

# Deep Learning Applications for collider physics

## Lecture 2

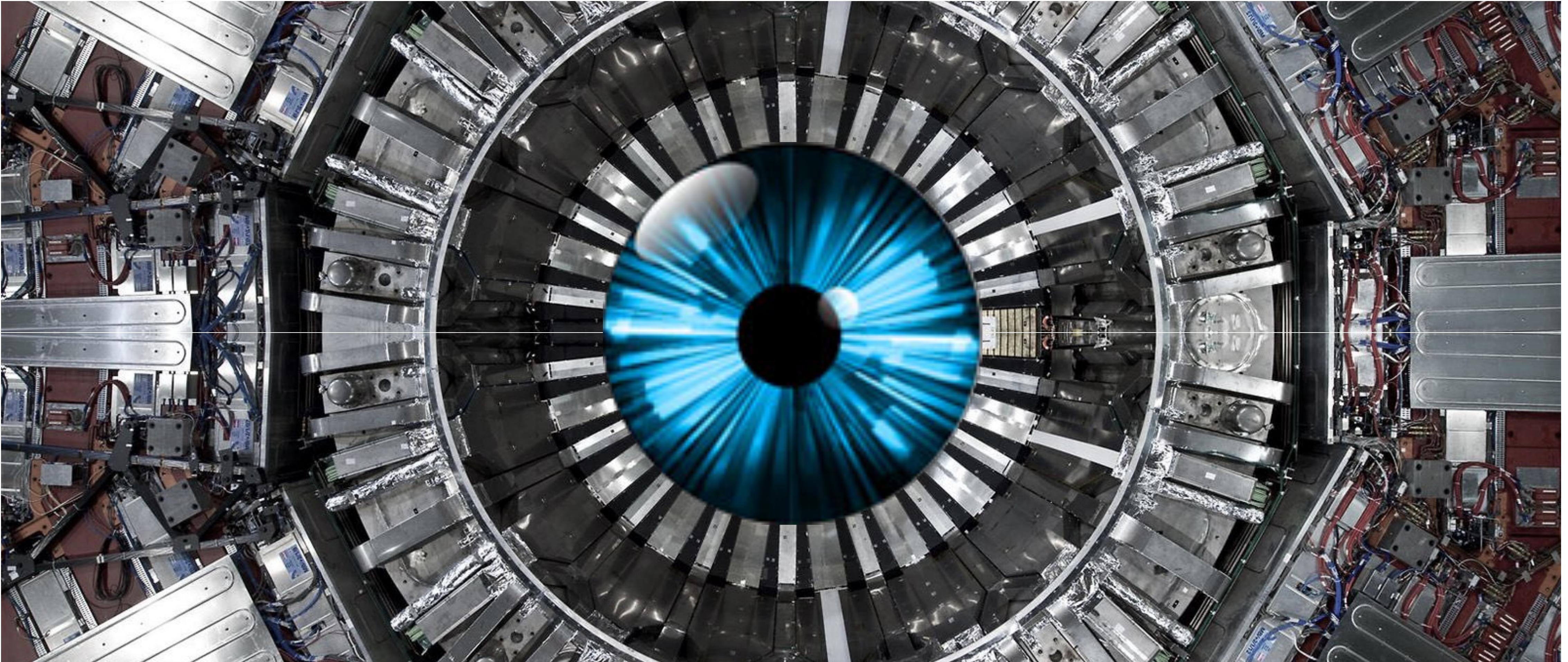
maurizio Pierini





# Plan for these lectures

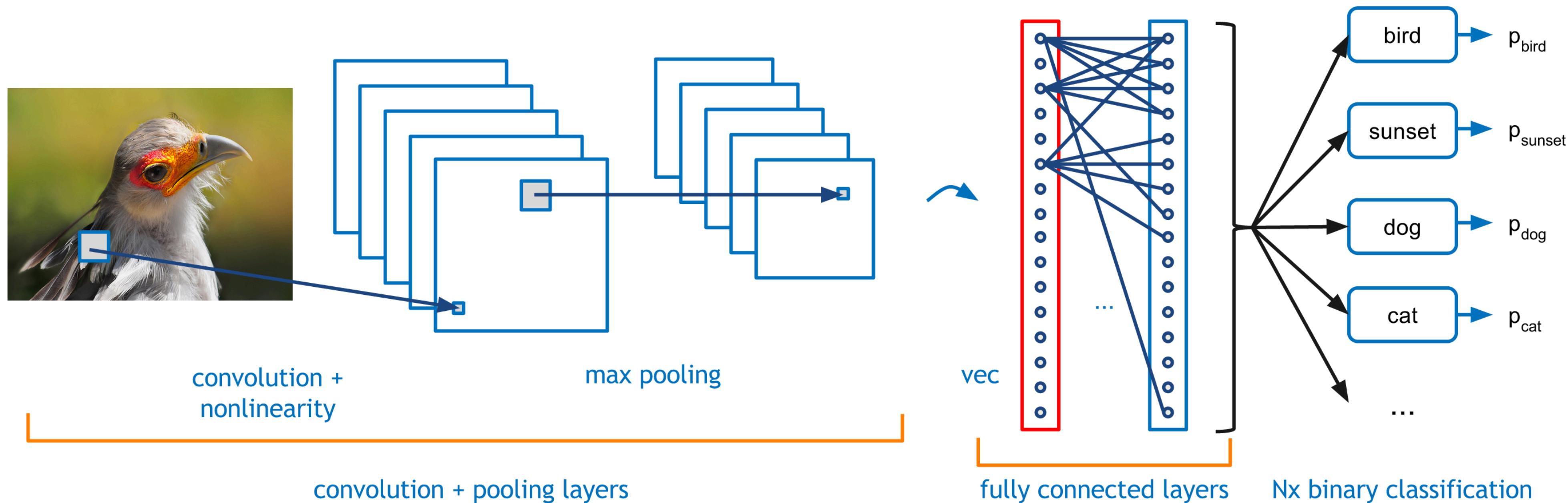
	Day1	Day2	Day3	Day4	Day5
1st hour	Introduction	ConvNN	LHC & fast Inference	Anomaly Detection	Graphs
2nd hour	Dense NNNs	GANs	RNNs	VAEs	Graphs
Tutorial	Dense NNNs	ConvNN	RNNs	AE	Graphs



# Particle Reconstruction & Computer Vision

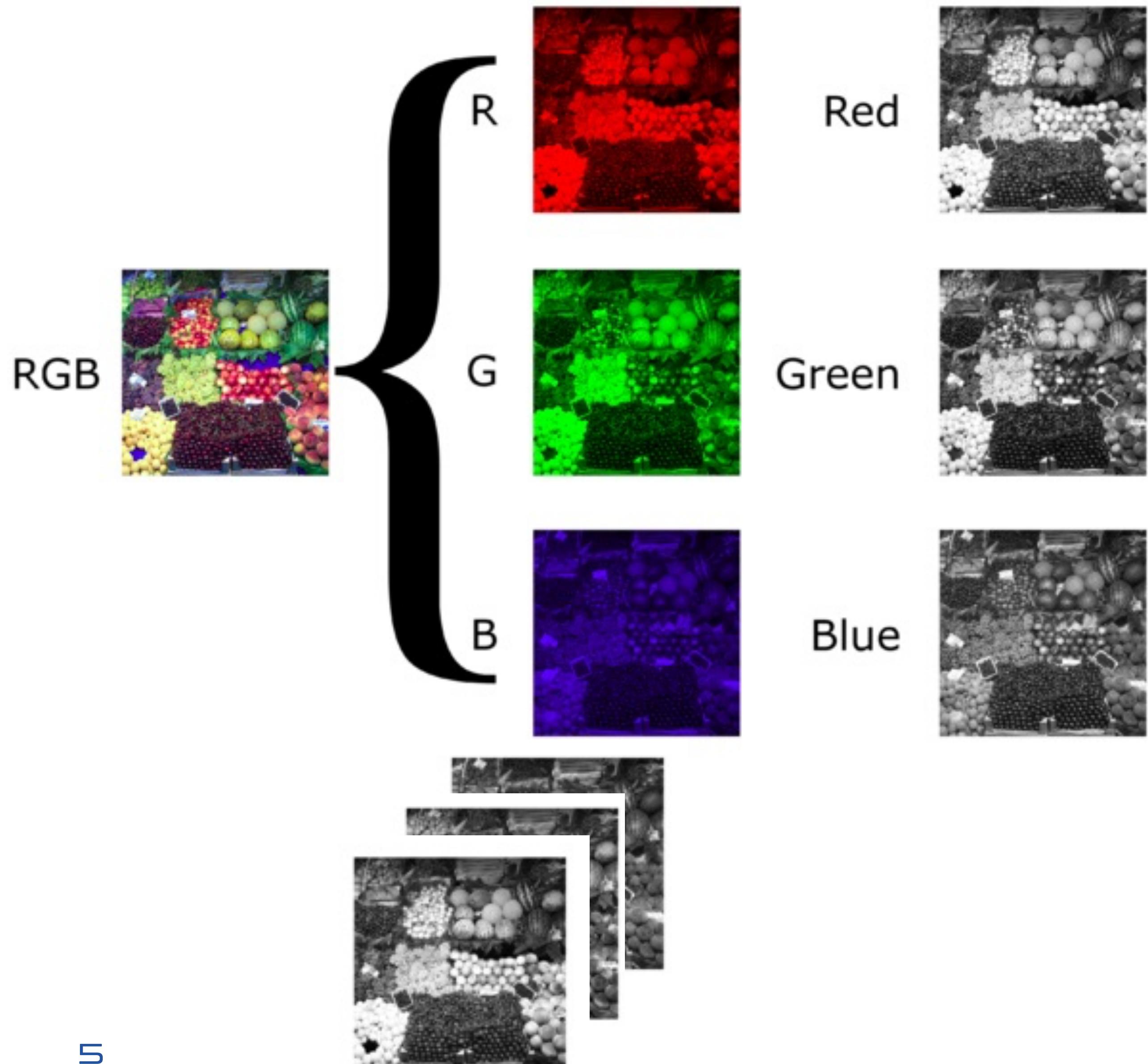
# Convolutional Layer

- Special architectures read the raw information (e.g., images) and convert them into “smart variables” (high-level features) to accomplish the task
- Typical example: convolutional neural networks for image processing & computing vision



# Digital images

- *Each image is a matrix of pixels*
- *Each pixel comes with a color*
- *In RGB scheme, these are three values  $i \in [0,1]$*
- *Each image becomes a 3D tensor*
- *3 channels of 2D pixelated images*



# Convolutional filter

- *The main ingredient of ConvNN is a filter, a  $k \times k'$  matrix of weights*
- *The filter scans the image and performs a scalar product of each image patch*
- *This results into a new matrix of values, with different dimensionality*

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

$$\begin{aligned}
 & 0 \times 4 - 3 \times 1 + 5 \times 4 + \\
 & -7 \times 2 + 4 \times 2 - 7 \times 5 + \\
 & 9 \times 3 + 1 \times 1 - 2 \times 6 = -8
 \end{aligned}$$



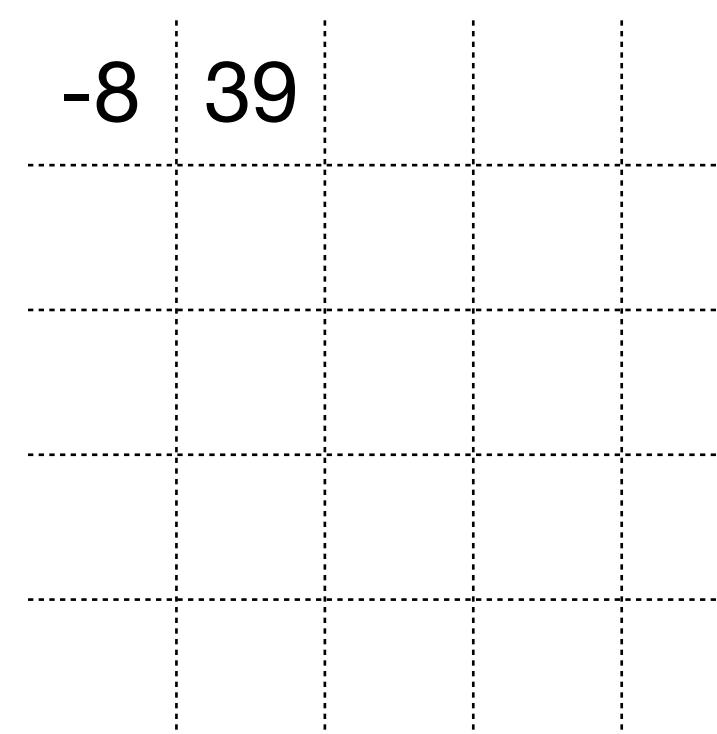
# Filtro convolutional

- The main ingredient of ConvNN is a filter, a  $k \times k'$  matrix of weights
- The filter scans the image and performs a scalar product of each image patch
- The scan is done shifting the filter by a stride (of  $q$ , 2, ... cells)
- This results into a new matrix of values, with different dimensionality

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

$$\begin{aligned}
 & 3 \times 4 - 5 \times 1 + 6 \times 4 + \\
 & -4 \times 2 + 7 \times 2 - 3 \times 5 + \\
 & 7 \times 3 + 2 \times 1 - 1 \times 6 = 39
 \end{aligned}$$



# 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	

- ...

# Padding

- When the filter arrived at the edge, it might exceeds it (if  $n/k$  is not an integer)

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

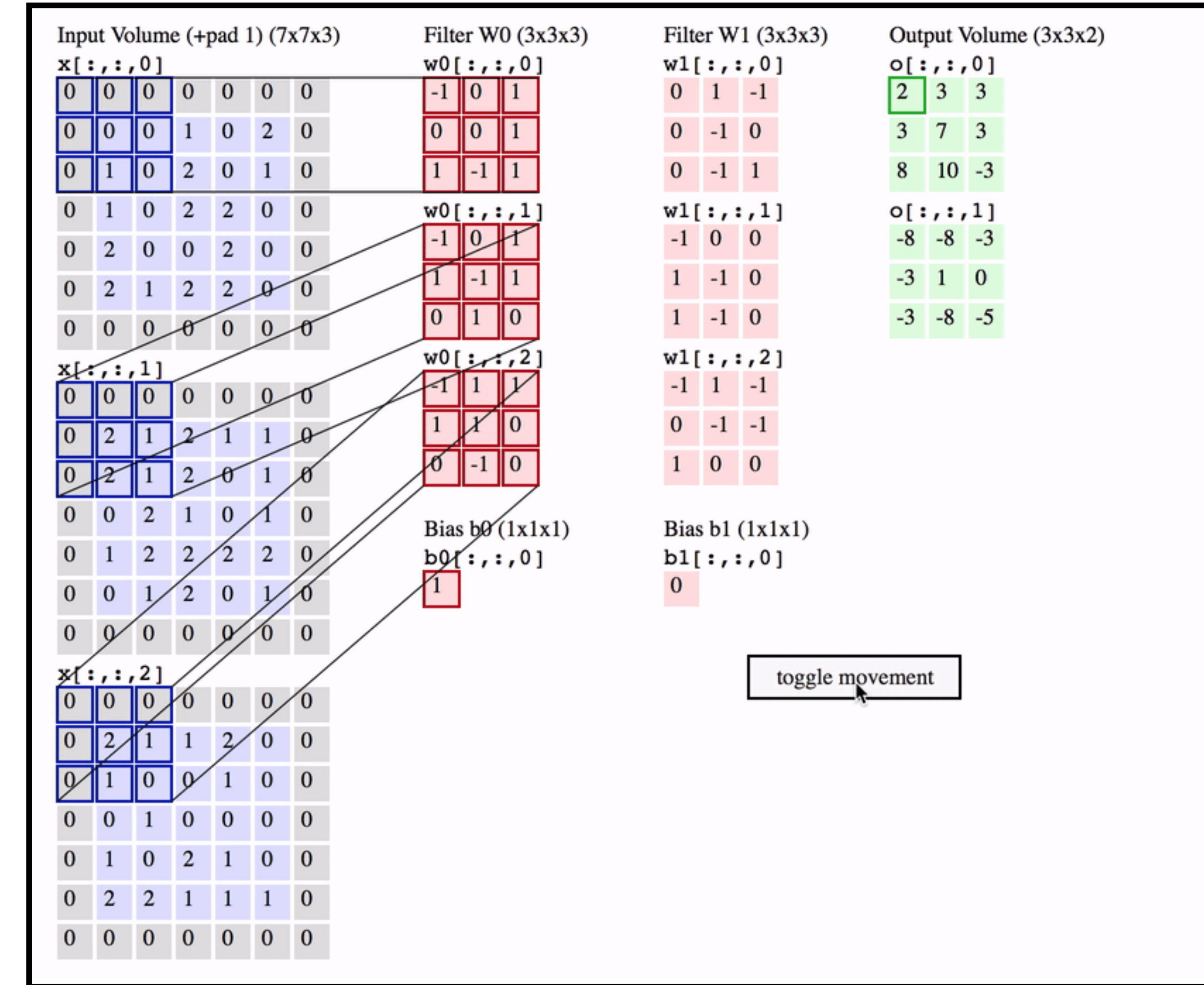
- In this case, a padding rule needs to be specified

- Same: repeat the values at the boundary

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

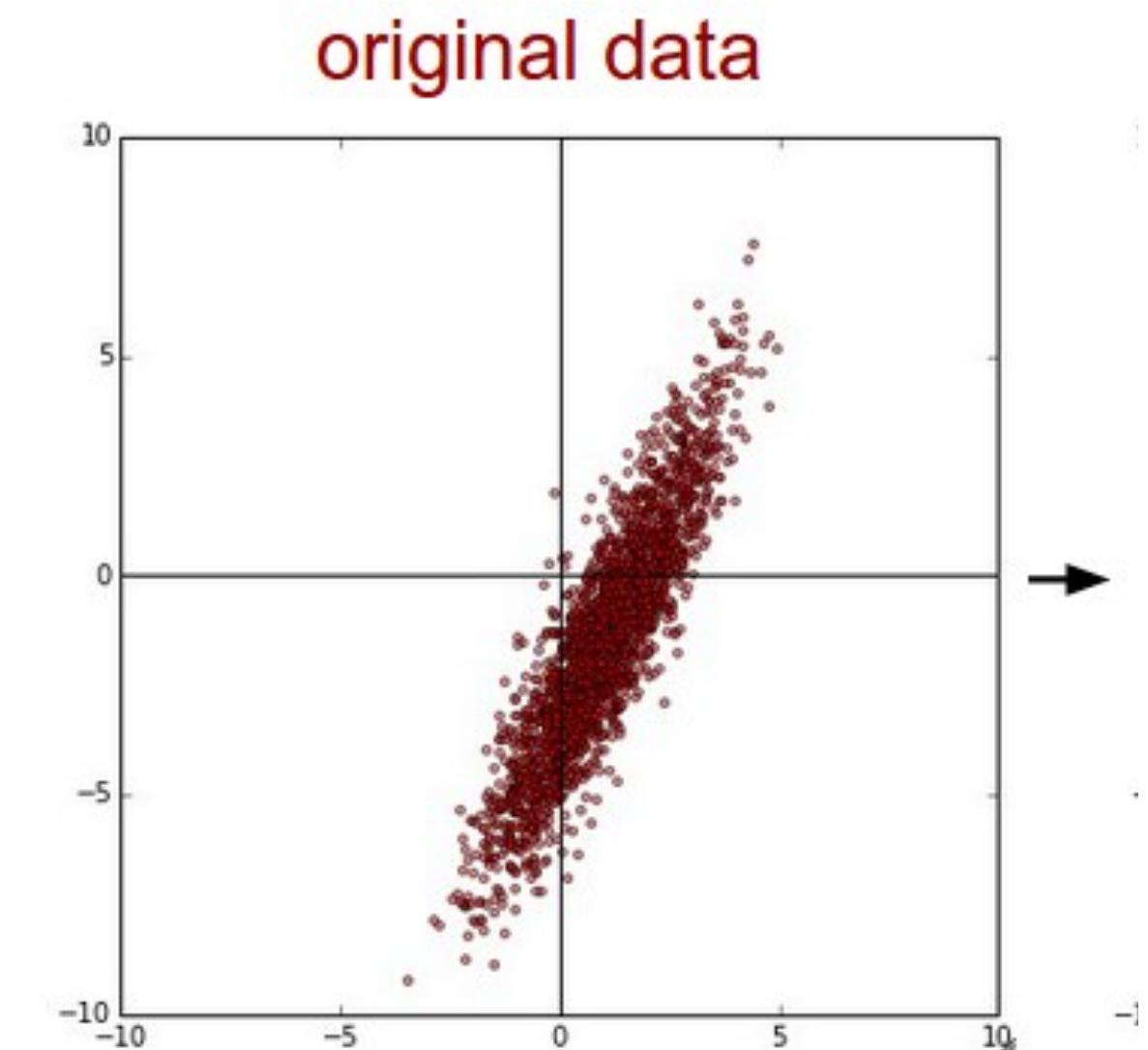
- Zero: fill the extra columns with zeros

# Convolutional Layer

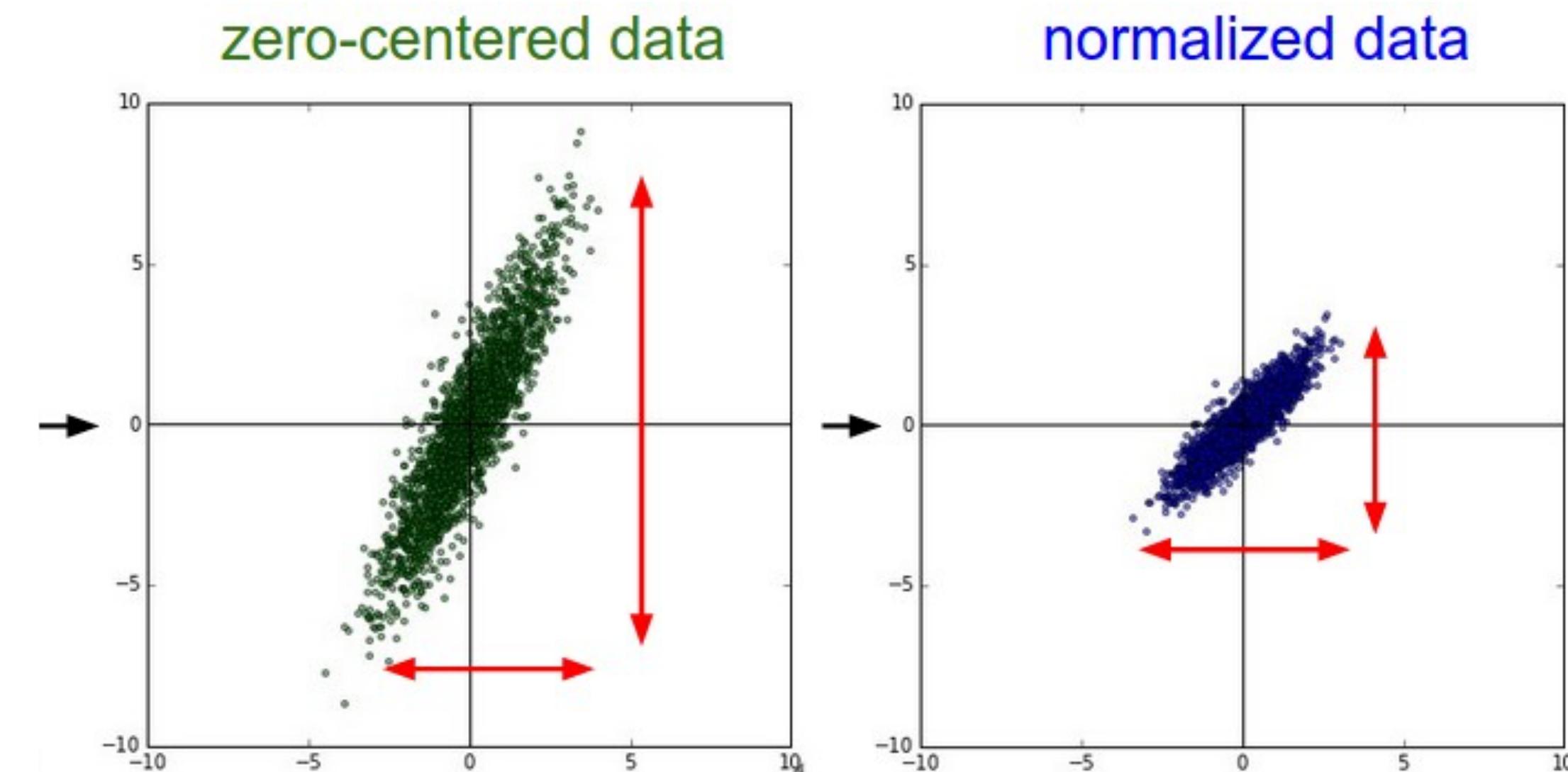


# BatchNormalization Layer

- 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



# more complex structures

- Dense NN architectures can be made more complex

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

- Multiple inputs

- Multiple outputs

- Different networks branches

- This is possible thanks to layer-manipulation layers

- Add, Subtract, etc.

- Concatenation

- Flattening

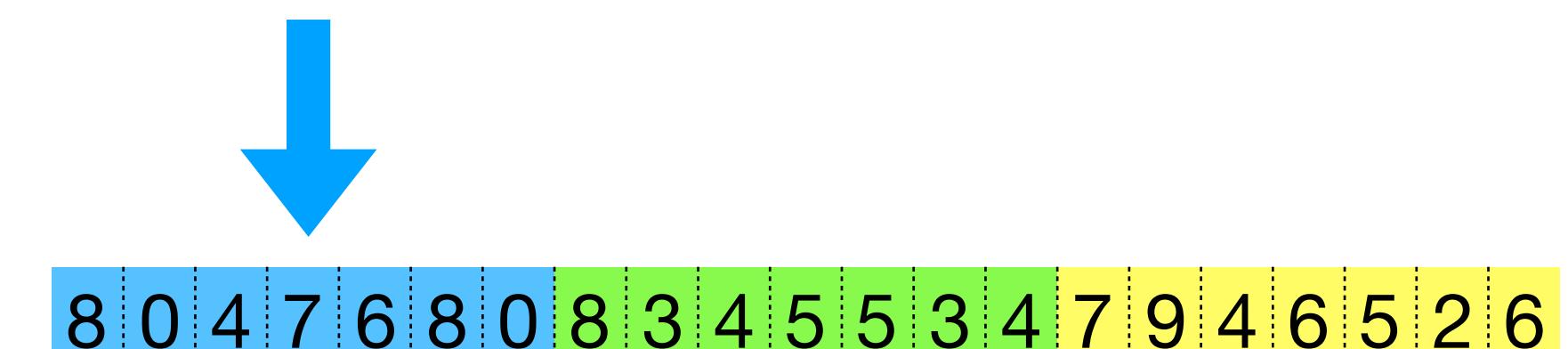
- All these operations are usually provided with NN training libraries



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

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

Flattening

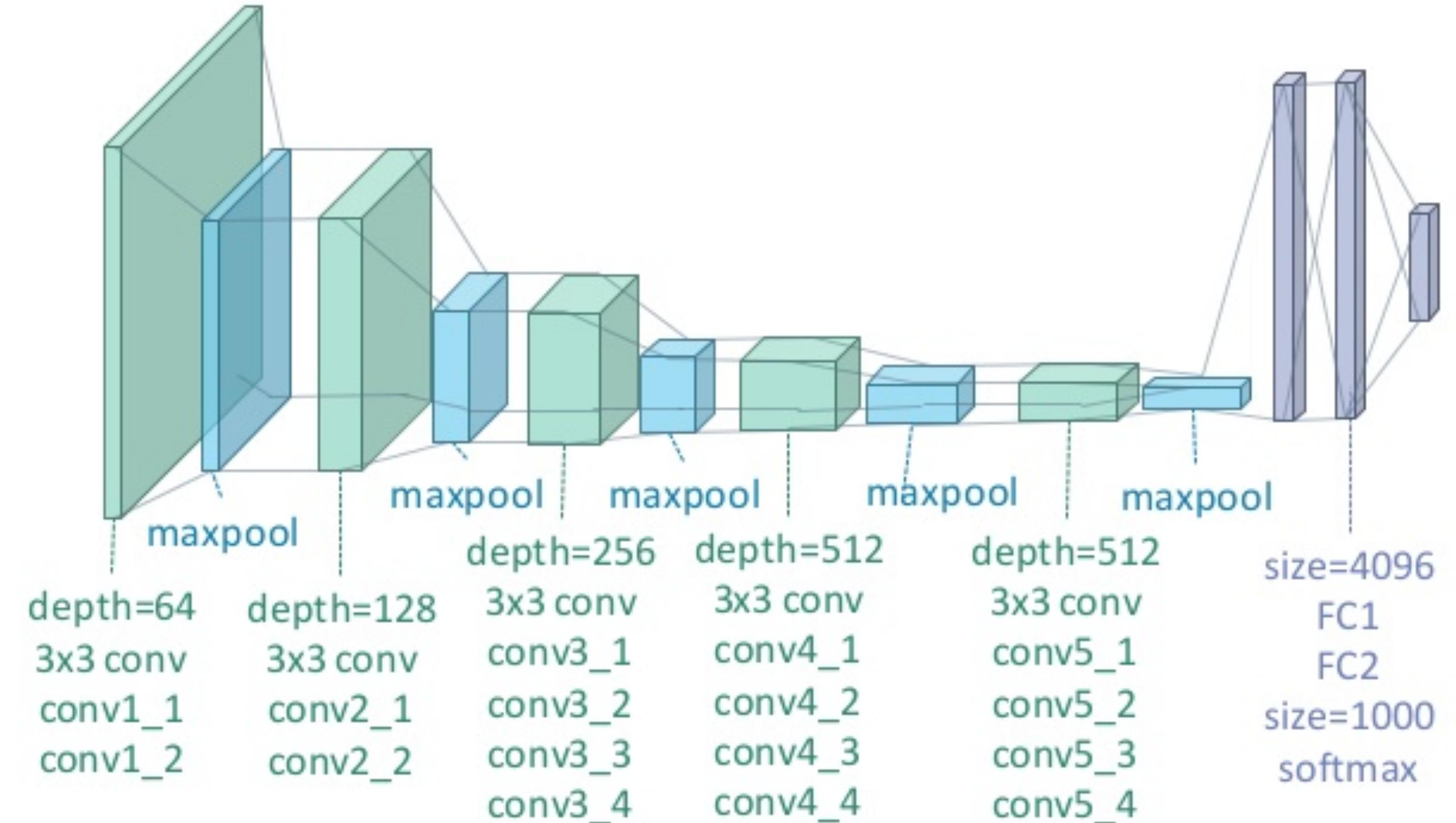


8|0|4|7|6|8|0|8|3|4|5|5|3|4|7|9|4|6|5|2|6

# 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

VGG 19



# What does a ConvNN learn?

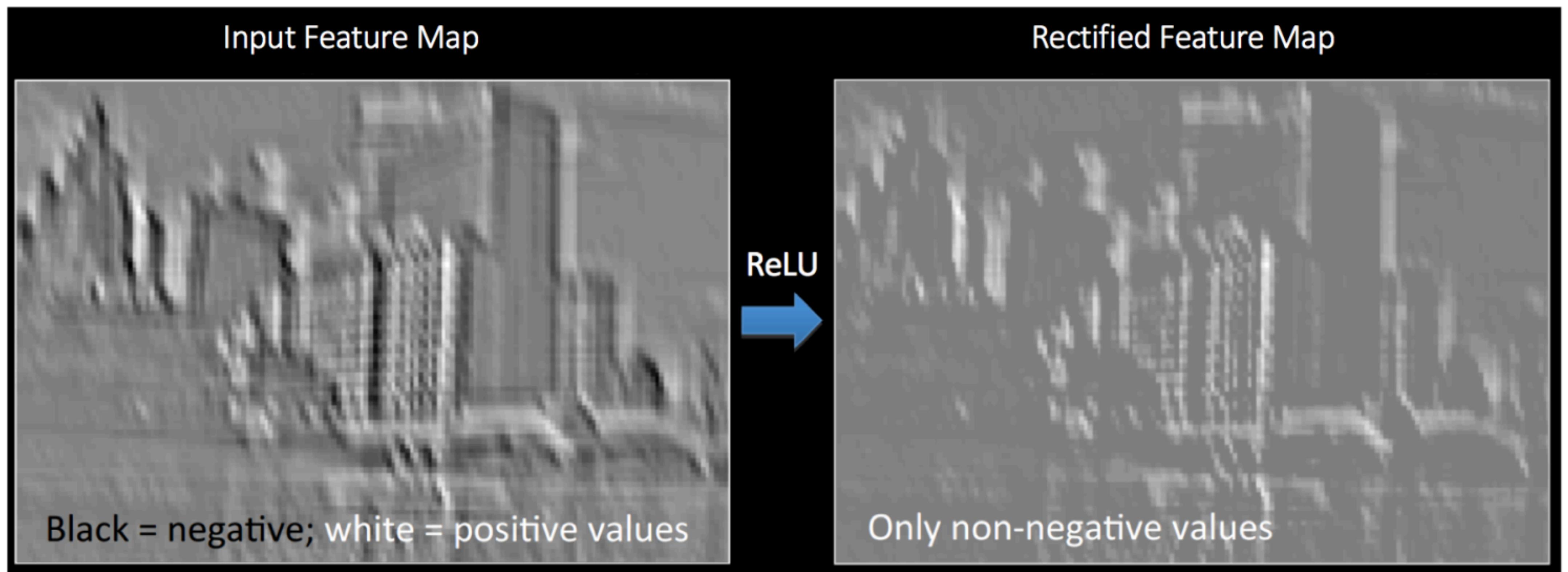
- *Each filter alters the image in a different way, picking up different aspects of the image*
- *edges oriented in various ways*
- *enhancing / blur of certain features*
- *It is interesting to check what each filter is doing (and to produce DeepDreams)*

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



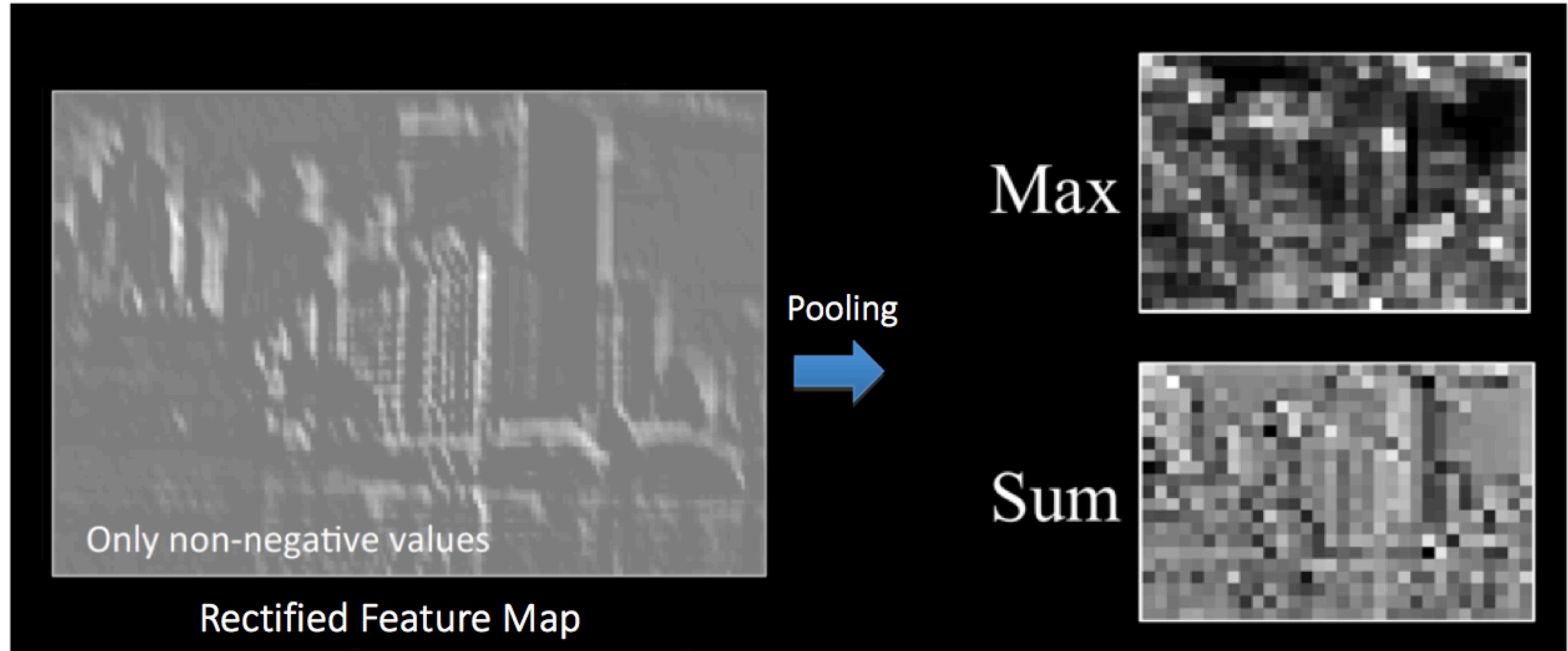
# What does a ConvNet learn?

- *The use on non-linear activation functions plays a special role in enhancing features*



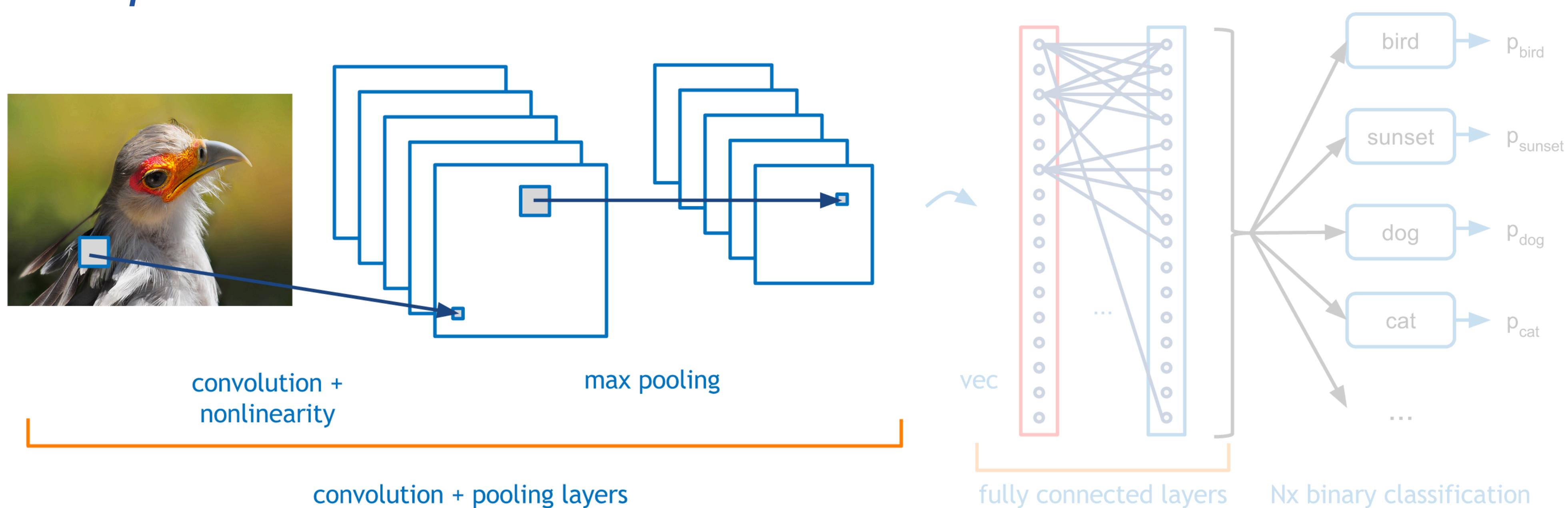
# What does a ConvNet learn?

- *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)*



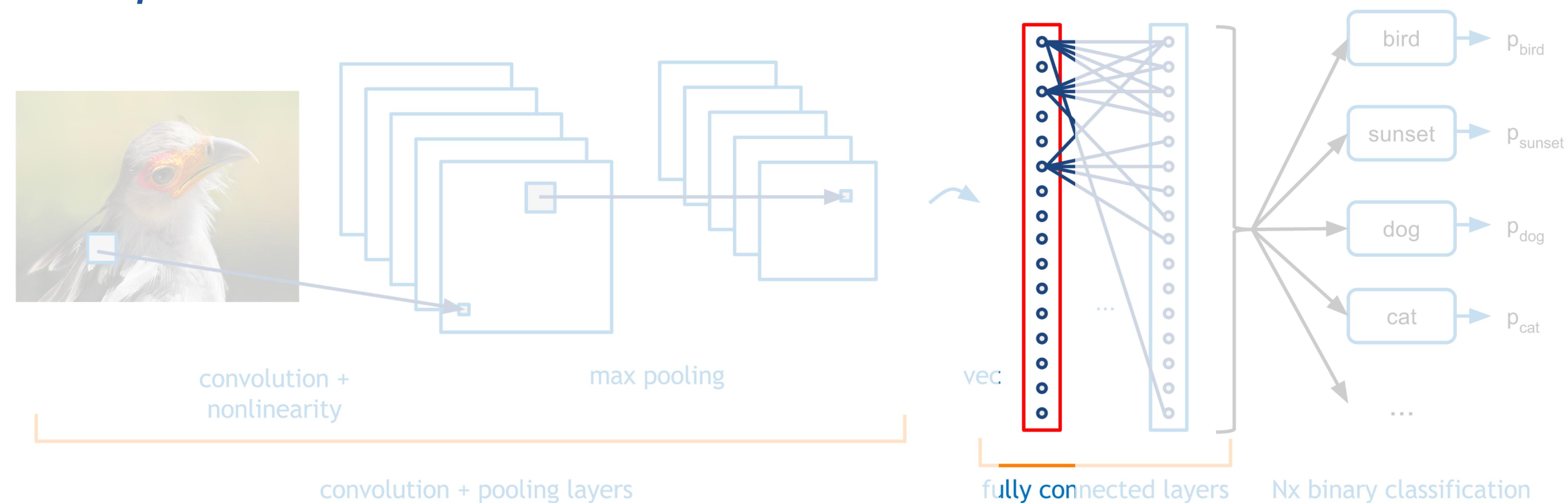
# Two tasks in one network

- *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*



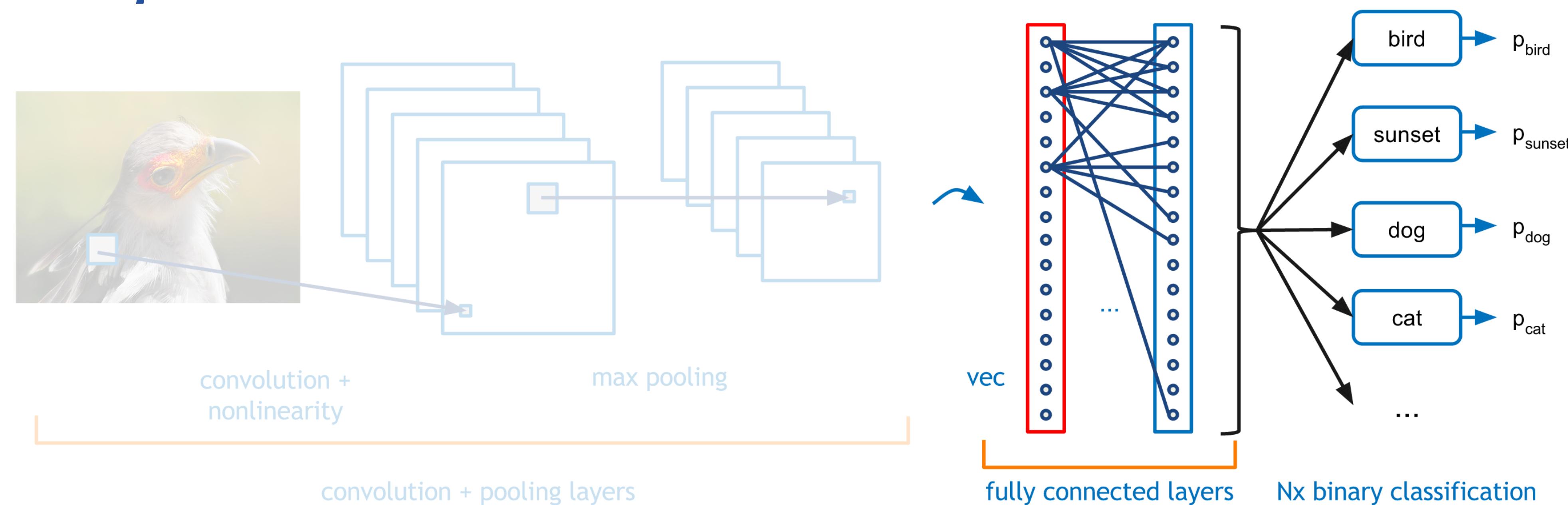
# Two tasks in one network

- 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



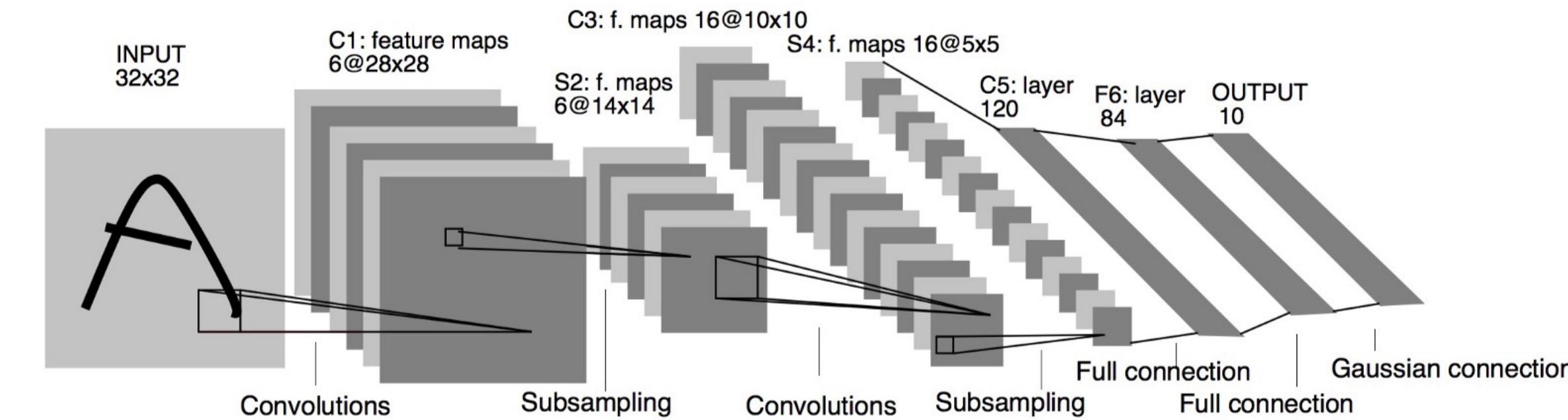
# Two tasks in one network

- 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

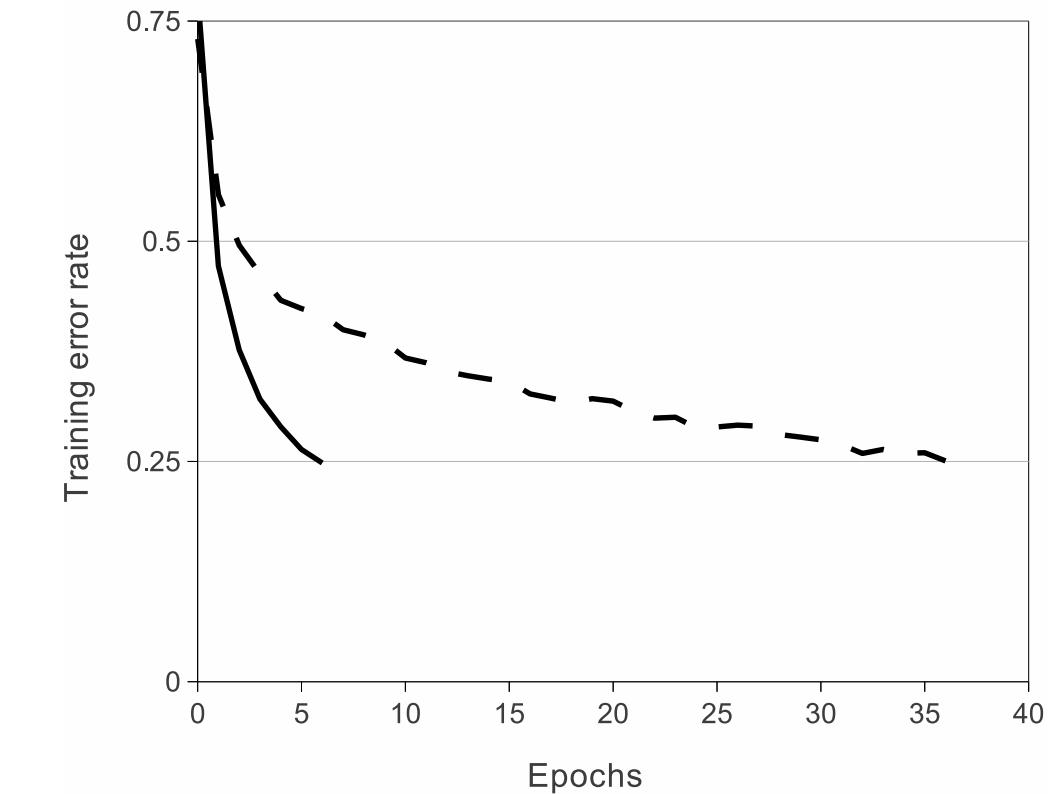
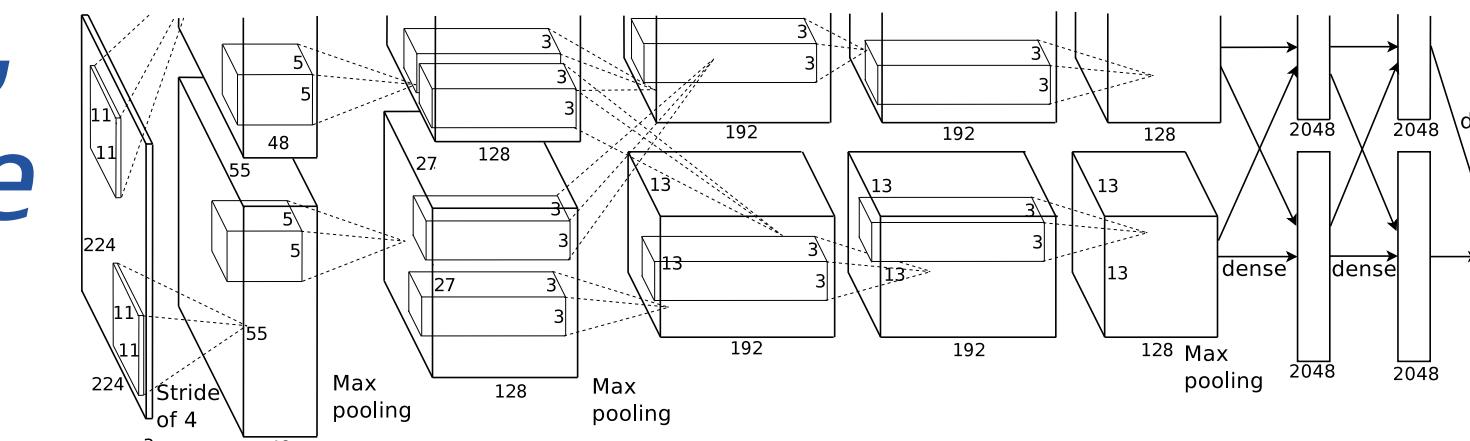


# A history of ConvNns

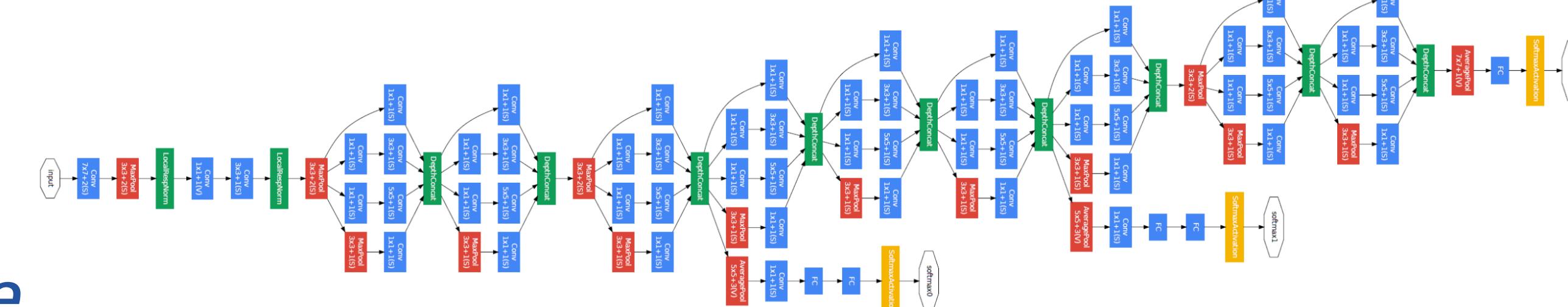
- LeNet (1990s): the very first ConvNN, designer for digit recognition (ZIP codes)



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

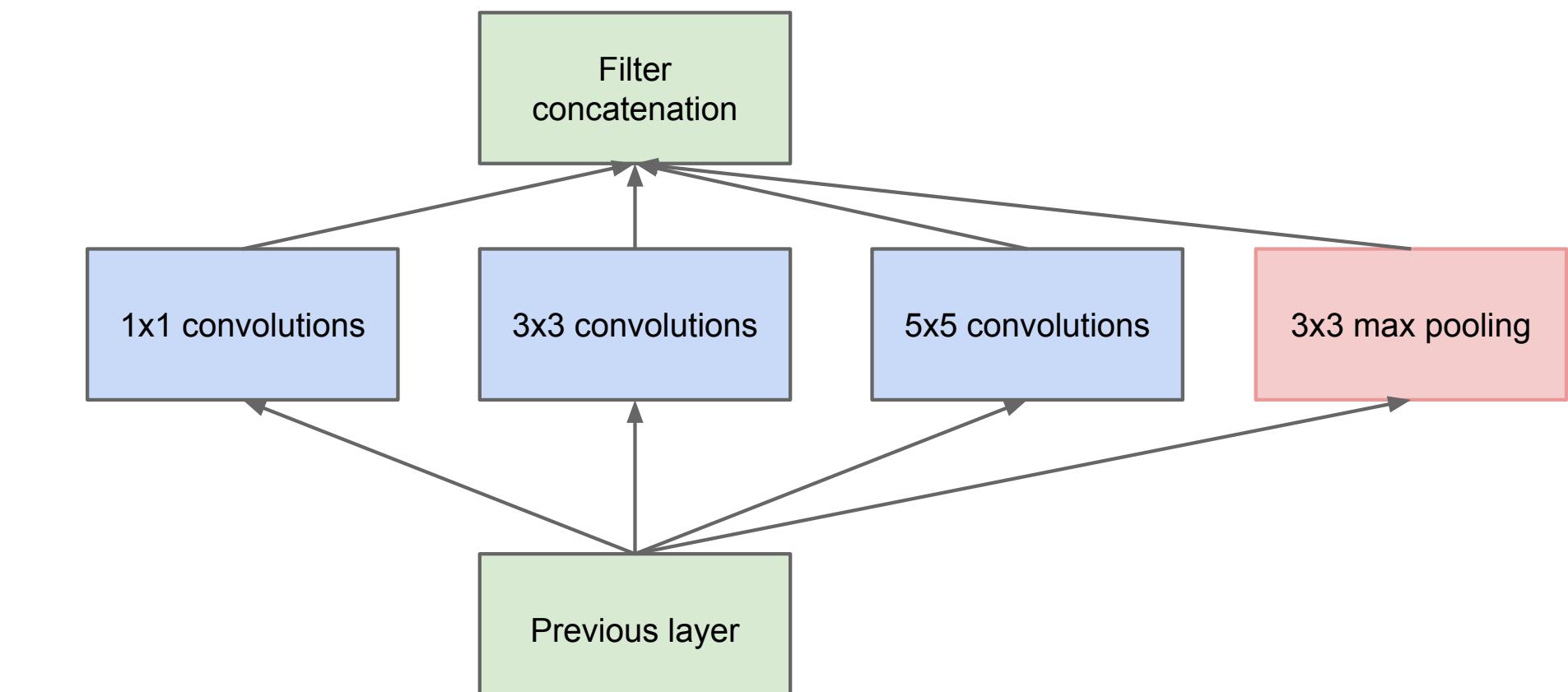
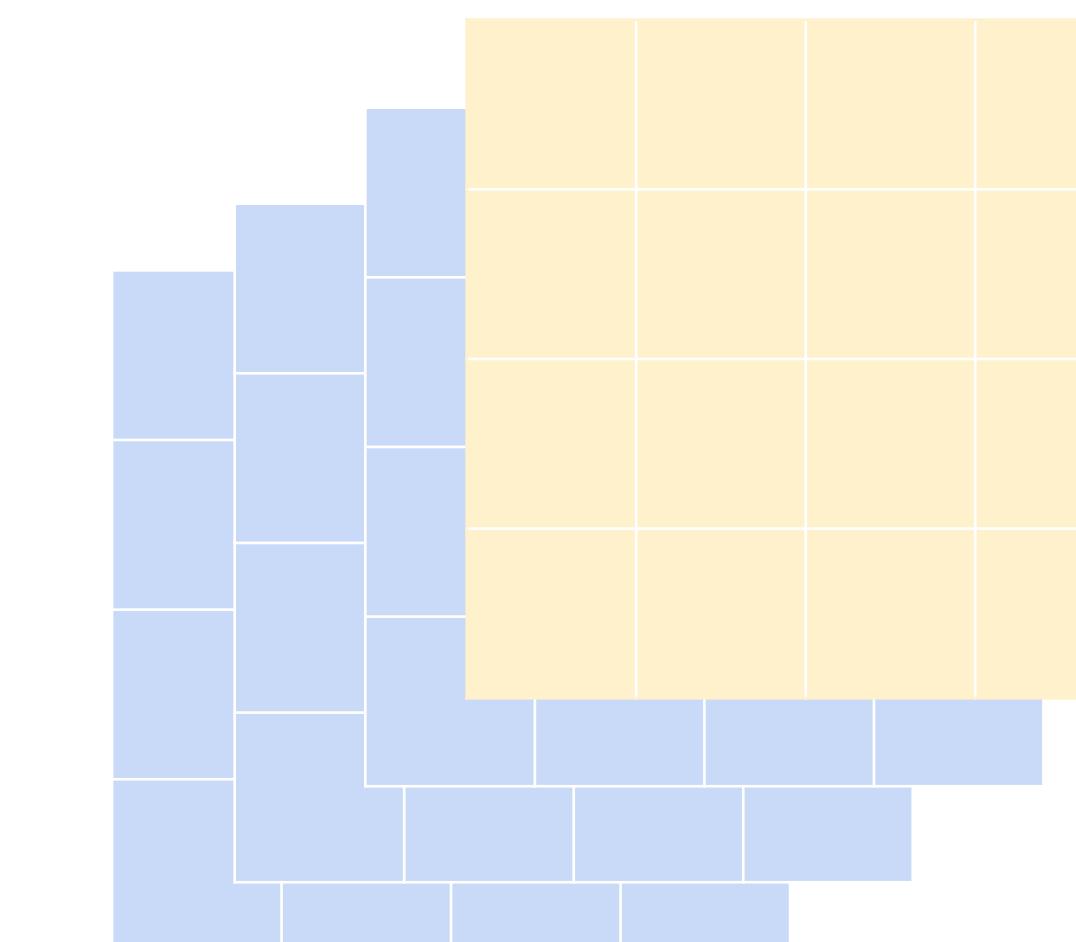


- GoogLeNet (2014): built on AlexNet, introduced an inception model to reduce the number of parameters

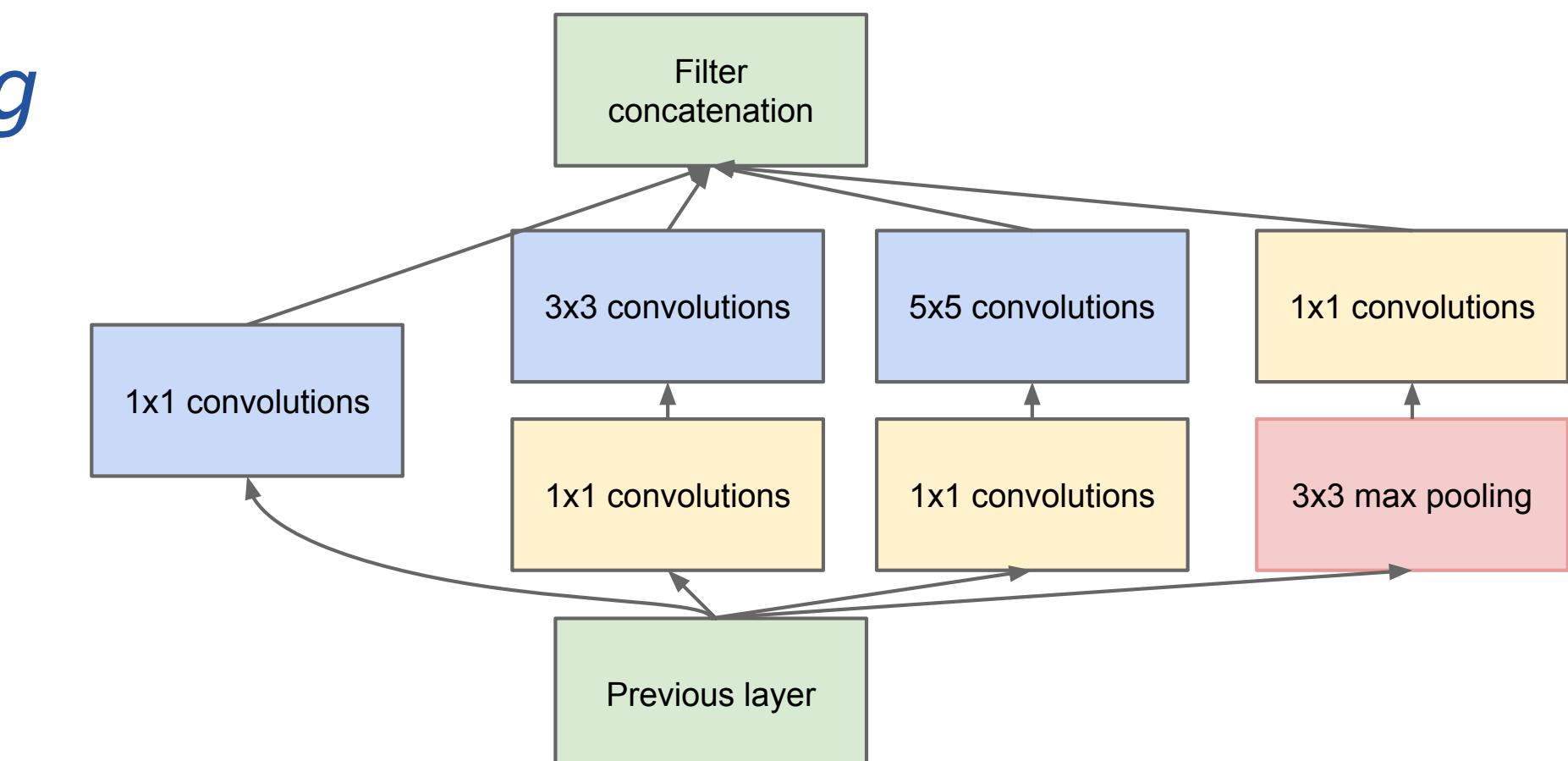


# Inception module

- 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 ass channels of an image



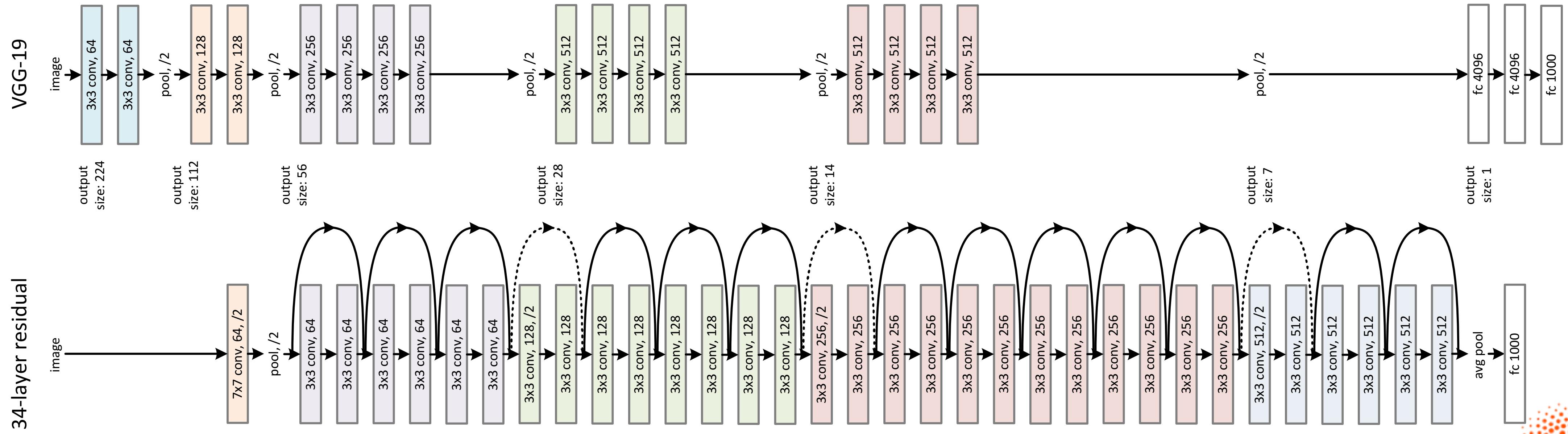
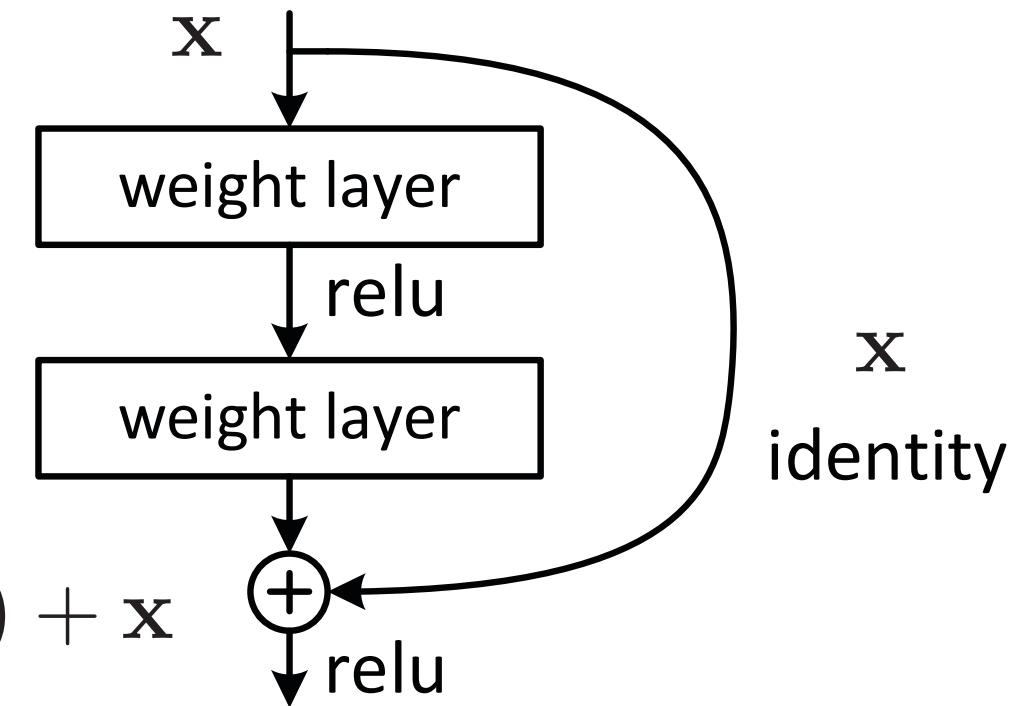
(a) Inception module, naïve version



(b) Inception module with dimension reductions

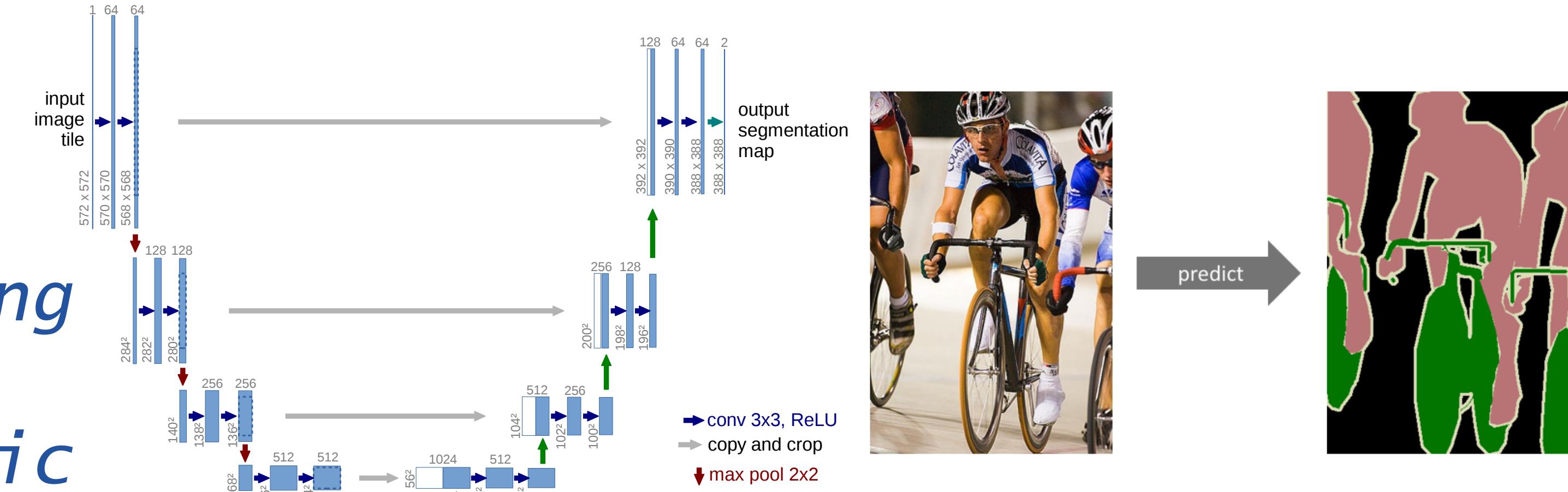
# A history of ConvNNS

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

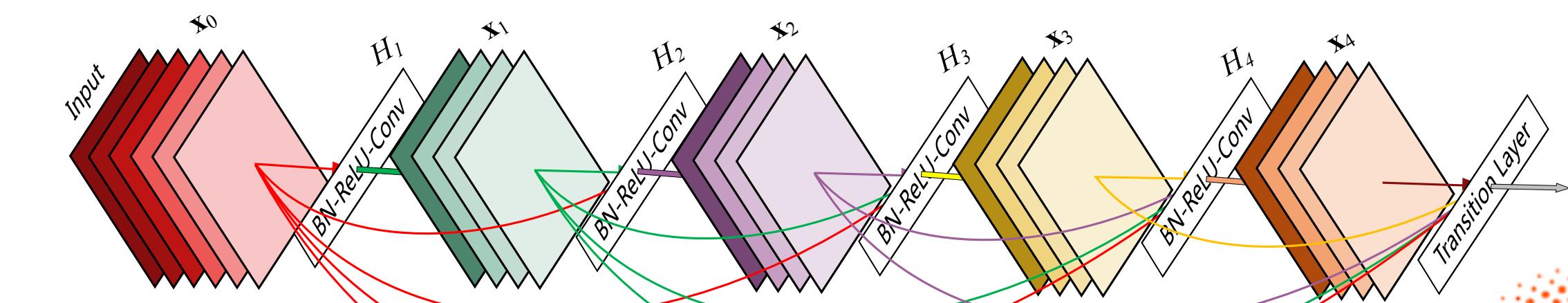
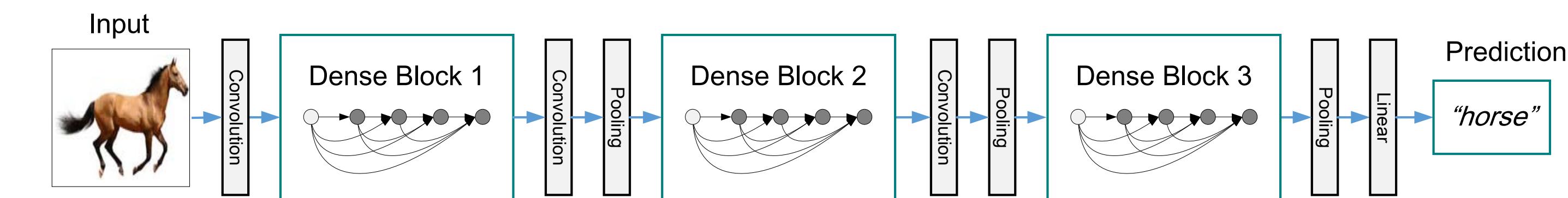


# A history of ConvNets

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



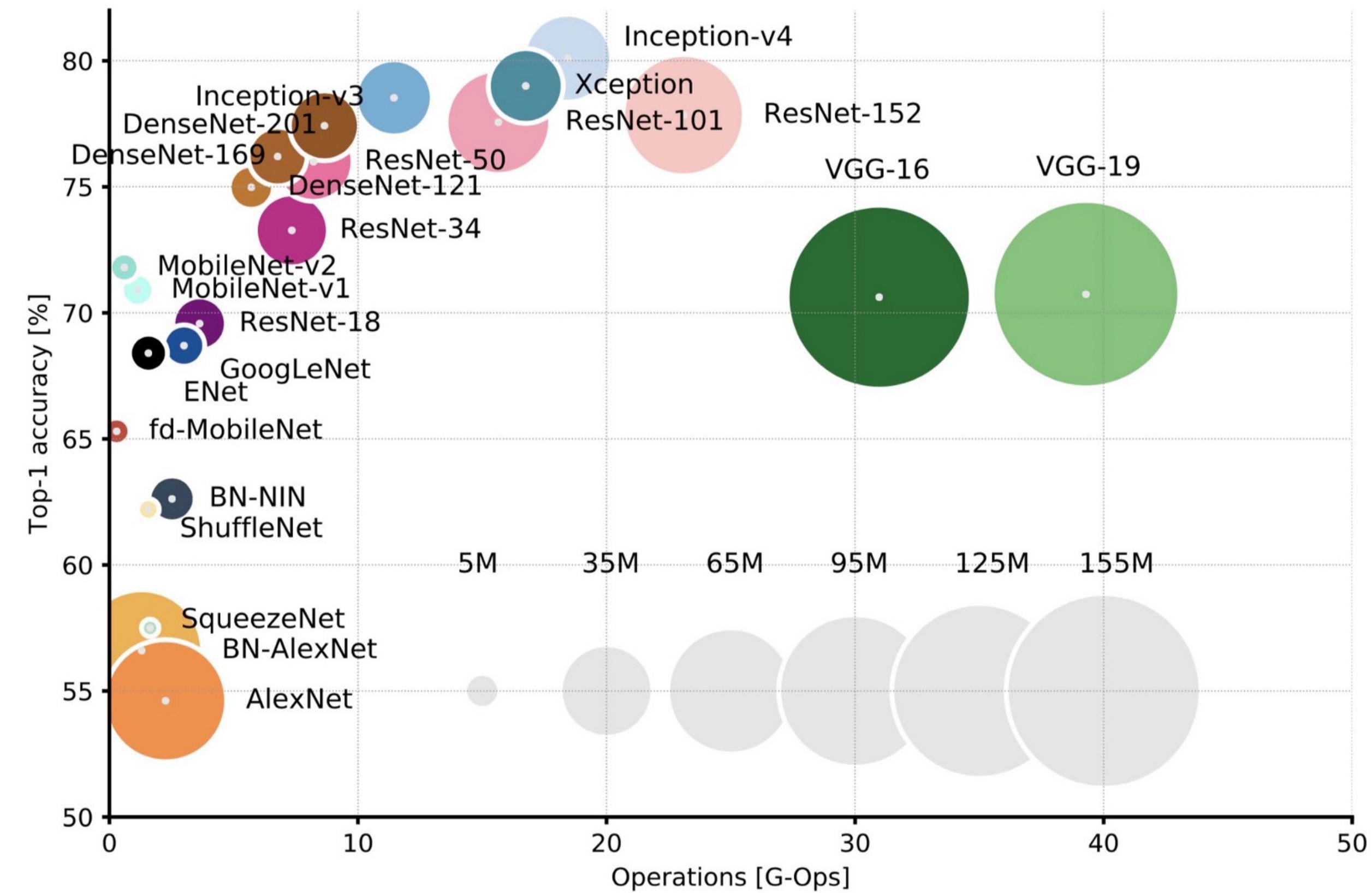
- DenseNet (2016): uses skip connections between a given layer and all the layers downstream in a dense block, with Conv layers in between



# The computational cost

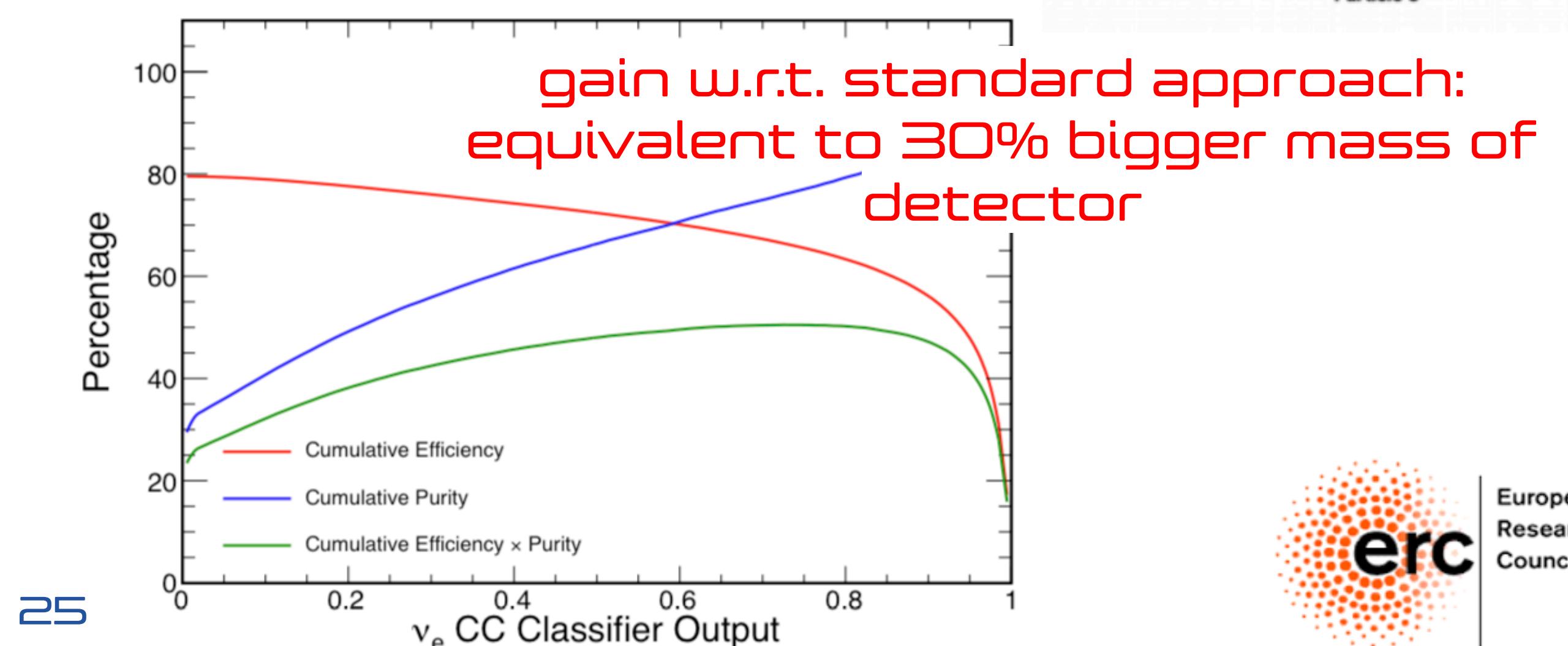
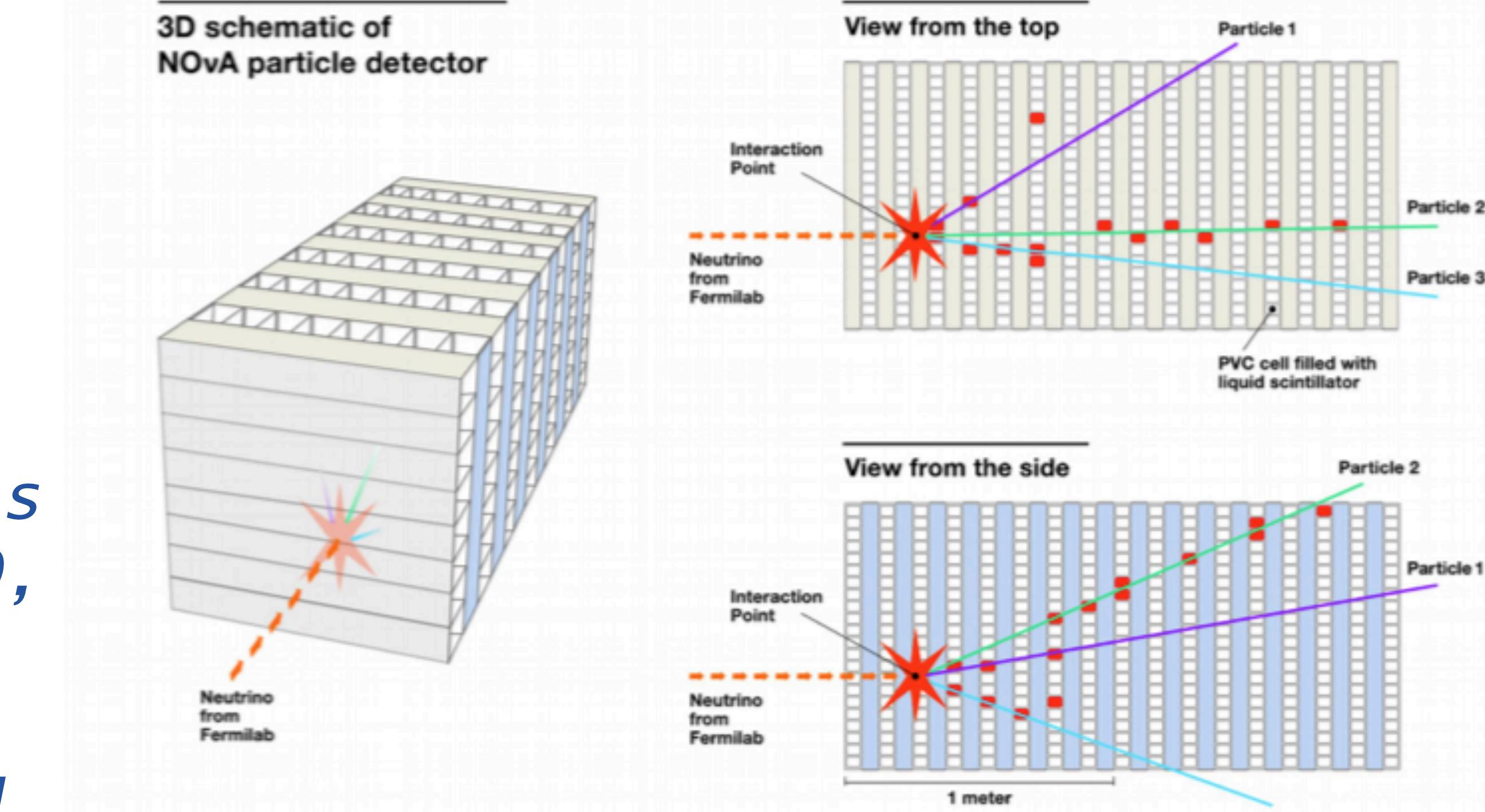
- In this evolution, computing-vision networks drastically improved in performance
- This came at a big cost in complexity (number of parameters and operations)
- The inference with this network became particularly slow
  - big interest in optimize these networks on dedicated resources, e.g. FPGAs
- 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>



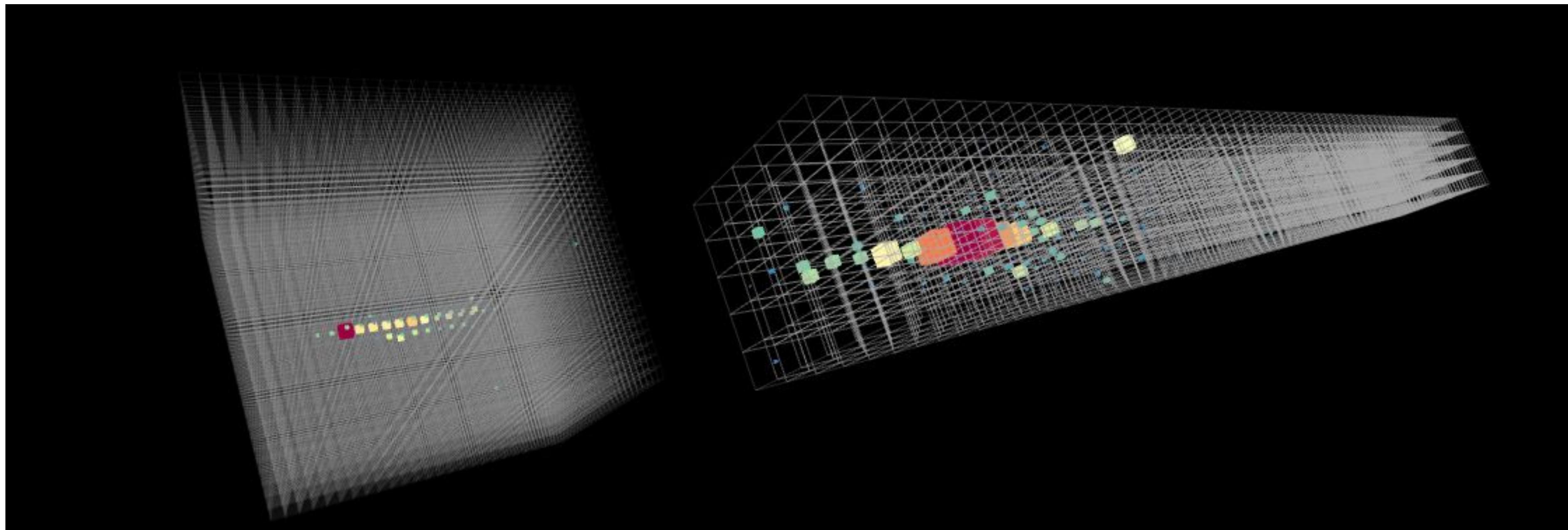
# Example: PID for $\nu$ experiments

- 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)



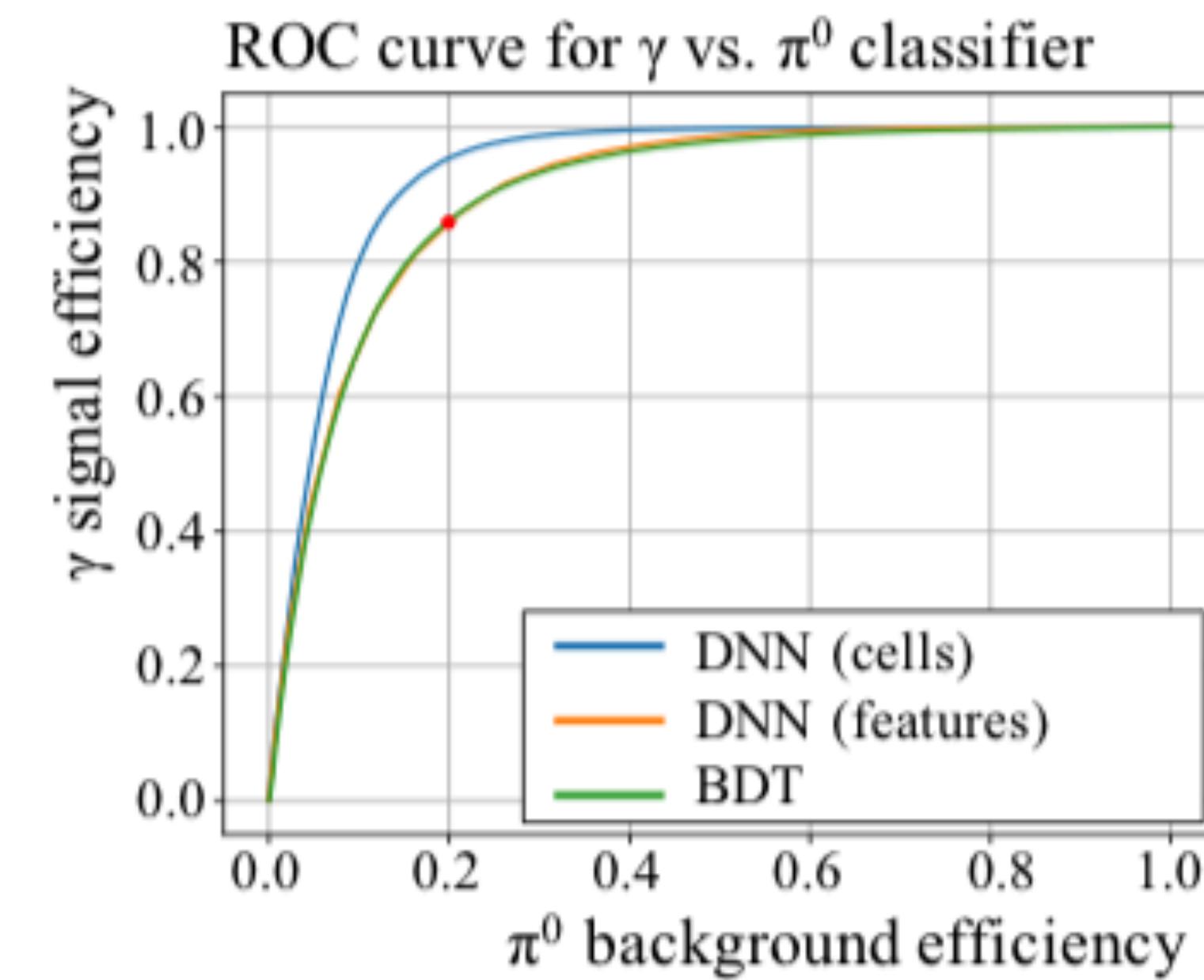
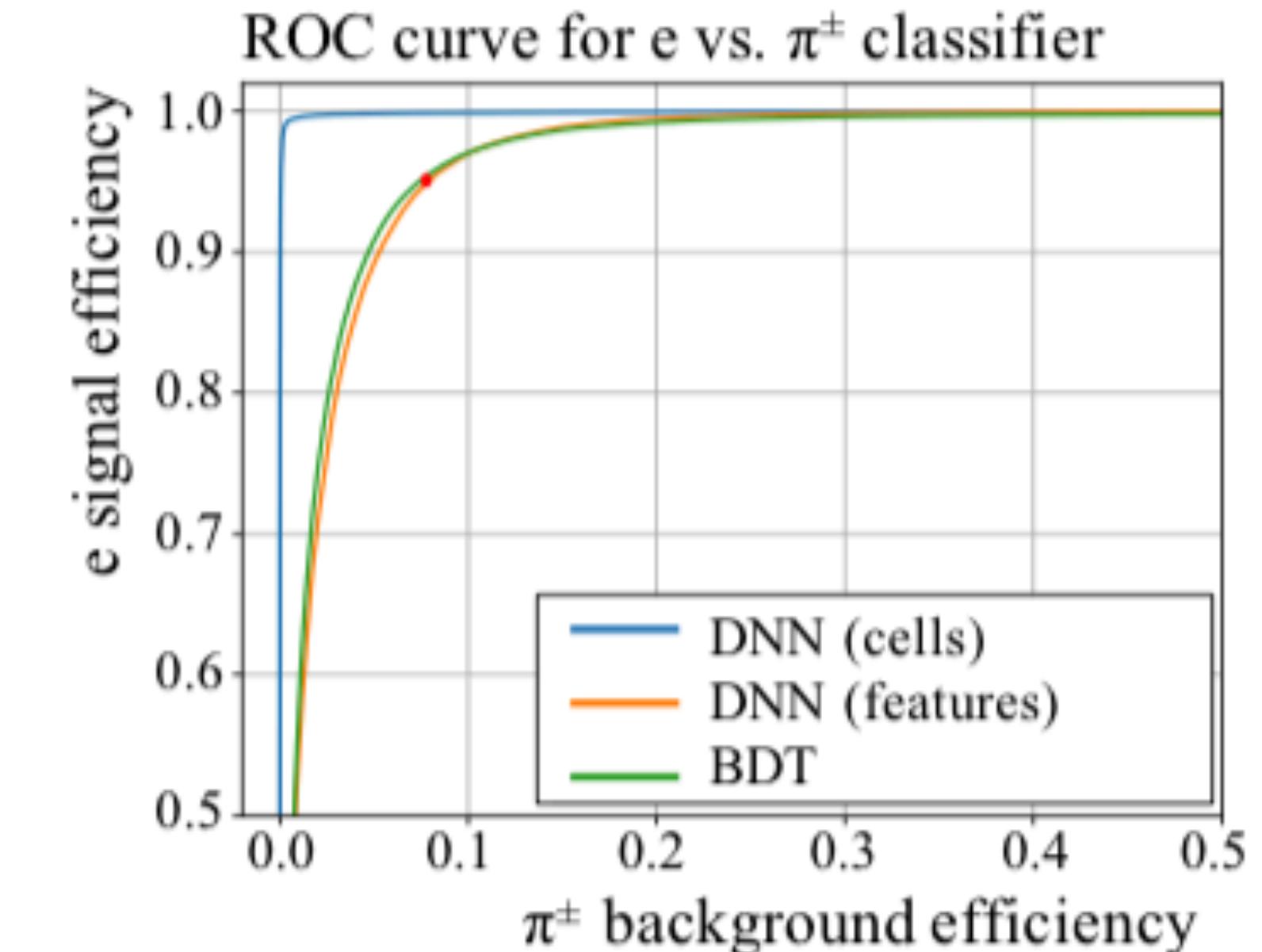
# Calorimetry & Computer Vision

- (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



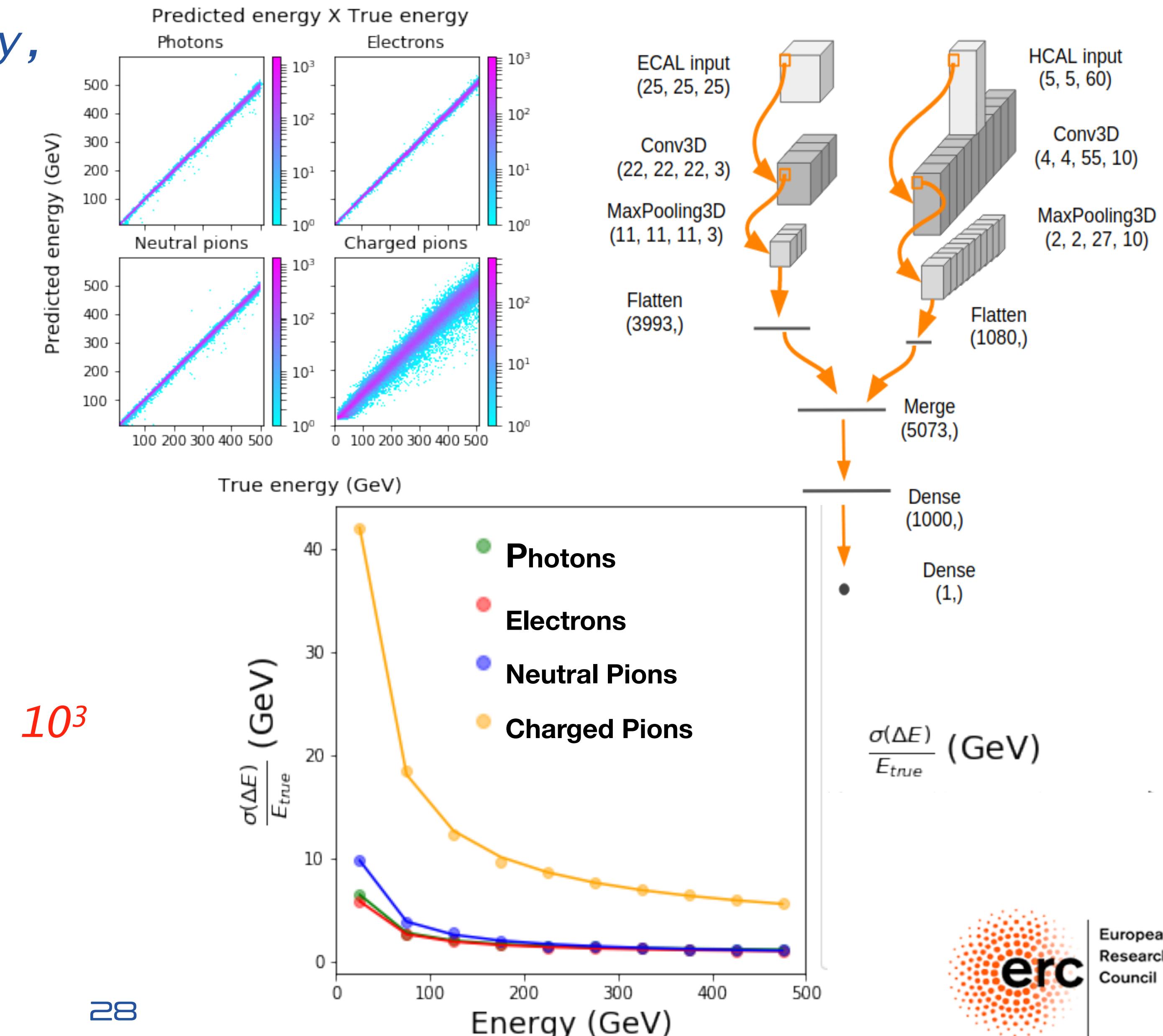
# Example: Particle ID

- We tried particle ID on a sample of simulated events
  - one particle/event ( $e$ ,  $\gamma$ ,  $\pi^0$ ,  $\pi$ )
  - 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

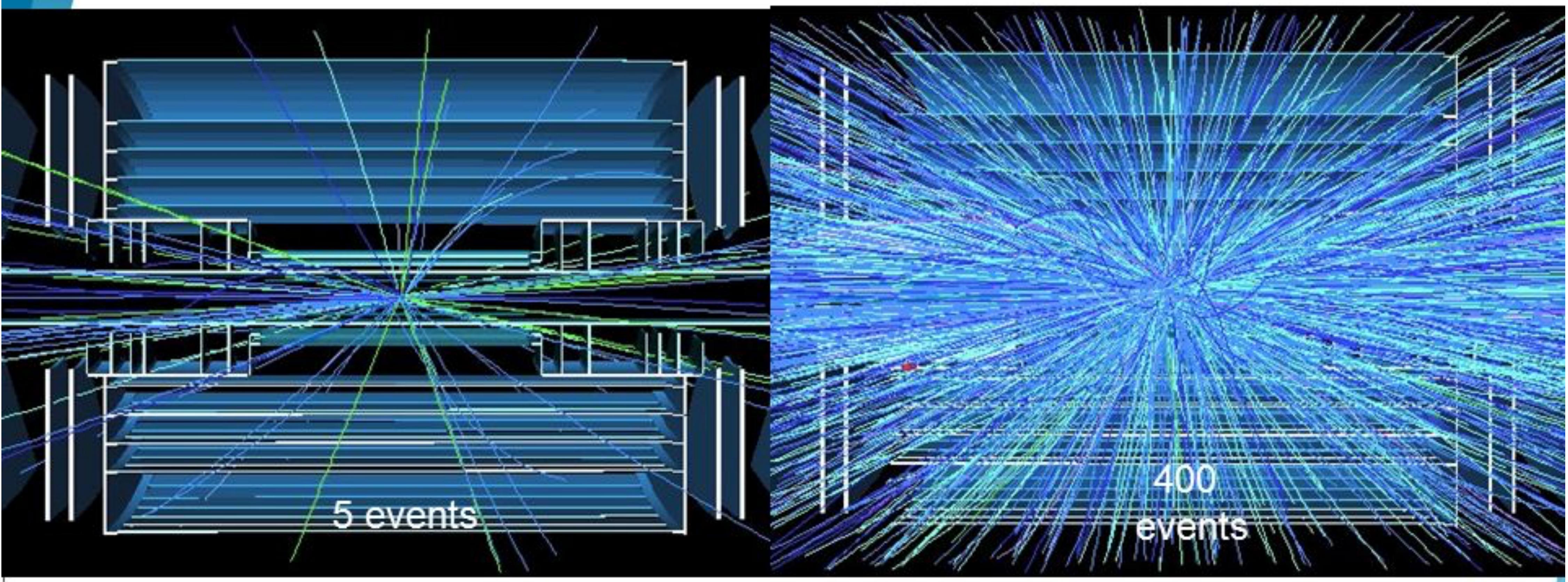


# Example: Energy Regression

- *Correctly reconstruct energy, with physics meaningful performances*
- *ECAL performances better than HCAL (as expected)*
- *$\pi^0$  resolution  $\sim \sqrt{2}$  γ resolution (as expected)*
- *FAST: used only RAW data as inputs -> no pre-processing*
  - ▶ *Processing time reduced by  $10^3$  wrt traditional approaches*
  - ▶ *Potentially usable both online and offline*



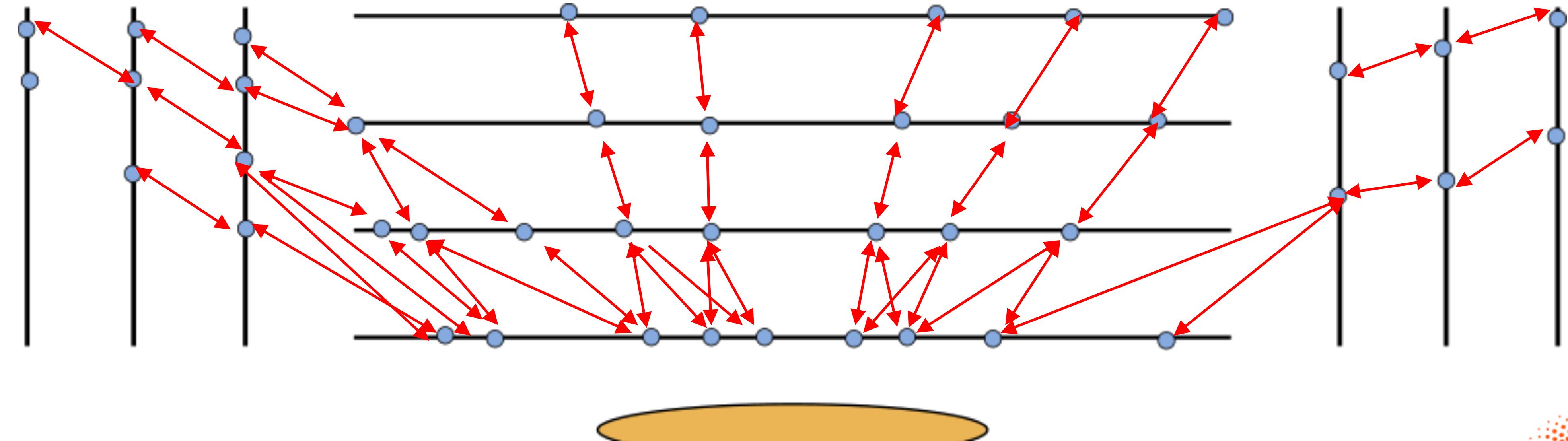
# Combinatorics Reduction



- *Tracking is the most expensive workflow we have in RECO*
- *The more tracks we have, the more problematic it becomes (non-linearity due to combinatoric when connecting dots)*

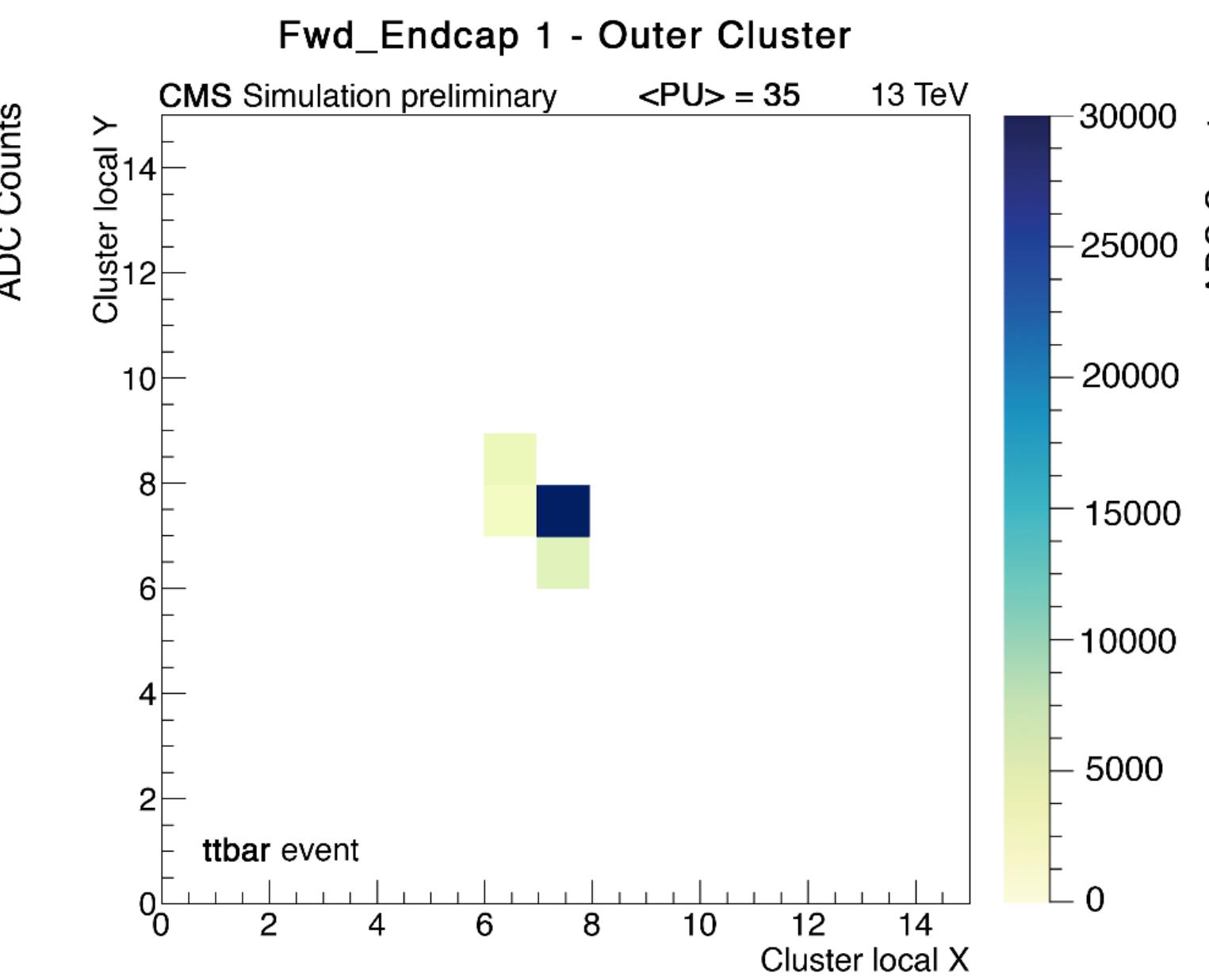
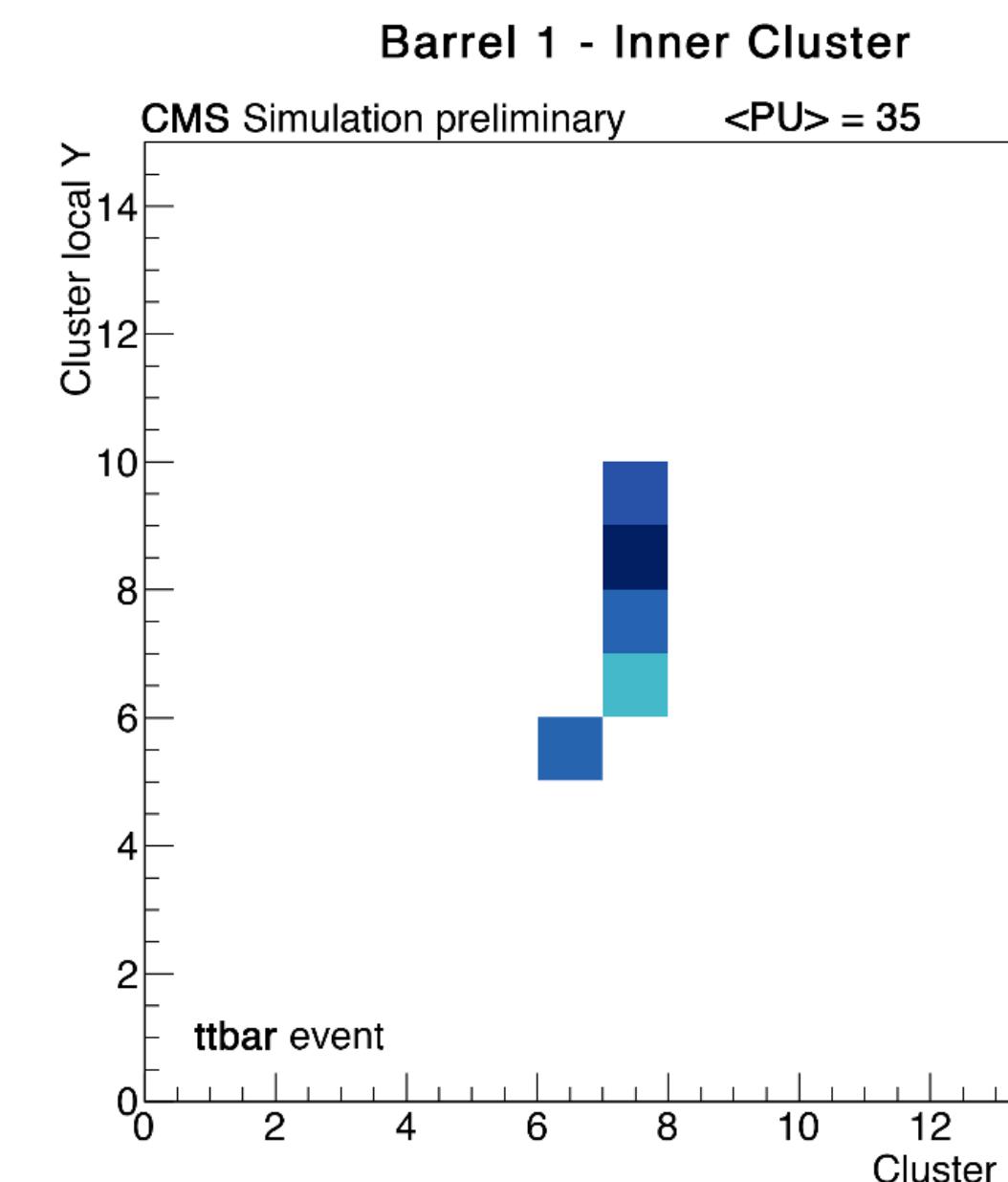
# Combinatorics Reduction

- Track reco starts with a see in the inner layers (triplet of hits)
- Finding seeds is not the most CPU intensive aspect
- But its outcome determines the complexity of the following steps
  - one can speed up tracking by reducing the number of fake seeds
- We tried to solve this problem using a ConvNN



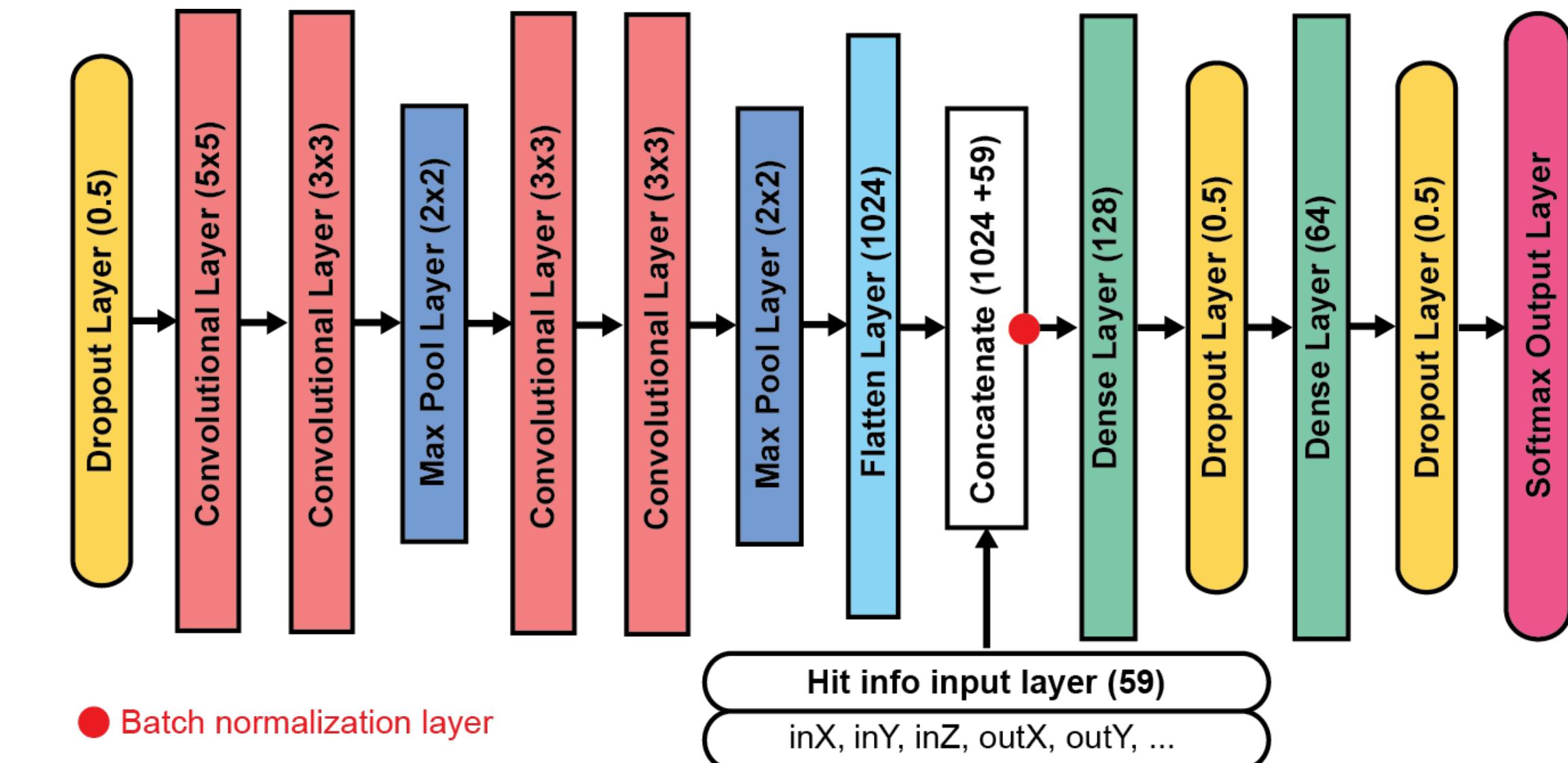
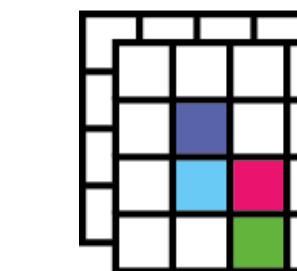
# Seeds as images

- *The detector sees the charge deposited by the crossing particle: a hit*
- *A hit is a window of sensors (16x16 here) with its deposited charge. This can be seen as a sparse digital image.*
- *Given two images, one can train a network to decide if a pair of hits is a good or bad match*



# PixelSeed ConvNN

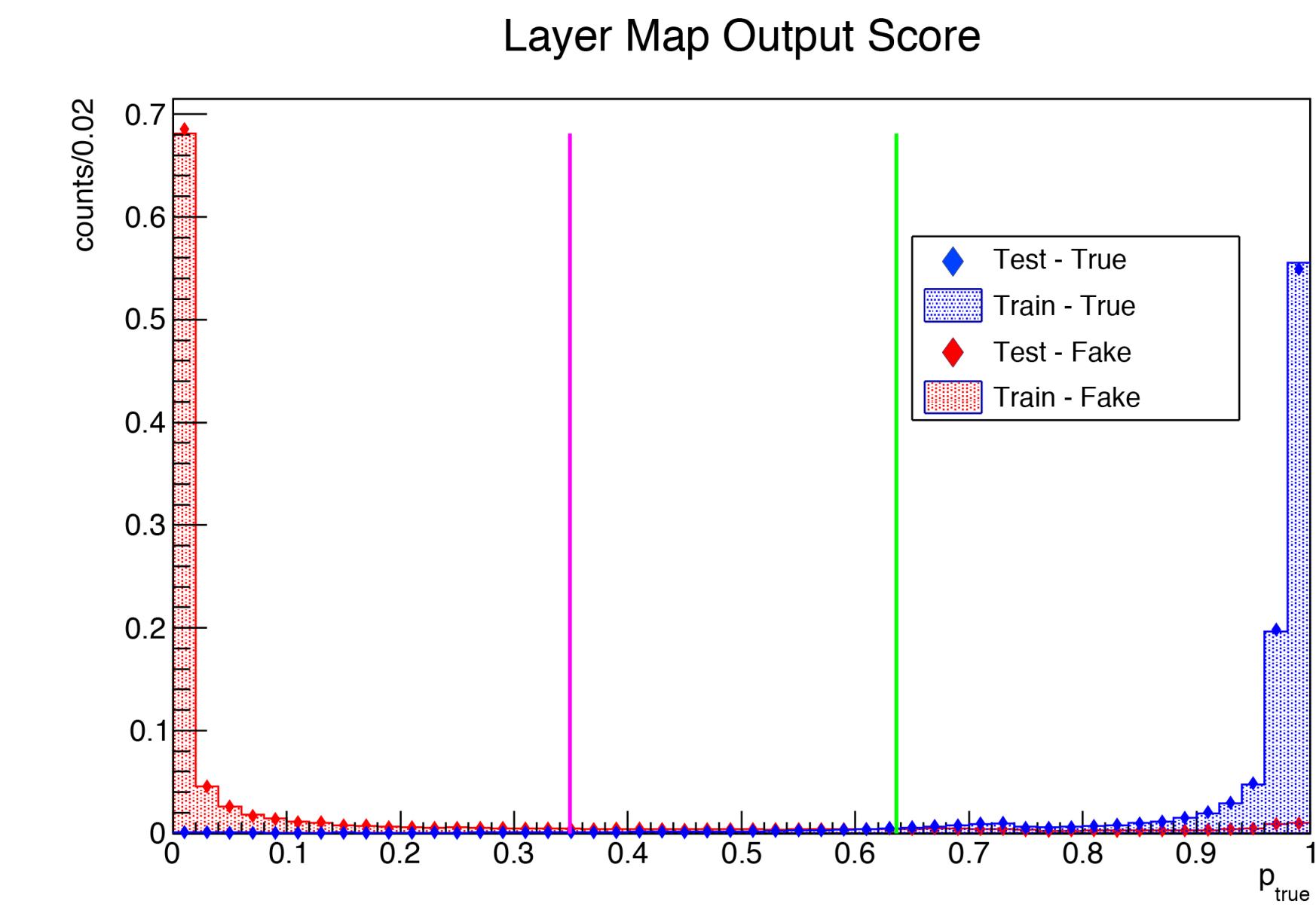
- The final model uses two sets of inputs:
  - the hit images
  - a set of expert features (e.g., position of the hits in the detector) to help the learning process



- The trained model shows a good separation of true vs fake seeds
- One can reduce the fake rate by one order of magnitude with a few % loss in efficiency

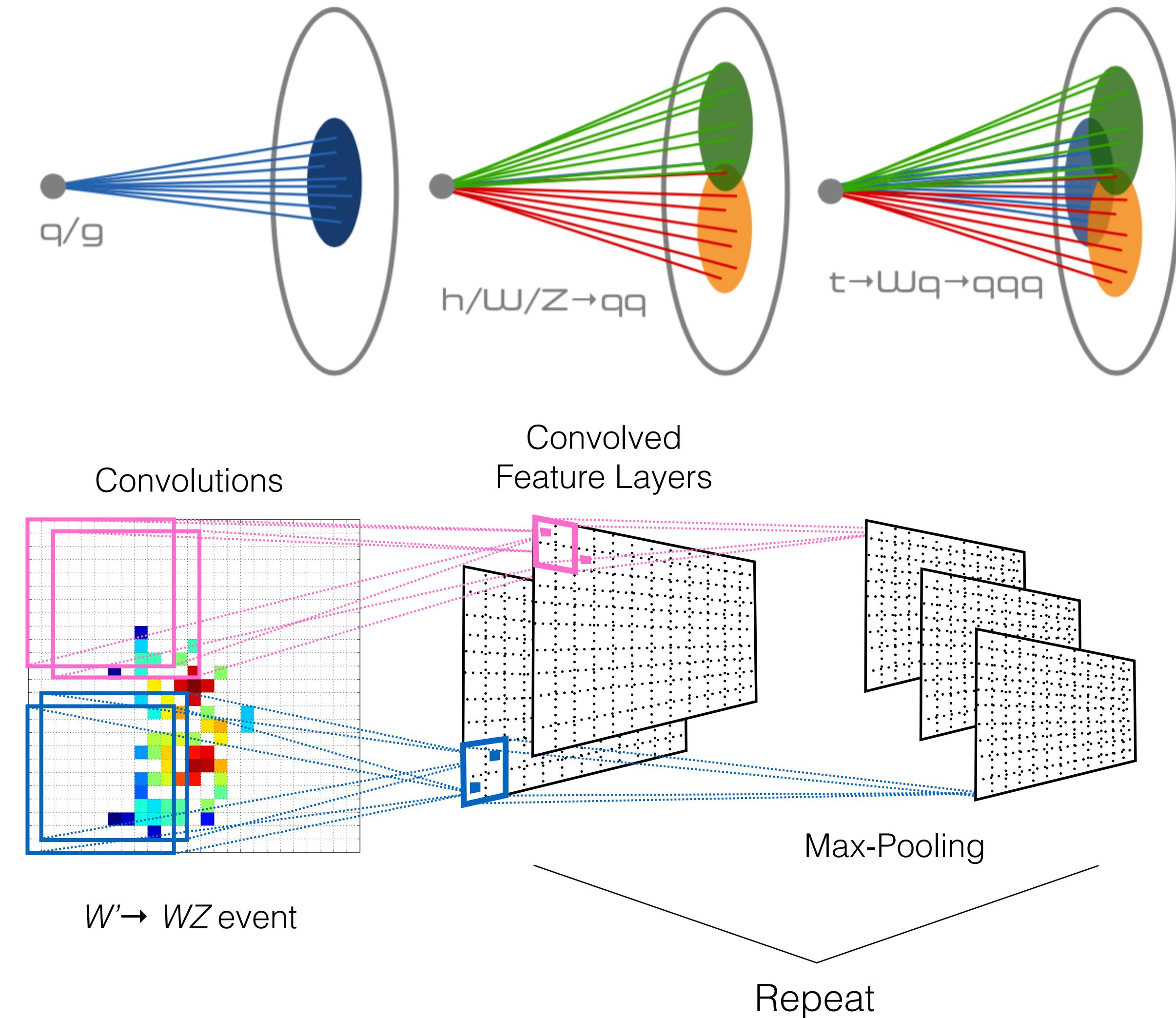
Efficiency (tpr) @ fake rejection

```
tpr @ rej 50%: 0.998996700259
tpr @ rej 75%: 0.990524391331
tpr @ rej 90%: 0.922210826719
tpr @ rej 99%: 0.338669401587
```



# Jet as images (for ConvNN)

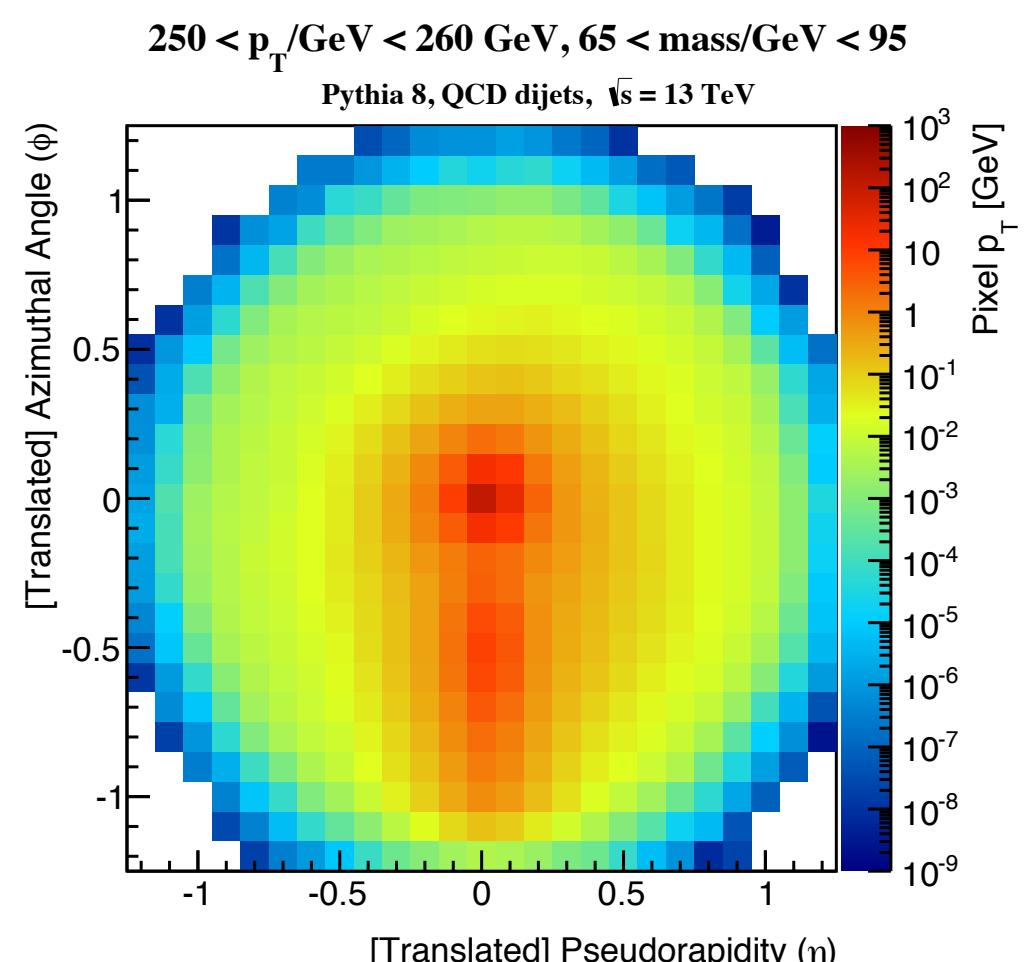
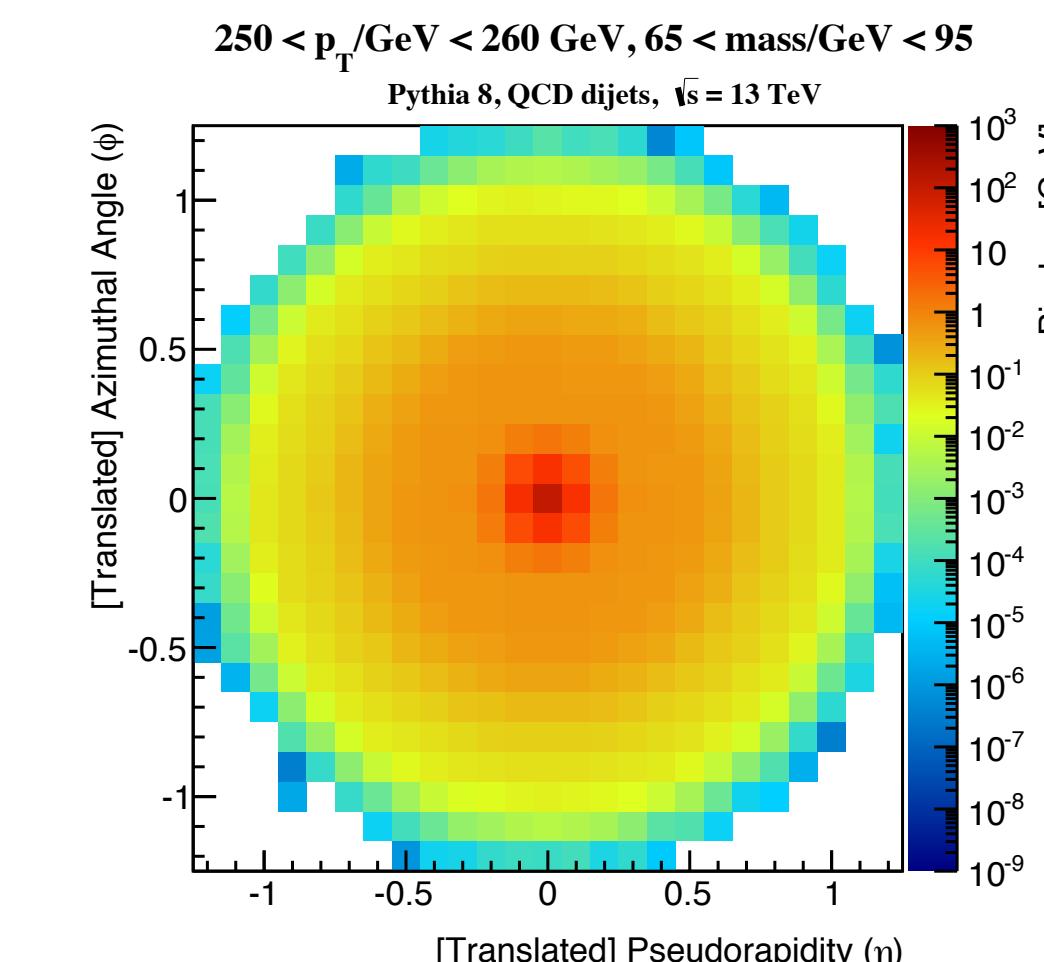
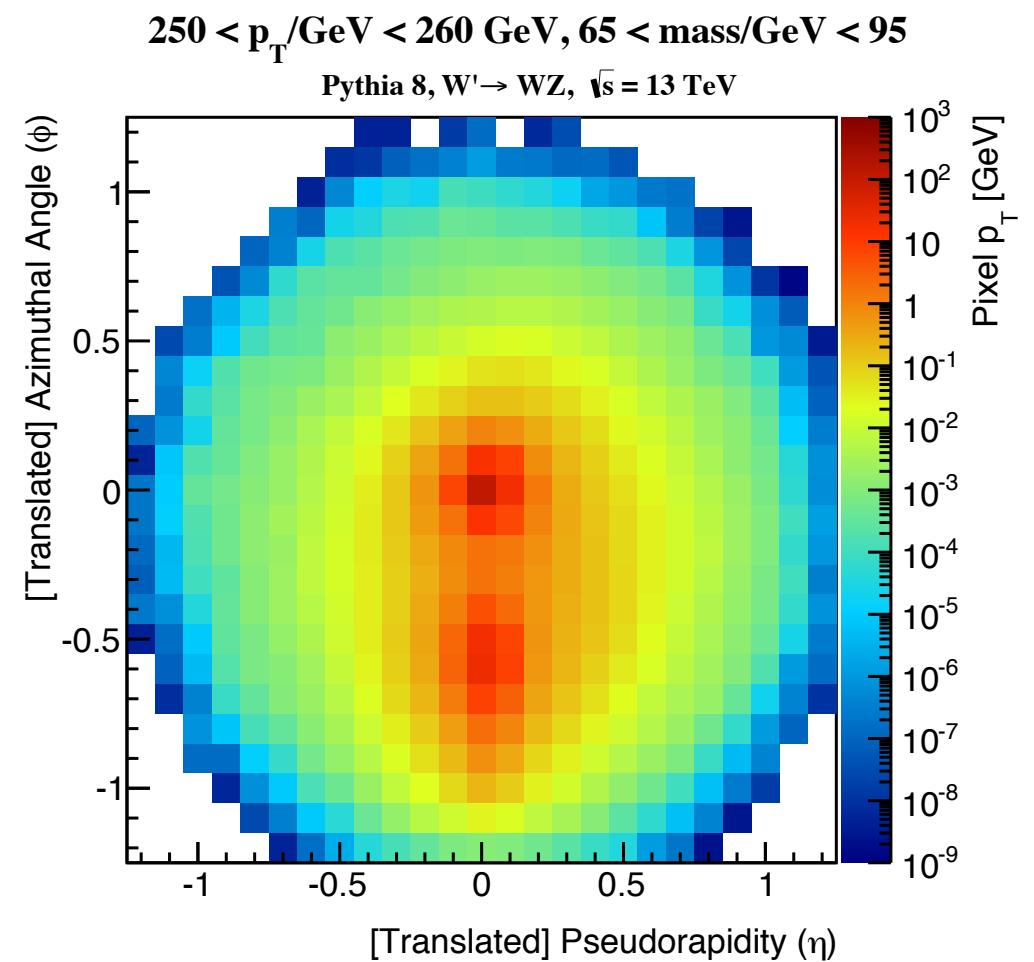
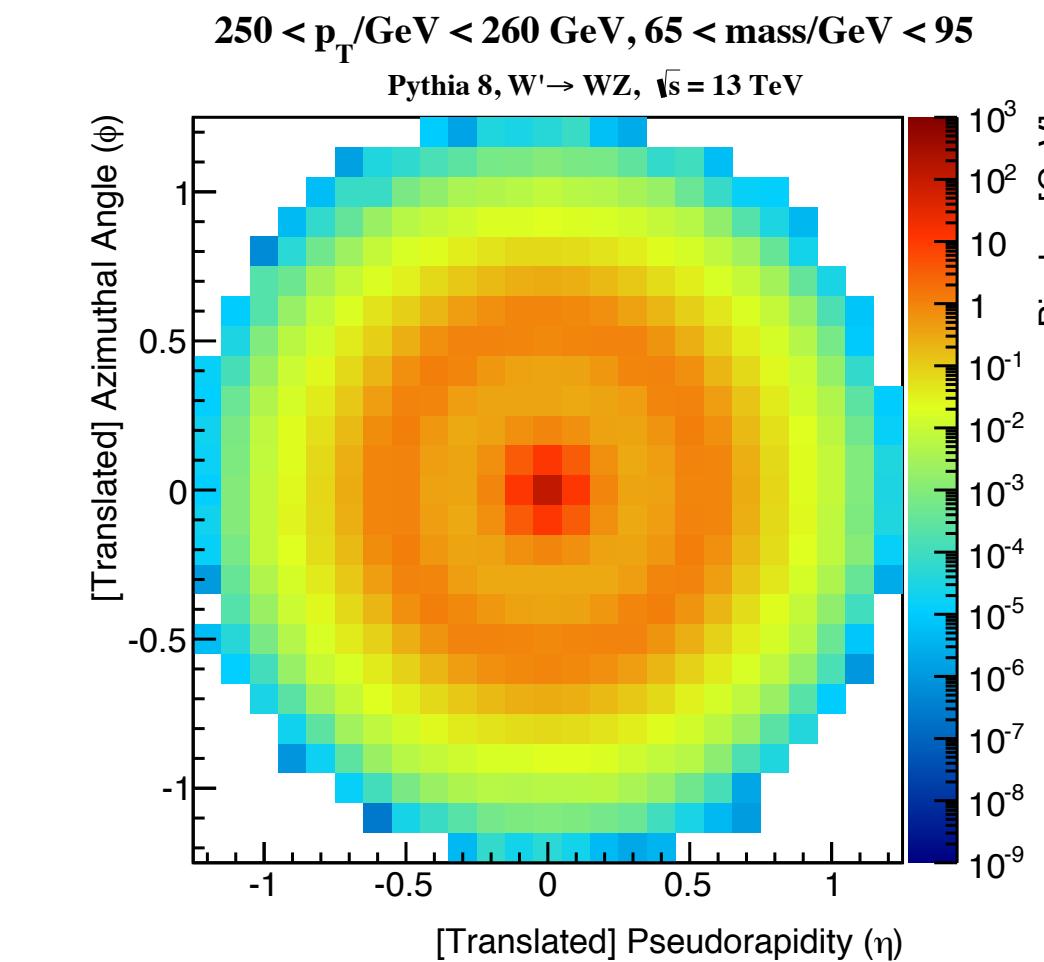
- One can pixelate the surface crossed by the jet and create an image with the momentum deposited in each cell
- Such an image can then be processed with computing-vision techniques
- Pros: can benefit of the progresses made in optimizing computing vision
- Cons: underlying assumption on detector geometry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



<https://arxiv.org/abs/1511.05190>

# 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





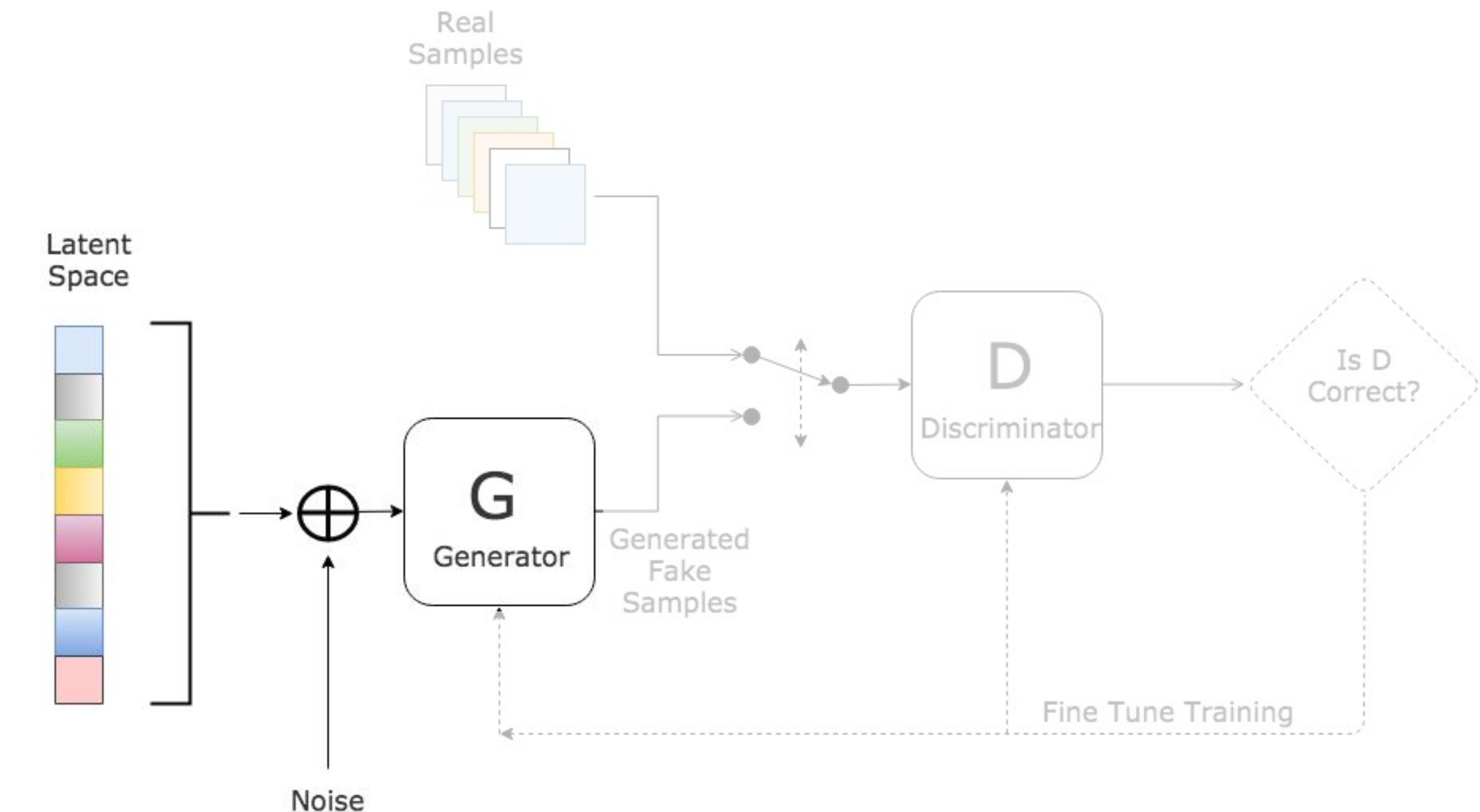
Generating large datasets  
with small resources



European  
Research  
Council

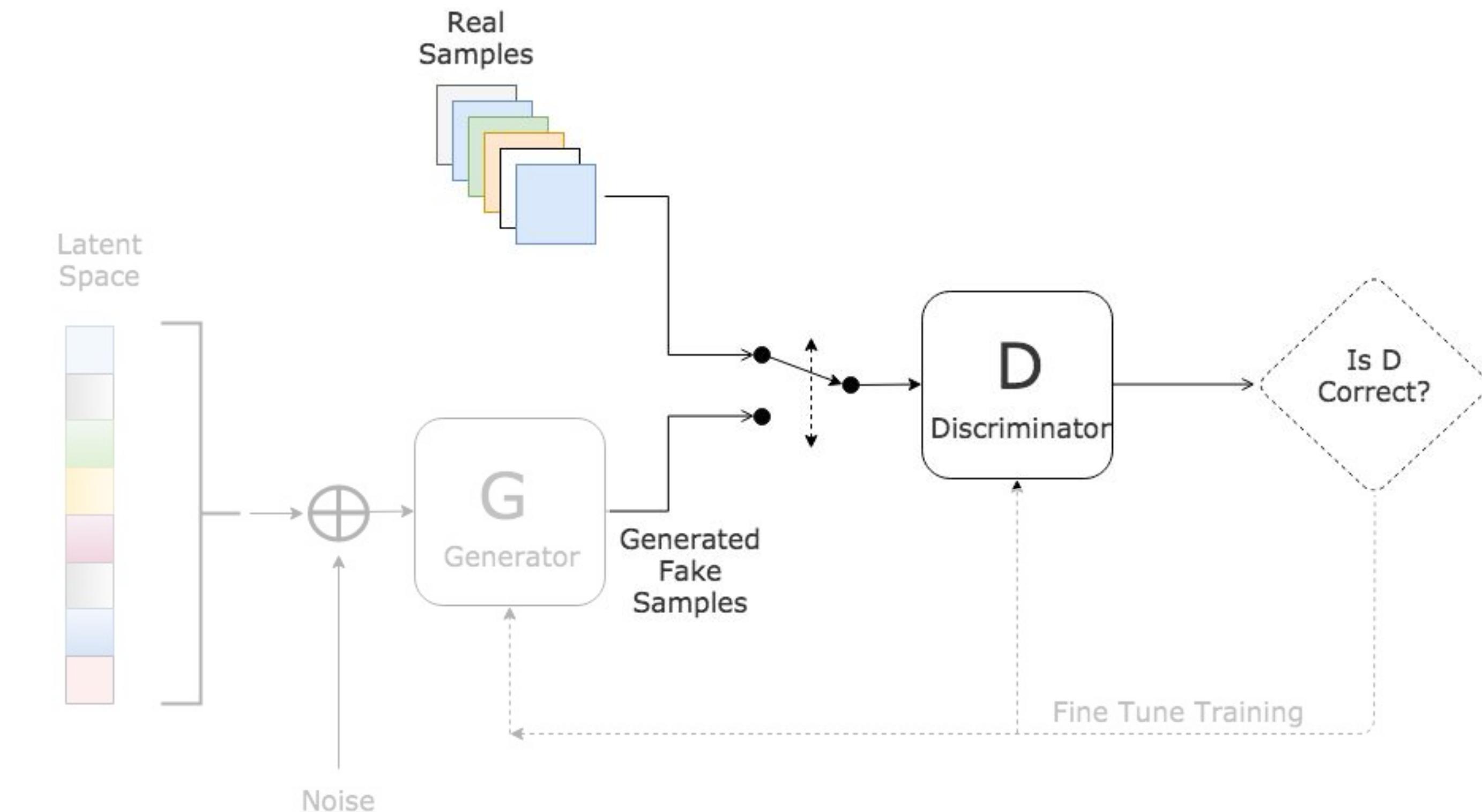
# Generative Adversarial Training

- Two networks trained against each other
- **Generator:** create images (from noise, other images, etc)
- **Discriminator:** tries to spot which image comes from the generator and which is genuine
- Loss function to minimise:  $\text{Loss}(\text{Gen}) - \text{Loss}(\text{Disc})$ 
  - Better discriminator  $\rightarrow$  bigger loss
  - Better generator  $\rightarrow$  smaller loss
  - Trying to fool the discriminator, generator learns how to create more realistic images



# Generative Adversarial Training

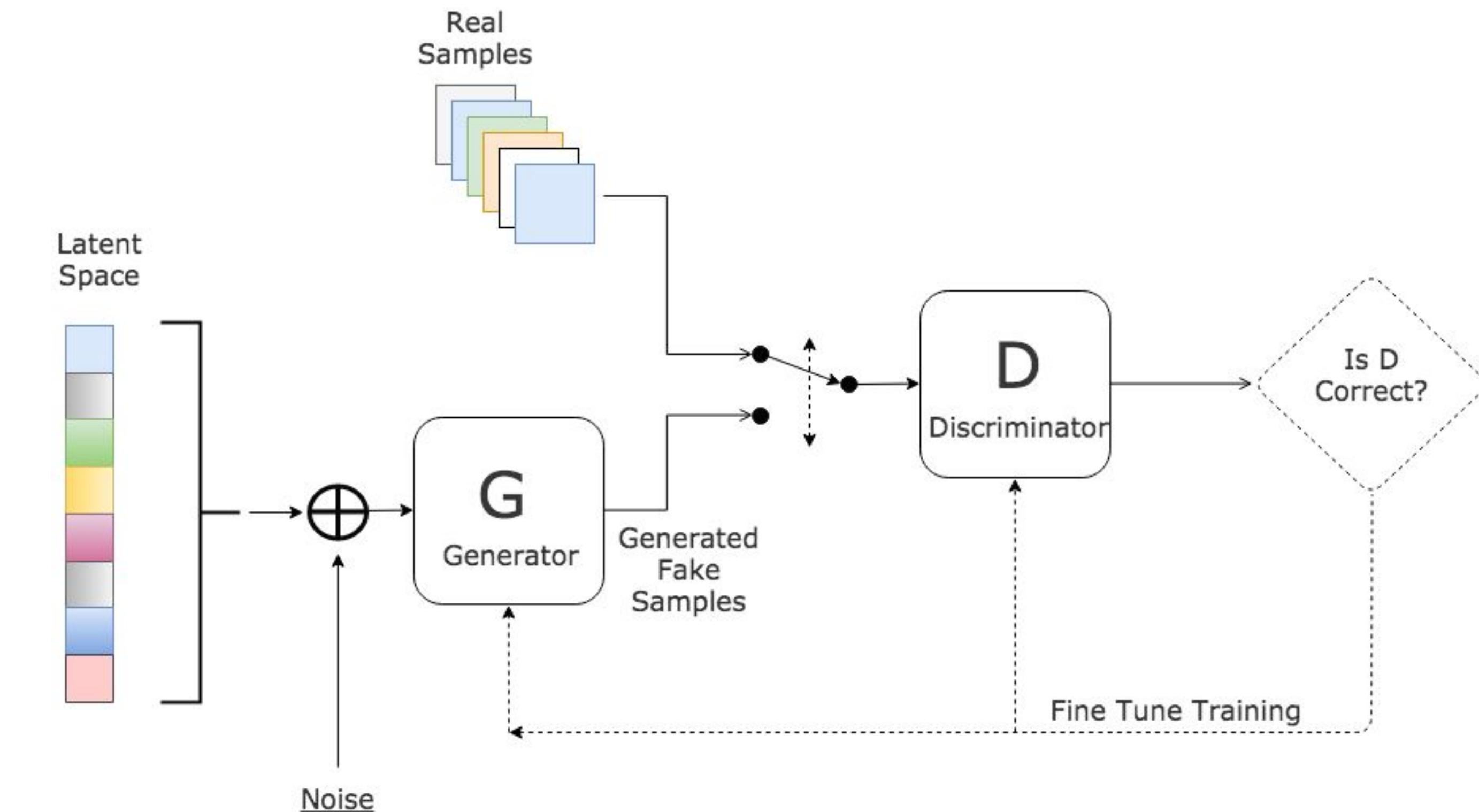
- Two networks trained against each other
- Generator: create images (from noise, other images, etc)
- Discriminator: tries to spot which image comes from the generator and which is genuine



- Loss function to minimise:  $\text{Loss}(\text{Gen}) - \text{Loss}(\text{Disc})$ 
  - Better discriminator  $\rightarrow$  bigger loss
  - Better generator  $\rightarrow$  smaller loss
  - Trying to fool the discriminator, generator learns how to create more realistic images

# Generative Adversarial Training

- Two networks trained against each other
- Generator: create images (from noise, other images, etc)
- Discriminator: tries to spot which image comes from the generator and which is genuine



- Loss function to minimise:  $\text{Loss}(\text{Gen}) - \text{Loss}(\text{Disc})$ 
  - Better discriminator  $\rightarrow$  bigger loss
  - Better generator  $\rightarrow$  smaller loss
  - Trying to fool the discriminator, generator learns how to create more realistic images



# Generative Adversarial Training



PROGRESSIVE GROWING OF GANs FOR IMPROVED  
QUALITY, STABILITY, AND VARIATION

Submitted to ICLR 2018



European  
Research  
Council

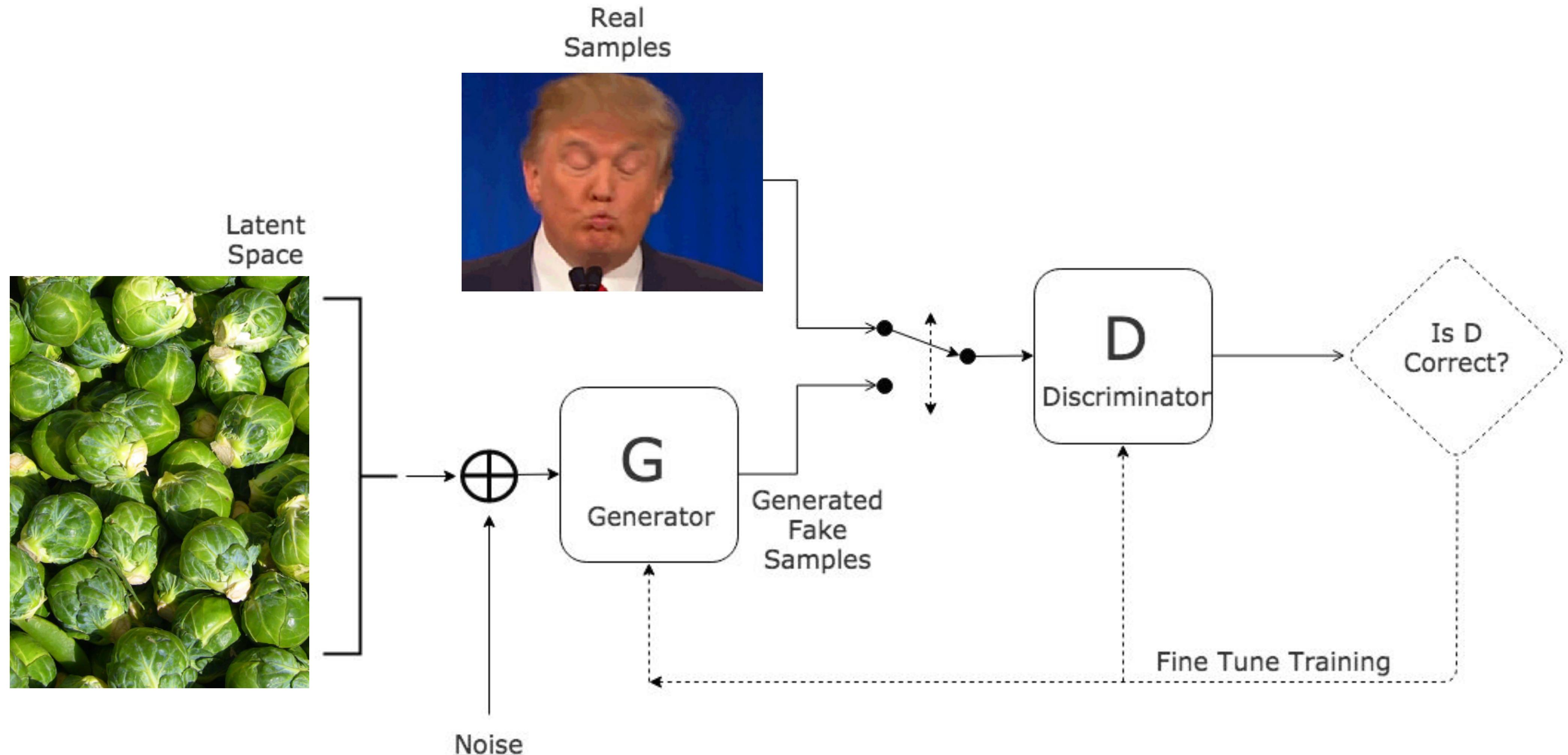


# Generative Adversarial Training



European  
Research  
Council

# Adversarial training in azione



# Adversarial training in azione



# Adversarial training in azione



Source Sequence

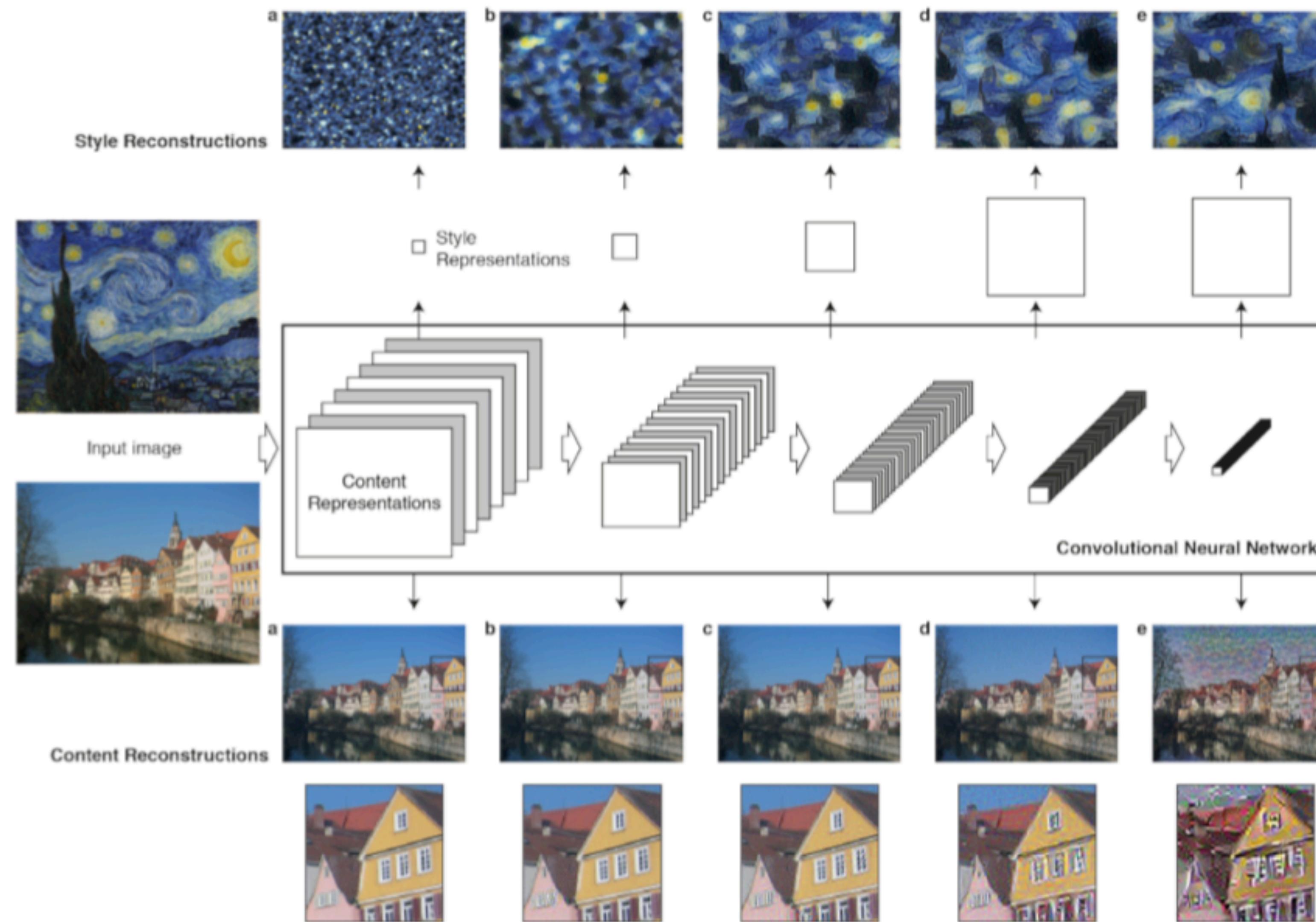


Our Reenactment  
(Full Head)

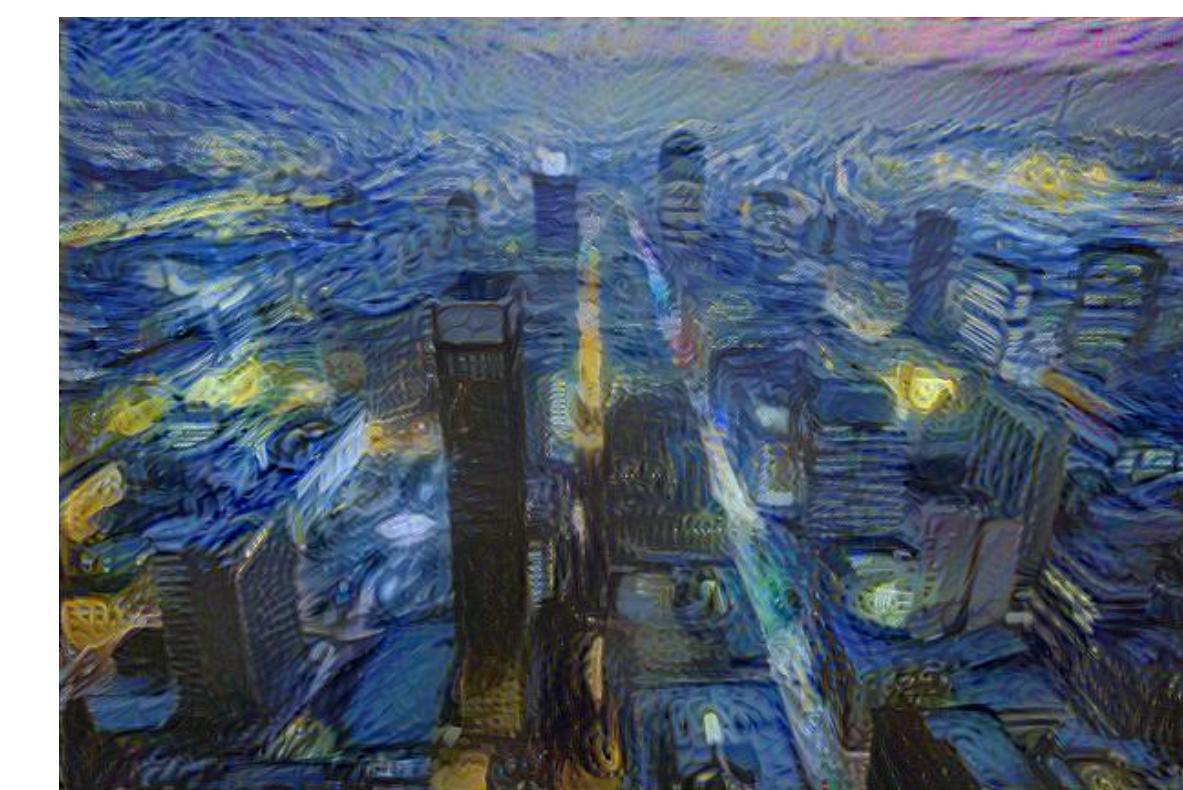
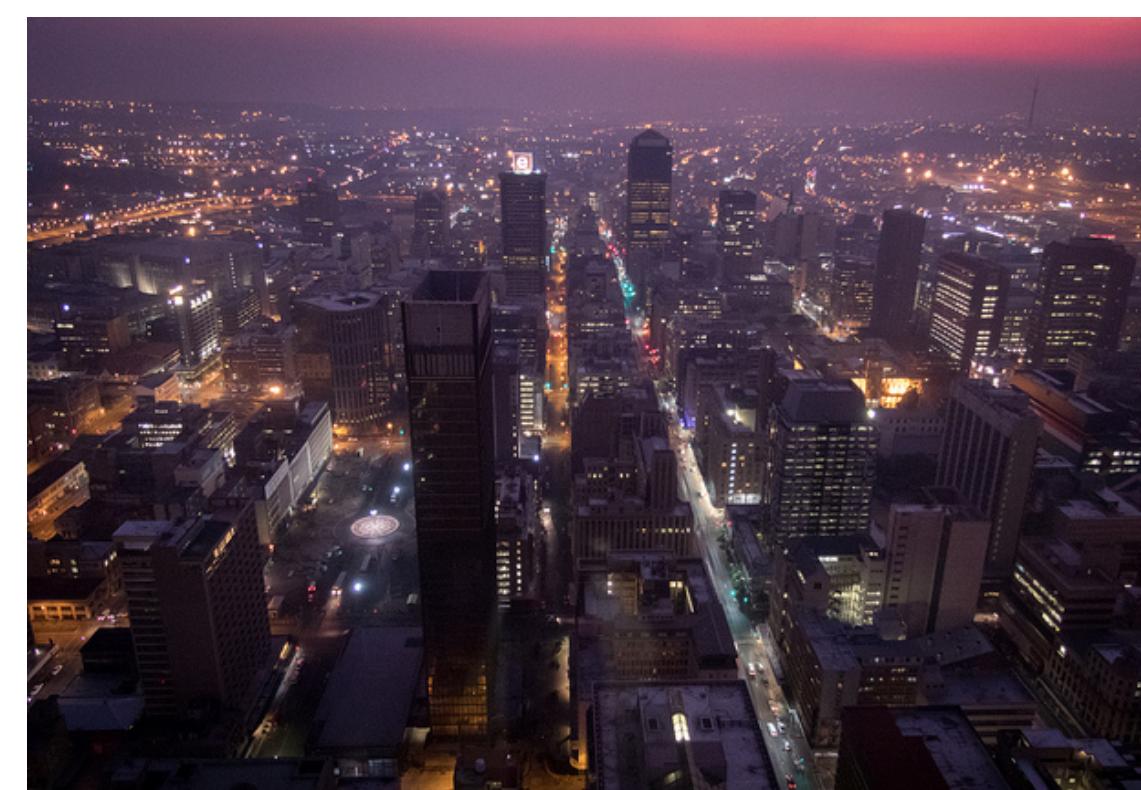
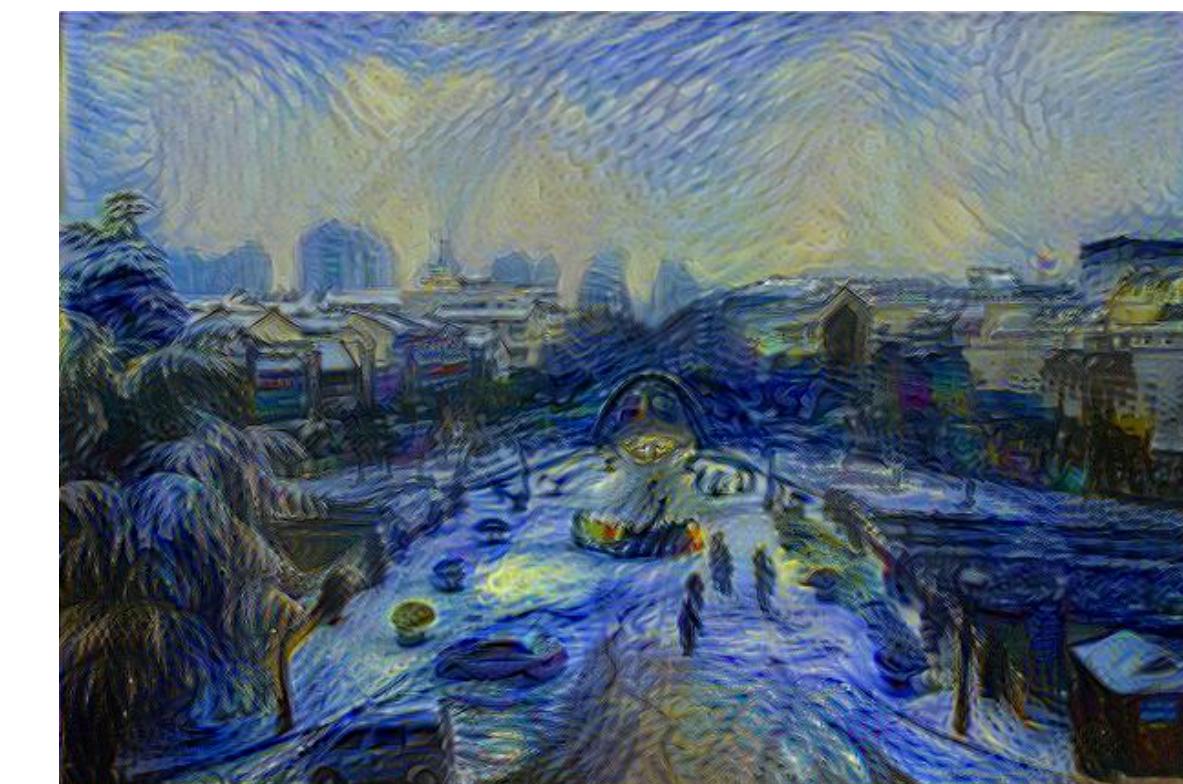
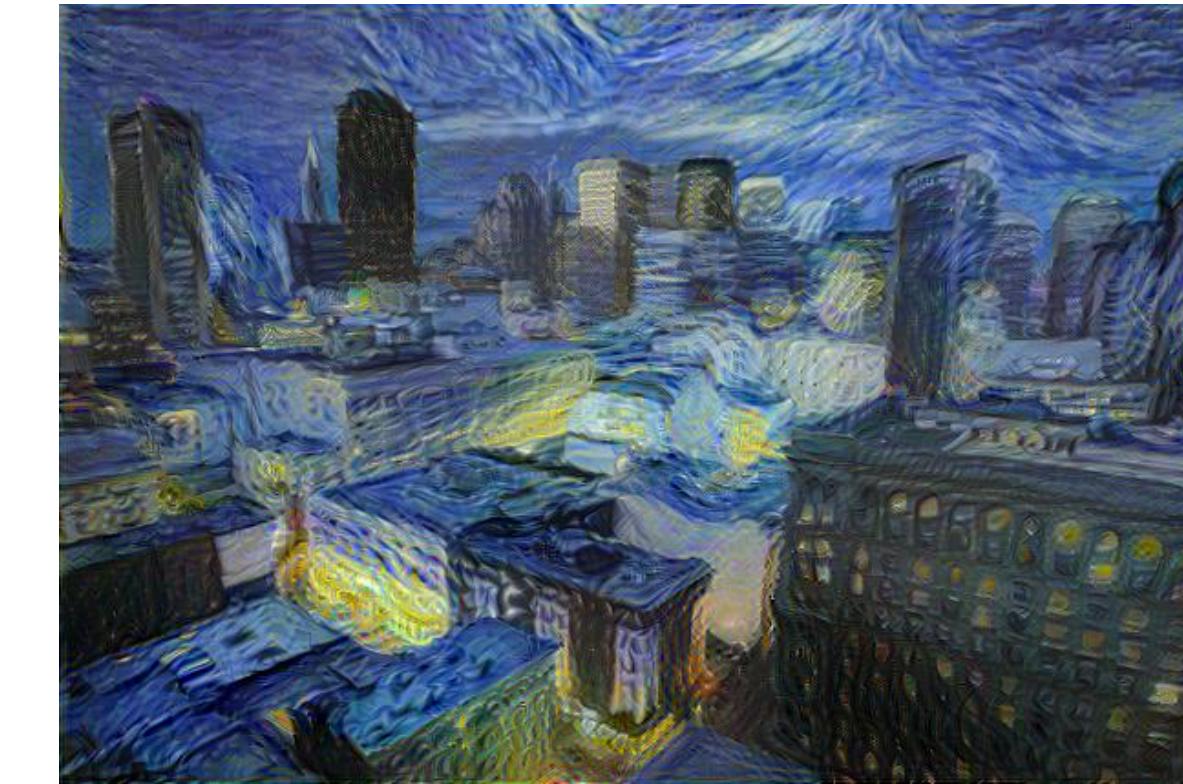


Averbuch-Elor et al. 2017

# Style Transfer



# Style Transfer





# Style Transfer

Artistic style transfer for videos

Manuel Ruder  
Alexey Dosovitskiy  
Thomas Brox

University of Freiburg  
Chair of Pattern Recognition and Image Processing

# minimax game

- *The loss function is*

$$\min_G \max_D V(D, G) = \mathbb{E}_{\mathbf{x} \sim p_{\text{data}}(\mathbf{x})} [\log D(\mathbf{x})] + \mathbb{E}_{\mathbf{z} \sim p_{\mathbf{z}}(\mathbf{z})} [\log(1 - D(G(\mathbf{z})))].$$

The generator should aim to return 1 for real and 0 for kales

The generator should aim to return 1 for fakes

- *GAN training is a minimax (or zero-sum) problem:*

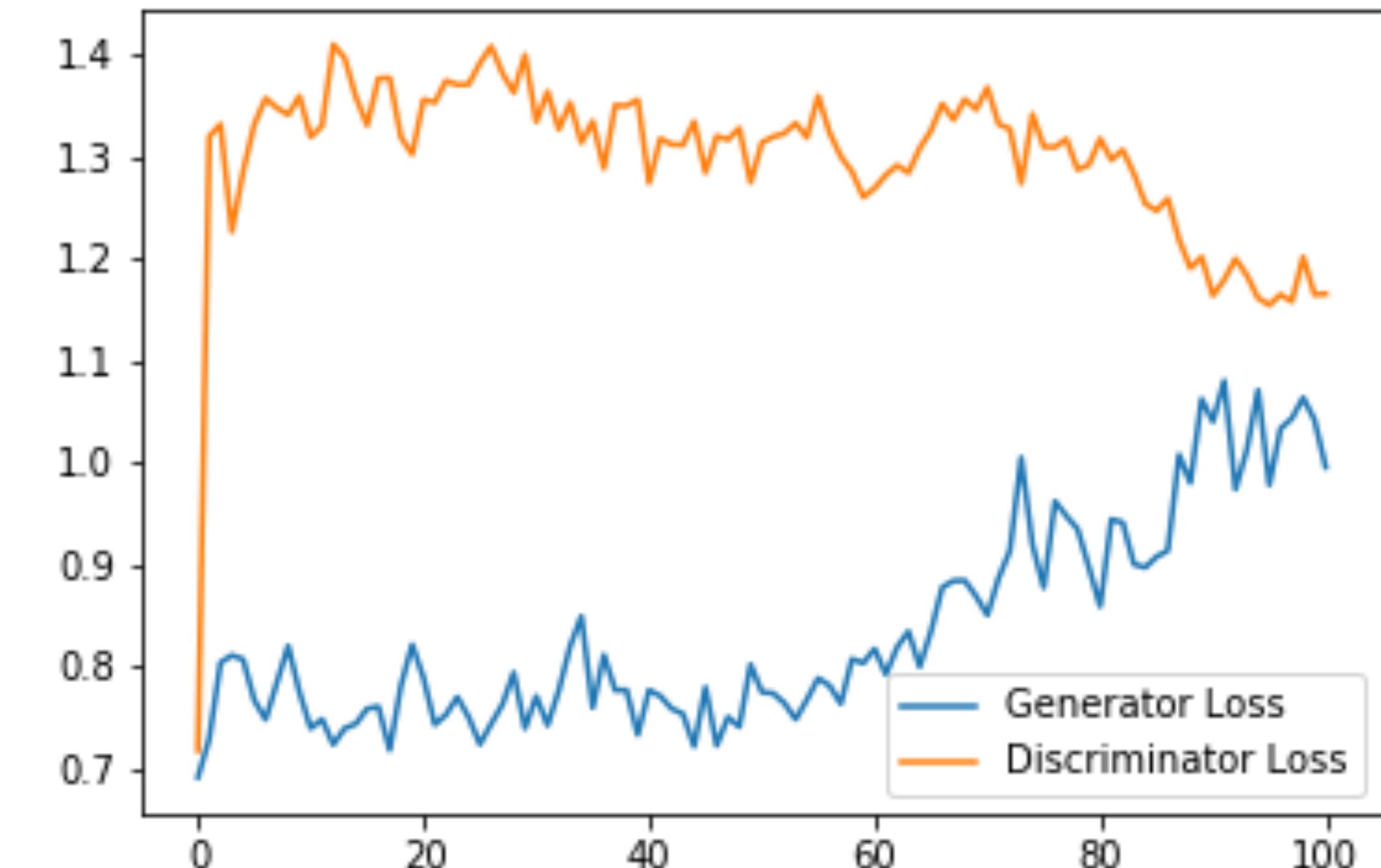
- *one wins, the other looses*

- *The problem is that (unless one is close to the right answer to start with) the training can be quite hard:*

- *oscillations*

- *mode collapse*

- *etc.*



# What does the loss represent?

- At its best, the loss can be written as

$$L(G, D^*) = 2D_{JS}(p_r \| p_g) - 2 \log 2$$

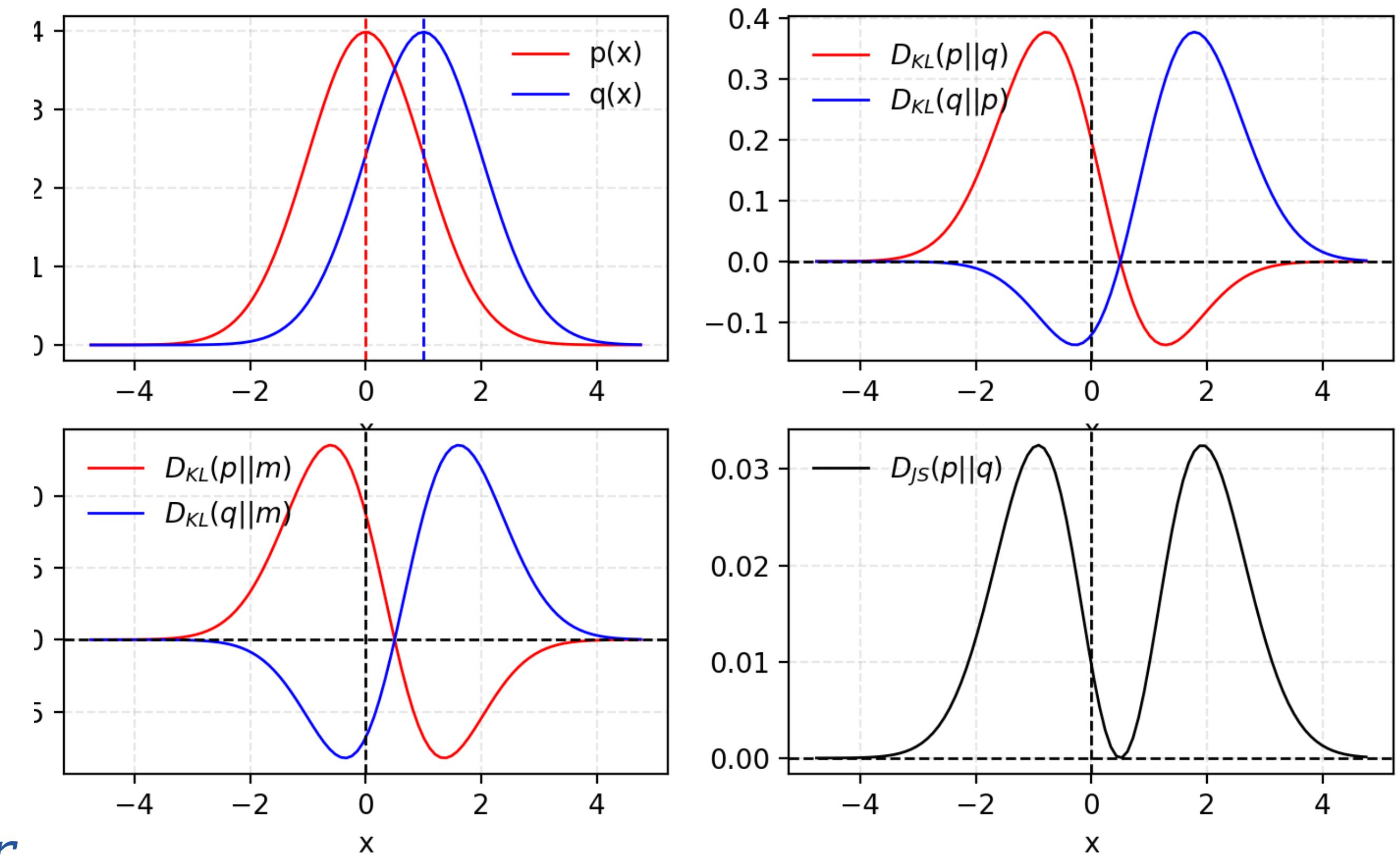
- This means, the optimal is reached when the distance between the GAN images and the real ones are zero

- The distance, in this context, is given by the Jensen-Shannon divergence

$$D_{JS}(p \| q) = \frac{1}{2}D_{KL}(p \| \frac{p+q}{2}) + \frac{1}{2}D_{KL}(q \| \frac{p+q}{2})$$

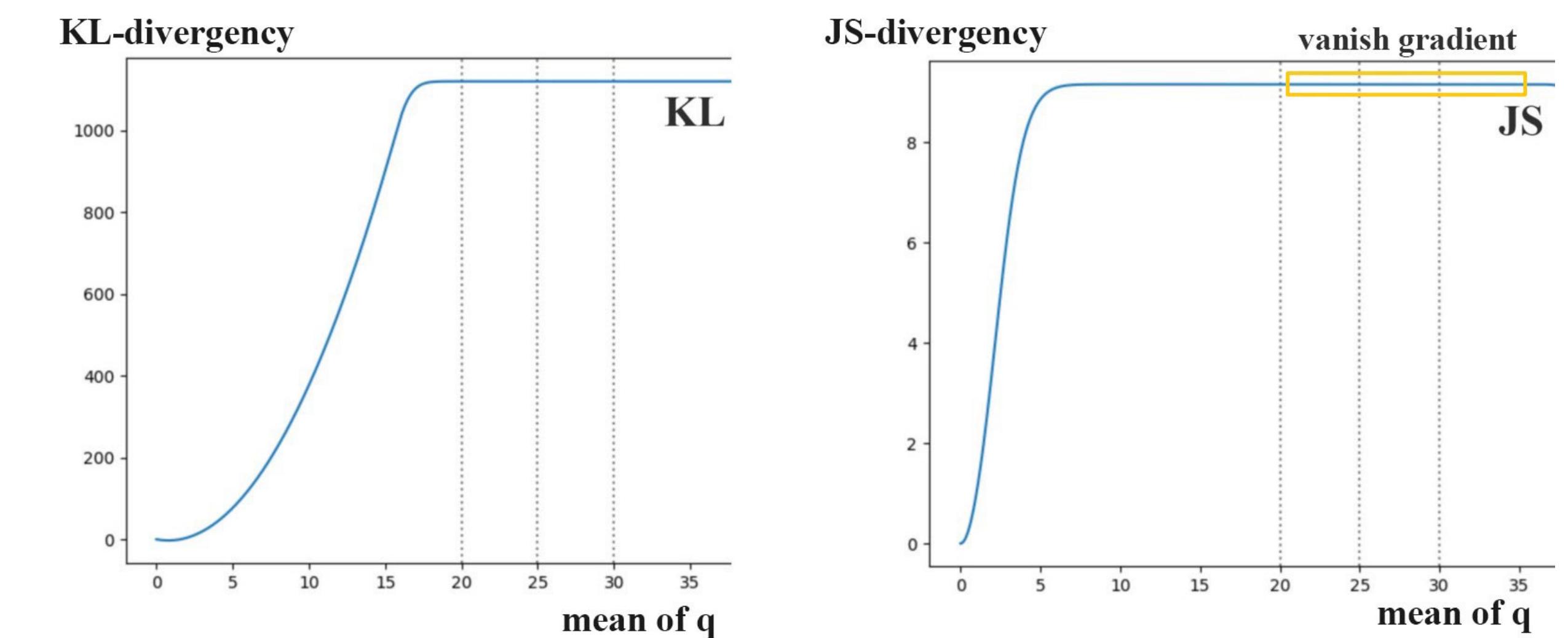
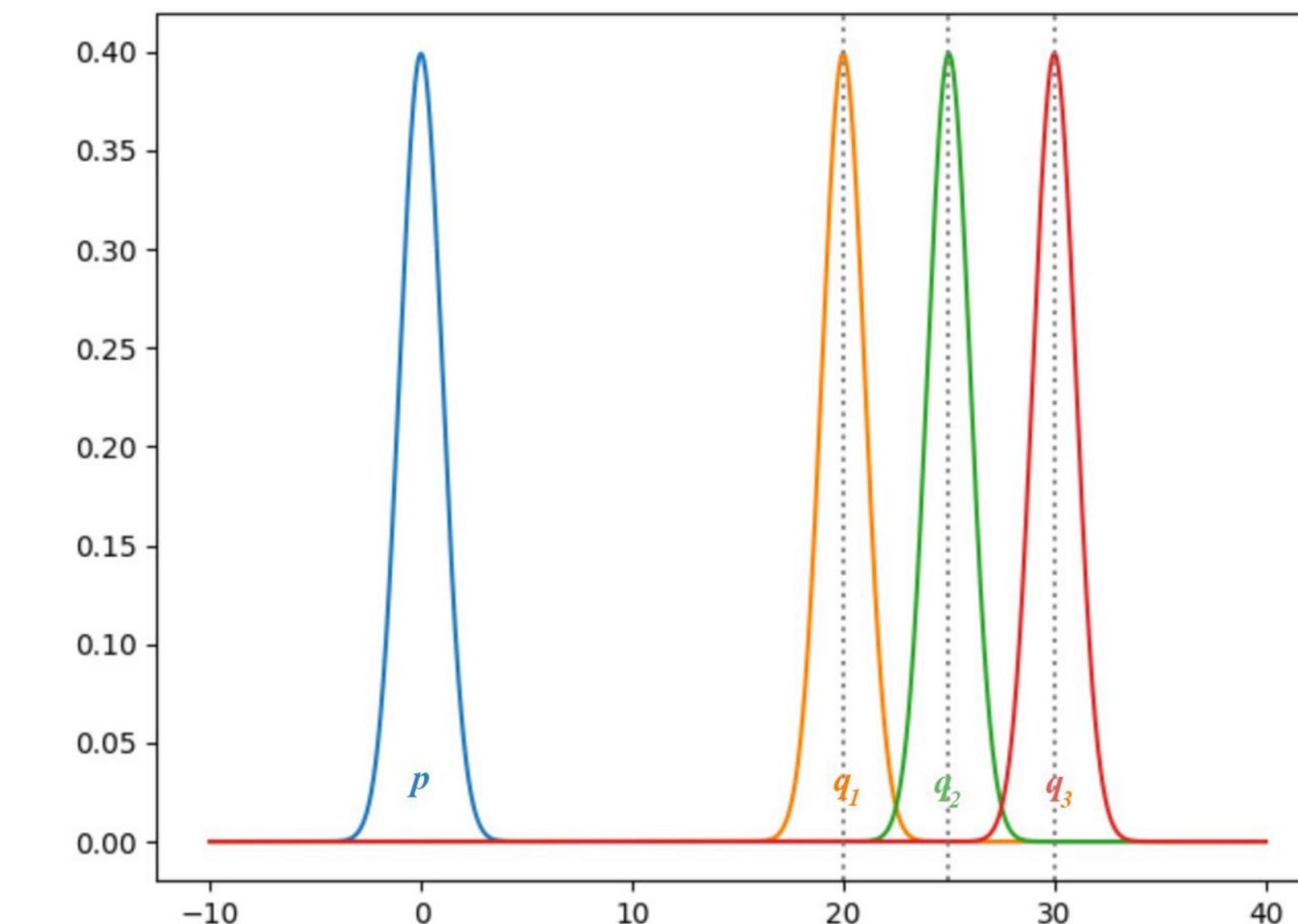
where  $D_{KL}$  is the Kullback-Leibler divergence

$$D_{KL}(p \| q) = \int_x p(x) \log \frac{p(x)}{q(x)} dx$$



# Zero gradient

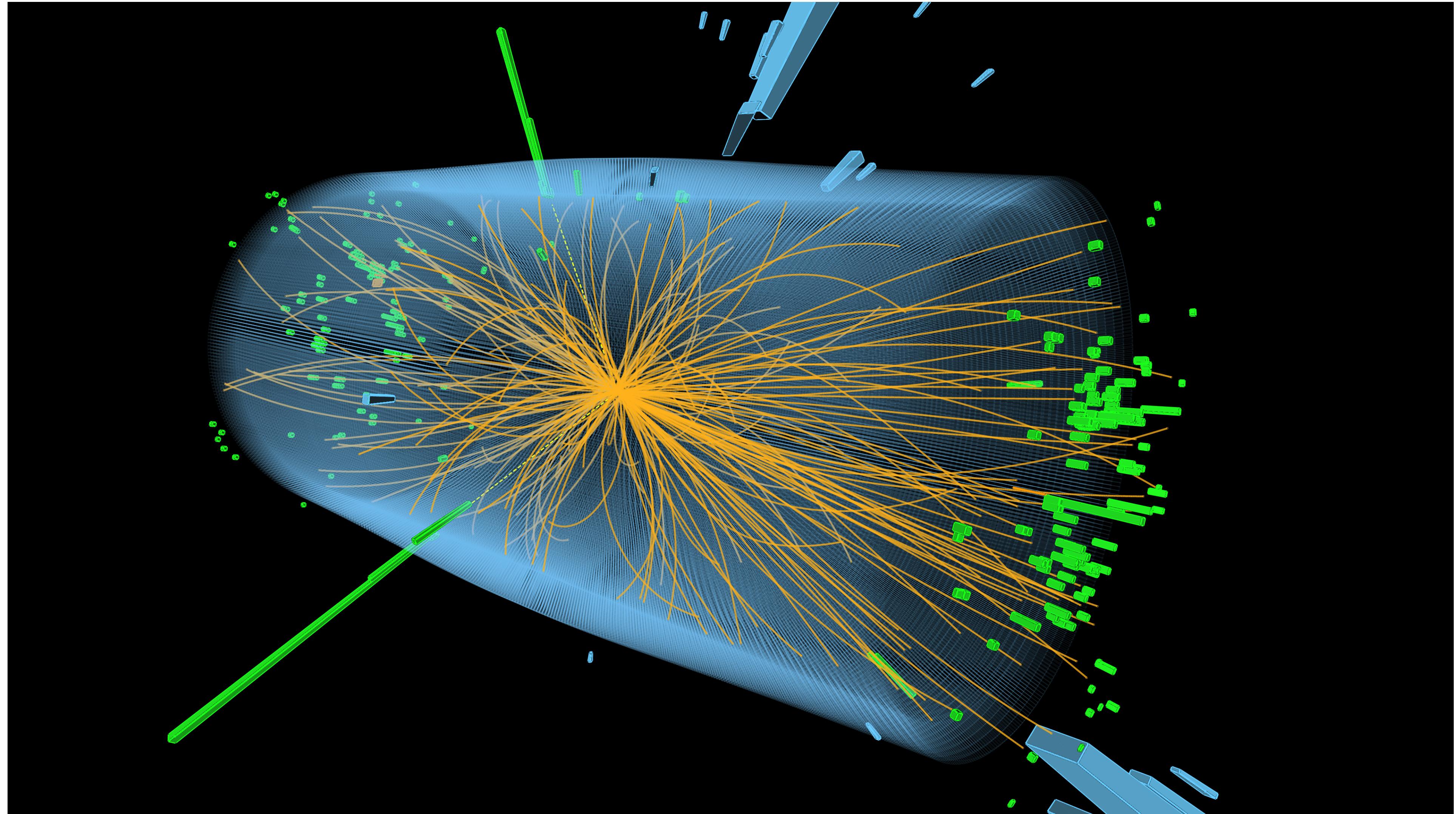
- *Training the classifier is much easier than training the discriminator*
- *When the discriminator is way off, the gradient is flat and the discriminator cannot be improved with a sgd*



# Alternative approaches

- *Several proposes to stabilize the gradient*
- *Set of thumb rules to help convergence*
- *Changing the loss expression while keeping the desired behavior (LSGAN)*
- *Changing the metric for the fake/real distance (wGAN)*
- *Evolving field of research*

# What can we use this for?

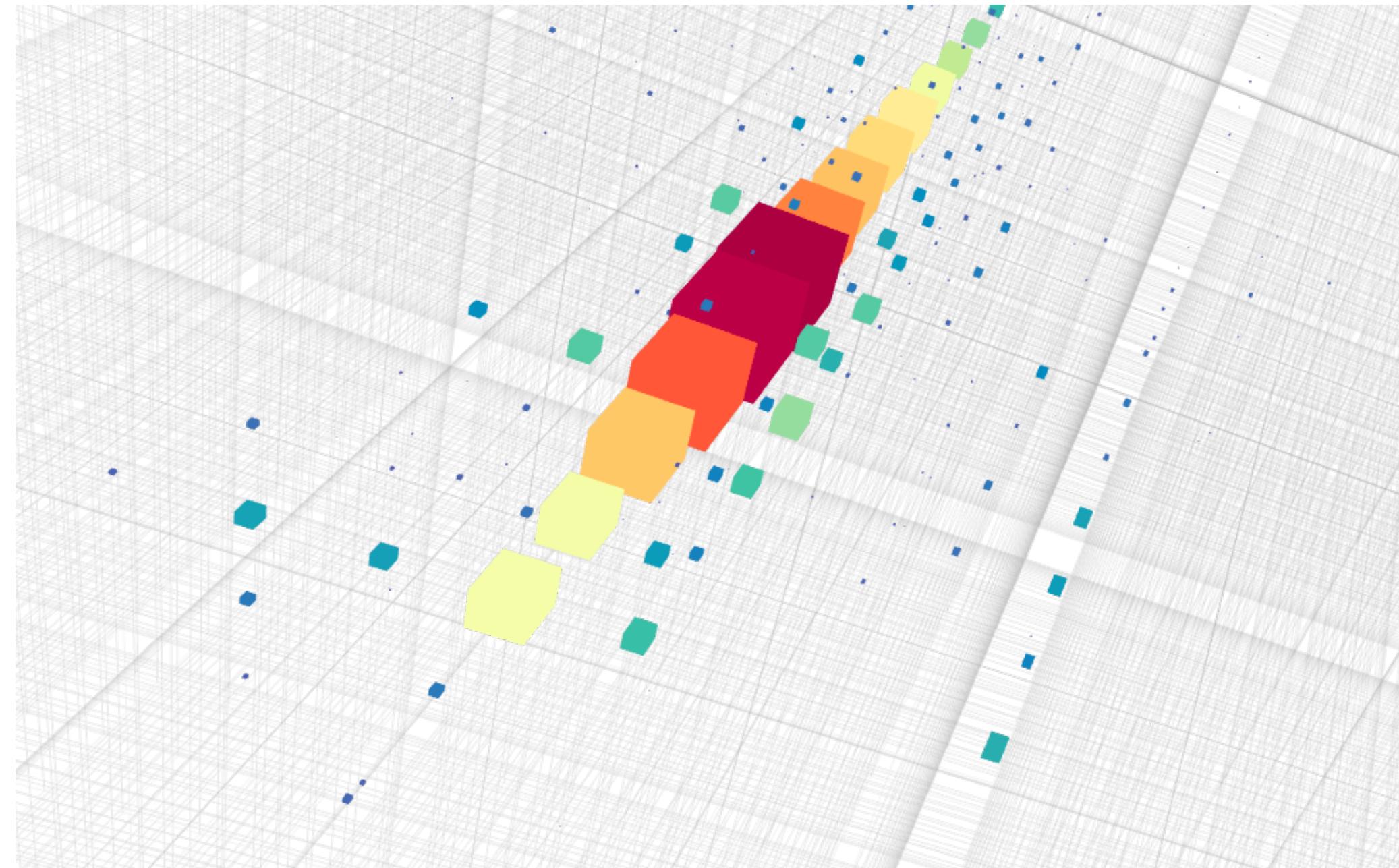


# Particle shower generation

See contribution to NIPS workshop

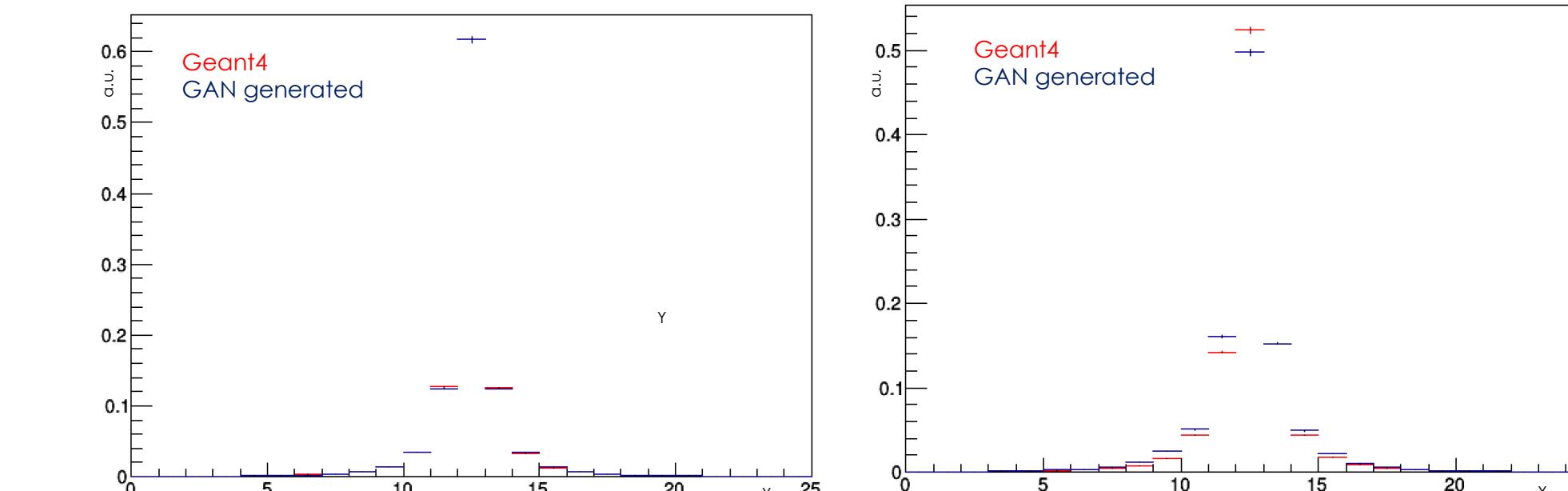
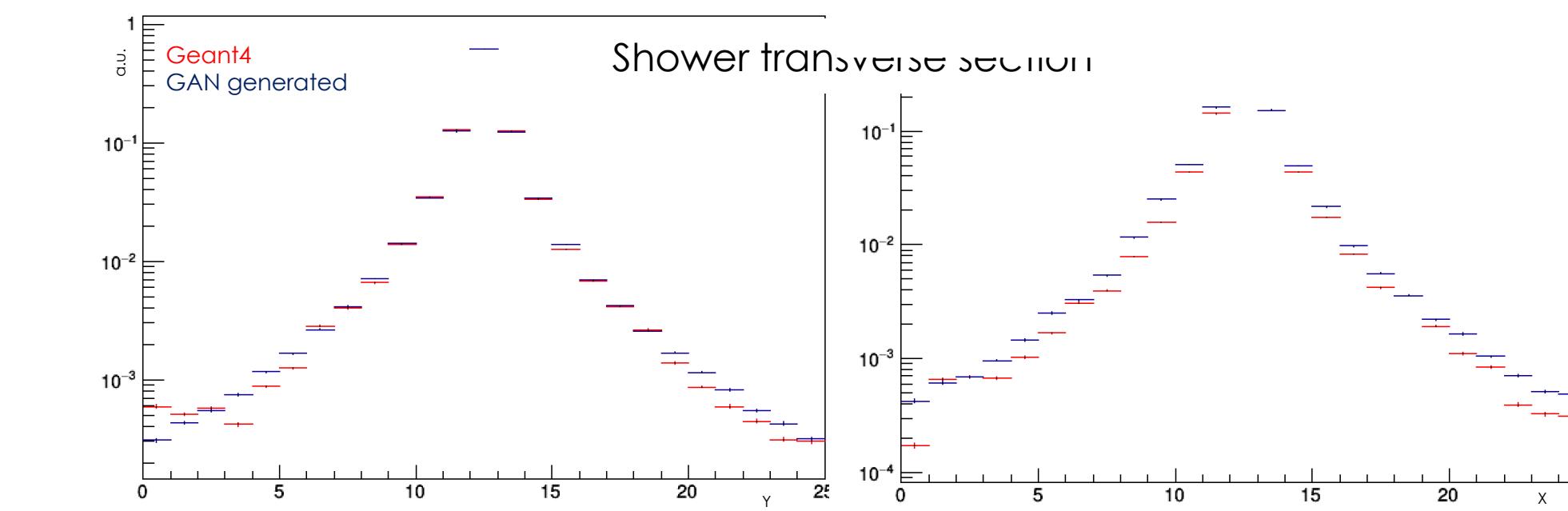
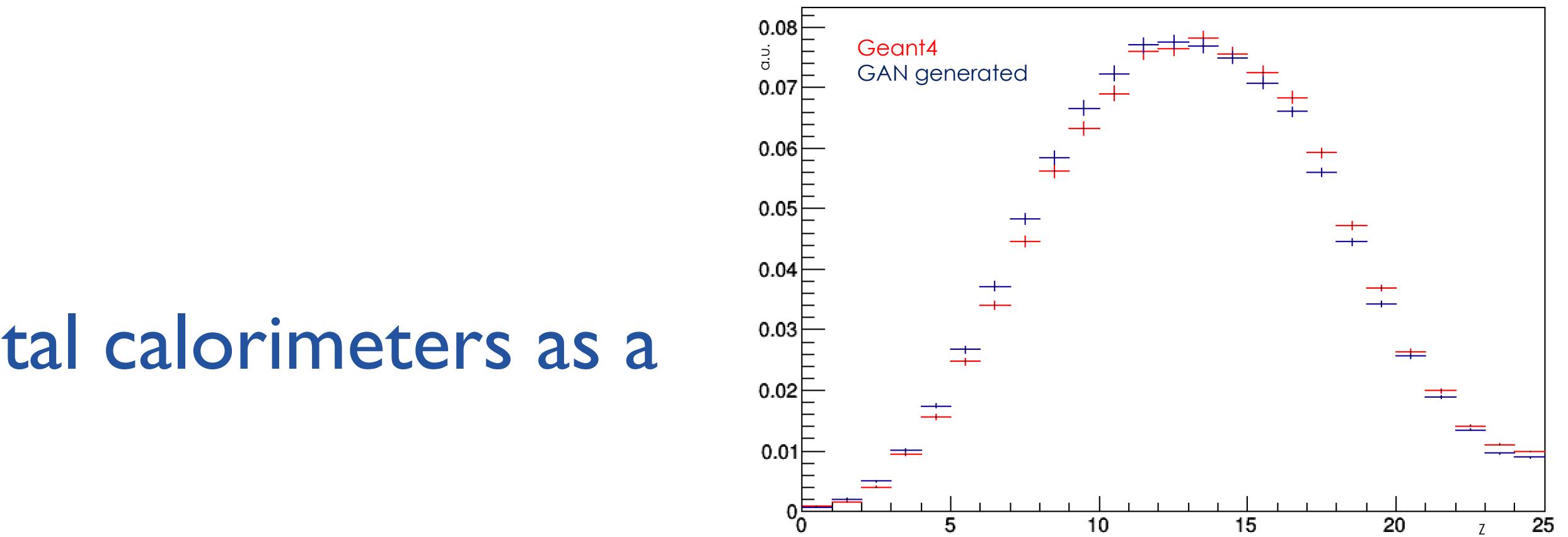
Shower longitudinal section

- Start from random noise
- Works very well with images
- Applied to electron showers in digital calorimeters as a replacement of GEANT



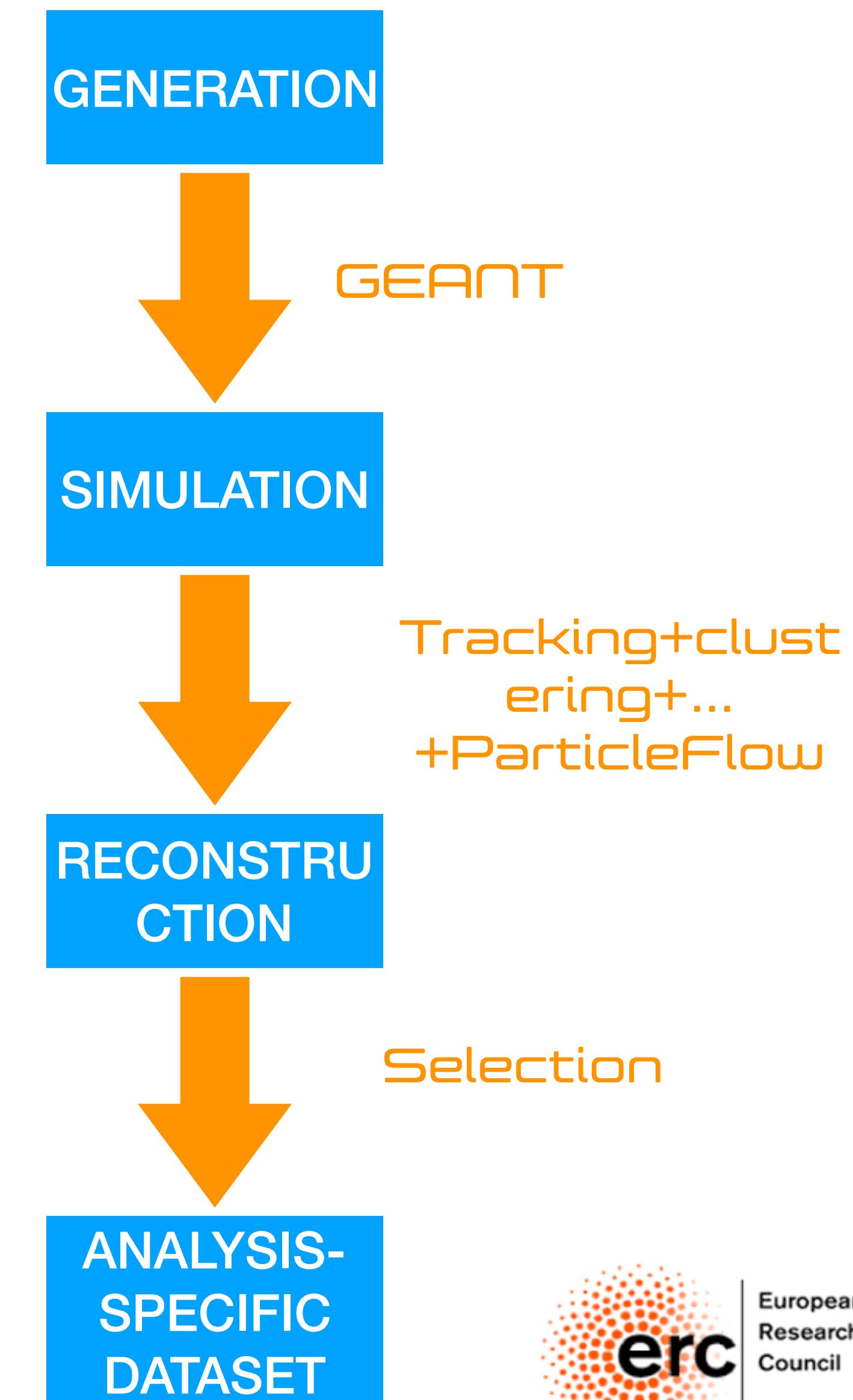
see also de Olivera, Paganini, and Nachman

<https://arxiv.org/abs/1712.10321>



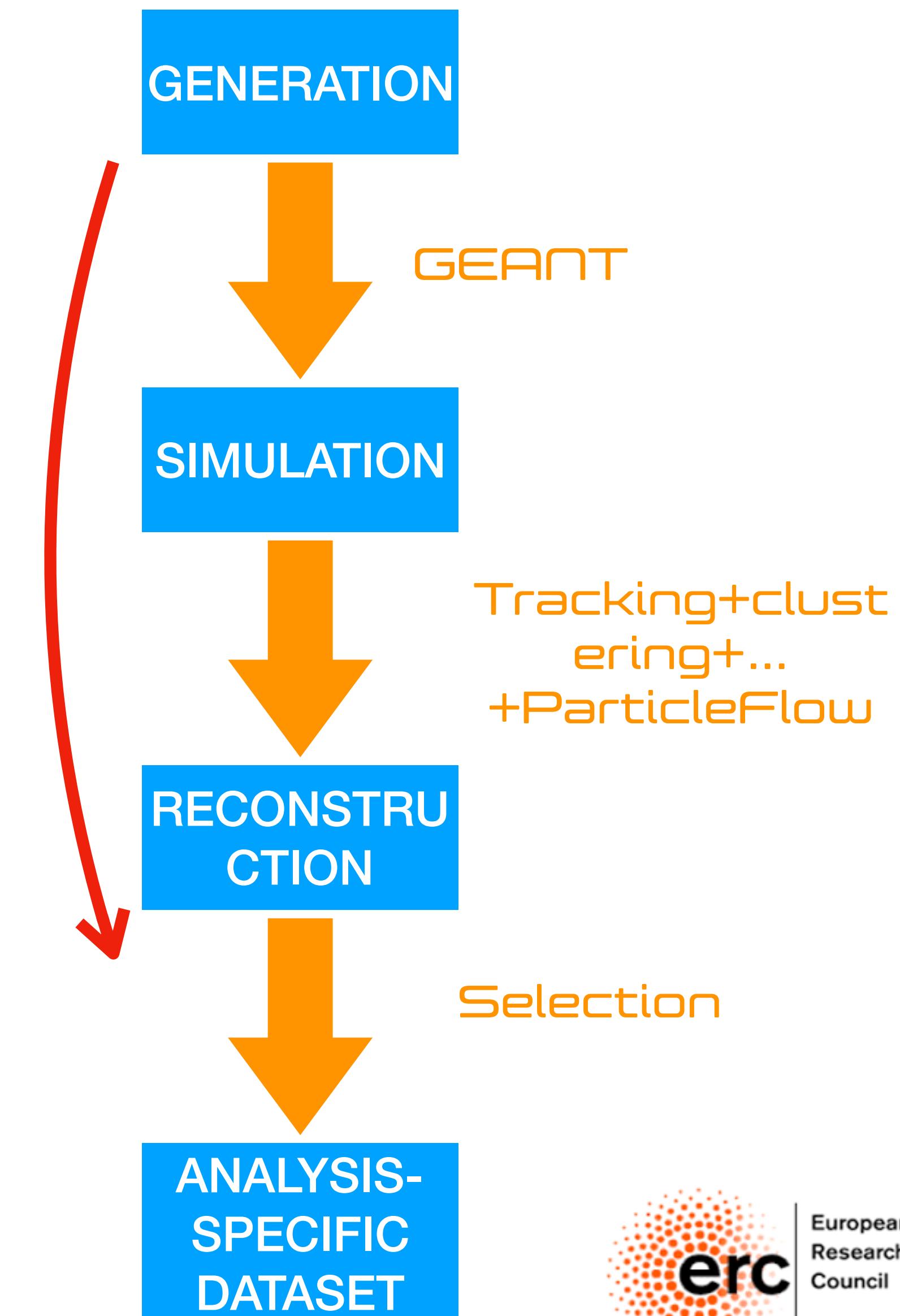
# Simulation is half of the problem

- *The simulation step (proton collision) and the interaction of the produced particles with the detector) is 1/2 of the CPU load in a simulation workflow*
- *Its output needs to be digitised (emulation of detector electronics) to look like recorded data*
- *The digitised event is then processed by standard reconstruction code (as heavy as the SIM step) to then be used in analysis*



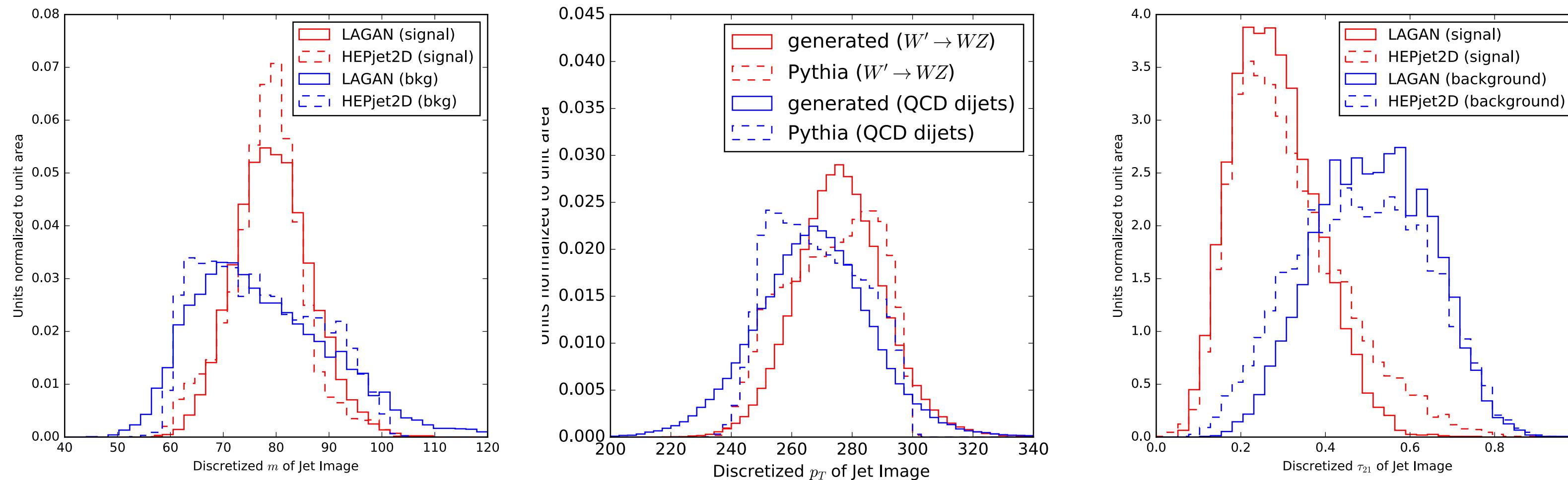
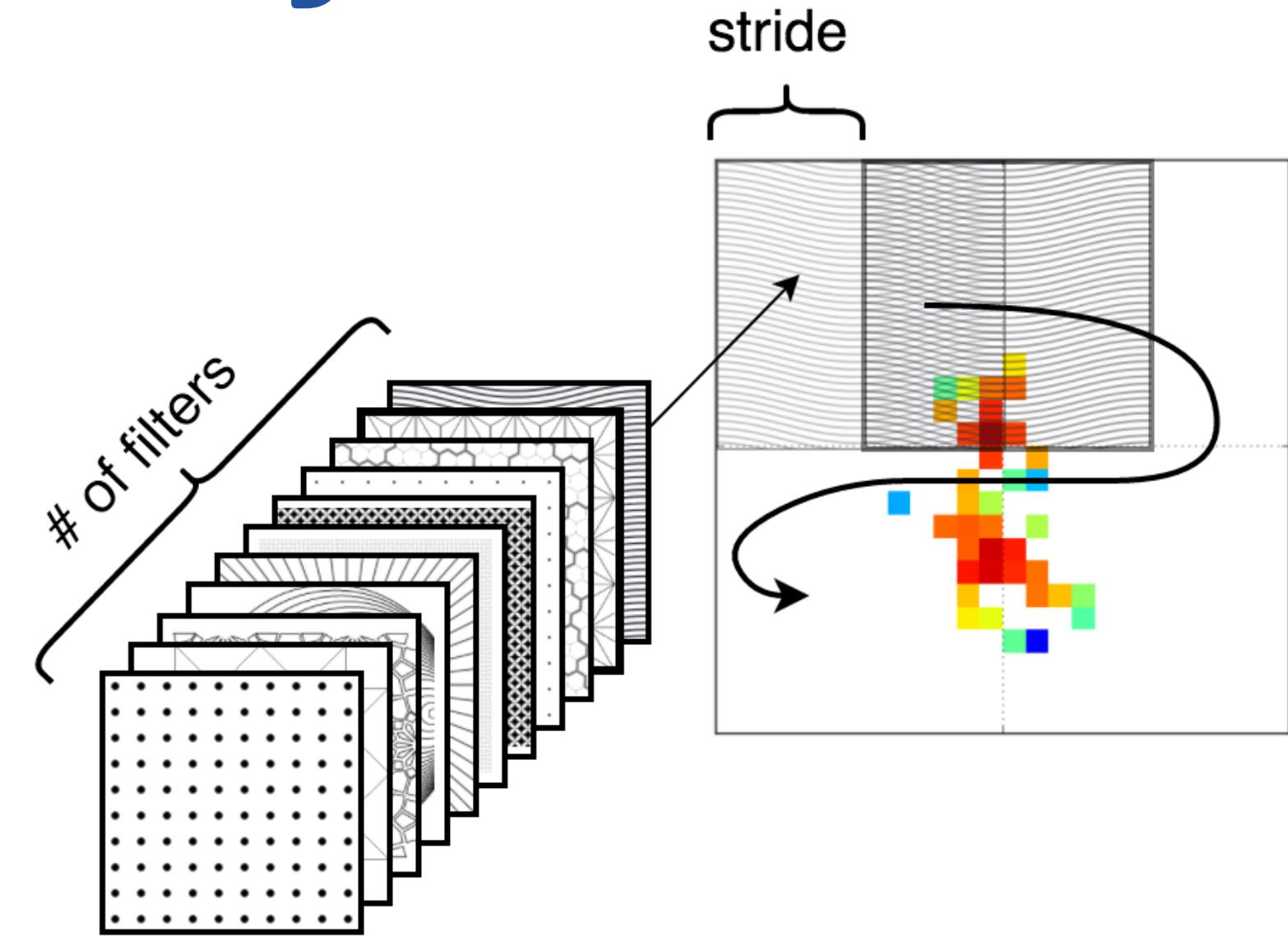
# Simulation is half of the problem

- Reconstruction involves more than one detector (e.g., tracker + calorimeter) and produces (at least in CMS) a list of particles
- GANs were proved to be useful to emulate the SIM+RECO step in one go!
- jets out of full particle reconstruction emulated in a GAN
- trained on actual SIM+RECO synthetic data by CMS



# Generating full jets

- Start from random noise
- Works very well with images
- Applied to electron showers in digital calorimeters as a replacement of GEANT

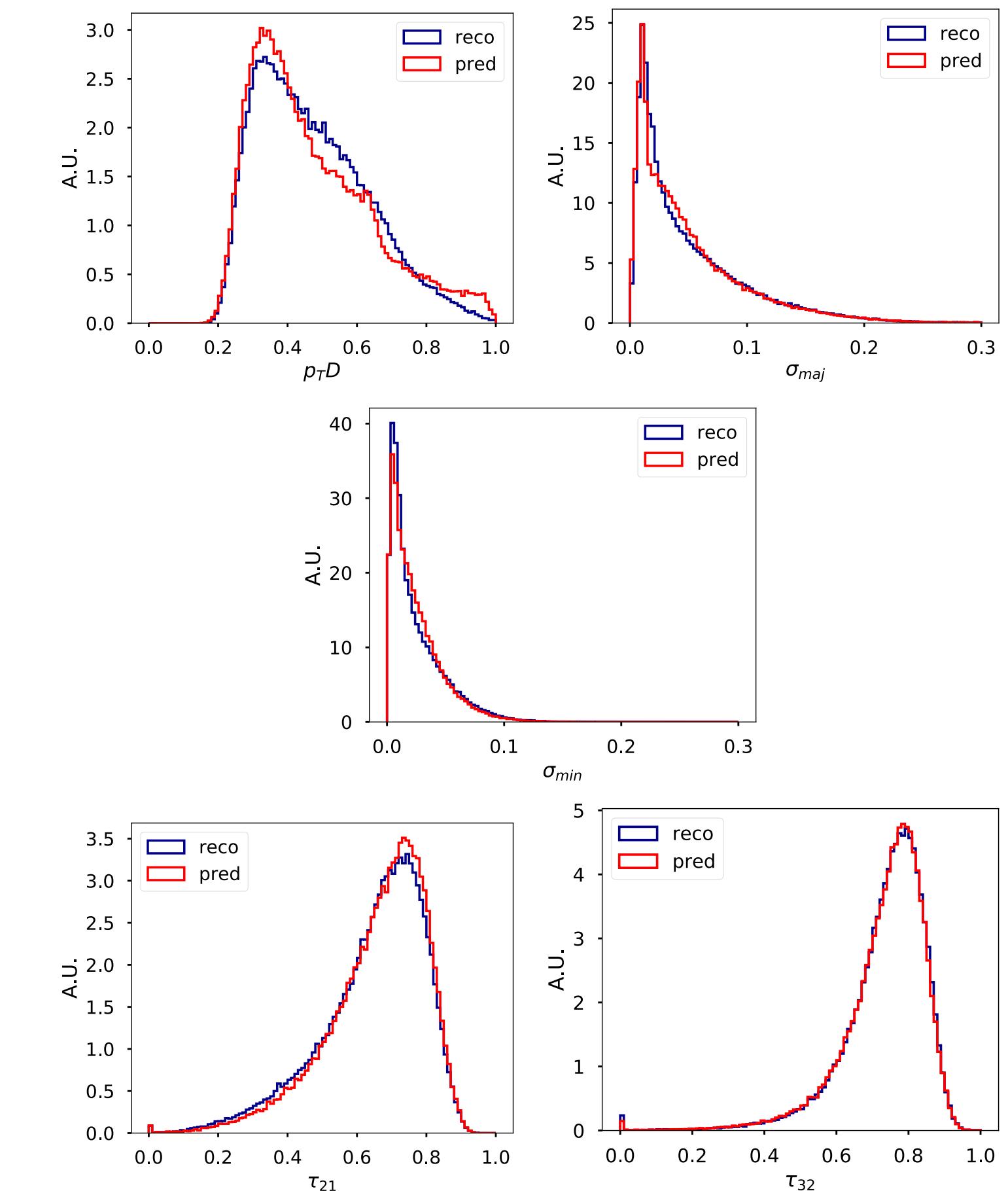


**de Olivera, Paganini, and Nachman**  
<https://arxiv.org/pdf/1701.05927.pdf>

**Figure 6:** The distributions of image mass  $m(I)$ , transverse momentum  $p_T(I)$ , and  $n$ -subjettiness  $\tau_{21}(I)$ . See the text for definitions.

# Simulation is half of the problem

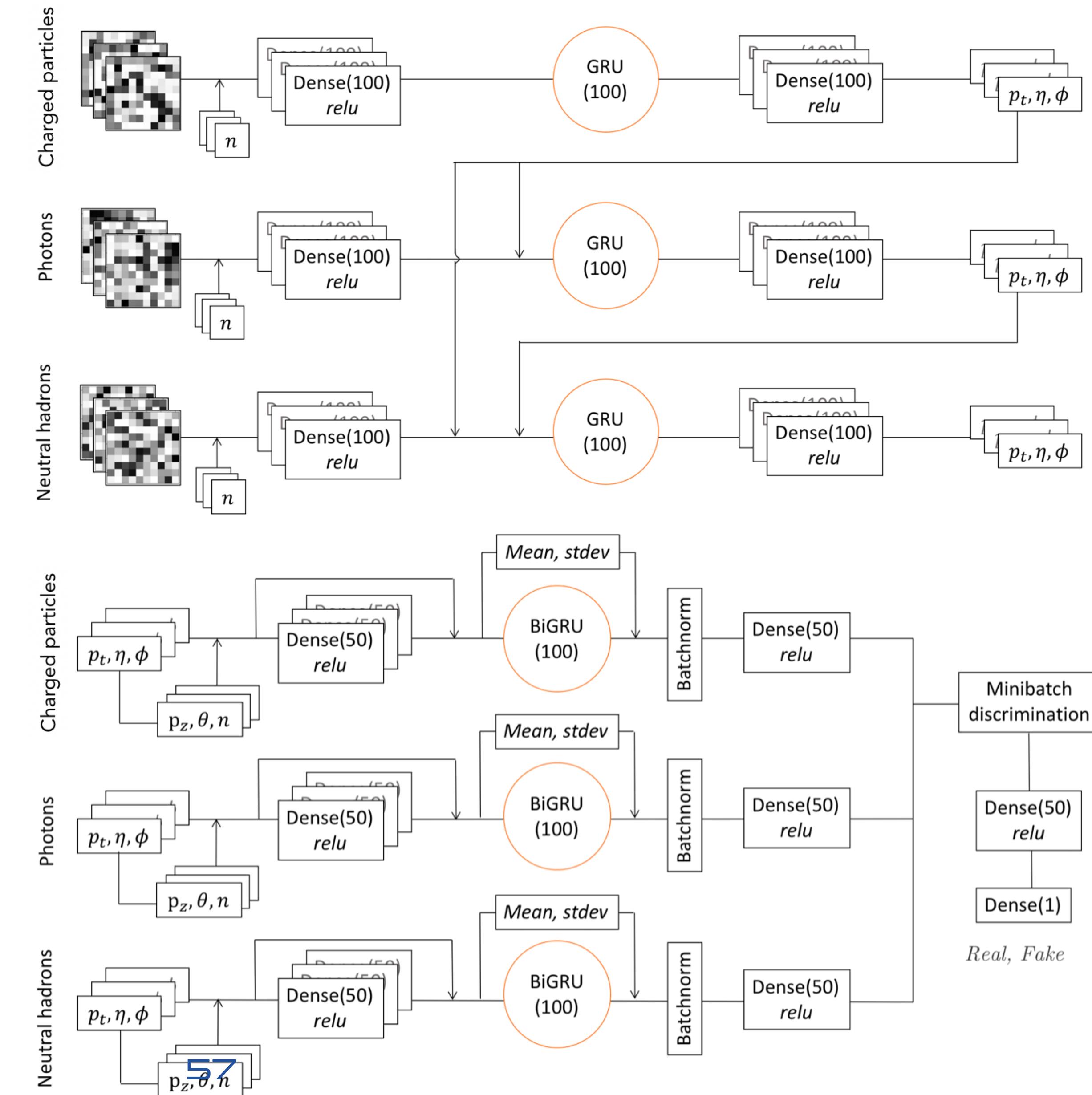
- Reconstruction involves more than one detector (e.g., tracker + calorimeter) and produces (at least in CMS) a list of particles
- GANs were proved to be useful to emulate the SIM+RECO step in one go!
- jets out of full particle reconstruction emulated in a GAN
- trained on actual SIM+RECO synthetic data by CMS



**Fig. 6** Distribution of high level variables used for quark/gluon discrimination (first two rows) and merged jets tagging (last row). Blue histograms are obtained from the input data, while red ones are obtained using the generative model.

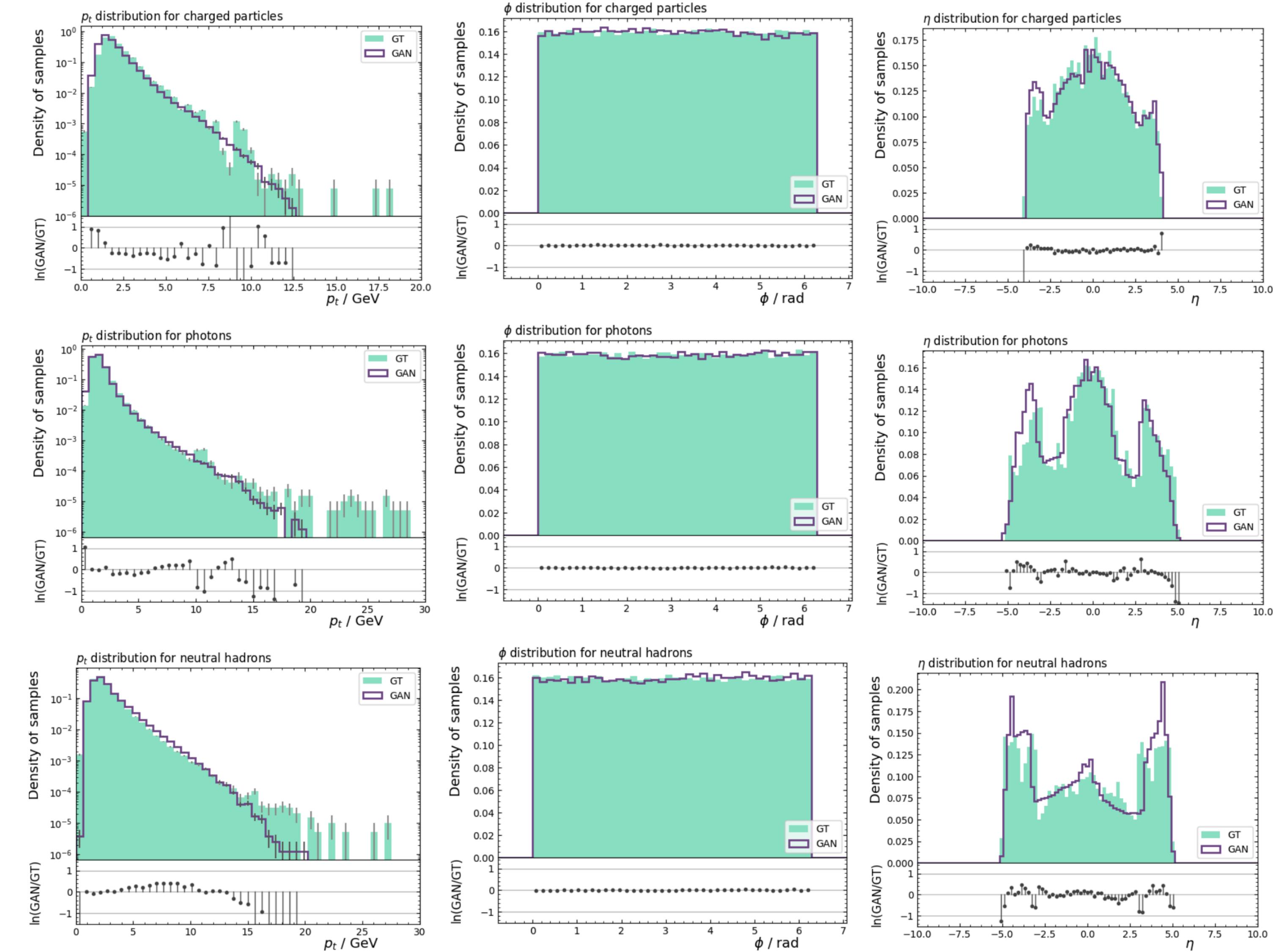
# Generating particles

- *Downstream reconstruction assumes particles as input*
- *Generating images is not practical*
- *Even in this case, a change of architecture can produce an adequate event representation*



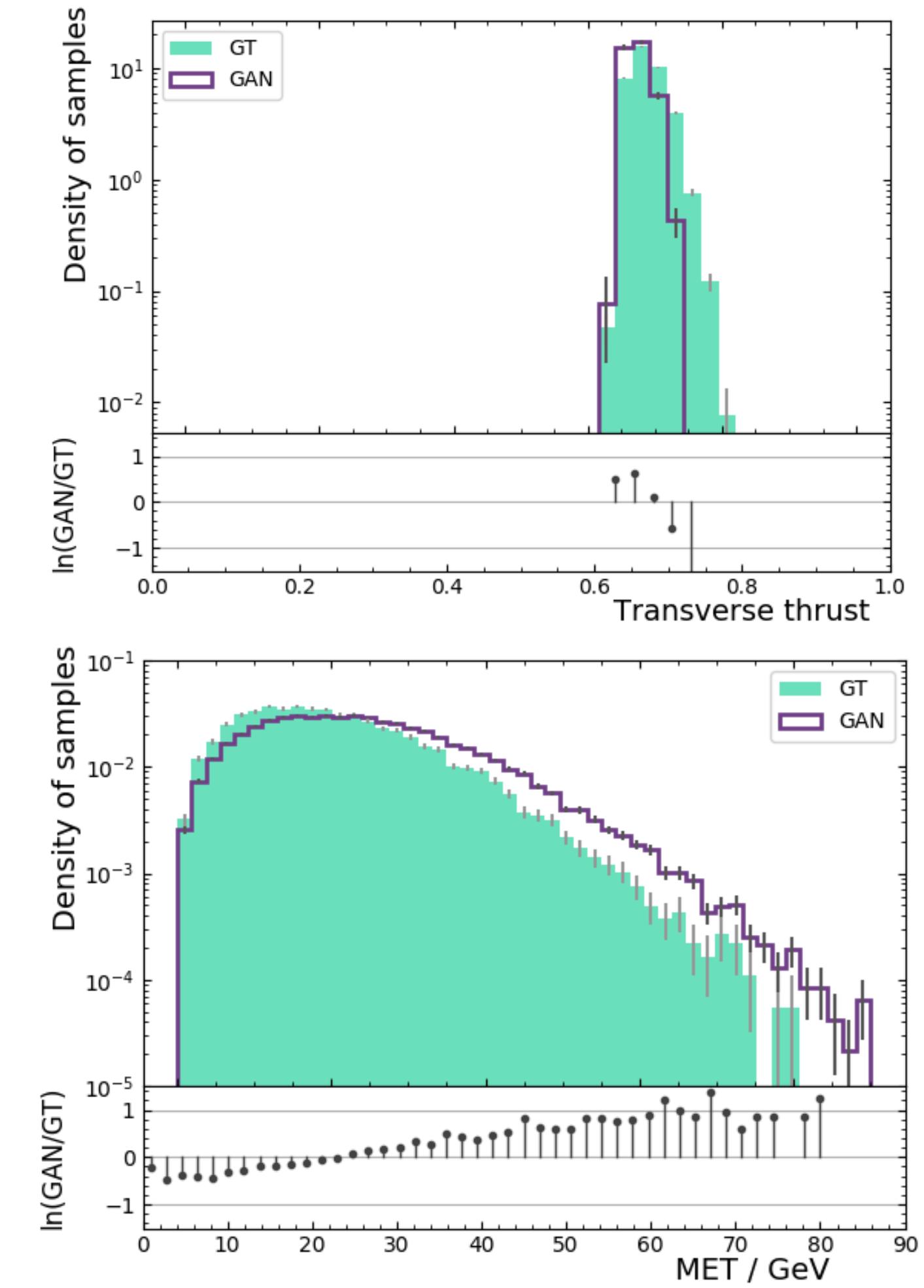
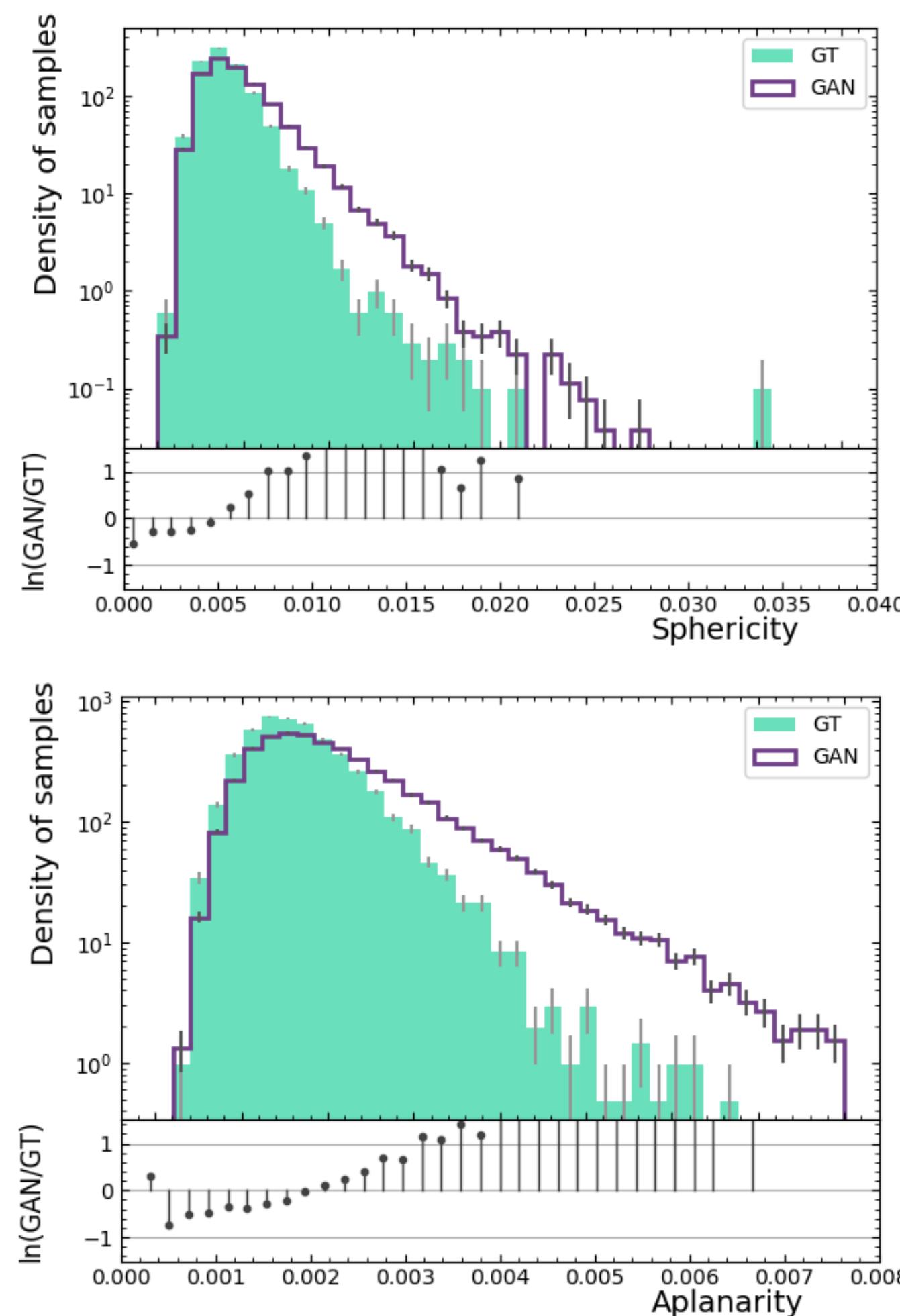
# Generating particles

● *Input features individually reproduced to an acceptable level*



# Generating particles

- But large discrepancies observed on collective quantities
- Work needed to find a more suitable architecture (graph?)



;

# Summary

---

- *Conv Networks are the most striking example of the success of Deep Learning*
- *A story of improvement from the 90s to now*
- *Very effective as a fast tool for particle reconstruction*
- *Same geometry can be used for image generation with adversarial training*
- *Can speed up one of the heaviest tasks in particle physics*