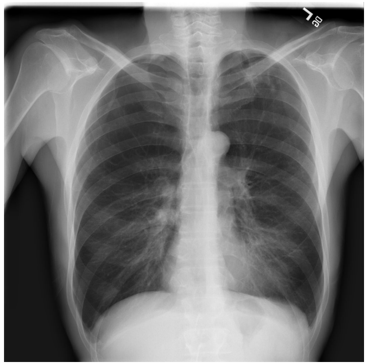
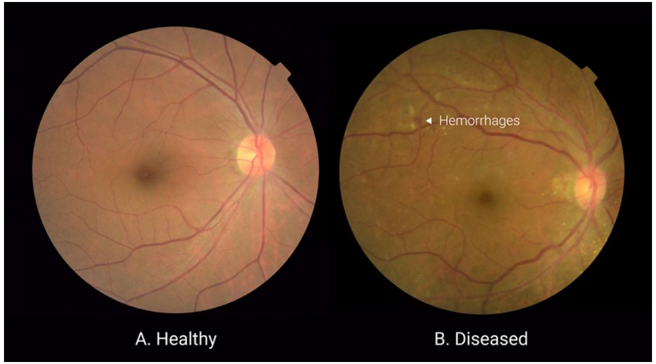
Deep Learning: A Technical Overview

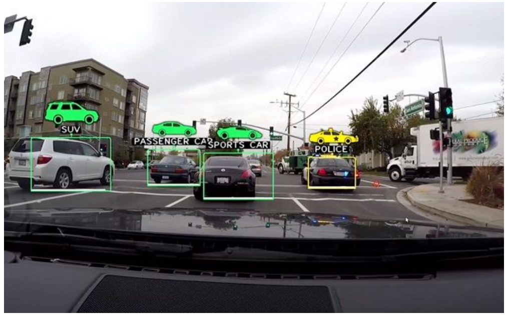
Mustafa Mustafa
NERSC
@mustafa240m

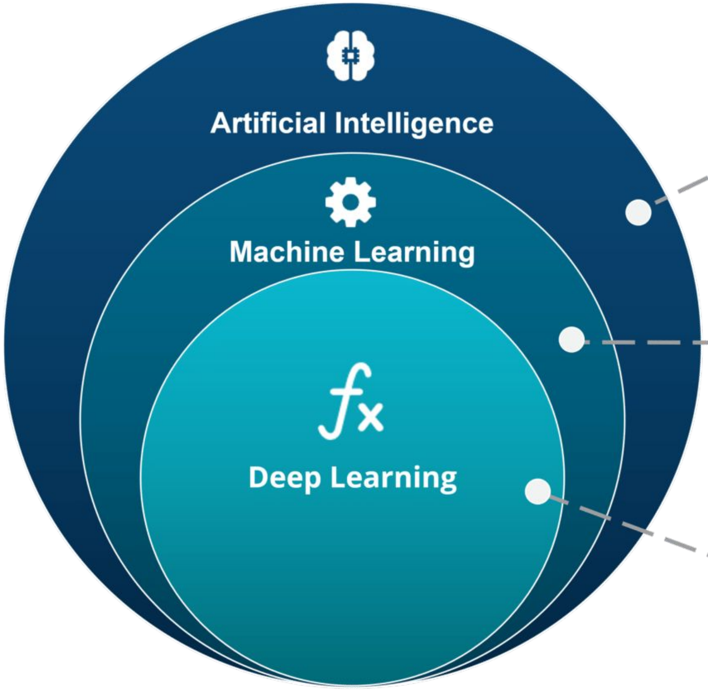
GPUs for Science Day
July 2019, Berkeley Lab

Deep Learning is powering many recent technologies









ARTIFICIAL INTELLIGENCE

A technique which enables machines to mimic human behaviour

MACHINE LEARNING

Subset of AI technique which use statistical methods to enable machines to improve with experience

DEEP LEARNING

Subset of ML which make the computation of multi-layer neural network feasible

Diagram source https://www.edureka.co/blog/ai-vs-machine-learning-vs-deep-learning/

What can Deep Learning do?

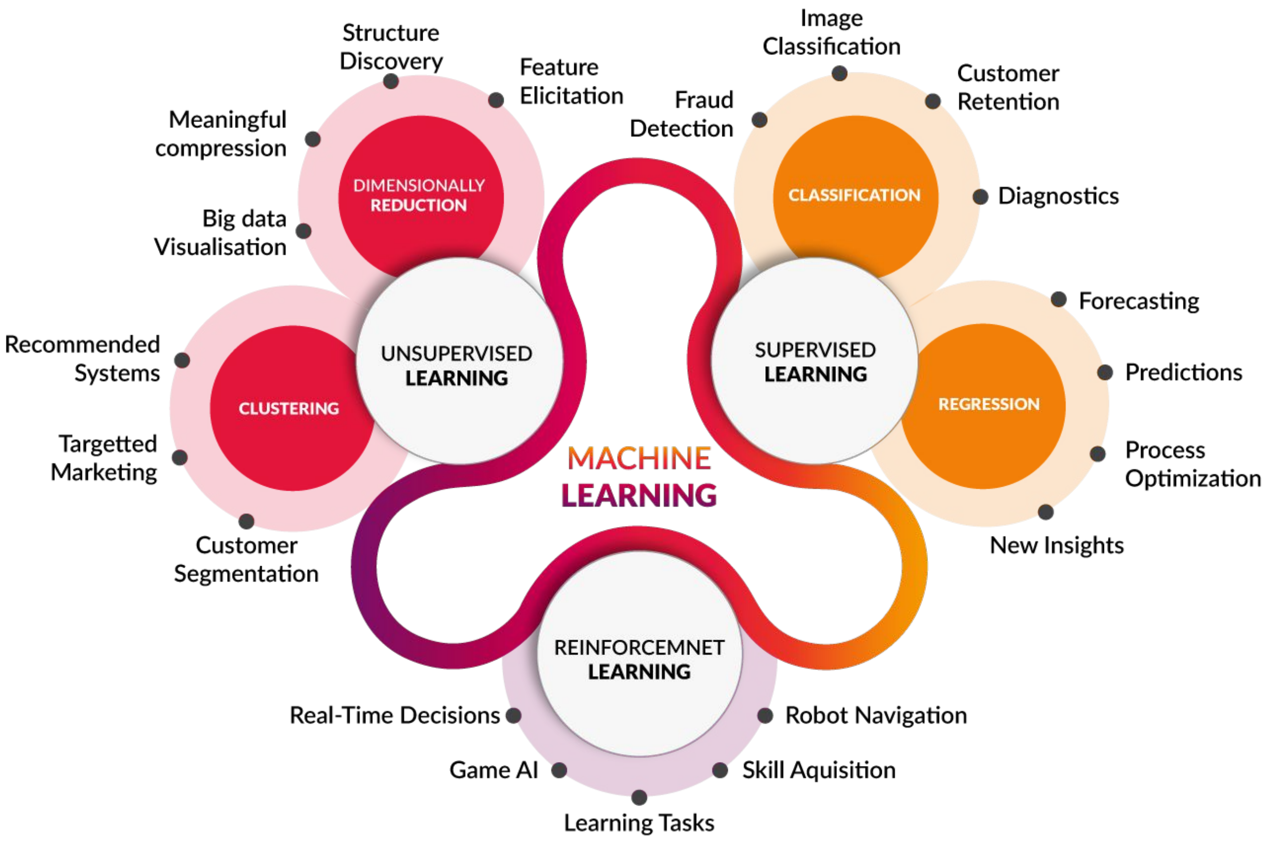


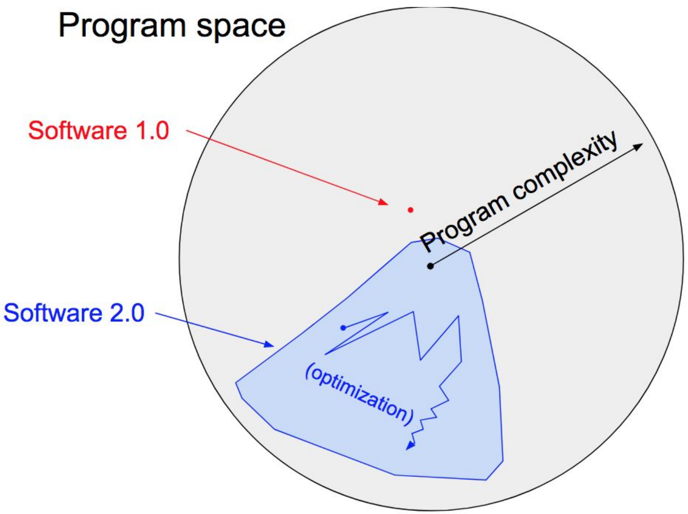
Diagram source: http://www.cognub.com/index.php/cognitive-platform/

Deep Learning is a new programming paradigm, Software 2.0

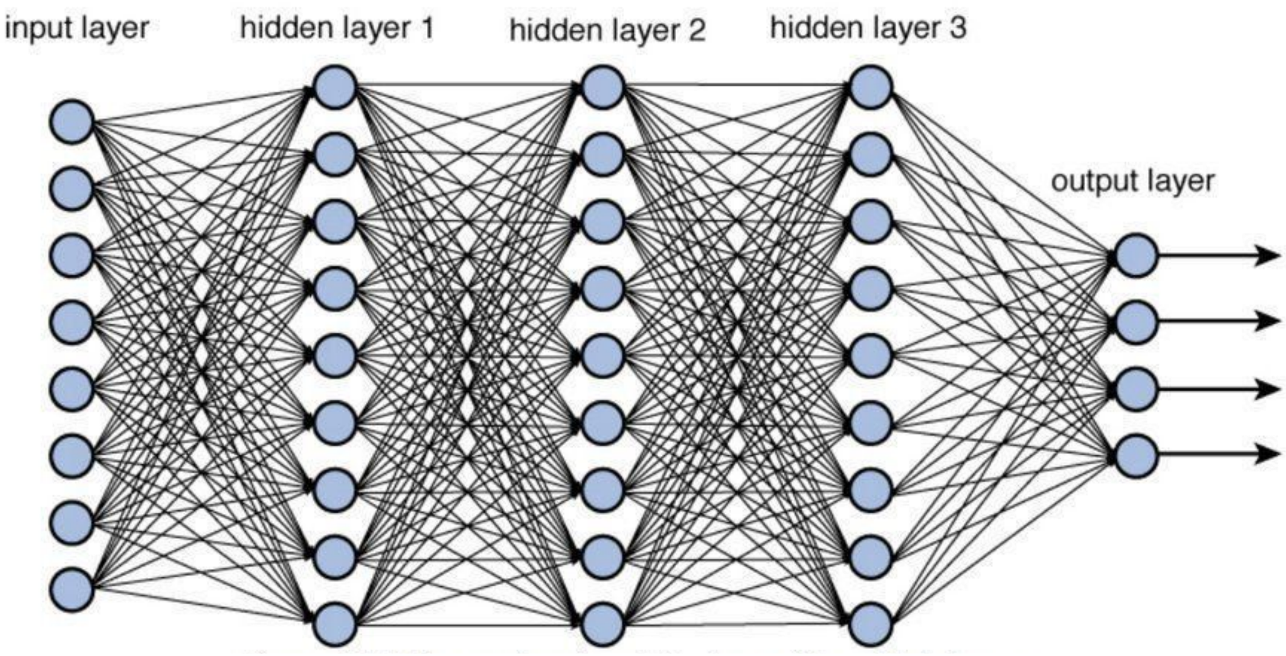
"It turns out that a large portion of real-world problems have the property that **it is significantly easier to collect the data (or more generally, identify a desirable behavior) than to explicitly write the program**."

-- Andrew Karpath,

https://medium.com/@karpathy/software-2-0-a64152b37c35

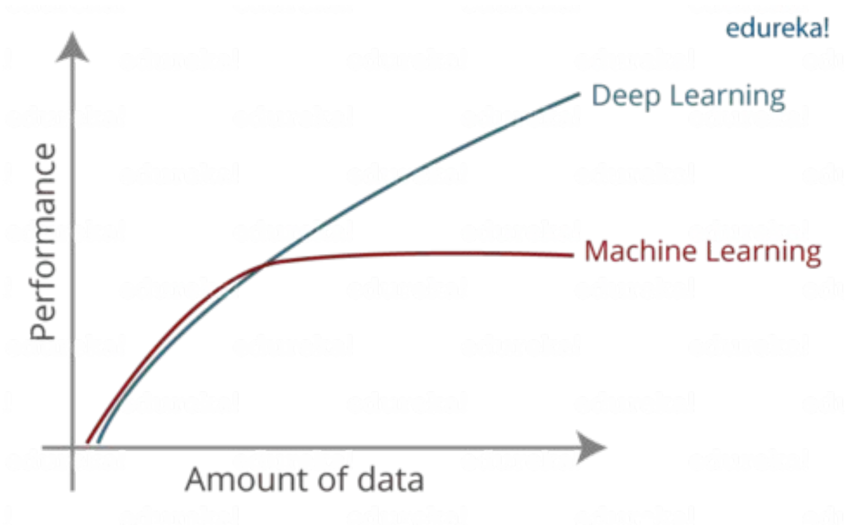


Deep Learning is powered by Deep Neural Networks

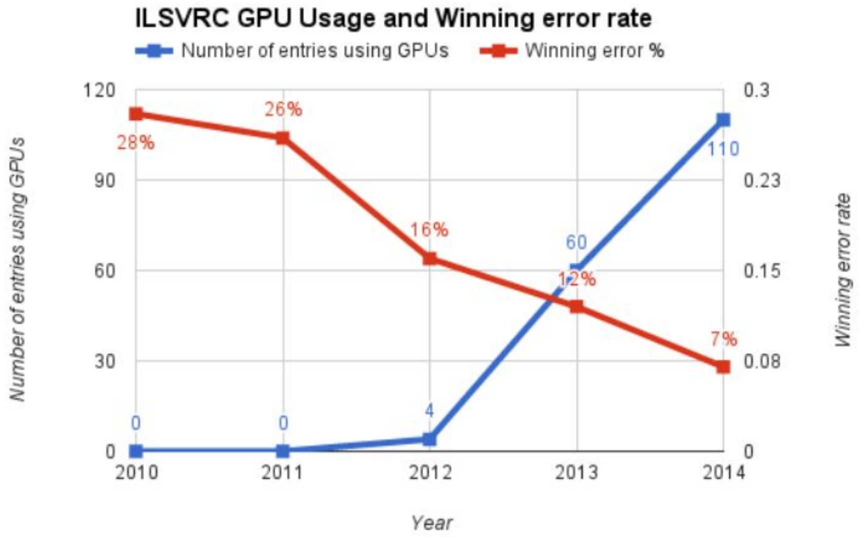


Why do Neural Networks finally work now?

1) Data: large curated datasets



2) GPUs: linear algebra accelerators



3) Algorithmic advances: optimizers, regularization, normalization … etc.

What are Deep Neural Networks?

Long story short:

"A family of **parametric**, **non-linear** and **hierarchical** **representation learning functions**, which are massively optimized with stochastic gradient descent to **encode domain knowledge**, i.e. domain invariances, stationarity." -- Efstratios Gavves

Deep Forward Neural Networks (DNNs)

The objective of NNs is to approximate a function:

$$y = f^*(x)$$

The NN learns an approximate function $y = f(x; W)$ with parameters W. This approximator is hierarchically composed of simpler functions

$$y = f^n(f^{n-1}(\cdots f^2(f^1(x))\cdots))$$

Deep Forward Neural Networks (DNNs)

The objective of NNs is to approximate a function:

$$y = f^*(x)$$

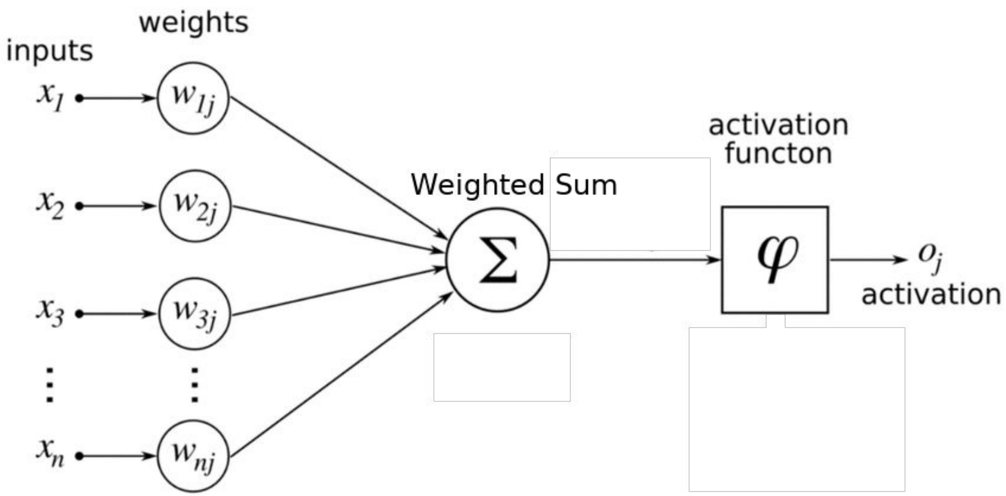
The NN learns an approximate function $y = f(x; W)$ with parameters W. This approximator is hierarchically composed of simpler functions

$$y = f^n(f^{n-1}(\cdots f^2(f^1(x))\cdots))$$

A common choice for the atomic functions is an affine transformation followed by a non-linearity (an activation function $\varphi(x)$):

$$h_1 = \varphi(W_1 x + b_1)$$
$$h_2 = \varphi(W_2 h_1 + b_2)$$
$$\vdots$$
$$y = f(h_n)$$

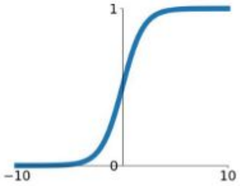
Activation functions



Activation functions

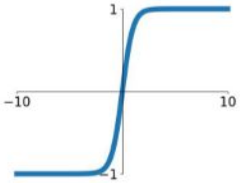
Sigmoid

$\sigma(x) = \frac{1}{1+e^{-x}}$



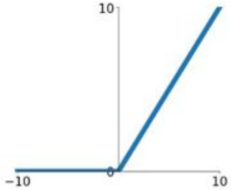
tanh

$\tanh(x)$



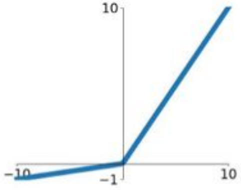
ReLU

$\max(0, x)$



Leaky ReLU

$\max(0.1x, x)$

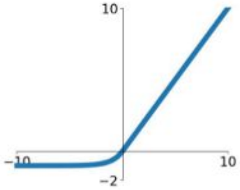


Maxout

$\max(w_1^T x + b_1, w_2^T x + b_2)$

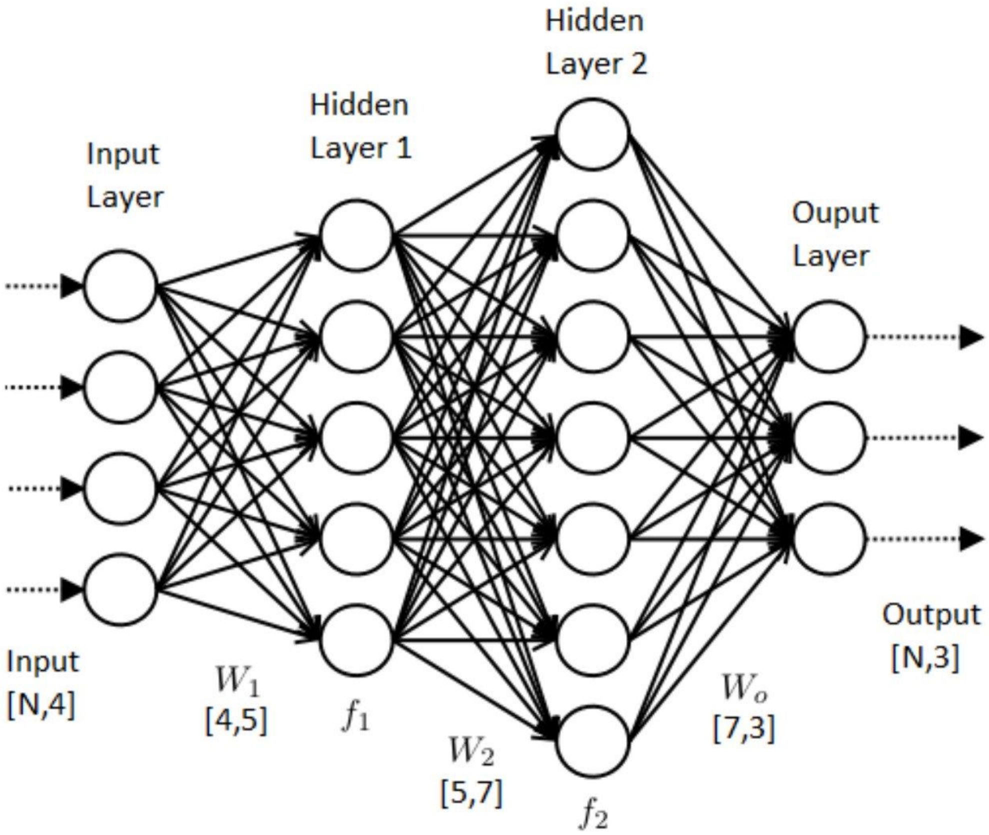
ELU

$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$



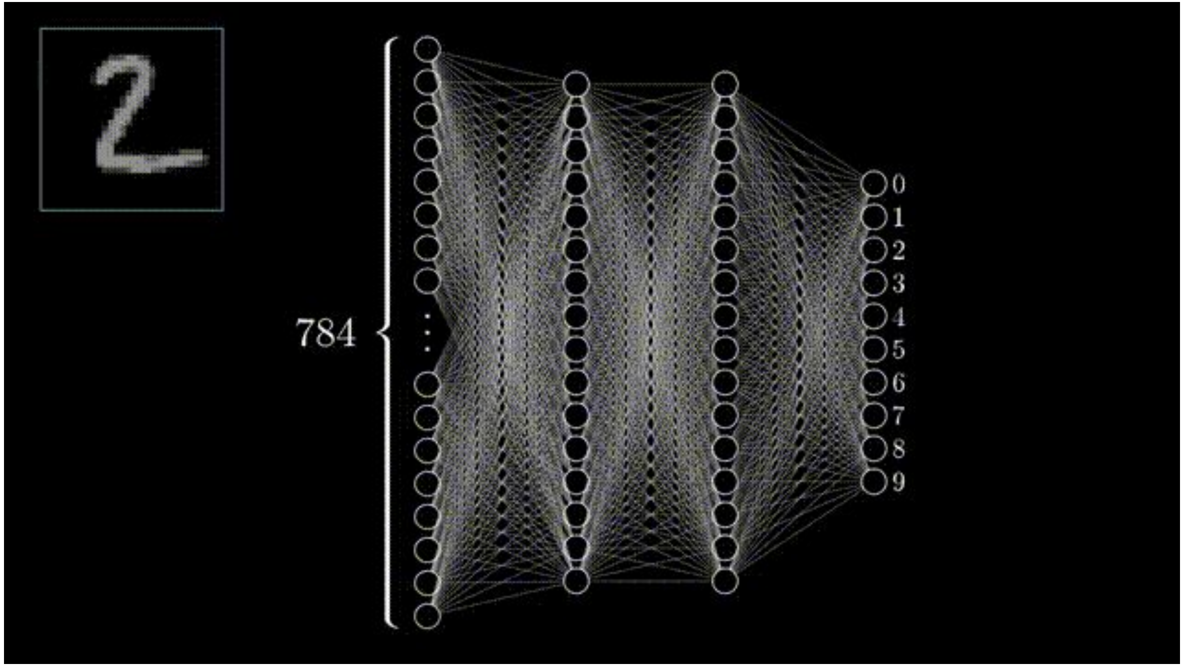
Deep Forward Neural Networks (DNNs)

$$h_1 = \varphi(W_1 x + b_1)$$
$$h_2 = \varphi(W_2 h_1 + b_2)$$
$$\vdots$$
$$y = f(h_n)$$



Deep Forward Neural Networks (DNNs)

$$h_1 = \varphi(W_1 x + b_1)$$
$$h_2 = \varphi(W_2 h_1 + b_2)$$
$$\vdots$$
$$y = f(h_n)$$



Animation adapted from: youtube.com/watch?v=aircAruvnKk&feature

Cost function & Loss

To optimize the network parameters for the task at hand we build a cost function on the training dataset:

$$J(W) = \mathbb{E}_{x,y \sim \hat{p}_{data}} L(f(x;W), y)$$

Cost function & Loss

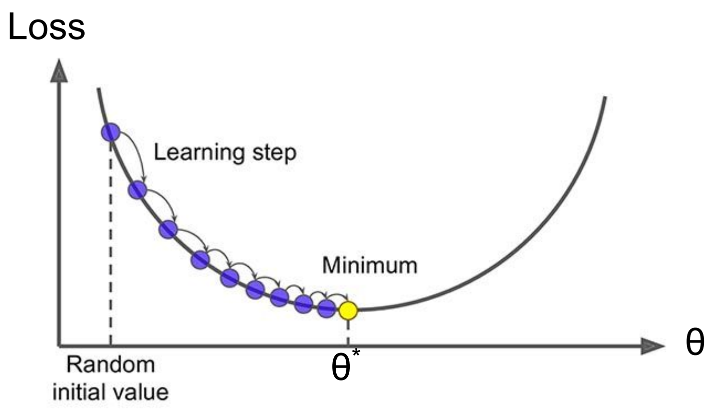
To optimize the network parameters for the task at hand we build a cost function on the training dataset:

$$J(W) = \mathbb{E}_{x,y \sim \hat{p}_{data}} L(f(x;W), y)$$

Most NNs are trained using a maximum likelihood (i.e. find the parameters that maximize the probability of the training dataset):

$$J(W) = -\mathbb{E}_{x,y \sim \hat{p}_{data}} \log p_{model}(y|x)$$

Gradient Descent



Gradient descent is the dominant method to optimize networks parameters θ to minimize loss function L(θ).

The update rule is (**α** is the “learning rate”):

$$W_{i+1} \leftarrow W_i - \alpha \nabla L(W)$$

The gradient is typically averaged over a minibatch of examples in minibatch **stochastic gradient descent.**

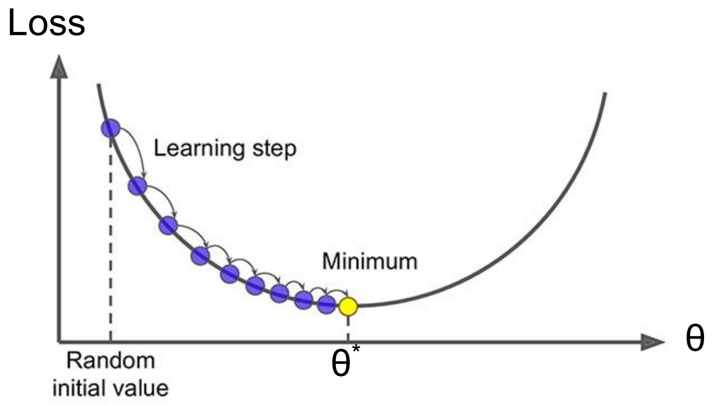
Gradient Descent

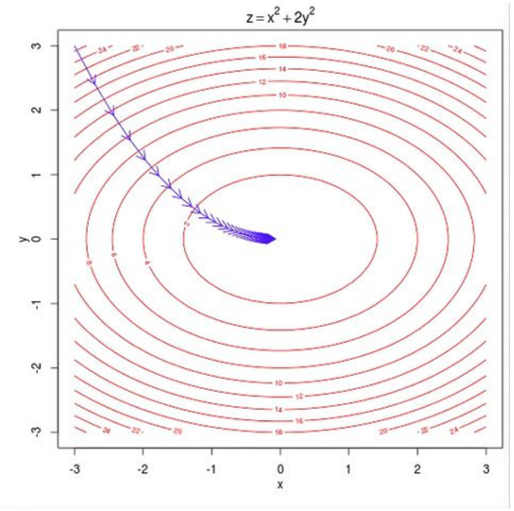
Gradient descent is the dominant method to optimize networks parameters θ to minimize loss function L(θ).

The update rule is (**α** is the “learning rate”):

$$W_{i+1} \leftarrow W_i - \alpha \nabla L(W)$$

The gradient is typically averaged over a minibatch of examples in minibatch **stochastic gradient descent.**





Cost function & Loss

$$J(W) = \mathbb{E}_{x,y \sim \hat{p}_{data}} L(f(x; W), y)$$

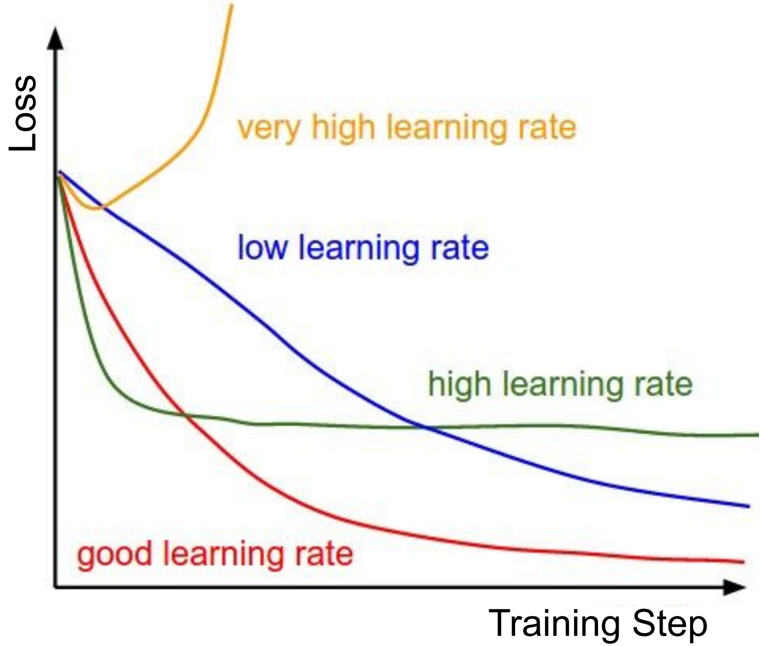
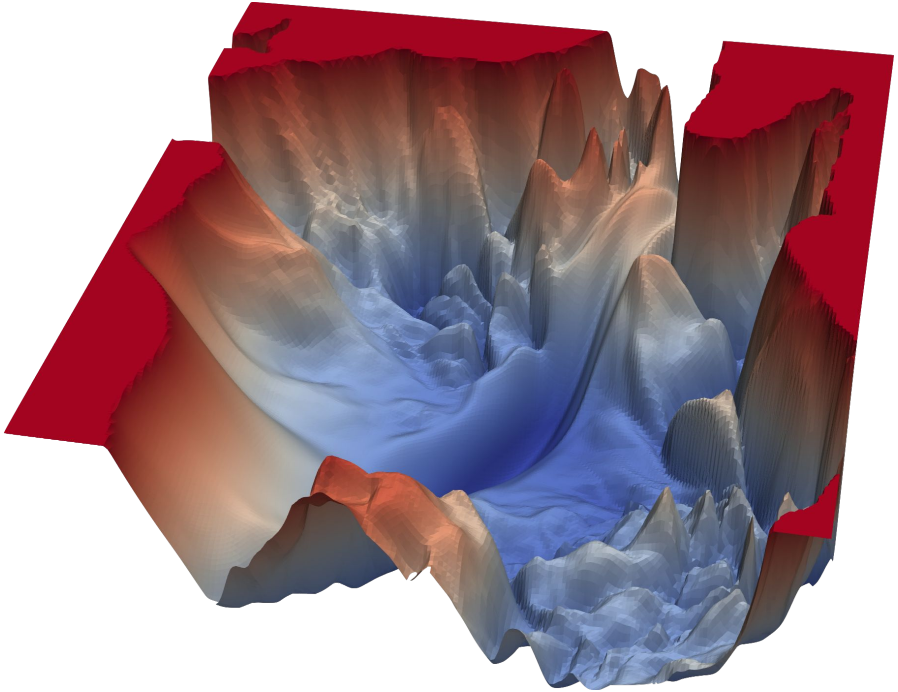


Fig. credit: http://cs231n.github.io/neural-networks-3/

Stochastic Gradient Descent variants

Gradient descent can get trapped in the abundant saddle points, ravines and local minimas of neural networks loss functions.



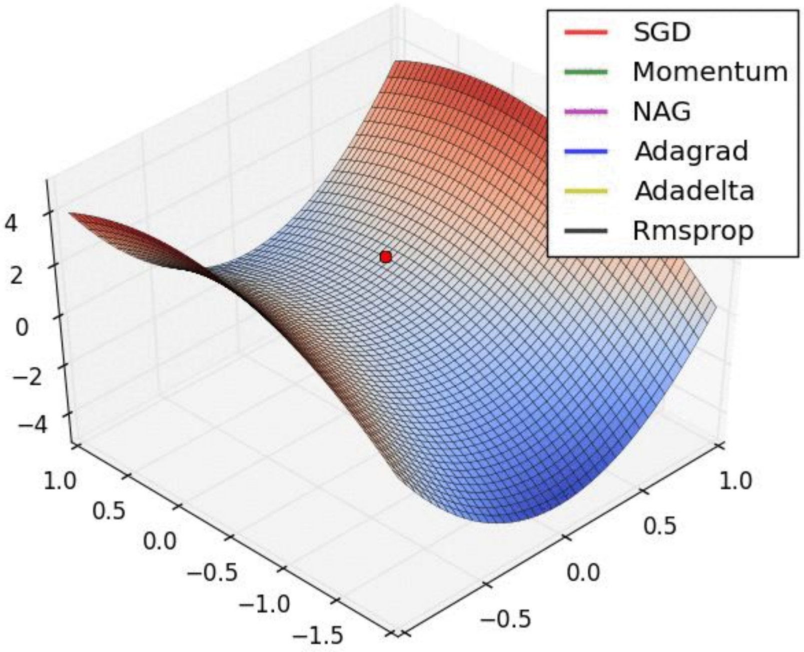
VGG-56 loss landscape: [arXiv:1712.09913](https://arxiv.org/abs/1712.09913)

Stochastic Gradient Descent variants

Gradient descent can get trapped in the abundant saddle points, ravines and local minimas of neural networks loss functions.

To accelerate the optimization on such functions we use a variety of methods:

- SGD + Momentum
- Nestrov
- AdaGrad
- RMSProp
- …
- Adam



The generalization of networks trained with adaptive optimizers (e.g. Adam) has been questioned in many recent studies. SGD + Momentum is still used for many state-of-the-art networks (e.g. ResNet variants). For example, arXiv:1711.05101

Convolutional Neural Networks (CNNs)

CNNs implement the convolution operation over input.

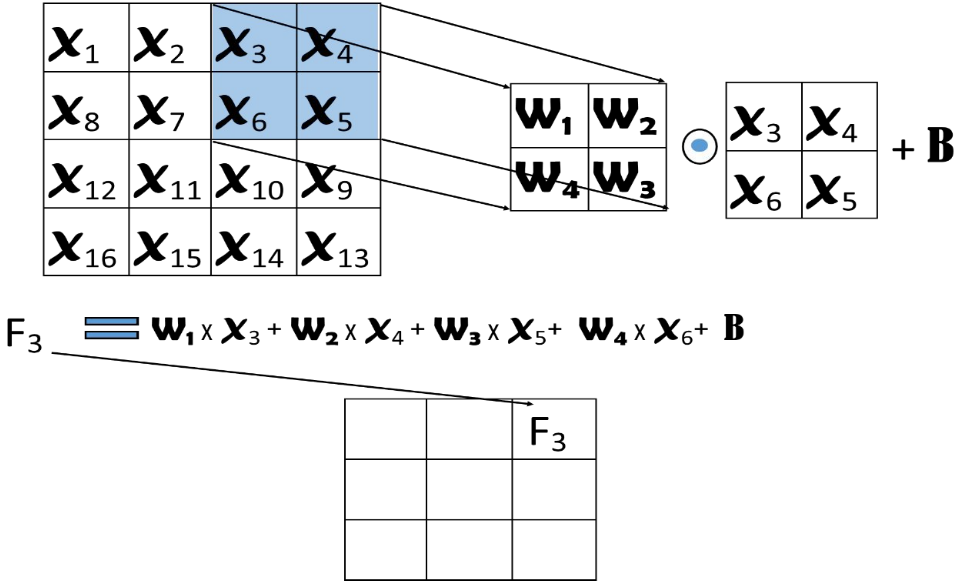


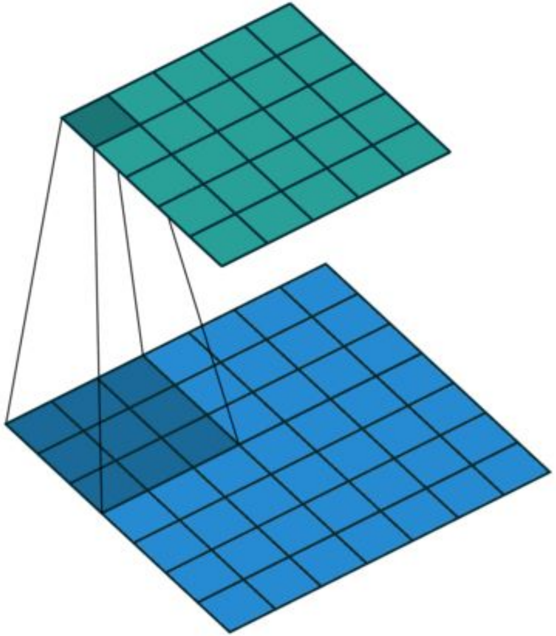
Fig. credit: goo.gl/4Qgn5U

Convolutional Neural Networks (CNNs)

CNNs implement the convolution operation over input.

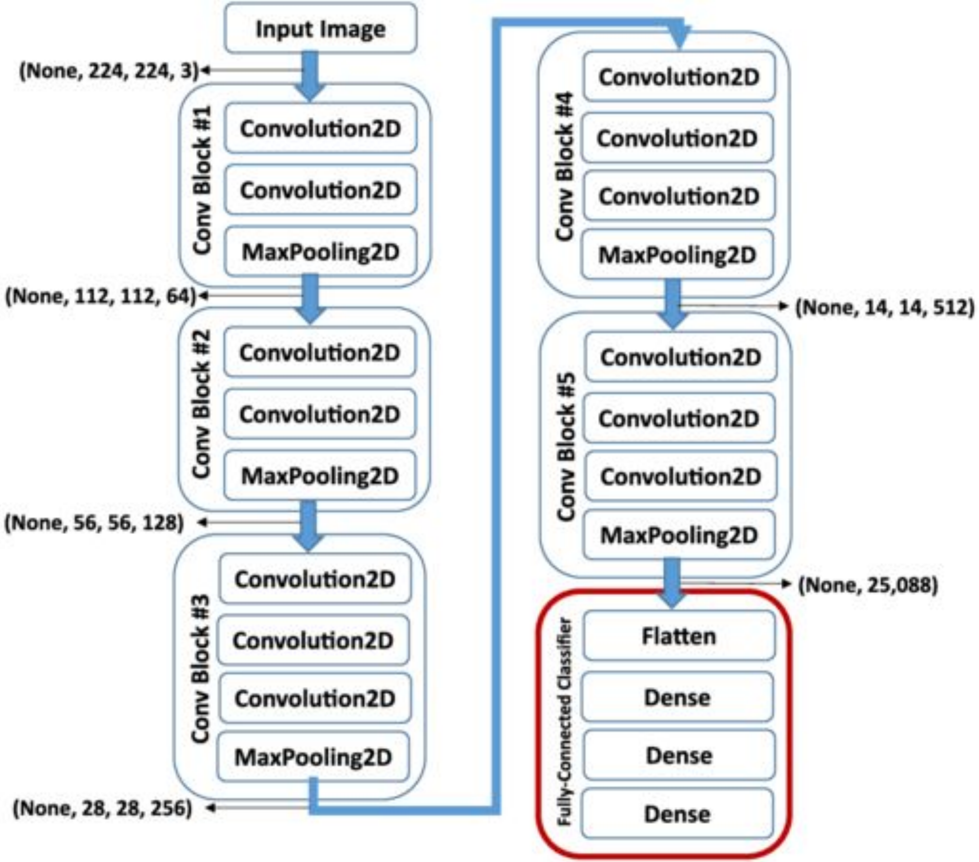
CNNs are translation equivariant by construction.

CNNs achieve: sparse connectivity, parameter sharing and translation equivariance.



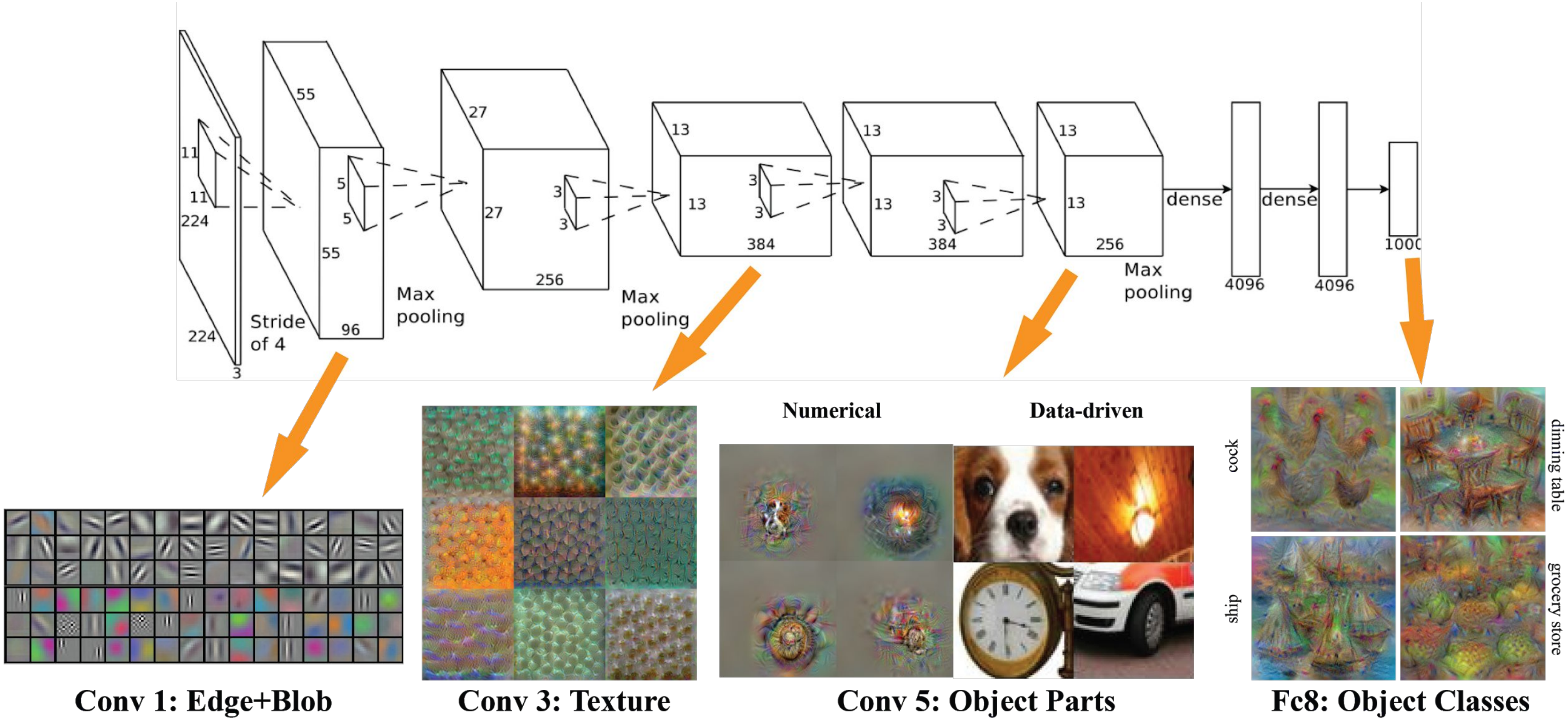
Sliding convolution kernel with size 3x3 over an input of 7x7.

Let us put it all together: a typical CNN network architecture



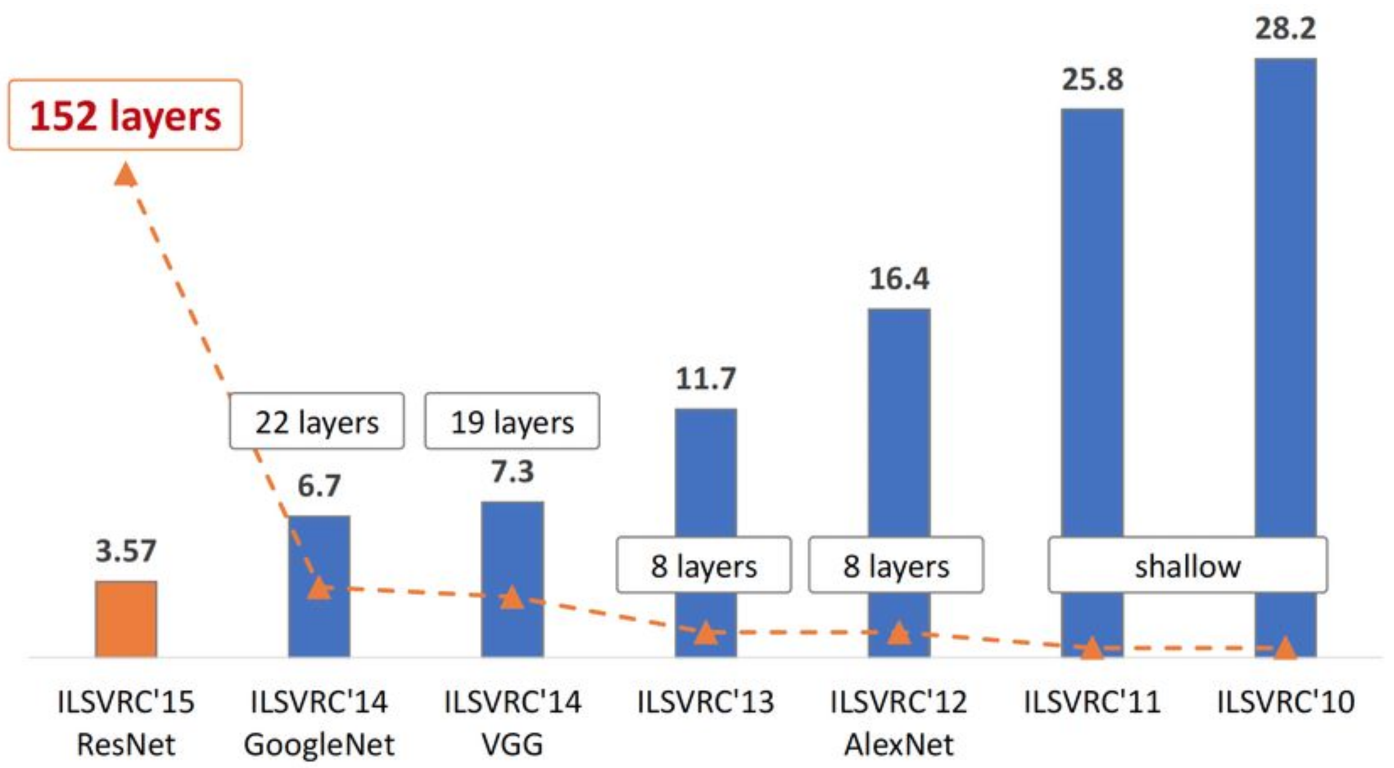
A schematic of VGG-16 Deep Convolutional Neural Network (DCNN) architecture trained on ImageNet (2014 ILSVRC winner)

AlexNet: the onset of Deep Learning winning streak



mNeuron: A Matlab Plugin to Visualize Neurons from Deep Models vision03.csail.mit.edu/cnn_art/index.html

The Revolution (or revelation) of Depth



What are Deep Neural Networks?

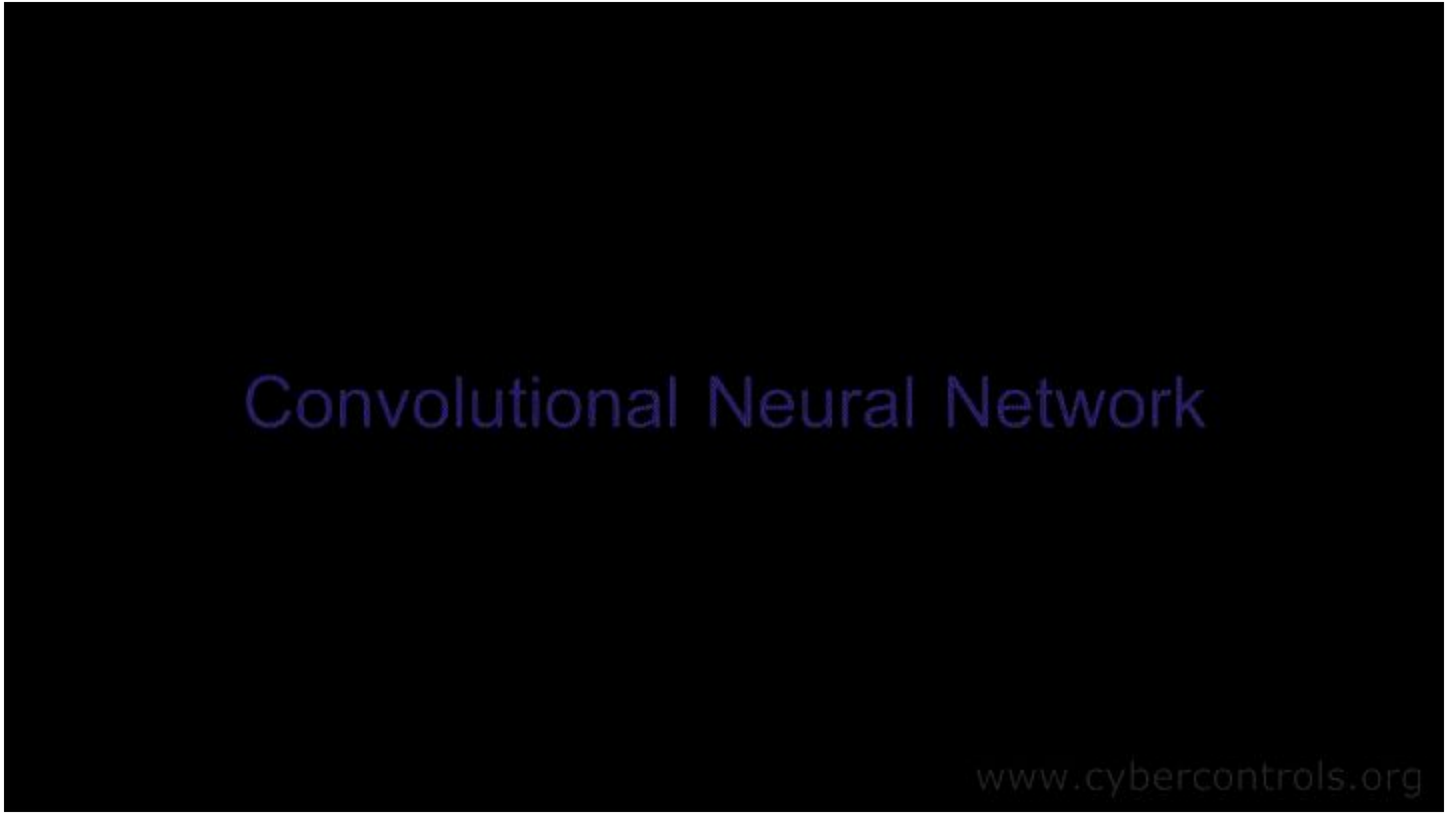
Long story short:

“A family of **parametric**, **non-linear** and **hierarchical** **representation learning functions**, which are massively optimized with stochastic gradient descent to **encode domain knowledge**, i.e. domain invariances, stationarity.” -- Efstratios Gavves

Fully connected neural networks

Animation adapted from: youtube.com/watch?v=3JQ3hYko51Y&feature=youtu.be

Convolutional Neural Networks (CNNs)



Animation adapted from: youtube.com/watch?v=3JQ3hYko51Y&feature=youtu.be

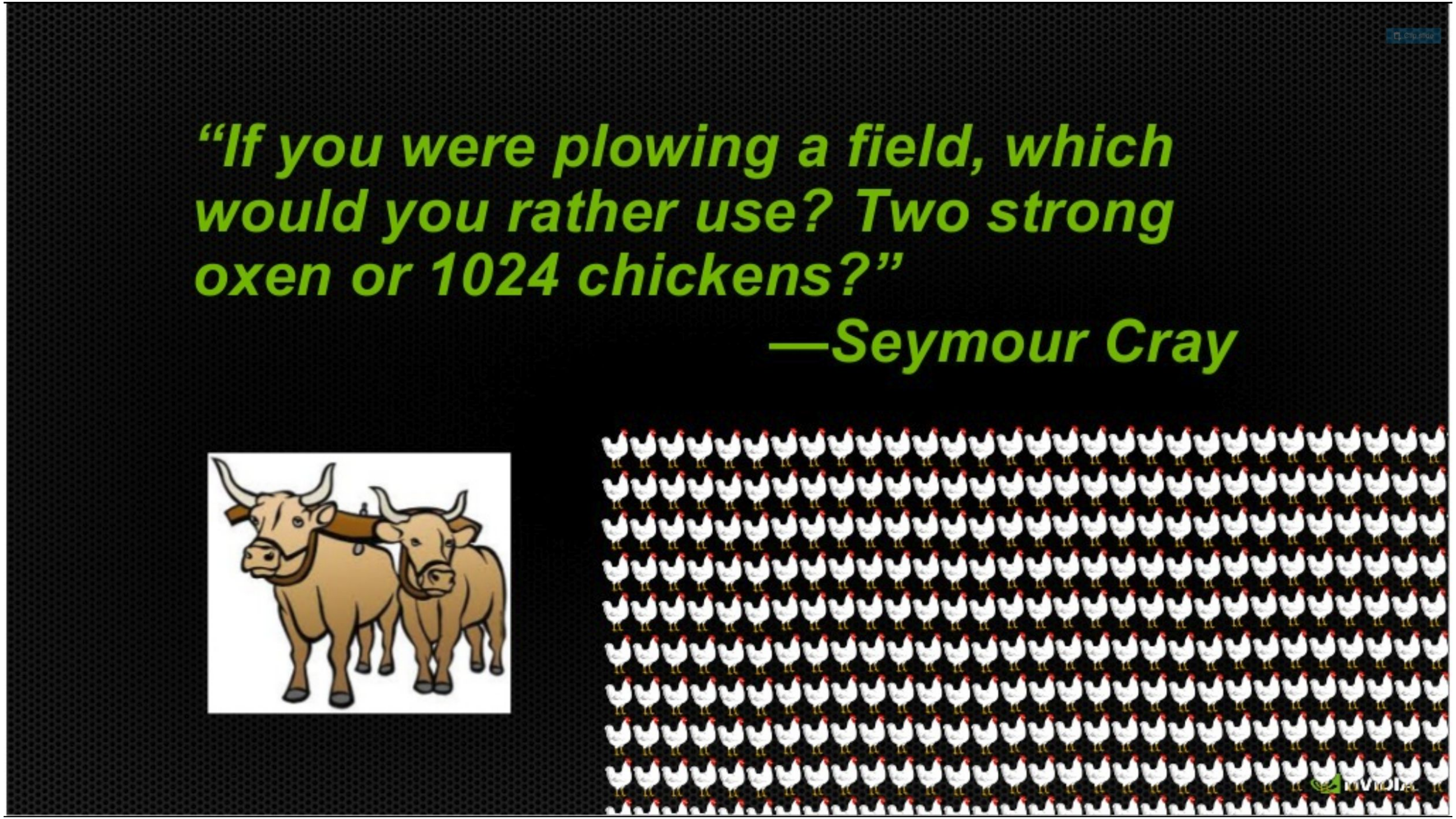
GPUs vs. CPUs

Modified version of comparison made in CS231n cs231n.stanford.edu/slides/2019/cs231n_2019_lecture06.pdf

	Cores	Clock speed	Memory	Memory bandwidth	Price	Peak perf.
CPU (Intel Core i7-7700k)	4	4.2 GHz	System RAM	~35 GB/s	~$350	~540 GFLOPs FP32
GPU (Nvidia RTX 2080 Ti)	3584	1.6 GHz	11 GB GDDR6	616 GB/s	~$1200	~13.4 TFLOPs FP32

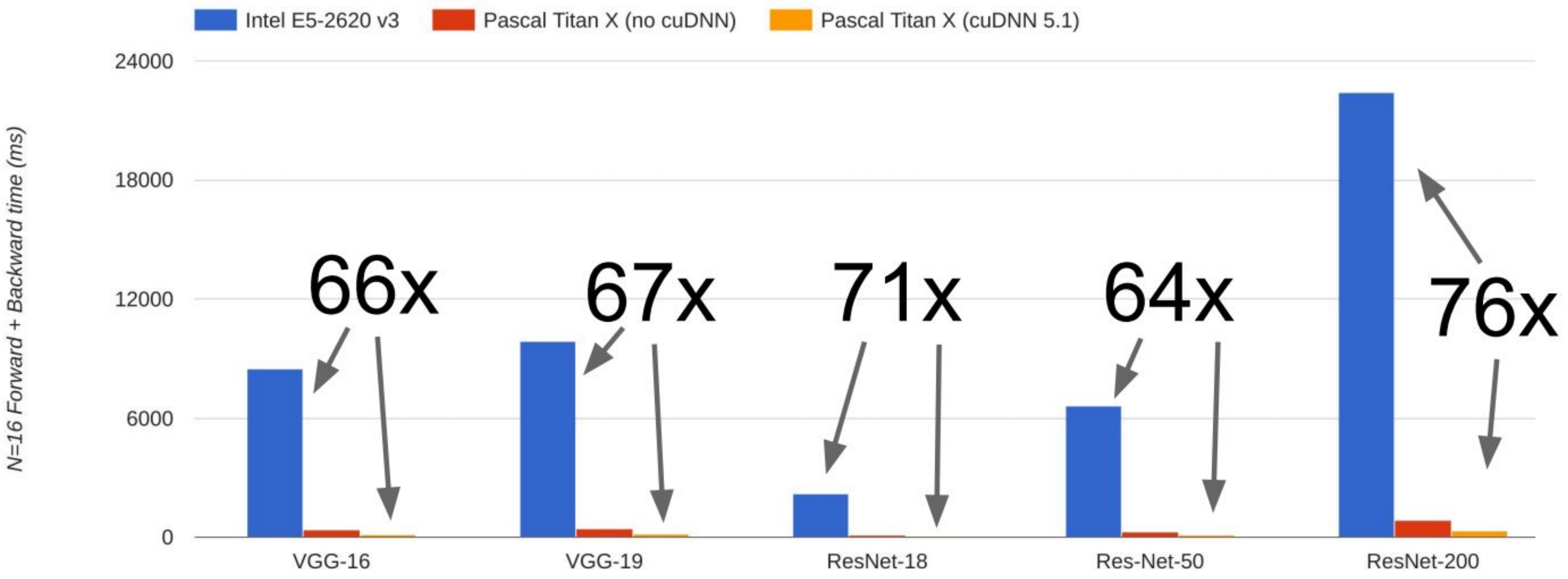
- CPU have less but more powerful cores, GPU have many “mini”-cores great for parallel computations of linear algebra/neural networks
- GPU have much higher memory bandwidth, great for shuffling big tensors
- GPU memory is bandwidth optimized while CPU are latency optimized

GPUs vs. CPUs


"If you were plowing a field, which would you rather use? Two strong oxen or 1024 chickens?"
—Seymour Cray

GPUs vs. CPUs

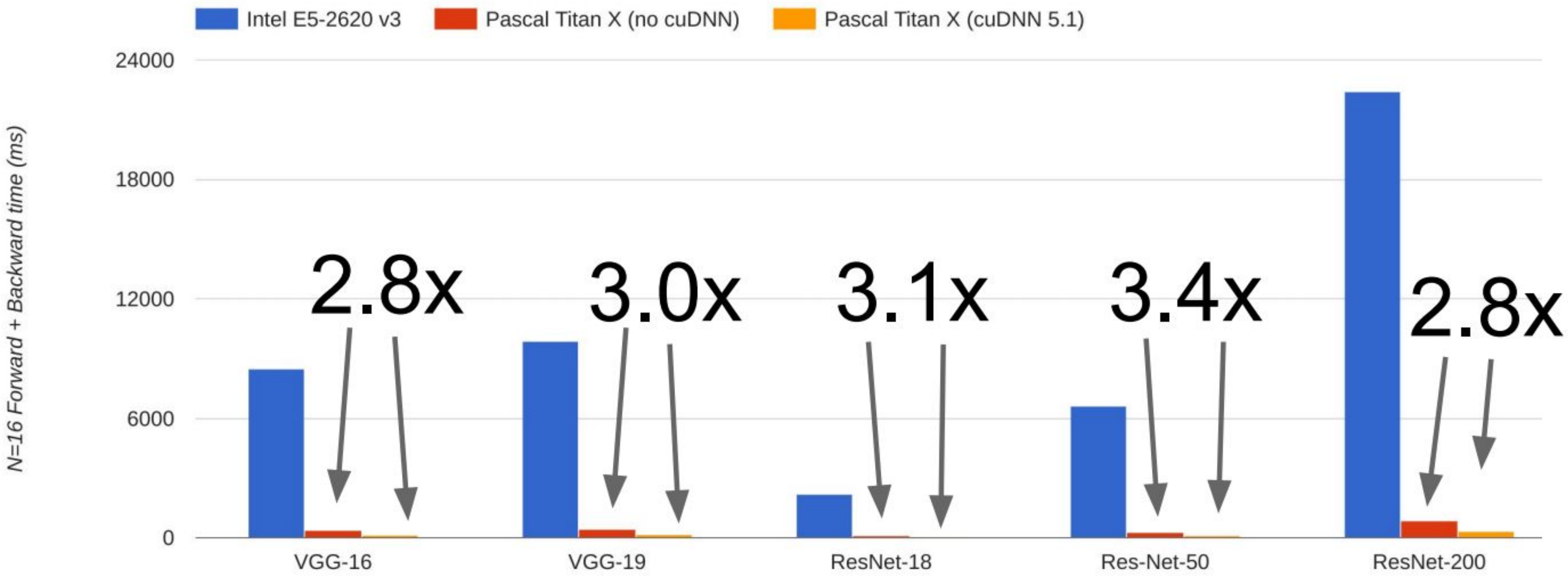
Figure from CS231n cs231n.stanford.edu/slides/2019/cs231n_2019_lecture06.pdf



Unfair comparison to unoptimized CPU primitives, optimized primitives are 5-10x faster

GPUs vs. CPUs

Figure from CS231n cs231n.stanford.edu/slides/2019/cs231n_2019_lecture06.pdf

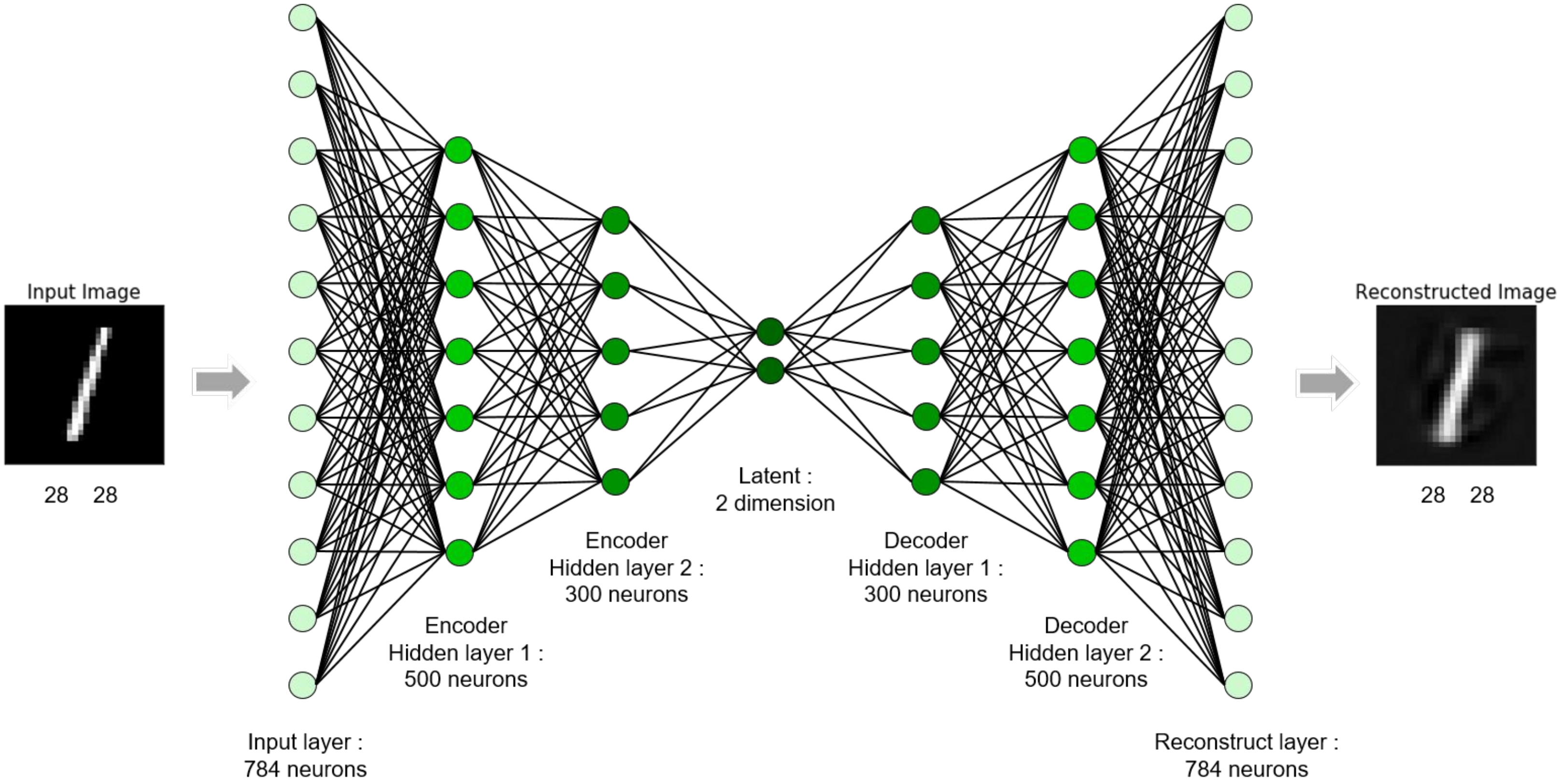


cuDNN library provides GPU accelerated deep learning primitives

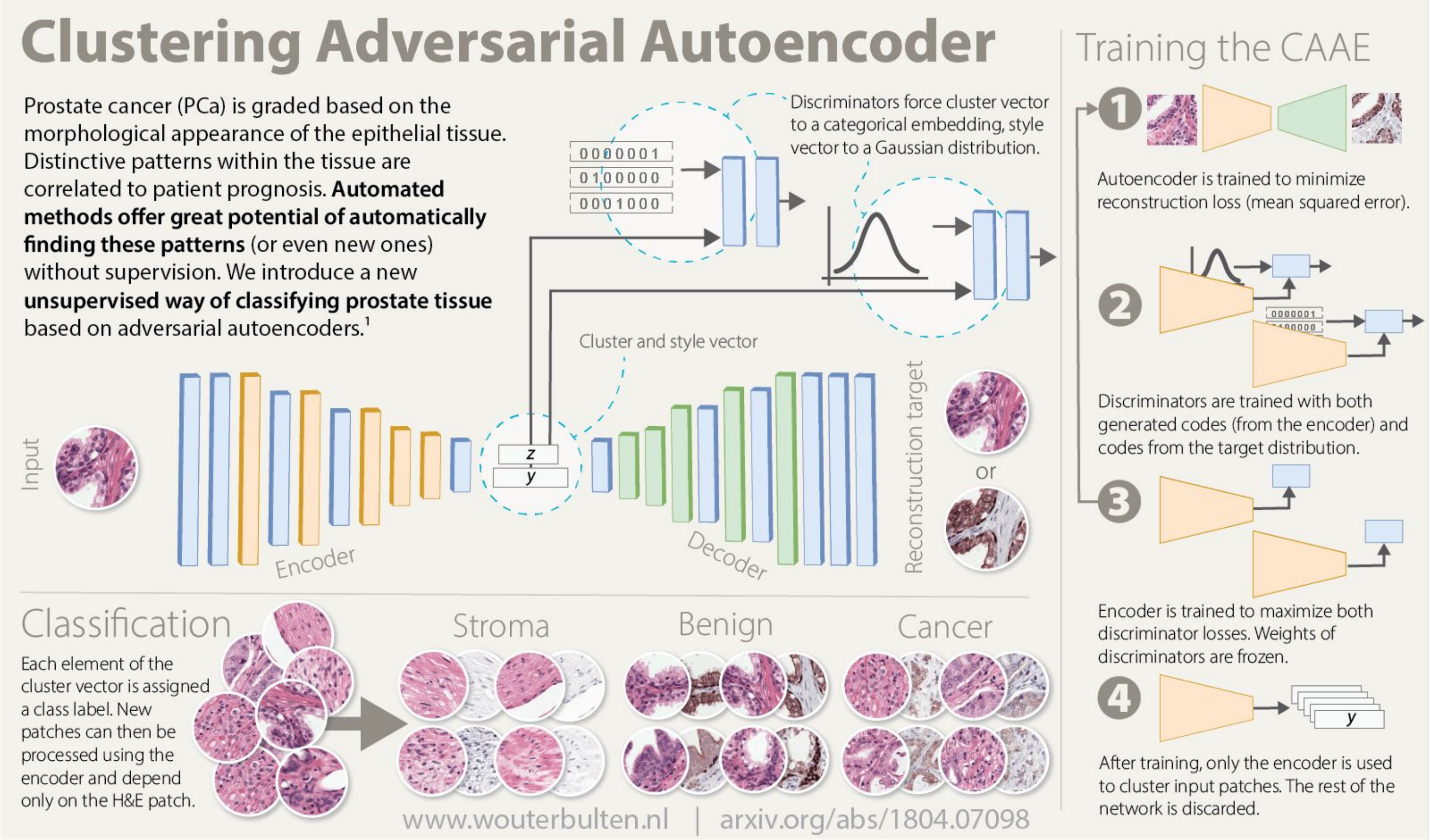
Thank You

Applications of Deep Learning

Autoencoders

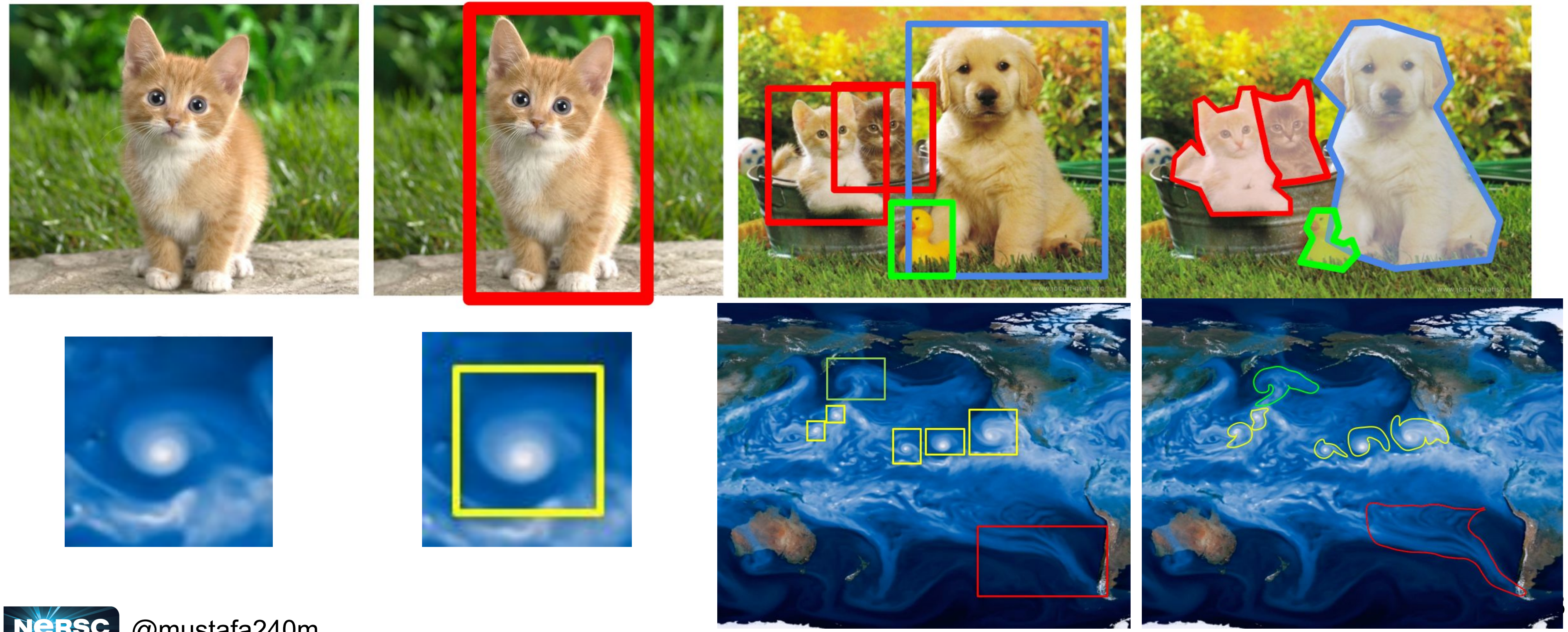


Unsupervised diagnostics using autoencoders

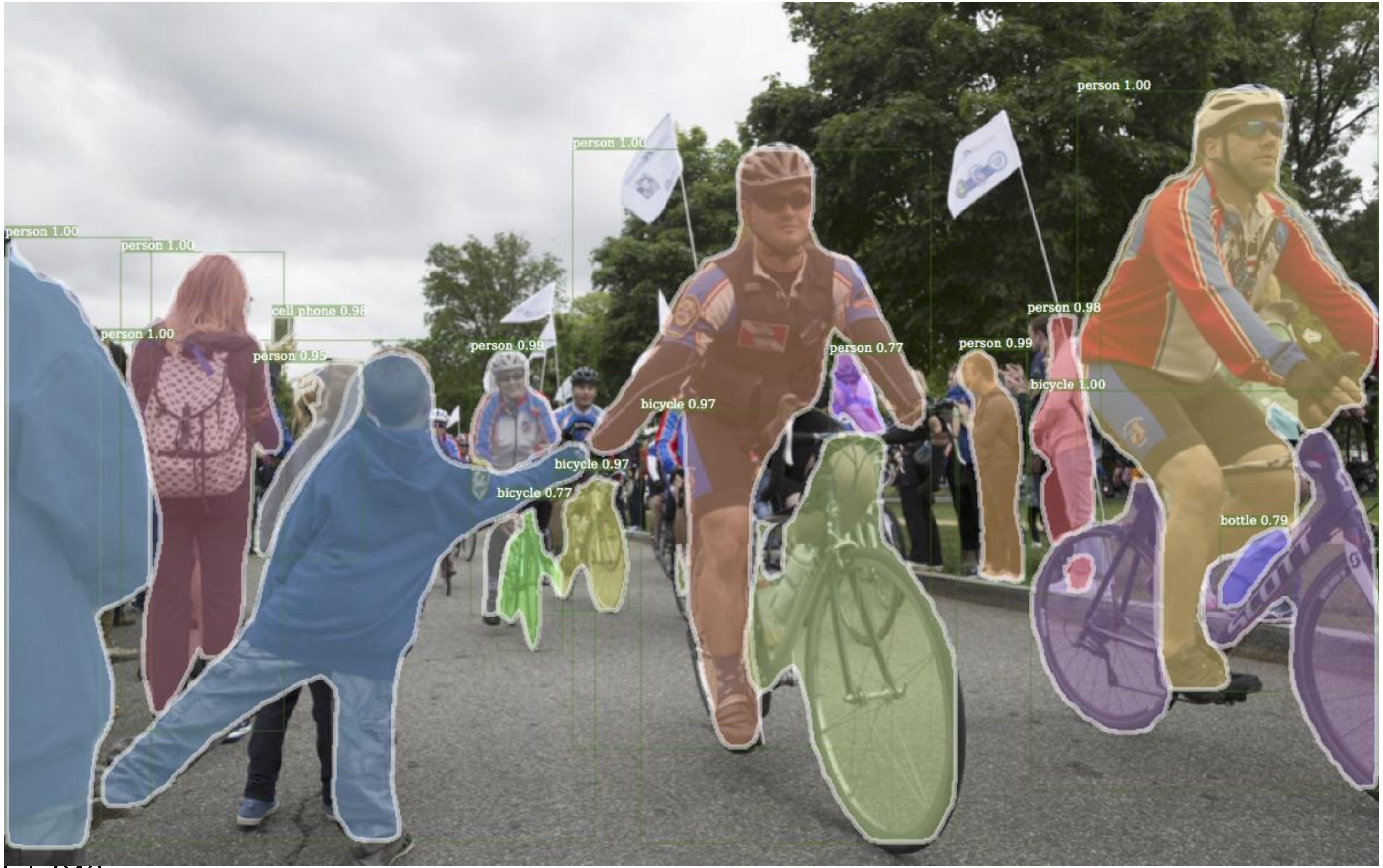


Applications of DNNs: Classification and Segmentation

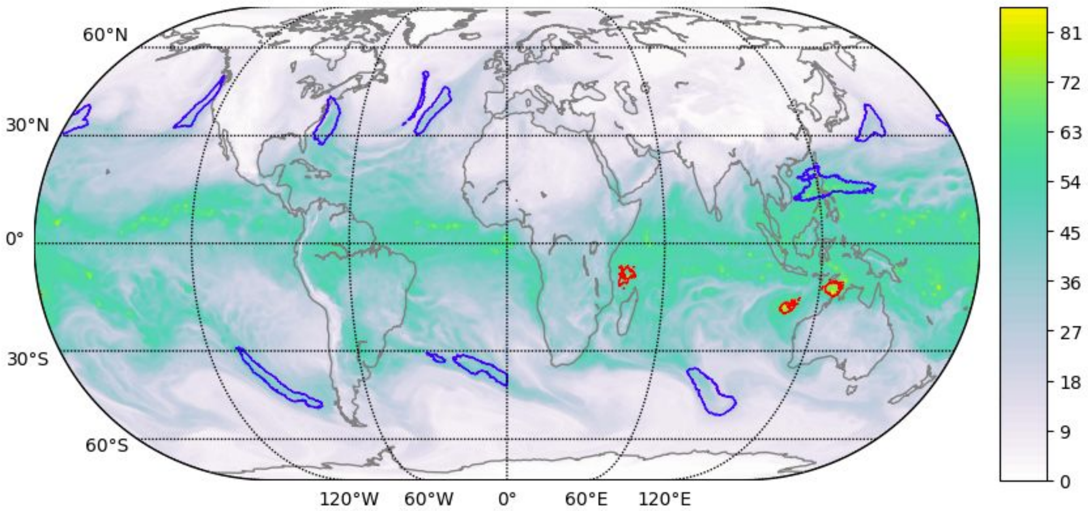
Classification | **Classification + Localization** | **Object Detection** | **Instance Segmentation**



Applications of DNNs: Classification and Segmentation



Climate Segmentation Results



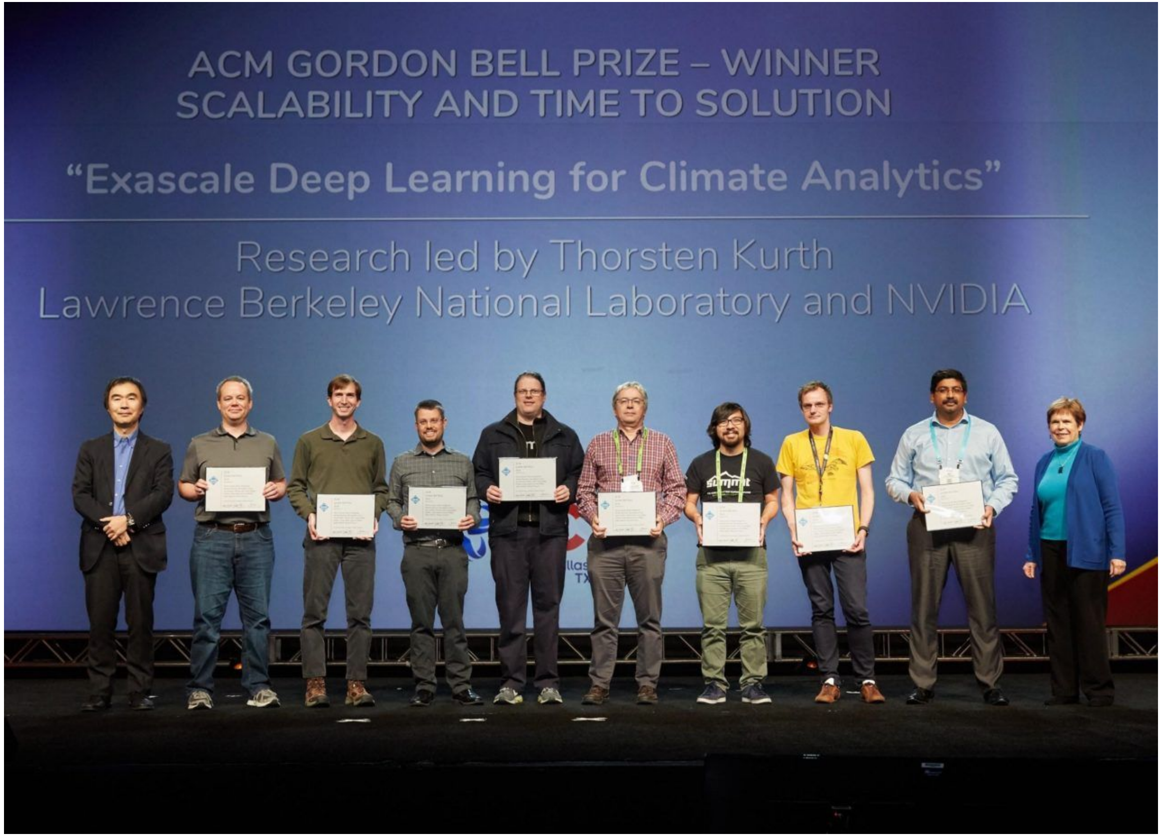
Collaboration between NERSC, NVIDIA, UCB, OLCF

Pixel-level classification of extreme weather phenomena

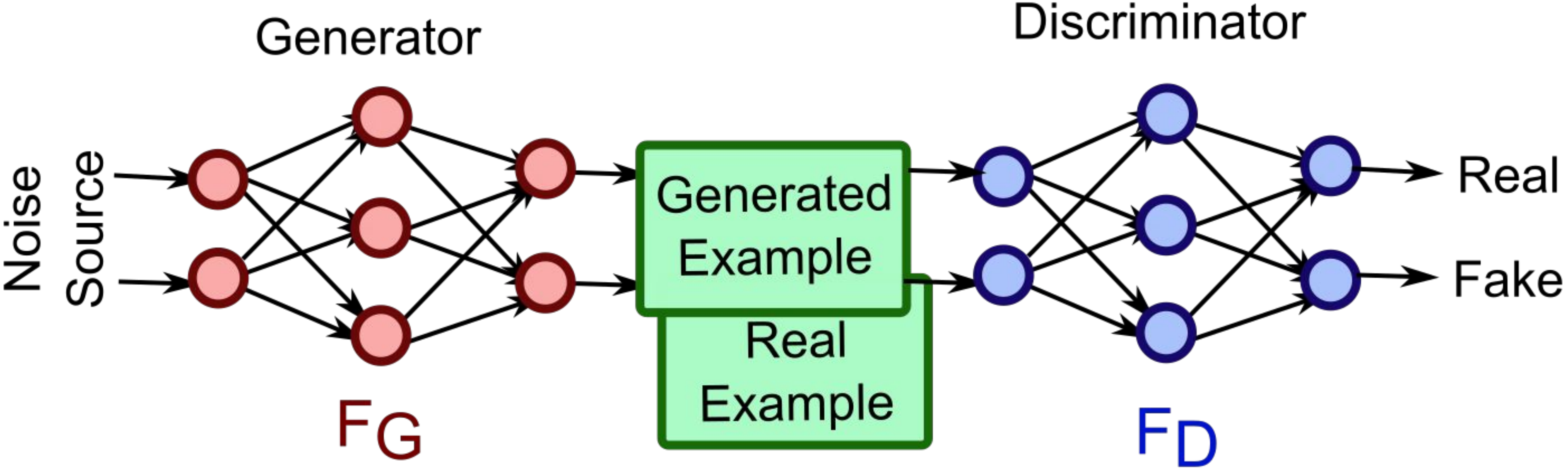
- 3 classes: atmospheric river, tropical cyclone, background

Exascale Deep Learning for Climate Analytics: Thorsten Kurth et al. arXiv:1810.01993

Climate Segmentation Results



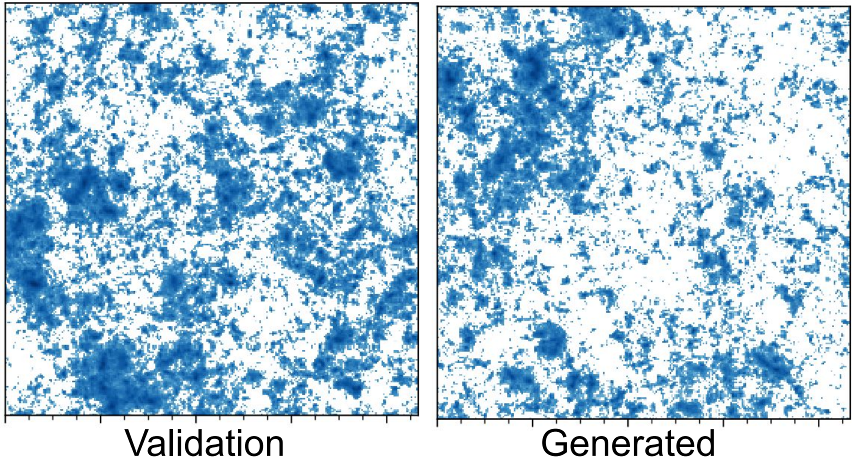
Applications of DNNs: Generative Adversarial Networks

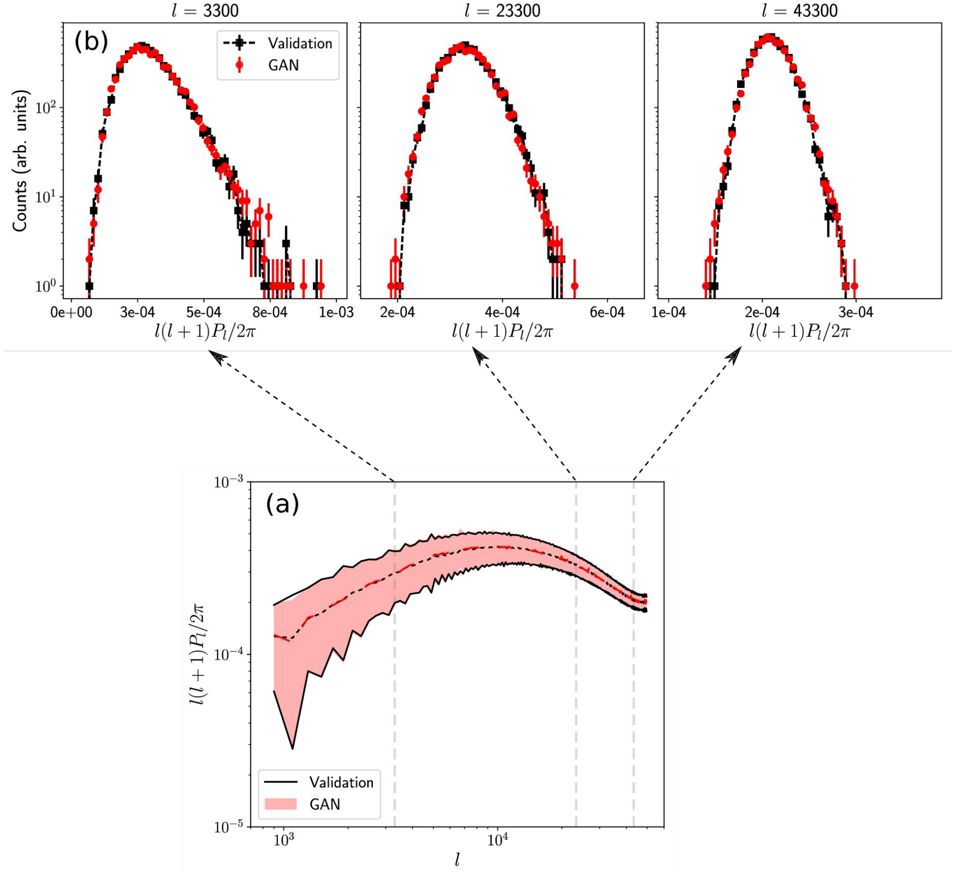


Applications of DNNs: Generative Adversarial Networks

Figure 5: 1024 × 1024 images generated using the CELEBA-HQ dataset. See Appendix F for a larger set of results, and the accompanying video for latent space interpolations.

CosmoGAN: generating high-fidelity cosmology maps





GANs generated maps exhibit the same gaussian and non-gaussian structures as full simulations.

CosmoGAN: Mustafa, Bard, Bhimji, Lukic, Al-Rfou, Kratochvil, arXiv:1706.02390

CaloGAN: Particle Calorimeter Showers