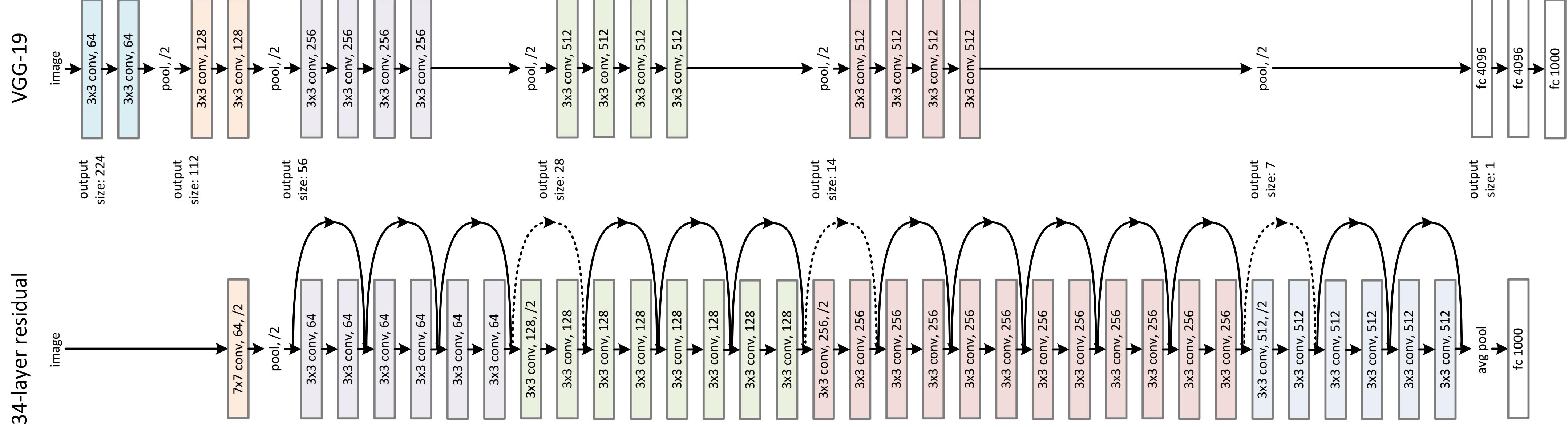
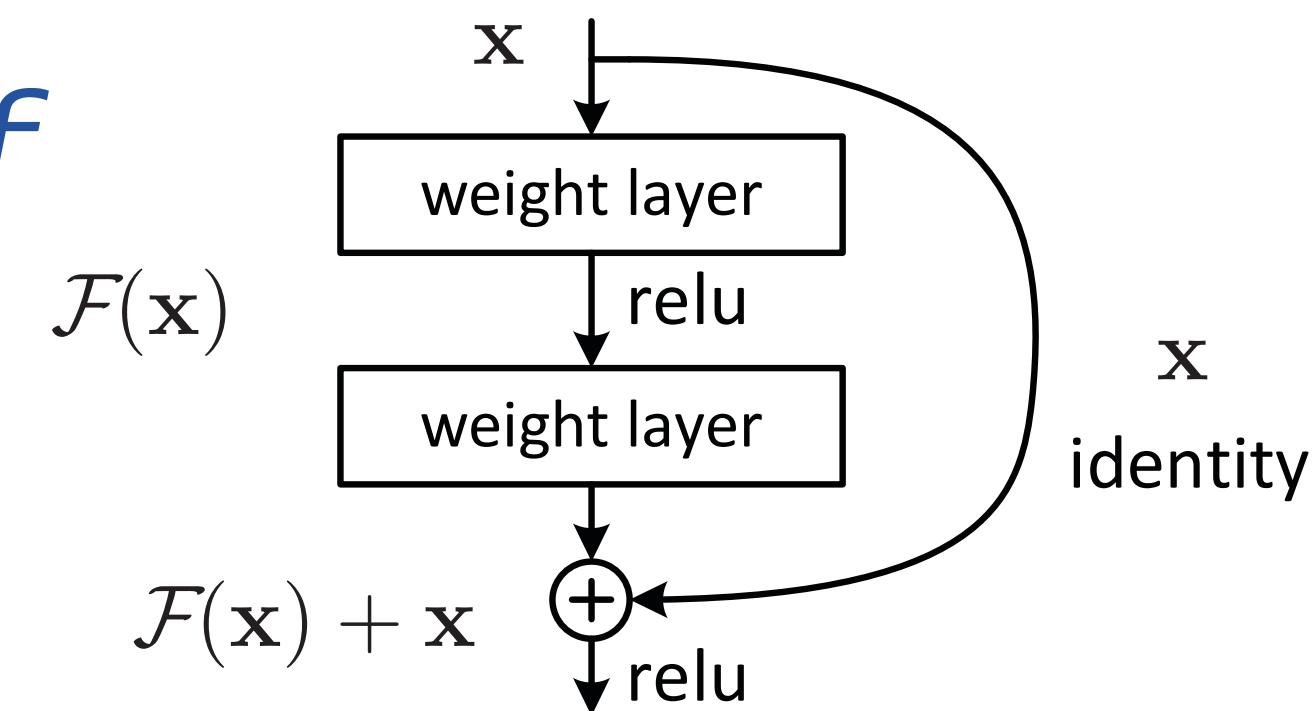
Inception module

- Rather than going deeper and deeper, inception architecture go wider
- Several conv layers, with different filter size, process the same inputs
- This way, more features can be detected from the same image
- The outcome of this parallel processing is then recombined through a concatenation step as channels of an image



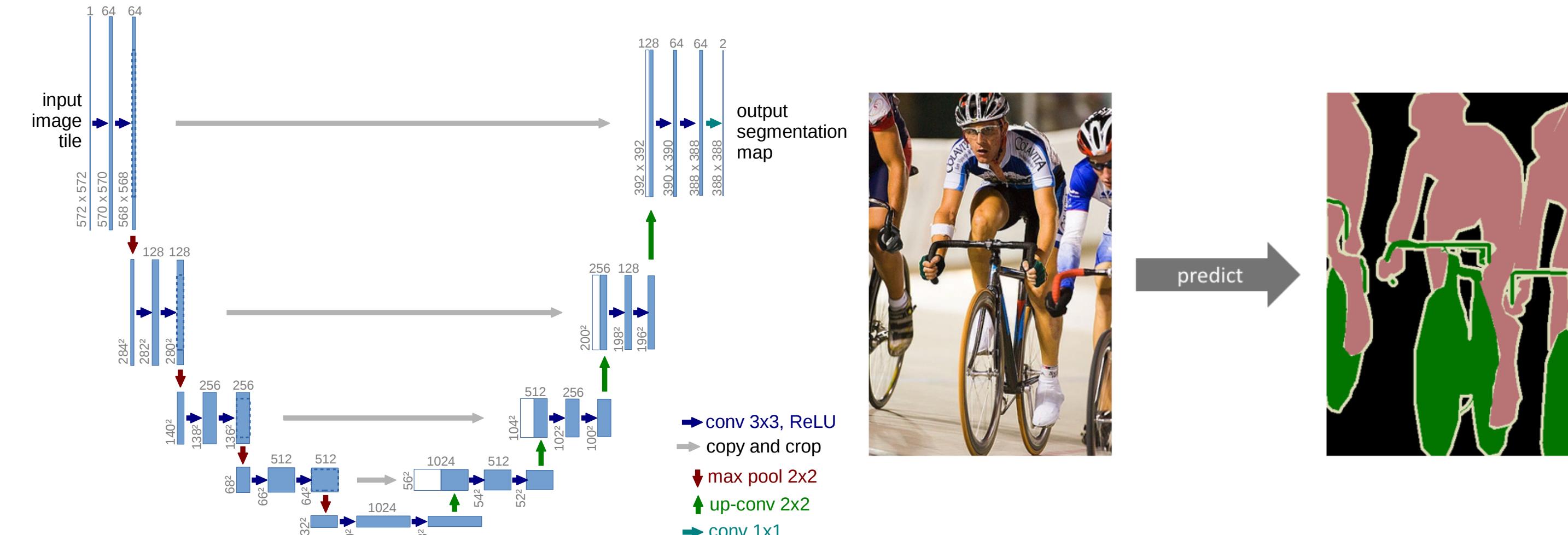
A history of ConvNns

- VGGNet (2014): exploit small filters (3×3 , 1×1), previously considered not optimal in a stack of Conv layers (rather than 1Conv+1Pooling)
- ResNet (2015): implemented skip connections, which were proven to boost performances

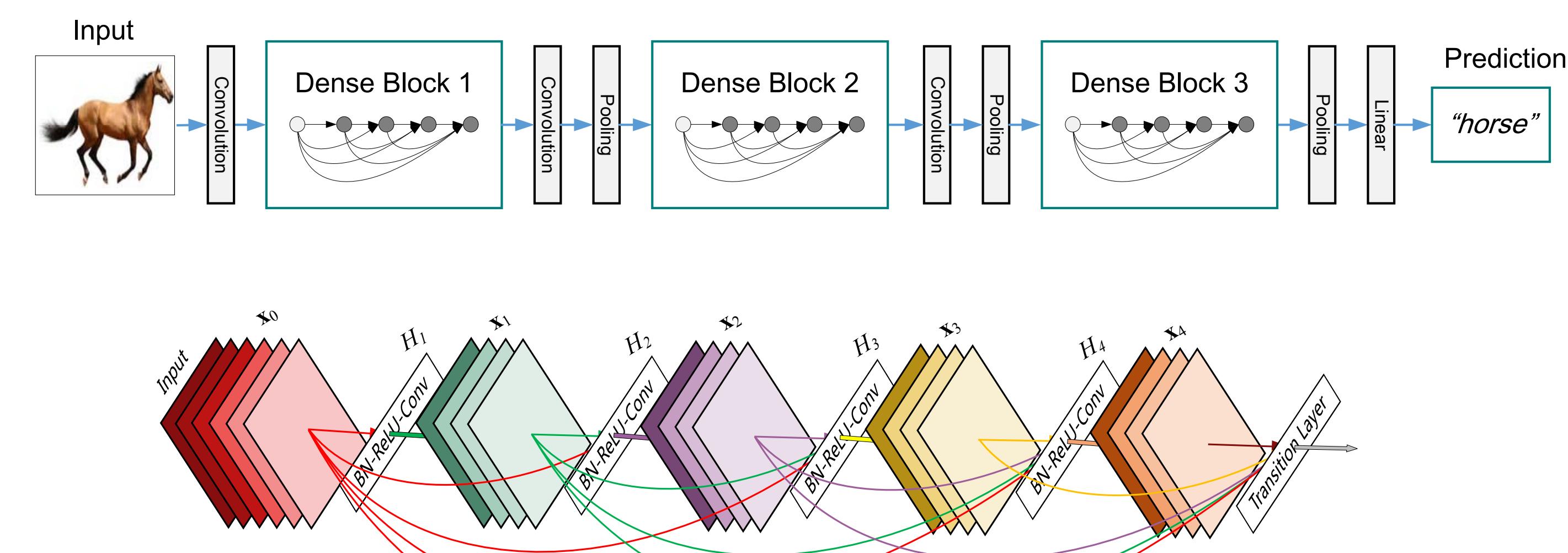


A history of Convans

- *U-net (2015): conv layer with skip connections, forming a downsampling+upsampling U-shaped sequence.*
Introduced for semantic segmentation



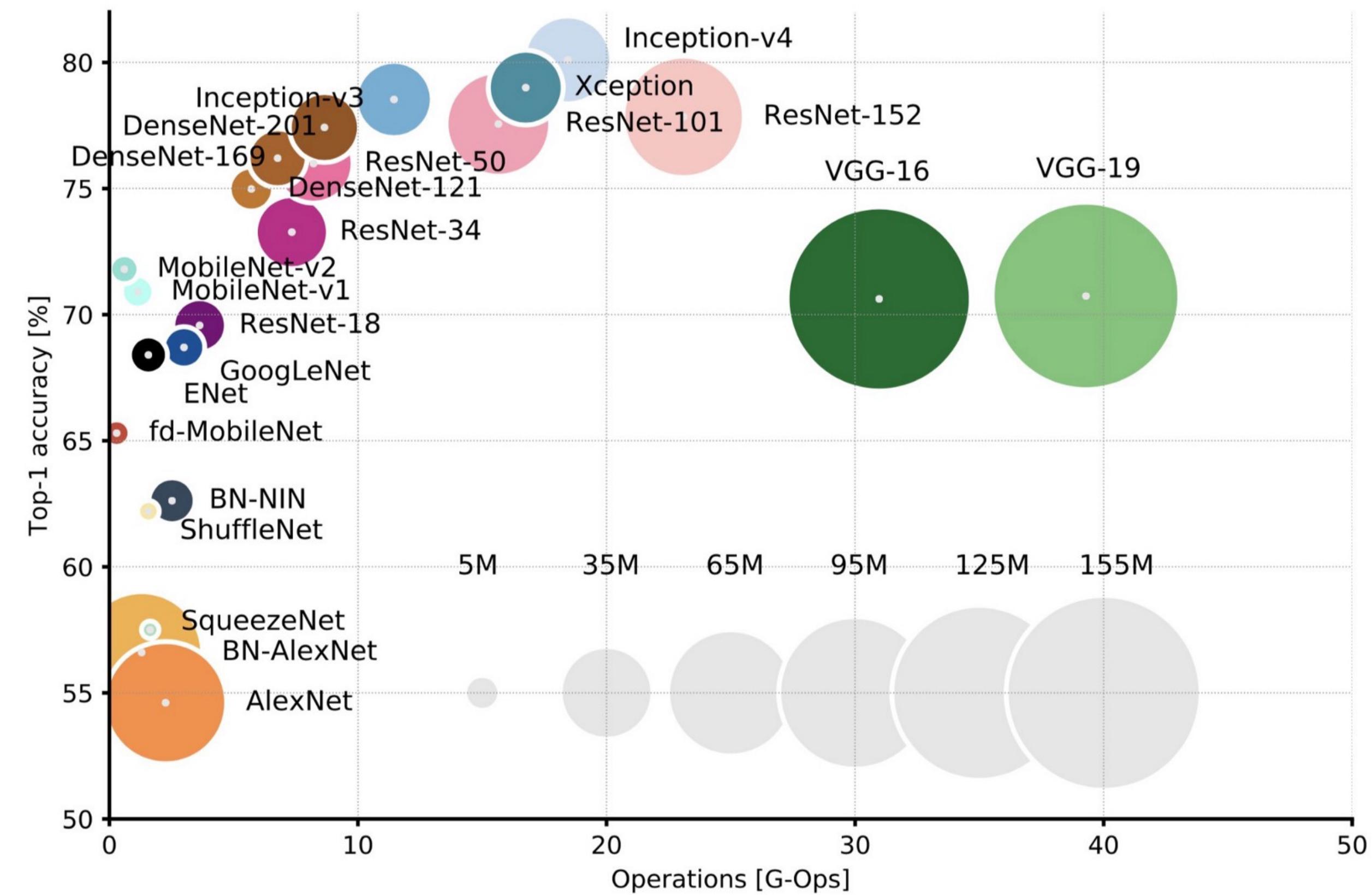
- DenseNet (2016): uses skip connections between a given layer and all the layers downstream in a dense block, with Conv layers in between



The computational cost

- In this evolution, computing-vision networks drastically improved in performance
- This came at a big cost in complexity (number of parameters and operations)
- The inference with this network became particularly slow
 - big interest in optimize these networks on dedicated resources, e.g. FPGAs
- Many cloud providers provide optimized versions of these networks:
 - you can just use them (re-training if needed), rather than inventing your own one

<https://towardsdatascience.com/neural-network-architectures-156e5bad51ba>





Conclusions

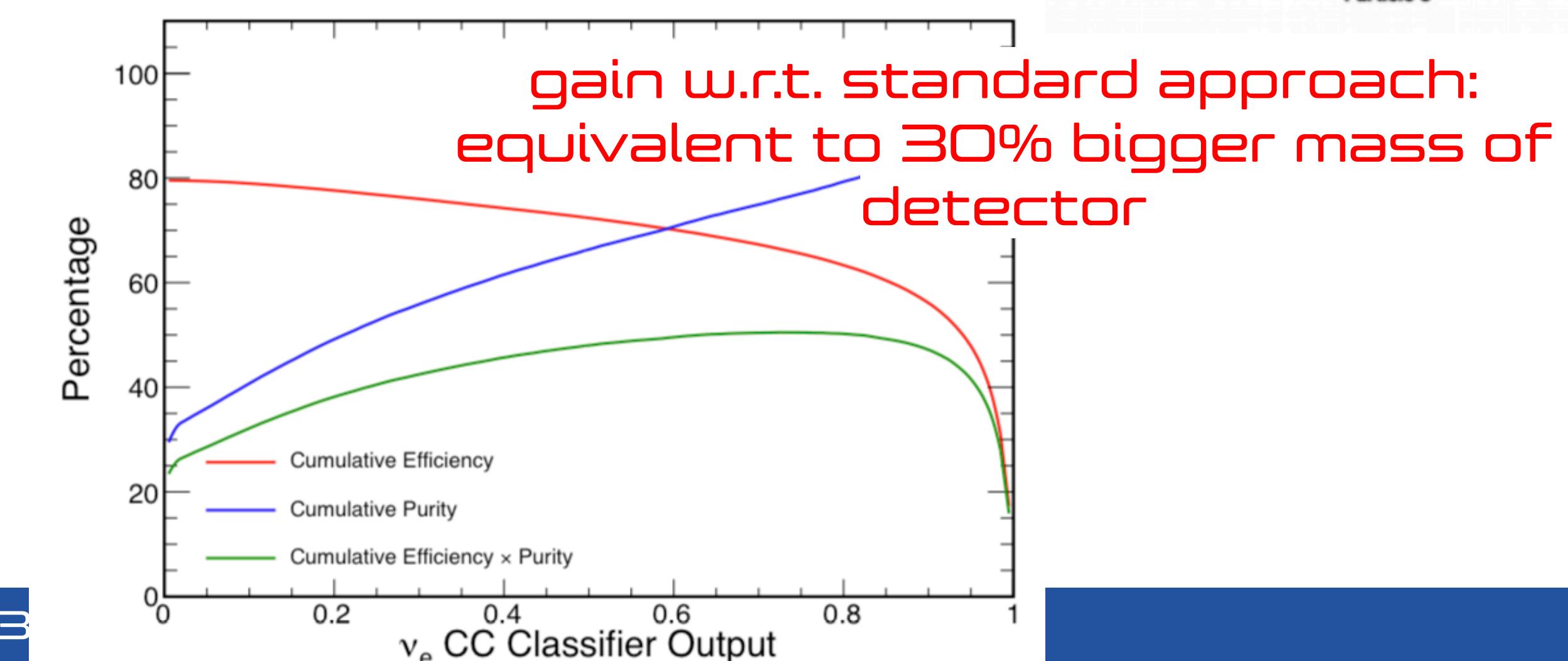
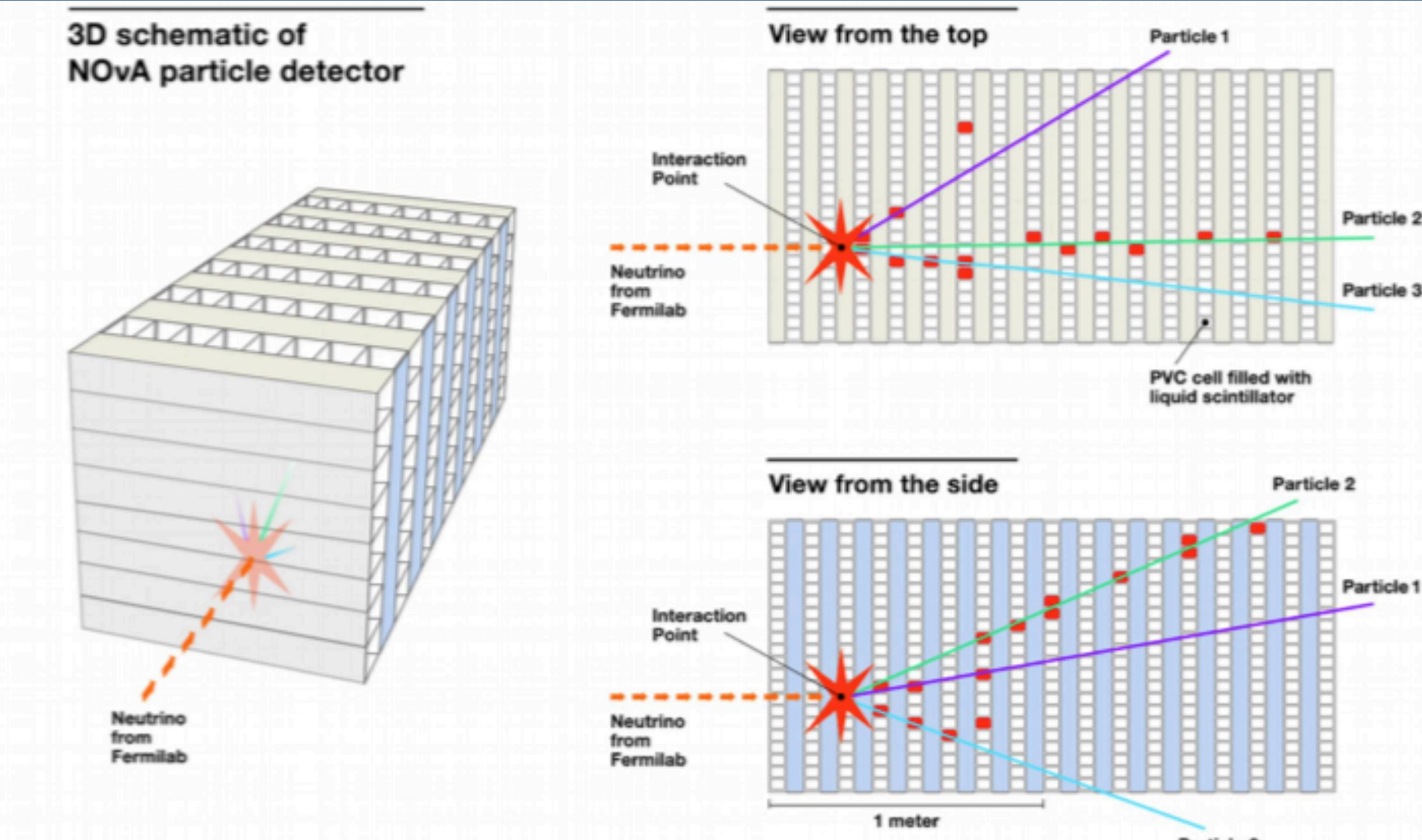
- *Convolutional Neural Networks has redefined computing vision*
- *We reviewed the basic ingredients*
- *We looked at the historical evolution*
- *We looked at performance cost*
- *We will discuss more the inference efficiency at the end of the course*



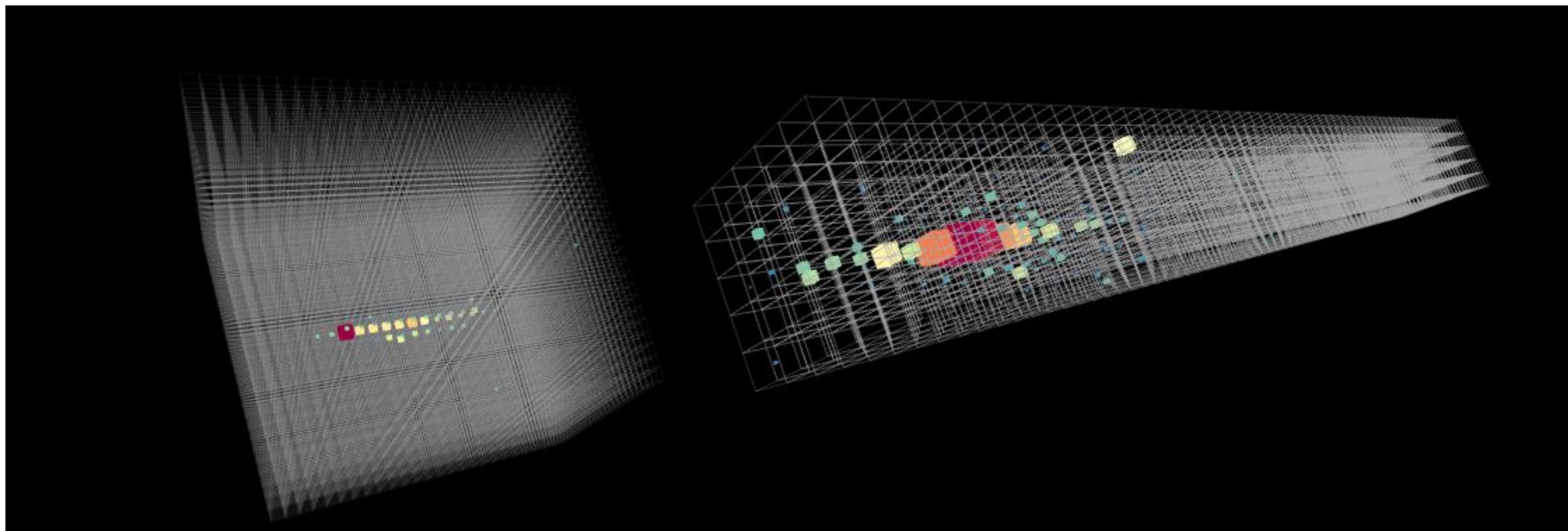
Backup

Example: PID for ν experiments

- Many HEP detectors (particularly underground) more and more structured as regular arrays of sensors
- Modern computer-vision techniques work with images as arrays of pixel sensor (in 1D, 2D, and 3D)
- These techniques were applied by Nova on electron and muon ID
- Impressive gain over traditional techniques (comparable to +30% detector == \$\$\$ saved)

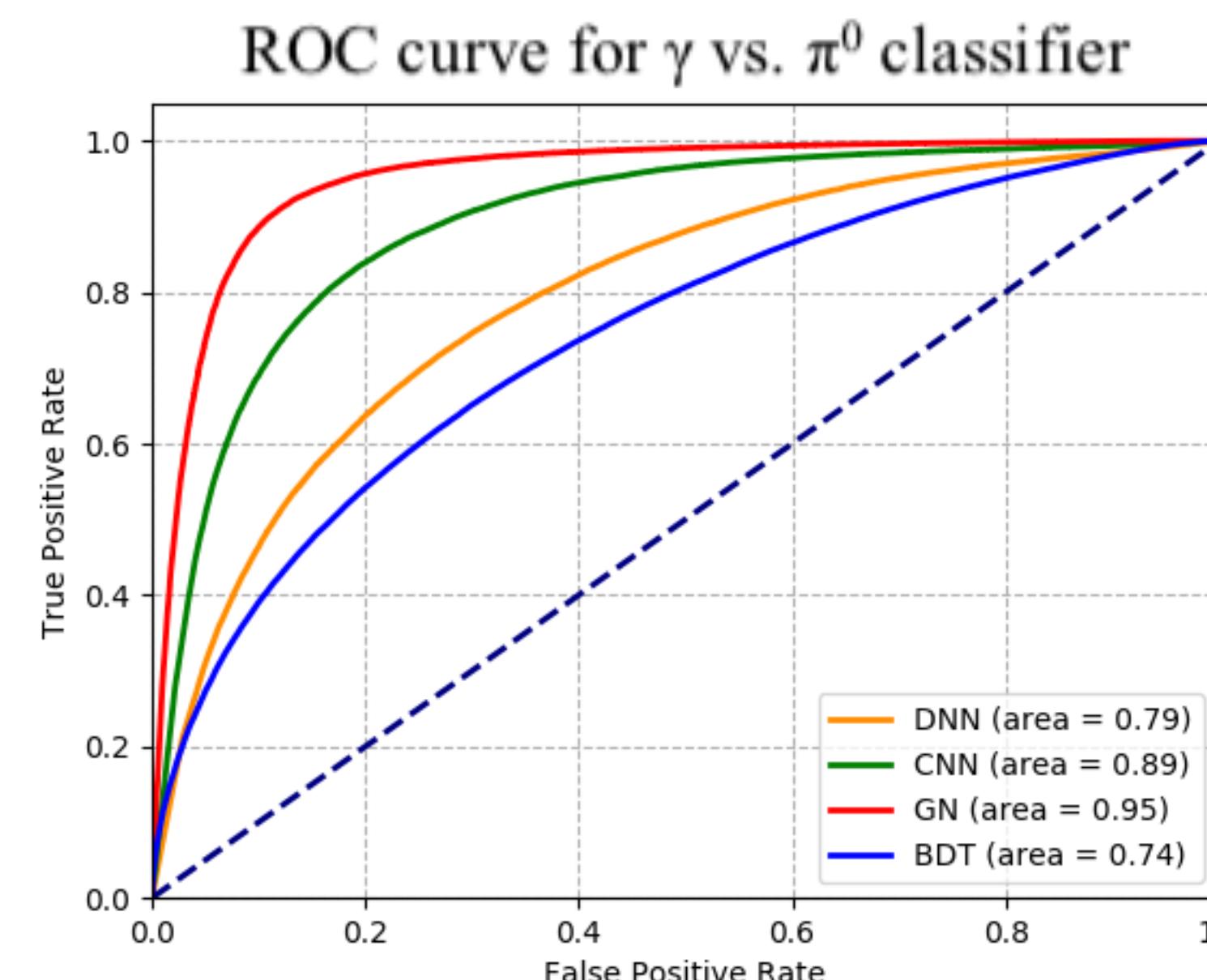
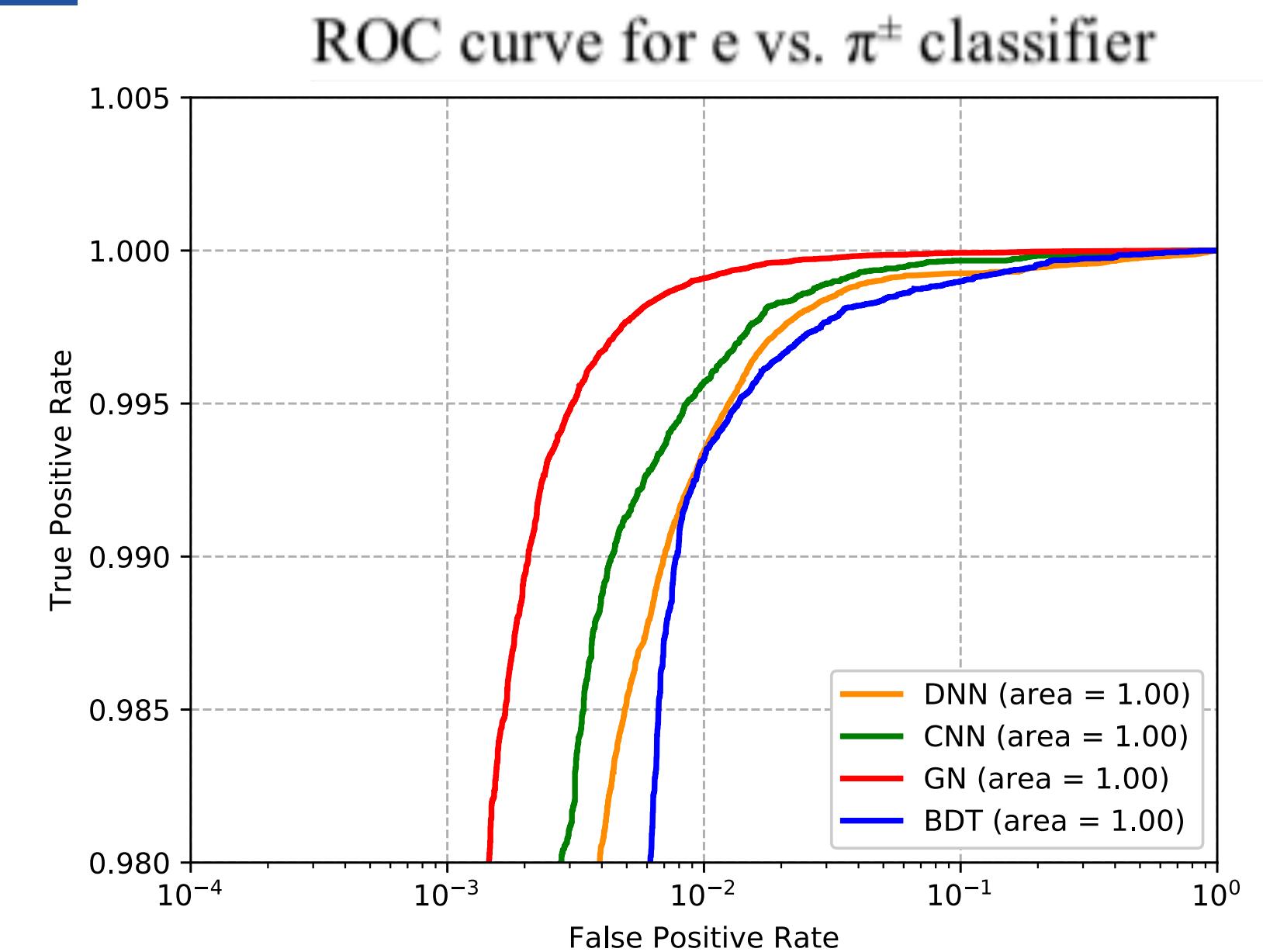


- *(next generation) digital calorimeters: 3D arrays of sensors with more regular geometry*
- *Ideal configuration to apply Convolutional Neural Network*
- *speed up reconstruction at similar performances*
- *and possibly improve performances*



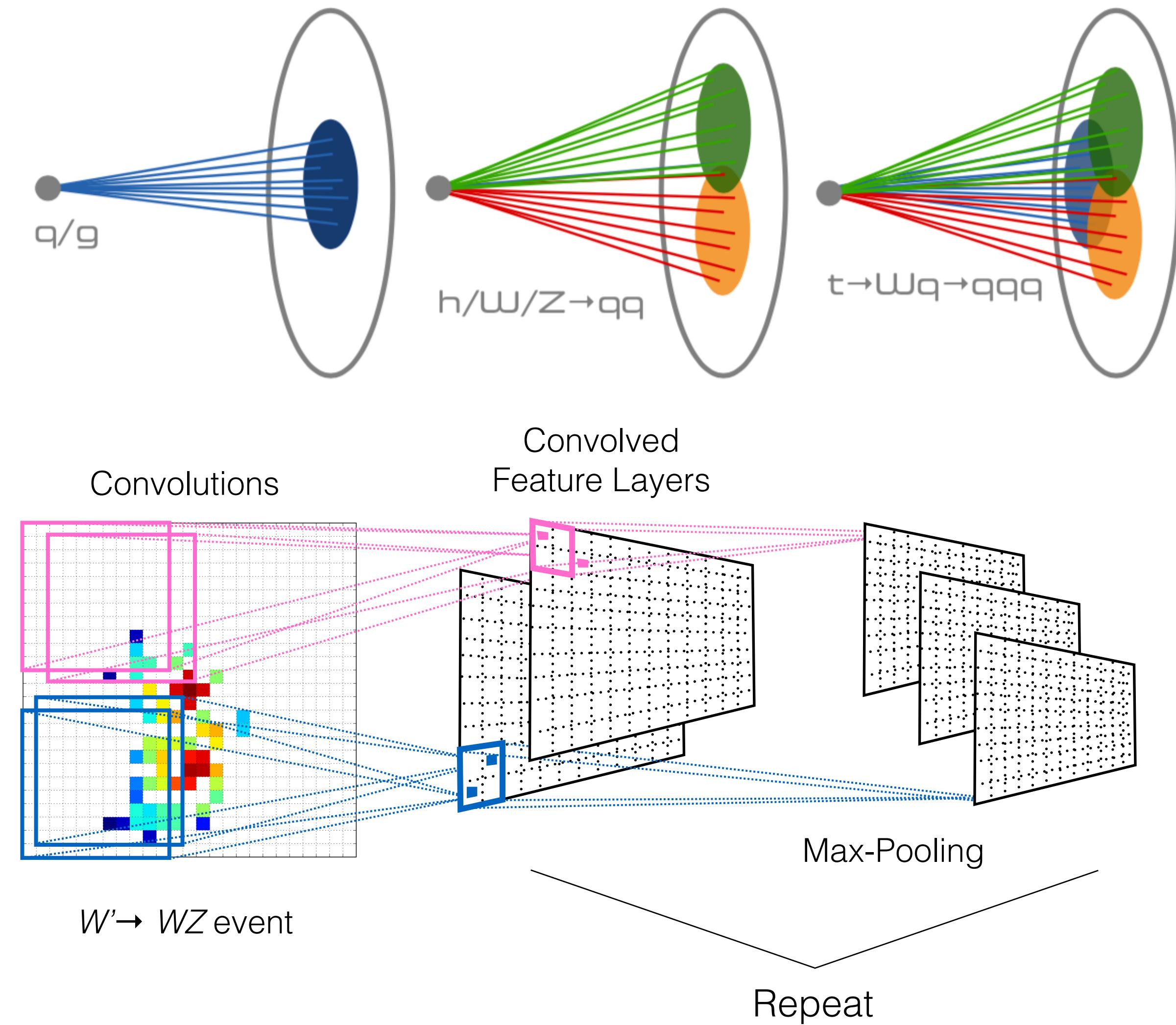
Example: Particle ID

- We tried particle ID on a sample of simulated events
- one particle/event (e , γ , π^0 , π)
- Different event representations
- high-level features related to event shape (moments of X, Y, and Z projections, etc)
- raw data (energy recorded in each cell)
- Pre-filtered pion events to select the nasty ones and make the problem harder



Jet as images (for ConvNN)

- One can pixelate the surface crossed by the jet and create an image with the momentum deposited in each cell
- Such an image can then be processed with computing-vision techniques
- Pros: can benefit of the progresses made in optimizing computing vision
- Cons: underlying assumption on detector geometry (regular array of pixels) made sacrificing information of the actual detector



Jet as images (for ConvNN)

- One can pixelate the surface crossed by the jet and create an image with the momentum deposited in each cell
- Such an image can then be processed with computing-vision techniques
- Pros: can benefit of the progresses made in optimizing computing vision
- Cons: underlying assumption on detector geometry (regular array of pixels) made sacrificing information of the actual detector

