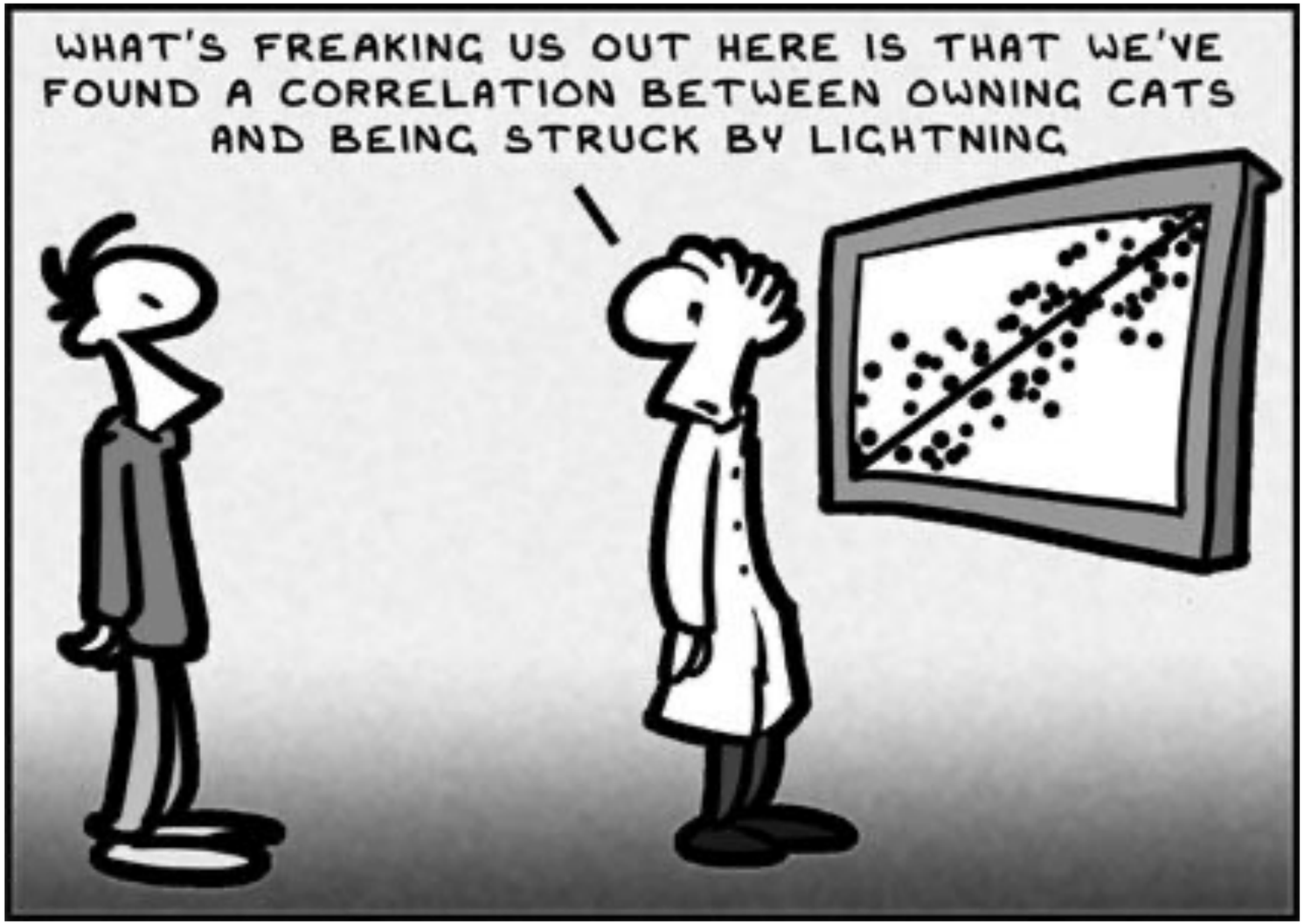
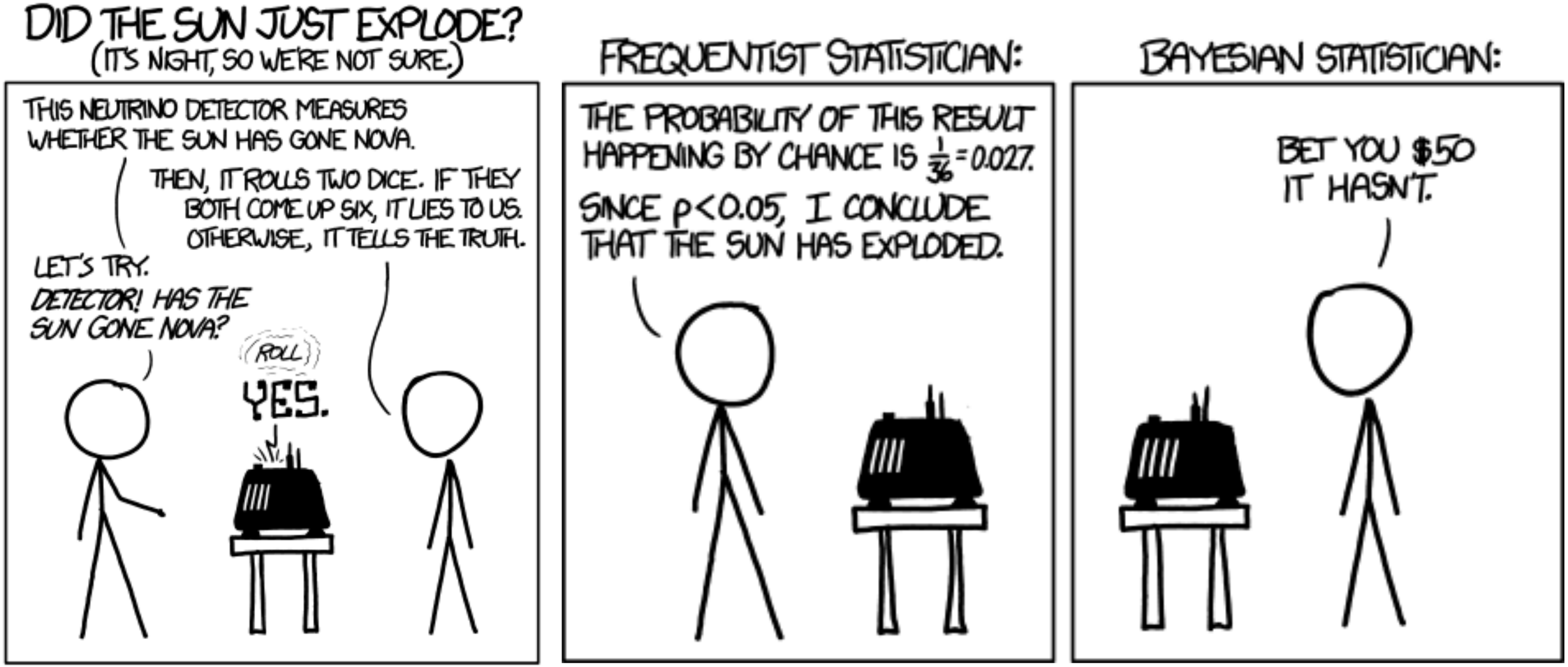
Lecture 3: Statistics

Maurizio Pierini (CERN), Thea Aarrestad (ETH)

Statistics


WHAT'S FREAKING US OUT HERE IS THAT WE'VE FOUND A CORRELATION BETWEEN OWNING CATS AND BEING STRUCK BY LIGHTNING

Statistics in a nutshell



From Probability Model to Likelihood

- ***Probability***: *When we introduced probability, we started from known distributions (e.g., a Poisson on known λ) and we tried to characterize a typical experiment outcome*

- ***Hypothesis Testing***: *Now we inverted the problem: we know the experiment outcome (e.g., we counted events above threshold during a one-year run) and we ask ourselves which of two λ values (bkg-only or sig+bkg) they come from*

- ***Inference***: *we could also just ask what is the value of λ more compatible with the observation (trivial question in this case - right? - but not in general). This is a typical application of maximum likelihood fits and a regression problem in Machine Learning*

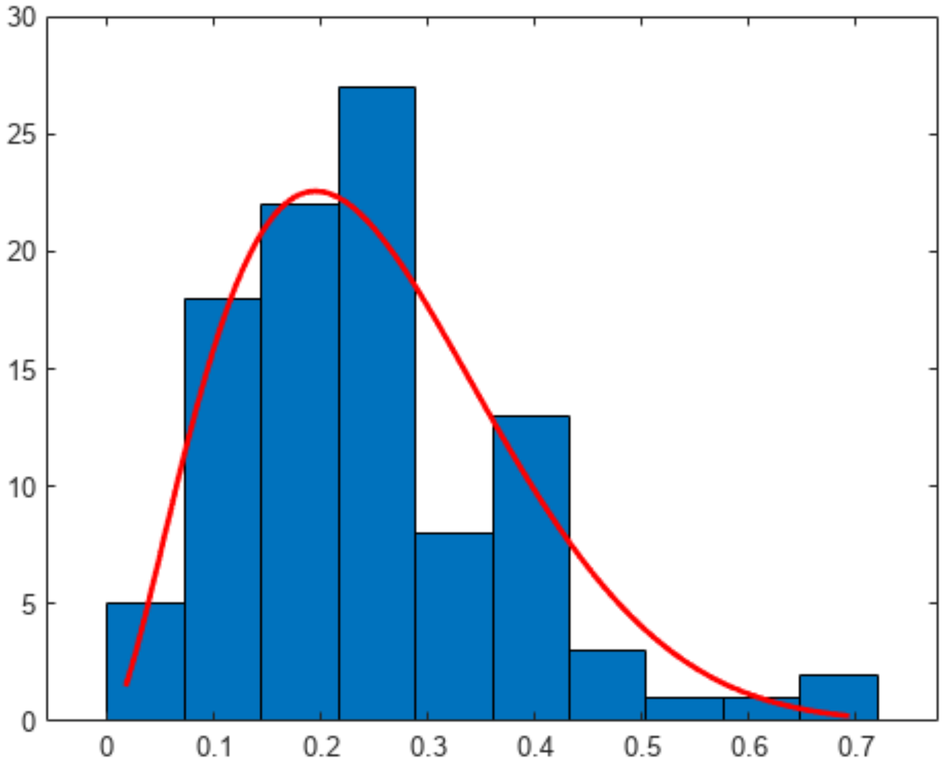
Likelihood

$$\Pr(X=k) = \frac{\lambda^k e^{-\lambda}}{k!}$$

- *Given a statistical model (e.g., our Poisson of known λ and unknown k), we can assess probabilities. Pr is a function of k*
- *Given a class of statistical models for k, function of unknown λ, we have a likelihood model*
  - *Formally the same function but a very different object*
  - *k is given (observed) and λ is unknown → A likelihood is a function of λ, given the observed k*

Likelihood

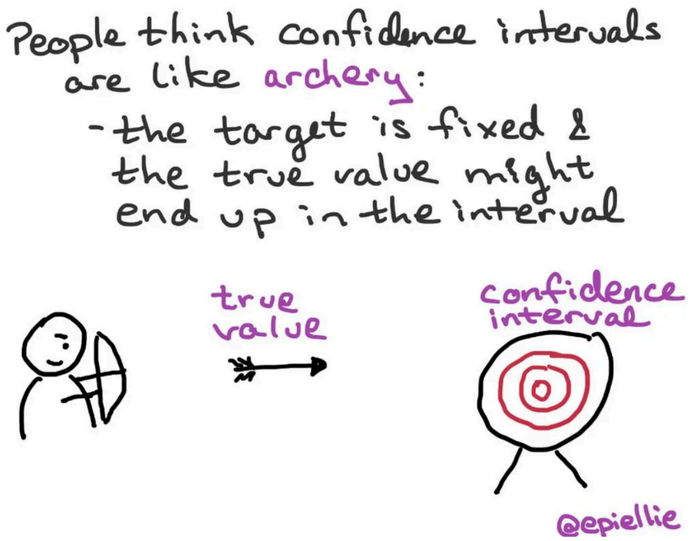
- *Let's imagine a histogram of a quantity x and a curve b(x) predicting the amount of expected background*
  - *for each bin centre x_i we can compute $b_i=b(x_i)$*
  - *the b_i values will depend on a set of parameters that describe the curve y = b(x)*
- *In each bin, we observe some counting n_i*
- *The likelihood of the model is given by*

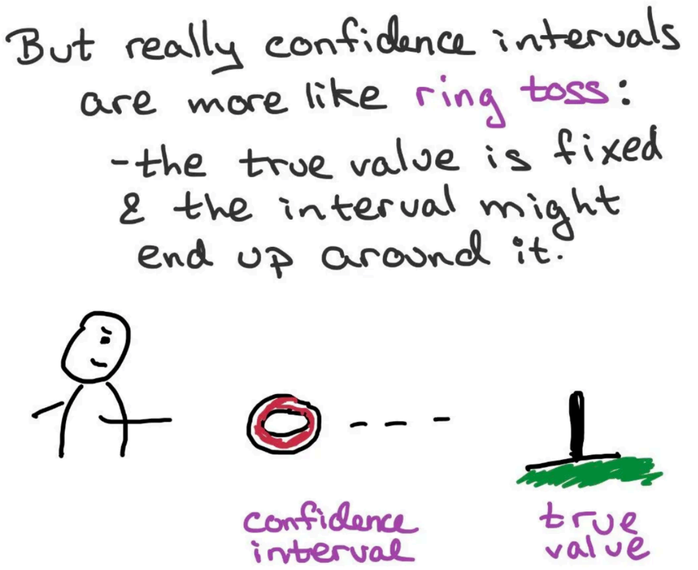


$$\mathscr{L}(\vec{n}\,|\,\vec{\alpha}) = \prod_i P(n_i\,|\,b_i(\vec{\alpha})) = \prod_i P(n_i\,|\,b(x_i\,|\,\vec{\alpha})) = \prod_i \frac{e^{-b(x_i|\vec{\alpha})} b(x_i\,|\,\vec{\alpha})^{n_i}}{n_i!}$$

Two Approaches

- ***Frequentist:***
  - *Frequentist statistics is a type of statistical inference that draws conclusions from sample data by emphasising the frequency or proportion of the data*
  - *Given an unaccessible true value the outcome of a measurement, frequentist statistics assess how typical the outcome is*
    - *The result is a confidence interval, defined based on a given probability (confidence level) that the true value is contained in an interval built as specified*

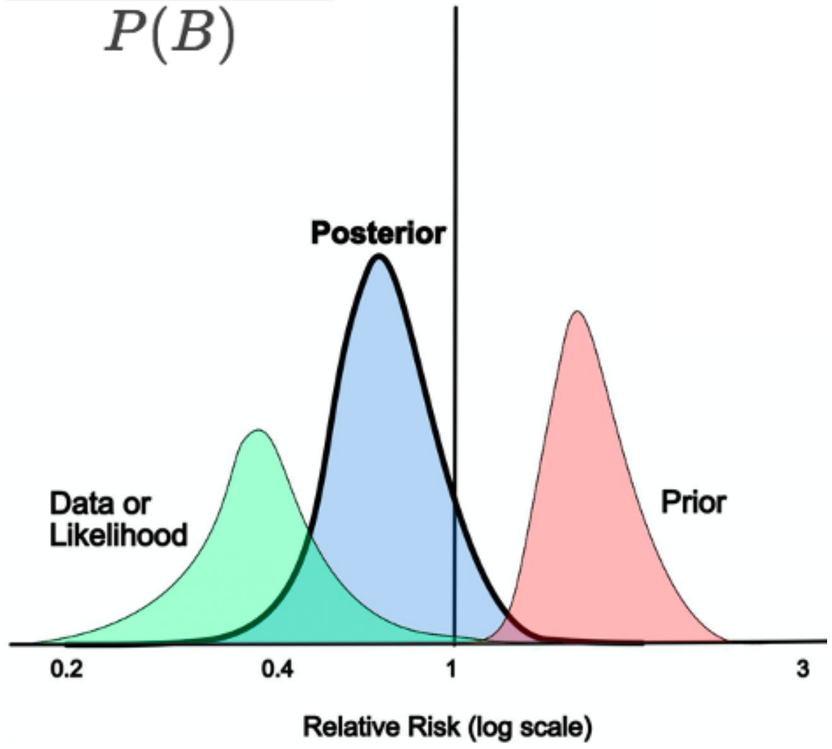




Two Approaches

- ***Bayesian:***
  - *Bayesian statistics is an approach to data analysis and parameter estimation based on Bayes' theorem. Unique for Bayesian statistics is that all observed and unobserved parameters in a statistical model are given a joint probability distribution, termed the prior and data distributions*
  - *Given an accessible true value and the outcome of a measurement, Bayesian statistics assesses a probability range (credibility interval) for the true value, based on the measurement outcome and prior knowledge of the true value*

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)}$$

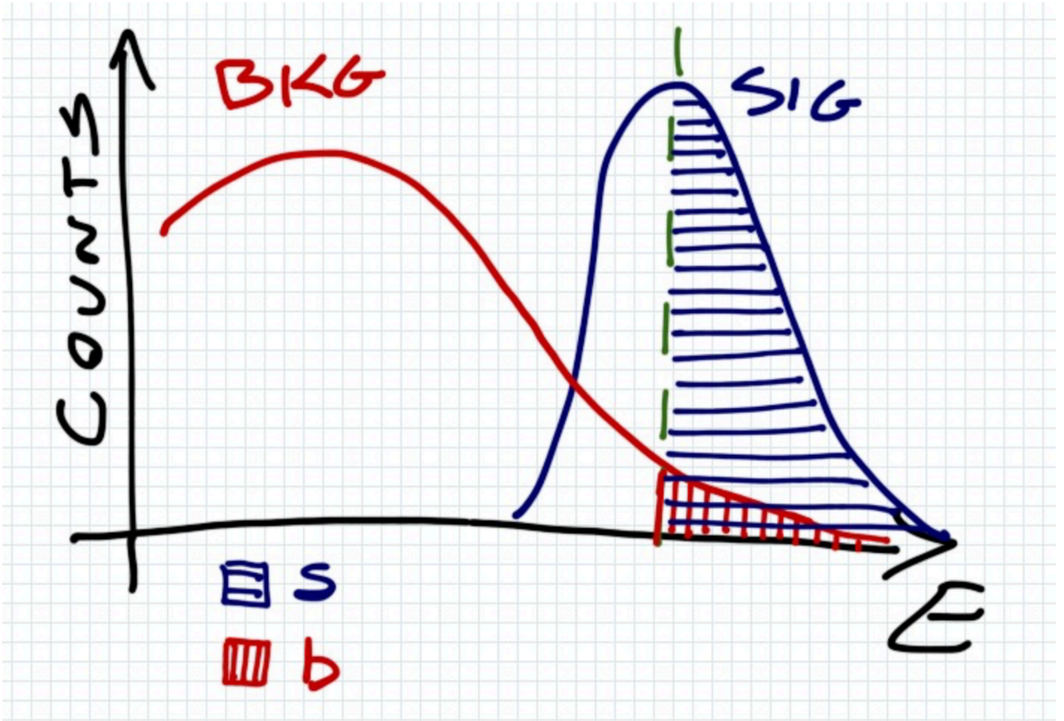


Building the likelihood

- *Let's consider the case of a Poisson process. The likelihood is a function $P(n|\lambda)$. Which λ? The signal+background $\lambda_S + \lambda_B$ or the background-only λ_B?*

- *The experiment is repeated multiple times and a set of n counting are observed. The likelihood is built as* $\prod_i P(n_i|\lambda)$

- *Once the likelihood is built, it is a function of λ. We want to estimate λ from what we observed*



Maximum Likelihood estimation

- We are given a likelihood model $\mathscr{L}(D|w)$ and some data D
  - D is known, w are unknown
- We want to find the $\hat{w}$ values that would make our data D the most probable outcome of the experiment
  - If we knew these $\hat{w}$ values, the probability of observing D would be maximal (here D would be the unknown and $\hat{w}$ the known quantities)
  - You can convince yourselves that

$$\hat{w} = \arg\max_{w} \mathscr{L}(D \,|\, w)$$

Example: Cross Entropy

- *Bernoulli's problem: probability of a process that can give 1 or 0*

$$\mathscr{L} = \prod_i p_i^{x_i}(1-p_i)^{1-x_i}$$

- *The corresponding likelihood is (as usual) the product of the probabilities across the events*

$$-\log \mathscr{L} = -\log[\prod_i p_i^{x_i}(1-p_i)^{1-x_i}]$$

- *Maximizing the likelihood corresponds to minimizing the -logL*

- *Minimizing the -logL corresponds to minimizing the binary cross entropy*

$$= -\sum_i \left[x_i \log p_i + (1-x_i)\log(1-p_i)\right]$$

- *We will use this expression as loss function in classification problems*

Example: regression & MSE

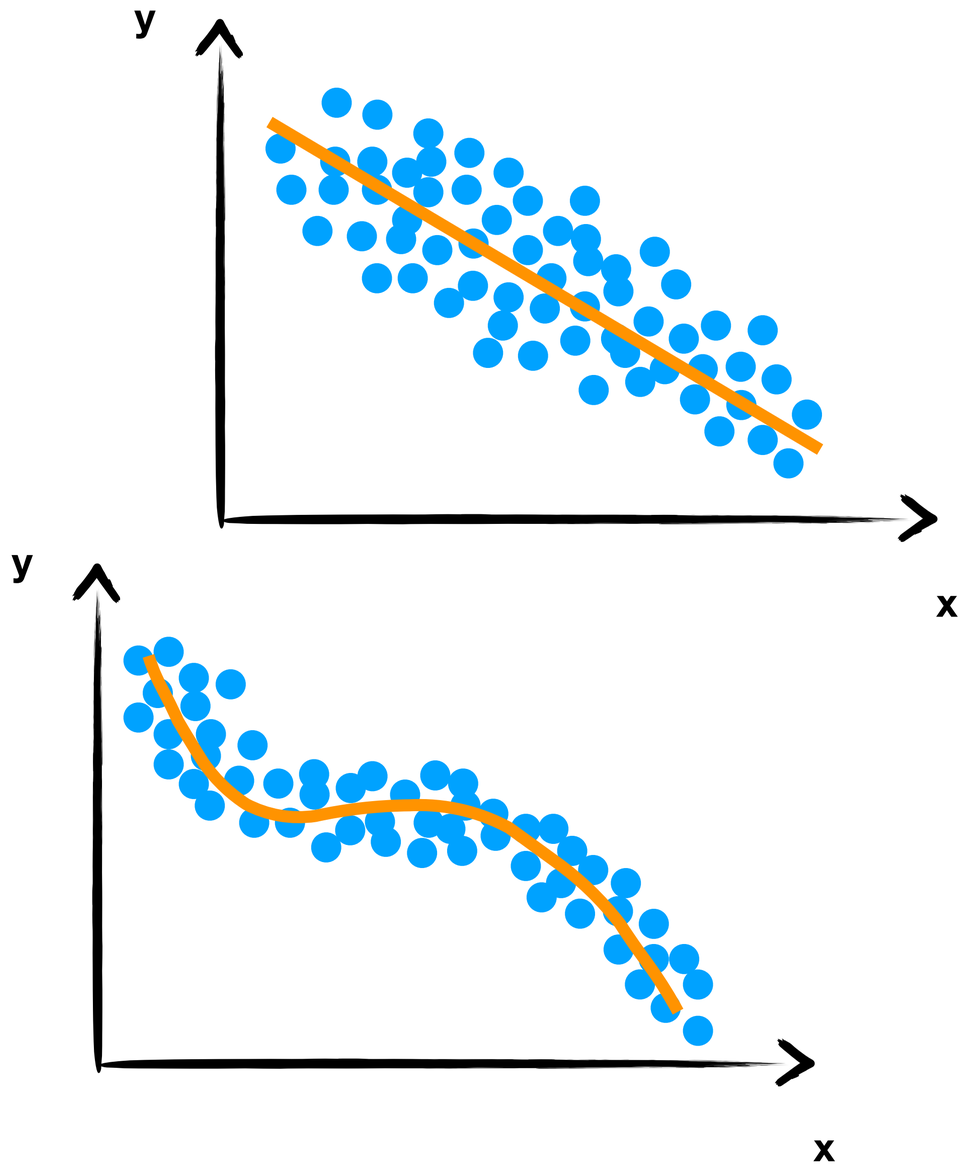
- Given a set of points, find the curve that goes through them
  - Can be a linear model

$$y_i = ax_i + b$$

  - Can be a linear function of non-linear kernel of the x. For instance, a polynomial basis

$$y_i = a\,\phi(x_i) + b$$

New feature, “engineered” from the input features



Example: regression & MSE

- Take some model (e.g., linear)

$$h(x_i \mid a, b) = ax_i + b$$

- Consider the case of a Gaussian dispersion of y around the expected value

$$y_i = h(x_i) + e_i \qquad p(e_i) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{e_i^2}{2\sigma^2}}$$

- Assume that the resolution σ is fixed

- Write down the likelihood

$$\mathscr{L} = \prod_i \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{e_i^2}{2\sigma^2}} = \prod_i \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(y_i - h(x_i))^2}{2\sigma^2}}$$

Example: regression & MSE

- *The maximisation of this likelihood corresponds to the minimisation of the mean square error (MSE)*

$$argmin[-2\log\mathscr{L}] = argmin\Big[-2\log[\prod_i \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(y_i - h(x_i))^2}{2\sigma^2}}]\Big]$$

$$= argmin\Big[\sum_i \frac{(y_i - h(x_i))^2}{\sigma^2}\Big] = argmin\Big[\sum_i (y_i - h(x_i))^2\Big] = MSE$$

- *MSE is one of the most popular loss functions when dealing with continuous outputs. We will use it a few times in the next days*

- ***BE AWARE OF THE UNDERLYING ASSUMPTION:*** *if you are using MSE, you are implicitly assuming that your y are Gaussian distributed, with fixed RMS*

- *What if the RMS is not a constant? Try to work it out…*

Summary

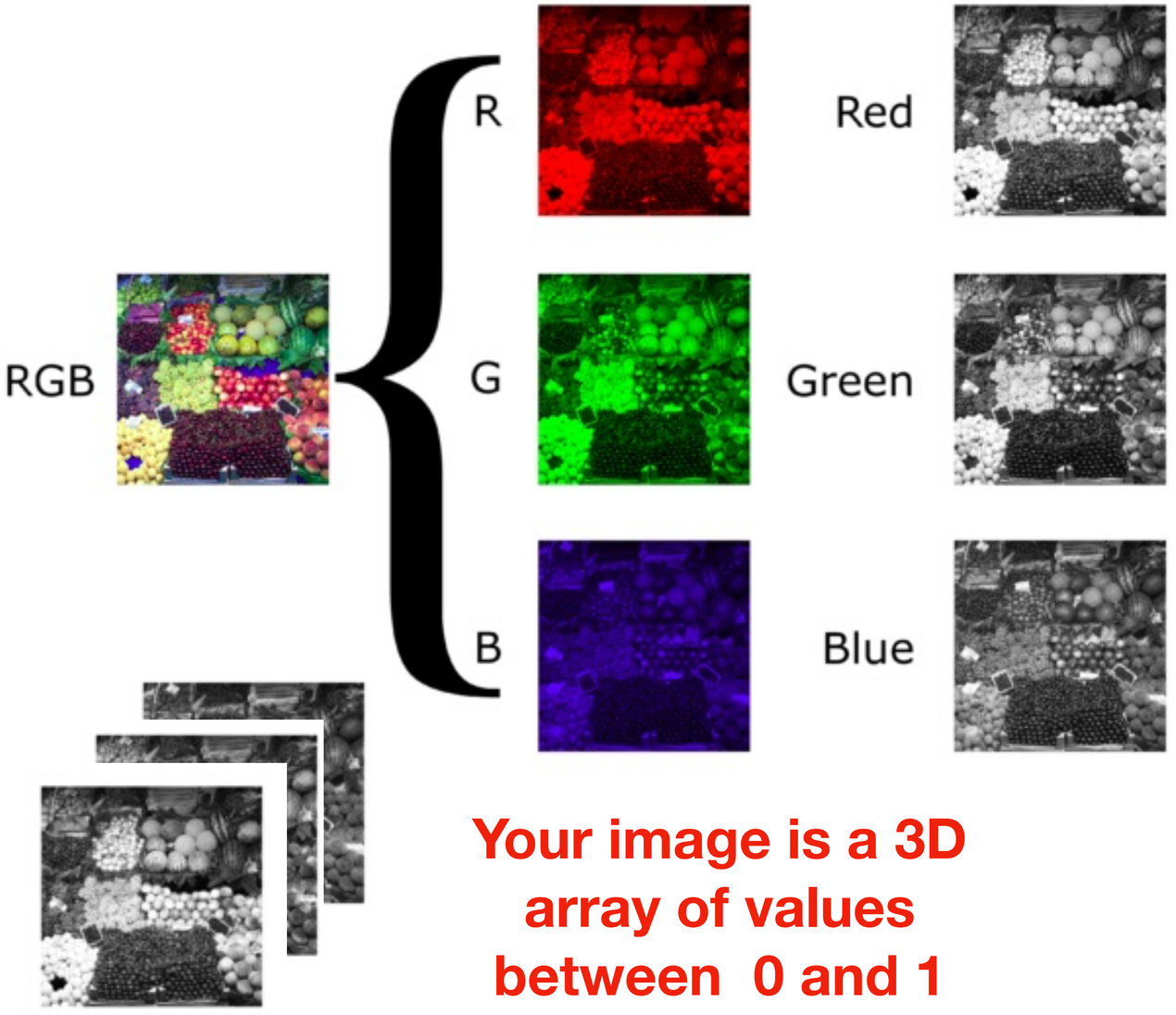
- *Introduced basic concepts of linear algebra, that will be useful for some of the math that we will find (and that you will find in literature)*
- *Introduced basic concepts of probability and statistics, since we will be dealing with non-deterministic situations when learning from data*
- *Showed how to derive popular loss functions from likelihood functions of specific problems: finding the best network corresponds to deriving a best parameter estimate from a likelihood*

Lecture 3: Convolutional Neural Networks

Maurizio Pierini (CERN), Thea Aarrestad (ETH)

Digital Image Processing

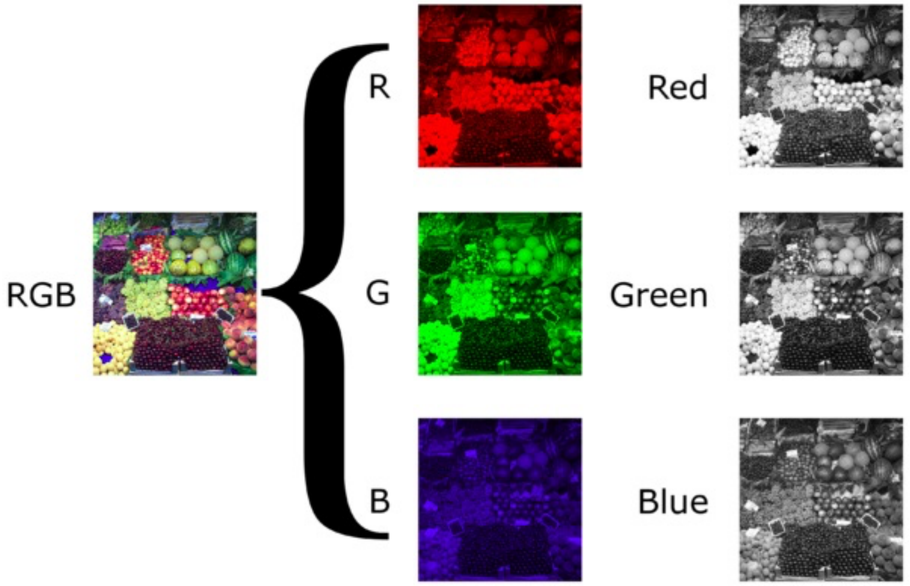
- *Each image is a matrix of pixels*
- *Each pixel comes with a color*
- *In RGB scheme, these are three colour values defined in [0,1]*
- *Each image becomes a 3D tensor*
  - *3 channels of 2D pixelated images*
- *Digital images can be processed with Convolutional Neural Networks*

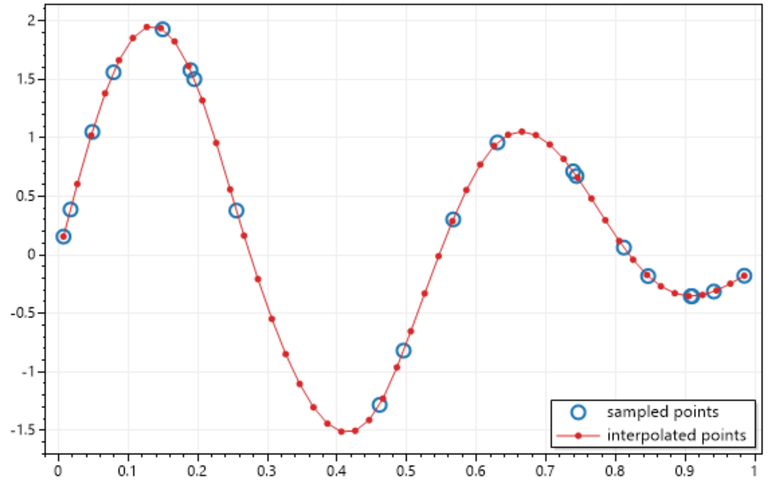


Your image is a 3D array of values between 0 and 1

What is a CNN

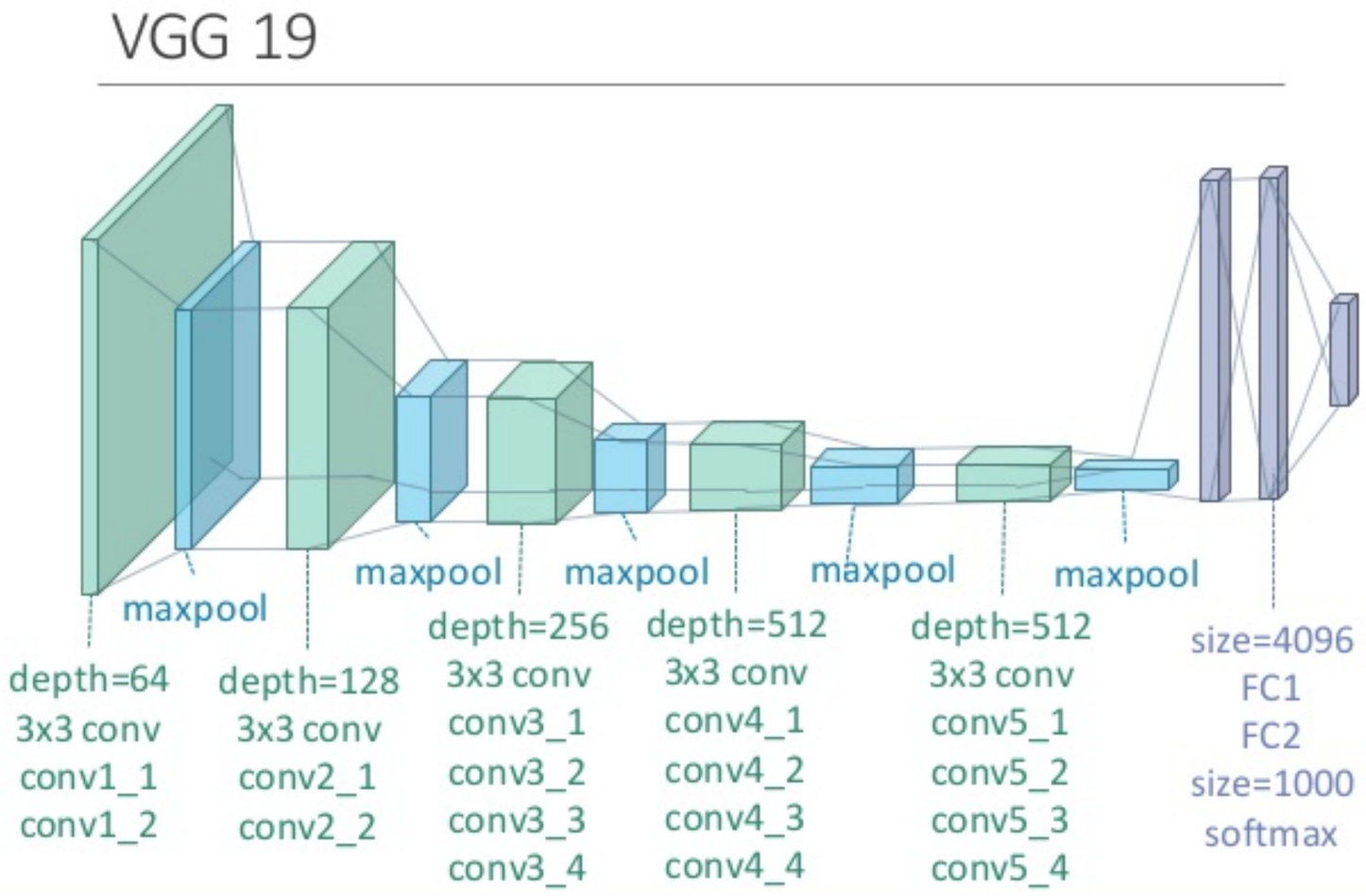
- *A special kind of NN designed process data coming in the form of a regular array*
  - *An image, coming in RGB format*
  - *A time series, i.e., the recording of a given readout at fixed intervals of time*
  - *…*
- *The name of the network comes from the mathematical operation that the network applies 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*



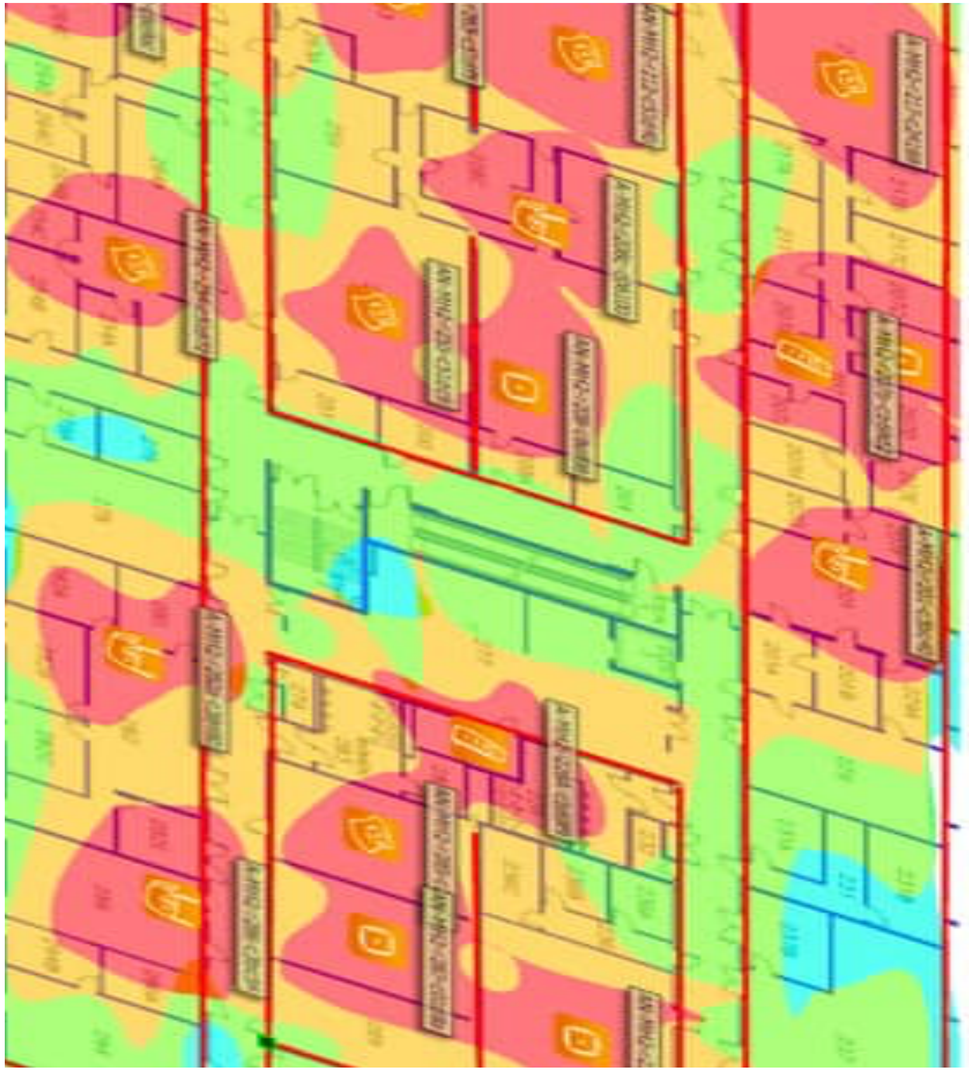
What is a convolution

- *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**

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

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 $k' \to k$ and $l' \to l$. Notice that for a given point (i,j) the convolution moves towards lower coordinates while one moves fwd in the kernel coordinates

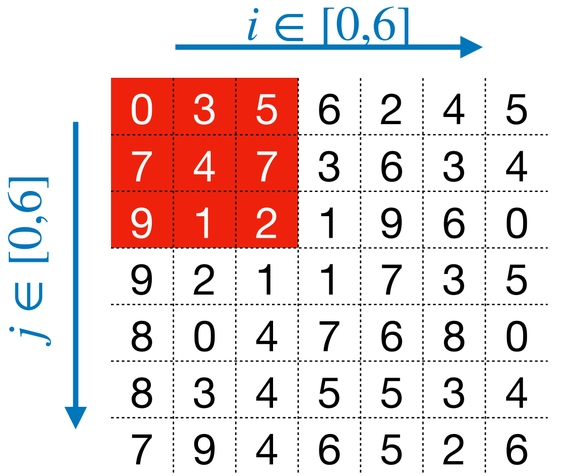
CNNs use Cross correlation

- *Typically, CNNs don't apply the convolution we just described*
- *Instead, they apply a similar expression, called cross correlation*

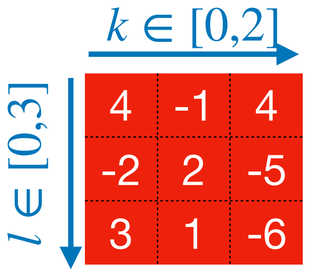
$$S(i,j) = (I * K)(i,j) = \sum_k \sum_l I(i+k, j+l) K(k,l)$$

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

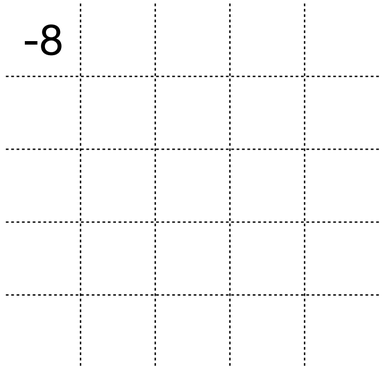
$I(i,j)$



$K(k,l)$



S(0,0) = 0x4 - 3x1 + 5x4 + -7x2 + 4x2 - 7x5 + 9x3 + 1x1 - 2x6 = -8



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*

$I(i,j)$

$i \in [0,6]$

$j \in [0,6]$

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

$K(k,l)$

$k \in [0,2]$

$l \in [0,3]$

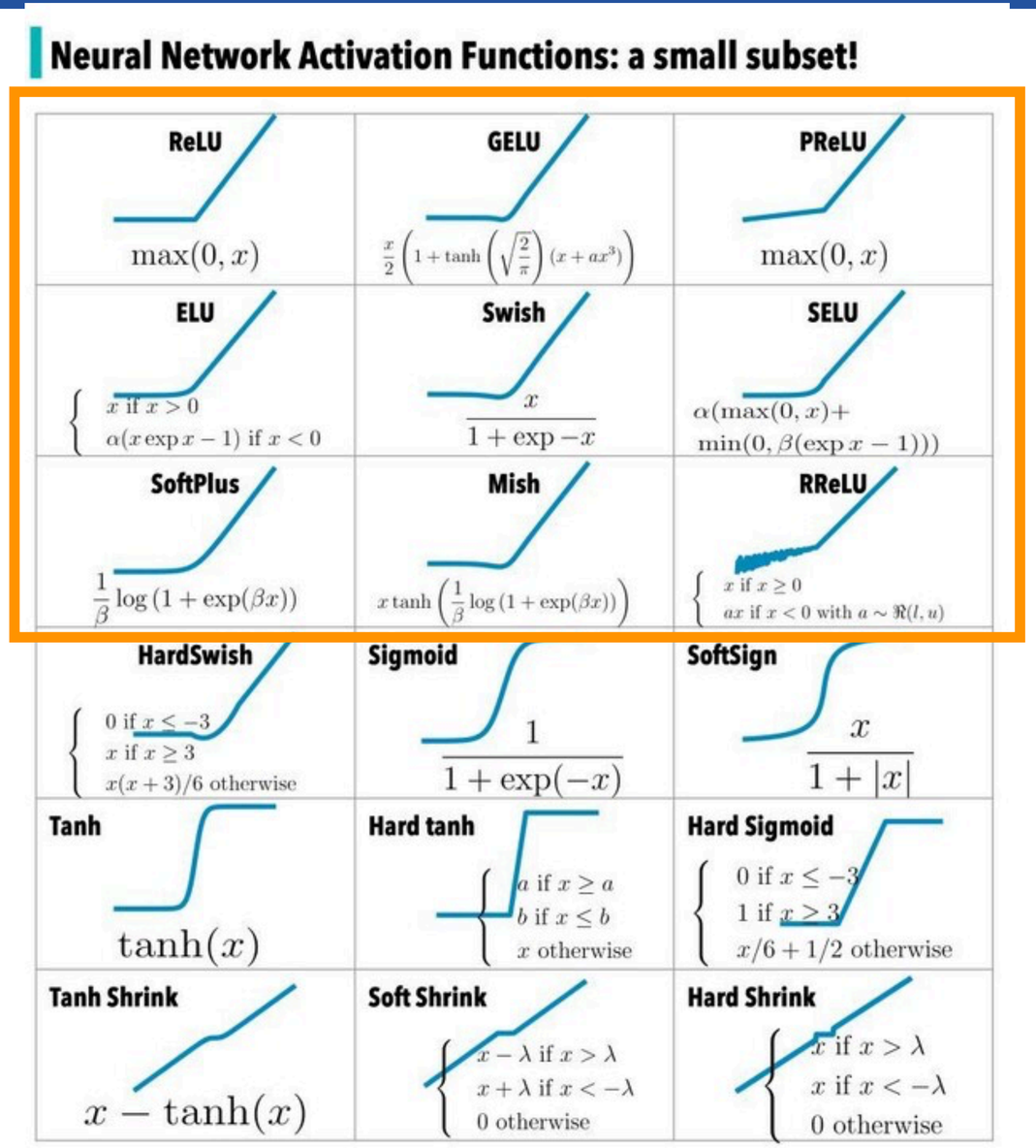
4	-1	4
-2	2	-5
3	1	-6

S(1,0) = 3x4 - 5x1 + 6x4 + -4x2 + 7x2 - 3x5 + 1x3 + 2x1 - 1x6 = 21

-8	21			

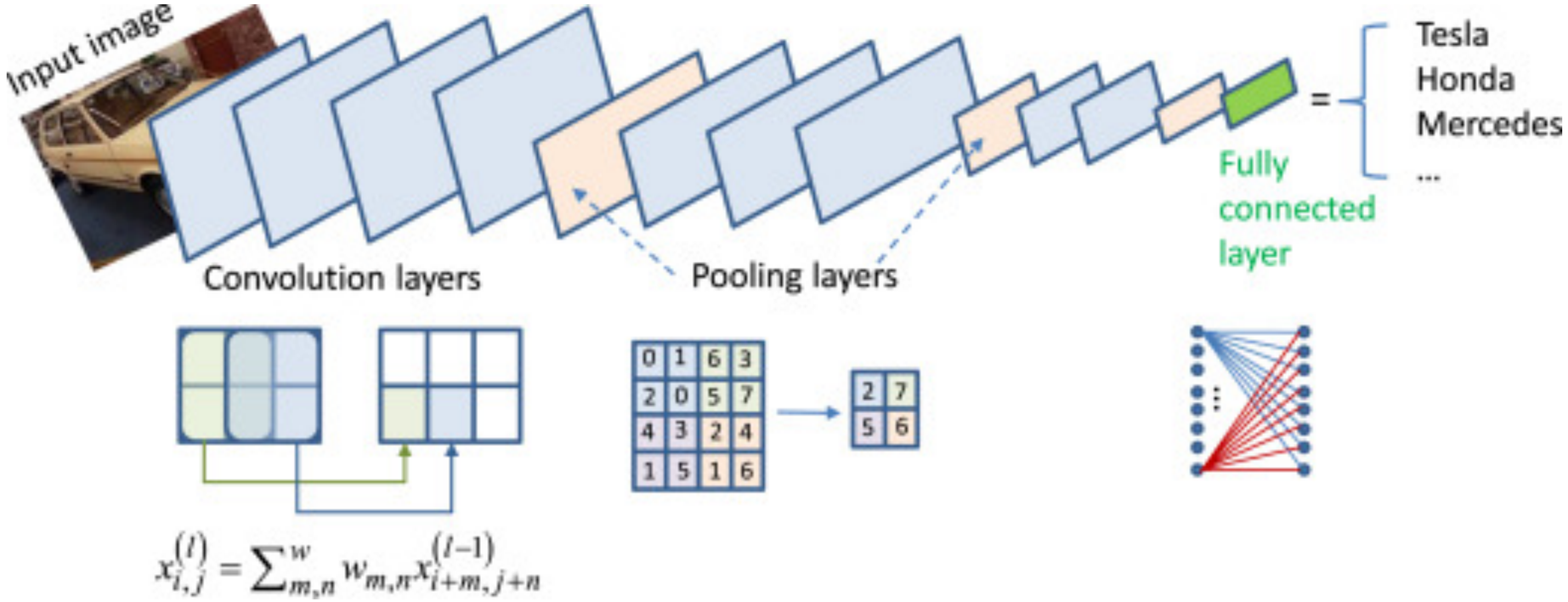
After convolution: activation

- *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*



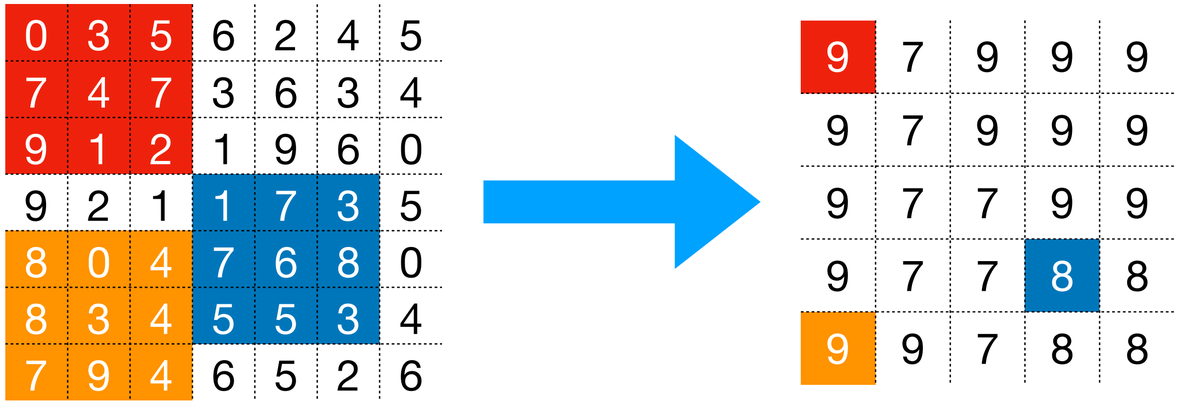
Pooling

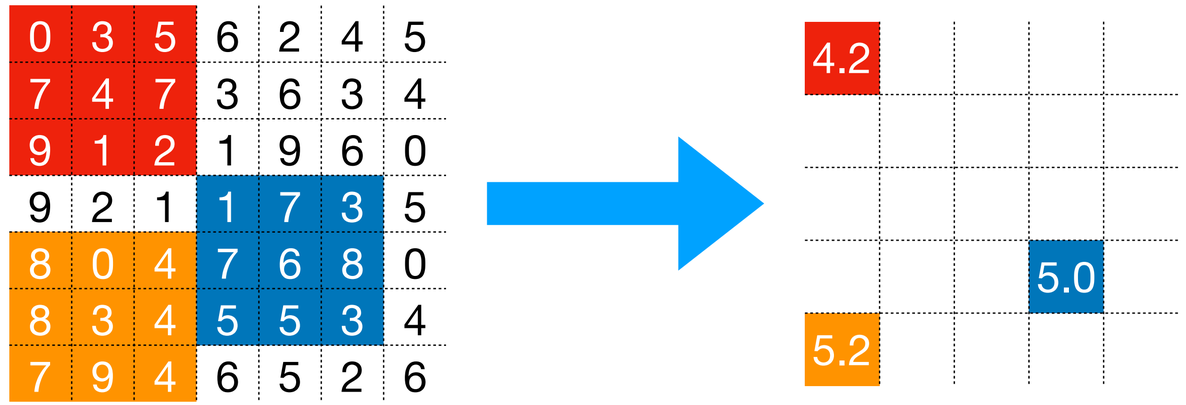
- *A pooling function is a manipulation of an array, in which the content of a pixel is replaced by a function of the nearby pixels*
- *As a result, the image size is reduced.*



Pooling

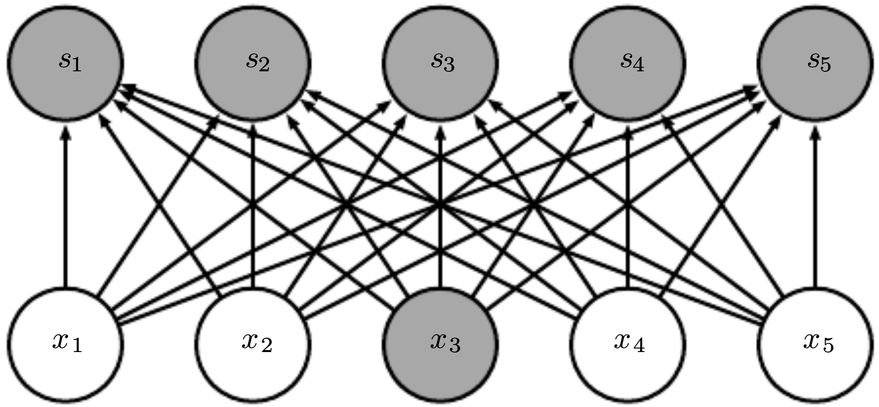
- *MaxPooling: Given an image and a filter of size k x k', scans the image and replaces each k x k' patch with its maximum*
- *AveragePooling: Given an image and a filter of size k x k', scans the image and replaces each k x k' patch with its average*
- …



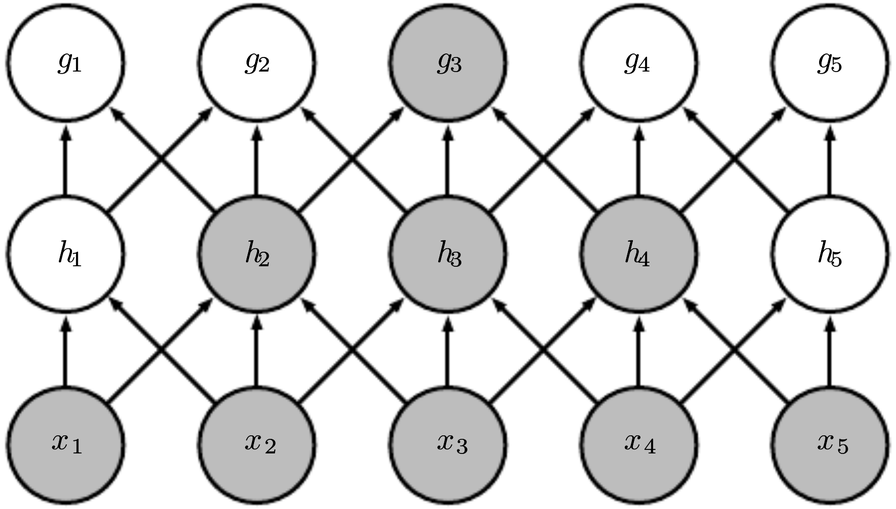


Why do CNN work?

- *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)*

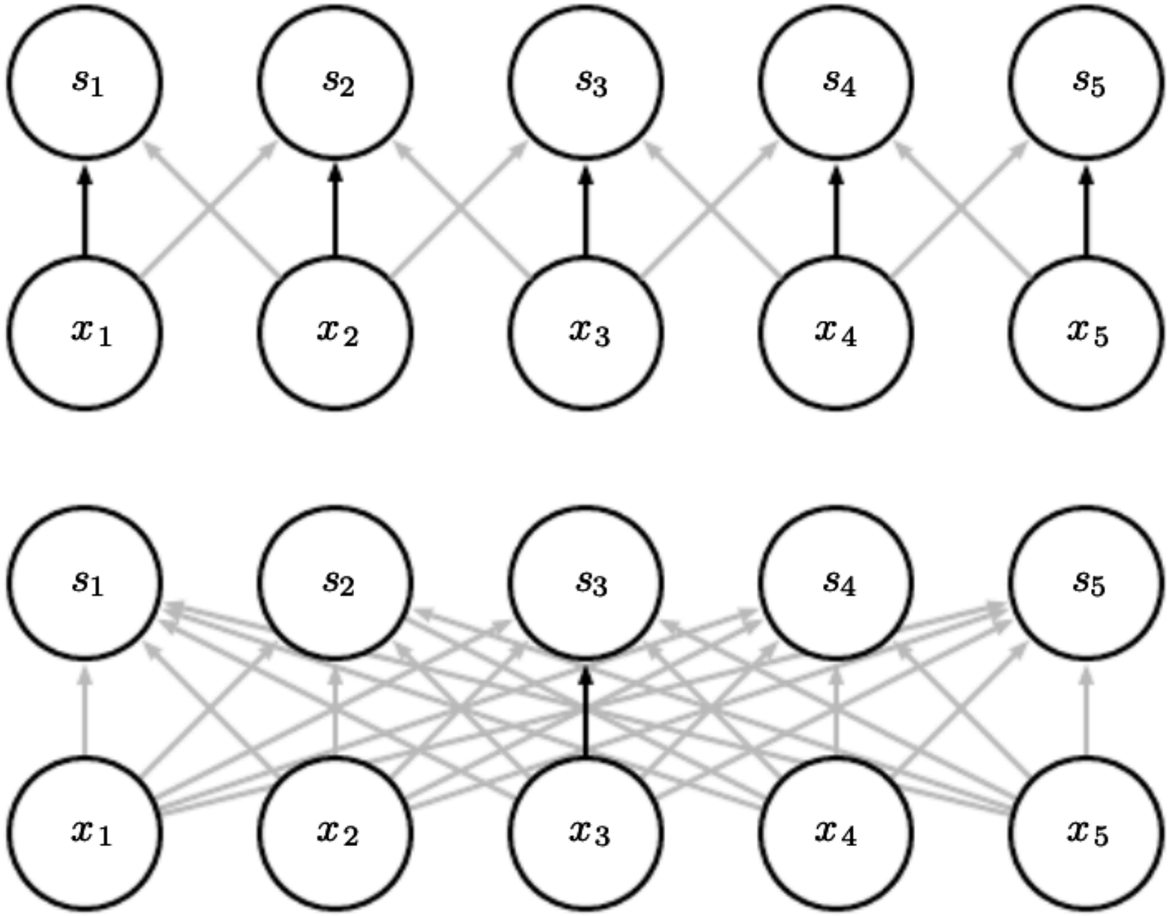


One extra layer but much less connections (i.e., parameters) reach the same receptive field



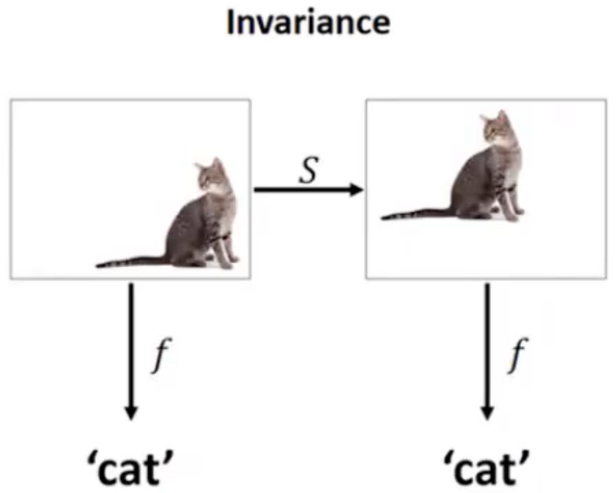
Why do CNN work?

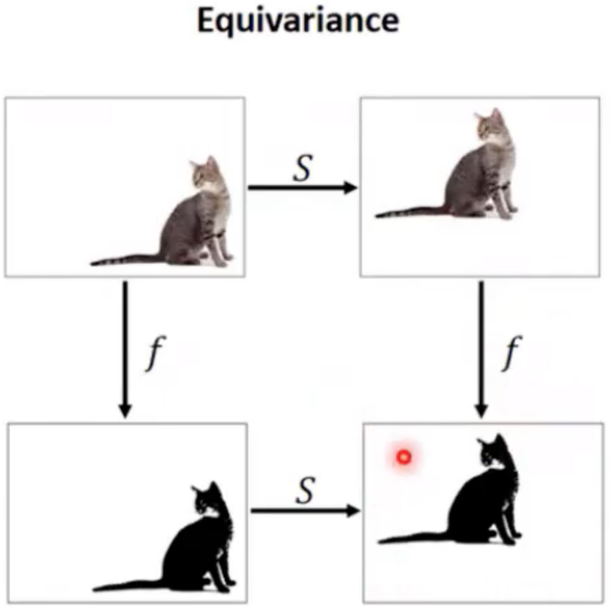
- *Parameter sharing: In a CNN, the same parameter acts on various parts of the network (dark arrows). In a DNN, each parameter is used once. While DNN parameters learn tasks related to specific inputs, a CNN parameter learns across the full image*
  - *Sharing weights has computational advantages, e.g.,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



Why do CNN work?

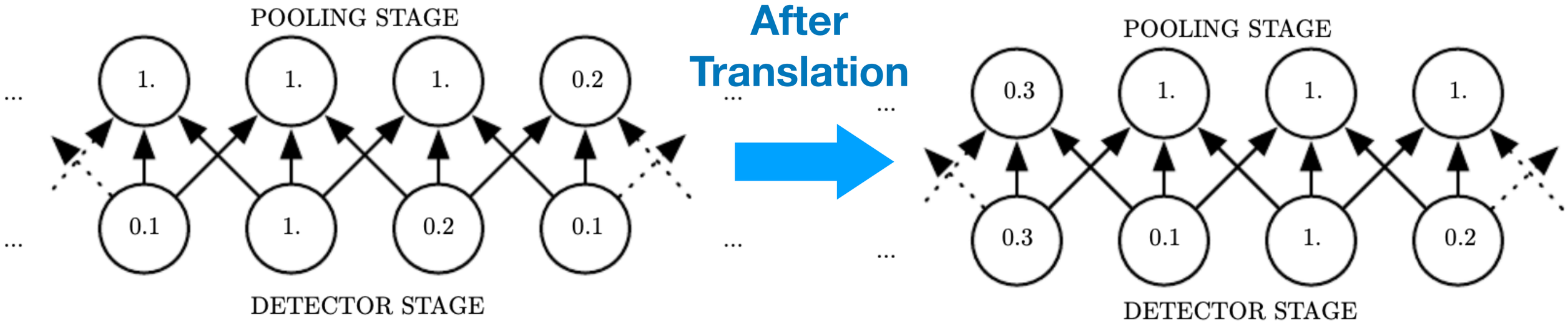
- *Equivariant representation: thanks to parameter sharing, CNN are equivariant to translation: if the input is shifted, so is the output*
- *A function* $f(x)$ *is equivariant to a function* $g(x)$ *if* $f(g(x)) = g(f(x))$
- *For CNN*
  - *Consider the operation* $S(i) = i - 1$ *that shifts the each 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 makes 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 minimum of the loss*





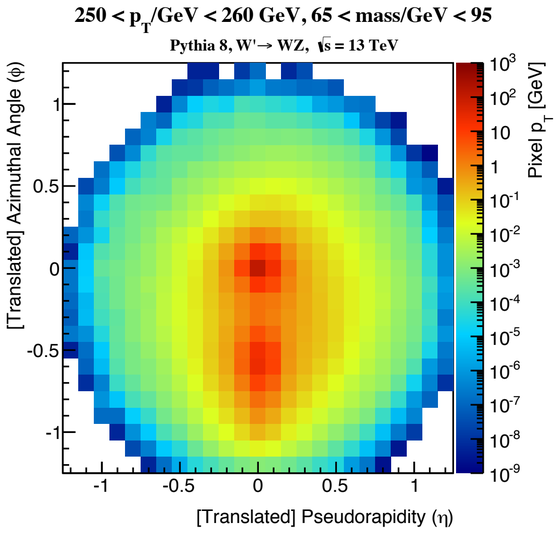
Approximate Invariance

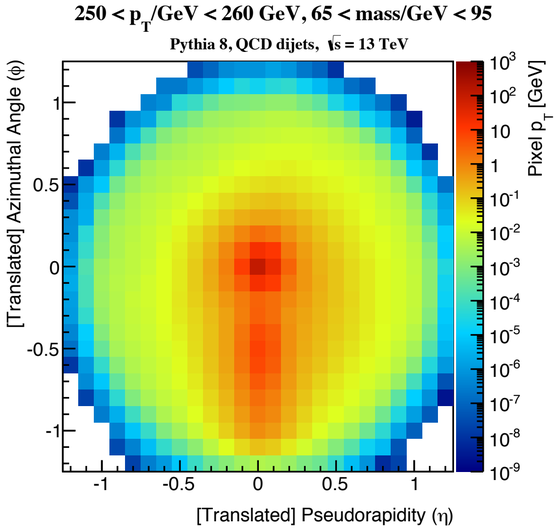
- *With pooling, network equivariant is promoted to approximate invariance: the output is about the same for small translations*
  - *See the example of MaxPooling*
- *This is imposing a strong prior via architecture choice to the learning process*



Pooling Choice

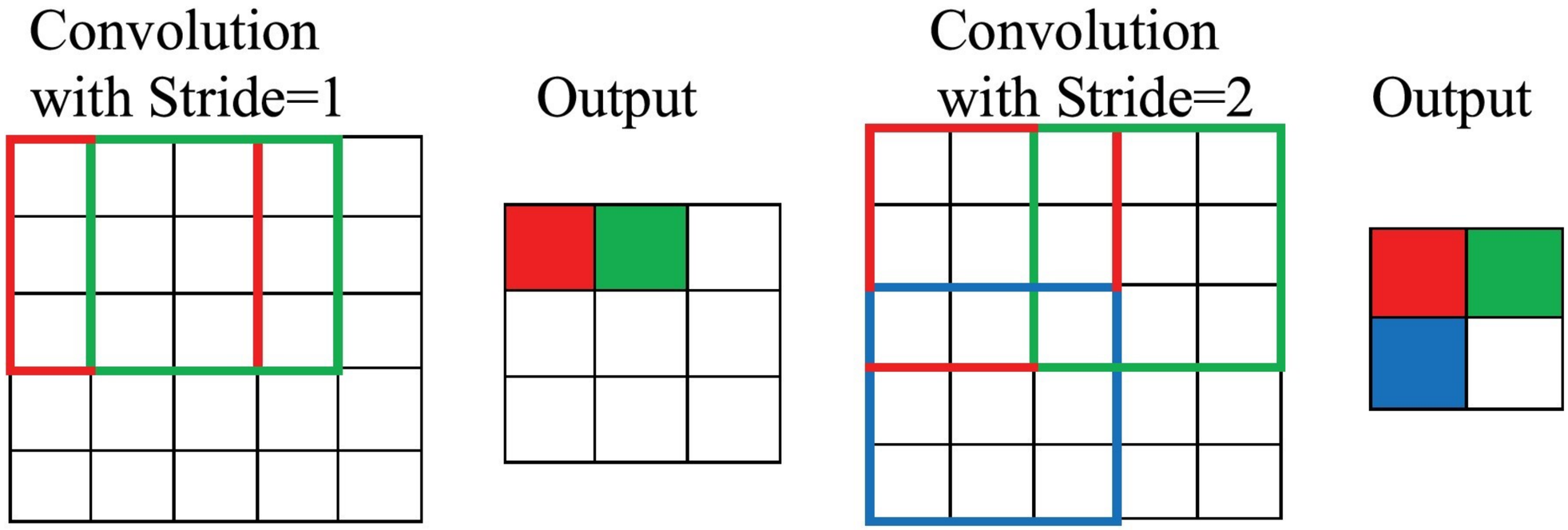
- *A strong prior could be bad if it goes against the problem nature*
  - *Example 1: try to estimate the energy of a particle from the energy shower left in your pixelated detector*
    - *Max pooling implies informations loss (some energy is not accounted for)*
    - *Average pooling makes more sense (all energy is accounted for)*
  - *Example 2: try to count how many overlapping showers are detected*
    - *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 exceed 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*

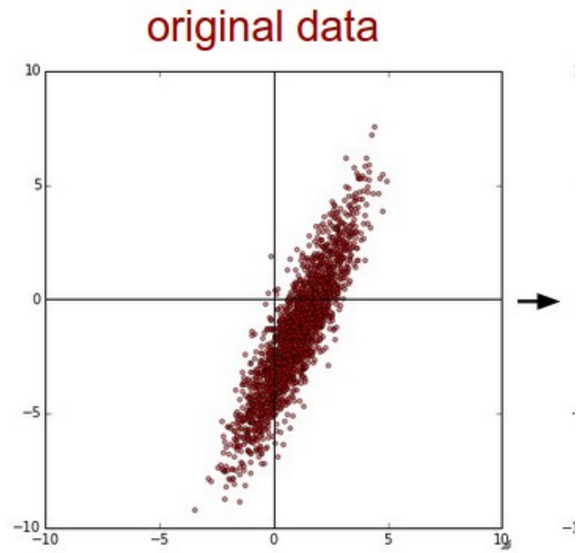
0	3	5	6	2	4
7	4	7	3	6	3
9	1	2	1	9	6
9	2	1	1	7	3
8	0	4	7	6	8
8	3	4	5	5	3

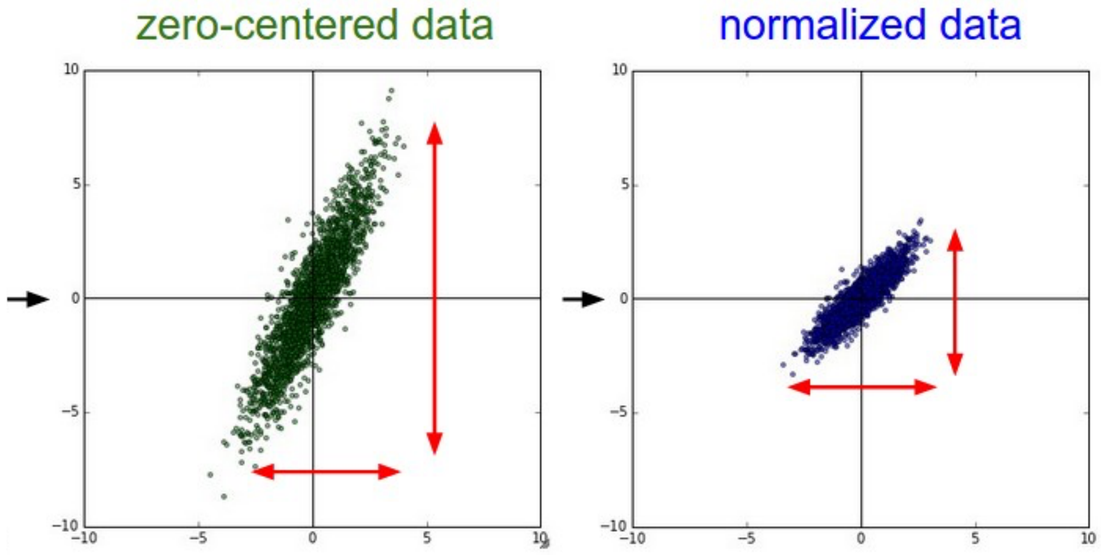
0	3	5	6	2	4	4
7	4	7	3	6	3	3
9	1	2	1	9	6	6
9	2	1	1	7	3	3
8	0	4	7	6	8	8
8	3	4	5	5	3	3
8	3	4	5	5	3	3

0	3	5	6	2	4	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

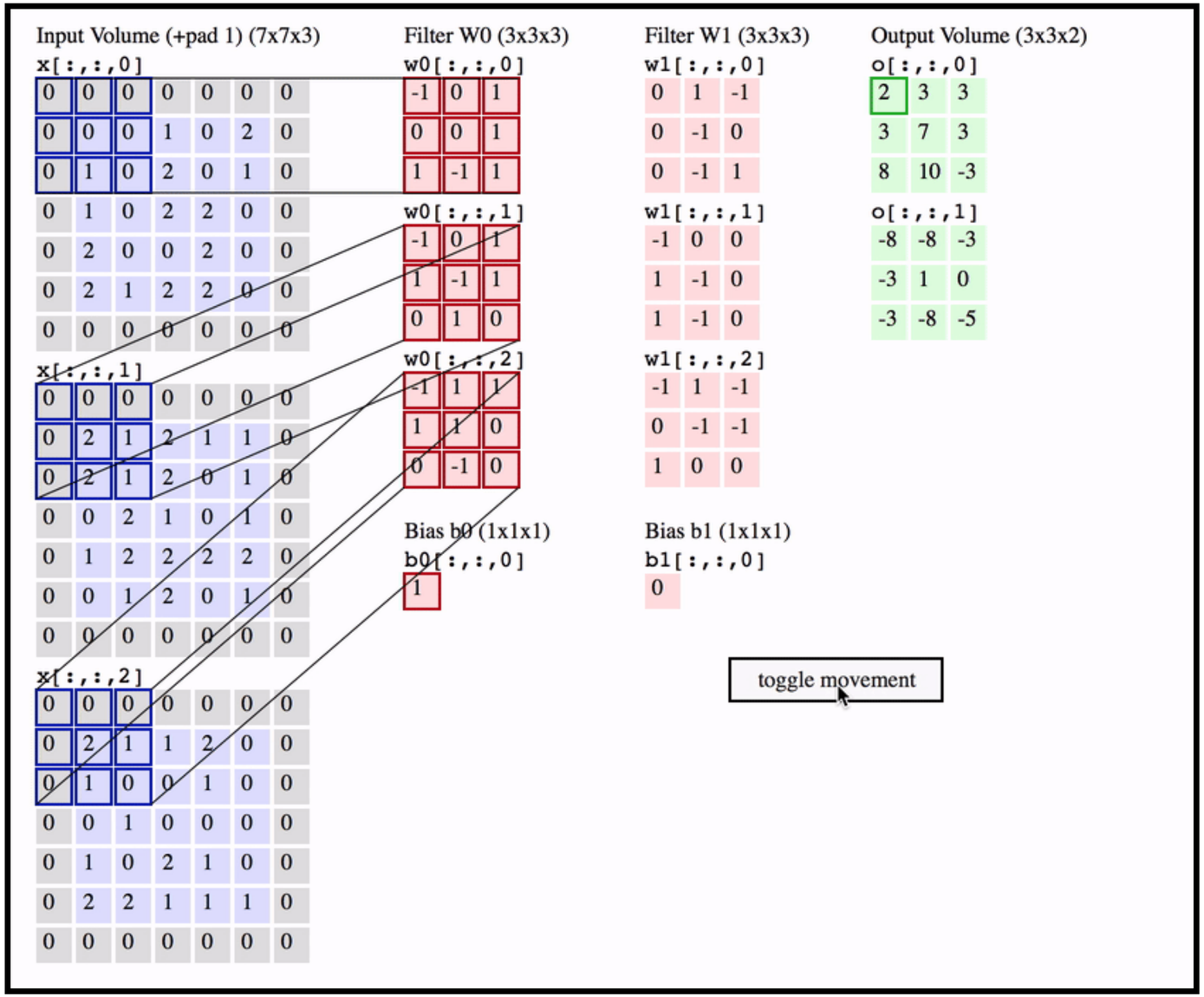
Batch Normalization

- *It is good practice to give normalized inputs to a layer*
  - *With all inputs having the same order of magnitude, all weights are equal 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*
  - *When this is used, bias is not needed (you might have noticed that CNNs have no bias parameters)*



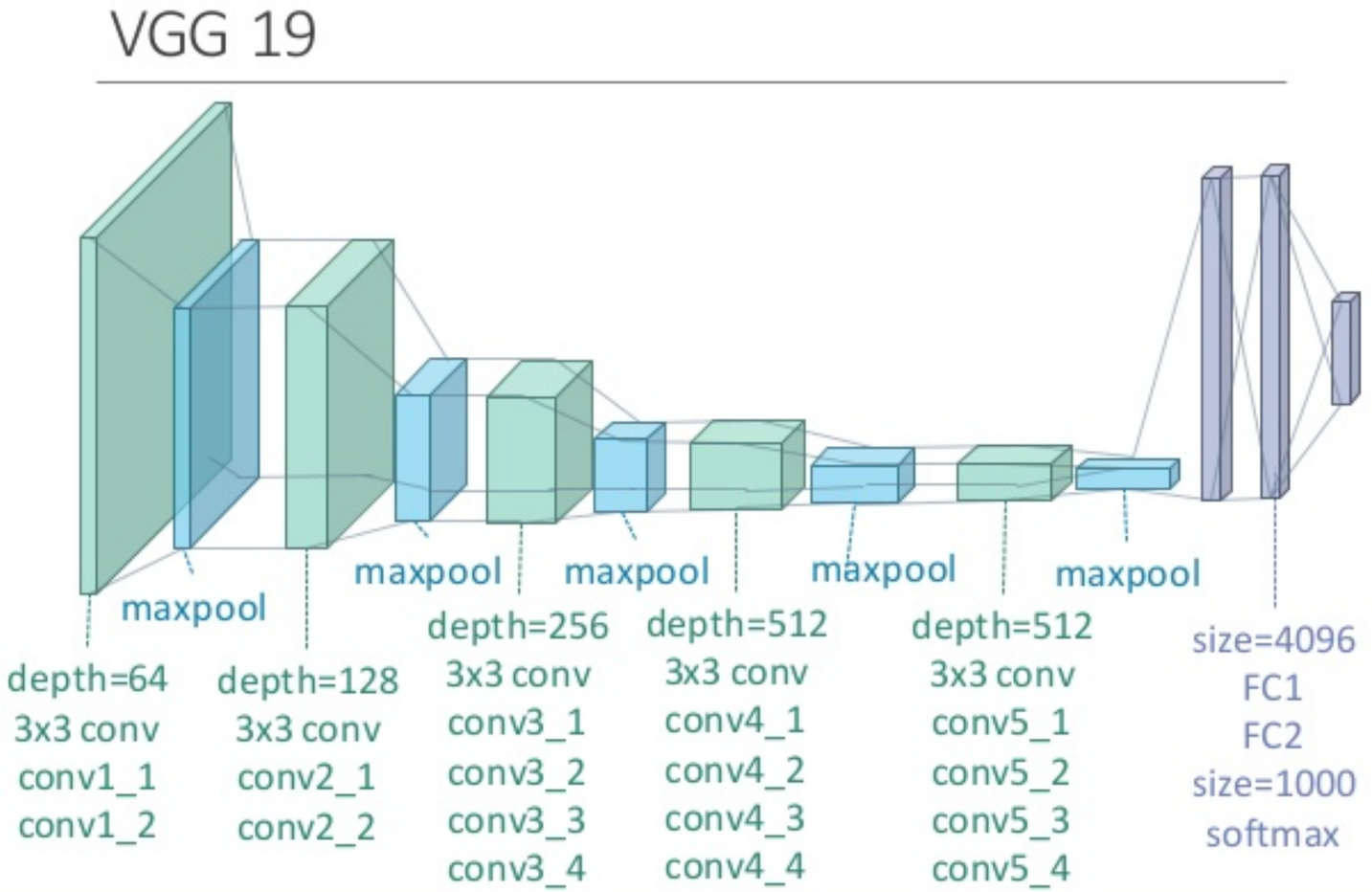


What CNN processing looks like...



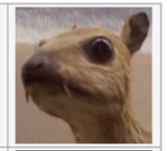
The full network

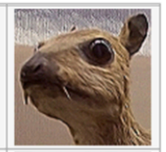
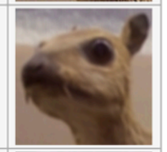
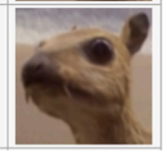
- *A full ConvNN is a sequence of Con2D+Pooling (+BatchNormalization +Dropout) layers*
- *The Conv+Pooling layer reduces the 2D image representation*
- *The use of multiple filters on the image makes the output grow on the third dimension (channels)*
- *One can flatten the output at some point and pass the result to a dense layer*



What does a ConvNN learn?

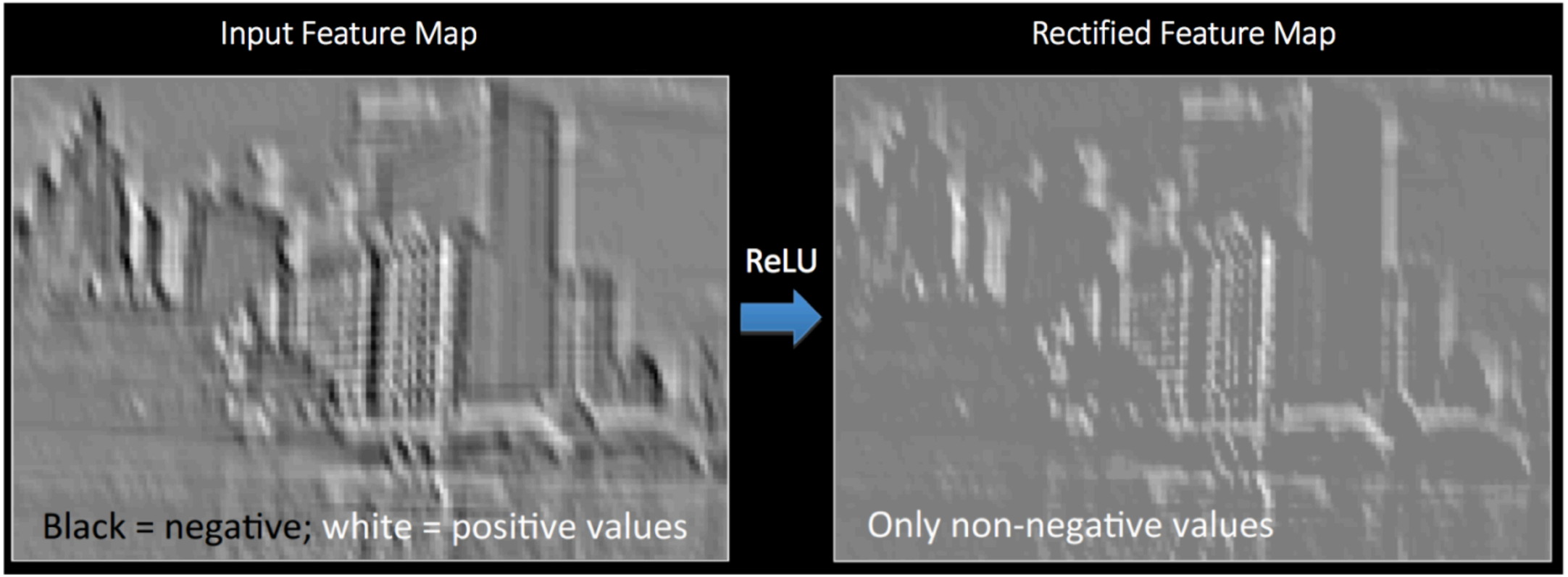
- *Each filter alters the image in a different way, picking up different aspects of the image*
  - *edges oriented in various ways*
  - *enhancing / blur of certain features*
- *It is interesting to check what each filter is doing*

Operation	Filter	Convolved Image
Identity	$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$	
Edge detection	$\begin{bmatrix} 1 & 0 & -1 \\ 0 & 0 & 0 \\ -1 & 0 & 1 \end{bmatrix}$	
	$\begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$	
	$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$	

Operation	Filter	Convolved Image
Sharpen	$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$	
Box blur (normalized)	$\frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$	
Gaussian blur (approximation)	$\frac{1}{16}\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$	

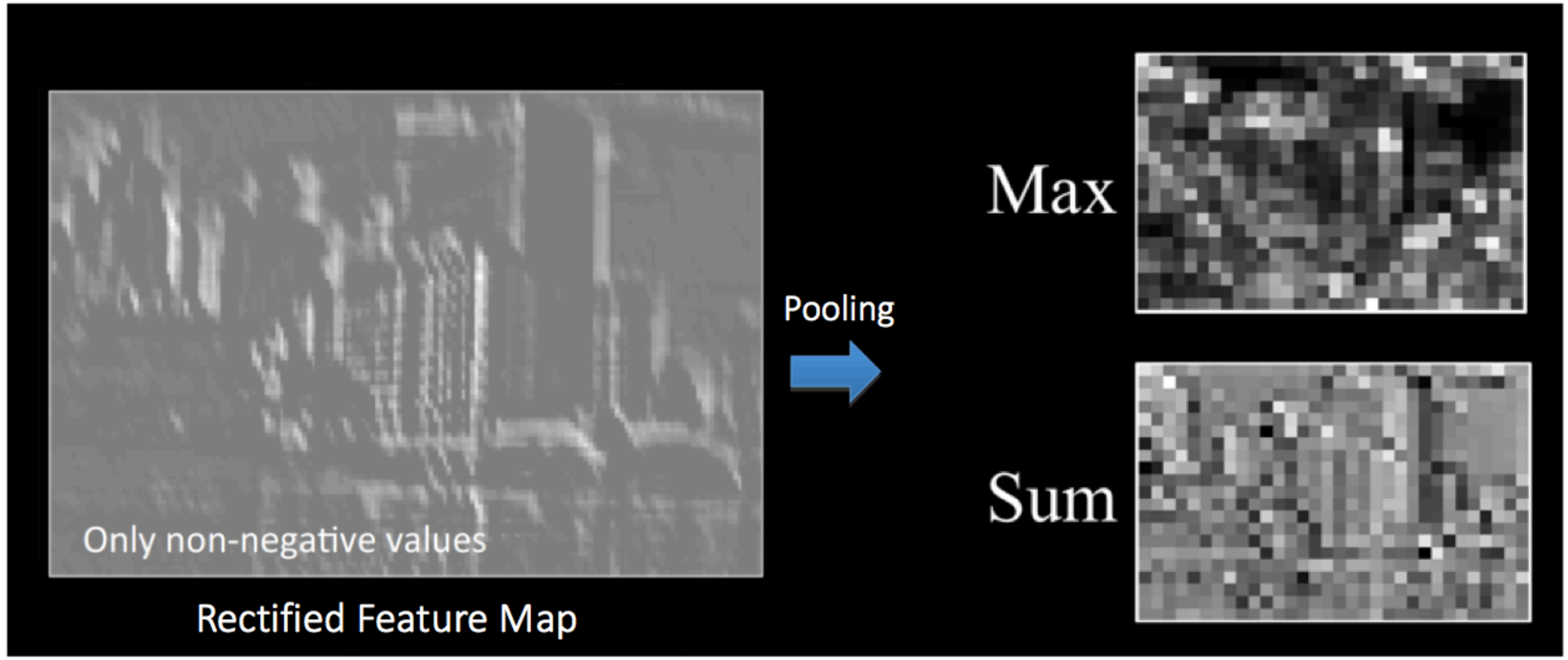
Check this blog entry on ConvNNs

What does a ConvNN learn?



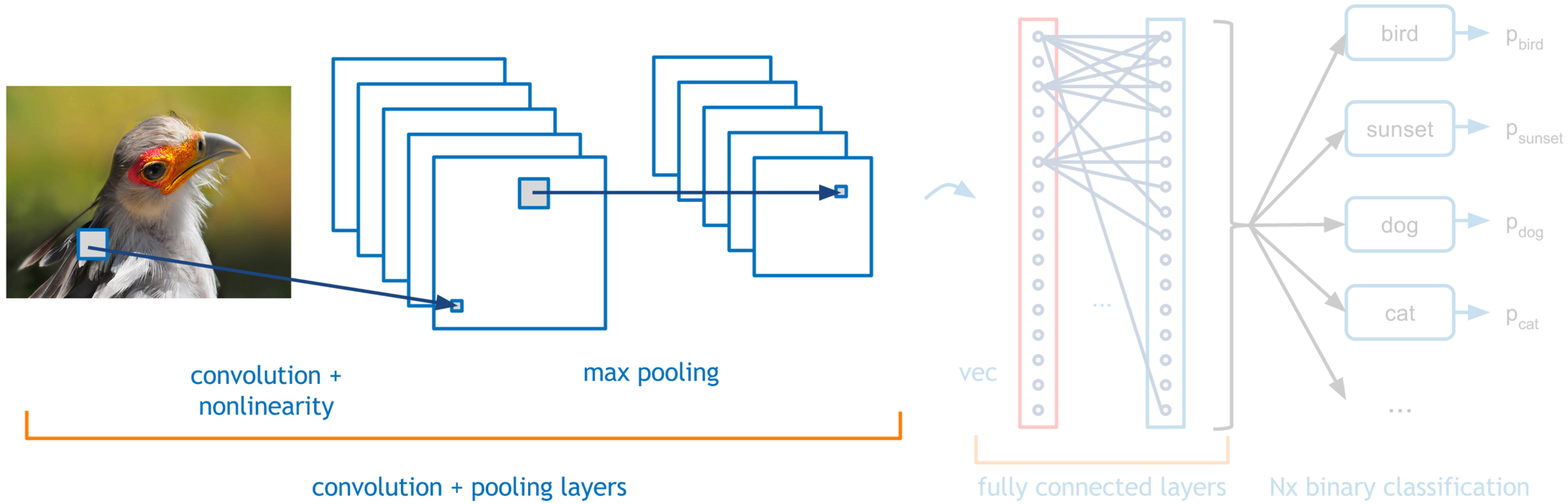
What does a ConvNN learn?

- *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)*



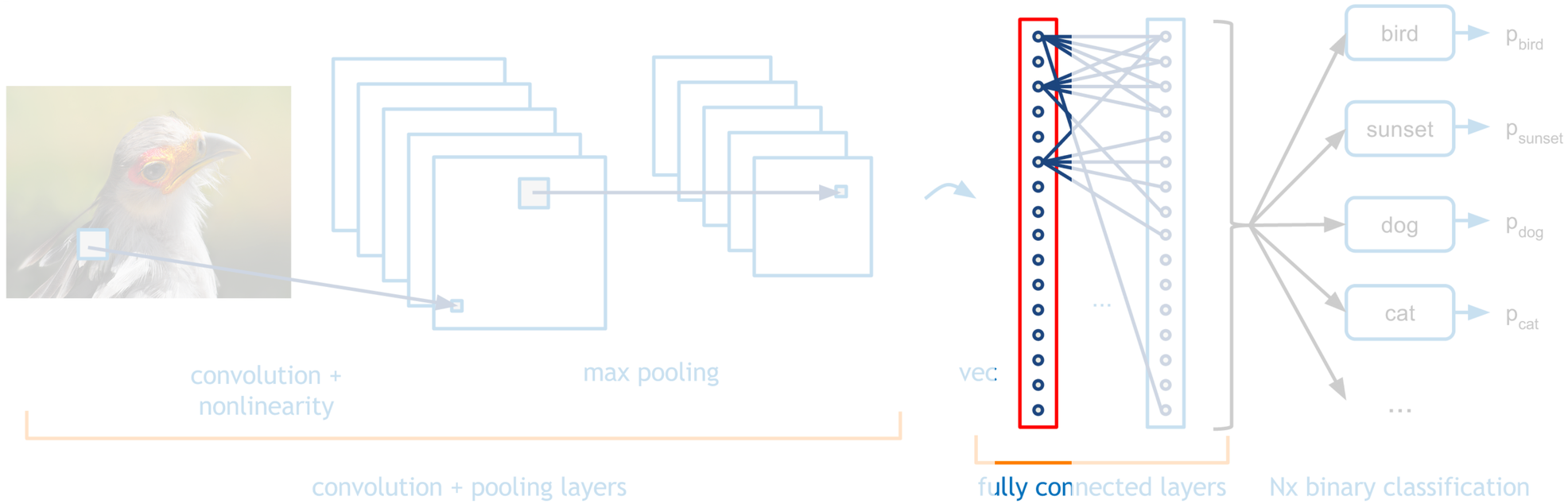
Two tasks in one network

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



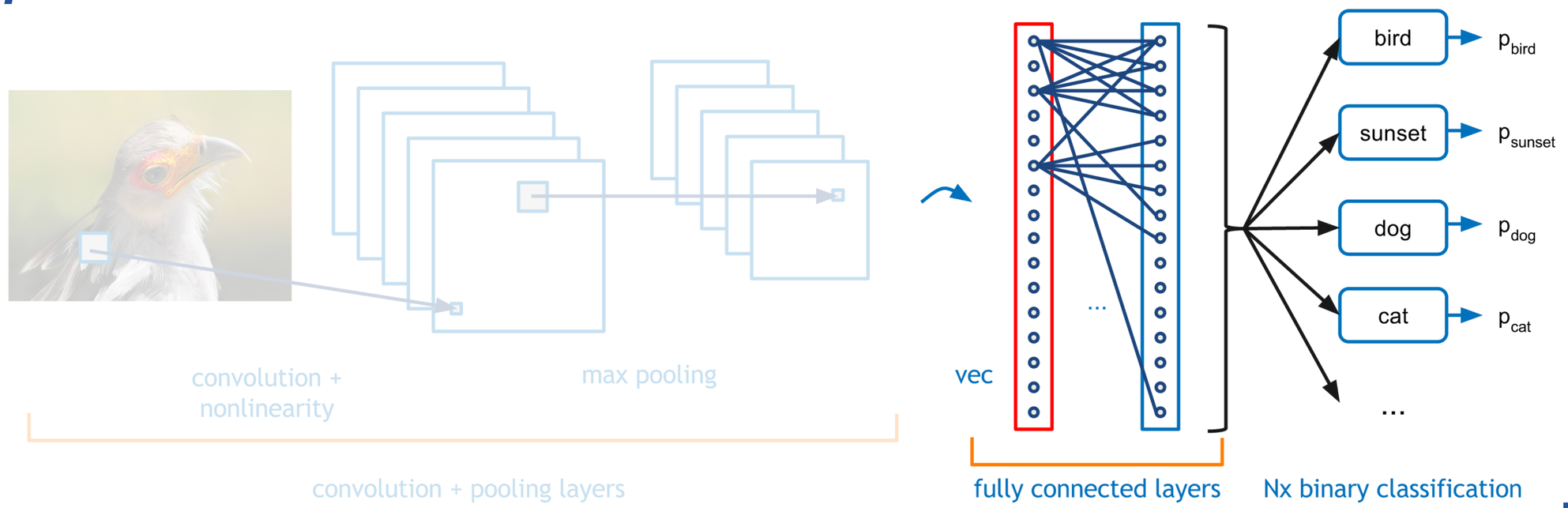
Two tasks in one network

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



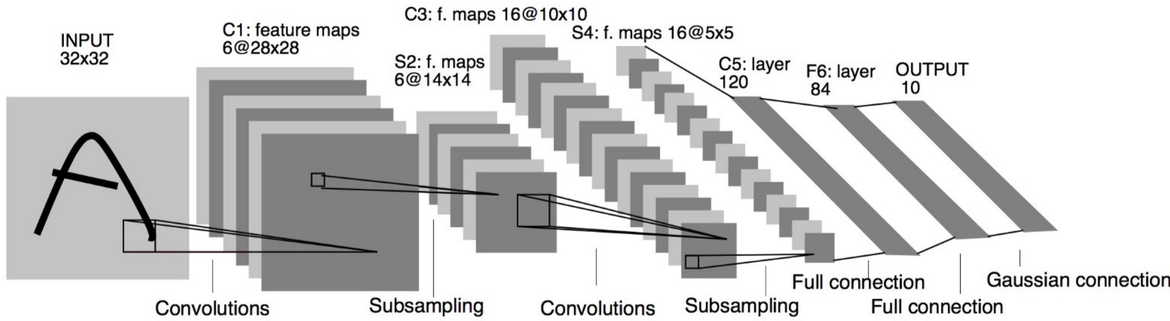
Two tasks in one network

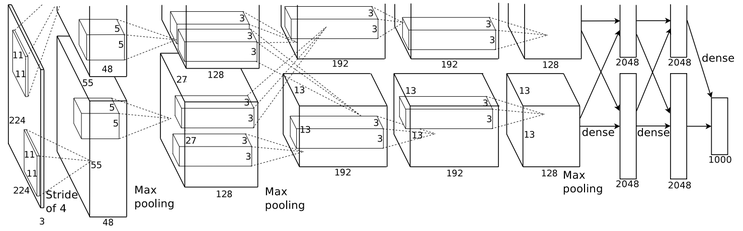
- *The Conv layer stars from RAW data and defines interesting quantities (high level features, HLF)*
- *The HLFs replace the physics-motivated inputs of a DNN*
- ***The DNN at the end exploits the engineered features to accomplish the task***

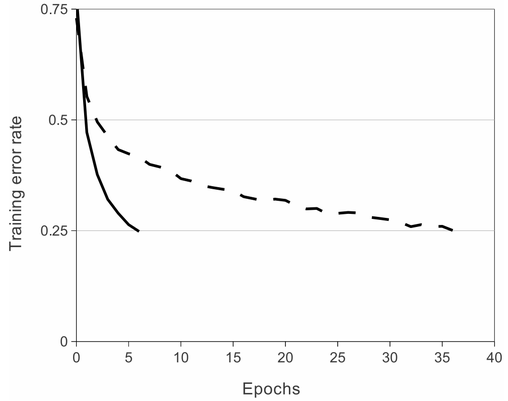


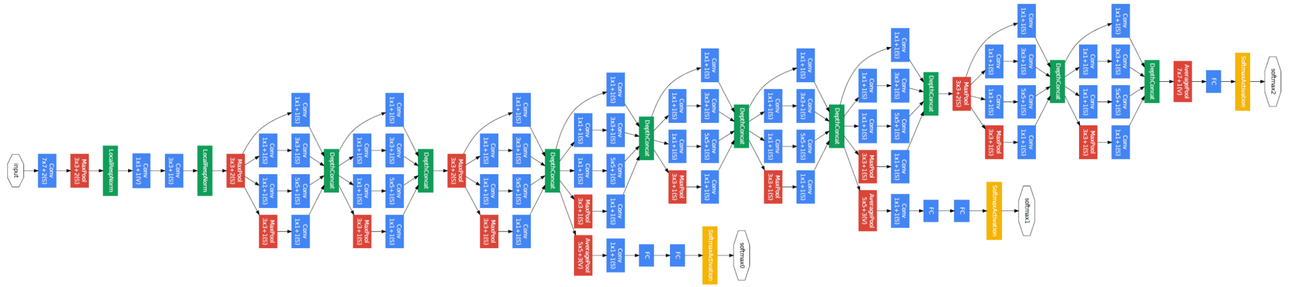
A history of ConvNNs

- *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 stage of the art: trained on GPUs, using ReLU and Dropout*
- *GoogleNet (2014): built on AlexNet, introduced an inception model to reduce e the number of parameters*



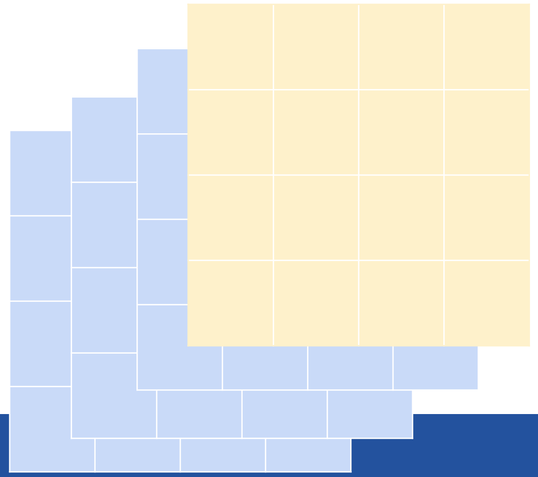


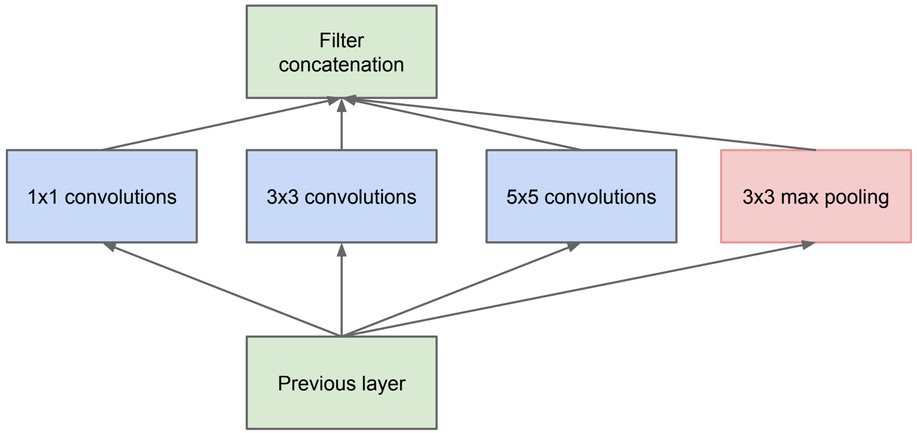




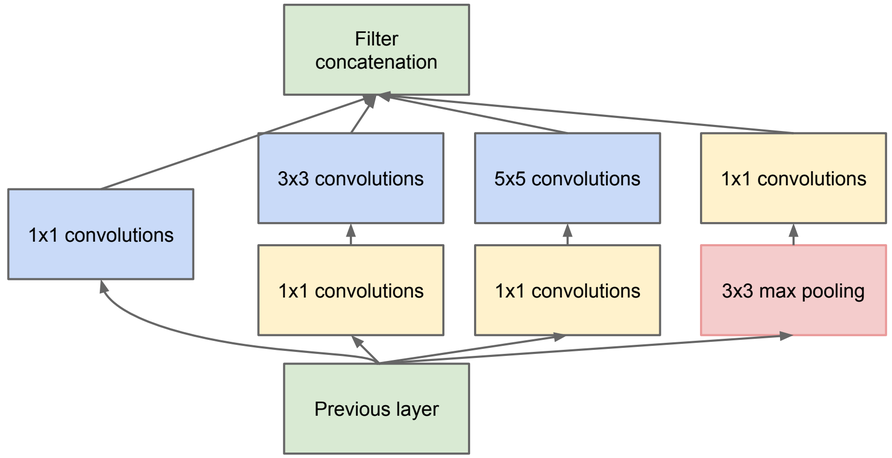
Inception Module

- *Rather than going deeper and deeper, inception architectures 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) Inception module, naïve version

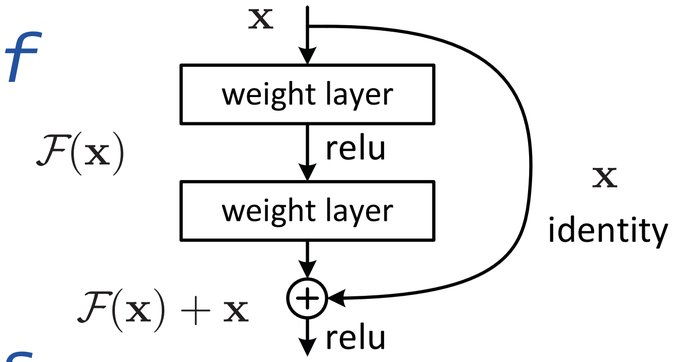


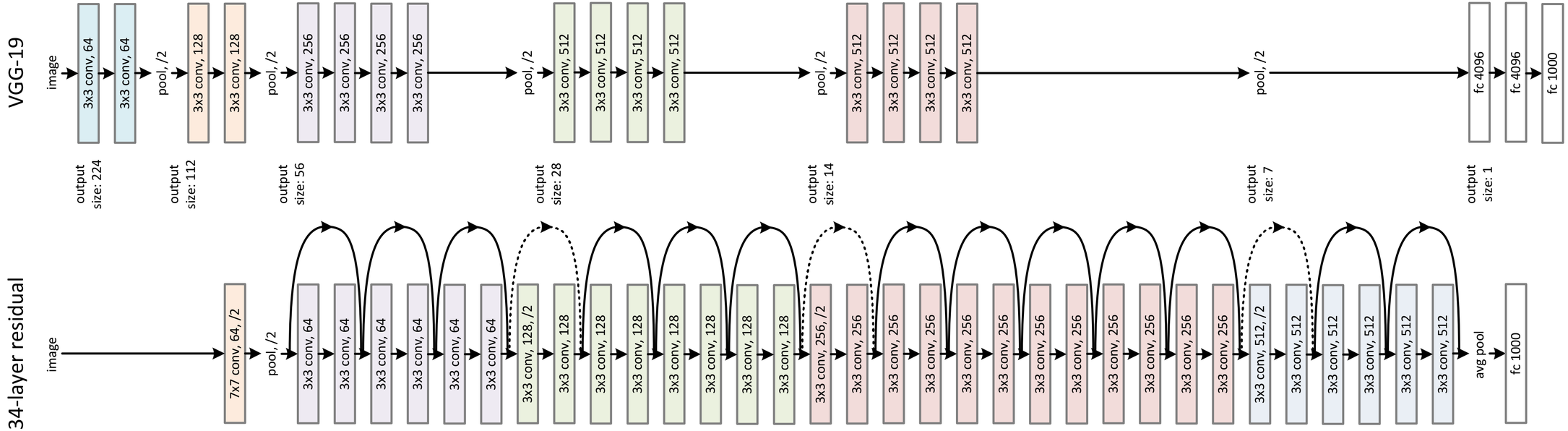
(b) Inception module with dimension reductions

A history of ConvNNs

- *VGGNet(2014):exploit small filters (3x3, 1x1), previously considered not optimal in a stack of Conv layers*

- *ResNet (2015):implemented skip connections, which allows to focus the learning on residuals (difference between inputs and outputs)*

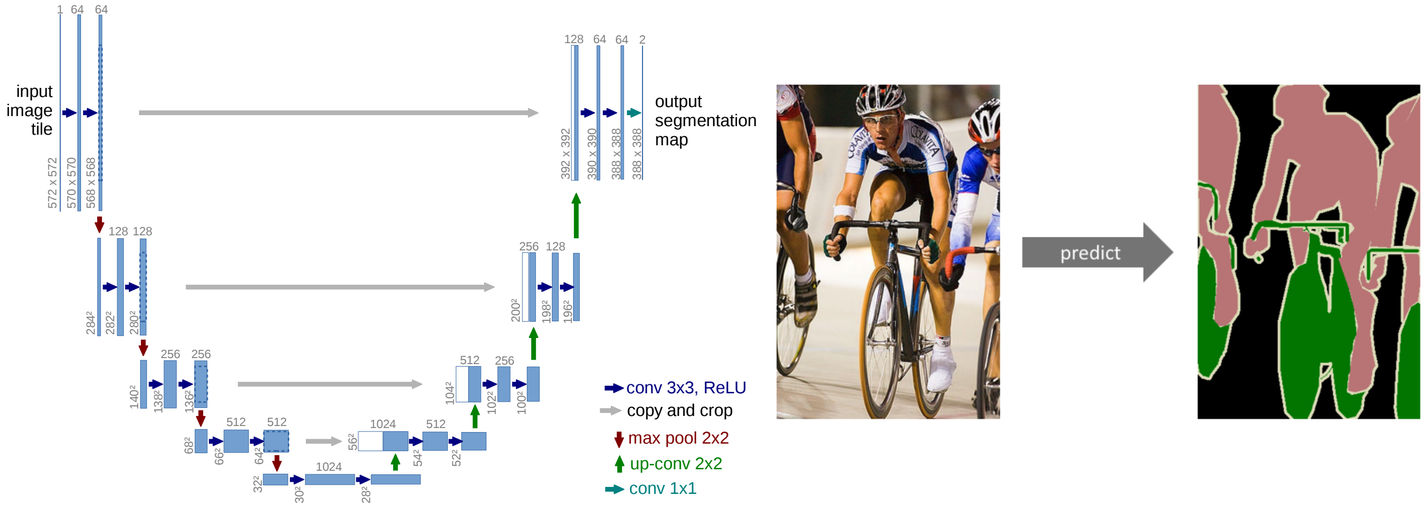




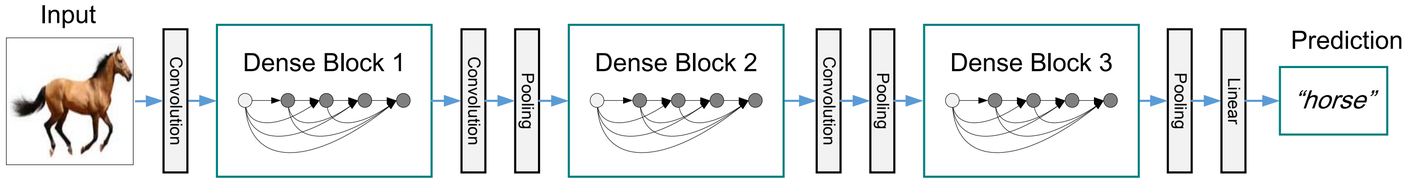
Check this blog entry on ConvNNs

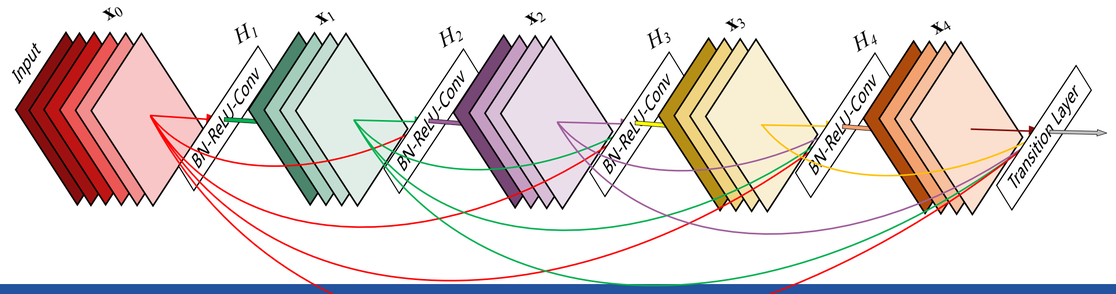
A history of ConvNNs

- *U-net (2015): conv layers with skip connections, in 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*

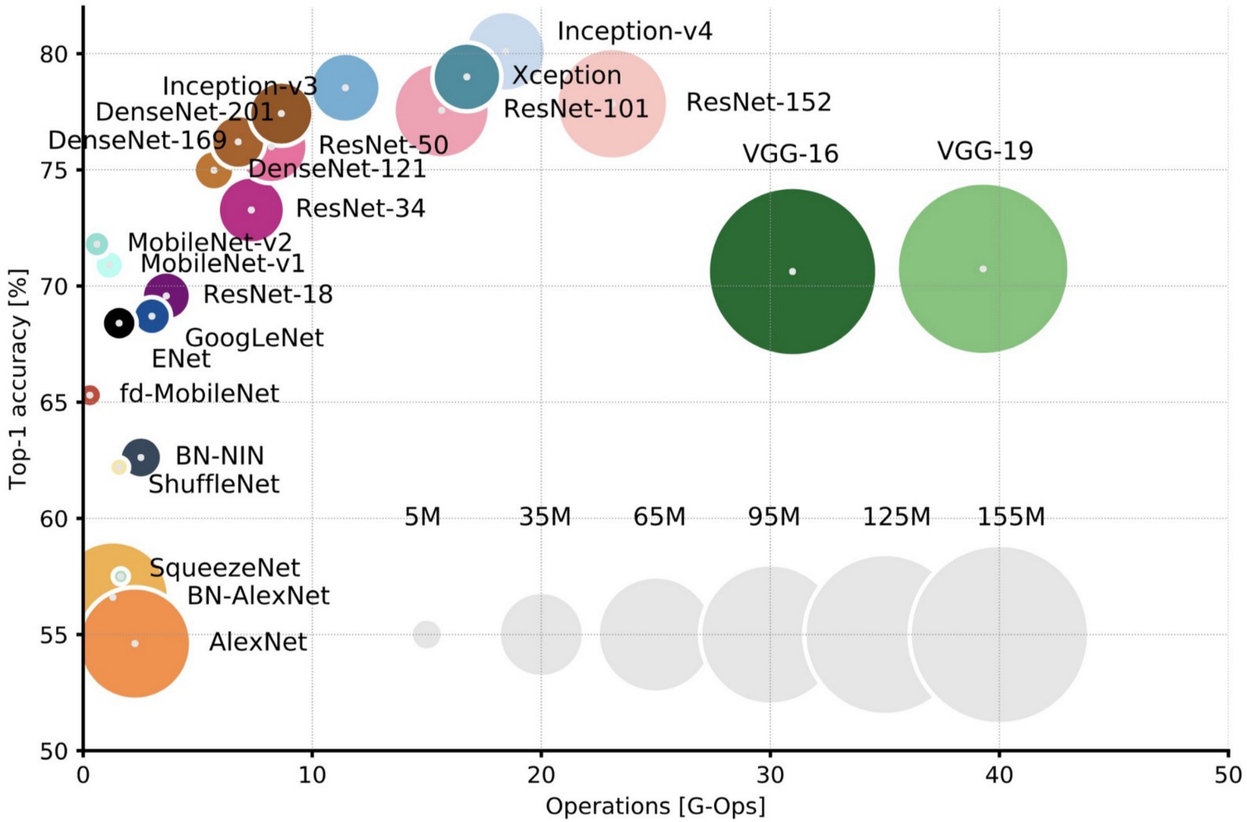




The computational cost

- *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

Linear Algebra

Linear Algebra in a Nutshell

- *Scalars: plain numbers (integer, real, etc)*
- *Vectors: ordered arrays of numbers. The i-th element of the vector x is labelled x_i*
- *Matrices: ordered table of numbers. The element A_{ij} of the matrix A occupies the i-th row and the the j-th column*
- *Tensors: matrix generalization to more than two dimensions. Similar notation as for matrices (A_{ijk} occupies the i-th position on first dimension, j-th position on second dimension, and k-th position on third dimension)*

$$\boldsymbol{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}$$

$$\begin{bmatrix} A_{1,1} & A_{1,2} \\ A_{2,1} & A_{2,2} \end{bmatrix}$$

Scalar, Vector, and Matrix operations

- *scalar matrix sum: sum each element of the matrix A to the scalar b*
  $(b + A)_{ij} = A_{ij} + b$

- *scalar matrix product: multiply each element of the matrix A by the scalar b*
  $(b \cdot A)_{ij} = bA_{ij}$

- *Vector transpose: turn a (column) vector x in a (row) vector x^{T}*

- *Matrix transpose: invert A_{ij} with A_{ji} for each i and j*

- *Matrix sum: $(A + B)_{ij} = A_{ij} + B_{ij}$*

$$2 \cdot \begin{bmatrix} 10 & 6 \\ 4 & 3 \end{bmatrix} = \begin{bmatrix} 2 \cdot 10 & 2 \cdot 6 \\ 2 \cdot 4 & 2 \cdot 3 \end{bmatrix}$$

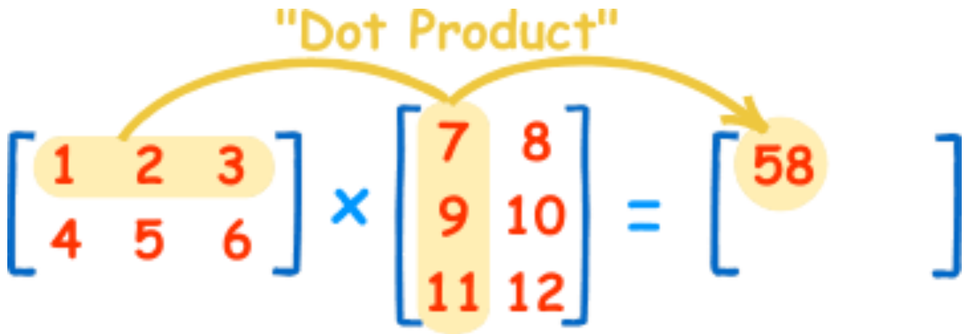
$$\begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_m \end{bmatrix}^{\mathrm{T}} = [\, x_1 \; x_2 \; \dots \; x_m \,].$$

Scalar, Vector, and Matrix operations

- *(same dimension) vector inner product* $xy = \sum x_i y_i$ *returns a scalar*
- *vector outer product* $(x^\top y)_{ij} = x_i y_j$ *returns a matrix*
- *Matrix product:* $(A \times B)_{ij} = \sum_k A_{ik} B_{kj}$
- *Hadamard product: multiply elements in corresponding position (for matrices of the same size)* $(A \quad B)_{ij} = A_{ij} B_{ij}$

$$\mathbf{a} = \begin{bmatrix} v \\ w \end{bmatrix} \qquad \mathbf{b} = \begin{bmatrix} x \\ y \\ z \end{bmatrix}$$

$$\mathbf{C} = \mathbf{ab'} = \begin{bmatrix} v*x & v*y & v*z \\ w*x & w*y & w*z \end{bmatrix}$$



Scalar, Vector, and Matrix operations

- Matrix product
  - is distributive $A(B+C) = AB + AC$
  - is associative $A(BC) = (AB)C$
  - is not commutative $AB \neq BA$
- Transpose of a product: $(AB)^\top = B^\top A^\top$
- Inverse of a matrix $A^{-1} : A^{-1}A = \mathbb{I}$, where $\mathbb{I}$ is the identity matrix
  - Not all matrices have an inverse
  - When they do, one can solve a set of linear equation $Ax = b \implies x = A^{-1}b$

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

Vector Norms

- A norm is a function f of a vector x such that
  - $f(x) = 0 \implies x = 0$
  - $f(x + y) \leq f(x) + f(y)$
  - $\forall \alpha \in \mathbb{R} f(\alpha x) = |\alpha| f(x)$
- In many occasions (e.g., when defining loss functions for a regression) we will deal with L^p norms

$$||x||_p = \left(\sum_i |x_i|^p \right)^{1/p}$$

  - for p=2 we obtain the Euclidean norm
  - ofter p=1 is used, because it grows equally faster near and far from zero (useful for minimisation convergence)

Special kinds of matrices

- *Diagonal: matrix with all elements out of diagonal equal to zero. Notation:* $diag(v)$ *= diagonal matrix with elements on diagonal given by vector v*
- *Symmetric: a matrix equal to its transpose (*$A = A^{\top}$*), i.e., with* $A_{ij} = A_{ji}$
- *Orthogonal:* $A^{-1} = A^{\top}$*. An orthogonal matrix has rows and columns which are mutually orthonormal, i.e., the corresponding vectors are orthogonal and with norm = 1*

Eigendecomposition

- An eigenvector v of A is such that $Av = \lambda v$, where λ is the eigenvalue of A. Applying A on v just rescales it by λ

- One can demonstrate that $A = V\,\mathrm{diag}(\lambda)\,V^{-1}$, where V is the matrix constructed such that the i-th column is the eigenvector $v^{(i)}$ and $\mathrm{diag}(\lambda)$ is the diagonal matrix with the eigenvalue $\lambda^{(i)}$ at the (i,i) position

- Any eigedecomposed matrix can be inverted. If one defines $A^{-1} = V\Lambda^{-1}V^{-1}$, with $(\Lambda^{-1})_{i,i} = \frac{1}{\lambda^{(i)}}$ and $(\Lambda^{-1})_{i,j} = 0$ for $i \neq j$, it is easy to show that:

$$A^{-1}A = V\Lambda^{-1}V^{-1}V\,\mathrm{diag}(\lambda)\,V^{-1} = V\Lambda^{-1}\,\mathrm{diag}(\lambda)\,V^{-1} = VV^{-1} = \mathbb{I}$$

- For a real symmetric matrix, the eigencomposition simplifies to $A = V\,\mathrm{diag}(\lambda)\,V^{\top}$

Important quantities

- *Determinant:*
  - *a scalar function of the entries of a matrix such that*
    - $\det(\mathbb{I}) = 1$
    - *The exchange of two rows multiplies the determinant by -1*
    - *Multiplying a row by a number multiplies the determinant by this number*
    - *Adding a multiple of one row to another row does not change the determinant*
  - *For a diagonalisable matrix, it is the product of the eigenvalues*

$$\begin{vmatrix} a & b \\ c & d \end{vmatrix} = ad - bc,$$

$$\begin{vmatrix} a & b & c \\ d & e & f \\ g & h & i \end{vmatrix} = aei + bfg + cdh - ceg - bdi - afh.$$

- *Trace:* $\mathrm{Tr}(A) = \sum_i A_{ii}$
  - *Invariant under transpose operation:* $\mathrm{Tr}(A) = \mathrm{Tr}(A^\top)$
  - *Invariant under cycling permutation:* $\mathrm{Tr}(AB) = \mathrm{Tr}(BA)$, $\mathrm{Tr}(ABC) = \mathrm{Tr}(CAB) = \mathrm{Tr}(BCA)$, *etc.*