Machine Learning:

A new toolbox for Theoretical Physics

Juan Rojo

VU Amsterdam & Theory group, Nikhef

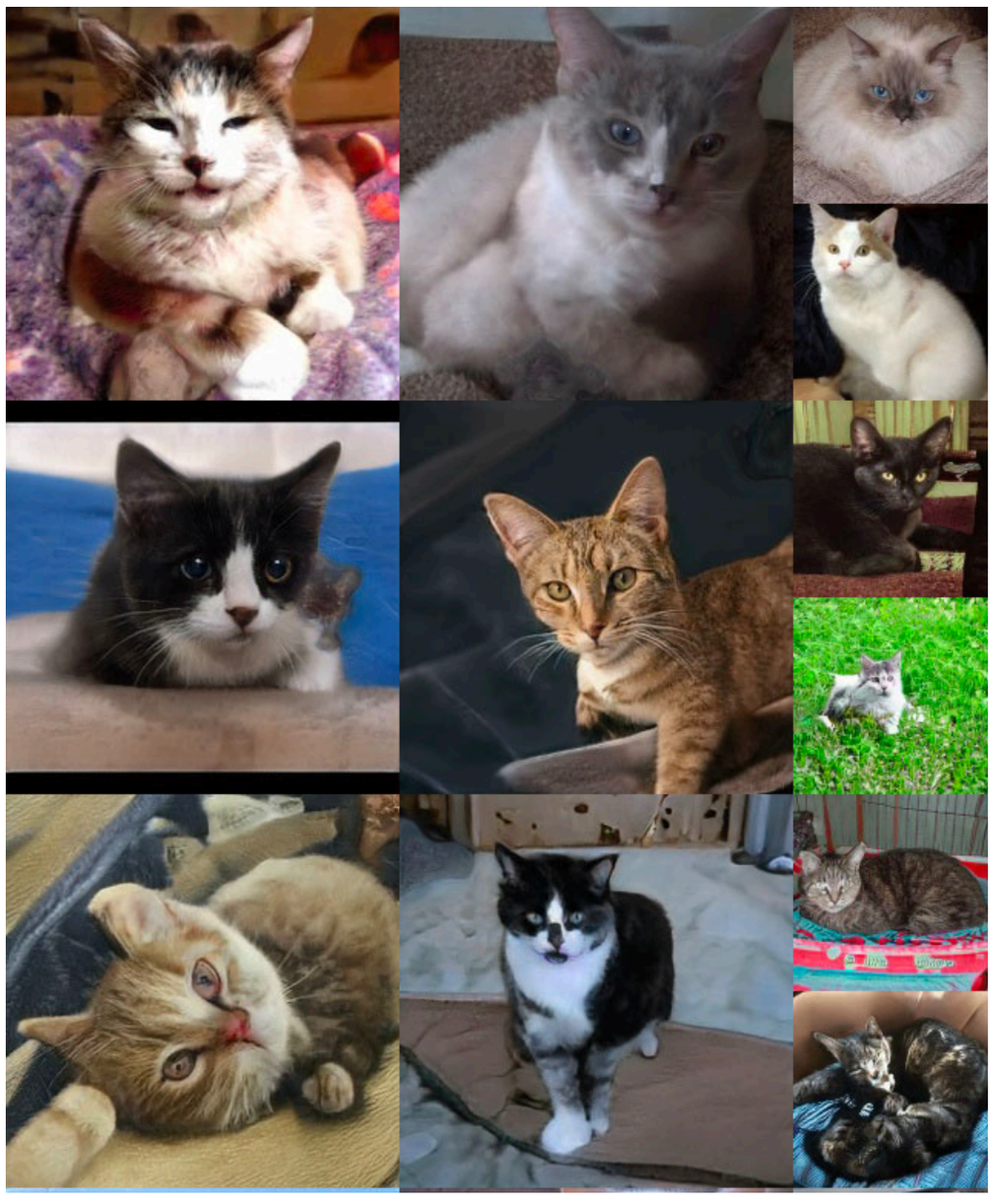
D-ITP Advanced Topics in Theoretical Physics

11/12/2019

Convolutional Neural Networks

Convolutional Neural Networks

Like physical systems, many datasets and supervised learning tasks also possess additional **symmetries and structure** what can (and should) be exploited



eg we want to train a classifier to identify pictures of cats. What **high-level features** must one learn first?

Convolutional Neural Networks

Like physical systems, many datasets and supervised learning tasks also possess additional **symmetries and structure** what can (and should) be exploited

eg we want to train a classifier to identify pictures of cats. What **high-level features** must one learn first?

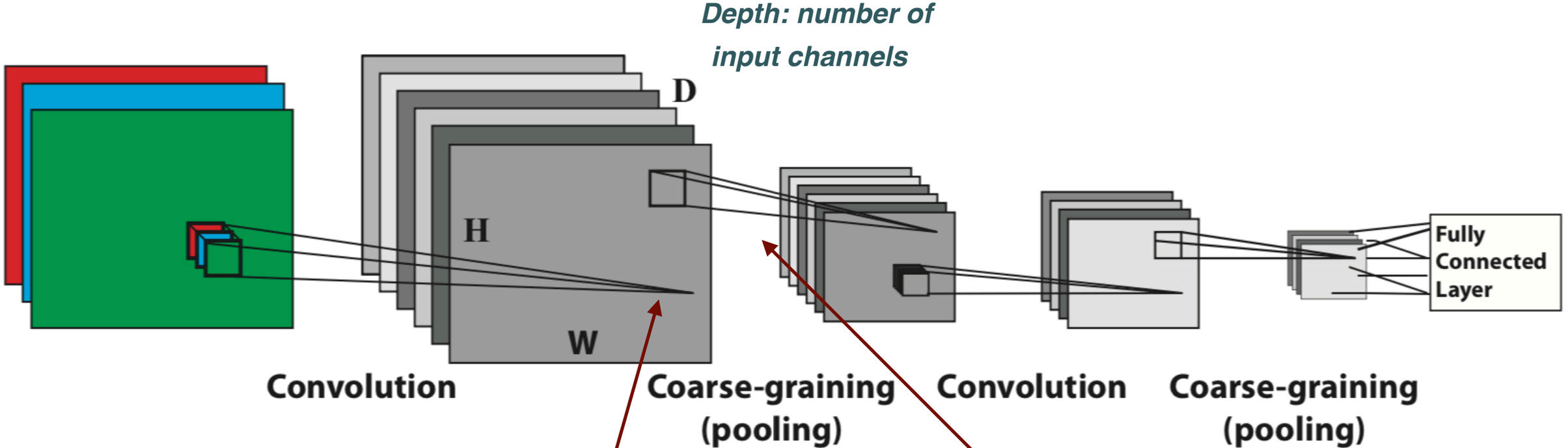
- *The features that define ``cat'' are local in the picture: whiskers, tail, paws …: **locality***
- *Cats can be anywhere in the image: **translational invariance***
- *Relative position of features must be respected (eg whiskers and tail shoaled appear in opposite sides of ``cat''): **rotational invariance***

Our classifier should exhibit all these high-level features

Convolutional Neural Networks

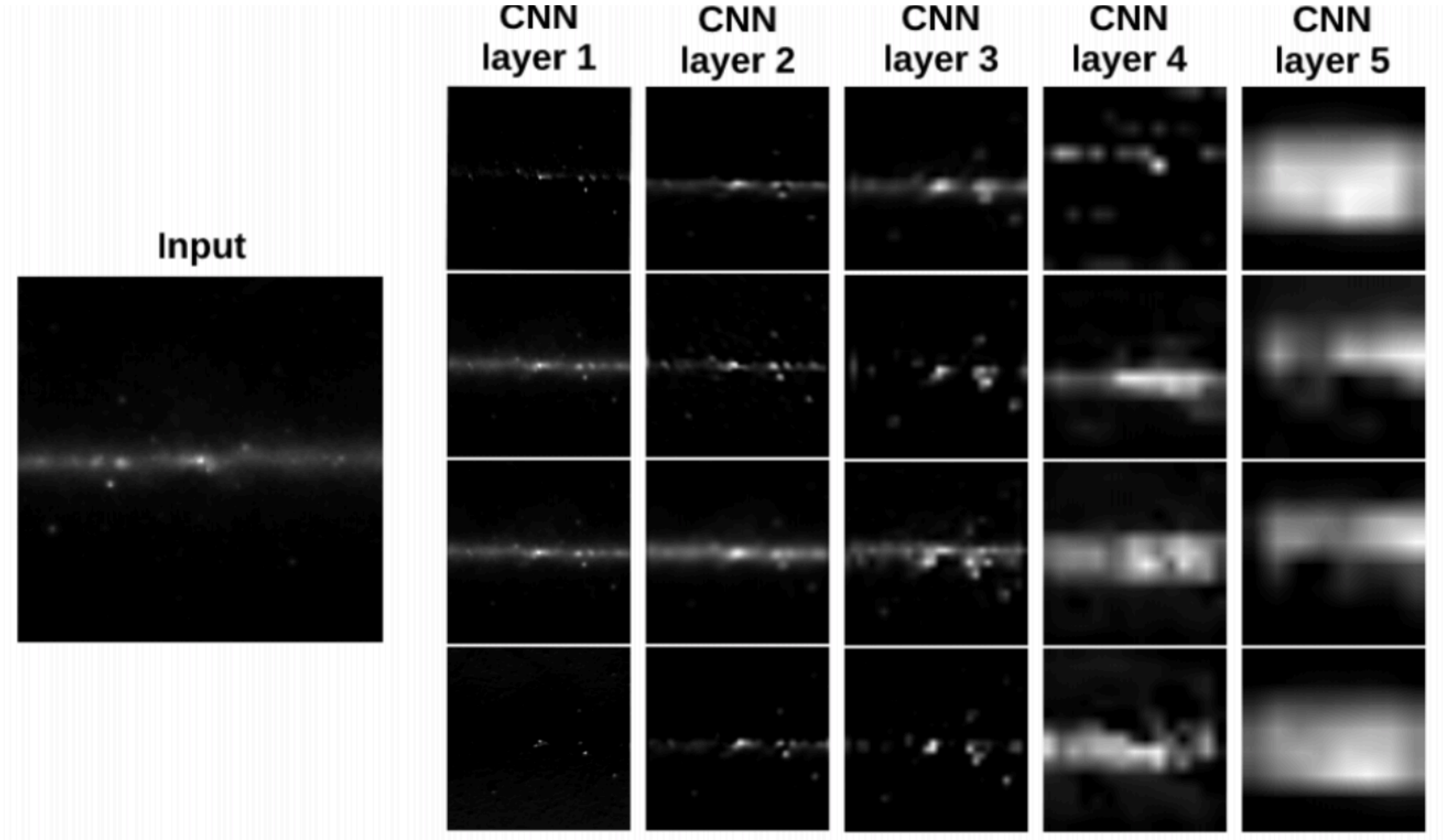
Convolutional Neural Networks (CNNs) are architectures that take **advantage of this additional high-level structures** that all-to-all coupled networks fail to exploit

A CNN is a translationally invariant neural network that respects locality of the input data



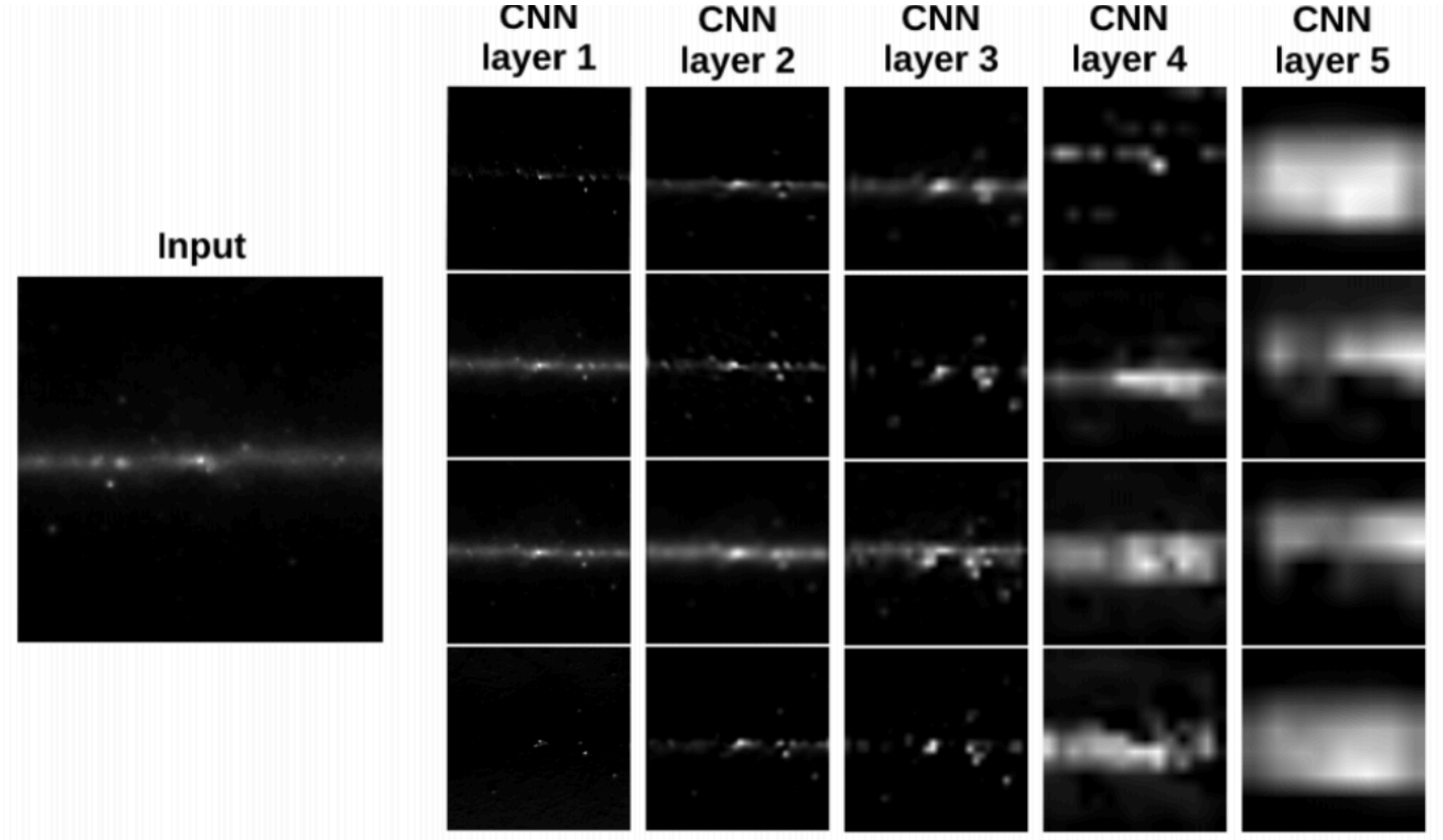
Application to Dark Matter searches

Use CNNs to discriminate between **point sources** (astrophysical origin) and **diffuse flux** (dark matter) in galactic centre images, to compare with predictions of DM models



Application to Dark Matter searches

Use CNNs to discriminate between **point sources** (astrophysical origin) and **diffuse flux** (dark matter) in galactic centre images, to compare with predictions of DM models



Energy-Based Models and Boltzmann Learning

Generative Models

Most ML models discussed here (supervised NNs, logistic regression, ensemble models) are **discriminative:** designed to identify differences between groups of data

e.g. cats vs dogs discrimination

these models cannot carry some tasks such as **drawing new examples** from an unknown probability distribution: for this we need **generative models**

e.g. learn how to draw new examples of cat and dog images

e.g. generate new samples of a given phase of the Ising model

generative models are a machine learning technique that allows to learn **how to generate new examples** similar to those found in a training dataset

GANs were a particular implementation of generative models

Here we consider **energy-based generative models**: close connection with statistical physics

Maximum Entropy Generative Models

basic concept is the **Shannon information-theoretic entropy**, which quantifies the statistical uncertainty one has about a random variable drawn from a probability distribution

$$S_p = \mathrm{Tr}_{\boldsymbol{x}} p(\boldsymbol{x}) \log \boldsymbol{x}$$

sum/integral over all possible values of variable

assume we have a set of models, functions of ***x,*** whose average should coincide with some observed values. What should be their **underlying prob dist**?

$$\{f_i(\boldsymbol{x})\} \qquad \langle f_i \rangle_{\mathrm{obs}}$$

models ***observations***

Principle of Maximum Entropy: choose the probability distribution with the largest uncertainty (Shannon entropy) subject to the observational constraints

$$\langle f_i \rangle_{\mathrm{model}} = \int d\mathbf{x} f_i(\mathbf{x}) p(\mathbf{x}) = \langle f_i \rangle_{\mathrm{obs}}$$

Maximum Entropy Generative Models

this condition can be expressed as a **Lagrange Multiplier problem** by minimising:

$$\mathscr{L}[p] = -S_p + \sum_i \lambda_i \left(\langle f_i \rangle_{\text{obs}} - \int d\mathbf{x} f_i(\mathbf{x}) p(\mathbf{x}) \right) + \gamma \left(1 - \int d\mathbf{x} p(\mathbf{x}) \right)$$

entropy ***conservation constraints*** ***normalisation***

whose solution gives us the **maximum entropy distribution**

$$p(\mathbf{x}) = \frac{\exp\left(\sum_i \lambda_i f_i(\mathbf{x}) \right)}{\int d\mathbf{x} \, \exp\left(\sum_i \lambda_i f_i(\mathbf{x}) \right)} = \frac{\exp\left(\sum_i \lambda_i f_i(\mathbf{x}) \right)}{Z}$$

$$E = -\sum_i \lambda_i f_i(\mathbf{x})$$

which is nothing but the **Boltzmann distribution in statistical mechanics,** and where the parameters of the distribution are fixed by the observations

$$\partial_{\lambda_i} \log Z = \langle f_i \rangle_{\text{data}}$$

Energy-based Generative Models

these MaxEnt models can be used to **infer the underlying probability distributions** from a finite set of observations, which subsequently can be used to generate new instances

training an energy-based generative model: using the data to infer the model parameters

$$E\left(\mathbf{x}, \boldsymbol{\theta}\right)$$

as in Supervised Learning, we need to specify a **cost function,** which however in the case of generative models is much subtler: what defines a good model?

the most useful method is to **maximise the log-likelihood** of the training set

$$\hat{\boldsymbol{\theta}} = \arg\,\min_{\theta} \mathscr{L}(\{\boldsymbol{\theta}\})$$

$$\mathscr{L}(\{\boldsymbol{\theta}\}) = \left\langle \log p_{\theta}(\boldsymbol{x}) \right\rangle_{\text{data}} = -\left\langle E(\boldsymbol{x};\boldsymbol{\theta}) \right\rangle_{\text{data}} - \log Z(\{\boldsymbol{\theta}\})$$

where we have used that the **generative probability distribution** is of the Boltzmann form and that the partition function does not depend on the data

Boltzmann machines

The training of **energy-based generative models** proceeds usually via SGD

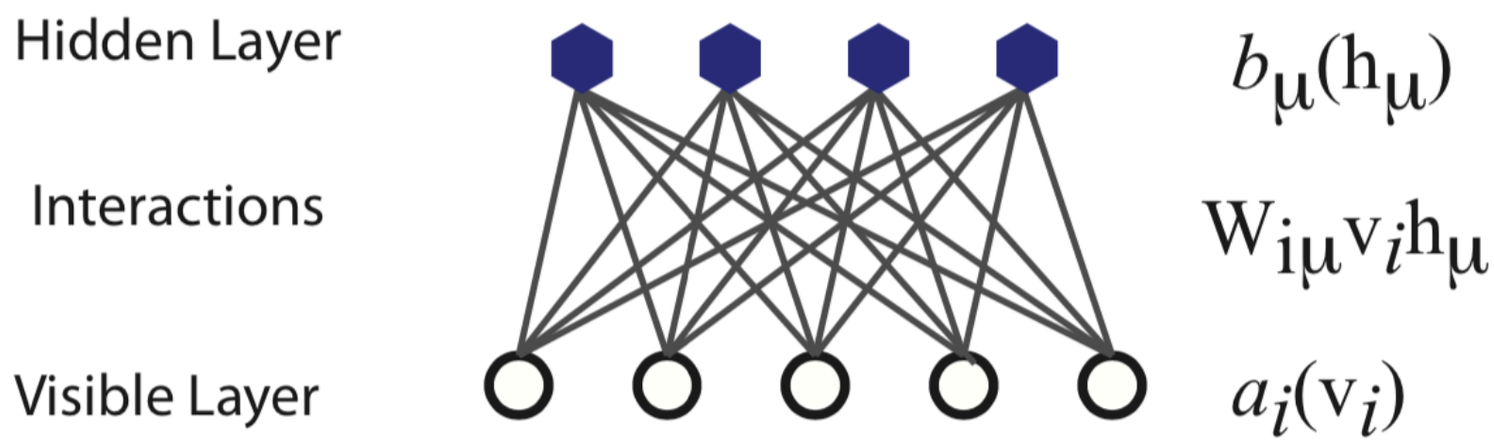
MaxEnt generative models are defined by the choice of the **energy**

$$p(\mathbf{x}) = \frac{\exp\left(\sum_i \lambda_i f_i(\mathbf{x})\right)}{\int d\mathbf{x}\, \exp\left(\sum_i \lambda_i f_i(\mathbf{x})\right)} = \frac{\exp\left(\sum_i \lambda_i f_i(\mathbf{x})\right)}{Z} = \frac{\exp\left(-E(\mathbf{x};\boldsymbol{\lambda})\right)}{Z}$$

one can construct various **other generative models** with different choices of the energy

e.g. Restricted Boltzmann machines

$$E(\mathbf{x};\boldsymbol{\theta}) = -\sum_i a_i(v_i) - \sum_\mu b_\mu(h_\mu) - \sum_{i\mu} W_{i\mu} v_i h_\mu$$



Generative Models & Adversarial Learning

The Kullback-Leibler divergence

The KL divergence, a measure of the similarity between two probability distributions *p(**x**)* and *q(**x**)* plays an important role in generative models

$$D_{KL}\left(p\,||\,q\right) = \int d\boldsymbol{x}\, p(\boldsymbol{x}) \log \frac{p(\boldsymbol{x})}{q(\boldsymbol{x})} \qquad D_{KL}\left(q\,||\,p\right) = \int d\boldsymbol{x}\, q(\boldsymbol{x}) \log \frac{q(\boldsymbol{x})}{p(\boldsymbol{x})}$$

which can be symmetrised to construct a **squared metric** (distance)

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

Jensen-Shannon divergence

The KL-divergence is **positive-definite**, and only vanishes when *p(**x**)=* q*(**x**)*

$$D_{KL}\left(p\,||\,q\right) \geq 0$$

The Kullback-Leibler divergence

In **generative models** such as GANs one deals with two probability distributions, which we would like to have as similar as possible

$$p_\theta(\boldsymbol{x}) \qquad p_{\text{data}}(\boldsymbol{x})$$

from model *from training data*

however subtleties about how we define **similarity** have large implications for the model training

maximising the **log-likelihood of the data under the model** is the same as **minimising the KL divergence** between the data distribution and the model distribution

$$D_{KL}\left(p_{\text{data}}||p_\theta\right) = \int d\boldsymbol{x}\, p_{\text{data}}(\boldsymbol{x}) \log \frac{p_{\text{data}}(\boldsymbol{x})}{p_\theta(\boldsymbol{x})}$$

$$= \int d\boldsymbol{x}\, p_{\text{data}}(\boldsymbol{x}) \log p_{\text{data}}(\boldsymbol{x}) - \int d\boldsymbol{x}\, p_{\text{data}}(\boldsymbol{x}) \log p_\theta(\boldsymbol{x})$$

$$= S[p_{\text{data}}] - \left\langle \log p_\theta \right\rangle_{\text{data}}$$

The Kullback-Leibler divergence

maximising the **log-likelihood of the data under the model** is the same as **minimising the KL divergence** between the data distribution and the model distribution

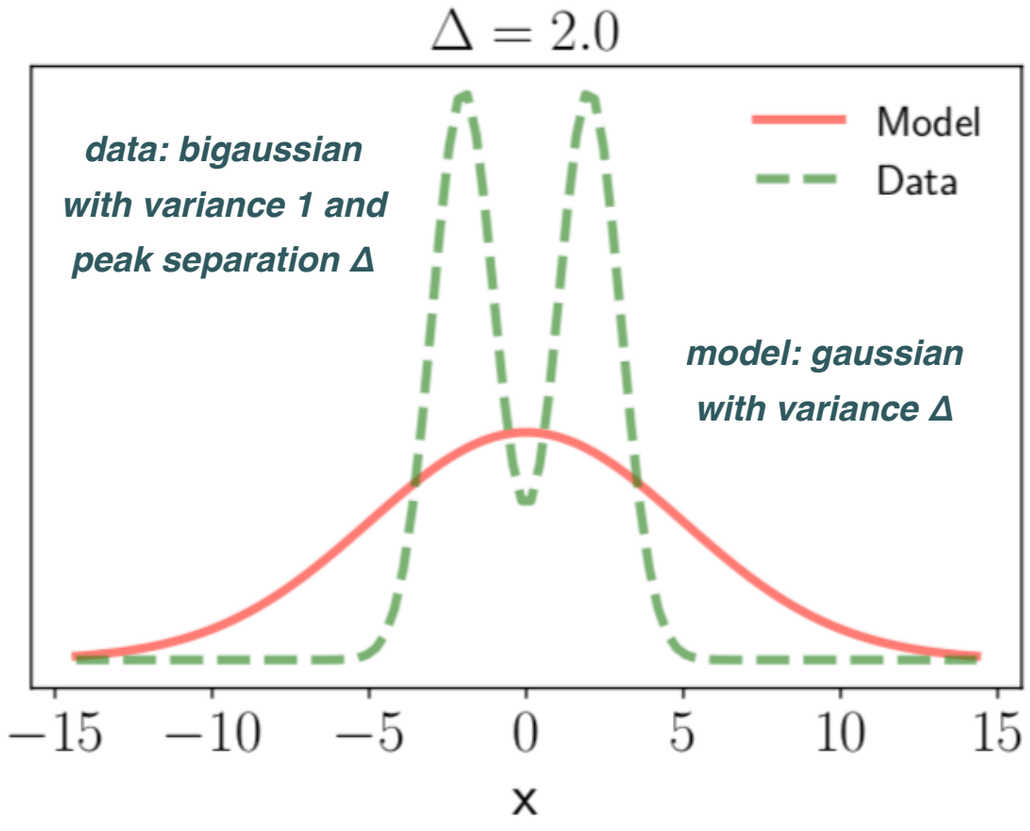
$$\left\langle \log p_\theta(\boldsymbol{x}) \right\rangle_{\text{data}} = S[p_{\text{data}}] - D_{KL}\left(p_{\text{data}} \,||\, p_\theta\right)$$

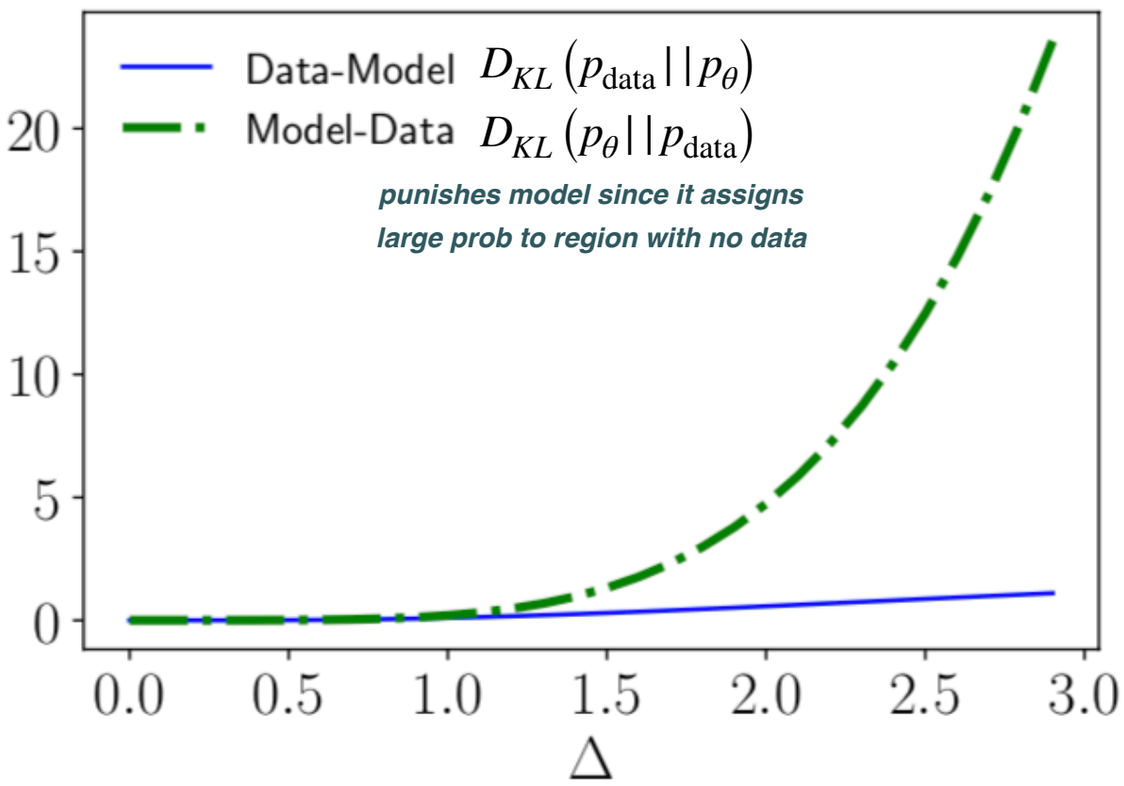
Log-likelihood of data under model

entropy of data: independent of model parameters

KL-divergence

similarity is a subtle concept ….





Adversarial Learning

(1) $D_{KL}\left(p_{\text{data}}||p_\theta\right)$ → ***Calculable using sampling***

(2) $D_{KL}\left(p_\theta||p_{\text{data}}\right)$ → ***Large when model over-weights low-density regions near real peaks***

→ ***but not calculable since p_{data} unknown ….***

In **Adversarial Learning** we achieve a similar goal as that of minimising ***(2)*** by training a **discriminator** to distinguish between real data points and samples from the model

By punishing the model for generating points that can be easily discriminated from the data, Adversarial Learning decreases the **weight of regions in the model space that are far away from data points,** regions that inevitably arise when maximizing the likelihood

Generative adversarial networks

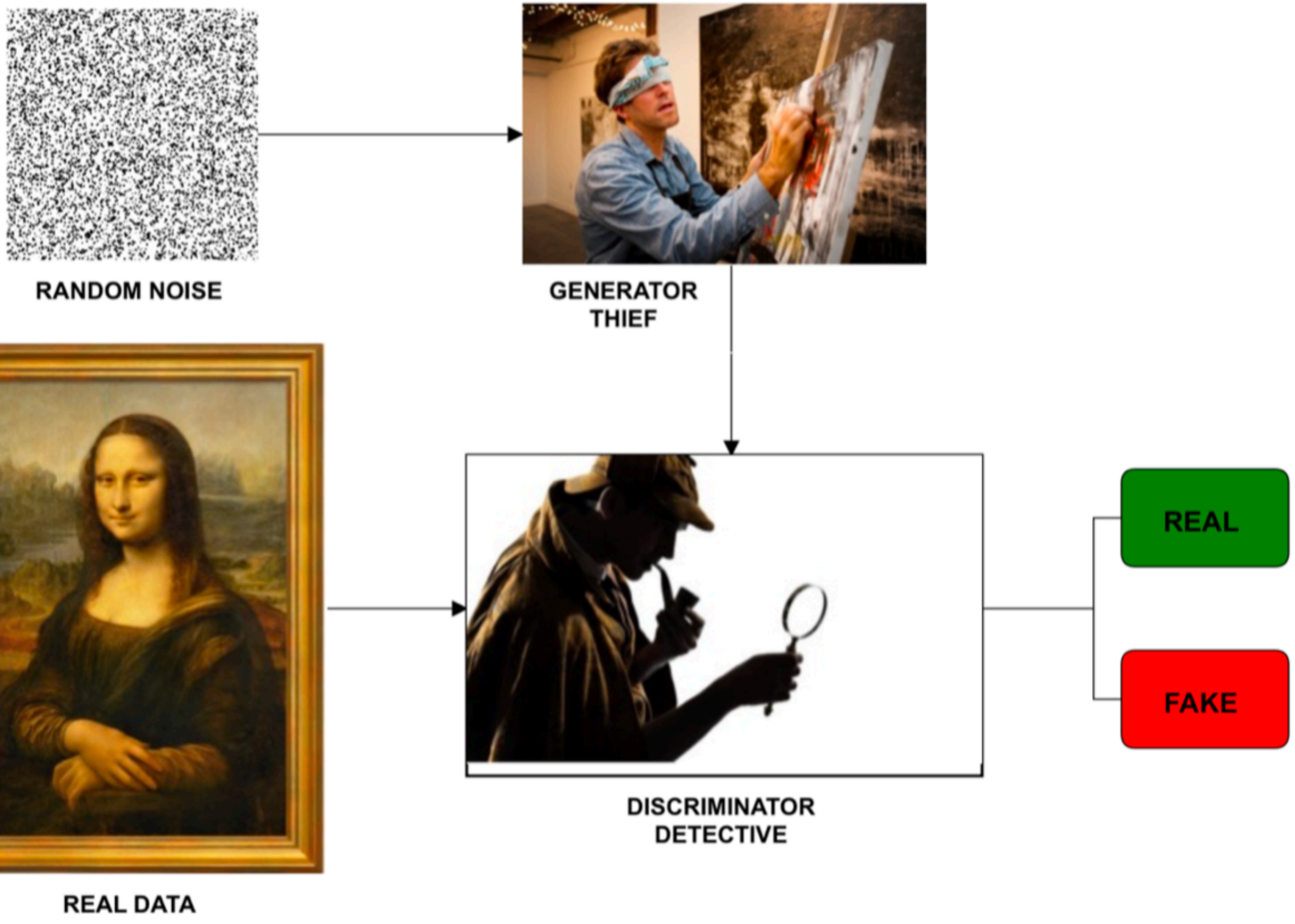
Generative Adversarial Networks (GANs) are deep neural networks architectures, composed by two independent NNs which **compete against each other**

- ***(1)*** A **generator G** NN that creates pseudo-data by inferring the probability distribution associated to the training dataset

- ***(2)*** A **discriminator D** NN which determines the probability of a given sample arises from the actual training data rather than having been produced by **G**

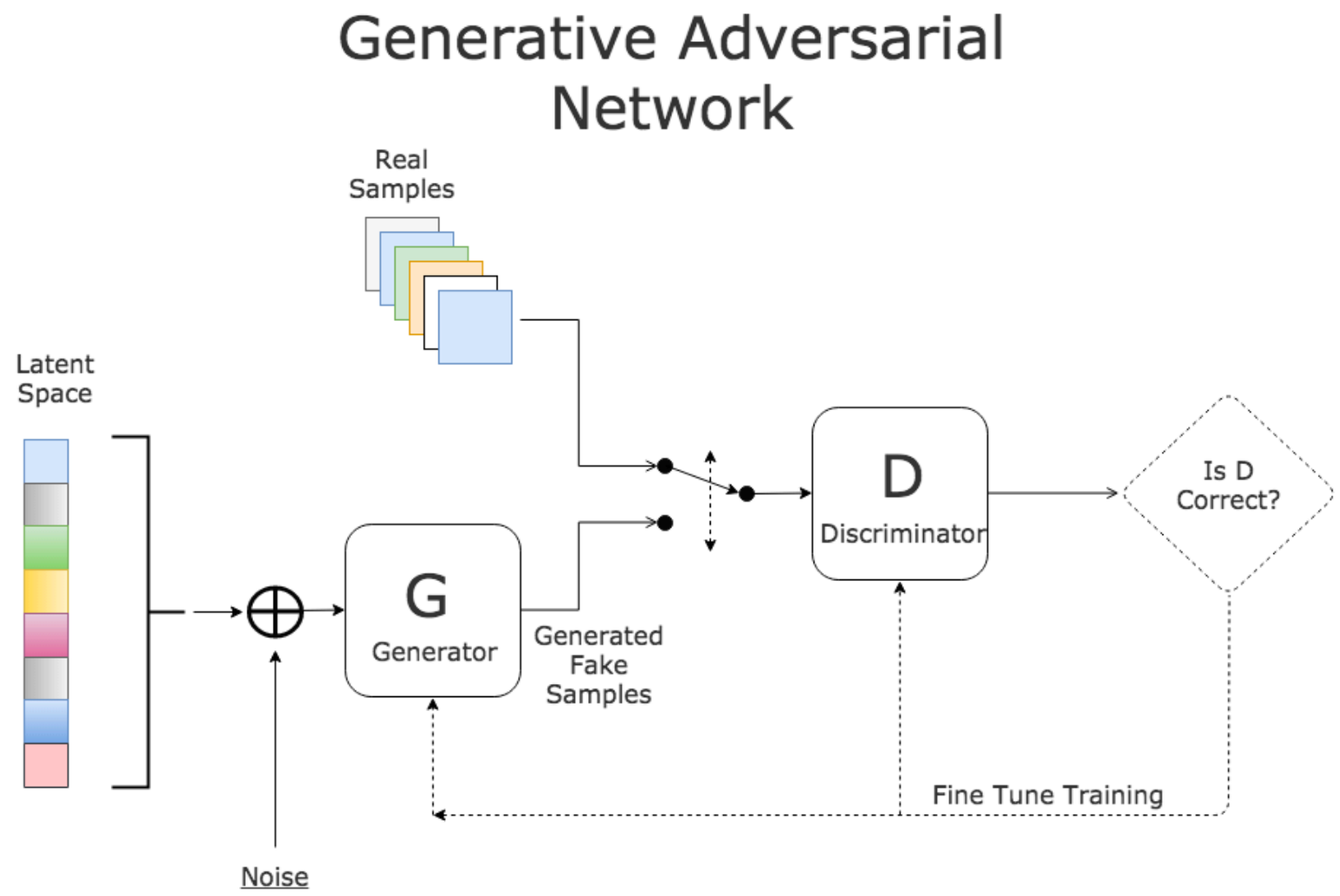
The generator network **G** should be trained to maximise the probability that the discriminator network **D** makes a mistake: that is, **G** should generate pseudo-data samples that are virtually indistinguishable from the actual data

Generative adversarial networks



Generative Adversarial Networks

- New architecture for an **unsupervised neural network training** (unlabelled samples)
- Based on two **independent nets** that work separately and act as **adversaries:**
  - the **Discriminator (D)** undergoes training and plays the role of classifier
  - the **Generator (G)** and is tasked to generate random samples that **resemble real samples** with a twist rendering them as fake samples.



GAN training

As with other NN architectures one uses GD to train GANs, but now one has to update sequentially the model parameters of both **G** and **D**

- Take a sample of *N* data points from the training set

$$\{\boldsymbol{x}_n\}_{n=1}^{N} \qquad \boldsymbol{x}_n = (x_{n,1}, x_{n,2}, \ldots, x_{n,p})$$

p = number of features per sample

- Produce a sample of *N* pseudo-data points from generator **G** (at ite0 this is random noise)

$$\{\boldsymbol{z}_n\}_{n=1}^{N} \qquad \boldsymbol{z}_n = (z_{n,1}, z_{n,2}, \ldots, z_{n,p})$$

- Evaluate the cost function: since we are dealing with binary classification (true/false) the appropriate cost function is the **cross-entropy**

$$C\left(\boldsymbol{\theta}_D, \boldsymbol{\theta}_G\right) = \frac{1}{N}\sum_{n=1}^{N}\left(\log D(\boldsymbol{x}_i) + \log(1 - D(G(\boldsymbol{z}_i)))\right)$$

NN params of D ***NN params of G*** ***output of D when input a real data sample*** ***output of D when input a ``fake'' data sample produced by G***

- Train **D** using GD to maximise its discrimination capability

GAN training

- Evaluate the cost function: since we are dealing with binary classification (true/false) the appropriate cost function is the **cross-entropy**

$$C\left(\boldsymbol{\theta}_D, \boldsymbol{\theta}_G\right) = \frac{1}{N}\sum_{n=1}^{N}\left(\log D(\boldsymbol{x}_i) + \log(1 - D(G(\boldsymbol{z}_i)))\right)$$

- Train **D** using GD to maximise its discrimination capability

$$\mathbf{v}_t = \eta_t \nabla_{\theta_D} C(\boldsymbol{\theta}_{D,t}, \boldsymbol{\theta}_G)\,, \qquad \boldsymbol{\theta}_{D,t+1} = \boldsymbol{\theta}_{D,} - \mathbf{v}_t$$

- At this point **D** can tell apart data from pseudo-data pretty well, so we need to train **G** to generate better (closer to the training set) pseudo-data samples
- Produce a sample of *N* pseudo-data points from the generator **G** $\{\boldsymbol{z}_n\}_{n=1}^{N}$

$$C\left(\boldsymbol{\theta}_D, \boldsymbol{\theta}_G\right) = \frac{1}{N}\sum_{n=1}^{N}\log(1 - D(G(\boldsymbol{z}_i)))$$

$$\mathbf{v}_t = \eta_t \nabla_{\theta_G} C(\boldsymbol{\theta}_{D,t}, \boldsymbol{\theta}_G)\,, \qquad \boldsymbol{\theta}_{G,t+1} = \boldsymbol{\theta}_{G,t} - \mathbf{v}_t$$

GAN training

The generator and discriminator are sequentially trained and iterated until convergence is achieved, at this point **D** cannot tell apart the pseudo-data from **G** from the real data

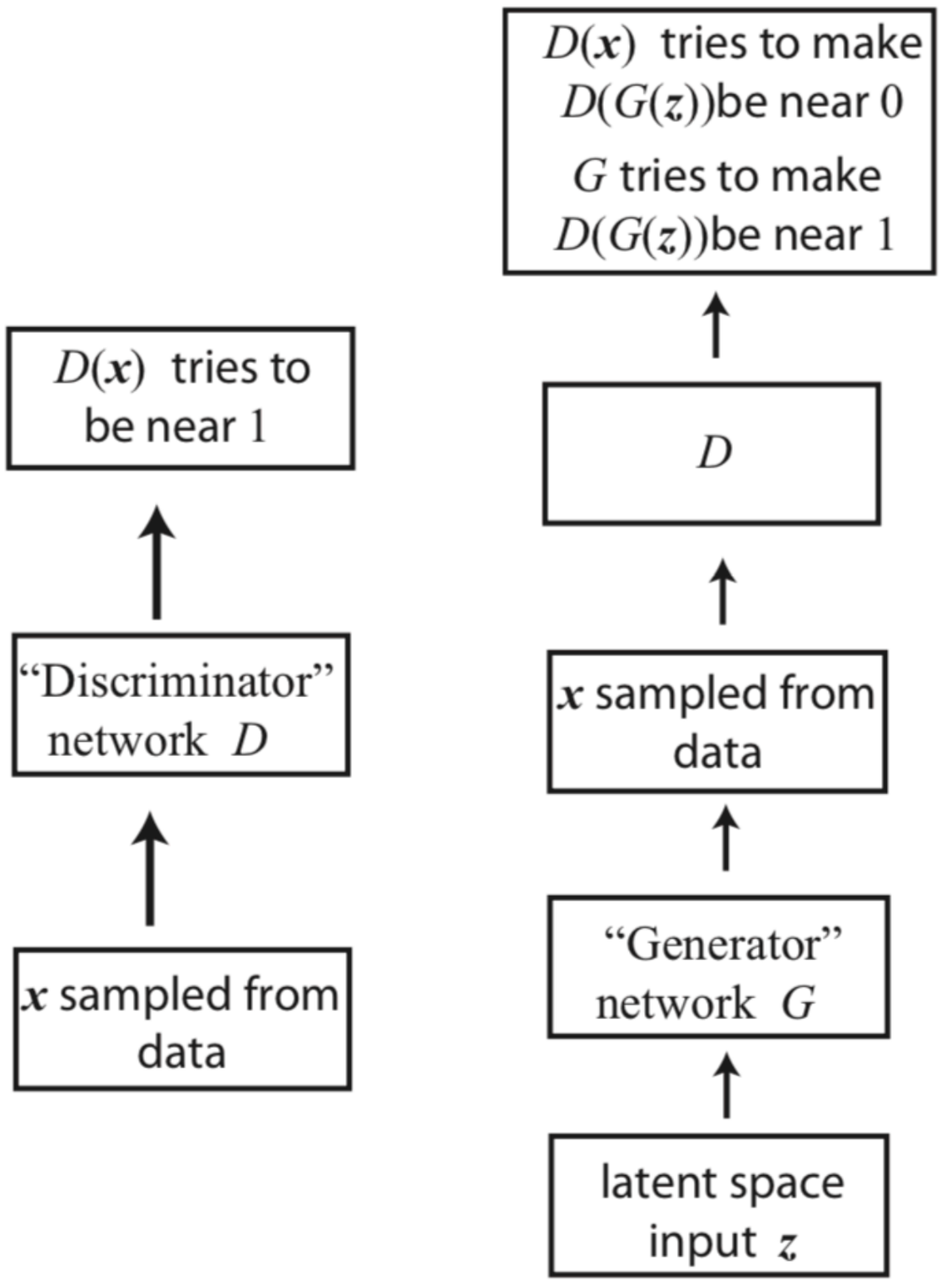
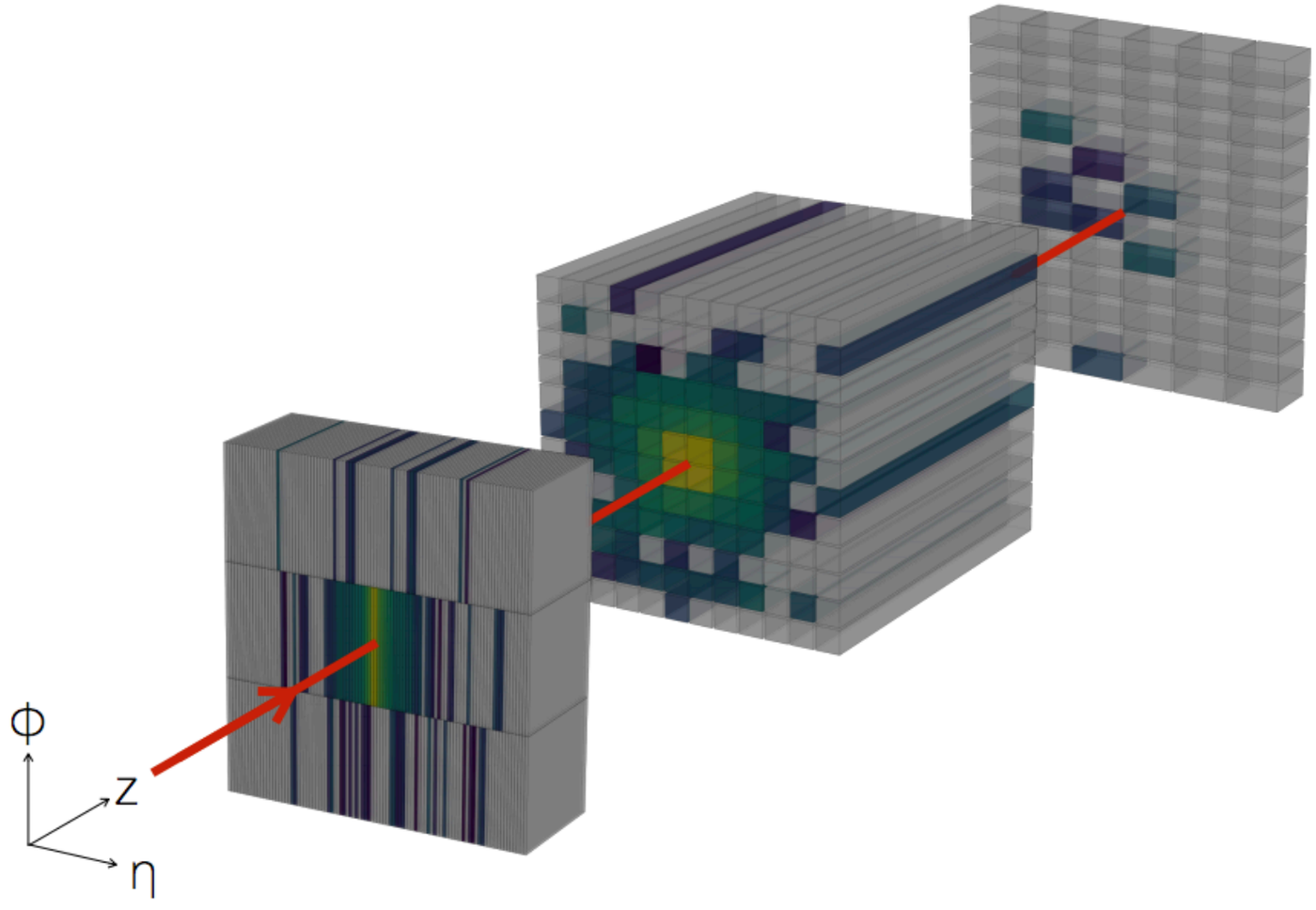


Image generation with GANs

https://thispersondoesnotexist.com/

GANs for detector simulation in HEP

- Modelling accurately the response of detectors with the **propagation of high energy particles** is an essential task for present and future HEP experiment
- **Detector simulation at the LHC** is a very CPU-intensive task, dominated by modelling of particle showers inside calorimeters
- **Generative Adversarial Networks** can speed up detector simulation by orders of magnitude



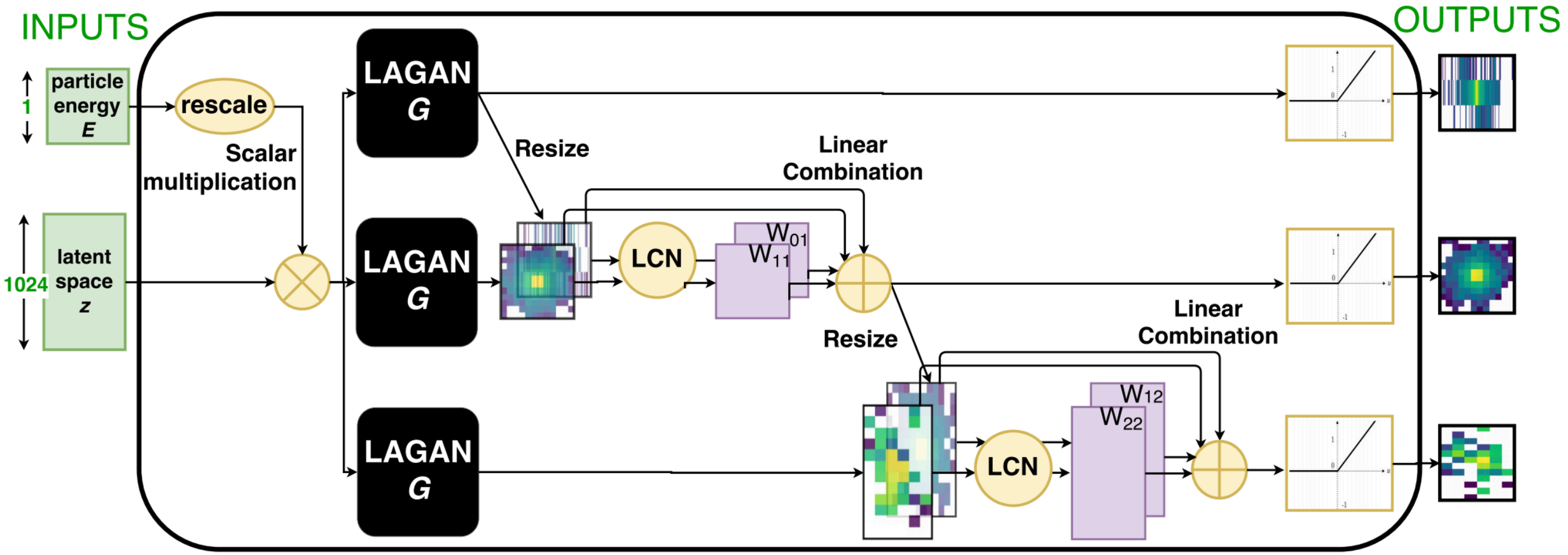
Task: to efficiently model the ***propagation of high energy particles*** *(and their interaction) within the layers of electromagnetic and hadronic calorimeters*

GANs for detector simulation in HEP

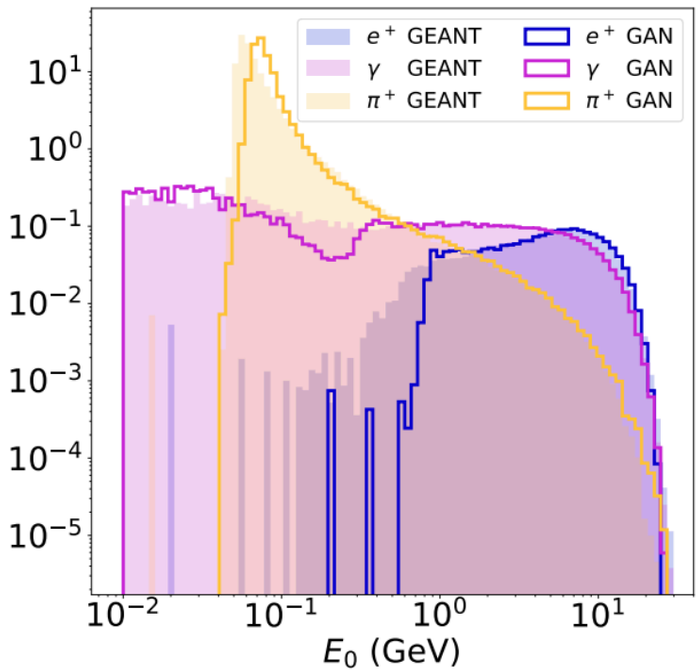
- Use GANs as a tool to **speed up full simulation of particle showers** in a HEP calorimeter
- The generator G learns a map from **a latent space** to space of **generated samples** for training
- Carefully understanding the **underlying physics of particle propagation** in a detector is crucial to optimise the training strategy, *e.g.* relationships between neighbouring detector layers

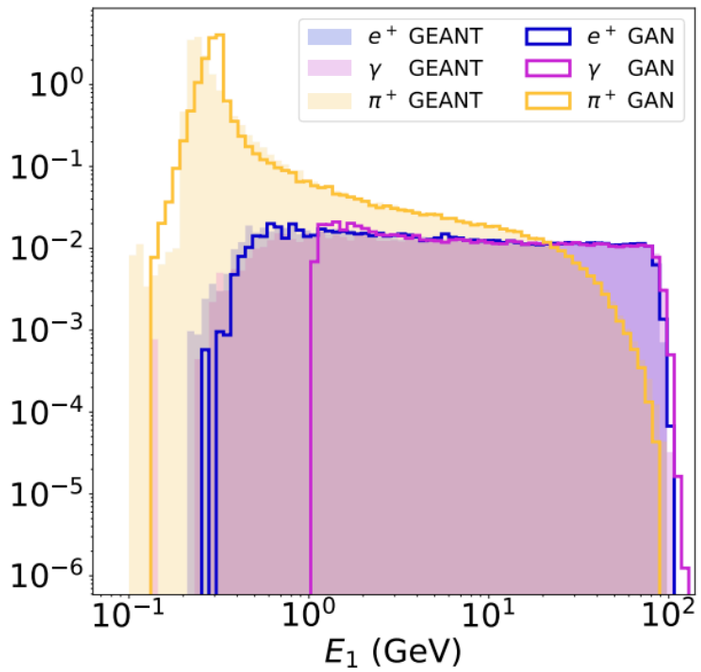
Generator Network

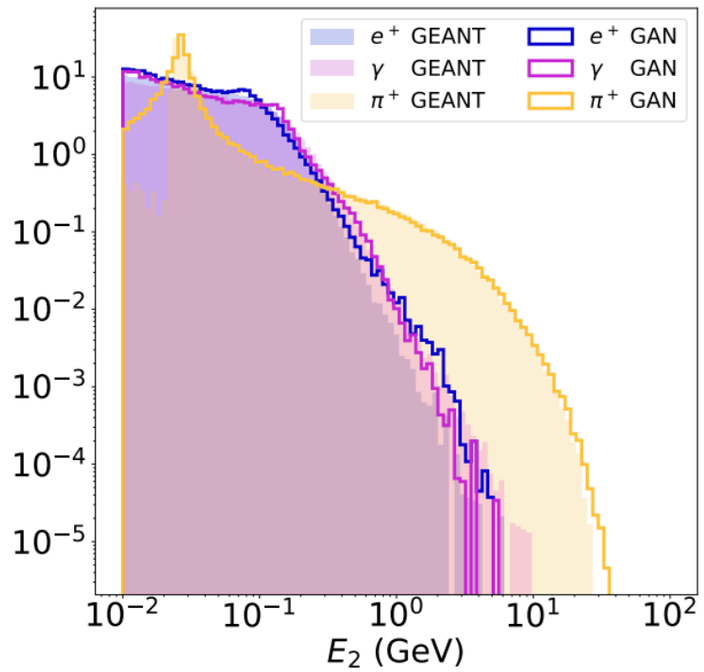
Output: ***generated samples*** *to be used for training D*

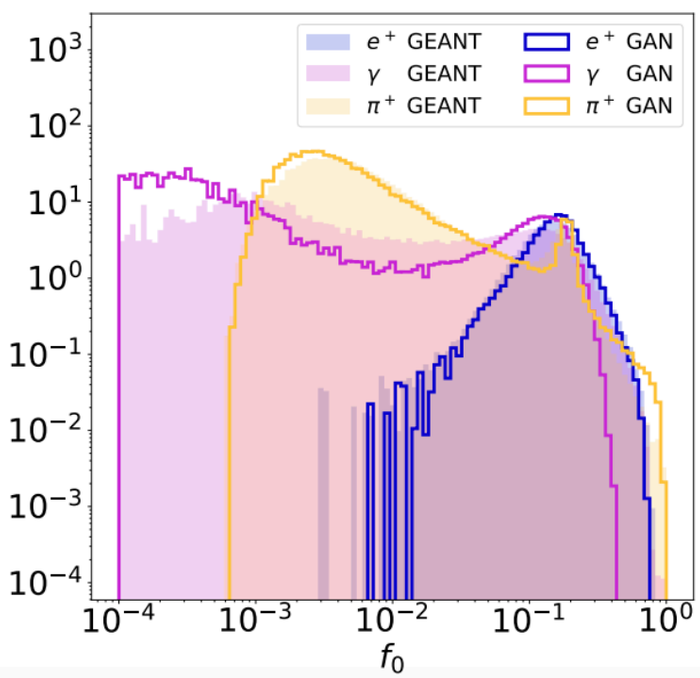


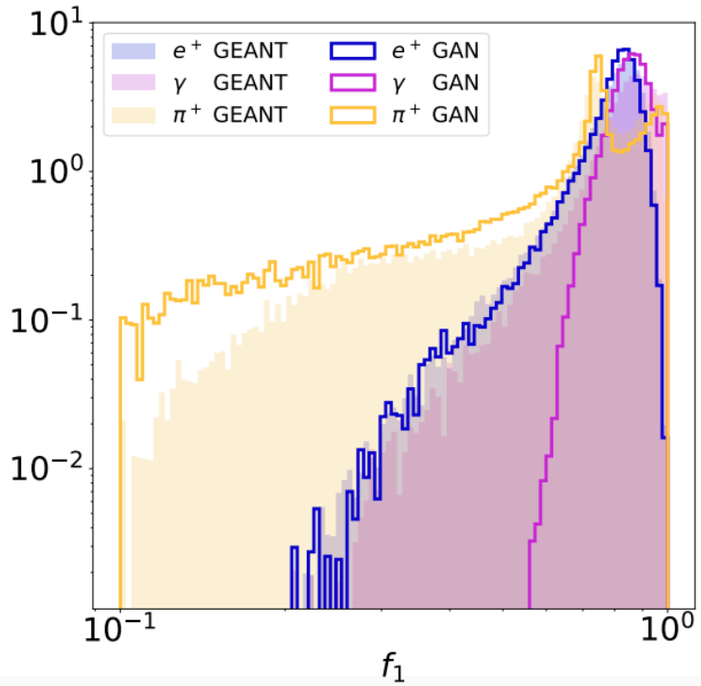
GANs for detector simulation in HEP

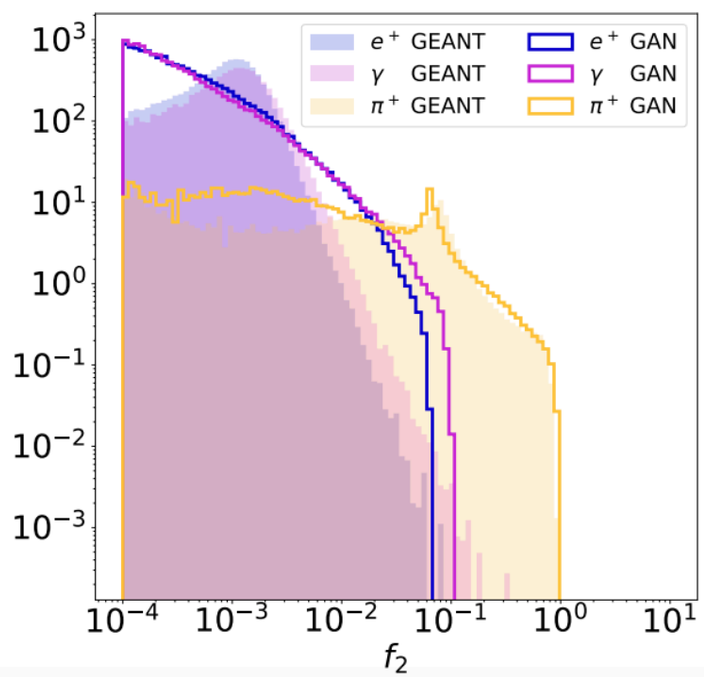












GANs for detector simulation in HEP

Simulator	Hardware	Batch Size	ms/shower
GEANT4	CPU	N/A	1772
CALOGAN	CPU	1	13.1
		10	5.11
		128	2.19
		1024	2.03
	GPU	1	14.5
		4	3.68
		128	0.021
		512	0.014
		1024	0.012

Speed-up by several orders of magnitude, specially when running in GPUs

Machine Learning and Quantum Computation