AI FOR SCIENCE
A PRACTICAL INTRODUCTION TO DEEP LEARNING

WITH KERAS AND TENSORFLOW

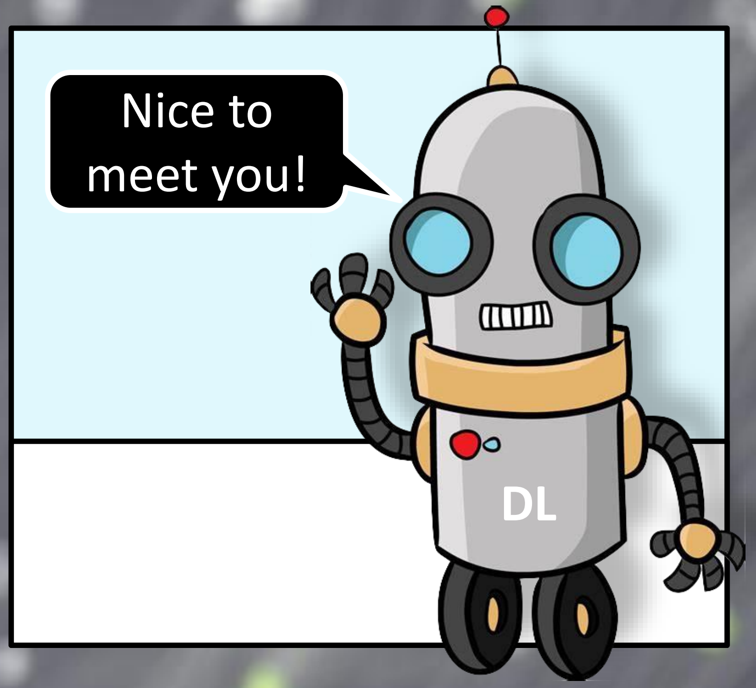
LEARNING GOALS

In couple of hours, we can only travel so far

Main goal:
Become familiar with main ideas and process

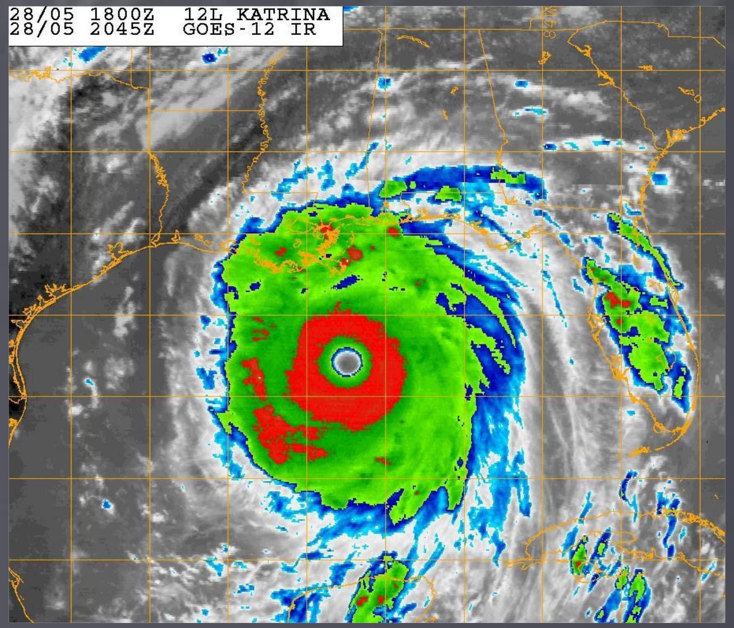
A starting point for solving your own problems

INTRO TO DL, PART 1

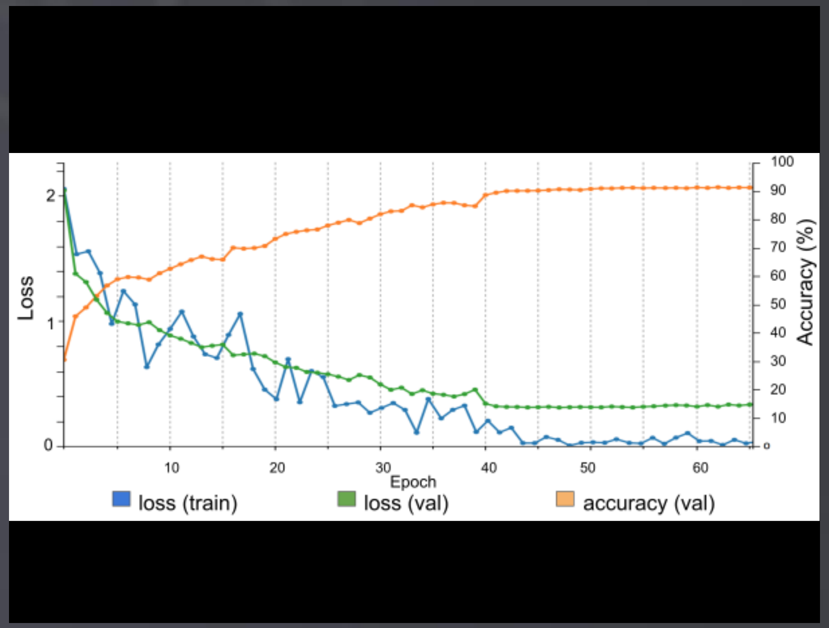


LAB 1: CNNs AND KERAS

LAB 2: TROPICAL CYCLONES



LAB 3: CFD STEADY FLOW

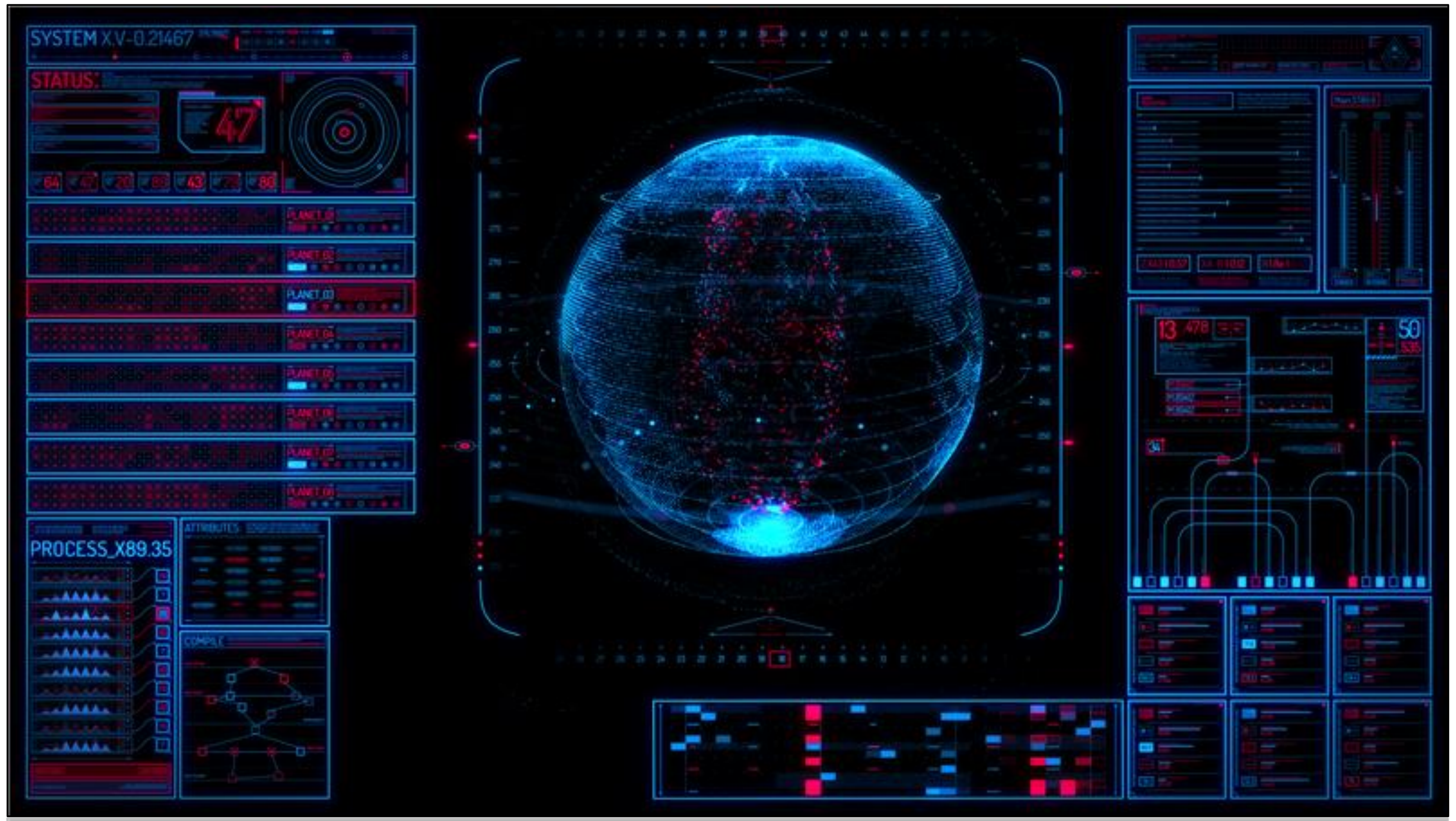


AGENDA

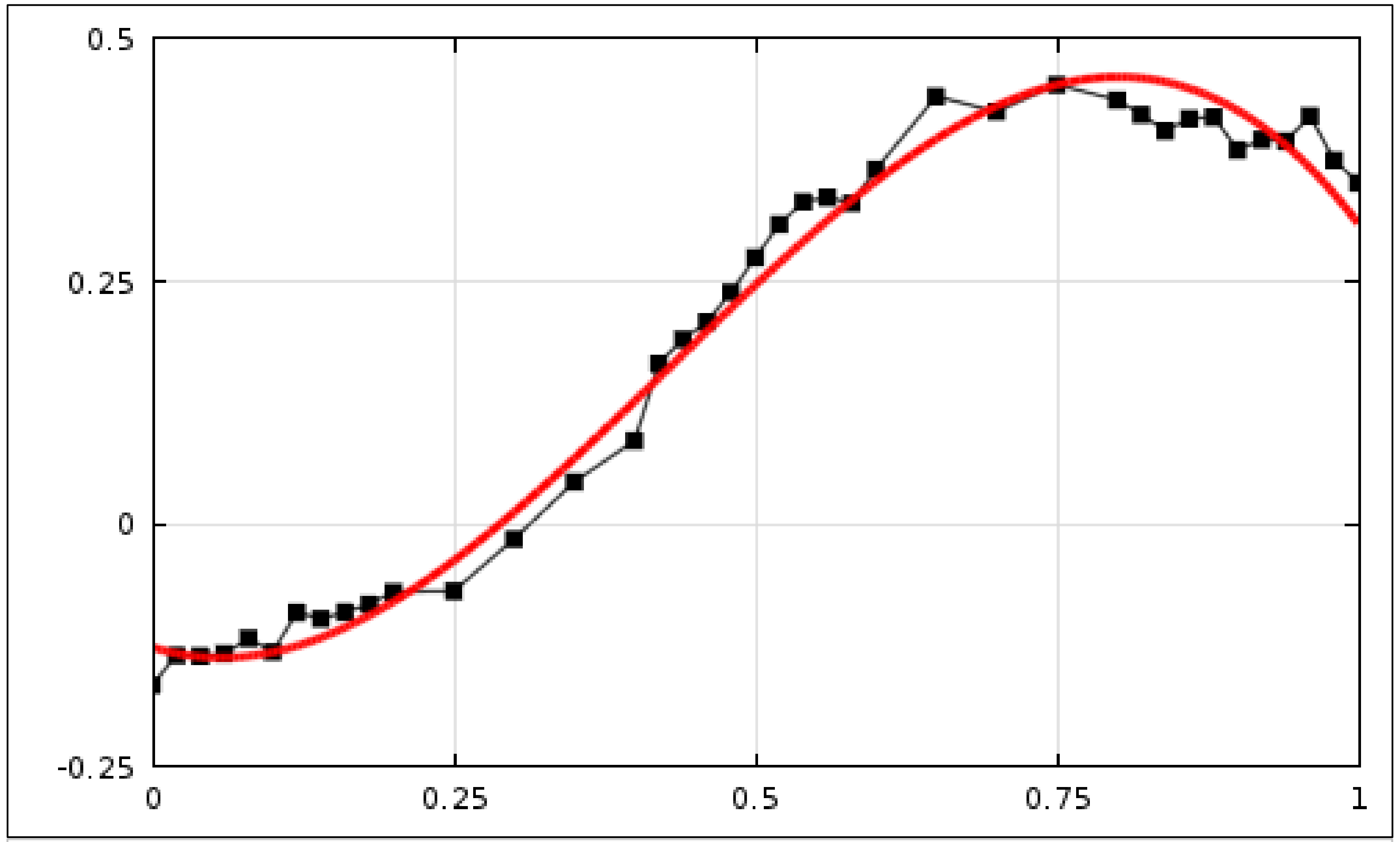
DEEP LEARNING ANALOGIES

What is this deep learning thing, anyway?

A NEW TYPE OF SOFTWARE



A GENERALIZATION OF CURVE FITTING



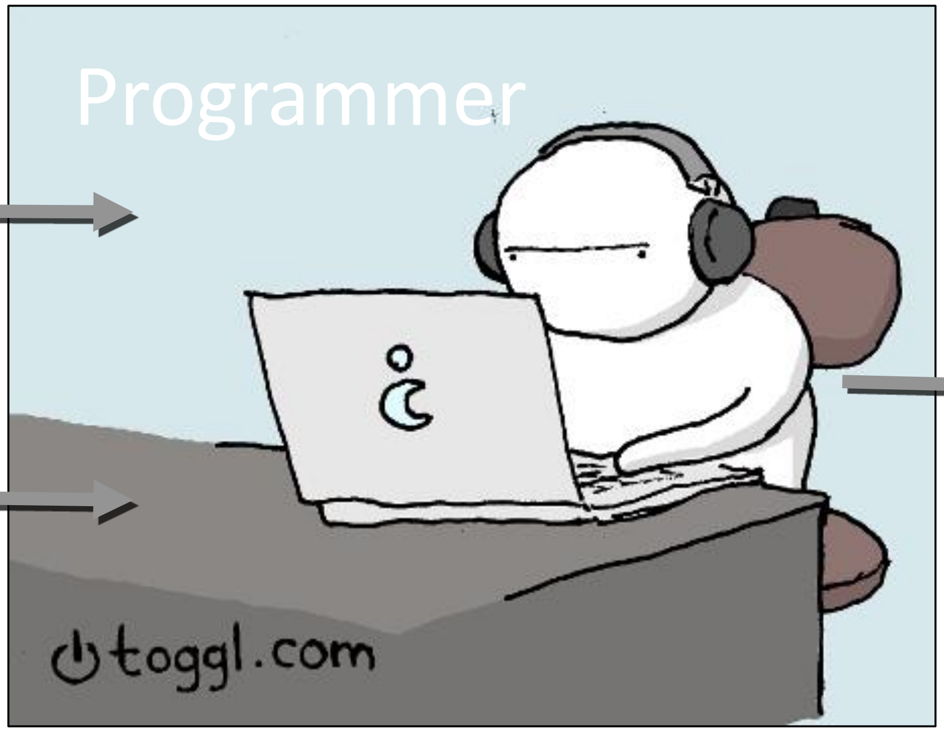
A NEW WAY TO BUILD SOFTWARE

Traditional Programming vs Machine Learning

SOFTWARE 1.0:
Traditional Programming

Task

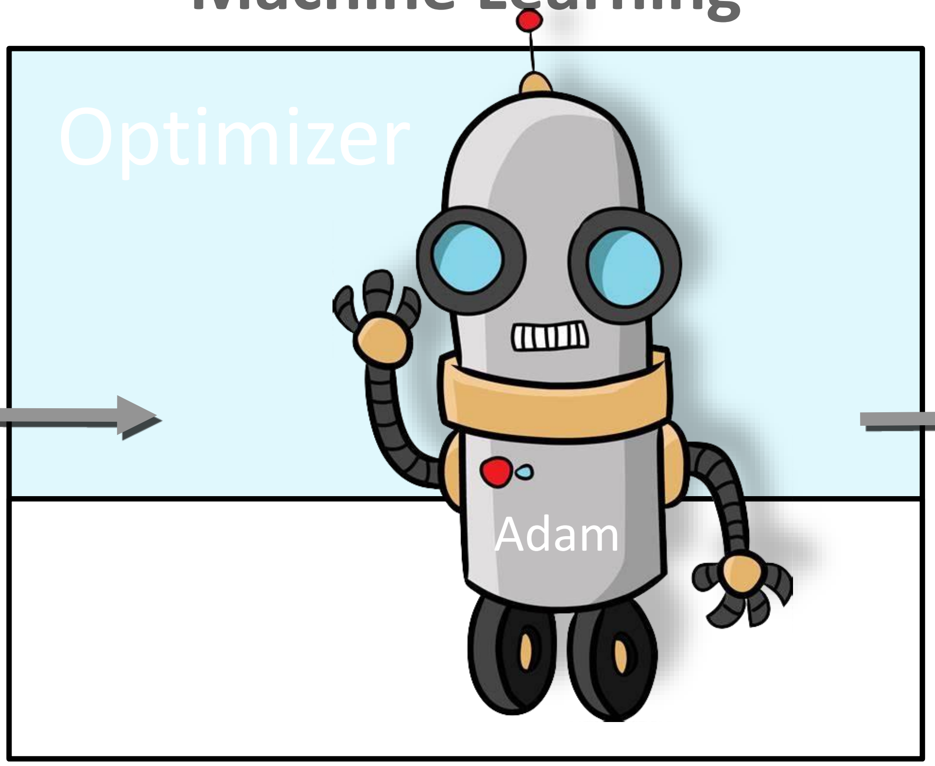
Expert Knowledge



Human Readable Function

SOFTWARE 2.0:
Machine Learning

Tons of Examples



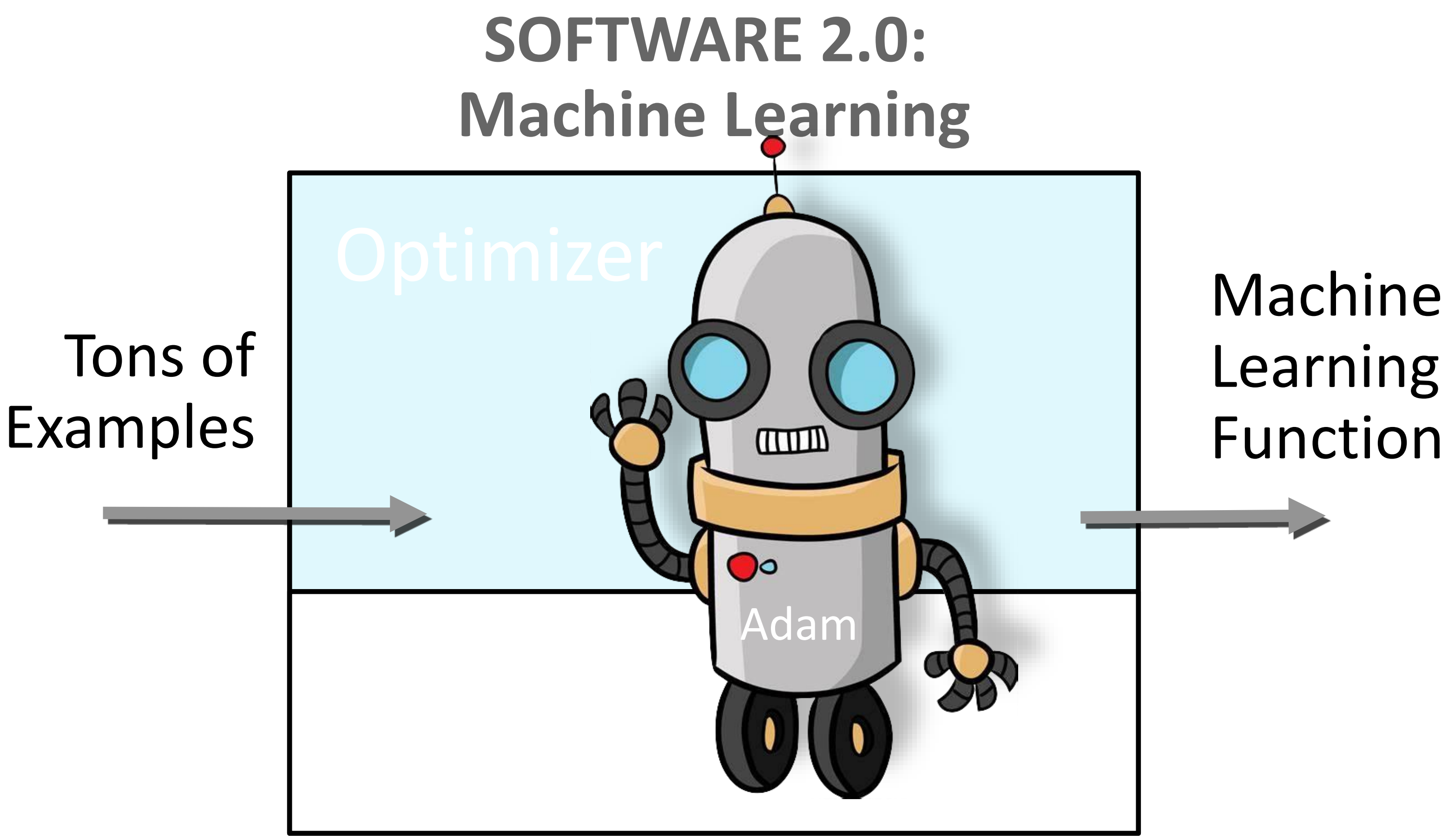
Machine Learning Function

A DIFFERENT WAY TO BUILD SOFTWARE

Traditional Programming vs Machine Learning

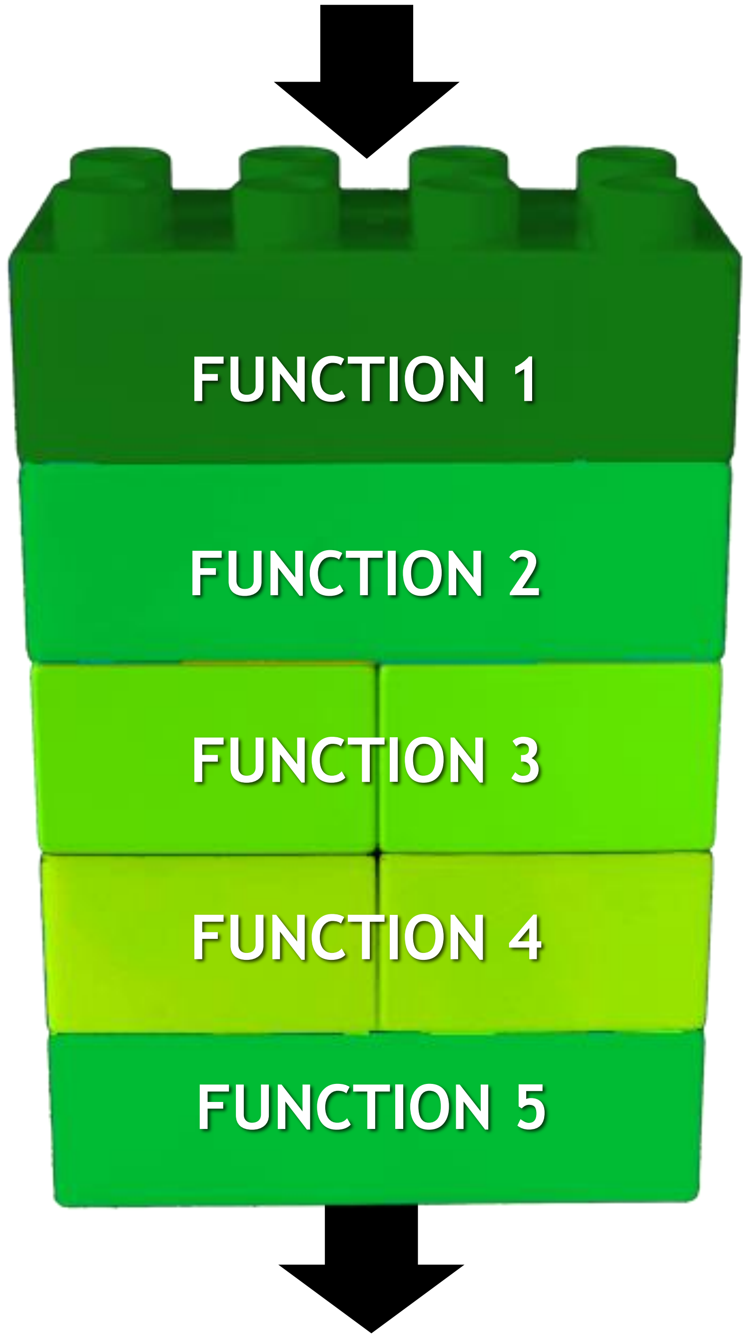
Goals for today:

1. **Learn to use this new approach**
2. **Revolutionize Science**



A DIFFERENT WAY TO BUILD SOFTWARE

Hand written vs learnt functions



HAND-WRITTEN FUNCTION

```
Function1(T,P,Q)

update_mass()

update_momentum()

update_energy()

do_macrophysics()

do_microphysics()

y = get_precipitation()

return y
```

Convert expert knowledge into a function

LEARNED FUNCTION

```
Function1(T,P,Q)

A = relu( w1 * [T,P,Q] + b1)

B = relu( w2 * A      + b2)

C = relu( w3 * B      + b3)

D = relu( w4 * C      + b4)

E = relu( w5 * D      + b5)

y = sigmoid(w6 * E    + b6)

return y
```

Reverse-engineer a function from inputs / outputs

A DIFFERENT WAY TO BUILD SOFTWARE

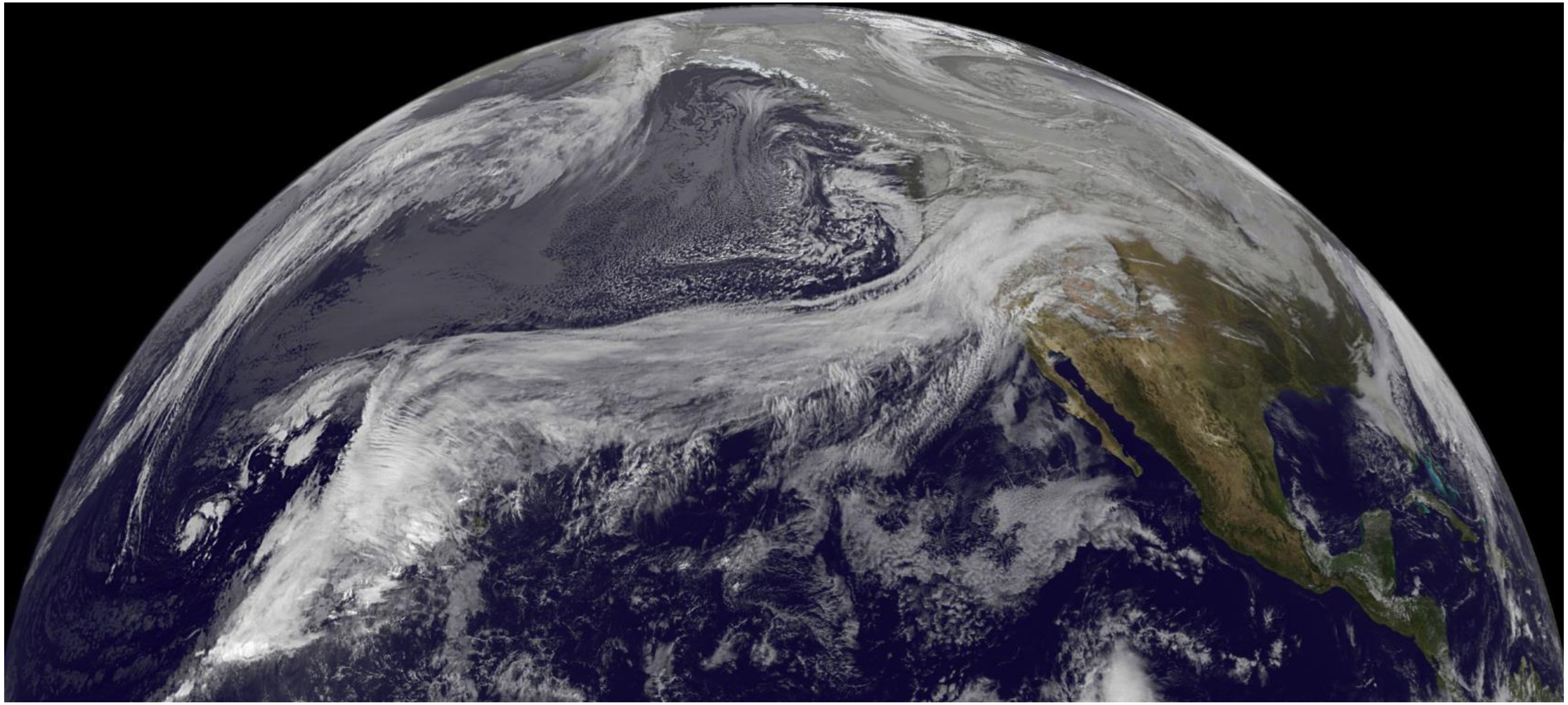
The two approaches are complimentary

MANUAL PROGRAMMING	MACHINE LEARNING
"SOFTWARE 1.0"	"SOFTWARE 2.0"
ENGINEERED	REVERSE-ENGINEERED
LABOR INTENSIVE	AUTOMATIC
EXPLICIT	IMPLICIT
EXPLAINABLE	SUBTLE
SIMPLE	COMPLEX
FROM EXPERTISE	FROM EXAMPLES

(For best results, combine as needed)

A DIFFERENT WAY TO BUILD SOFTWARE

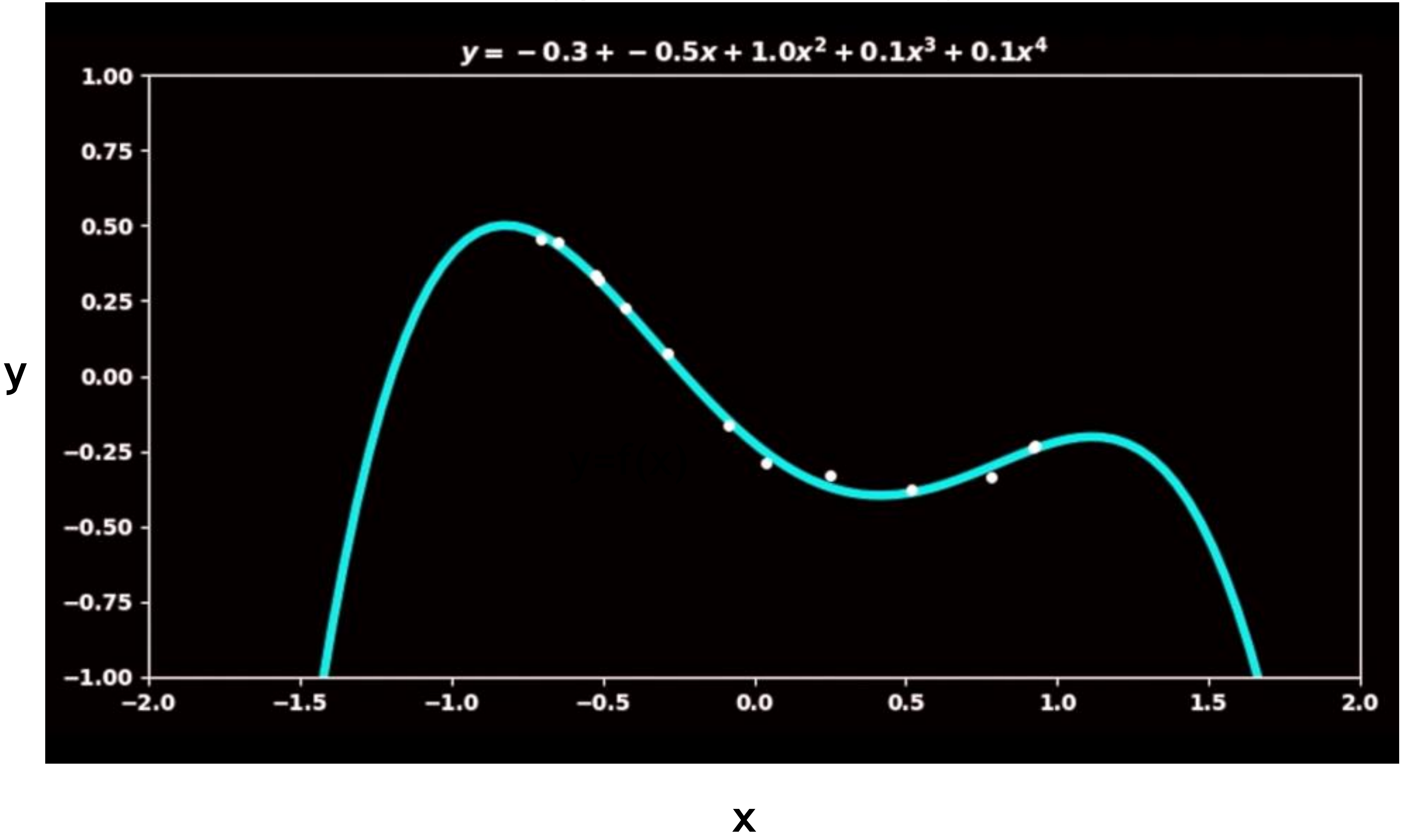
Complex phenomena are best described implicitly.



EXAMPLE: ATMOSPHERIC RIVER

A GENERALIZATION OF CURVE FITTING

Curve fitting provides the starting intution

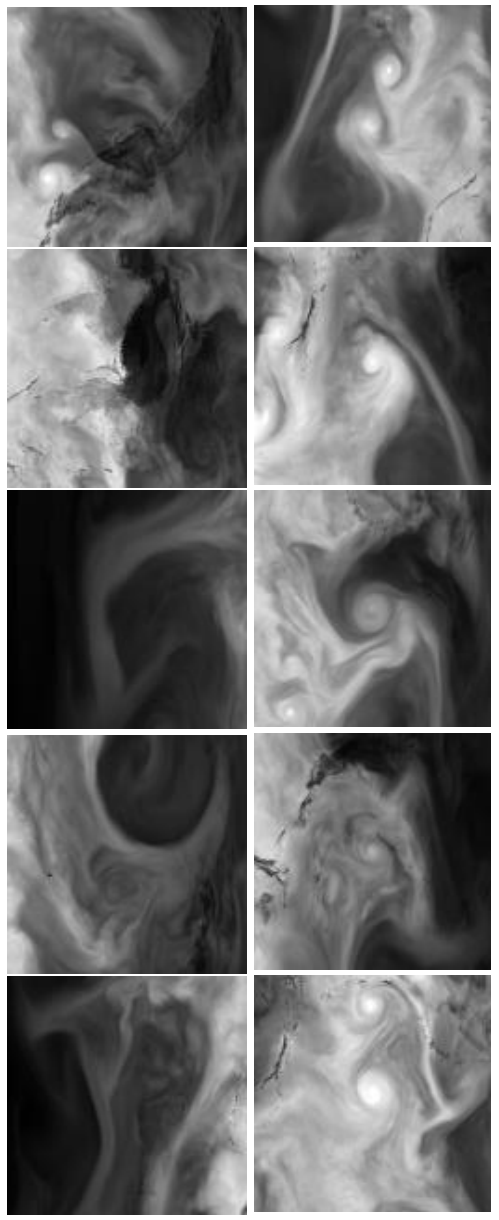


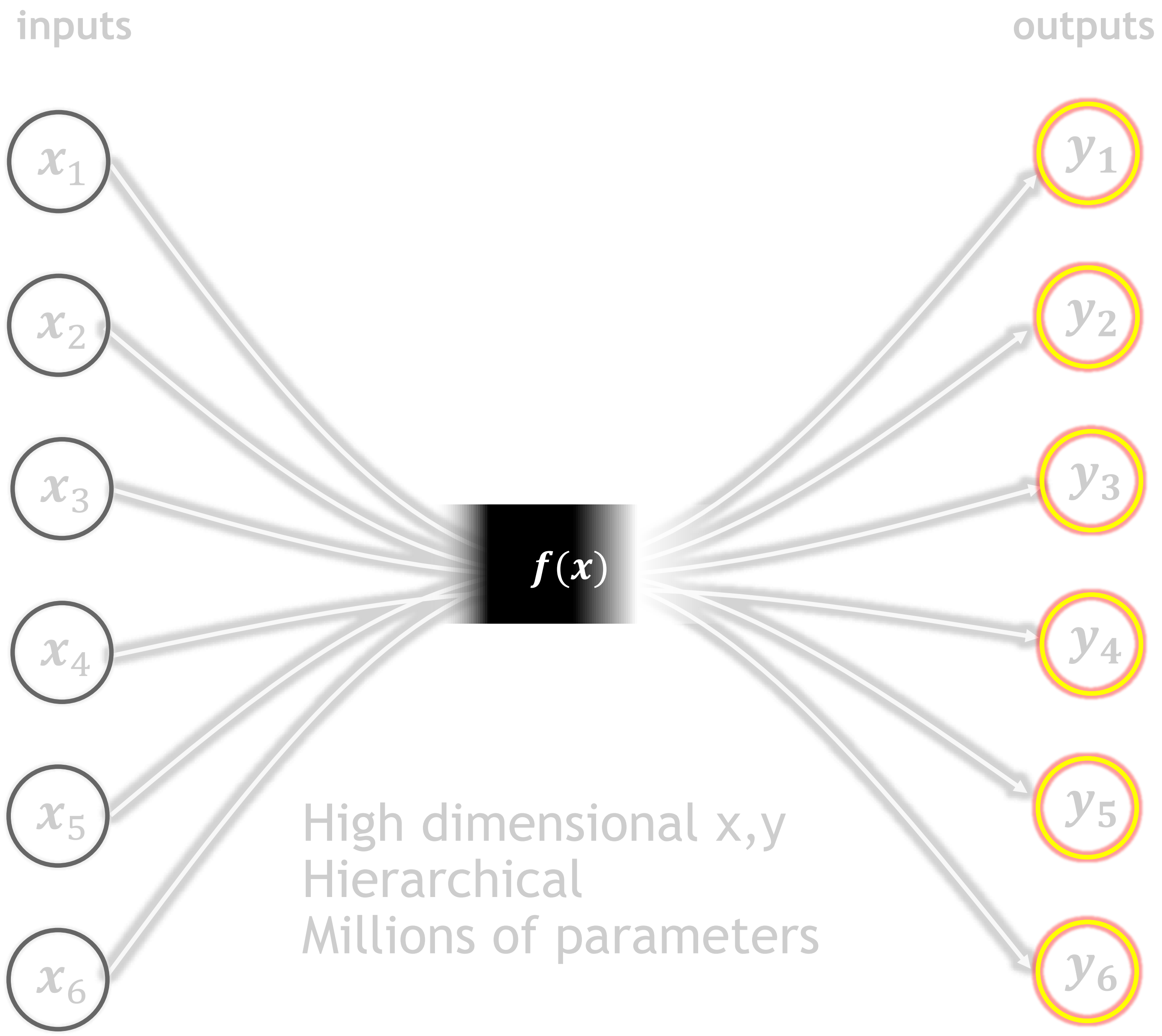
A GENERALIZATION OF CURVE FITTING

Differences from traditional curve fitting

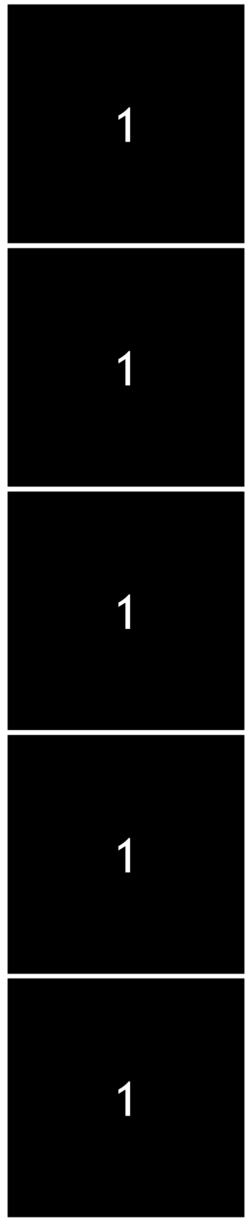
Find f, given x and y

Supervised Deep Learning





High dimensional x,y
Hierarchical
Millions of parameters



IMPLEMENTATION BASICS

AUTO-ML

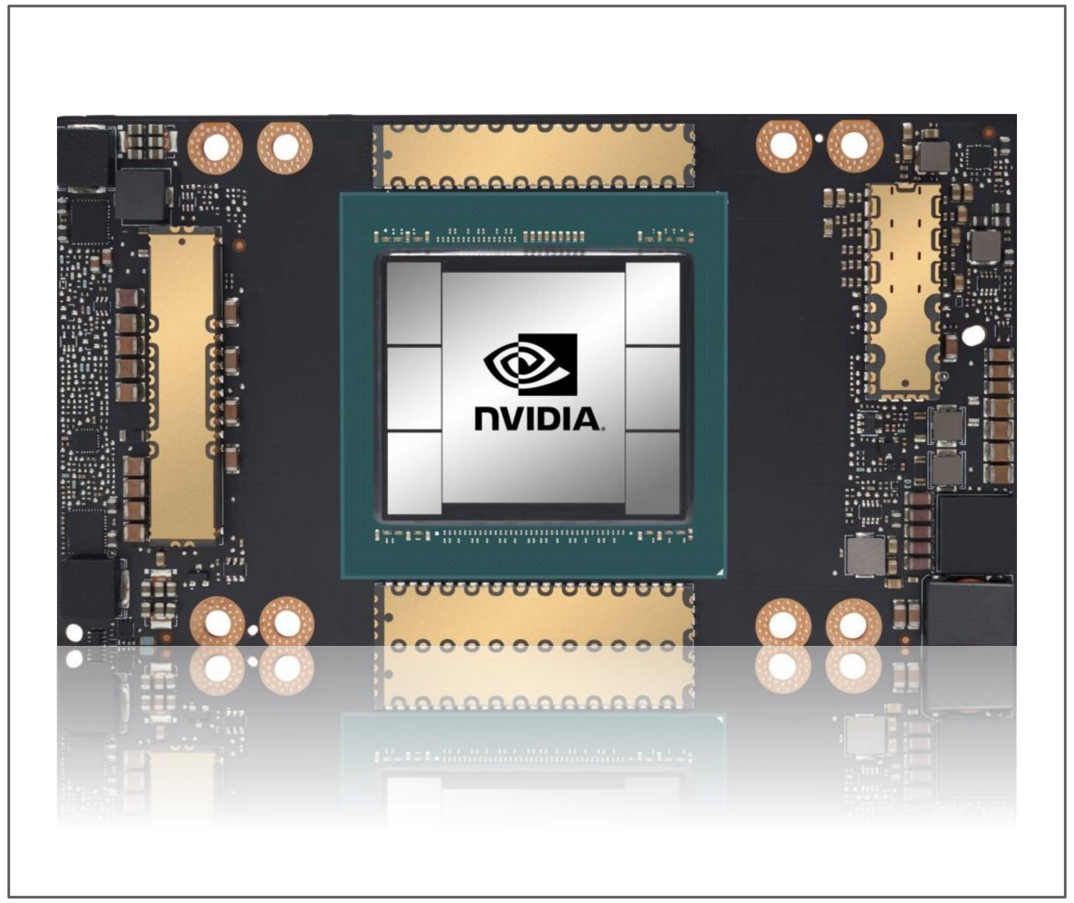
Eventually, the optimizer might be able to do everything for you

WHAT YOU NEED TO MAKE DEEP LEARNING WORK

You need three main ingredients (and some skill)

LARGE QUANTITIES OF DATA

ML FRAMEWORK



GPU ACCELERATOR

DEEP LEARNING FRAMEWORKS

Many frameworks to choose from (but not for Fortran)

PyTorch

K + TensorFlow

Python

mxnet

GLUON

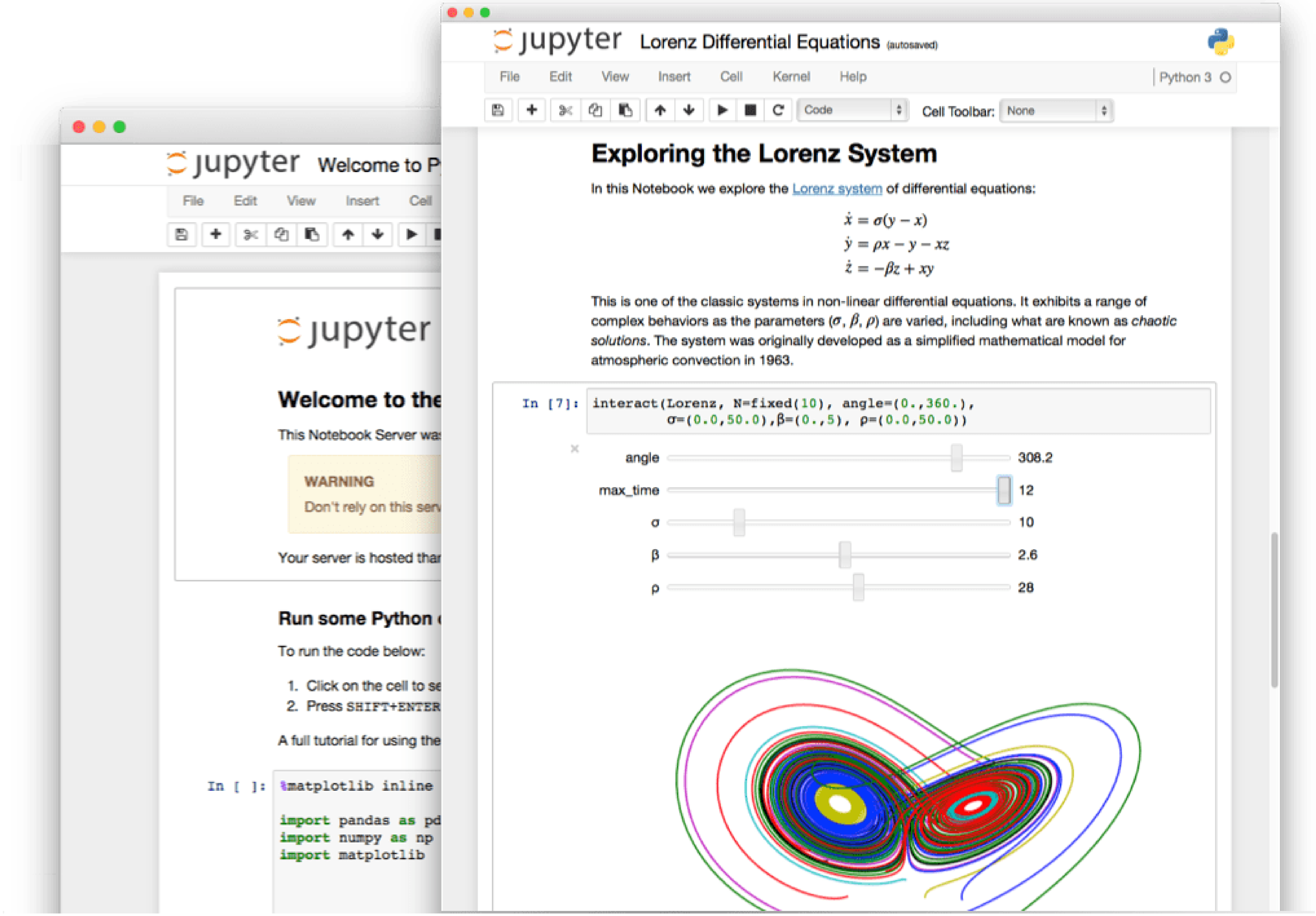
C++

flux

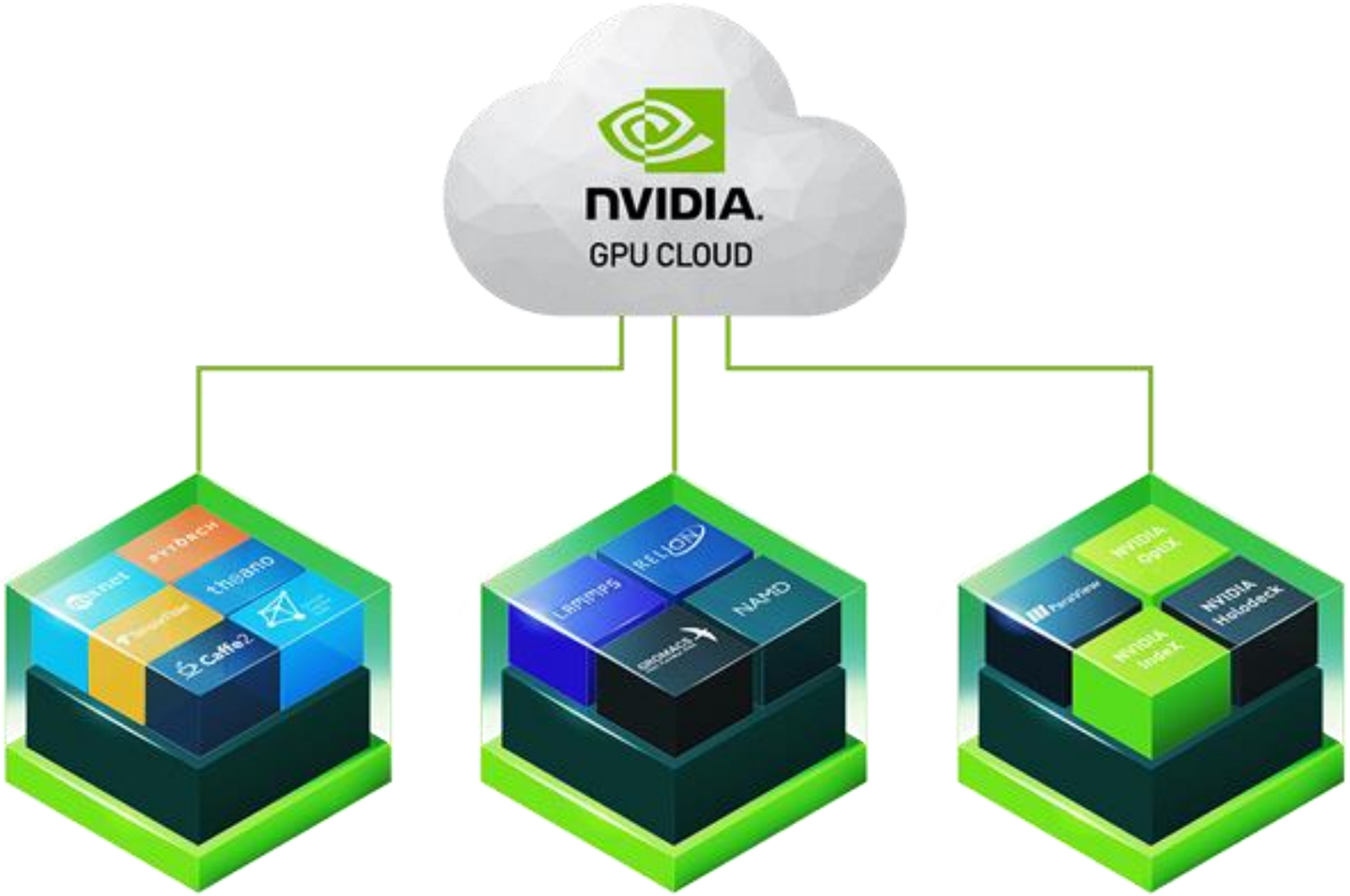
Julia

Chainer mxnet Caffe2 Microsoft CNTK TensorFlow Keras GLUON PYTORCH theano

DEVELOPMENT ENVIRONMENT
JUPYTER NOTEBOOKS



NVIDIA GPU CLOUD REGISTRY
CONTAINERIZED SOFTWARE



DEEP LEARNING

HPC APPS

HPC VISUALIZATION

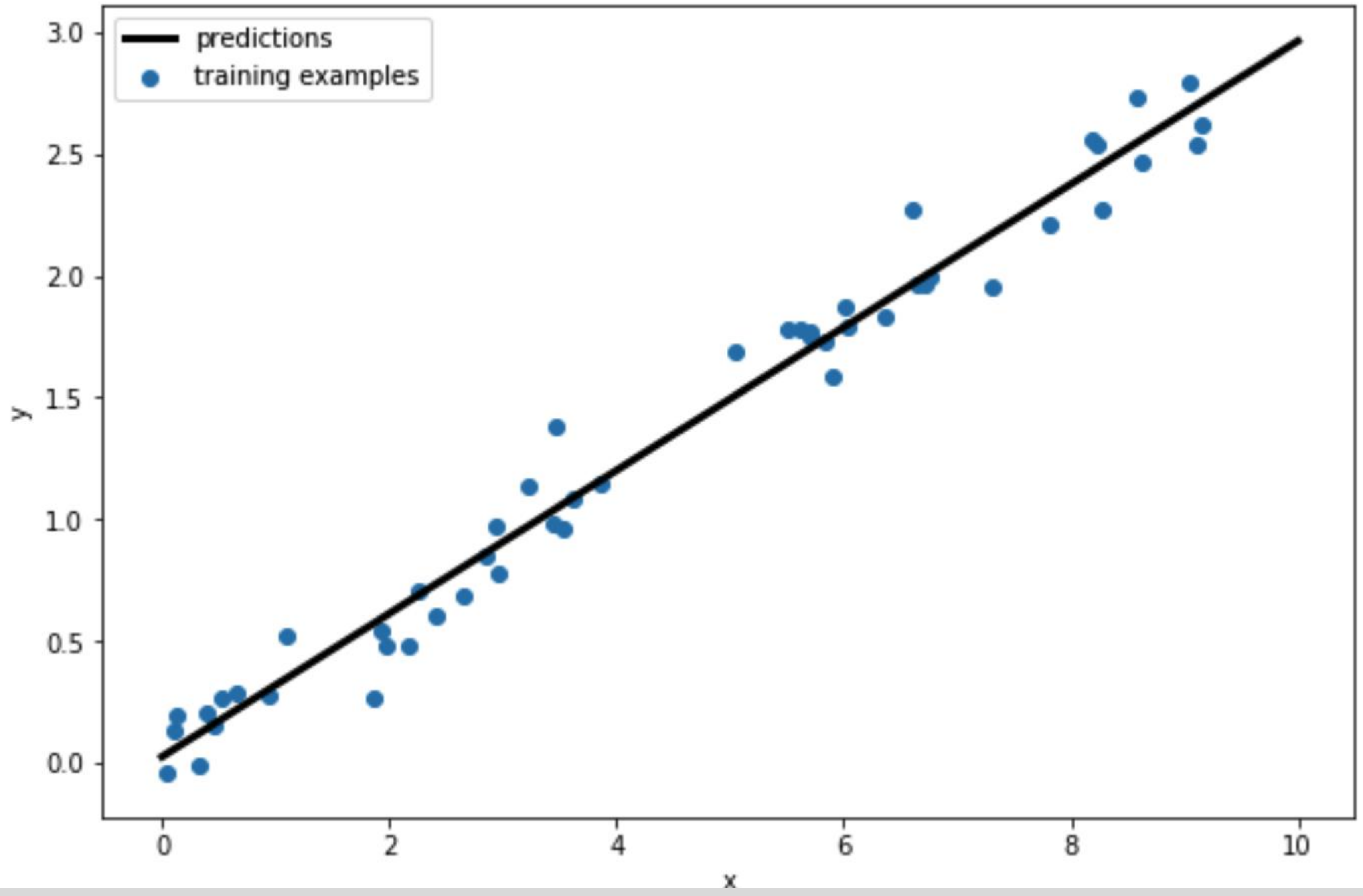
docker

Singularity

LINEAR REGRESSION

With Scikit Learn

```python
from sklearn.linear_model import LinearRegression
from numpy.random import rand
import numpy as np

# DATA
npts    = 50
x_train = 10*rand(npts,1)
y_train = 0.3*x_train + 0.5 * rand(npts,1)

# MODEL
model = LinearRegression()

# TRAIN
model.fit(x_train, y_train)

# TEST
x_test    = np.linspace(0,10,npts).reshape(-1,1)
y_predict = model.predict(x_test)
```



LINEAR REGRESSION

With TensorFlow and Keras

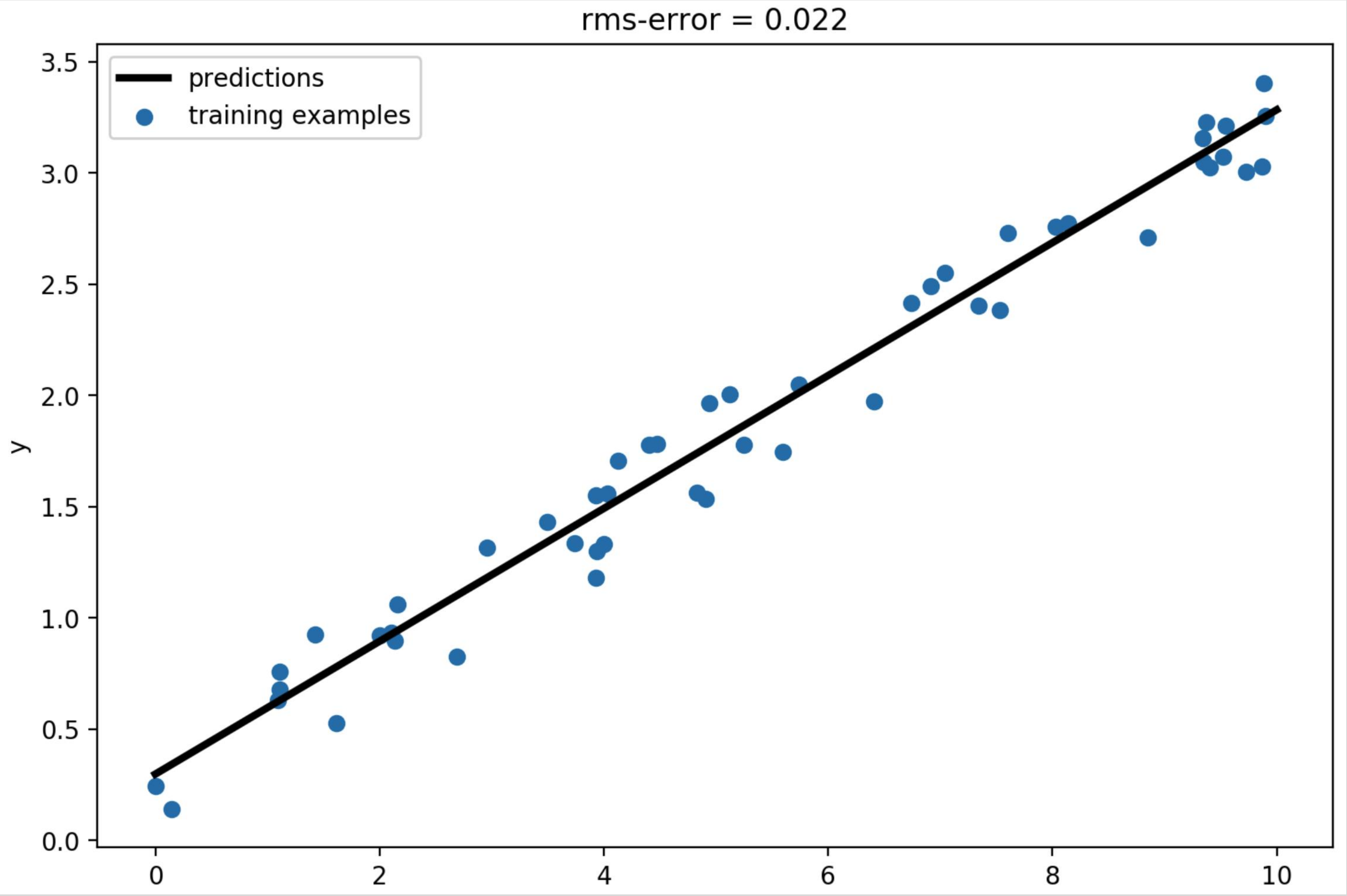
```
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import backend as K
from tensorflow.random import uniform
K.clear_session()
```

```
# DATA
npts     = 50
x_train = 10*uniform(shape=[npts,1])
y_train = 0.3*x_train + 0.5 * uniform(shape=[npts,1])
```

```
# MODEL
model = keras.models.Sequential()
model.add(keras.layers.Dense(1, input_shape=[1]))
optimizer = keras.optimizers.Adam(lr=1e-1)
model.compile(loss='mean_squared_error',optimizer=optimizer)
```

```
# TRAIN
hist = model.fit(x_train,y_train,epochs=100,verbose=0)
```

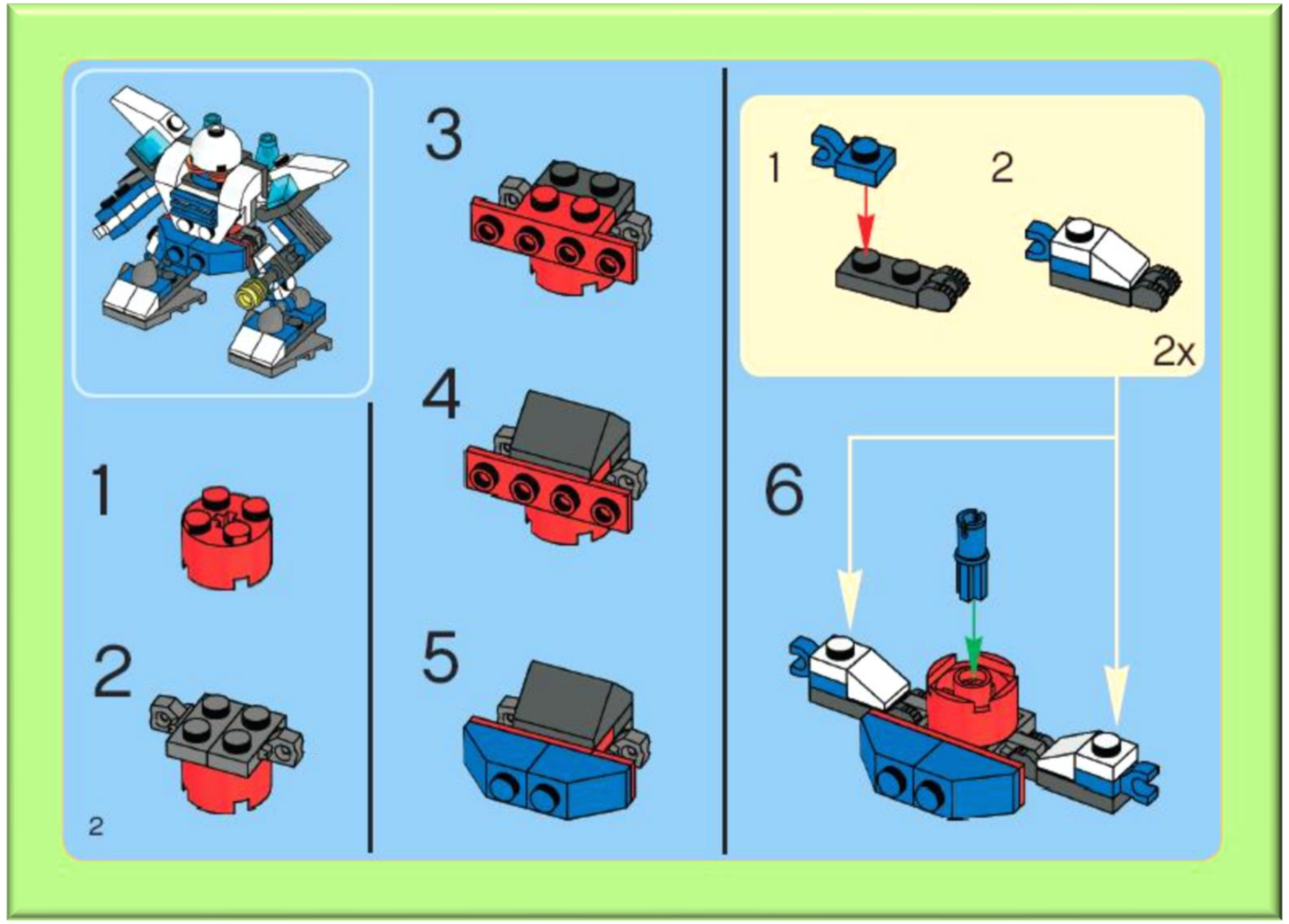
```
# TEST
x_test    = tf.linspace(0,10,npts)
x_test    = tf.reshape(x_test,[npts,1])
y_predict = model.predict(x_test)
```



TRAINING

TRAINING VS INFERENCE

TRAINING PHASE



SEARCH FOR THE RIGHT PIECES

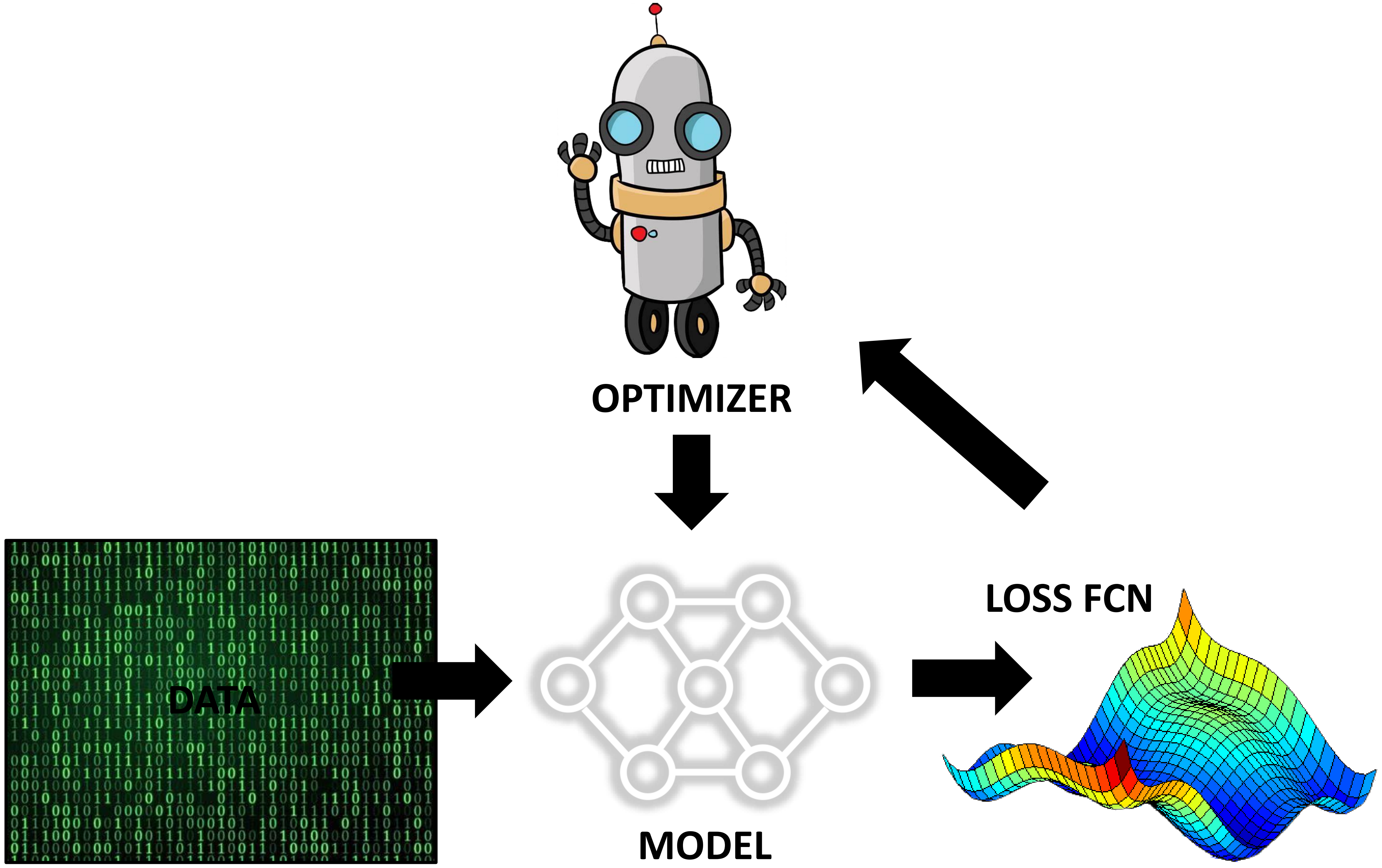
ONLINE
LEARNING

INFERENCE PHASE

APPLY THE COMPLETED MODEL

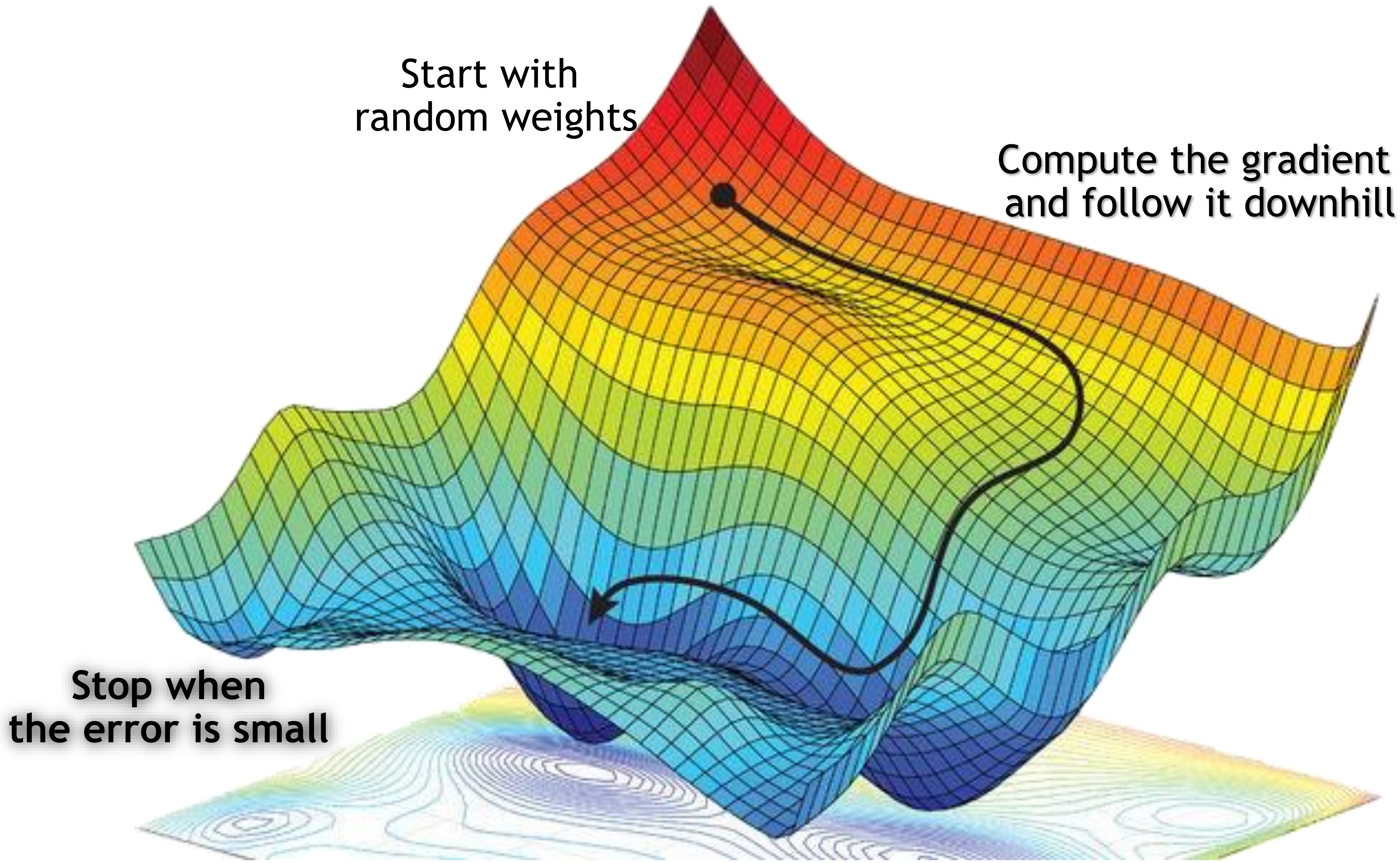
TRAINING: THE PLAYERS

DATA, MODEL, LOSS, AND OPTIMIZER



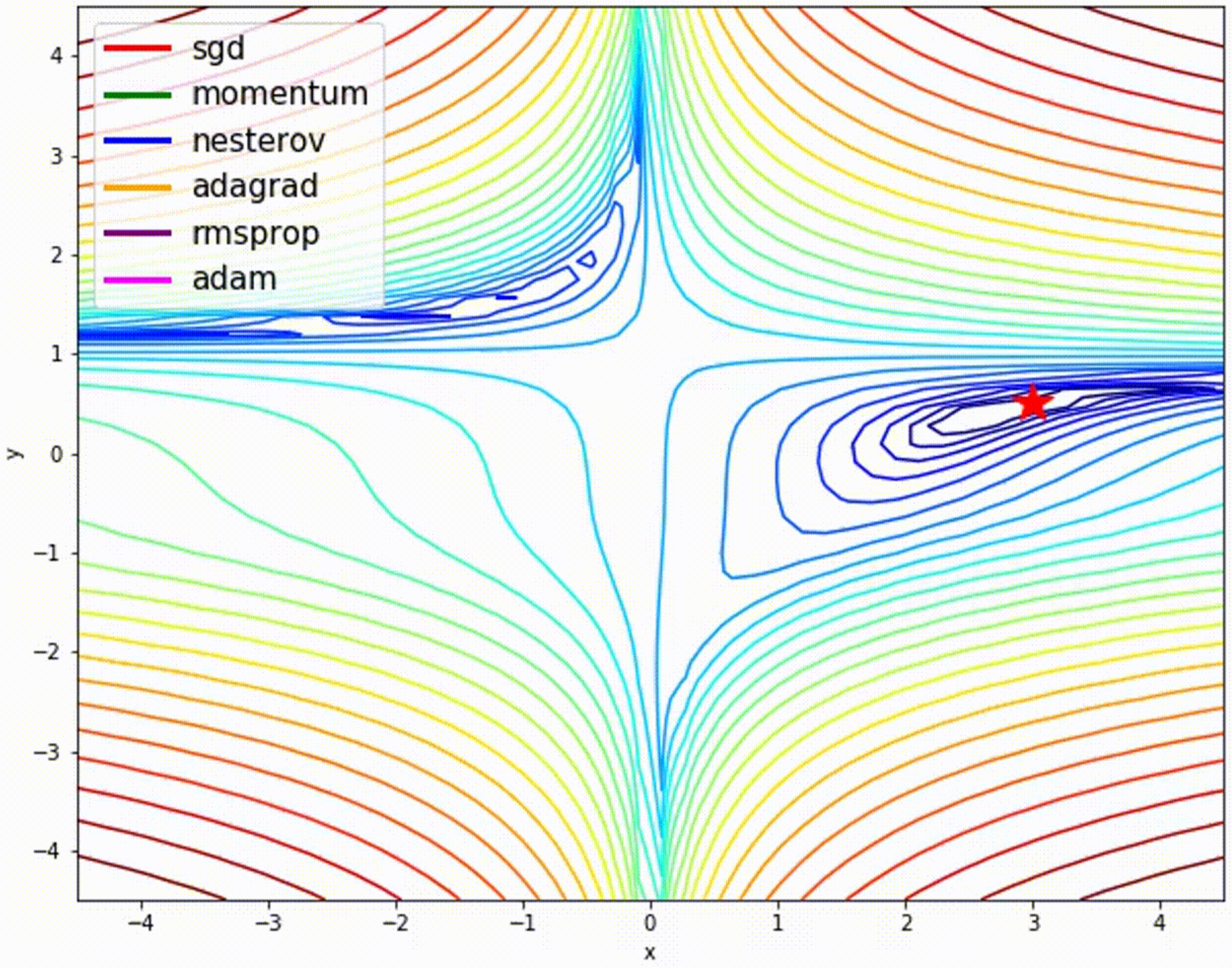
TRAINING: GRADIENT DESCENT

Finding as solution is as easy as falling down a hill



OPTIMIZERS

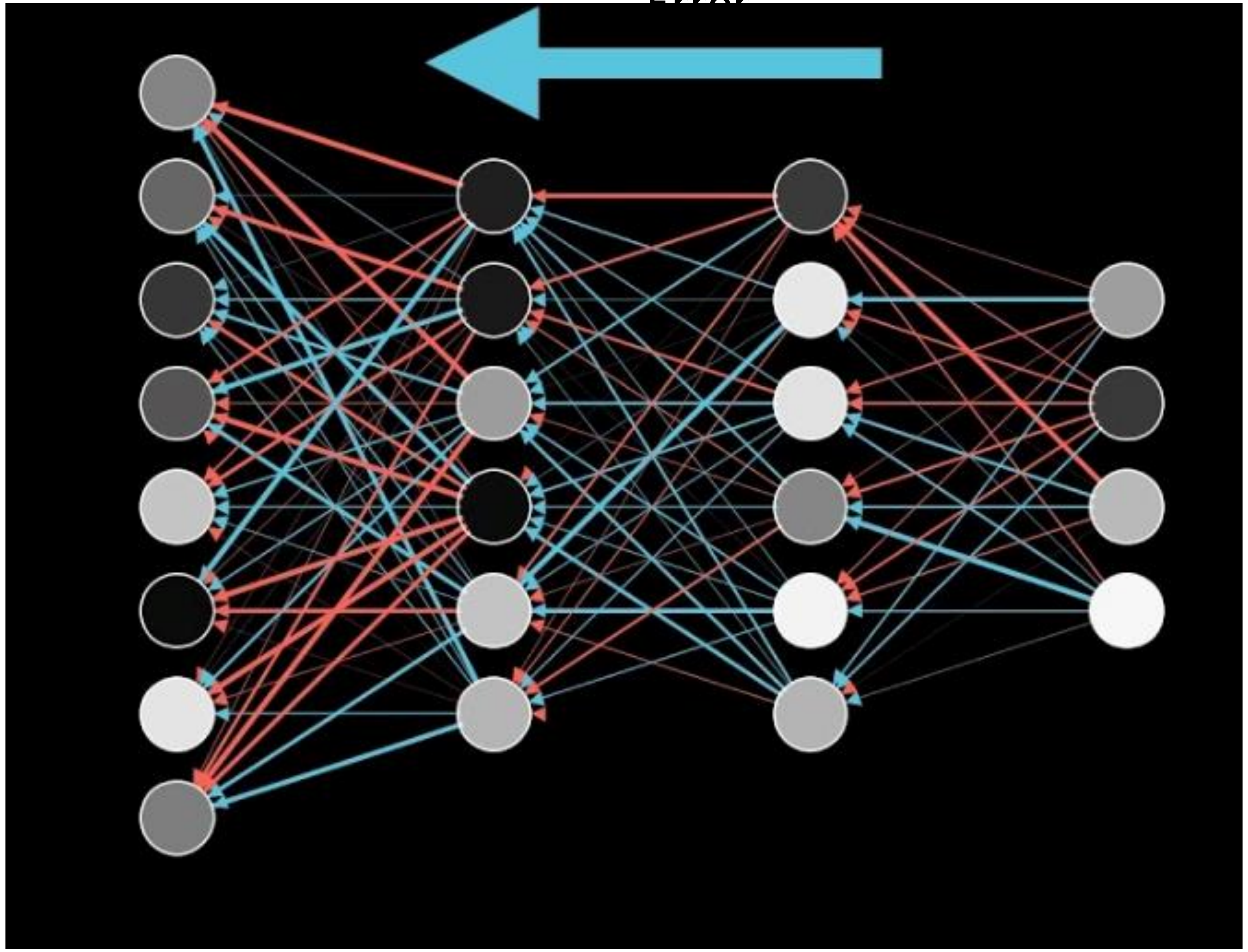
Many variations on stochastic gradient descent



TRAINING: BACKPROPAGATION

Compute the gradient, by efficiently assigning blame

Error

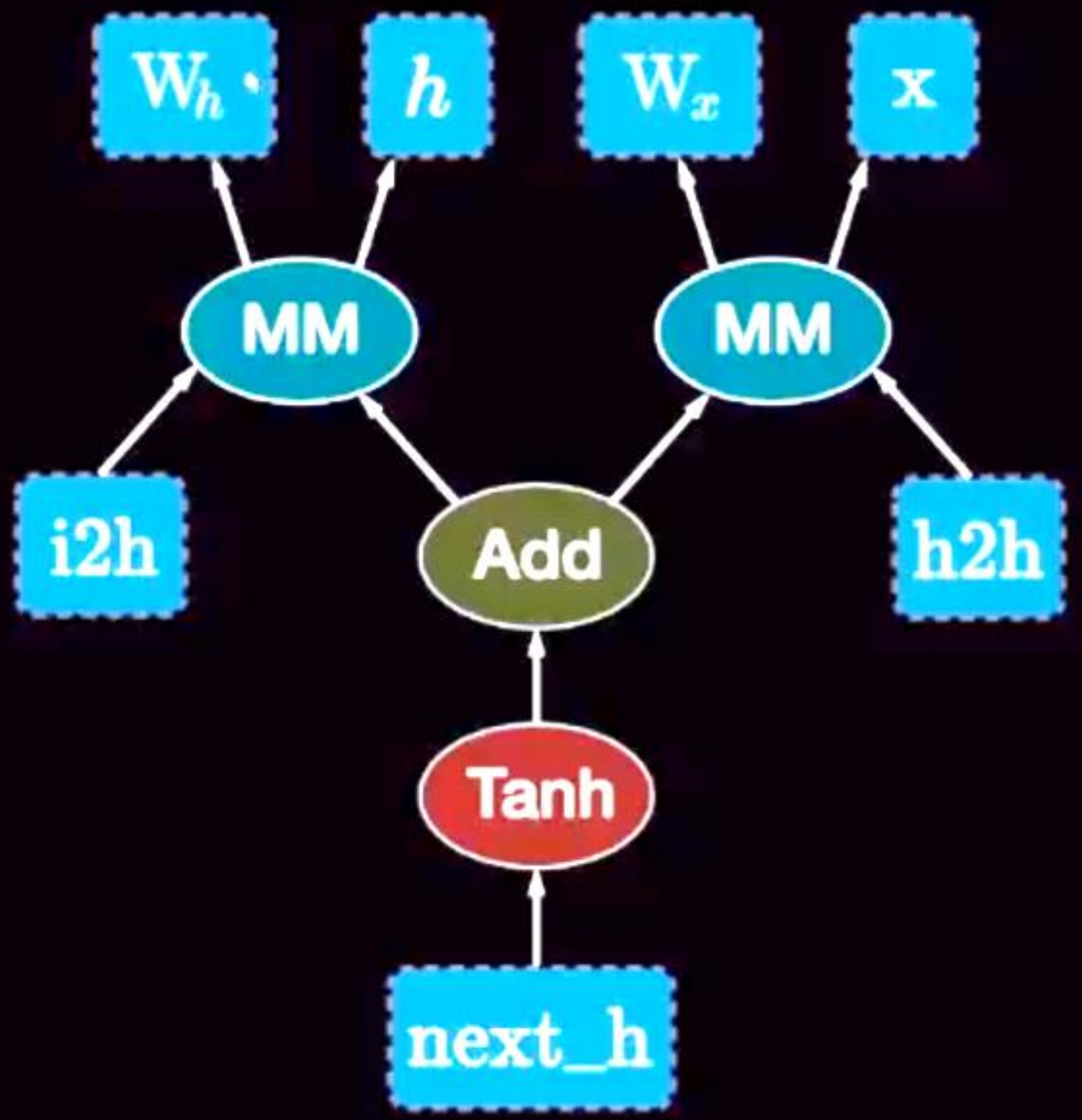


AUTOGRAD

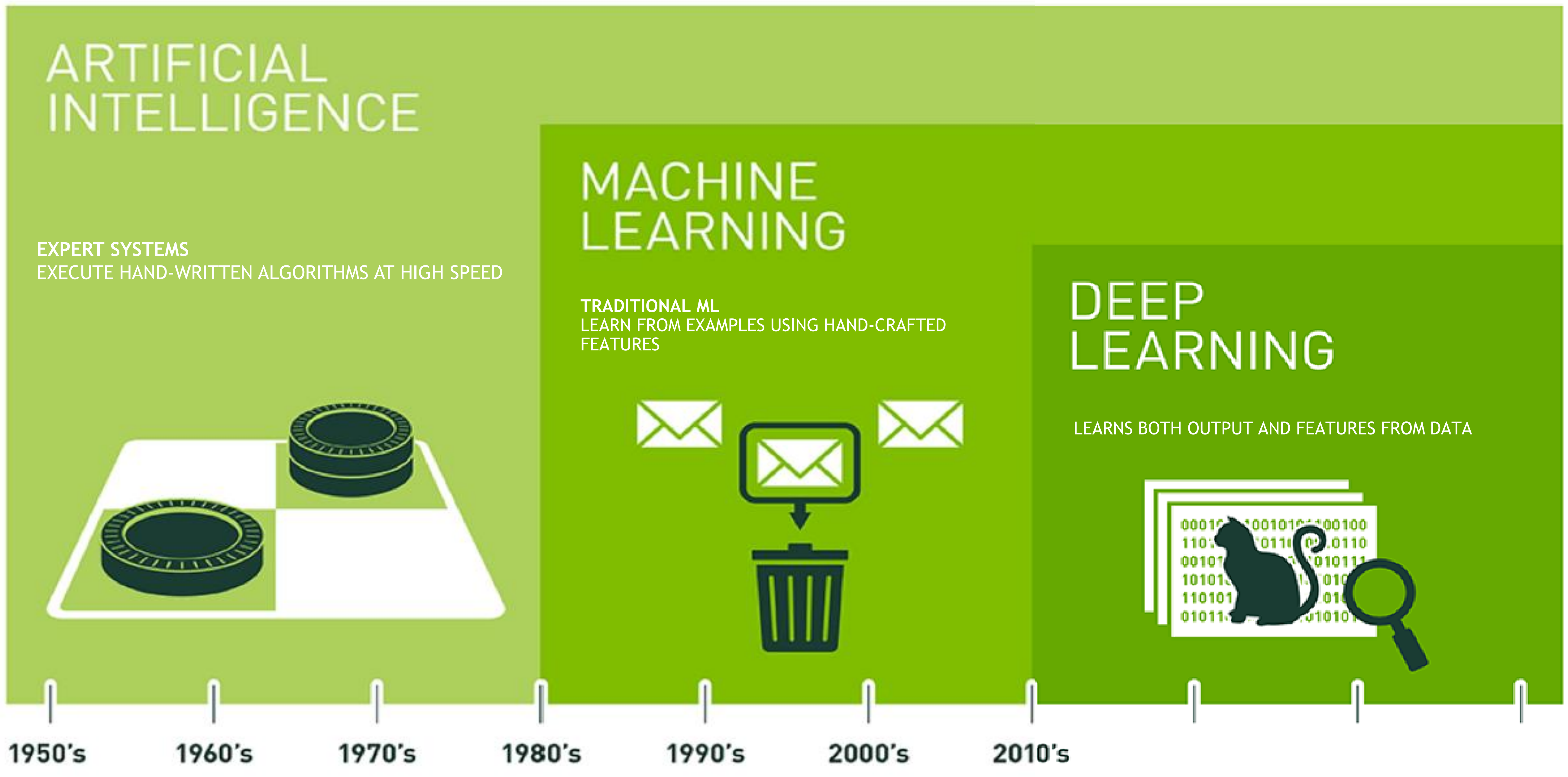
Let a framework keep track of your gradient, so you don’t have to

PyTorch Autograd

```
from torch.autograd import Variable

x = Variable(torch.randn(1, 10))
prev_h = Variable(torch.randn(1, 20))
W_h = Variable(torch.randn(20, 20))
W_x = Variable(torch.randn(20, 10))

i2h = torch.mm(W_x, x.t())
h2h = torch.mm(W_h, prev_h.t())
next_h = i2h + h2h
next_h = next_h.tanh()

next_h.backward(torch.ones(1, 20))
```

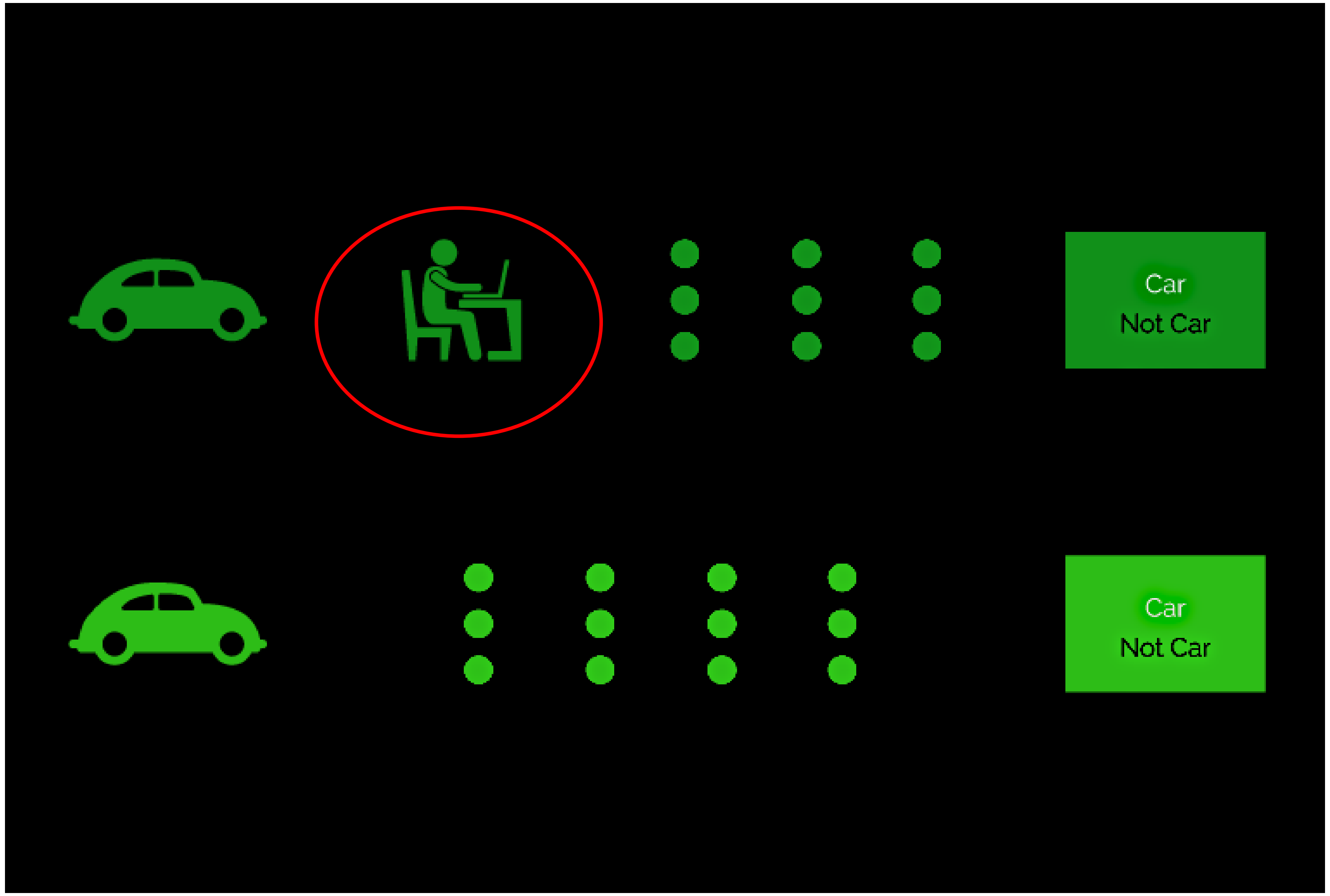


AI, MACHINE LEARNING, DEEP LEARNING



DEEP LEARNING VS. MACHINE LEARNING

When should I use deep learning vs traditional machine learning?



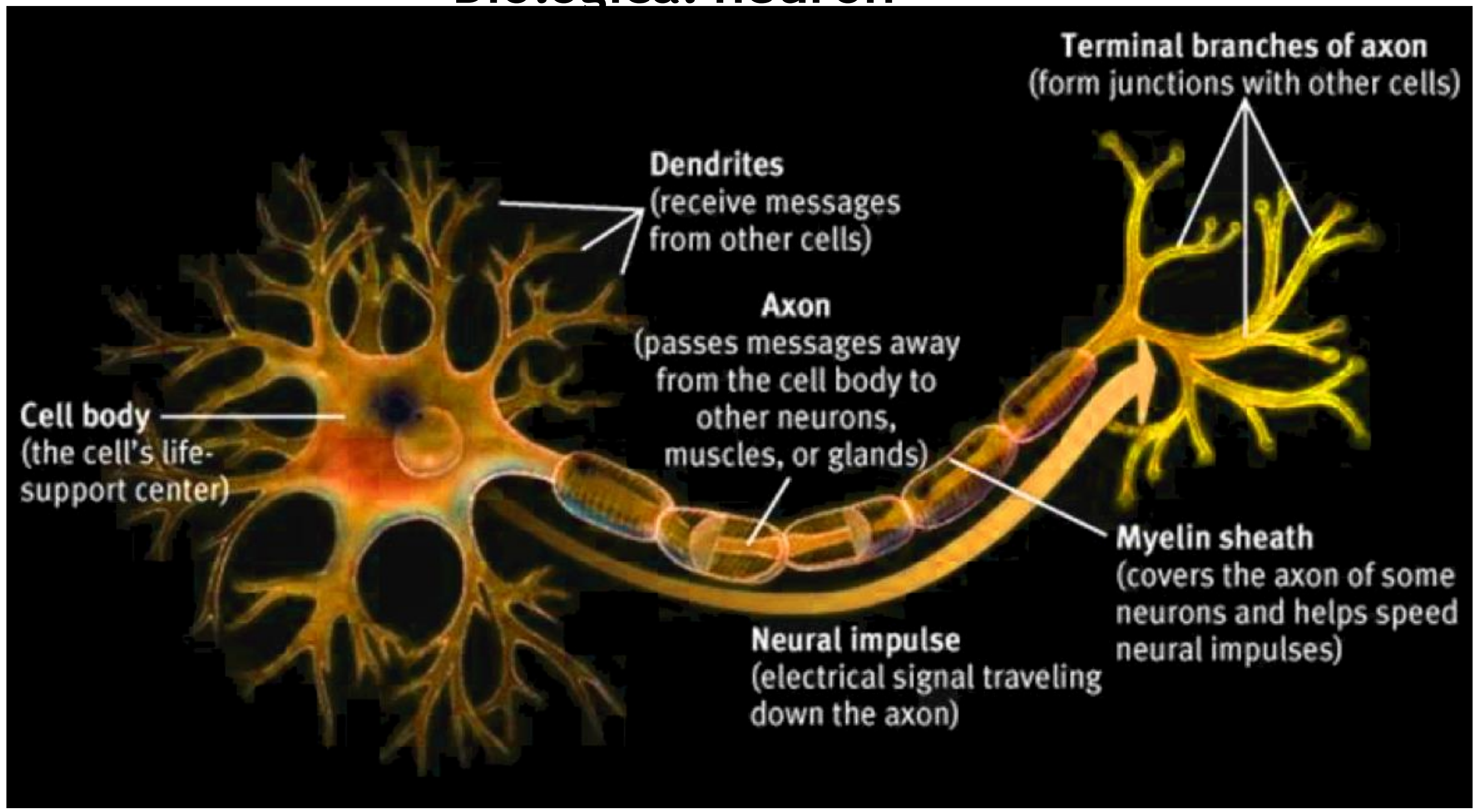
TRADITIONAL MACHINE LEARNING
Random forests, SVM, K-means, Logistic Regression
Features hand-crafted by experts
Small set of features: 10s or 100s
NVIDIA RAPIDS: orders of magnitude speedup

SUPERVISED DEEP LEARNING
CNN, RNN, LSTM, GAN, Variational Auto-encoders
Finds features automatically
High dimensional data: images, sounds, speech
Large set of labelled data (10k+ examples)
NVIDIA CU-DNN: accelerates DL frameworks

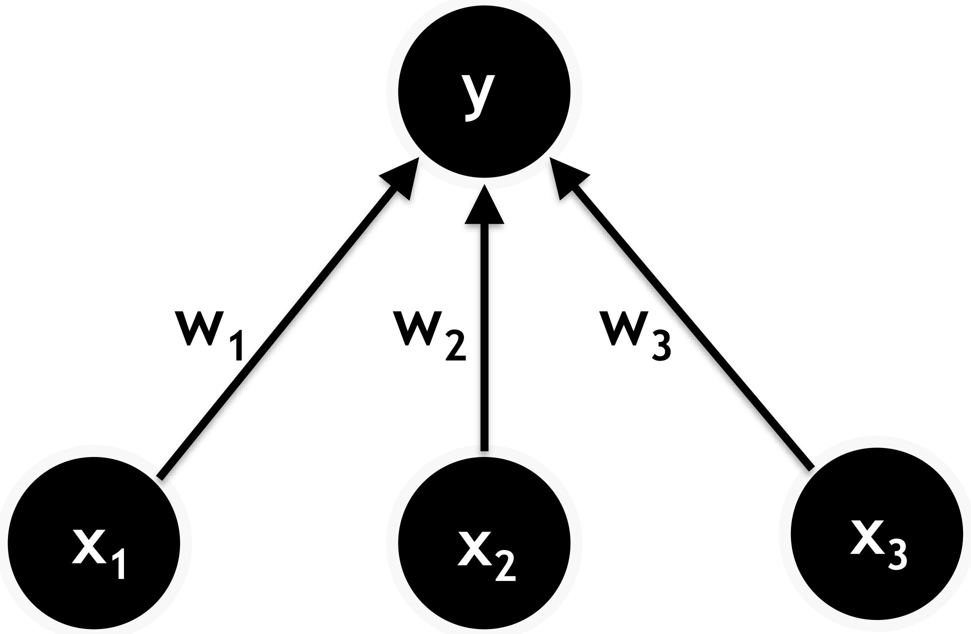
ARTIFICIAL NEURONS

Simple equations with adjustable parameters

Biological neuron



Artificial neuron

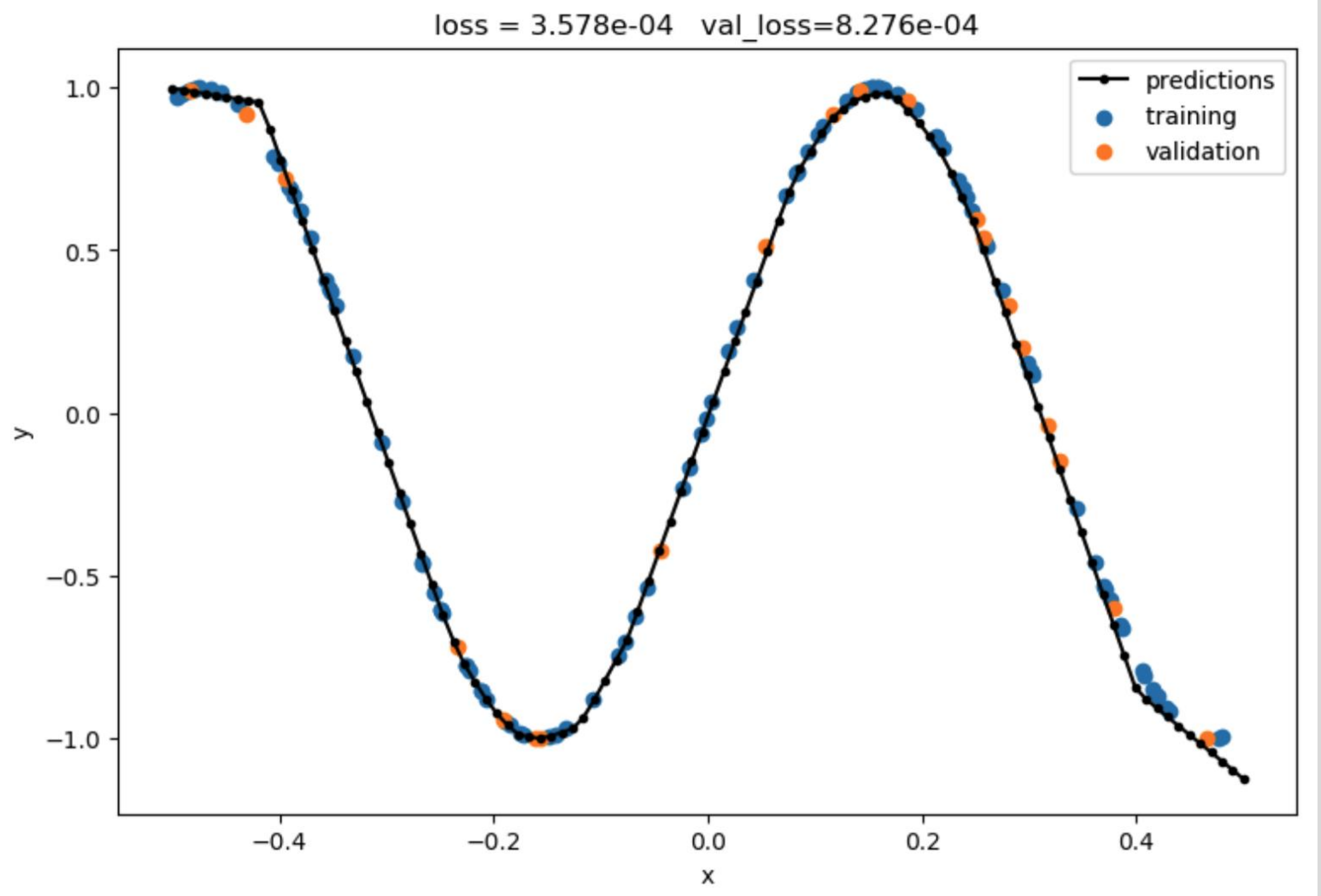


$$y = f(w_1x_1 + w_2x_2 + w_3x_3)$$

https://towardsdatascience.com/the-differences-between-artificial-and-biological-neural-networks-a8b46db828b7

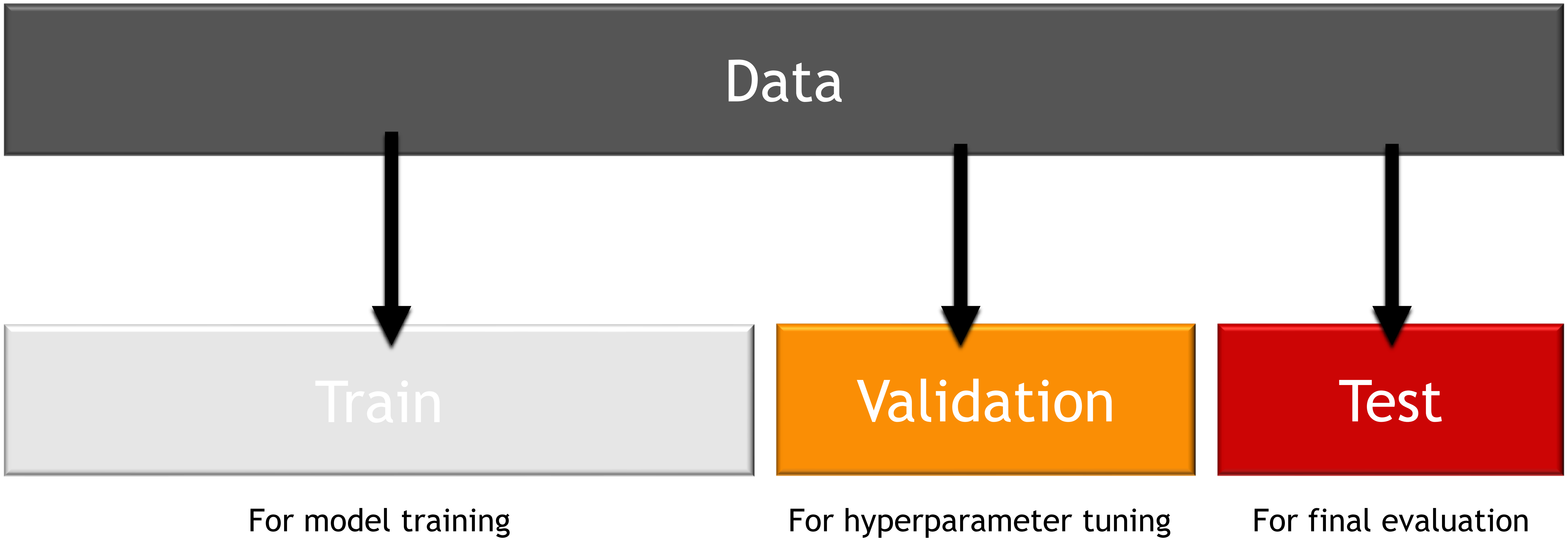
CURVE FIT WITH SINGLE LAYER NEURAL NETWORK

```python
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import backend as K
from tensorflow.random import uniform
K.clear_session()

# DATA
def f(x): return tf.math.sin(10*x)
x_train = uniform(shape=[100,1]) - 0.5
x_val   = uniform(shape=[ 20,1]) - 0.5
y_train = f(x_train)
y_val   = f(x_val)

# MODEL (using keras functional API)
inputs    = keras.Input(shape=[1])
x         = keras.layers.Dense(units=100,activation="relu")(inputs)
x         = keras.layers.Dense(1)(x)
outputs   = keras.layers.Add()([inputs,x])
model     = keras.Model(inputs=inputs, outputs=outputs)
optimizer= keras.optimizers.Adam(lr=1e-2)
model.compile(loss='mean_squared_error',optimizer=optimizer)

# TRAIN
hist = model.fit(x_train,y_train,validation_data=(x_val,y_val),
                 epochs=5000,verbose=0, batch_size=1)
# TEST
x_test    = tf.linspace(-.5,.5,100)
y_predict = model.predict(x_test)
```



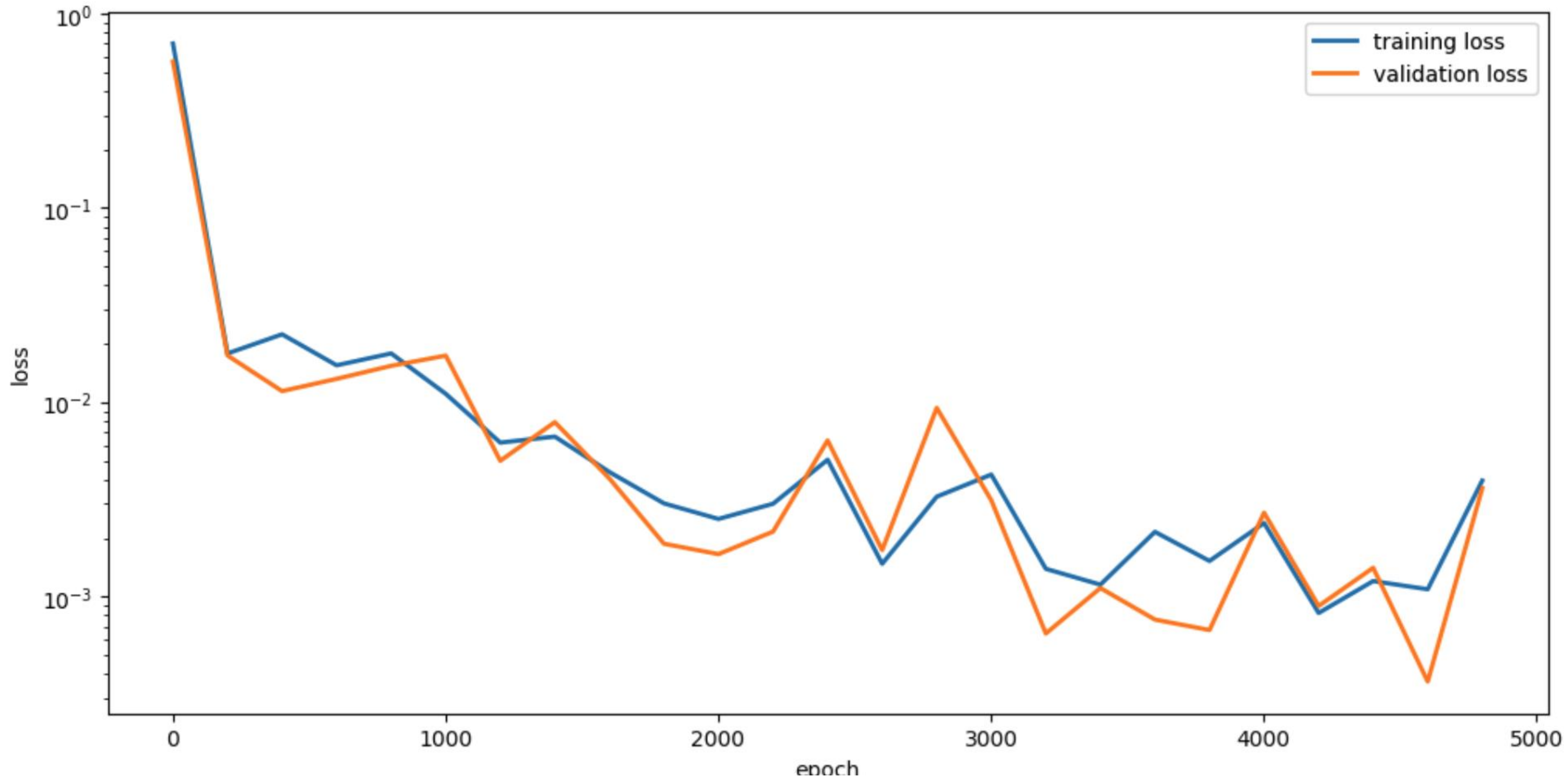
DATA SPLITTING

KEEP TEST, TRAINING, AND VALIDATION DATA SEPERATE

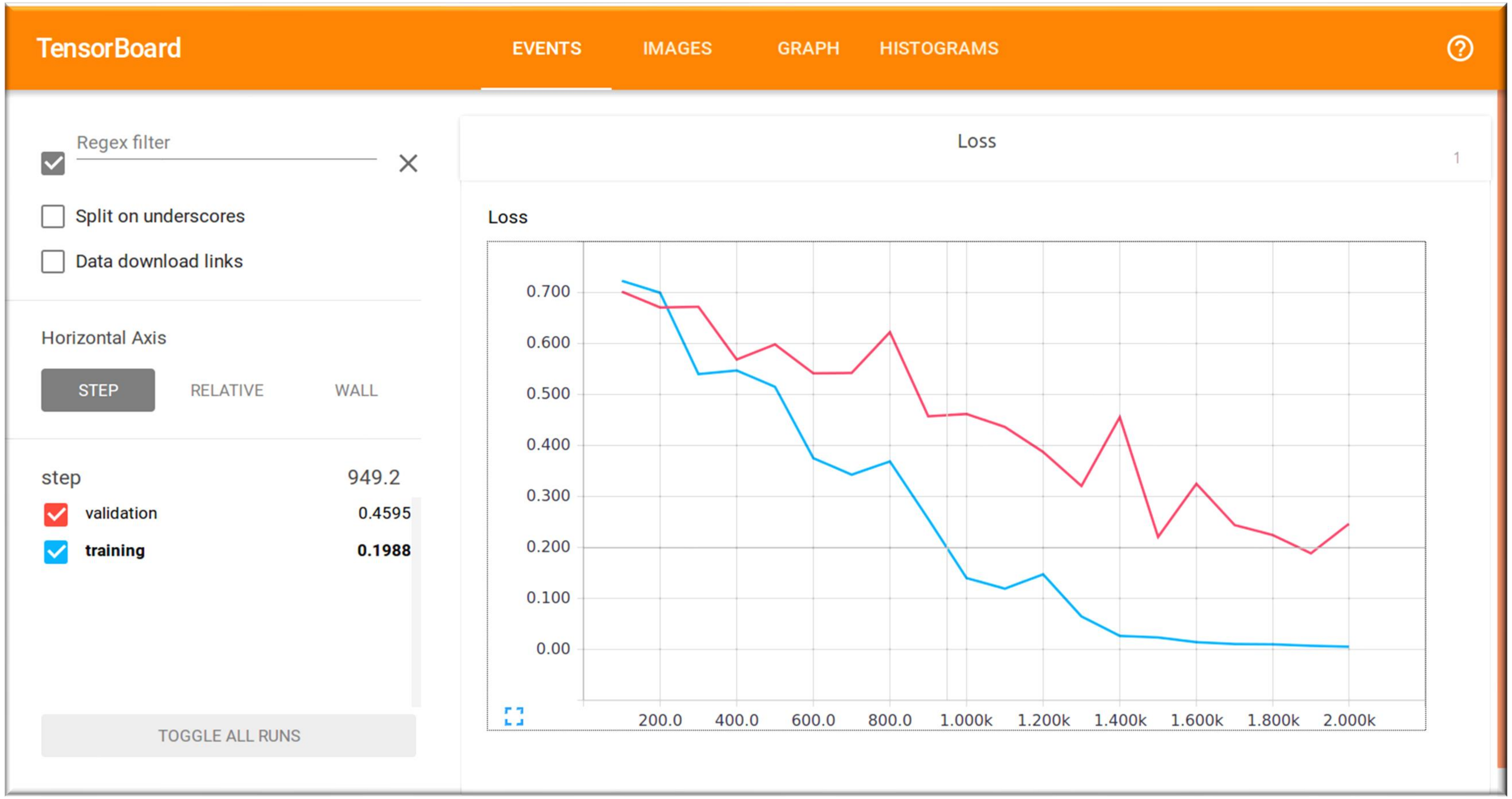


PLOTTING TRAINING AND VALIDATION LOSS

```
%matplotlib inline
import matplotlib.pyplot as plt

t_loss = hist.history['loss']
v_loss = hist.history['val_loss']
epoch  = tf.range(1,len(t_loss)+1)
step   = 200

plt.figure(figsize=(12,6),dpi=100)
plt.semilogy(epoch[::step],t_loss[::step],linewidth=2,label='training loss')
plt.semilogy(epoch[::step],v_loss[::step],linewidth=2,label='validation loss')
plt.xlabel('epoch'); plt.ylabel('loss'); plt.legend()
```



VISUALIZATION TOOLS: TENSORBOARD



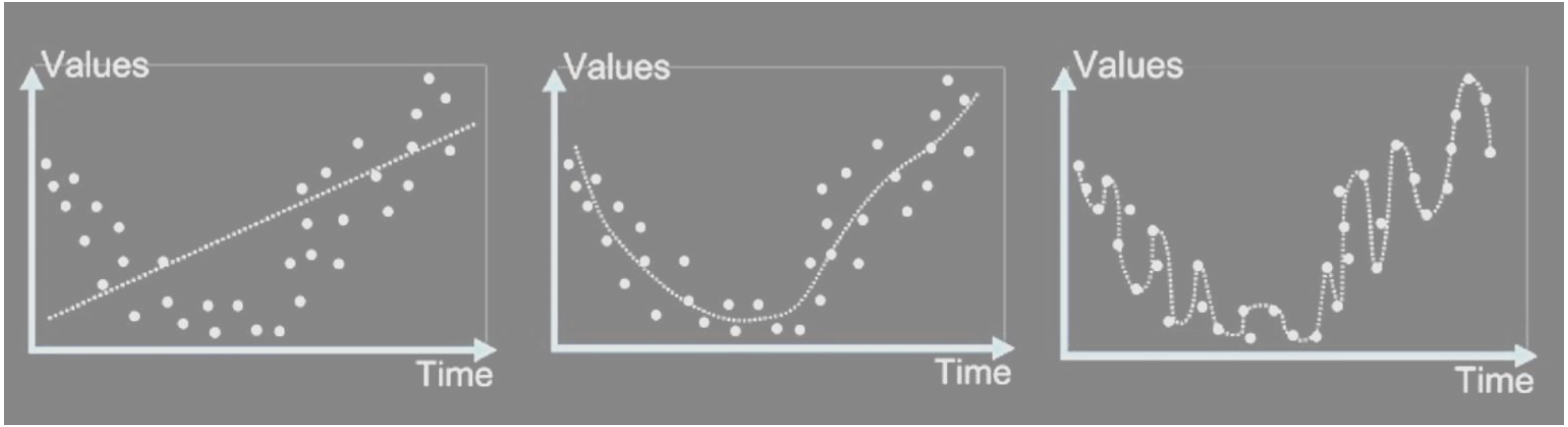

```
tensorboard_callback = tf.keras.callbacks.TensorBoard(log_dir="./logs")
model.fit(x_train, y_train, epochs=2, callbacks=[tensorboard_callback])
# run the tensorboard command to view the visualizations.
```

```
tensorboard --logdir=path_to_your_logs
```

MODEL CAPACITY AND REGULARIZATION

MODEL CAPACITY

A good model is one that generalizes to new data



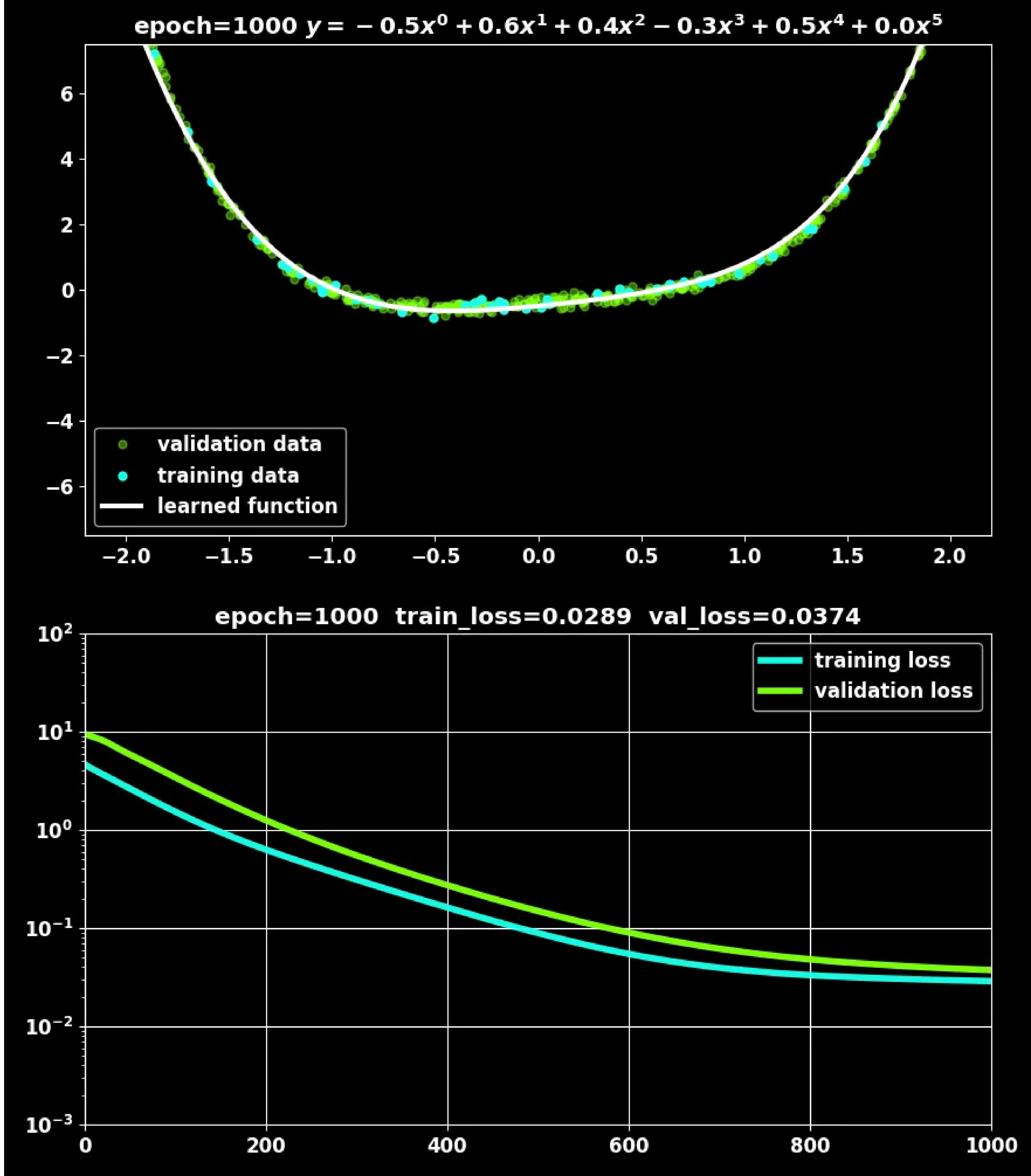
UNDERFIT **GOOD FIT** **OVER FIT**

GOOD-FIT

Checking for Generalization

Training Loop

```
xtrain, ytrain, xval, yval = load_data(npts=500, train_fraction = 0.10)

model     = taylor_series(order=5)
optimizer = torch.optim.Adam(model.params, lr=1.0e-2)
epochs    = 1000

for i in range(epochs+1):

    # training
    optimizer.zero_grad()
    yhat = model(xtrain)
    loss = (yhat - ytrain).pow(2).mean()
    loss.backward()
    optimizer.step()

    # validation
    yval_hat = model(xval)
    loss_val = (yval_hat - yval).pow(2).mean()
```



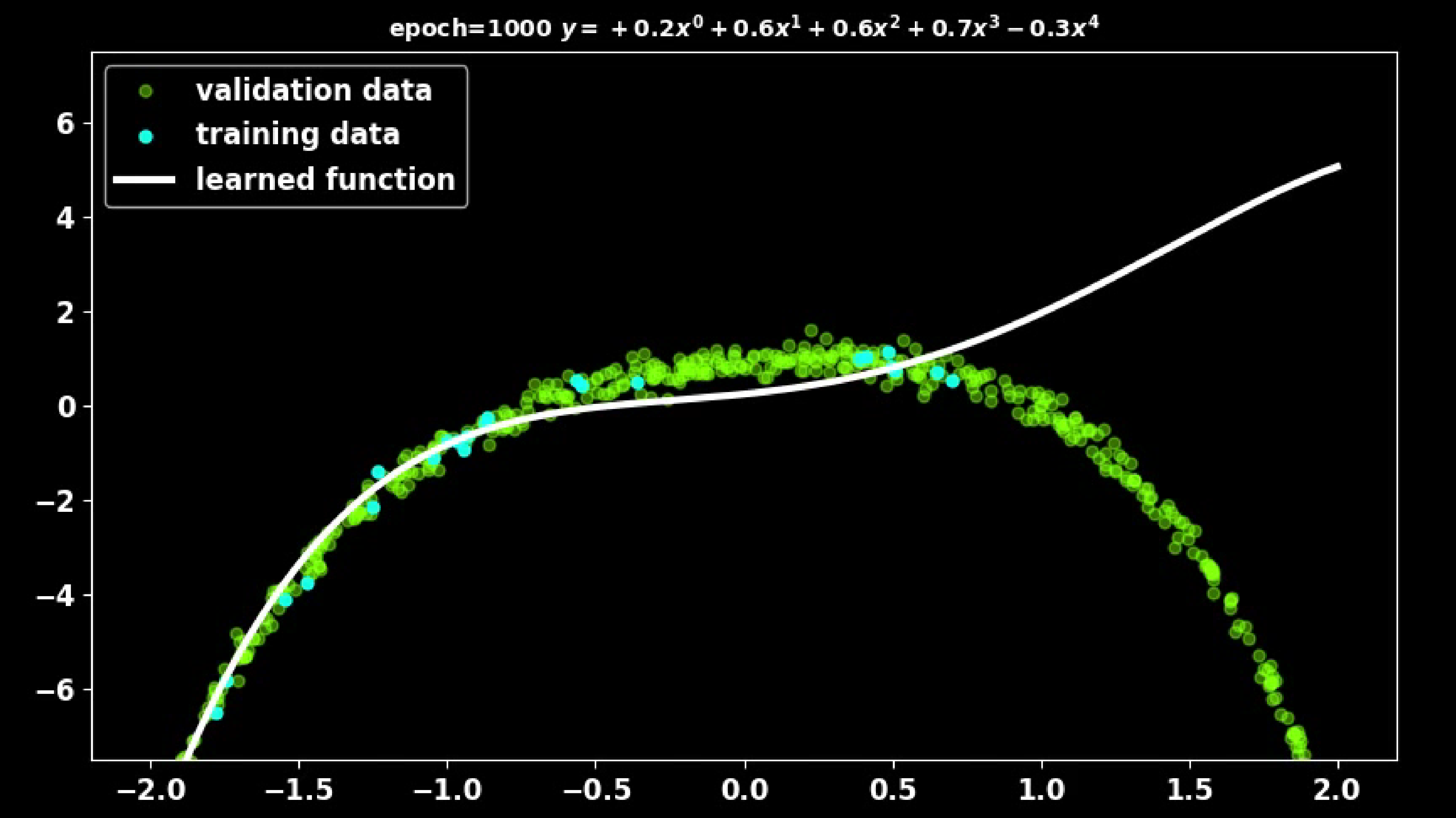
OVER-FITTING

Captures training data, but generalizes poorly

Training Loop

```
xtrain, ytrain, xval, yval = load_data(npts=500, train_fraction = 0.02)

model     = taylor_series(order=8)
optimizer = torch.optim.Adam(model.params, lr=5.0e-3)
epochs    = 1000

for i in range(epochs+1):

    # training
    optimizer.zero_grad()
    yhat = model(xtrain)
    loss = (yhat - ytrain).pow(2).mean()
    loss.backward()
    optimizer.step()

    # validation
    yval_hat = model(xval)
    loss_val = (yval_hat - yval).pow(2).mean()
```

Use more data points
Reduce model capacity



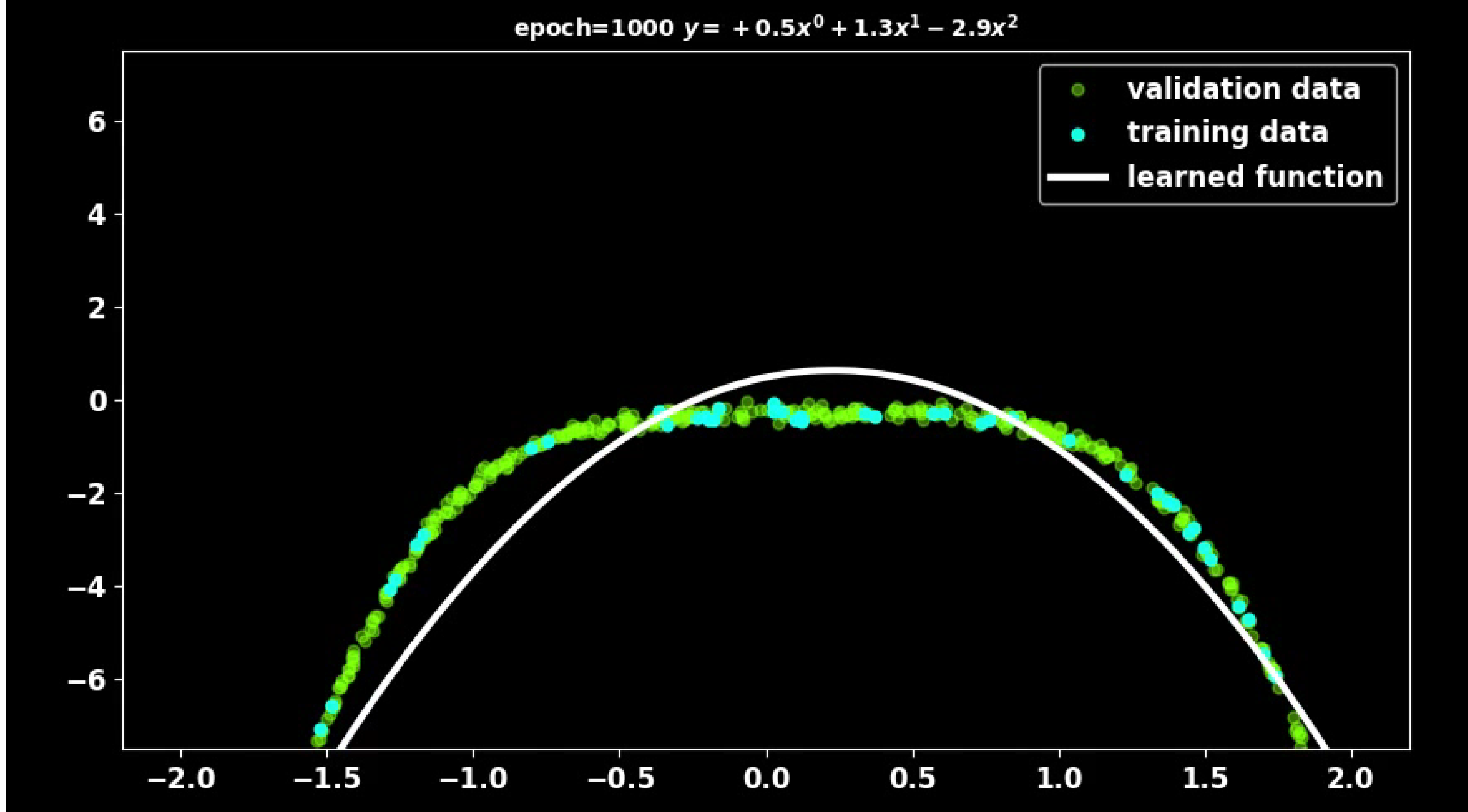
UNDER-FITTING

Model is too simple to fit the curve

Training Loop

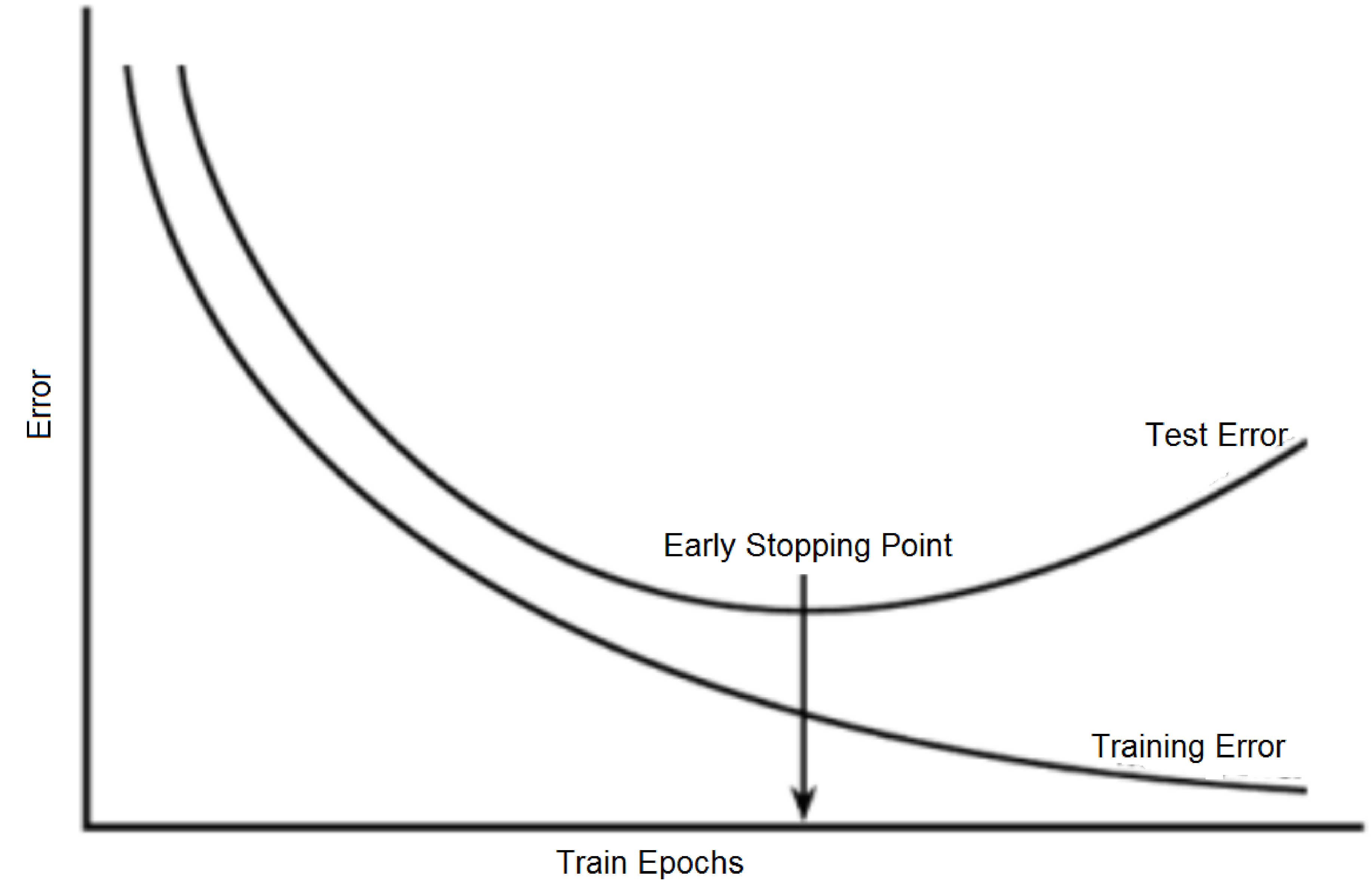
```
xtrain, ytrain, xval, yval = load_data(npts=500, train_fraction = 0.10)

model     = taylor_series(order=2)
optimizer = torch.optim.Adam(model.params, lr=1.0e-2)
epochs    = 1000

for i in range(epochs+1):

    # training
    optimizer.zero_grad()
    yhat = model(xtrain)
    loss = (yhat - ytrain).pow(2).mean()
    loss.backward()
    optimizer.step()

    # validation
    yval_hat = model(xval)
    loss_val = (yval_hat - yval).pow(2).mean()
```

Increase model capacity
Use a different model

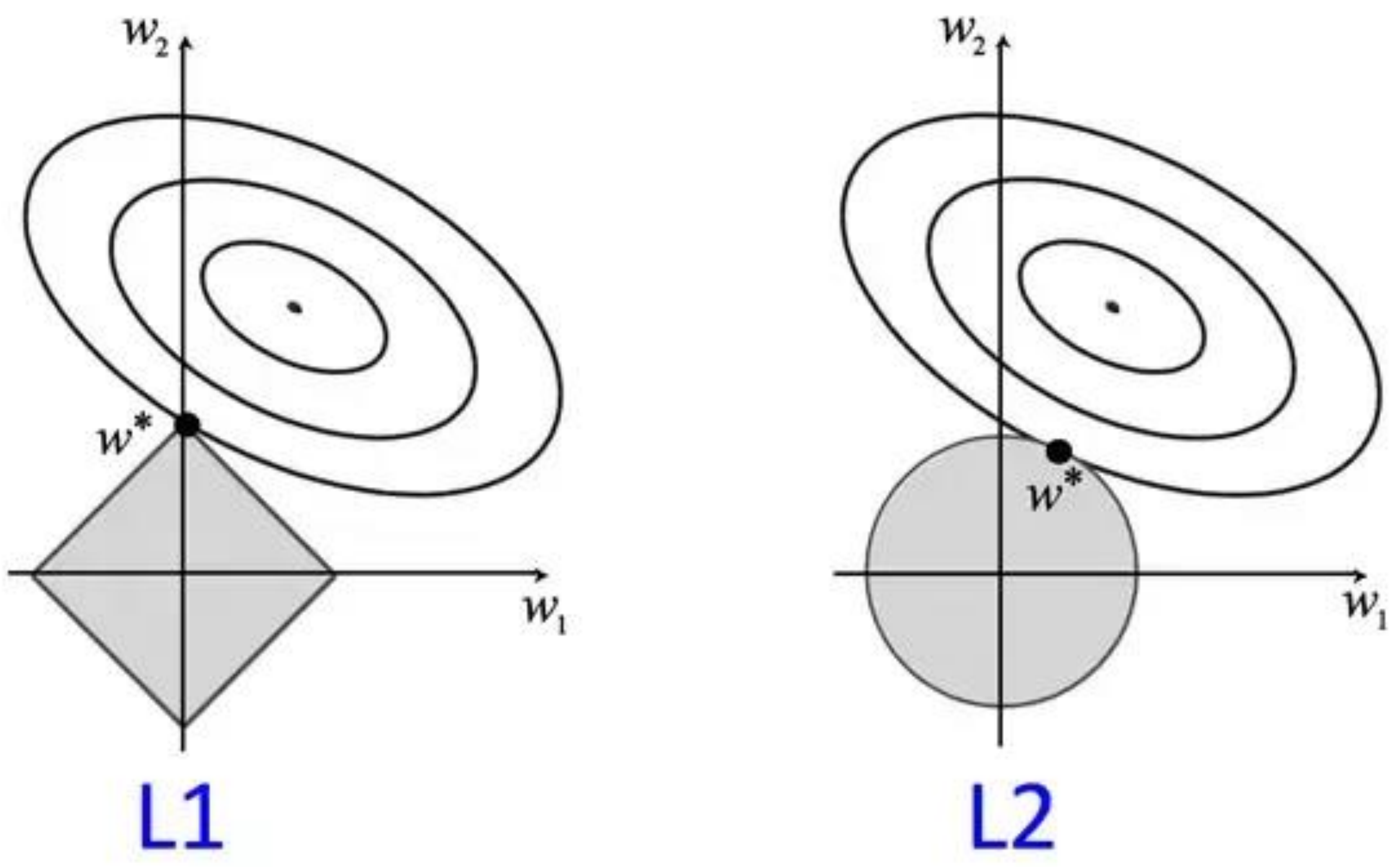


REGULARIZATION

Early Stopping and Layer Regularization



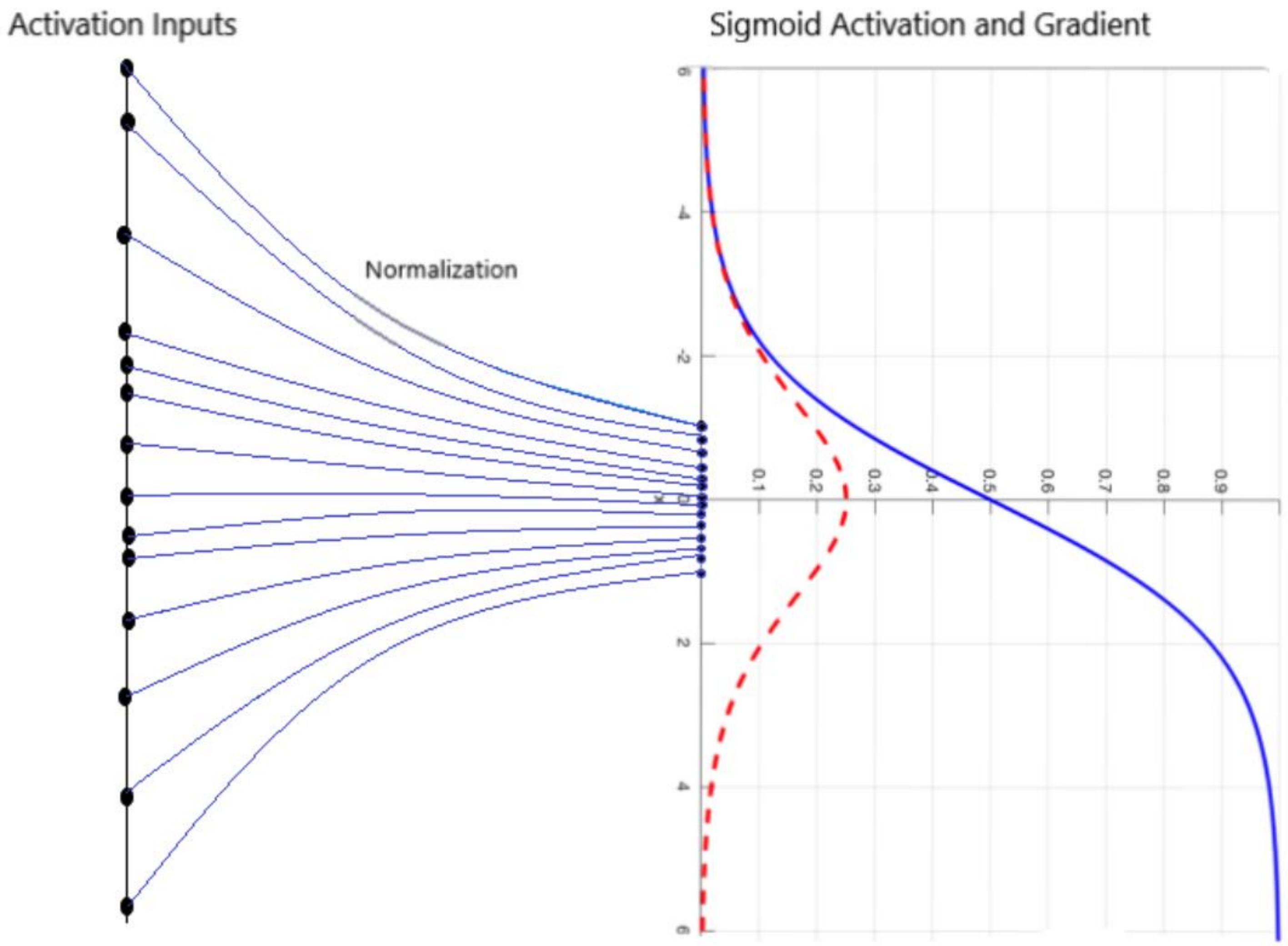

```
callback = tf.keras.callbacks.EarlyStopping(monitor='loss', patience=3)
```



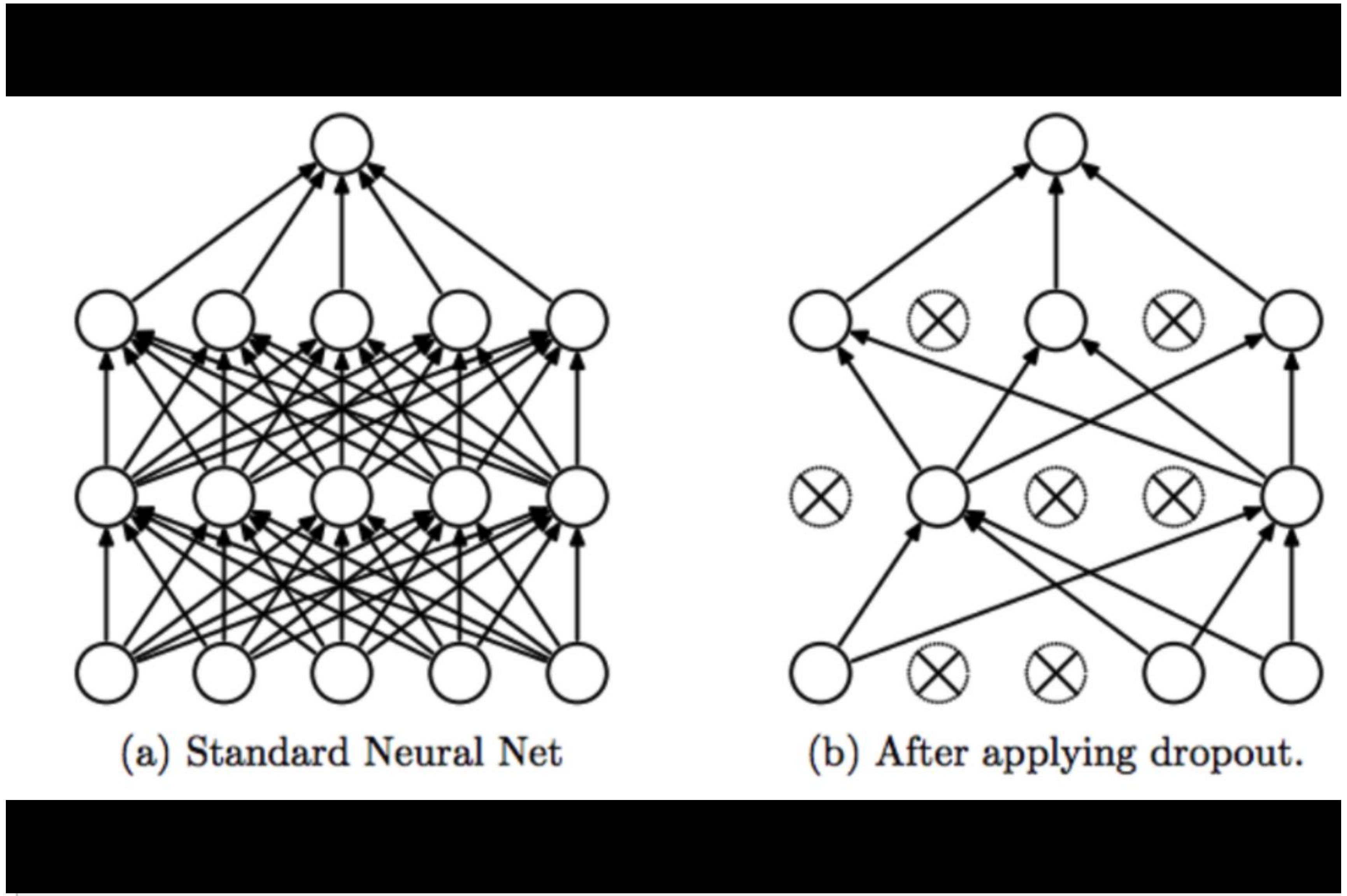

```
layer = tf.keras.layers.Dense(
    5, input_dim=5,
    kernel_initializer='ones',
    kernel_regularizer=tf.keras.regularizers.L1(0.01),
    activity_regularizer=tf.keras.regularizers.L2(0.01))
```

REGULARIZATION

BatchNorm and Dropout




```
model.add(Conv2D(60,3, padding = "same"))
model.add(BatchNormalization())
model.add(Activation("relu"))
```

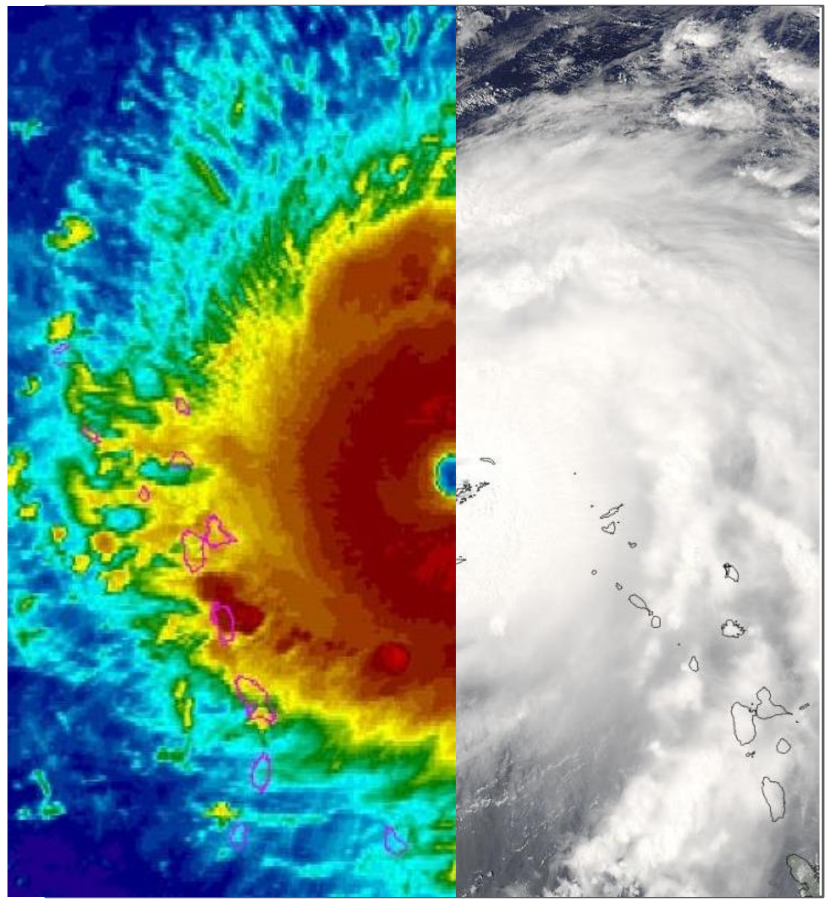



```
model=keras.models.Sequential()
model.add(keras.layers.Dense(150, activation="relu"))
model.add(keras.layers.Dropout(0.5))
```

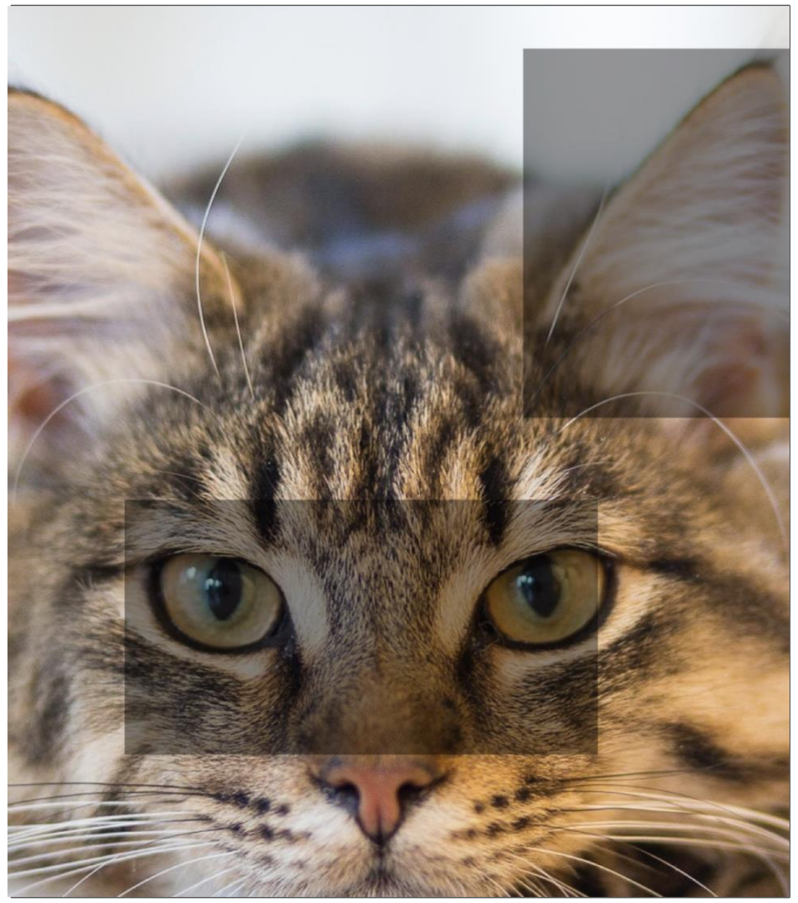
CHALLENGES AND POTENTIAL SOLUTIONS

LABELLING LARGE QUANTITIES OF DATA

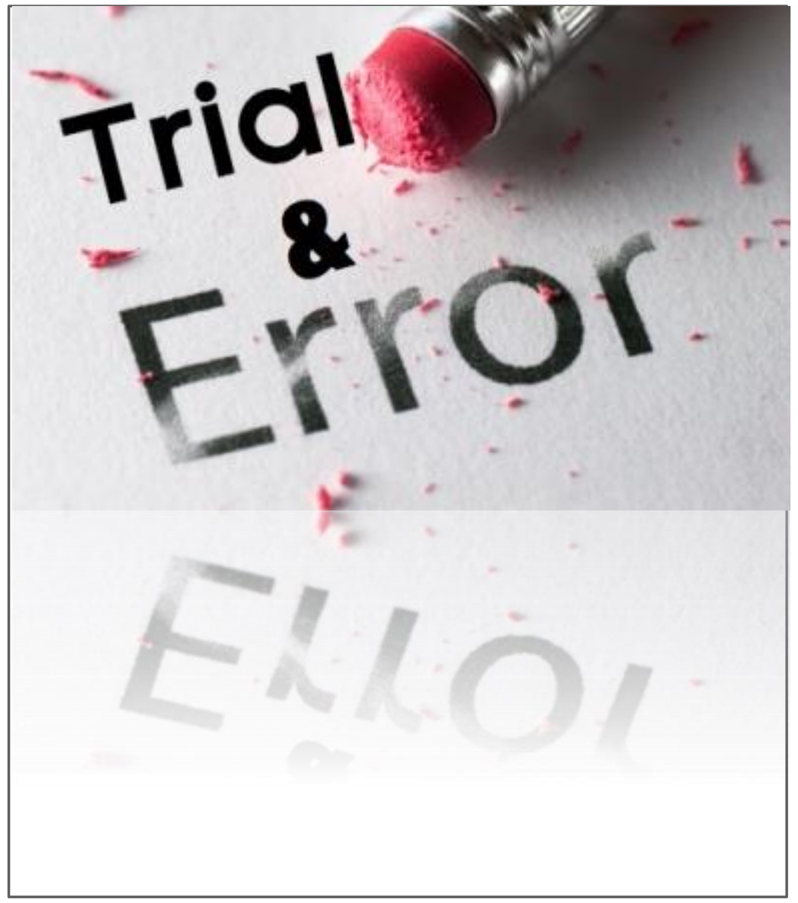
How can we overcome the need for manual labelling?



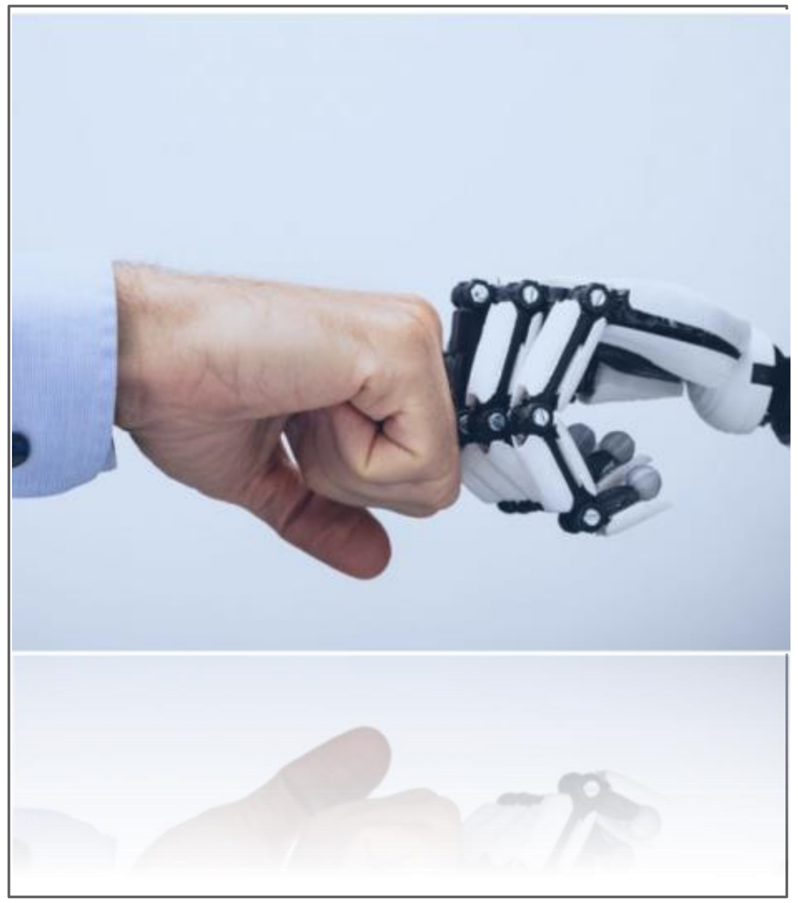
Data Fusion
Using one data source as the label for another



Self-Supervised Learning
Predicting input B from input A

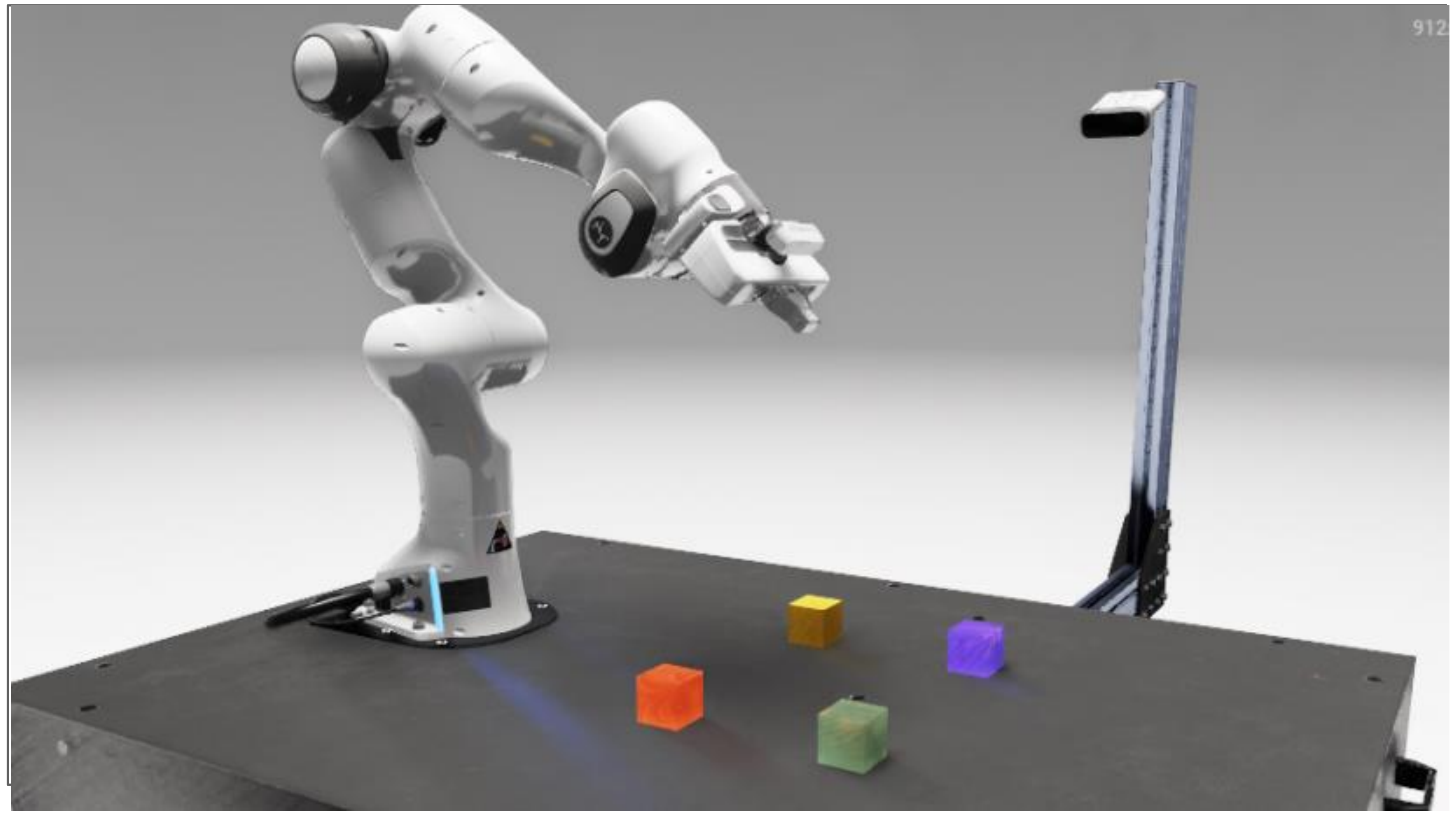


Reinforcement Learning
Obtaining labels directly from the environment or simulation

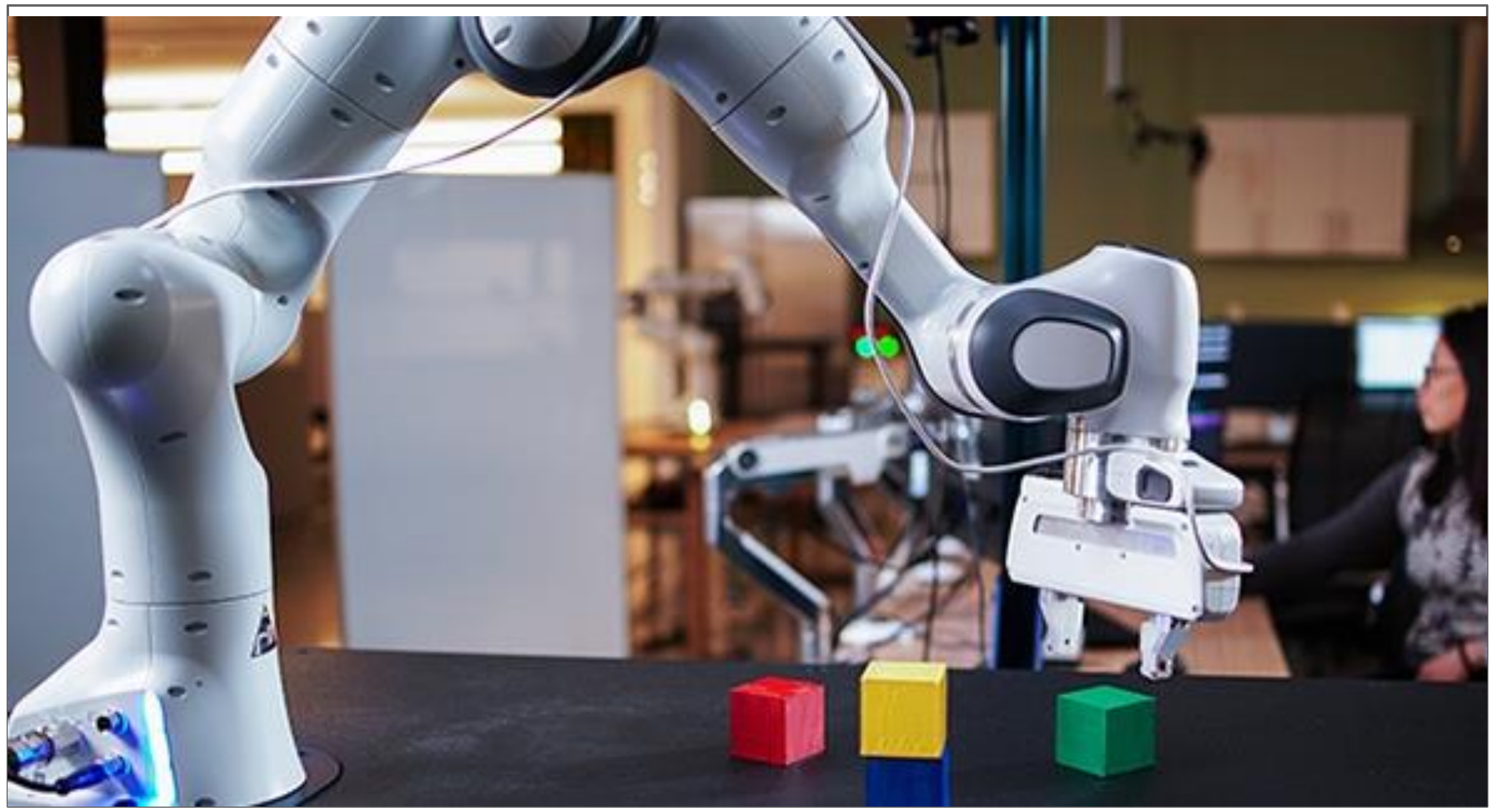


Human-in-the-loop
Using human machine iteration to make labelling easier

TRANSFER LEARNING: DON'T START FROM SCRATCH

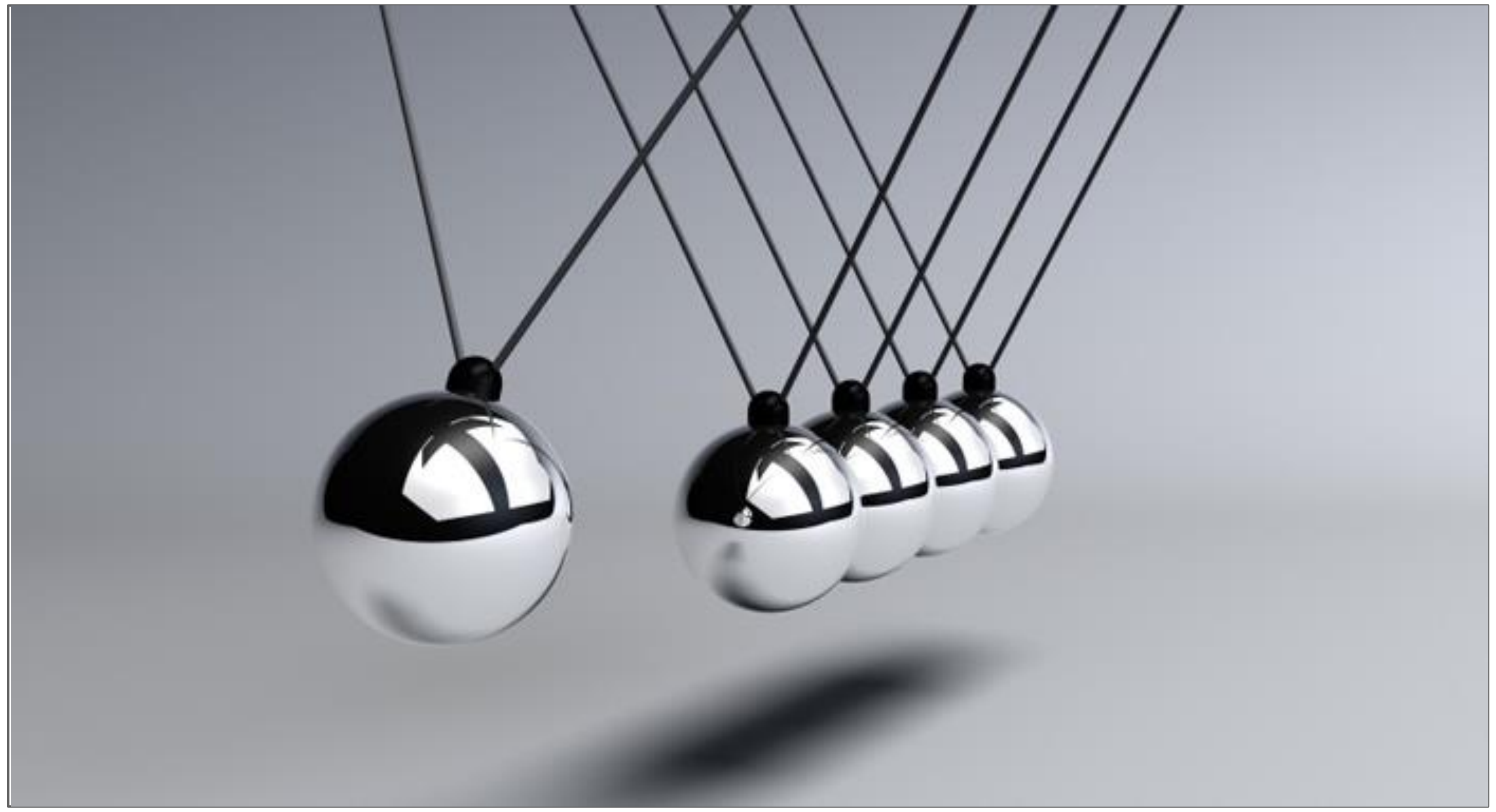


Train on simulated or related data

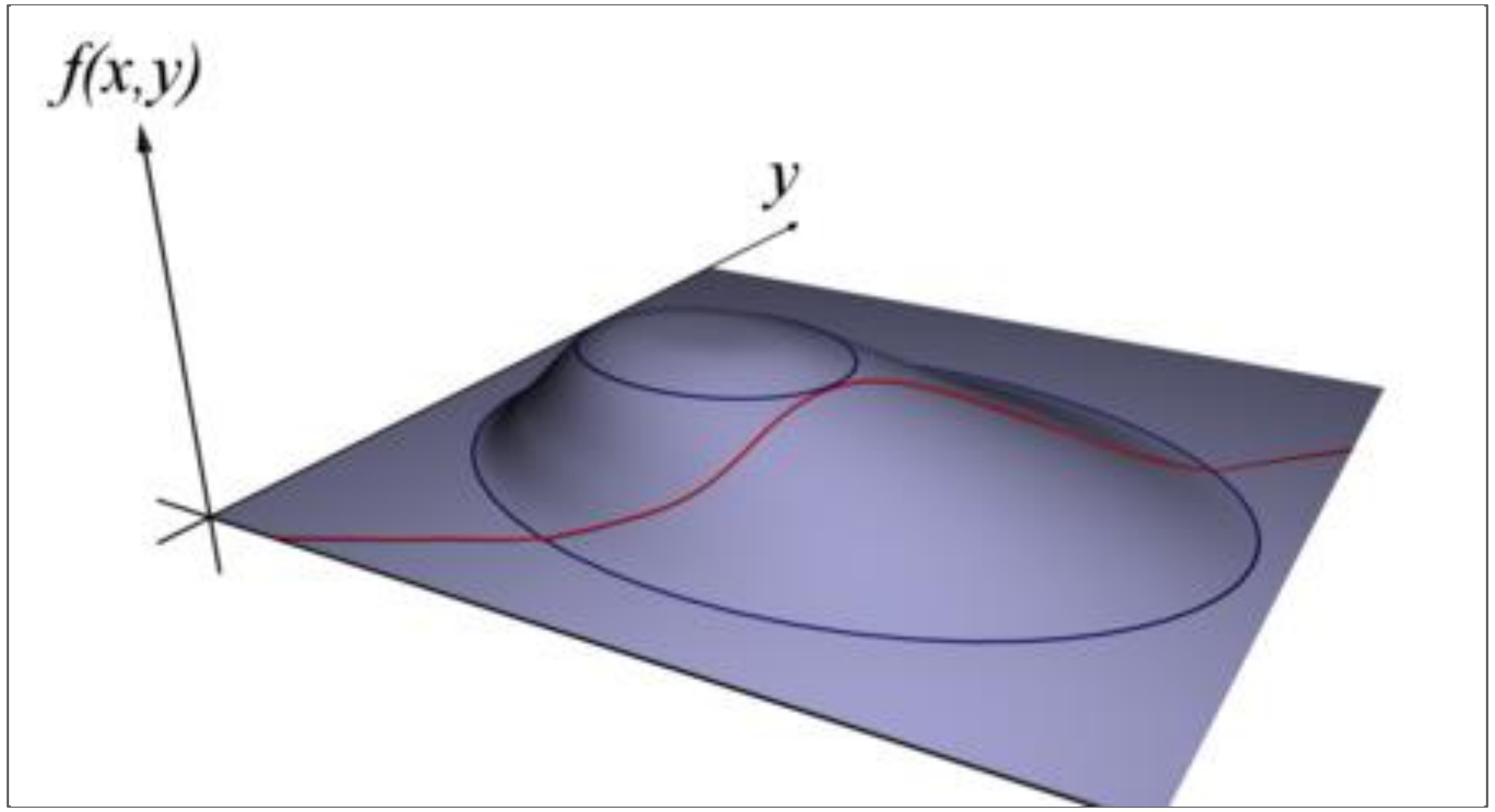


Fine-tune on the real data

ENFORCING PHYSICAL CONSTRAINTS

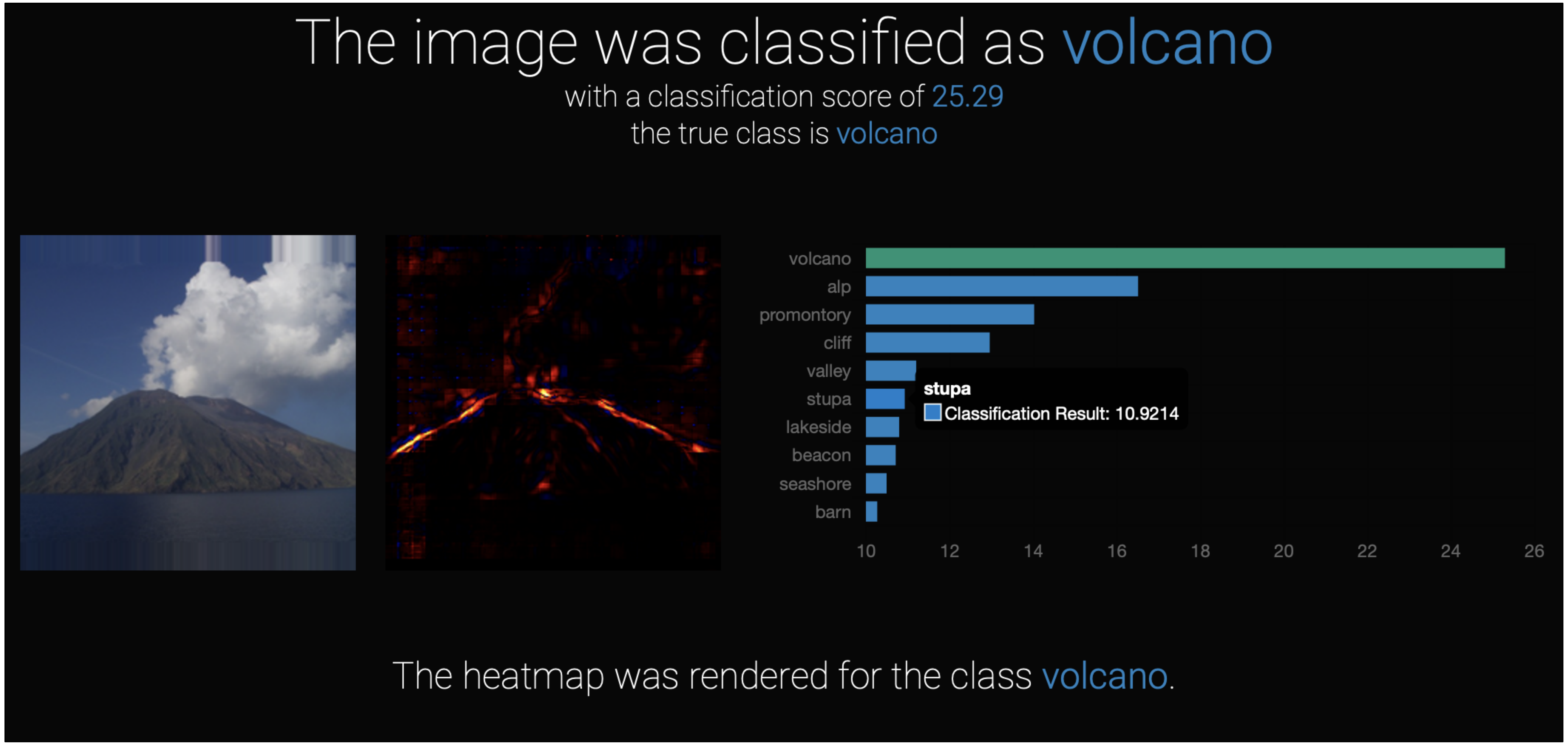


Conservation of Mass, Momentum, Energy, Incompressibility, Turbulent Energy Spectra, Translational Invariance



Lagrange multipliers (penalization), Hard Constraints, Projective Methods, Differentiable Programming

INTERPRETABILITY: EXPLAINABLE AI



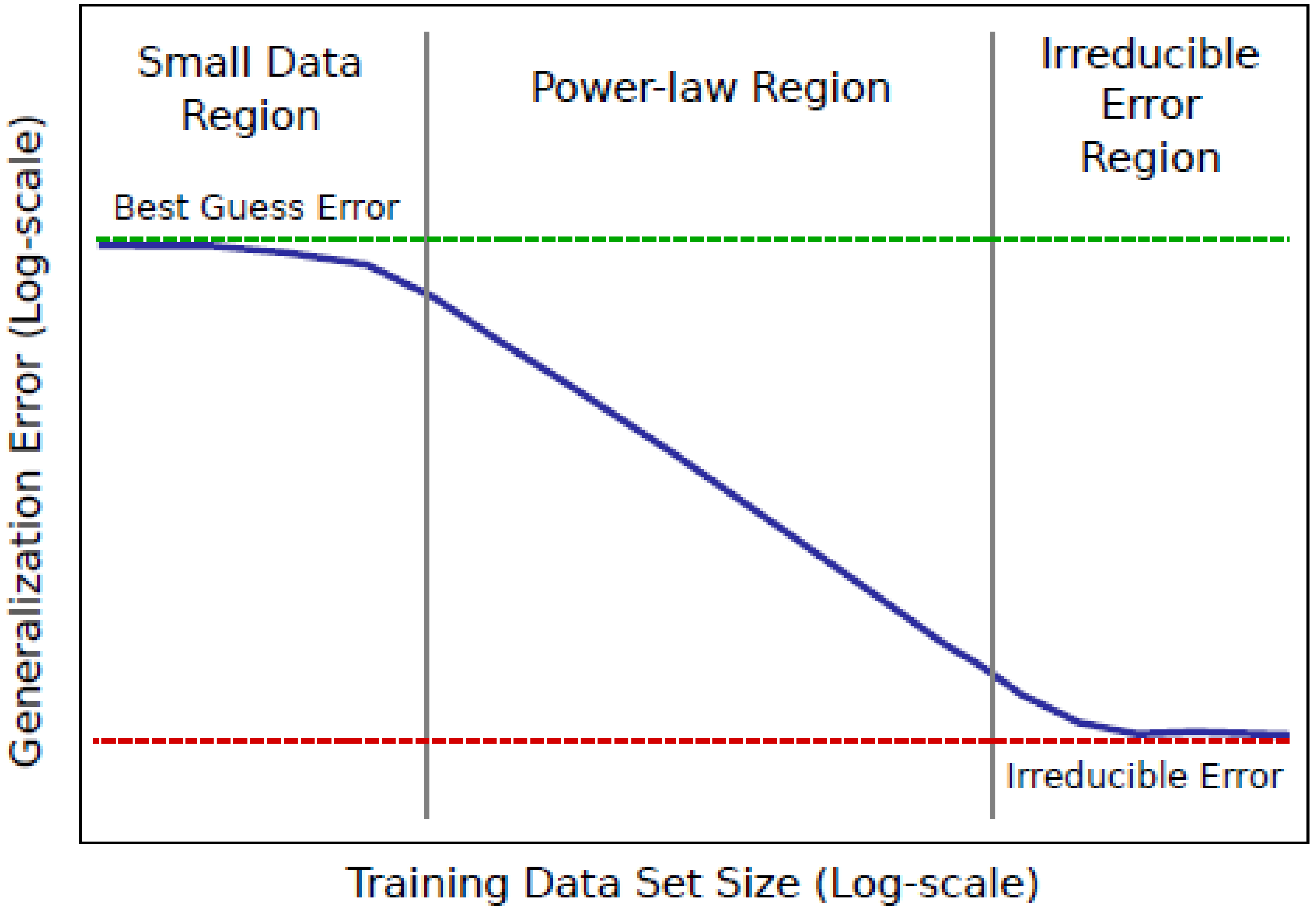
Layer-wise Relevance Propagation

https://lrpserver.hhi.fraunhofer.de/image-classification

USING YOUR GPU

EXPLODING DATASETS

Logarithmic relationship between the dataset size and accuracy



Hestness, J., Narang, S., Ardalani, N., Diamos, G., Jun, H., Kianinejad, H., ... & Zhou, Y. (2017). Deep Learning Scaling is Predictable, Empirically. arXiv preprint arXiv:1712.00409.

THE SCALING LAWS

As you increase the dataset size you must increase the model size

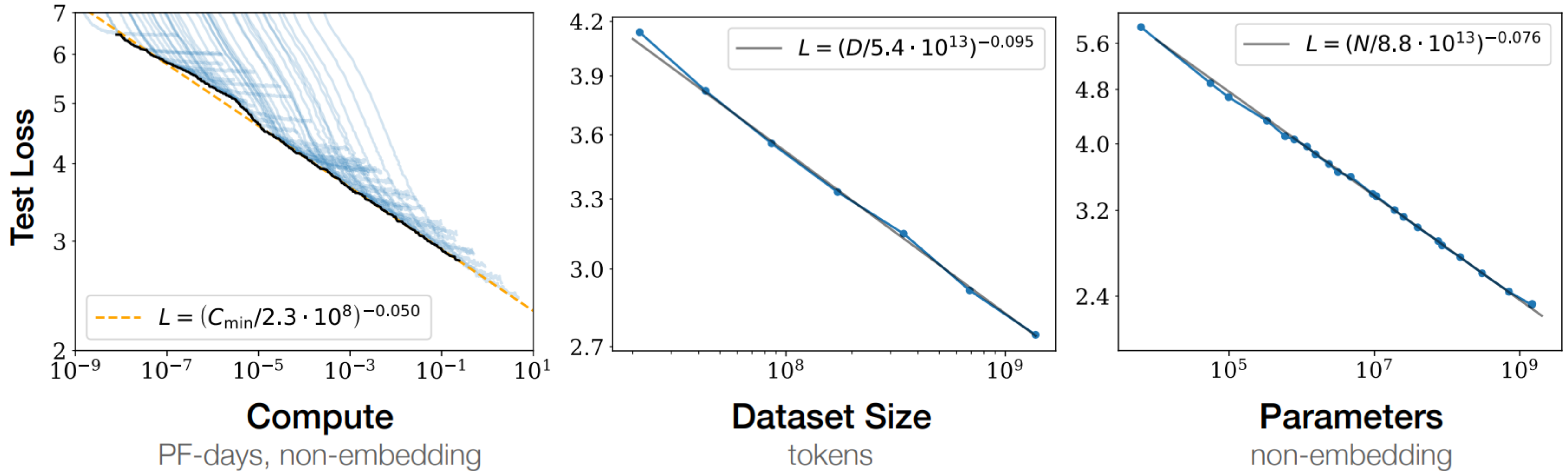
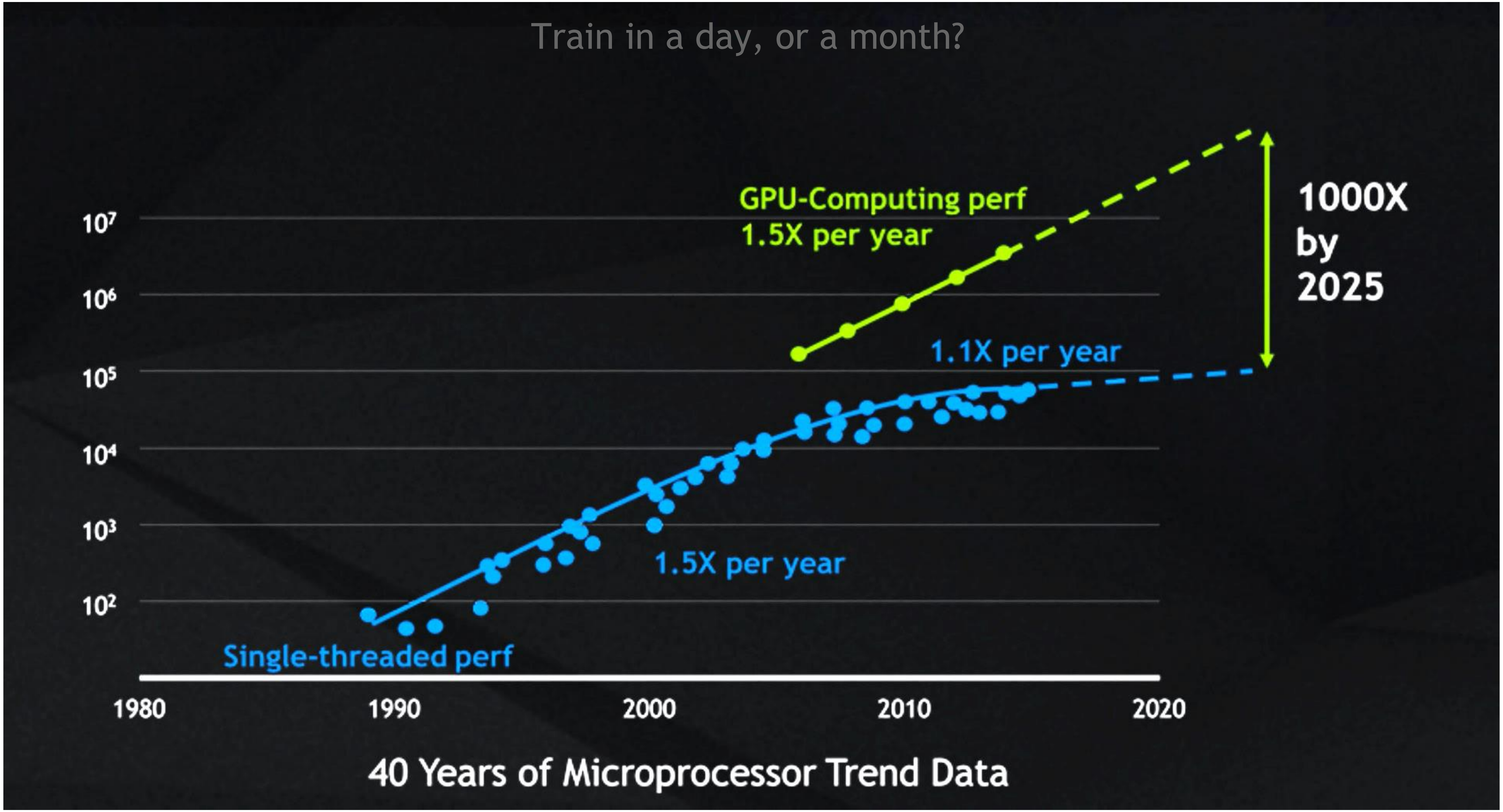


Figure 1 Language modeling performance improves smoothly as we increase the model size, datasetset size, and amount of compute[2] used for training. For optimal performance all three factors must be scaled up in tandem. Empirical performance has a power-law relationship with each individual factor when not bottlenecked by the other two.

Kaplan, J., McCandlish, S., Henighan, T., Brown, T. B., Chess, B., Child, R., ... & Amodei, D. (2020). Scaling Laws for Neural Language Models. *arXiv preprint arXiv:2001.08361*.

GPUS MAKE MACHINE LEARNING PRACTICAL

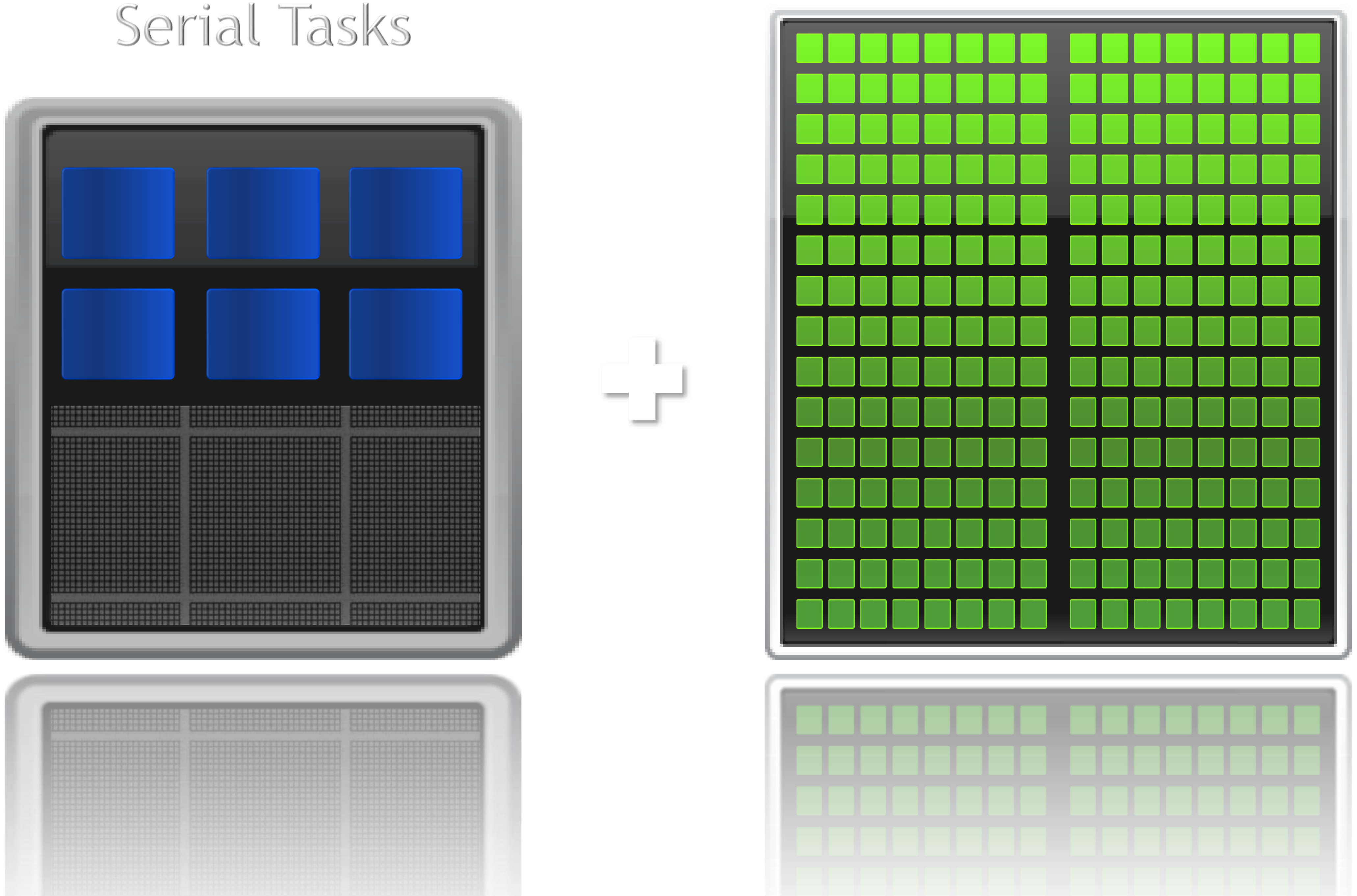


ACCELERATED COMPUTING

GPU Accelerator

CPU

Optimized for Serial Tasks

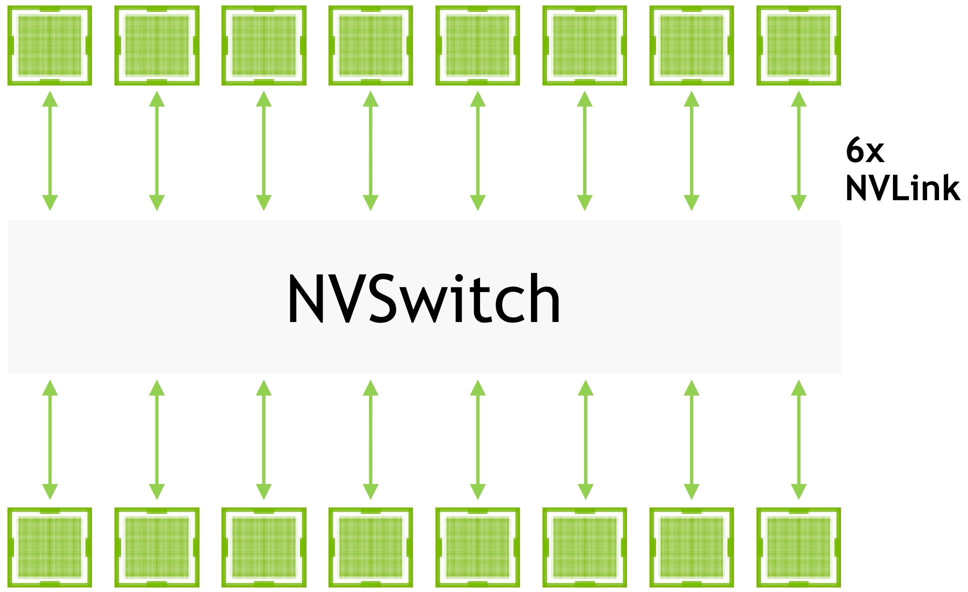


PILLARS OF DATA SCIENCE PERFORMANCE

CUDA Architecture

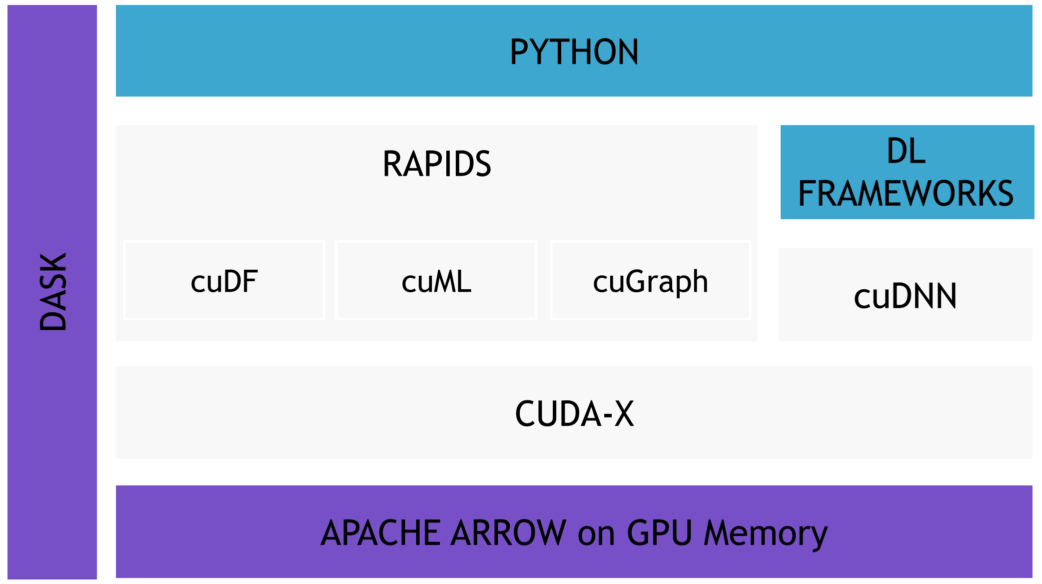
Massively Parallel Processing

NVLink/NVSwitch



High Speed Connecting between GPUs for Distributed Algorithms

CUDA-X AI



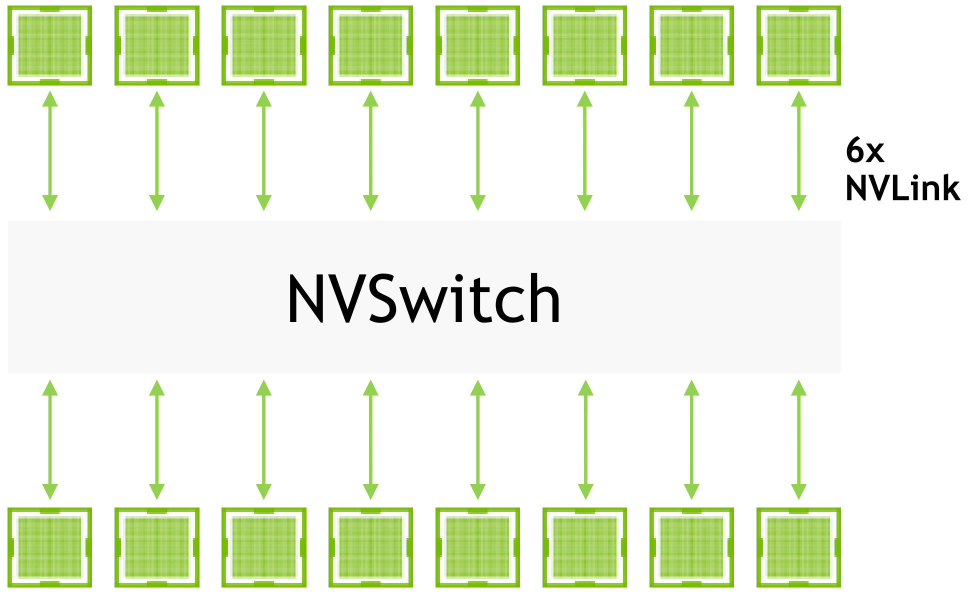
NVIDIA GPU Acceleration Libraries for Data Science and AI

PILLARS OF DATA SCIENCE PERFORMANCE

CUDA Architecture

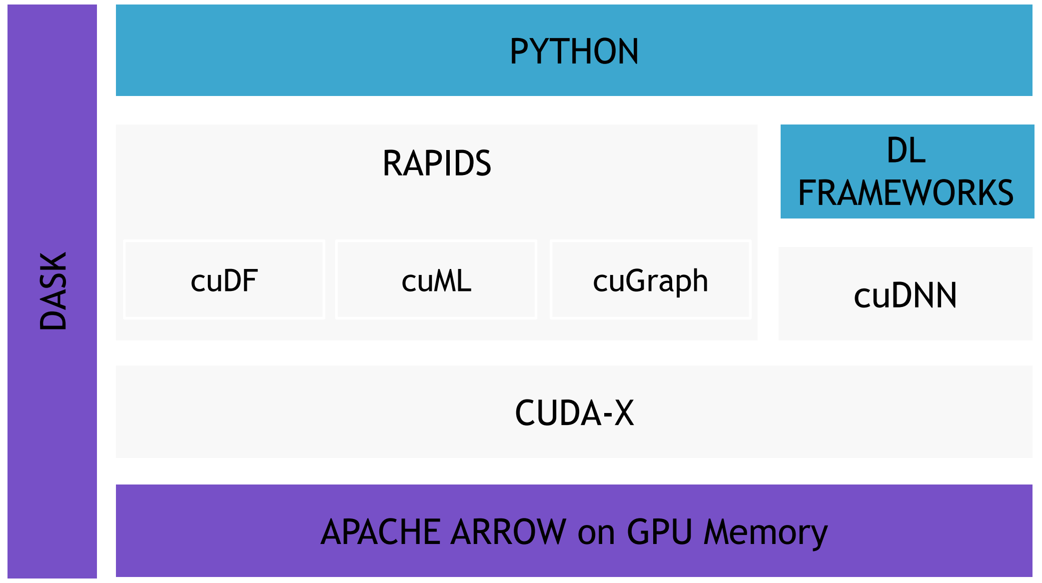
Massively Parallel Processing

NVLink/NVSwitch



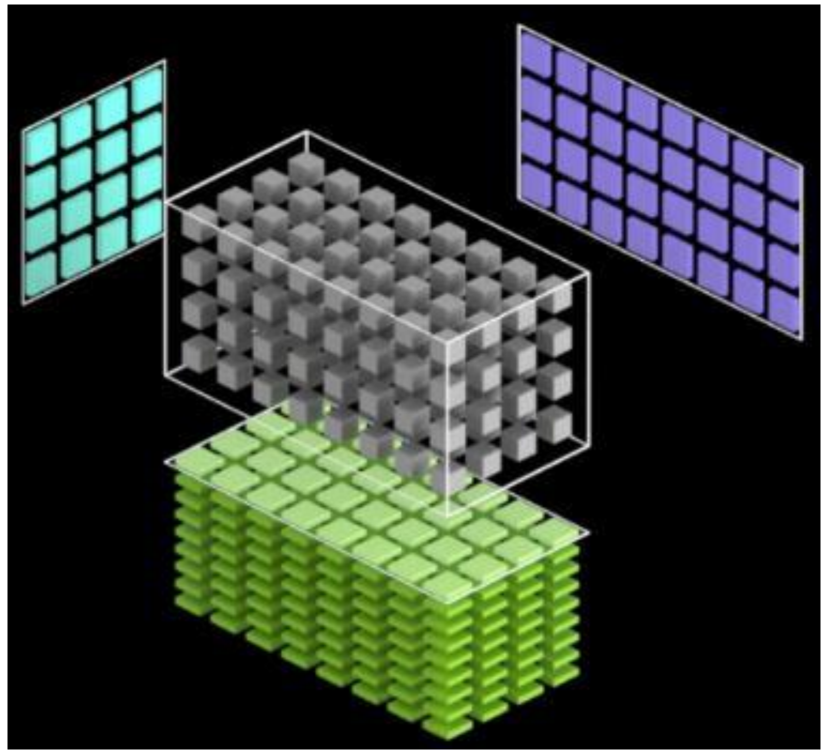
High Speed Connecting between GPUs for Distributed Algorithms

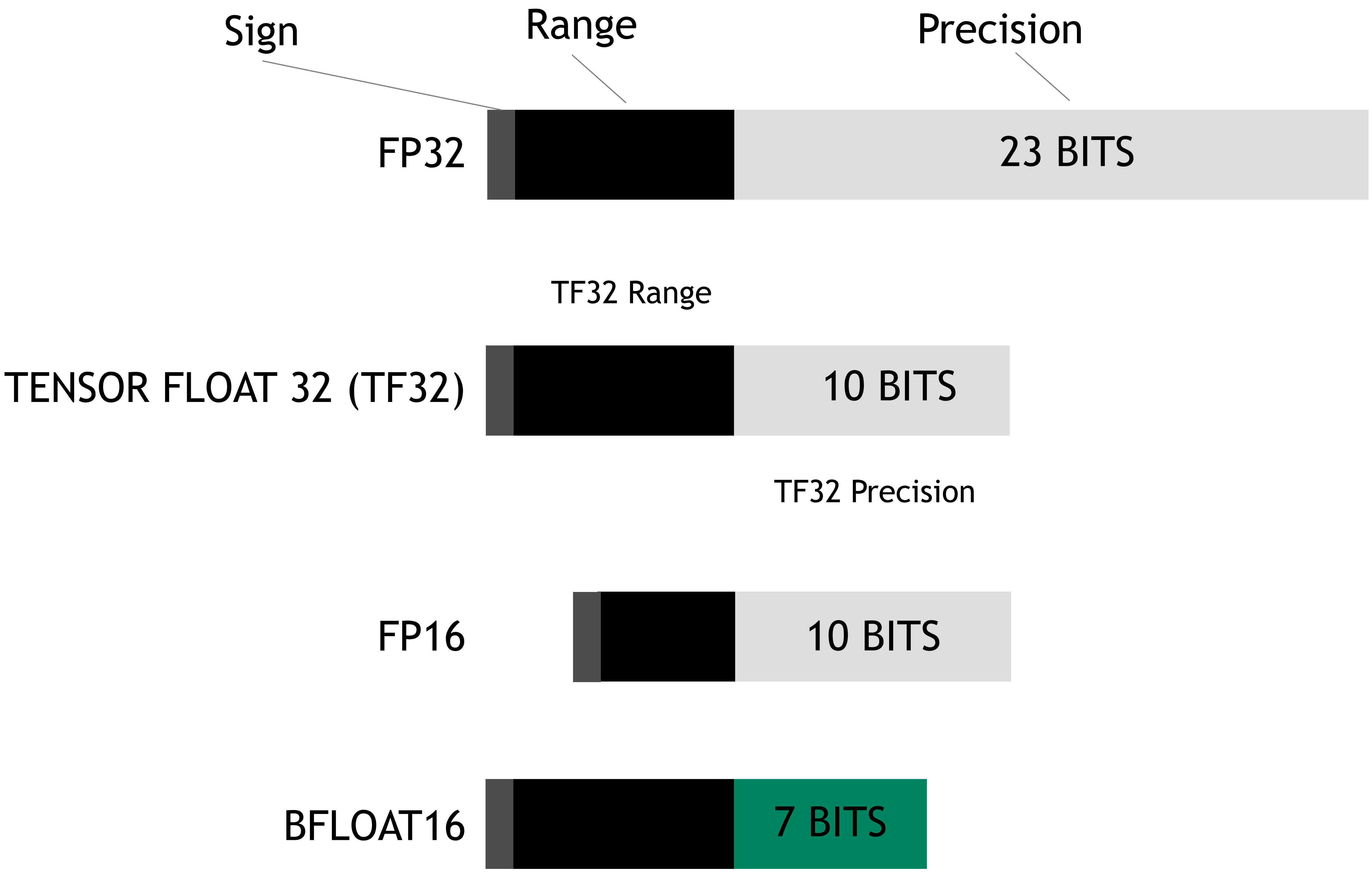
CUDA-X AI



NVIDIA GPU Acceleration Libraries for Data Science and AI

TENSOR CORES FOR DIFFERENT NEEDS





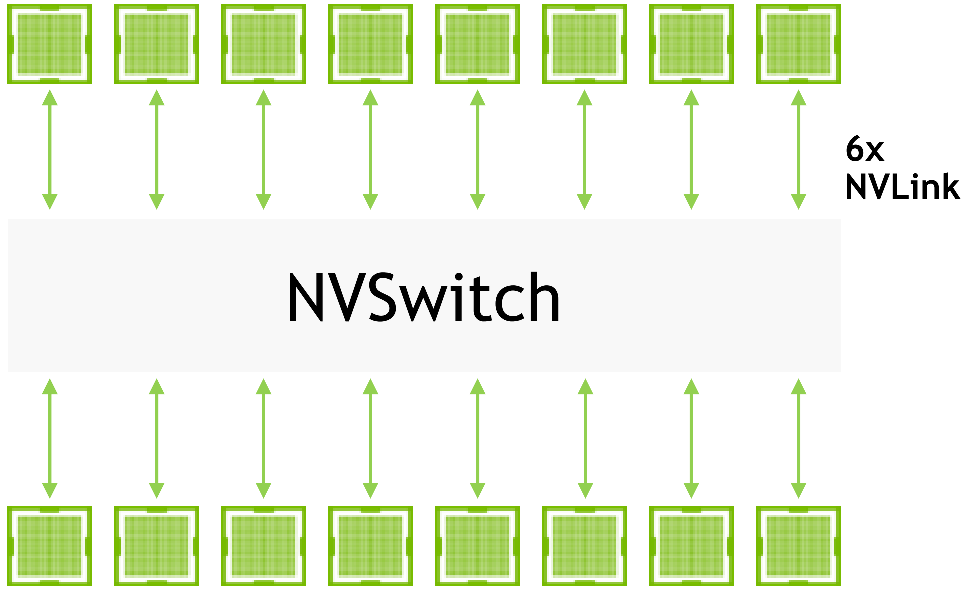
- No Code Change Speed-up for Training

PILLARS OF DATA SCIENCE PERFORMANCE

CUDA Architecture

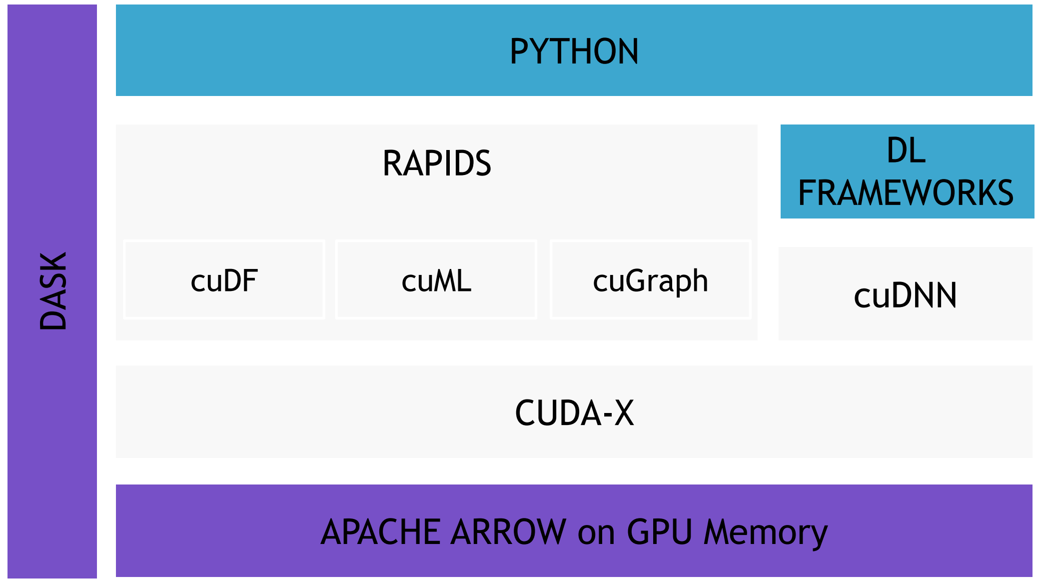
Massively Parallel Processing

NVLink/NVSwitch



High Speed Connecting between GPUs for Distributed Algorithms

CUDA-X AI



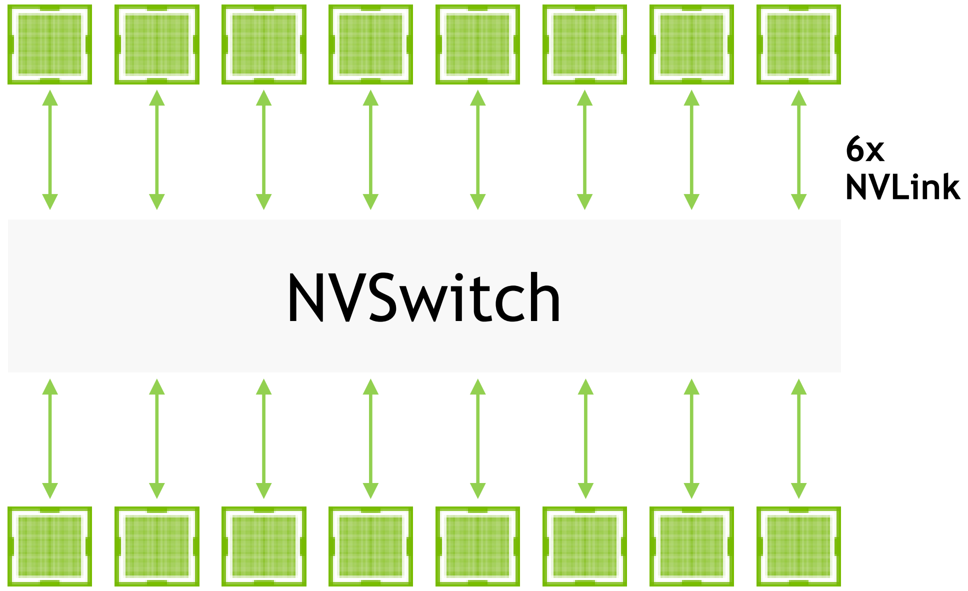
NVIDIA GPU Acceleration Libraries for Data Science and AI

PILLARS OF DATA SCIENCE PERFORMANCE

CUDA Architecture

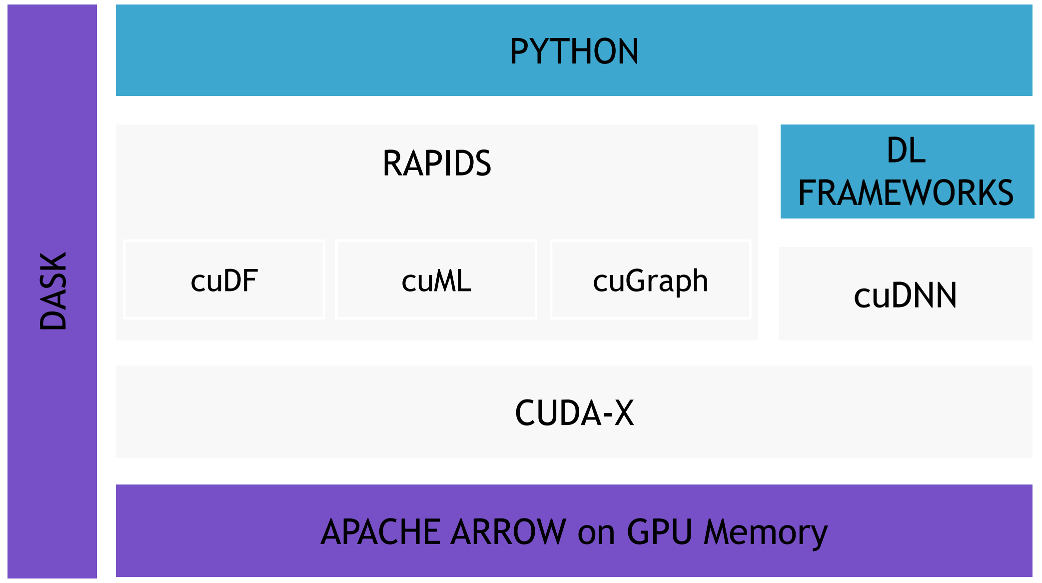
Massively Parallel Processing

NVLink/NVSwitch



High Speed Connecting between GPUs for Distributed Algorithms

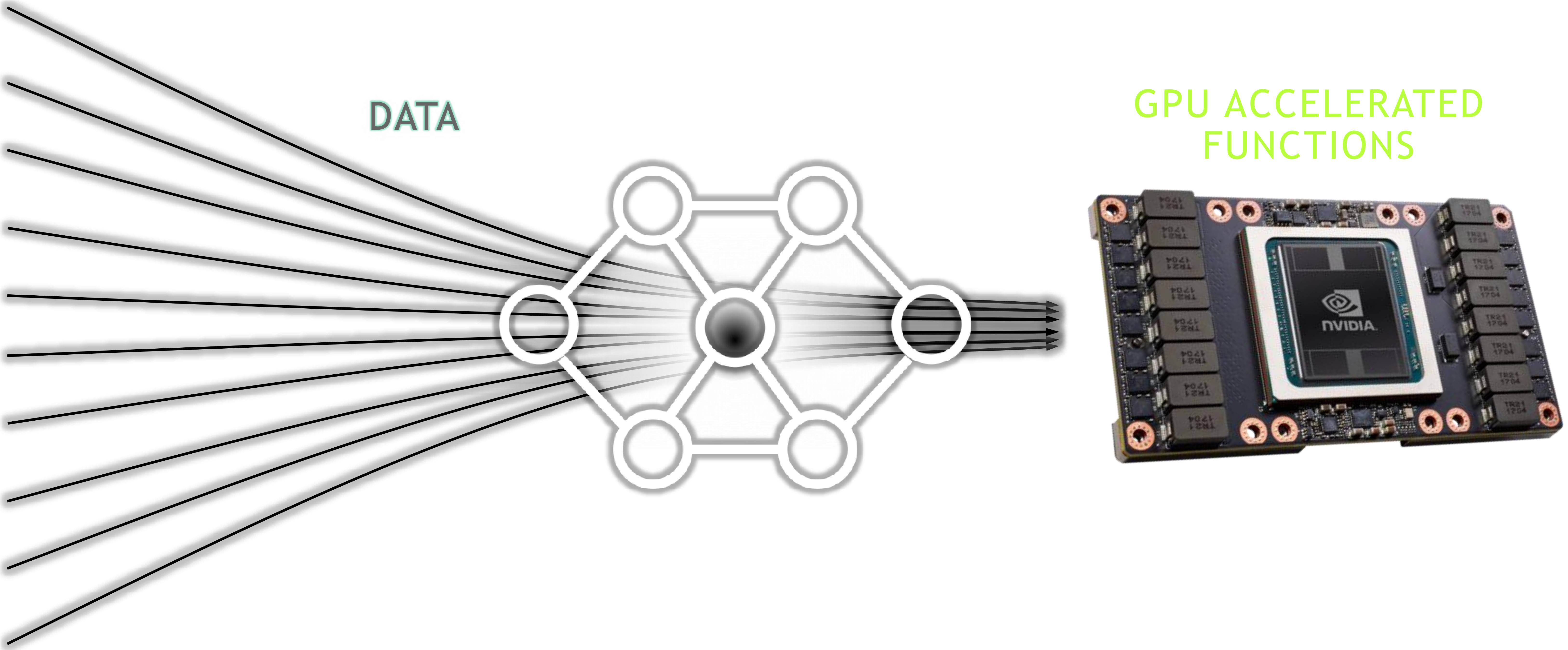
CUDA-X AI



NVIDIA GPU Acceleration Libraries for Data Science and AI

LEARNED FUNCTIONS ARE GPU ACCELERATED

Next level software. No porting required.

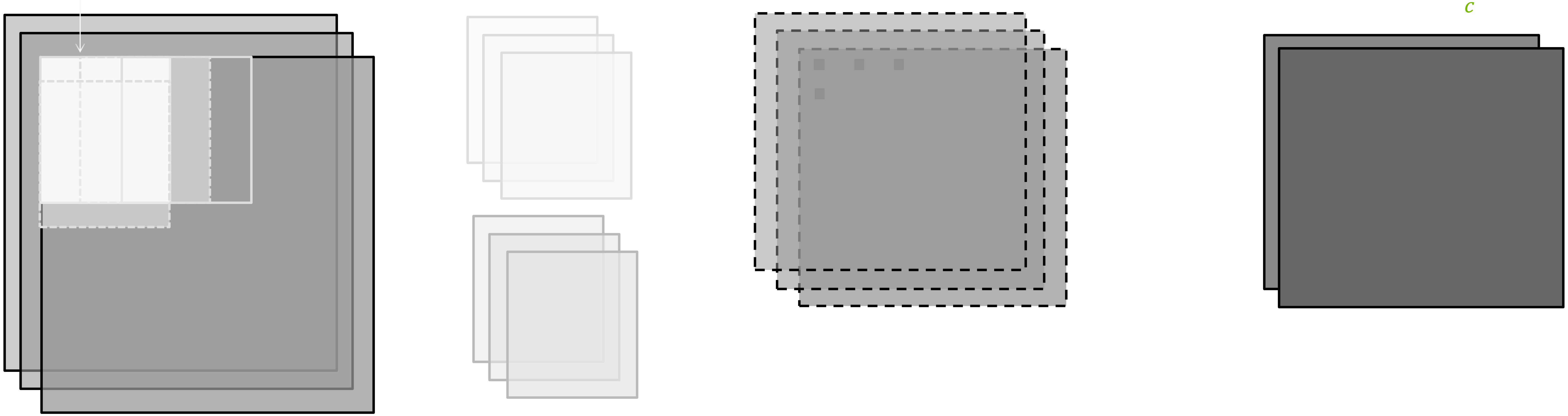


CONVOLUTION OPERATION

$$int[c,p,q] = \sum_{i,j\,\epsilon\,filter} Im[c][istart+i, jstart+j]\,.\,Filt[c][i,j]$$

$$output[p,q] = \sum_{c} int[c,p,q]$$

Pointwise multiply and sum, scalar output



Input Image | Input Filter | Intermediate output | Final Output

Why do it once if you can do it n times ? Batch the whole thing.

VERIFYING YOUR GPU

TensorFlow

```
from tensorflow.python.client import device_lib
print('GPU' in str(device_lib.list_local_devices()))
```

```
True
```

Keras

```
from keras import backend as K
K.tensorflow_backend._get_available_gpus()
```

```
Using TensorFlow backend.
```

```
['/job:localhost/replica:0/task:0/device:GPU:0',
 '/job:localhost/replica:0/task:0/device:GPU:1']
```

PyTorch

```
from torch import cuda
print(f"cuda.is_available: {cuda.is_available()}")
print(f"cuda.device_count(): {cuda.device_count()}")
print(cuda.get_device_name(cuda.current_device()))
```

```
cuda.is_available: True
cuda.device_count(): 2
Quadro GV100
```

Julia

```
using CUDAdrv, Printf
@printf("device 0 = %s \n" , CUDAdrv.name(CuDevice(0)))
@printf("device 1 = %s \n" , CUDAdrv.name(CuDevice(1)))
```

```
device 0 = Quadro GV100
device 1 = Quadro GV100
```

TRAINING ON A SINGLE GPU

Keras

Automatically uses GPU if available

Julia

```julia
use_gpu = true
to_device(x) = use_gpu ? gpu(x) : x
x     = to_device.(x)
y     = to_device.(y)
model = to_device(model)
```

PyTorch

```python
device = torch.device("cuda:1" if torch.cuda.is_available() else "cpu")

# move data onto the GPU
model = model.to(device)
X,Y   = X.to(device), Y.to(device)

# Alocate on the GPU directly
dtype = torch.cuda.FloatTensor
X = torch.zeros(100).type(dtype)

# set default location for tensors
torch.set_default_tensor_type('torch.cuda.FloatTensor')
```

NVIDIA-SMI

System Management Interface

```
Terminal  File  Edit  View  Search  Terminal  Help
Every 0.5s: nvidia-smi                                  Tue Sep 17 10:02:34 2019

Tue Sep 17 10:02:34 2019
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 396.82                 Driver Version: 396.82                    |
|-------------------------------+----------------------+----------------------+
| GPU  Name        Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Compute M. |
|===============================+======================+======================|
|   0  Quadro GV100        Off  | 00000000:04:00.0  On |                  Off |
| 49%   61C    P2    51W / 250W |   1451MiB / 32500MiB |      3%      Default |
+-------------------------------+----------------------+----------------------+
|   1  Quadro GV100        Off  | 00000000:84:00.0 Off |                  Off |
| 41%   58C    P2   116W / 250W |   1701MiB / 32508MiB |     41%      Default |
+-------------------------------+----------------------+----------------------+

+-----------------------------------------------------------------------------+
| Processes:                                                       GPU Memory |
|  GPU       PID   Type   Process name                             Usage      |
|=============================================================================|
|    0      1661      G   /usr/lib/xorg/Xorg                           328MiB |
|    0      2381      G   compiz                                        70MiB |
|    0      8598      G   ...quest-channel-token=8542253777236559196   102MiB |
|    0     10245      C   /home/dhall/anaconda3/envs/tf/bin/python     946MiB |
|    1     10245      C   /home/dhall/anaconda3/envs/tf/bin/python    1690MiB |
+-----------------------------------------------------------------------------+
```

Memory Utilizaiton

Processor Utilization

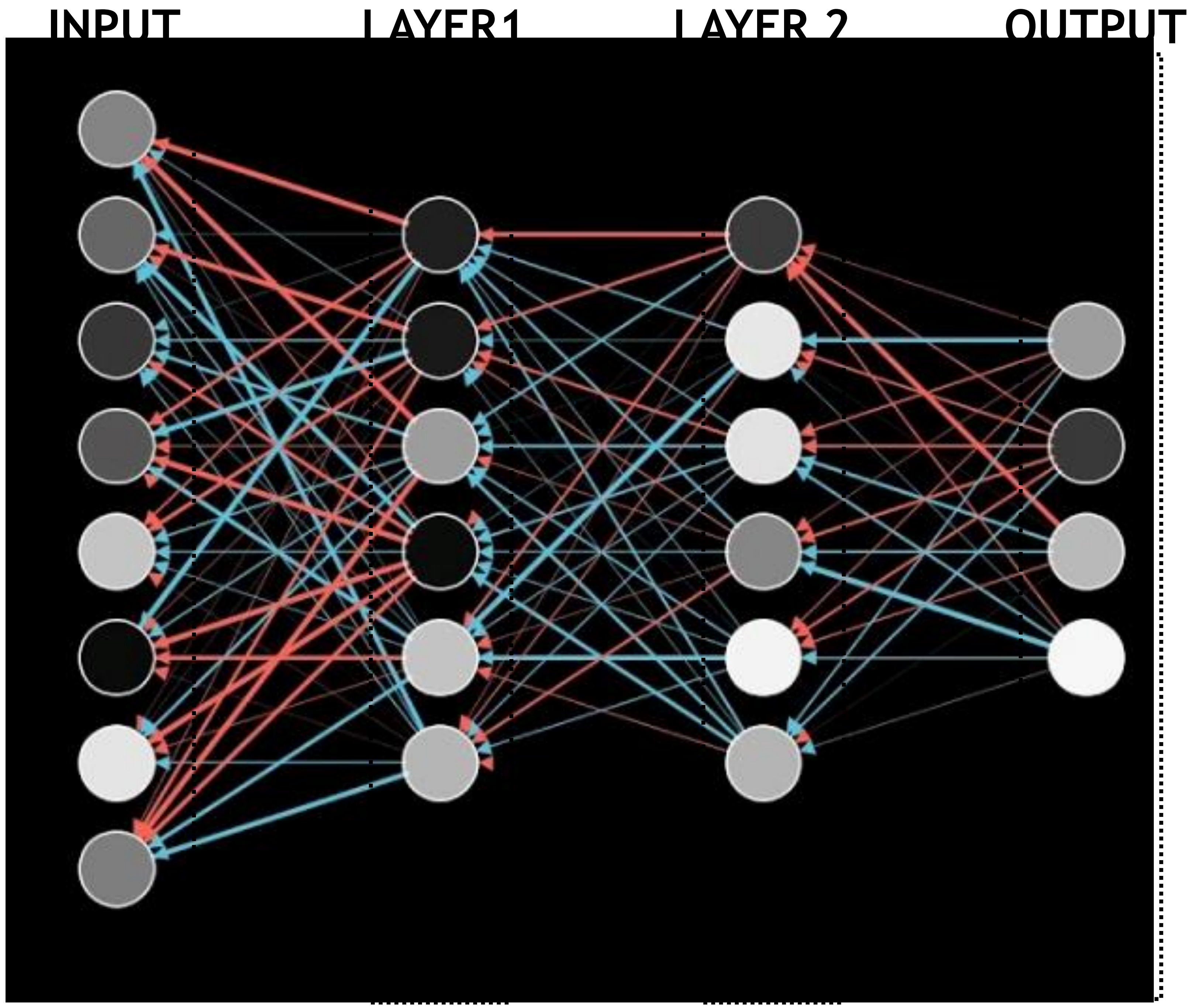
Process Info

FULLY CONNECTED NETWORKS

(MULTI-LAYER PERCPTRONS)

FULLY CONNECTED NETWORKS

A given neuron is connected to every neuron in the previous layer

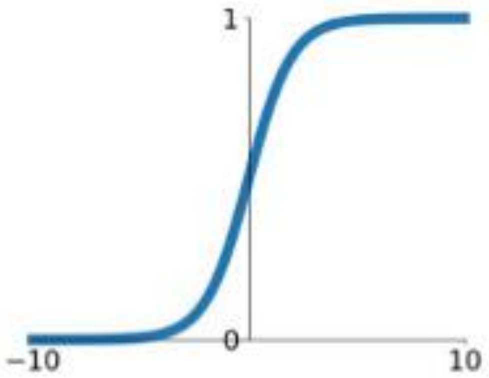


ACTIVATION FUNCTIONS

Many to choose from. But most use ReLU or LeakyReLU

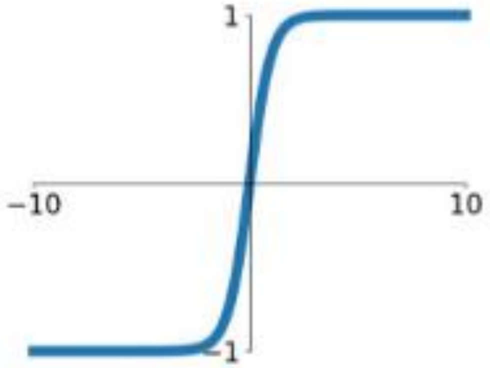
Sigmoid

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



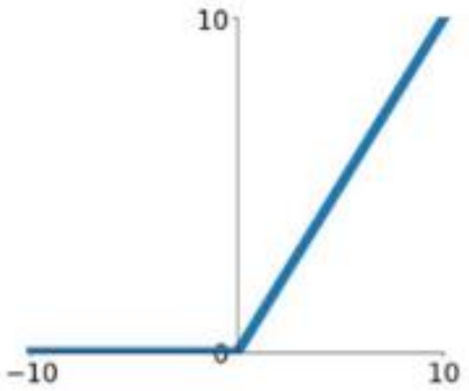
tanh

$\tanh(x)$



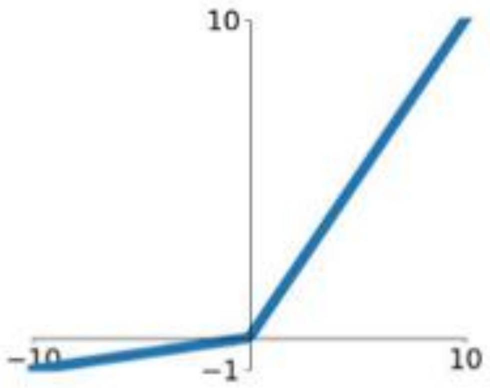
ReLU

$\max(0, x)$



Leaky ReLU

$\max(0.1x, x)$

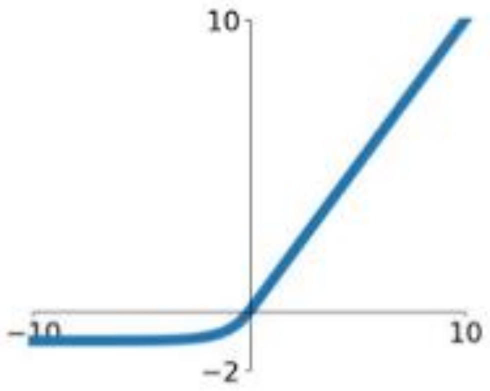


Maxout

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

ELU

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



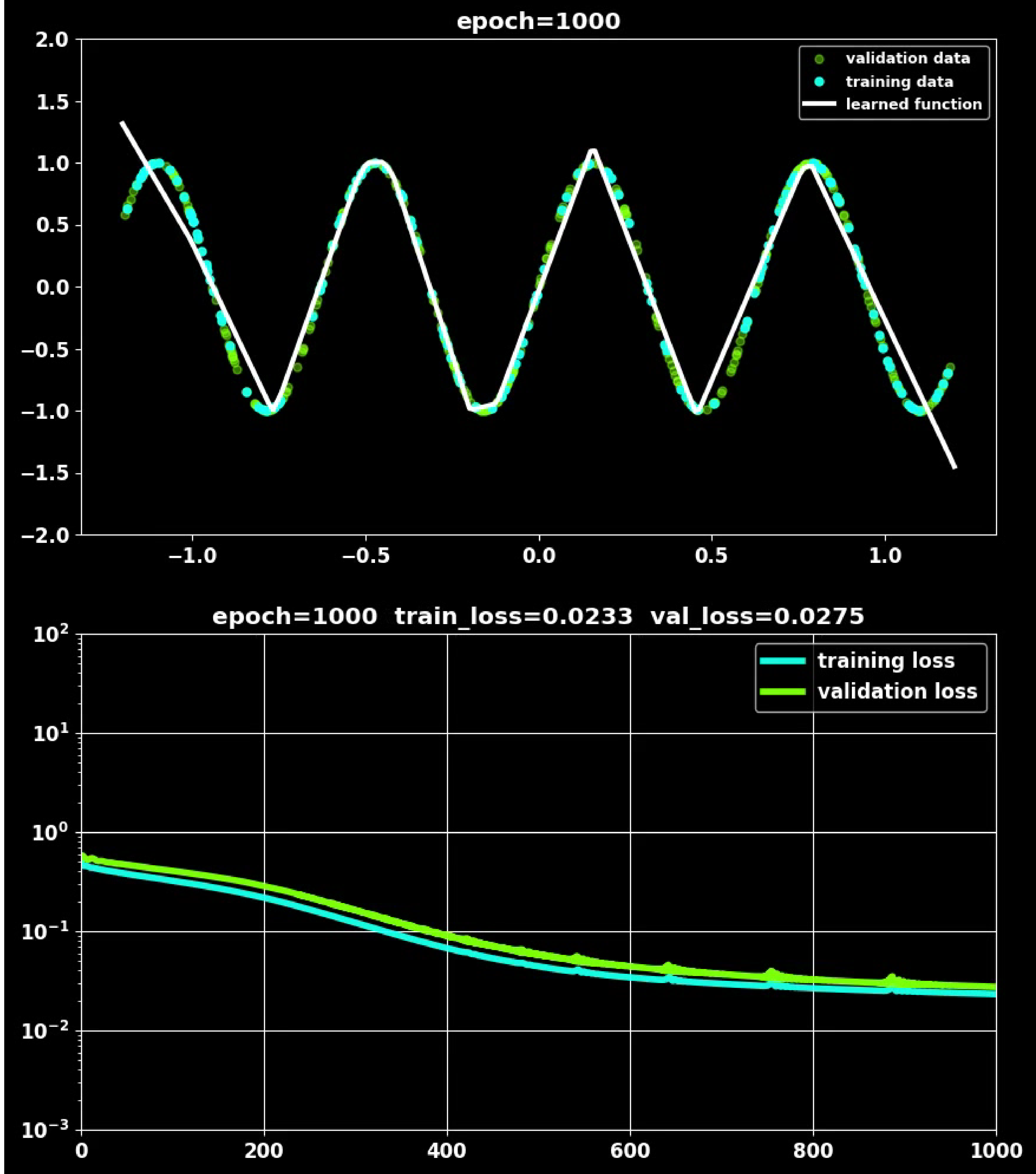
RELU BASIS FUNCTIONS

Piecewise continuous basis functions

Training Loop

```python
import torch.nn as nn
xtrain, ytrain, xval, yval = load_data(npts=500, train_fraction = 0.30)

# MODEL
n_bases = 100
model   = nn.Sequential(
    nn.Linear(1, n_bases),
    nn.ReLU(),
    nn.Linear(n_bases, 1))

optimizer = torch.optim.Adam(model.parameters(), lr=5.0e-3)
epochs    = 6000

for i in range(epochs+1):

    # training
    optimizer.zero_grad()
    yhat = model(xtrain)
    loss = (yhat - ytrain).pow(2).mean()
    loss.backward()
    optimizer.step()

    # validation
    yval_hat = model(xval)
    loss_val = (yval_hat - yval).pow(2).mean()
```

(Pytorch Code)

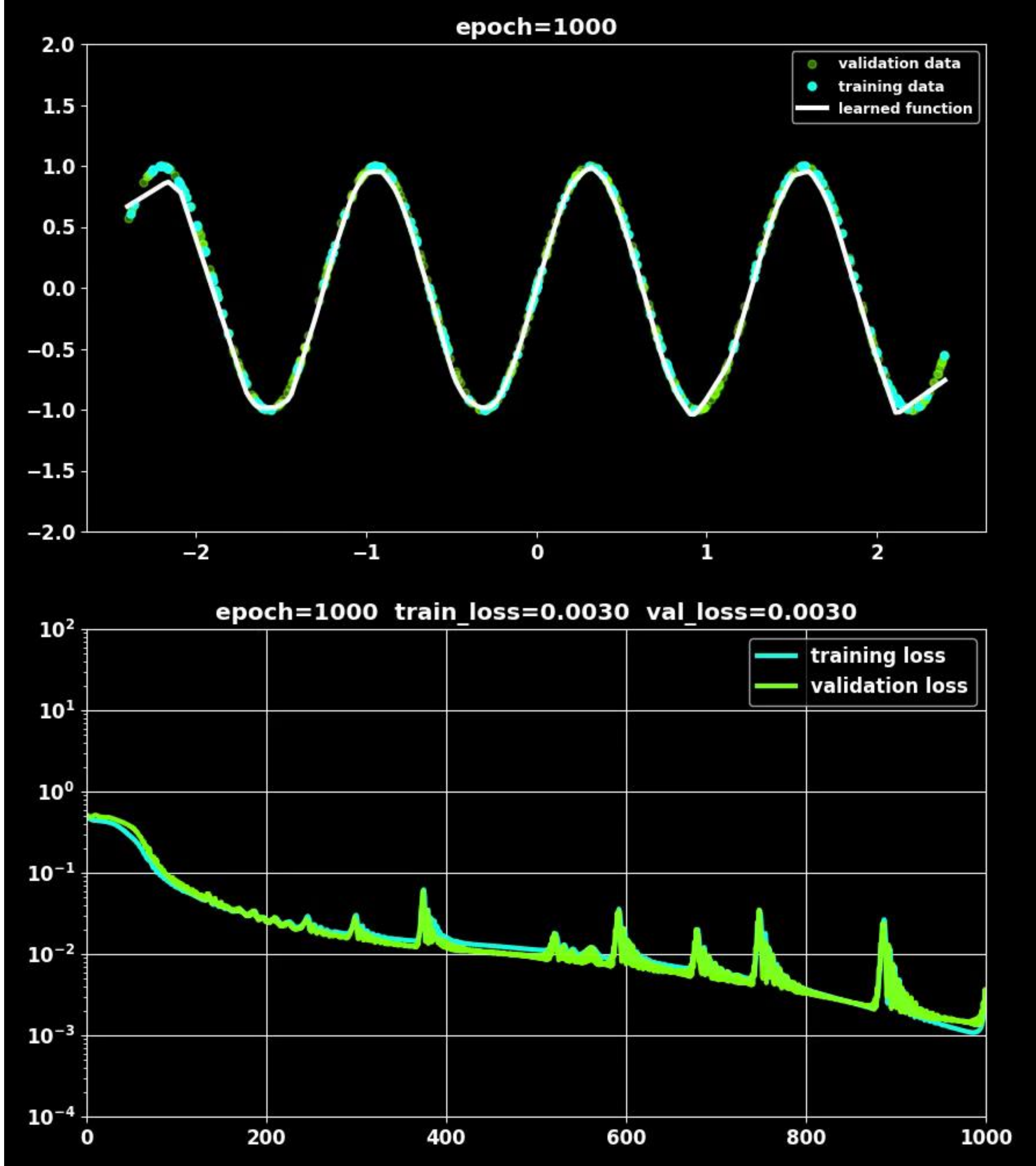


MULTI-LAYER NETWORKS

Piecewise continuous basis functions

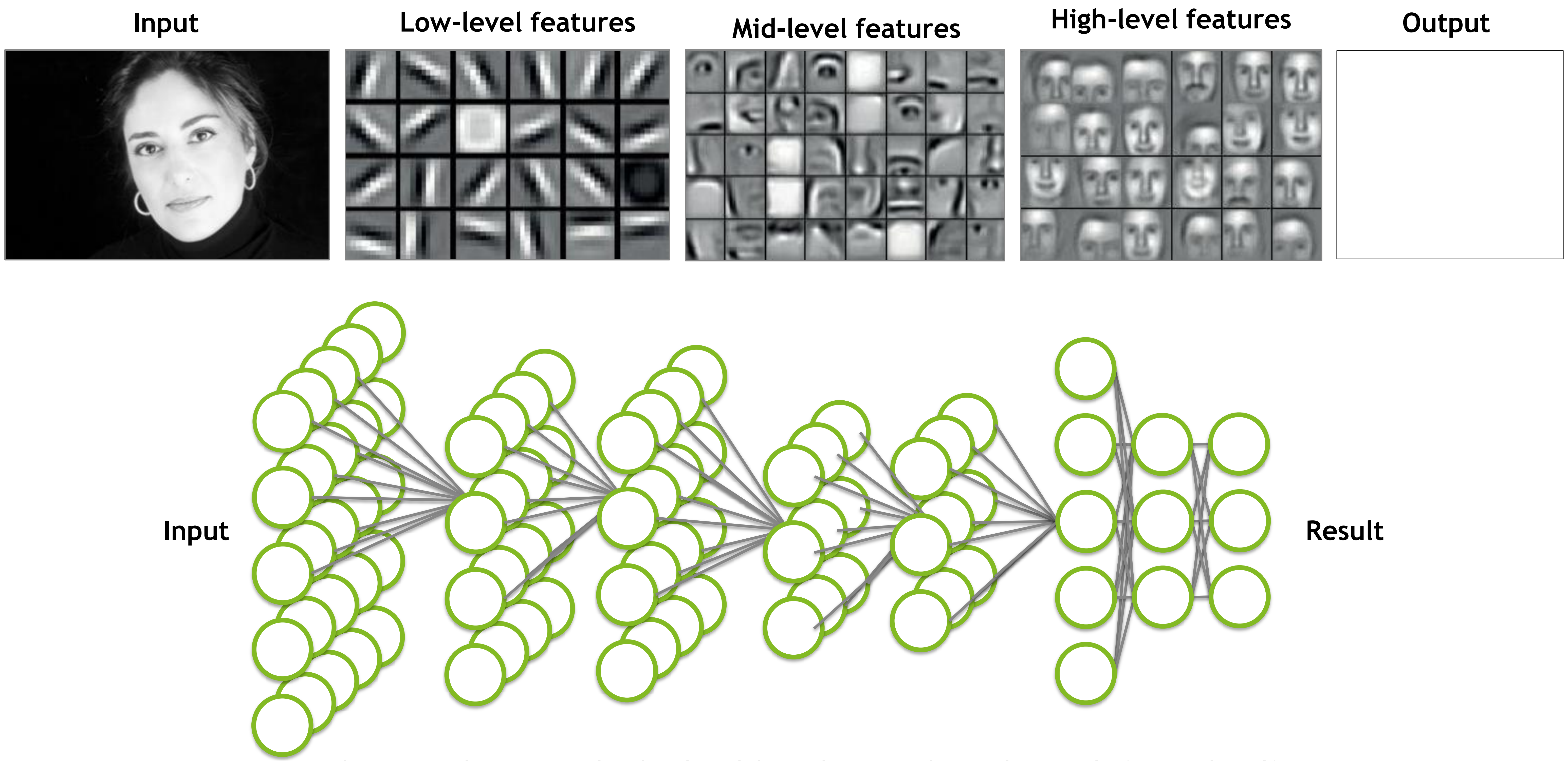
Training Loop

```
import torch.nn as nn

xtrain,ytrain,xval,yval=load_data(npts=500, train_fraction = 0.30)

# MODEL
n1,n2,n3 = 100,100,10

model = nn.Sequential(
    nn.Linear(1,  n1),  nn.ReLU(),
    nn.Linear(n1, n2), nn.ReLU(),
    nn.Linear(n2, n3), nn.ReLU(),
    nn.Linear(n3, 1 ))

optimizer = torch.optim.Adam(model.parameters(), lr=5.0e-3)

epochs    = 1000


for i in range(epochs+1):

    # training
    optimizer.zero_grad()
    yhat = model(xtrain)
    loss = (yhat - ytrain).pow(2).mean()
    loss.backward()
    optimizer.step()

    # validation
    yval_hat = model(xval)
    loss_val = (yval_hat - yval).pow(2).mean()
```



DEEPER NEURAL NETWORKS

More layers allows for more levels of abstraction



https://web.eecs.umich.edu/~honglak/icml09-ConvolutionalDeepBeliefNetworks.pdf

Large Scale Visual Recognition Challenge 2012

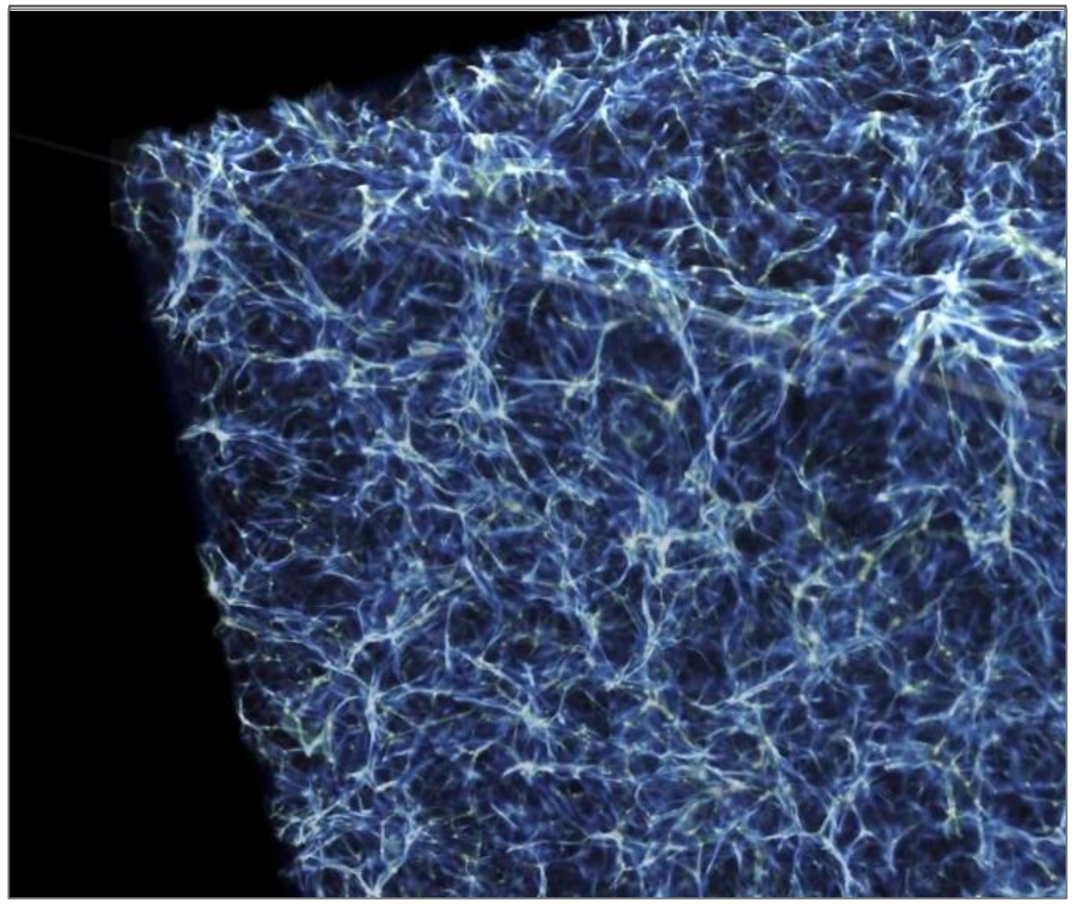
The Imagenet competition: Automatically classify images from 1000 different categories

CONVOLUTIONAL NEURAL NETWORKS

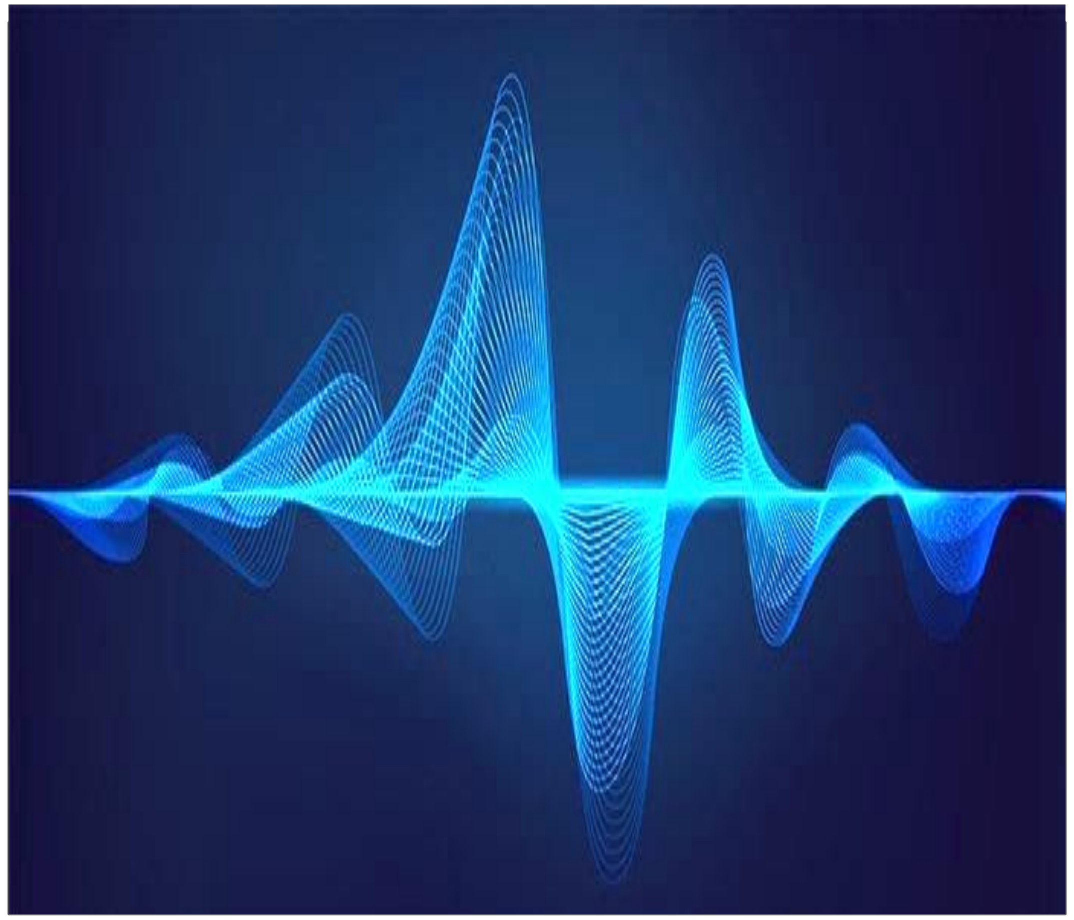
WHAT ARE CNNS USED FOR?

Problems with translational invariance

Computer Vision
Invariance in 2d space



Computational Physics
Invariance in 3d space



Audio and Time Series
Invariance in time

COMPUTER VISION TASKS

Each task requires a different model and data setup

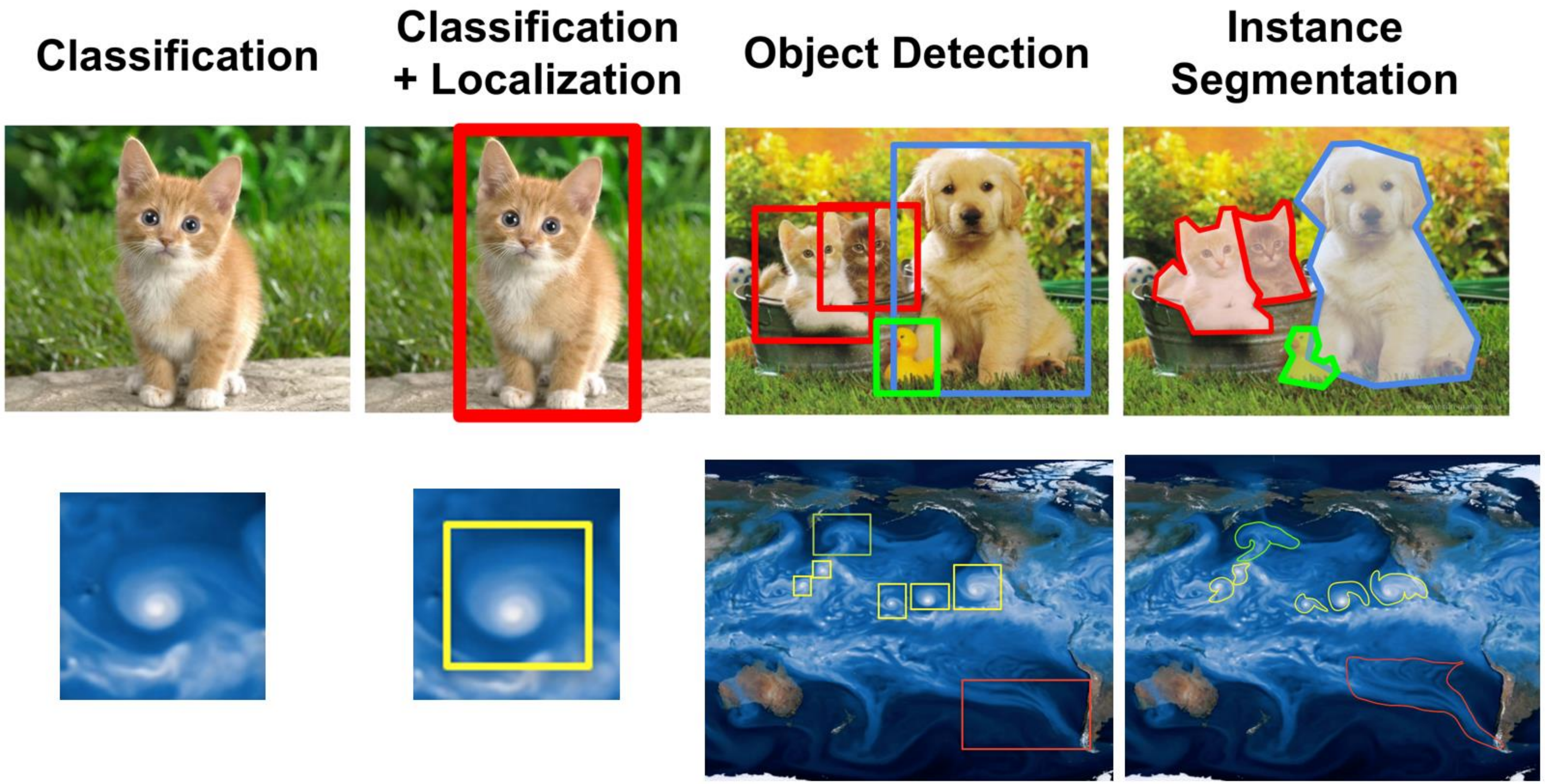


Image Credit: NERSC

CLASSIFICATION

Example: Classifying Land Use

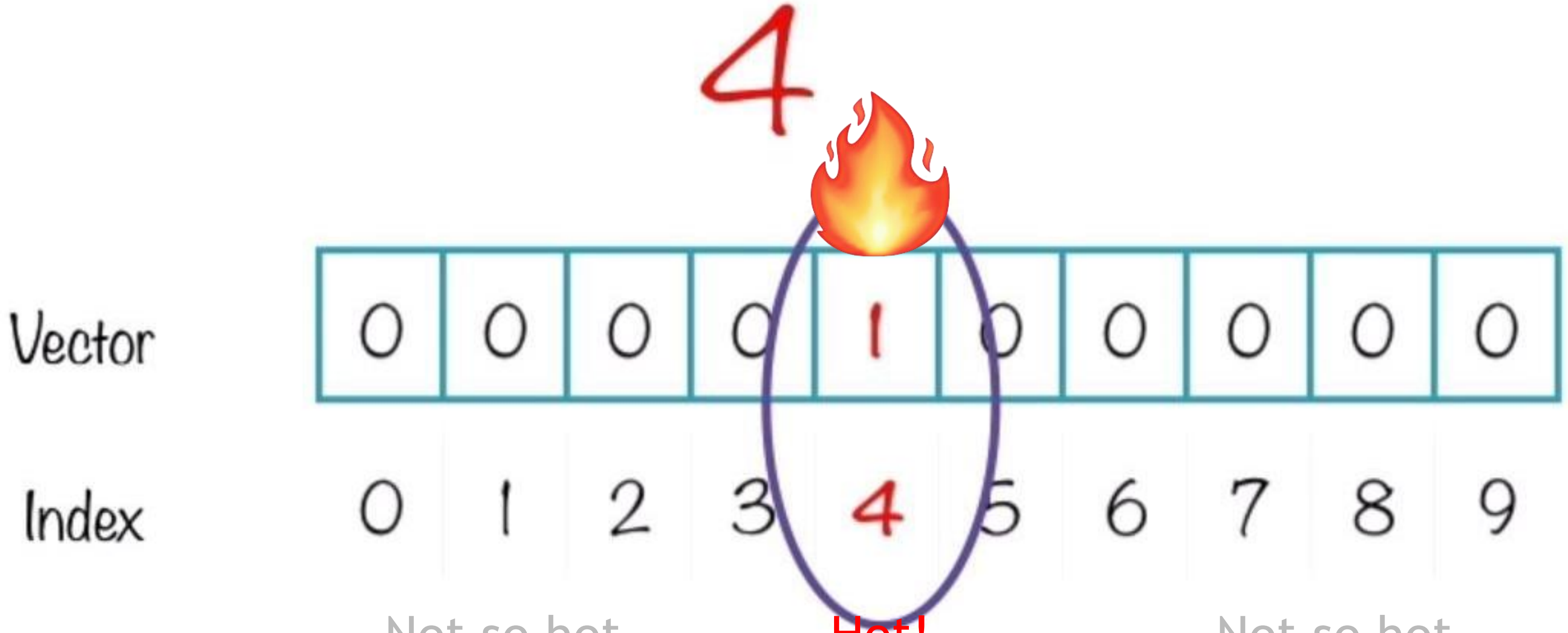
UC Merced Land Use Database

ONE-HOT ENCODING

Input: Pixels, Output: One-hot encoding

INPUT:PIXEL VALUES

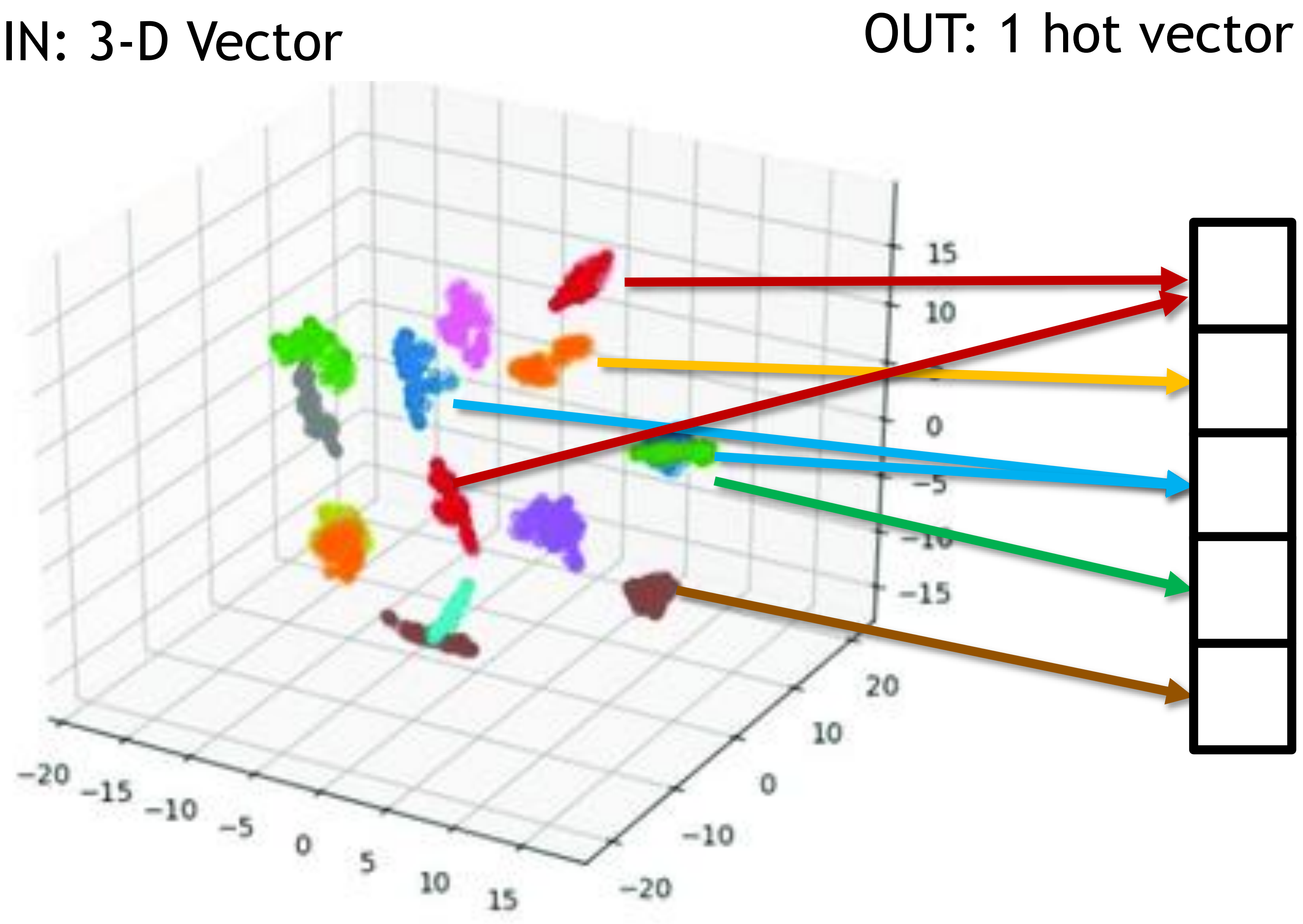
OUTPUT: ONE-HOT VECTOR

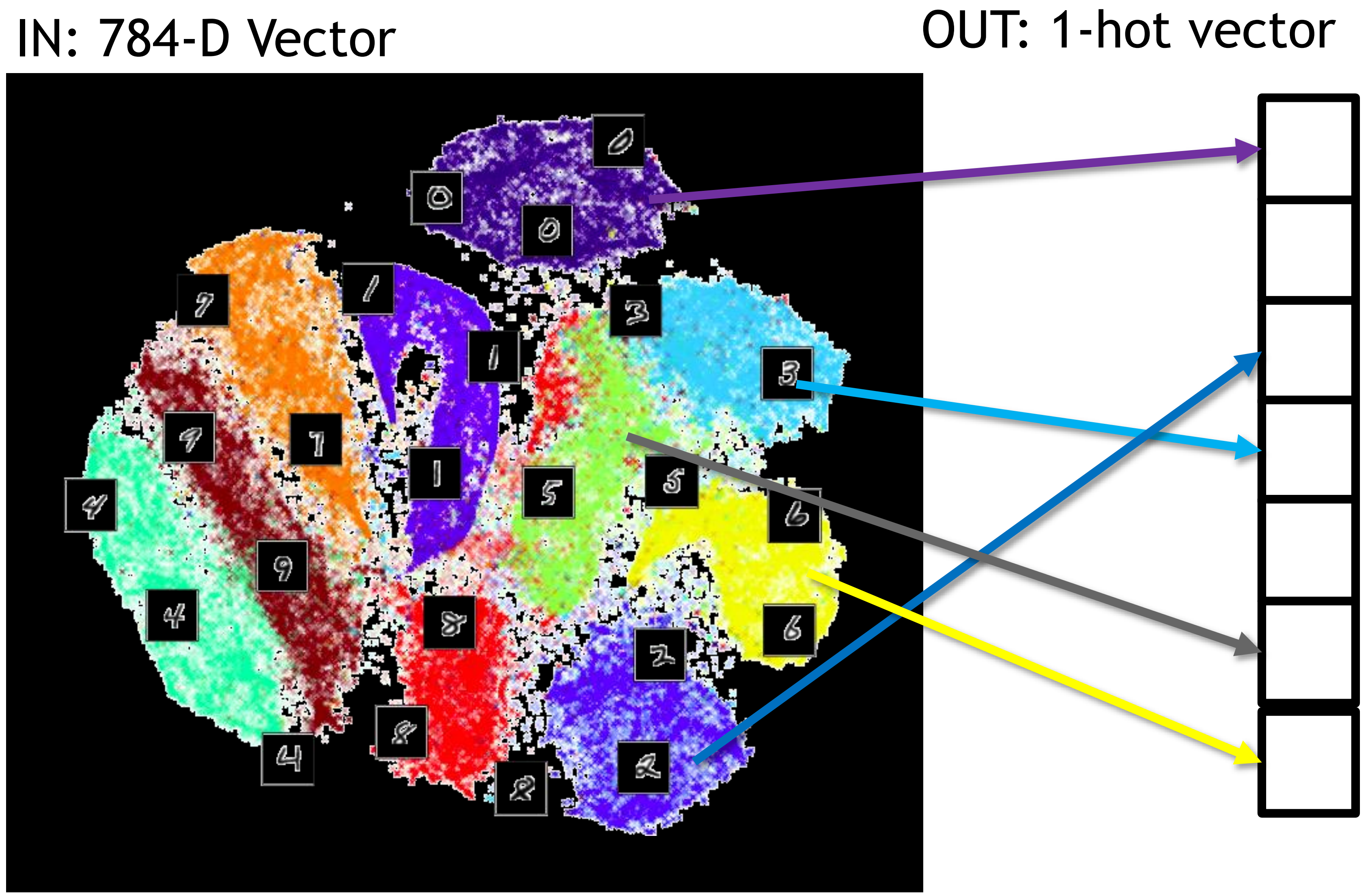


Not so hot Hot! Not so hot

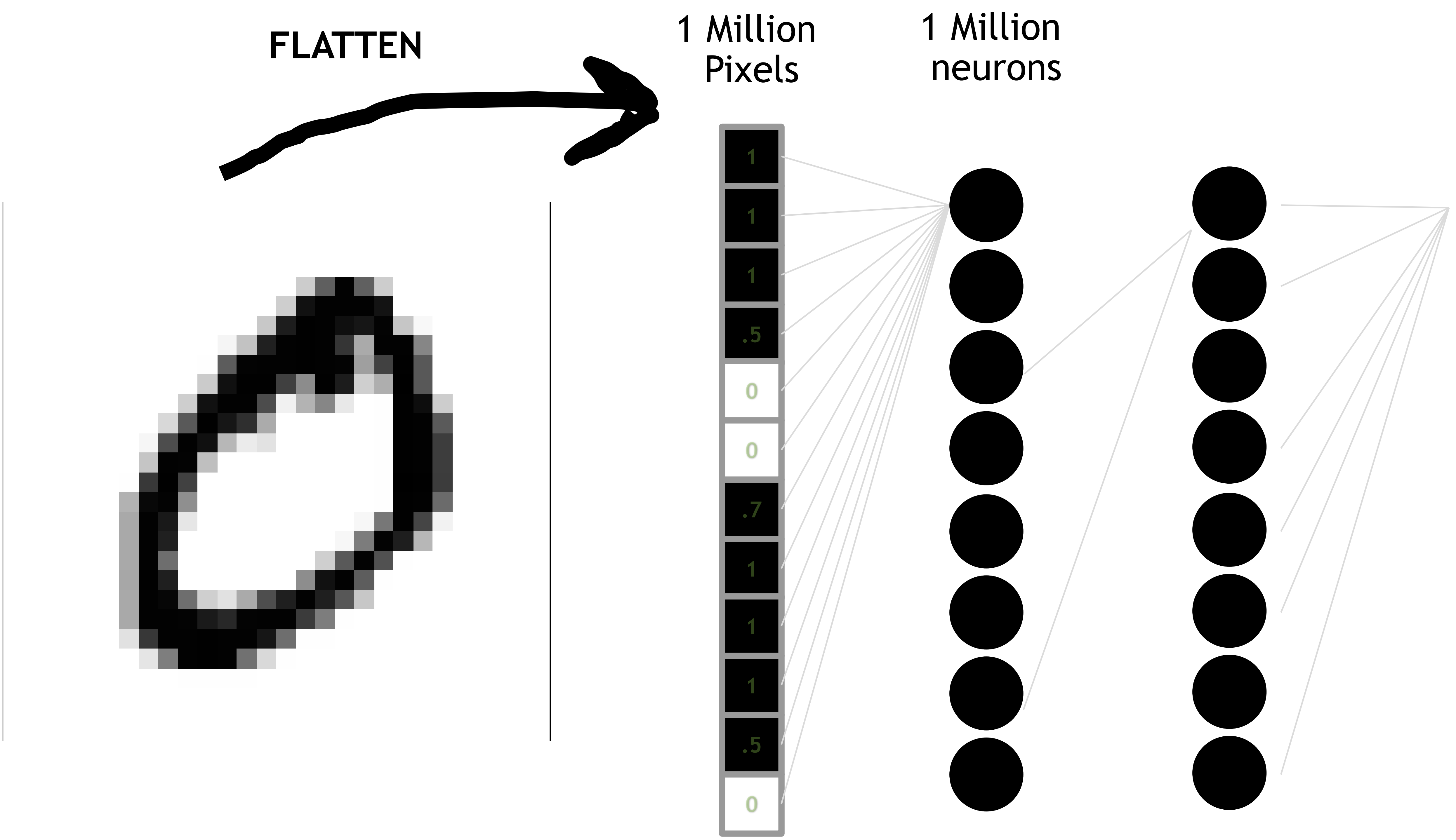
https://blog.carbonteq.com/practical-image-recognition-with-tensorflow/

IMAGES ARE POINTS, WITH MANY DIMENSIONS



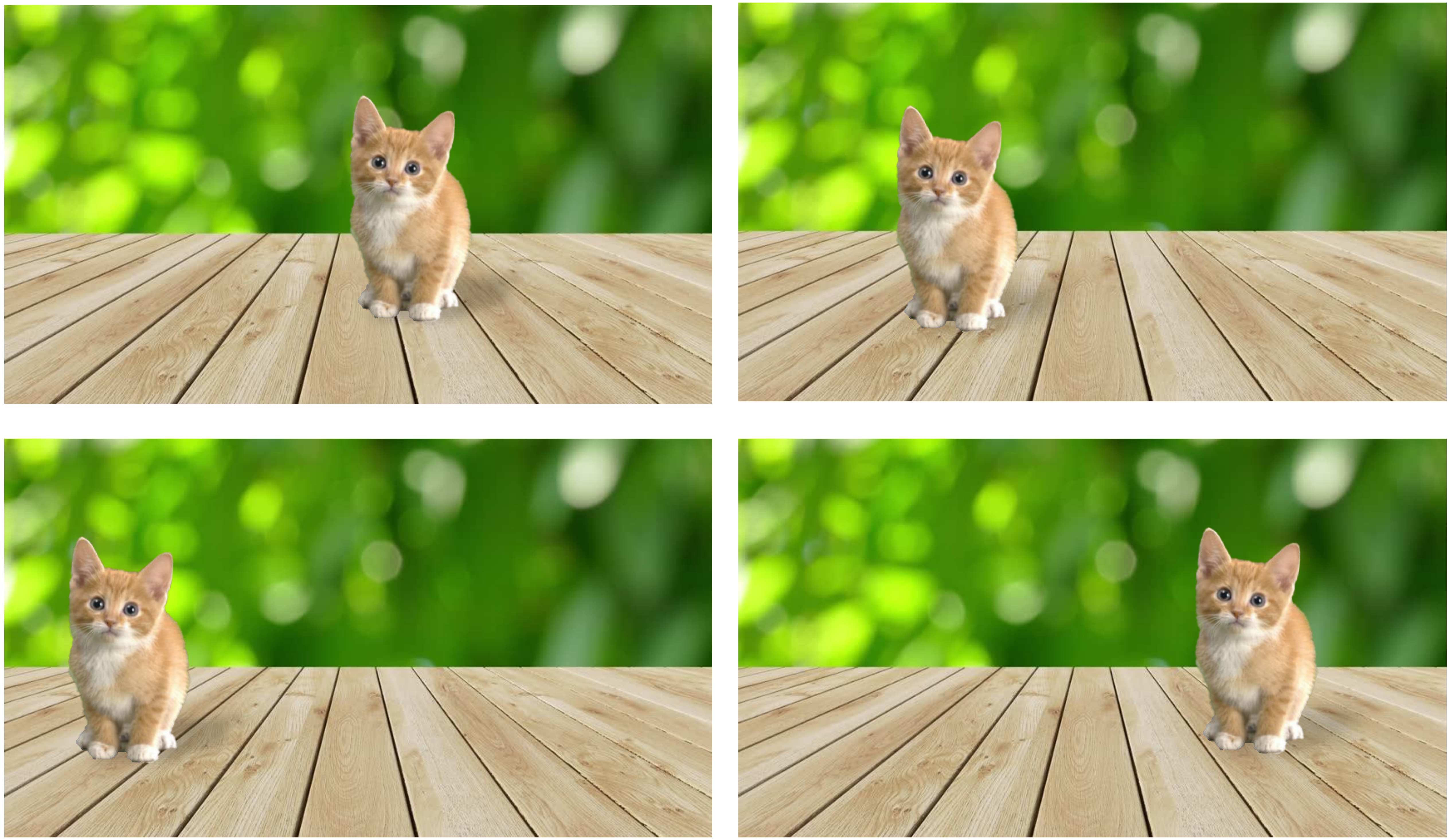


FULLY CONNECTED NETWORKS AND IMAGES DON’T MIX



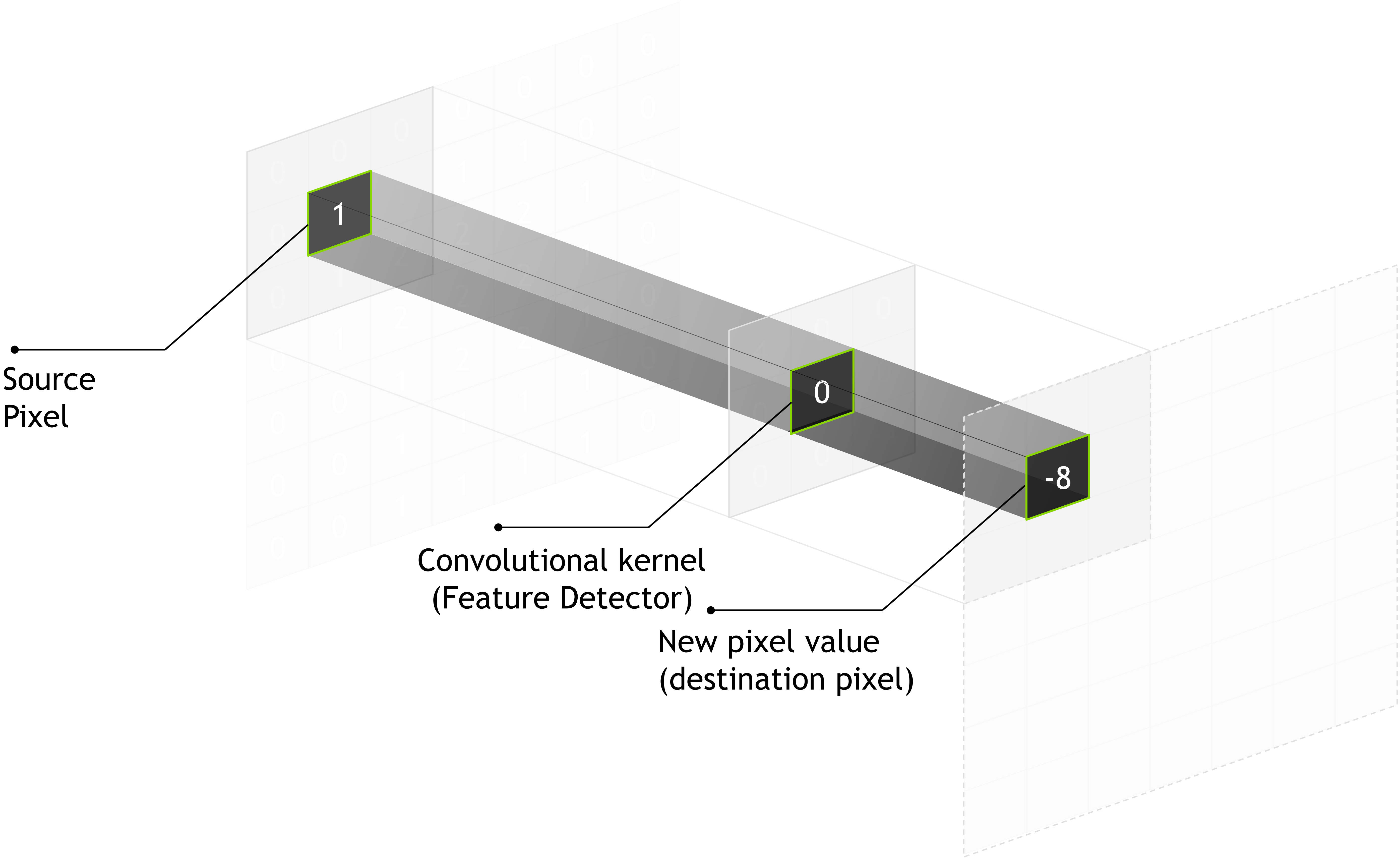
TRANSLATIONAL EQUIVARIANCE

Objects in nature look the same from place to place

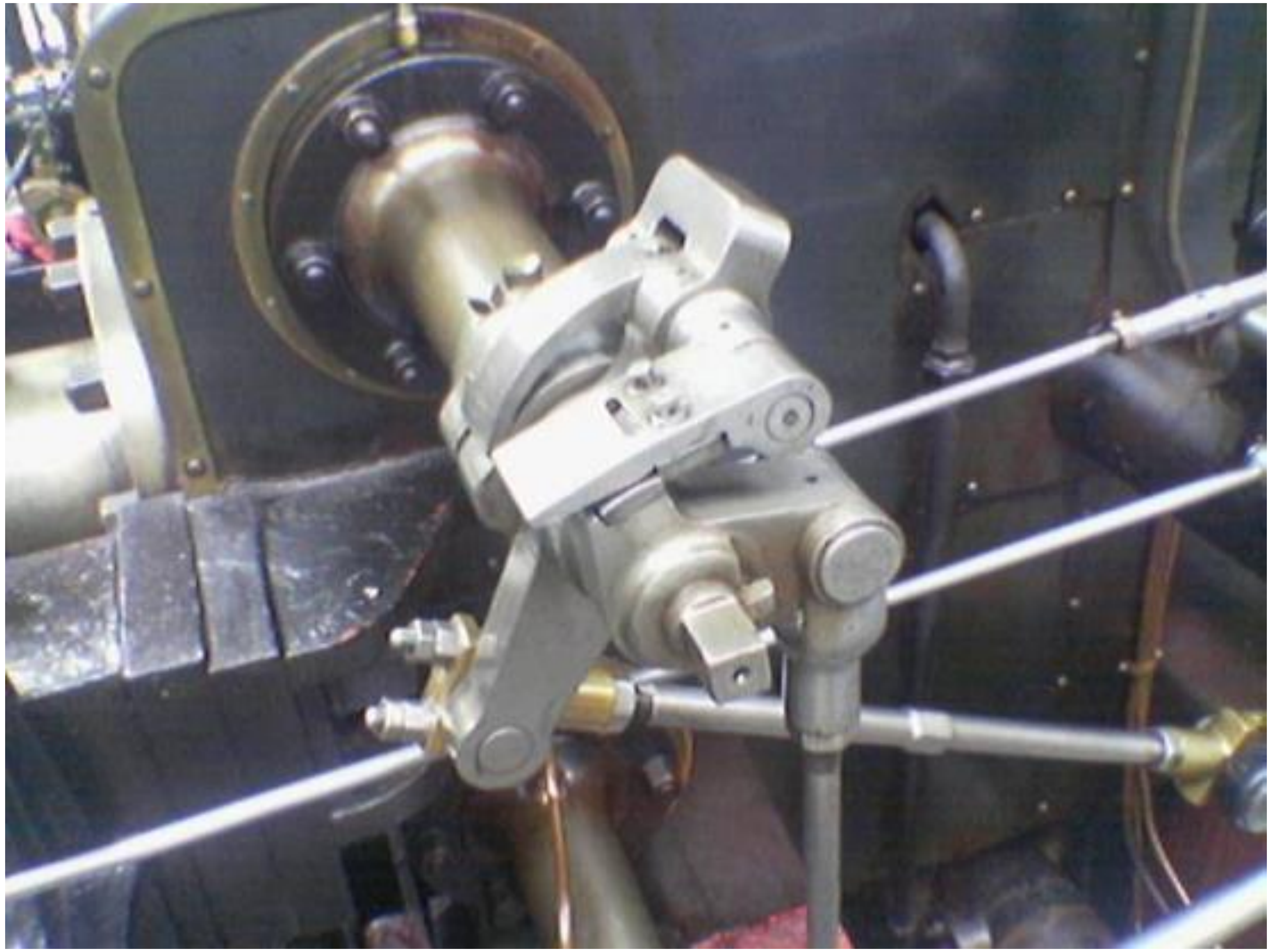


WHAT IS A CONVOLUTION?

A small matrix transformation, applied at each point of the image



CONVOLUTION EXAMPLE: SOBEL FILTER

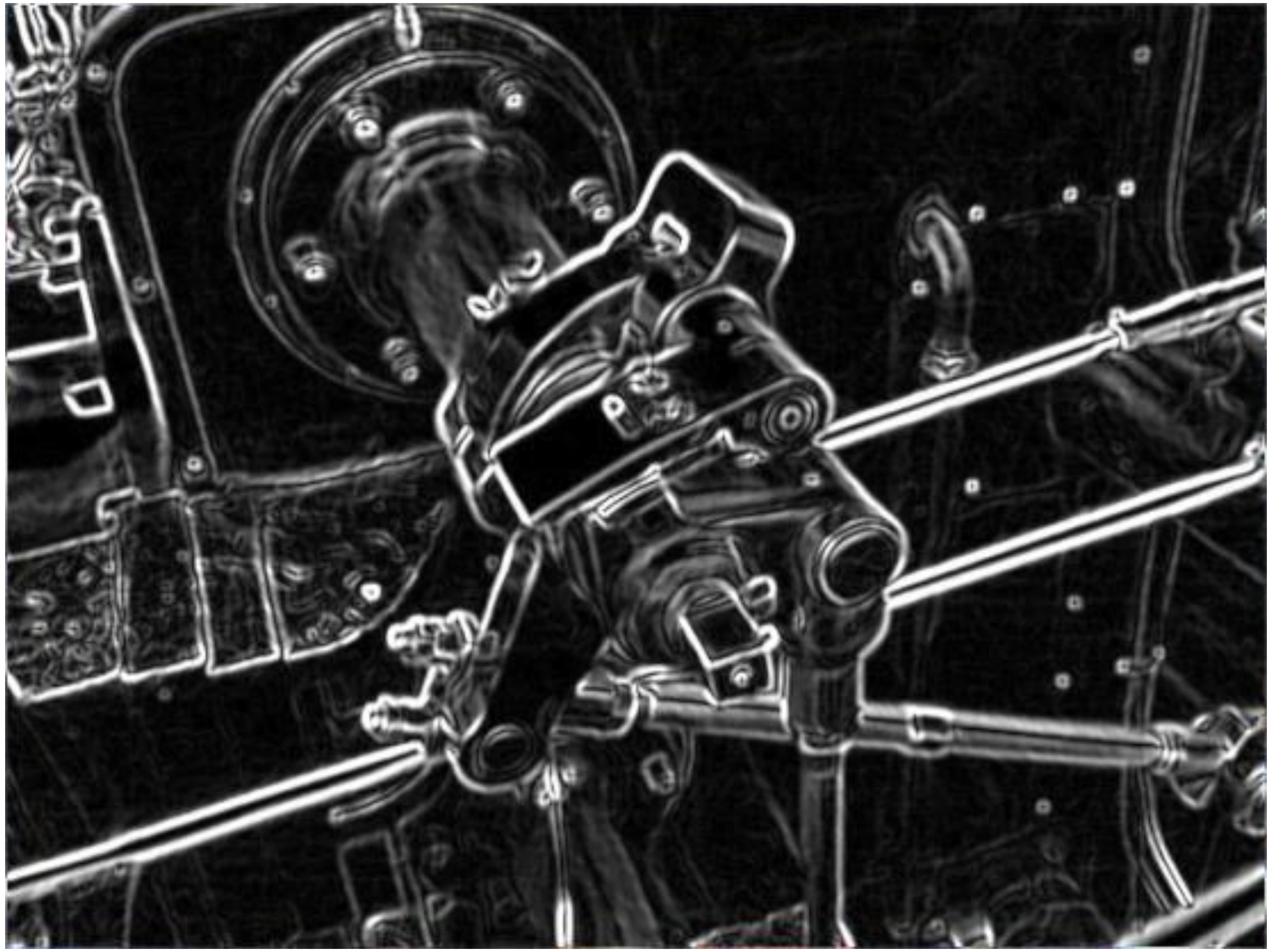


$$G_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}$$

$$G_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}$$

$$G = \sqrt{{G_x}^2 + {G_y}^2}$$

CONVOLUTION EXAMPLE: SOBEL FILTER



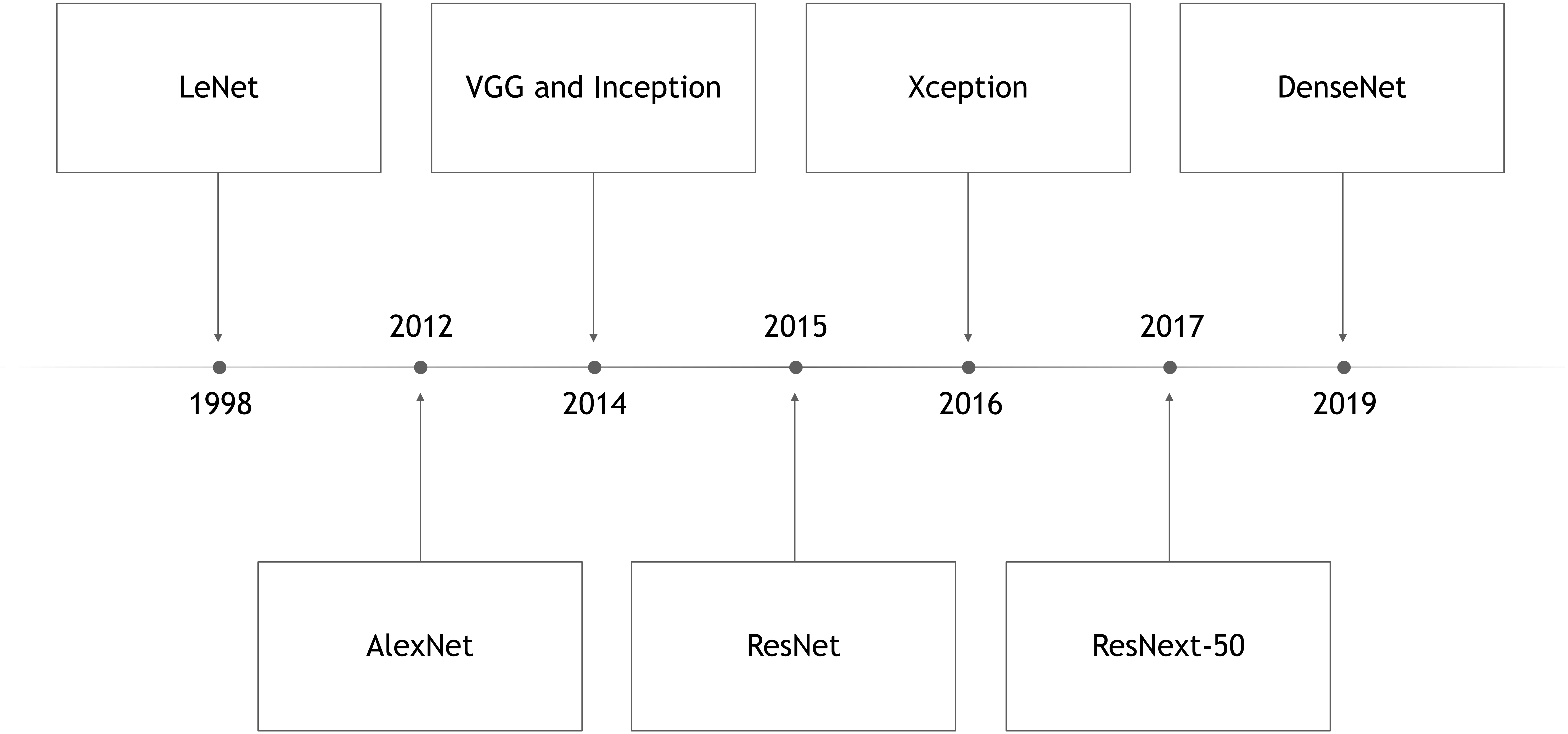
$$G_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}$$

$$G_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}$$

$$G = \sqrt{{G_x}^2 + {G_y}^2}$$

CLASSIFICATION

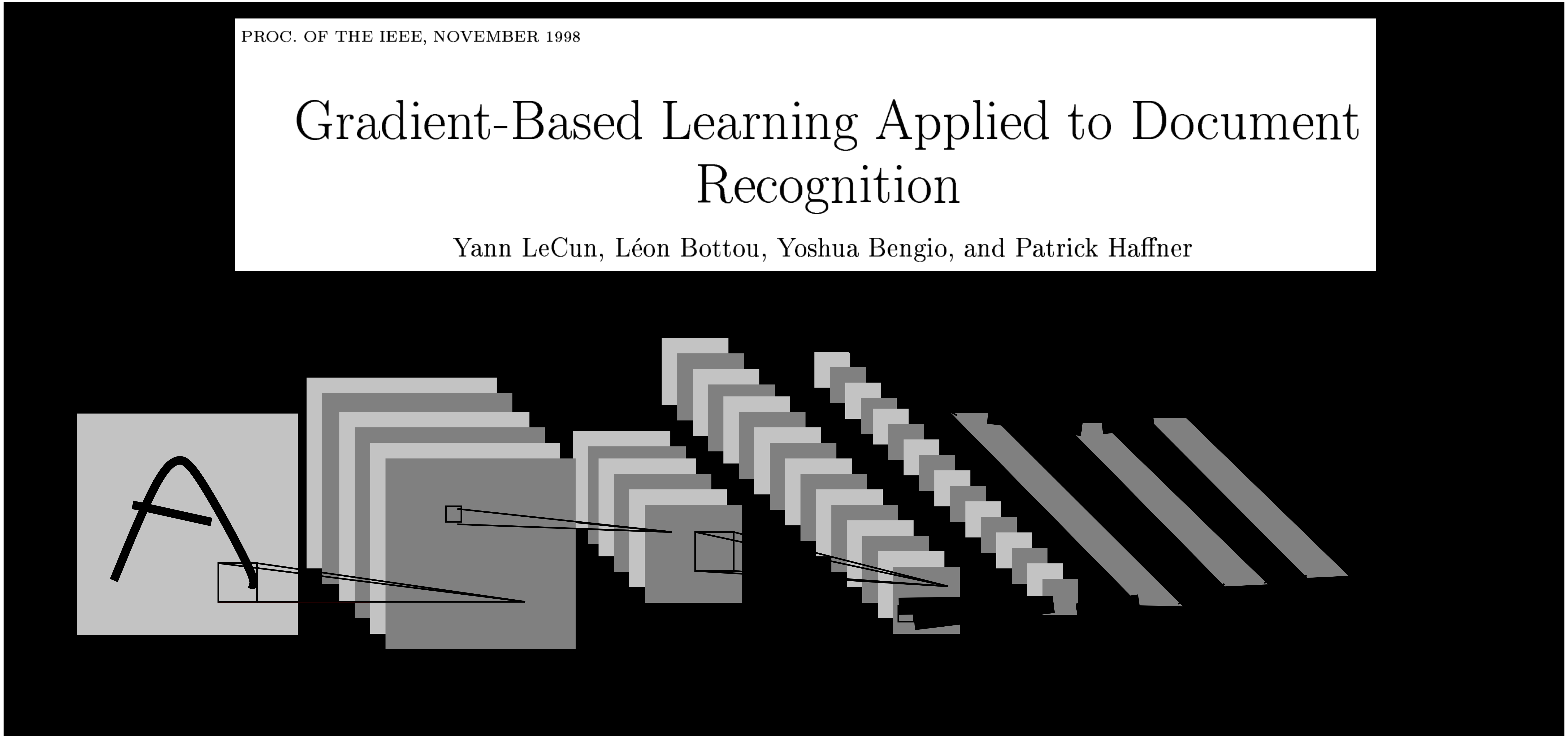
CLASSIFIER EVOLUTION OVER TIME



https://towardsdatascience.com/illustrated-10-cnn-architectures-95d78ace614d

LENET-5

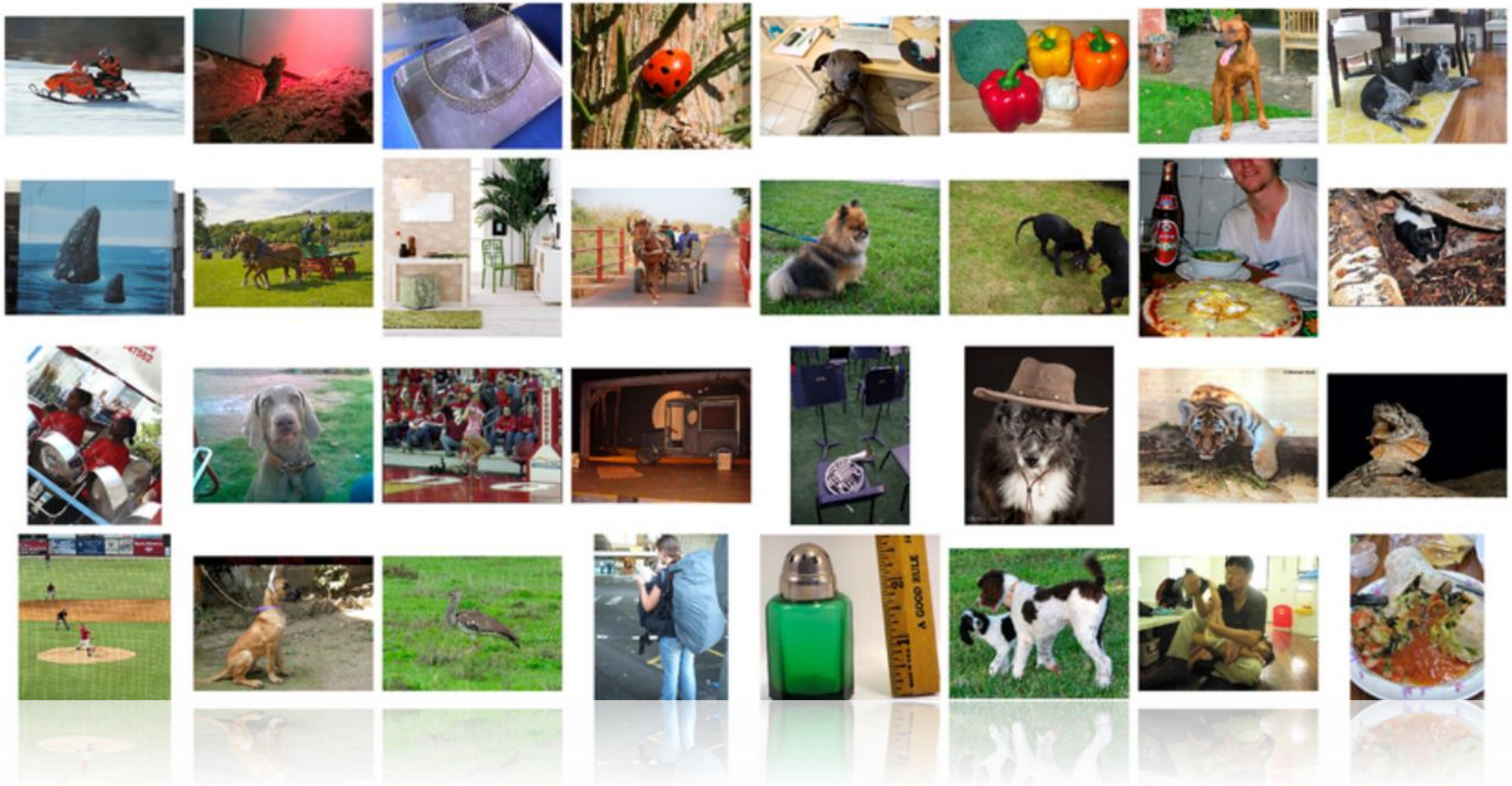
(1988) Yann LeCun. Hand written recognition. 60k parameters.



https://en.wikipedia.org/wiki/LeNet.

IMAGENET ILSVR COMPETITION

Large Scale Visual Recognition Competition (2010-2017)



https://en.wikipedia.org/wiki/ImageNet

ALEXNET

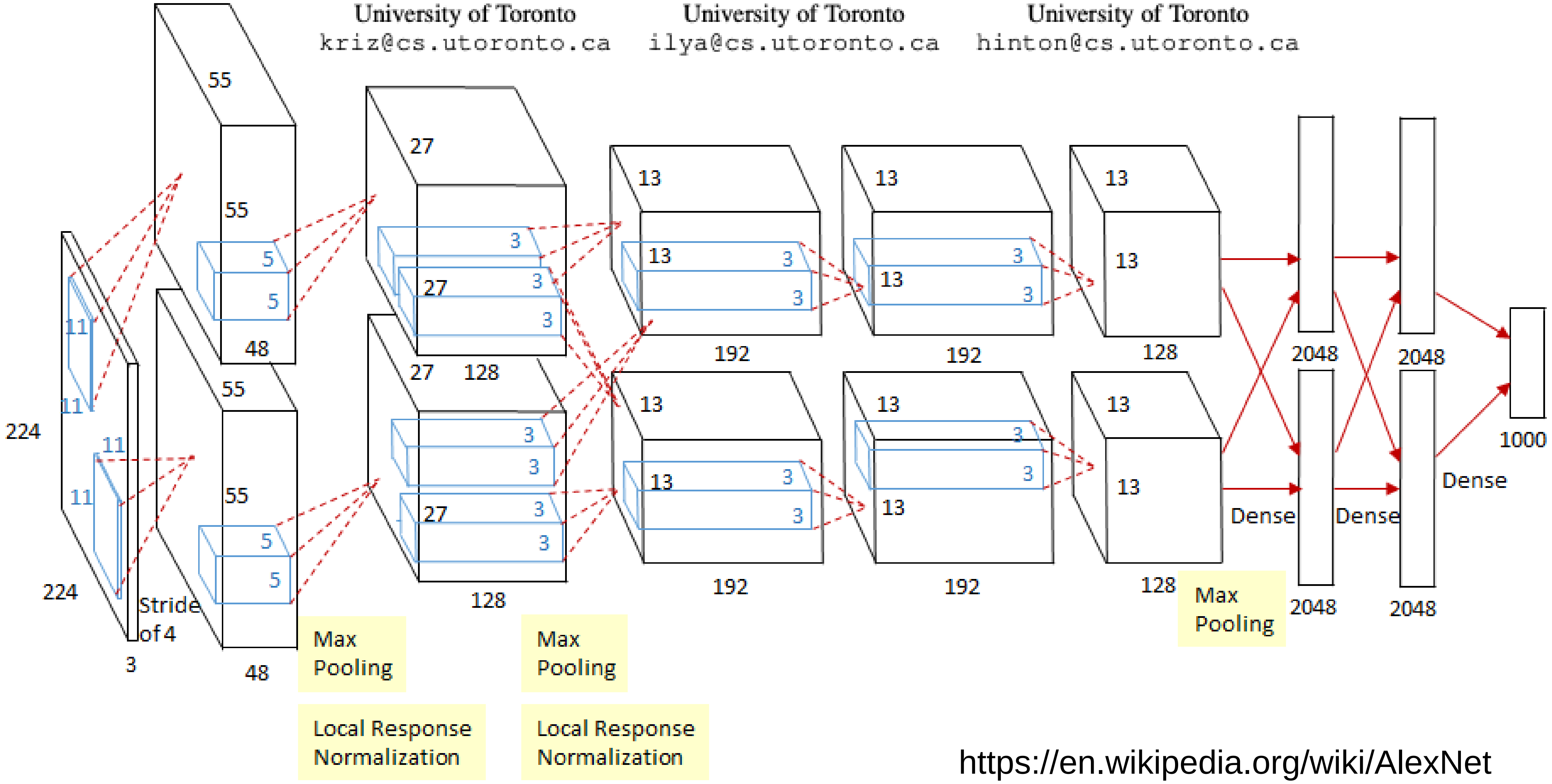
(2012): Krizevsky, Sutskever, Hinton. ImageNet winner.

ImageNet Classification with Deep Convolutional Neural Networks

Alex Krizhevsky
University of Toronto
kriz@cs.utoronto.ca

Ilya Sutskever
University of Toronto
ilya@cs.utoronto.ca

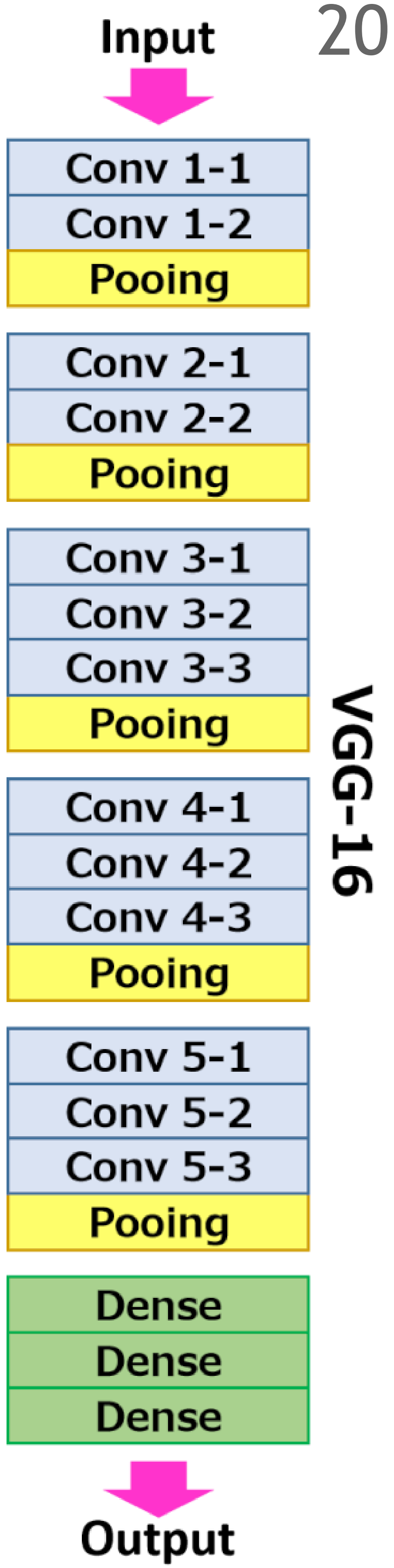
Geoffrey E. Hinton
University of Toronto
hinton@cs.utoronto.ca

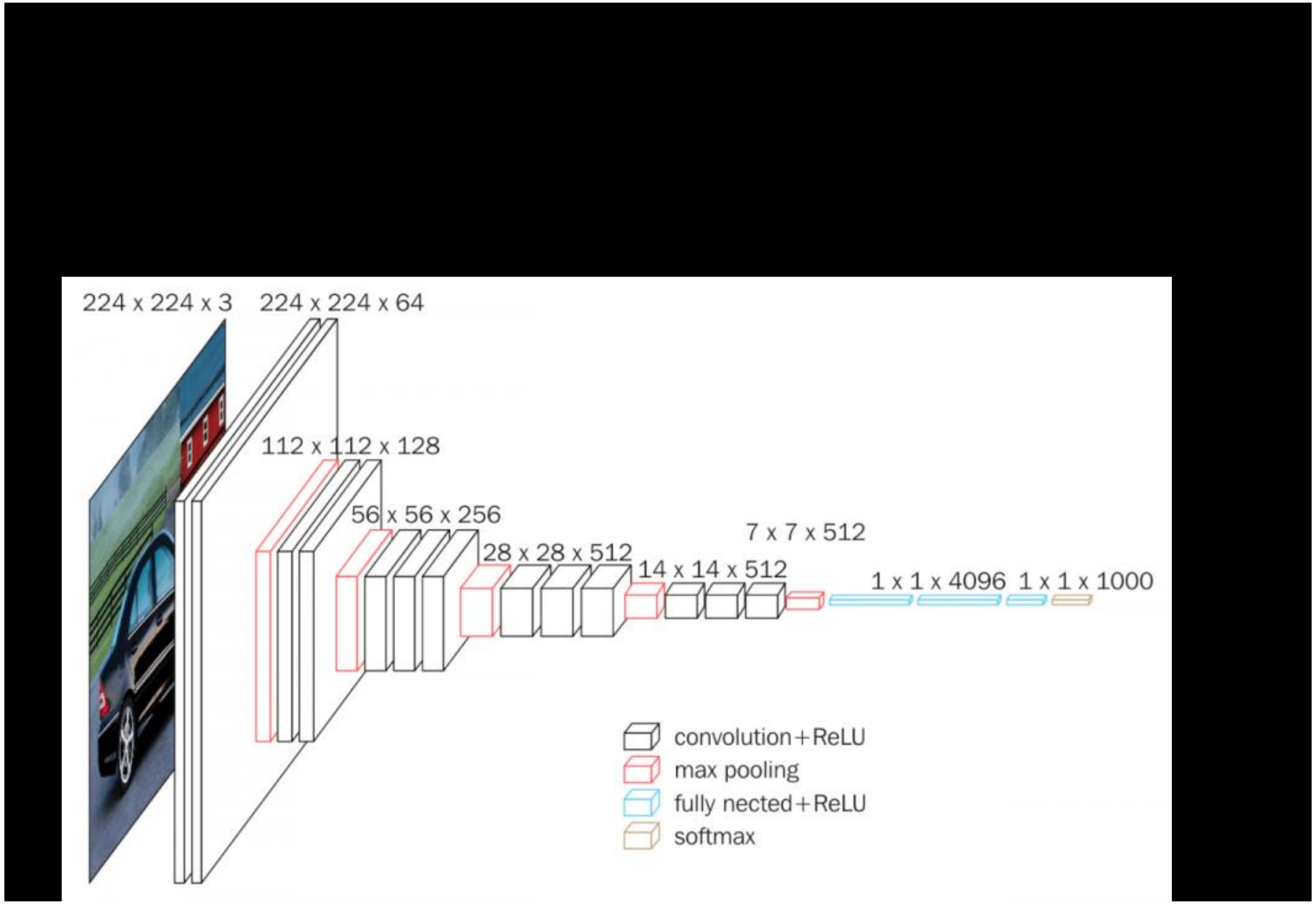


https://en.wikipedia.org/wiki/AlexNet

VGG-16

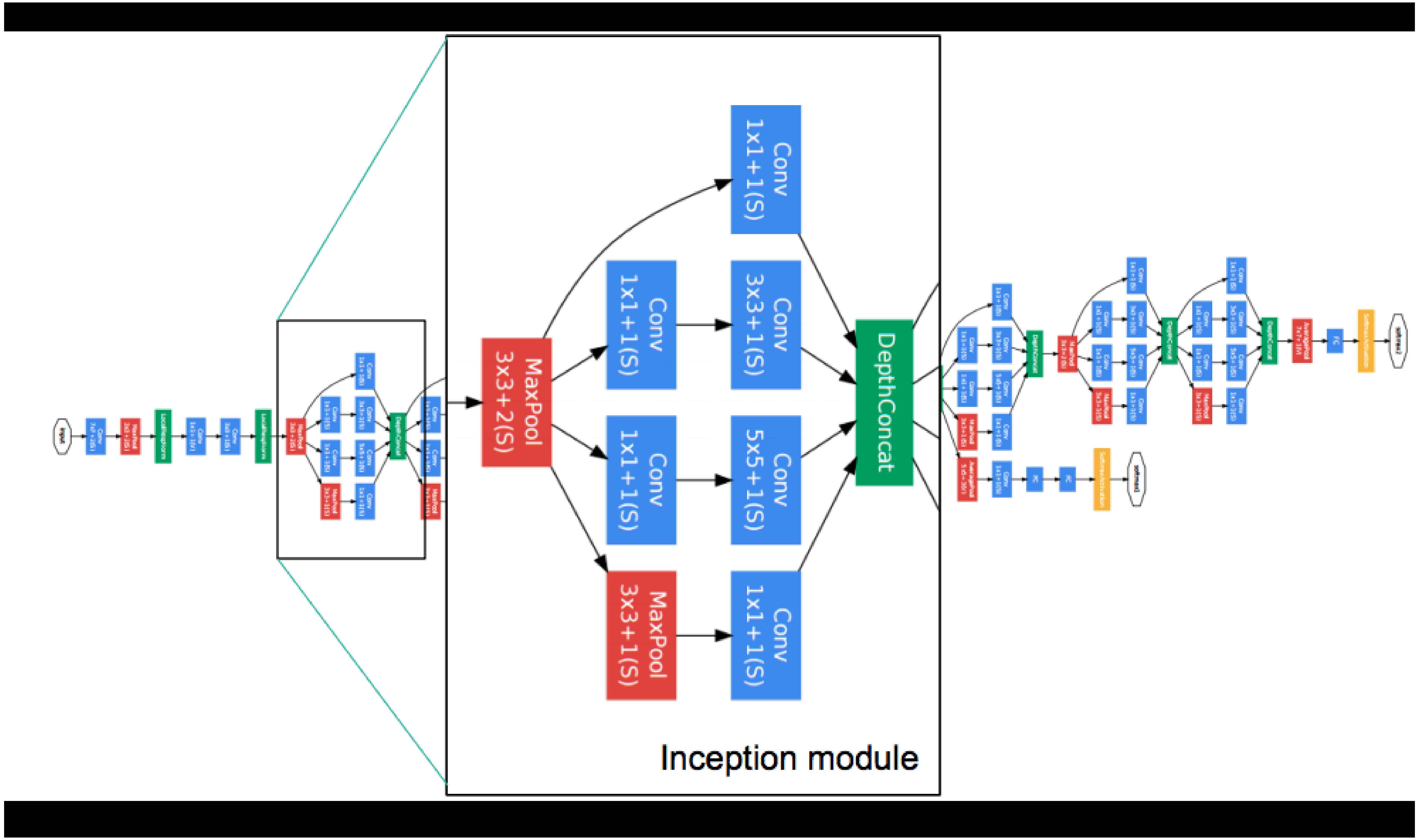
2014. ImageNet runner up. Simple, clean architecture.





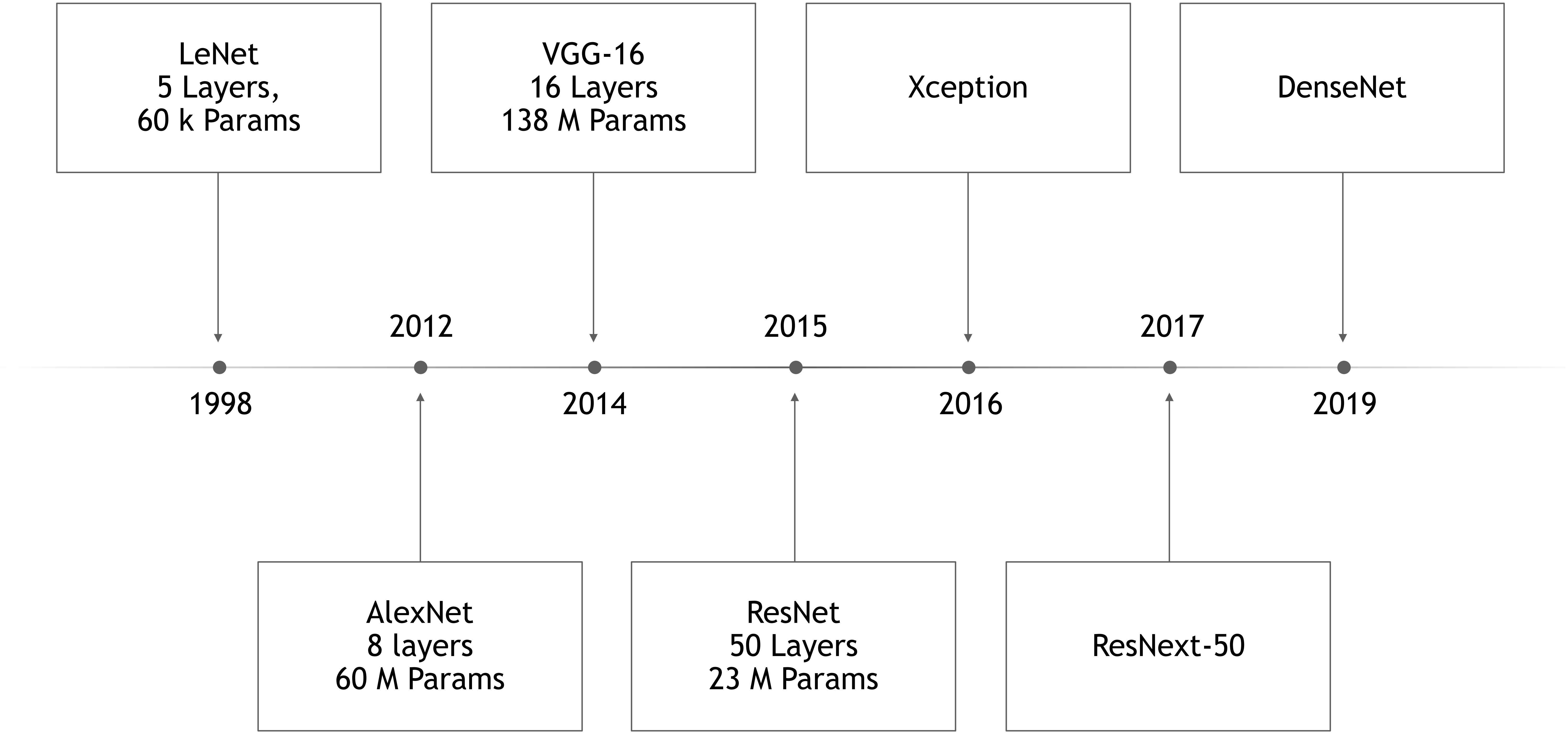
INCEPTION-V1 (GOOGLENET)

2014. Train different size convolutions in parallel



RESNETS

MODELING TRENDS: DEEPER AND LARGER

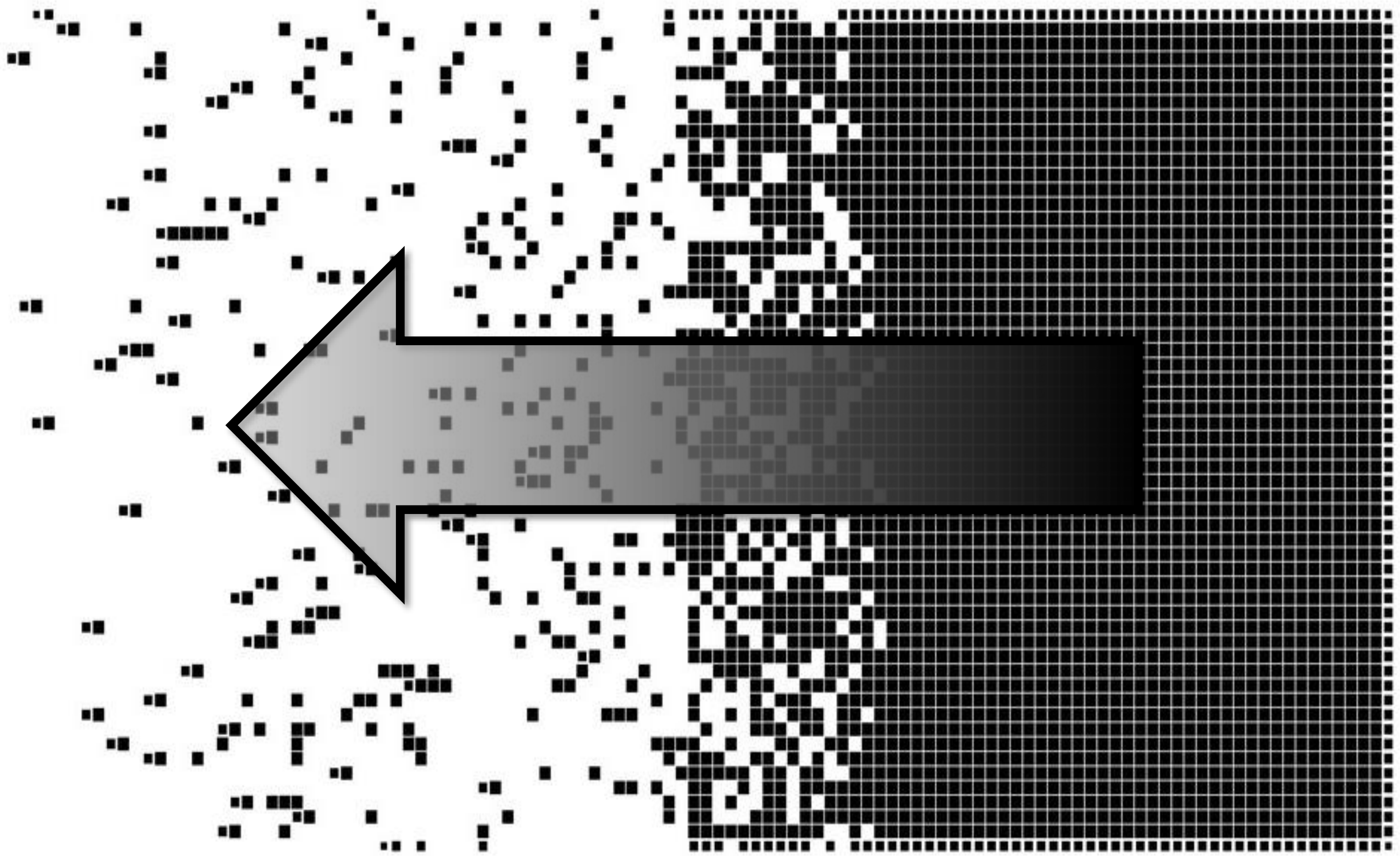


Source: Source information is 14 pt, italic

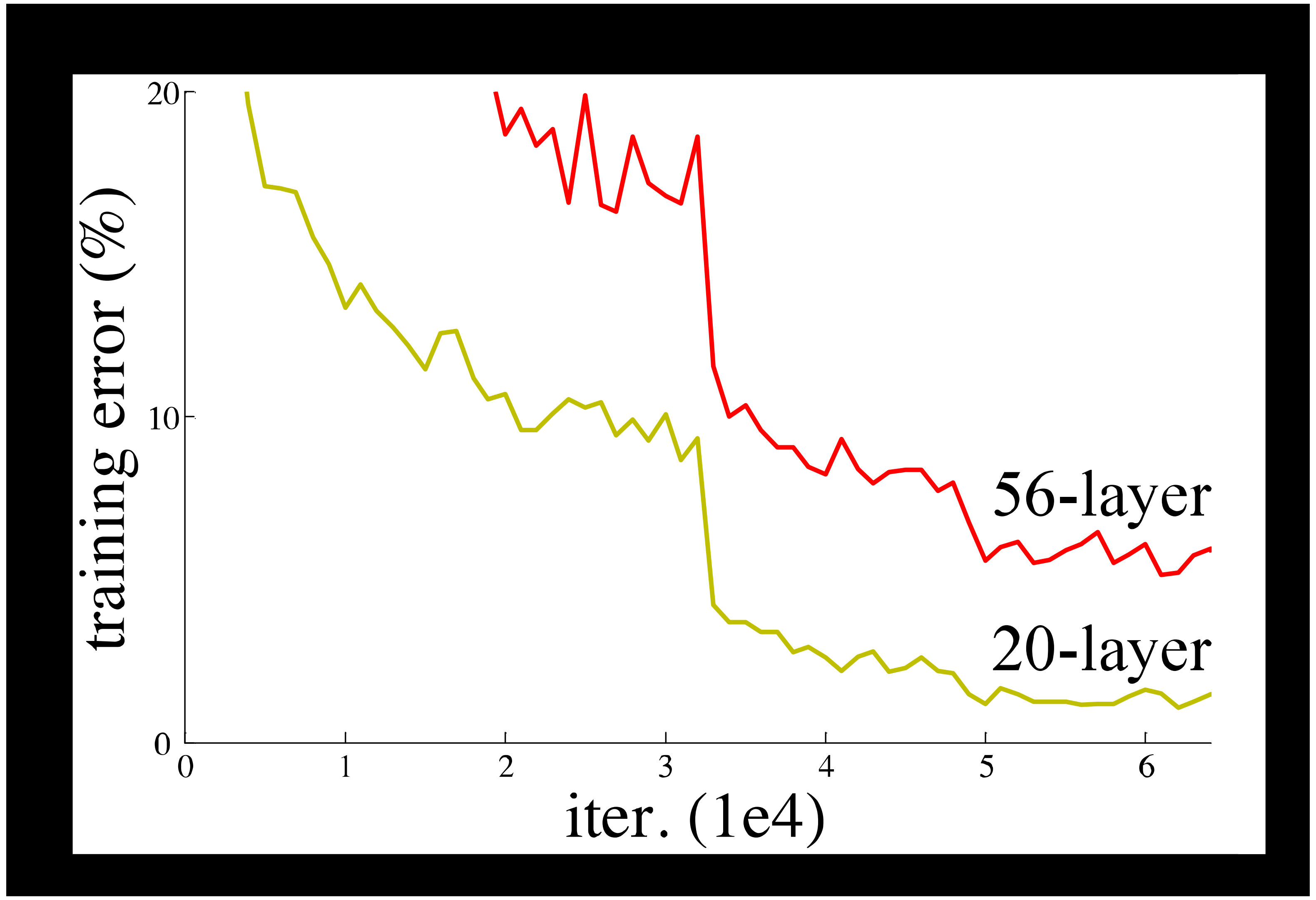
PROBLEM: VANISHING GRADIENTS

Error signal decays exponentially as it propagates backward through the network

ERROR SIGNAL VANISHES DURING BACKPROP



DEEPER NETWORKS WERE HARDER TO TRAIN

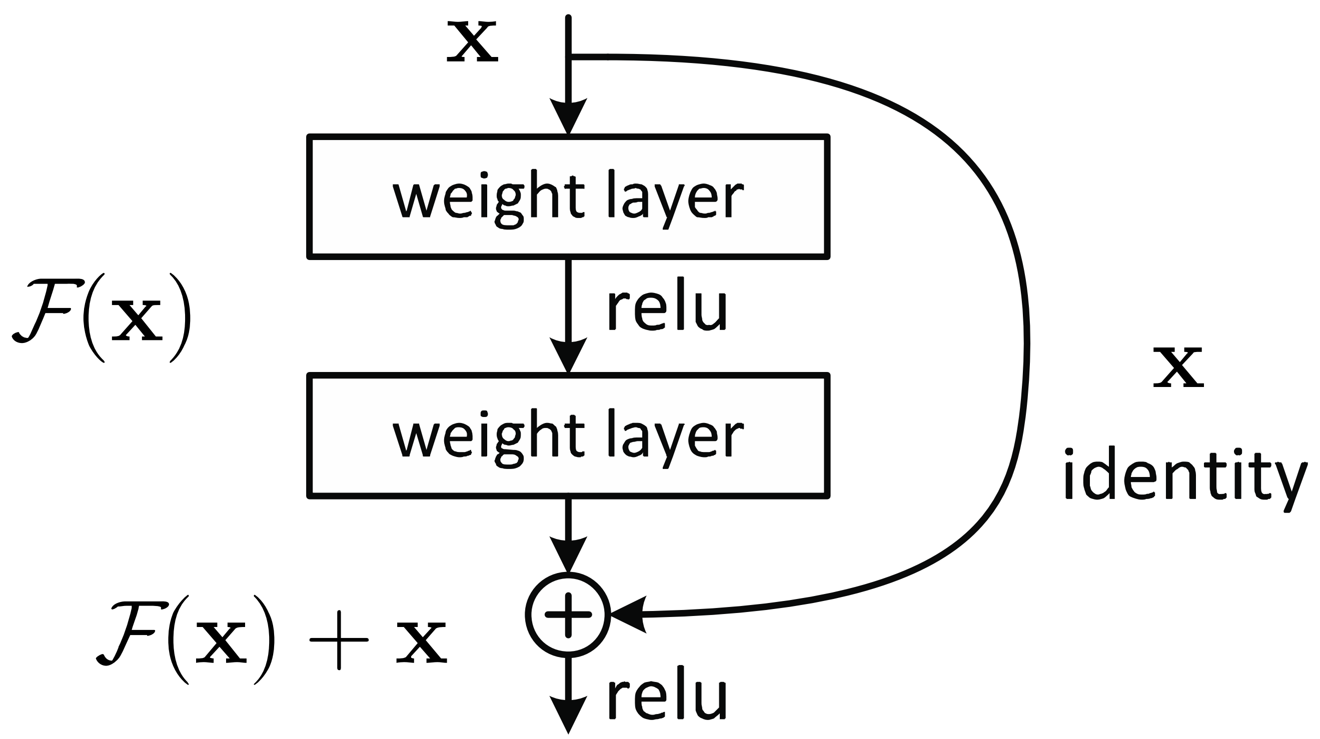


https://www.arxiv-vanity.com/papers/1512.03385/

RESNETS AND SKIP CONNECTIONS

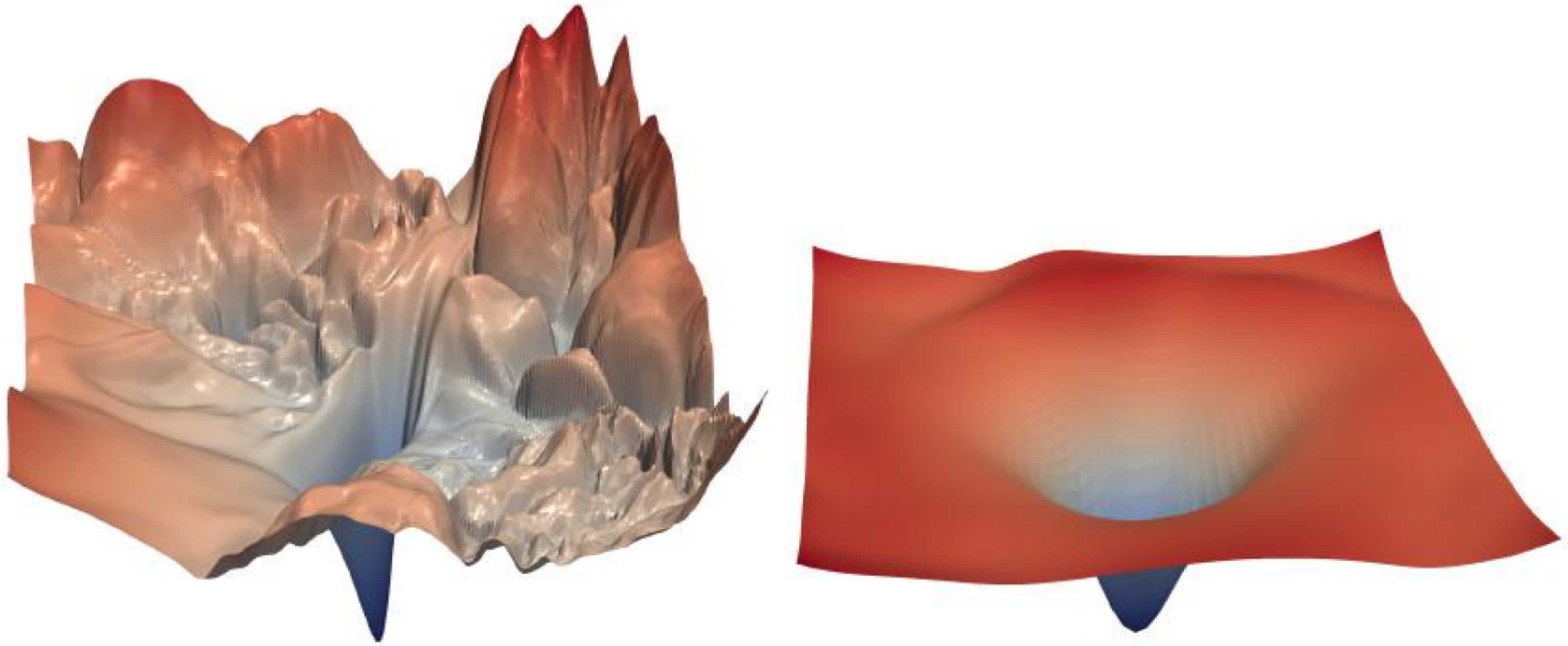
(aka Highway Networks)

ADD THE INPUT TO OUTPUT



https://arxiv.org/pdf/1512.03385.pdf

DRAMTICALLY SIMPLIFIES THE LOSS LANDSCAPE



(a) without skip connections

(b) with skip connections

https://arxiv.org/abs/1712.09913

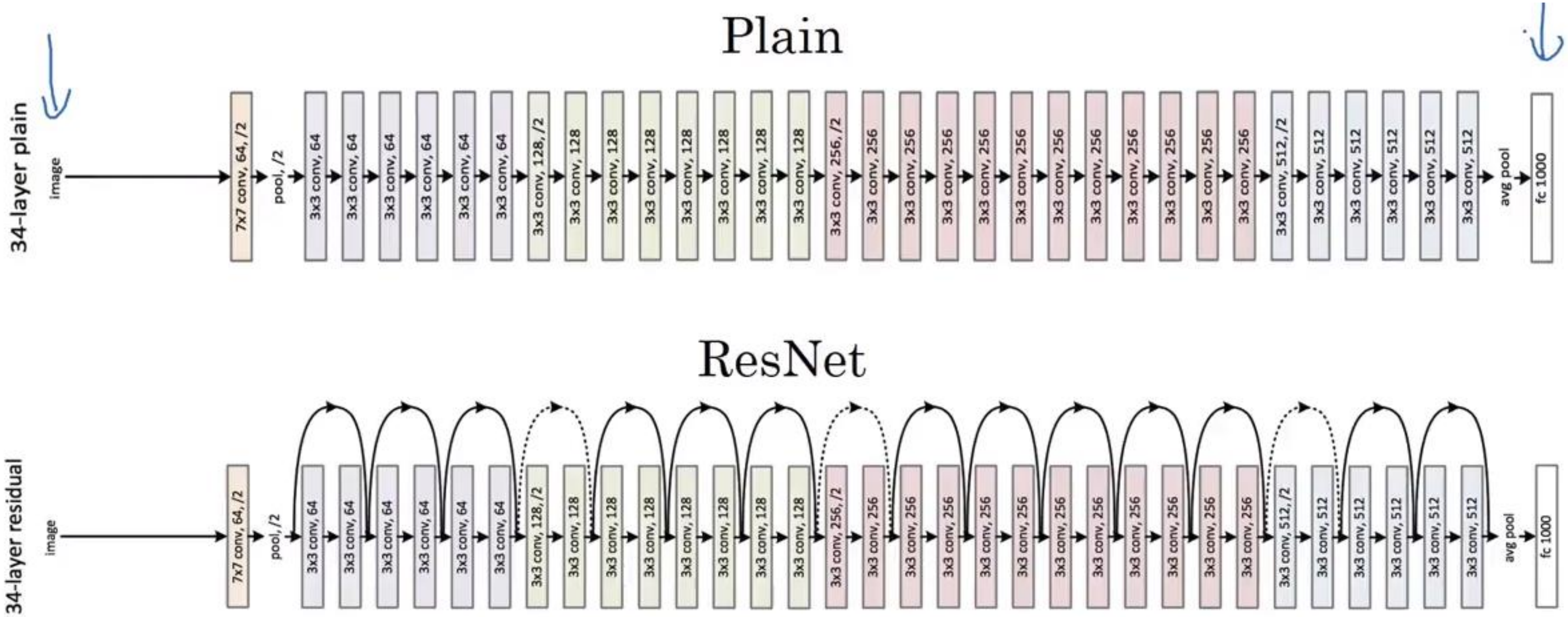
https://jithinjk.github.io/blog/nn_loss_visualized.md.html

RESNET-50

2015 Microsoft Research. 50 Layers, 23M params.

Deep Residual Learning for Image Recognition

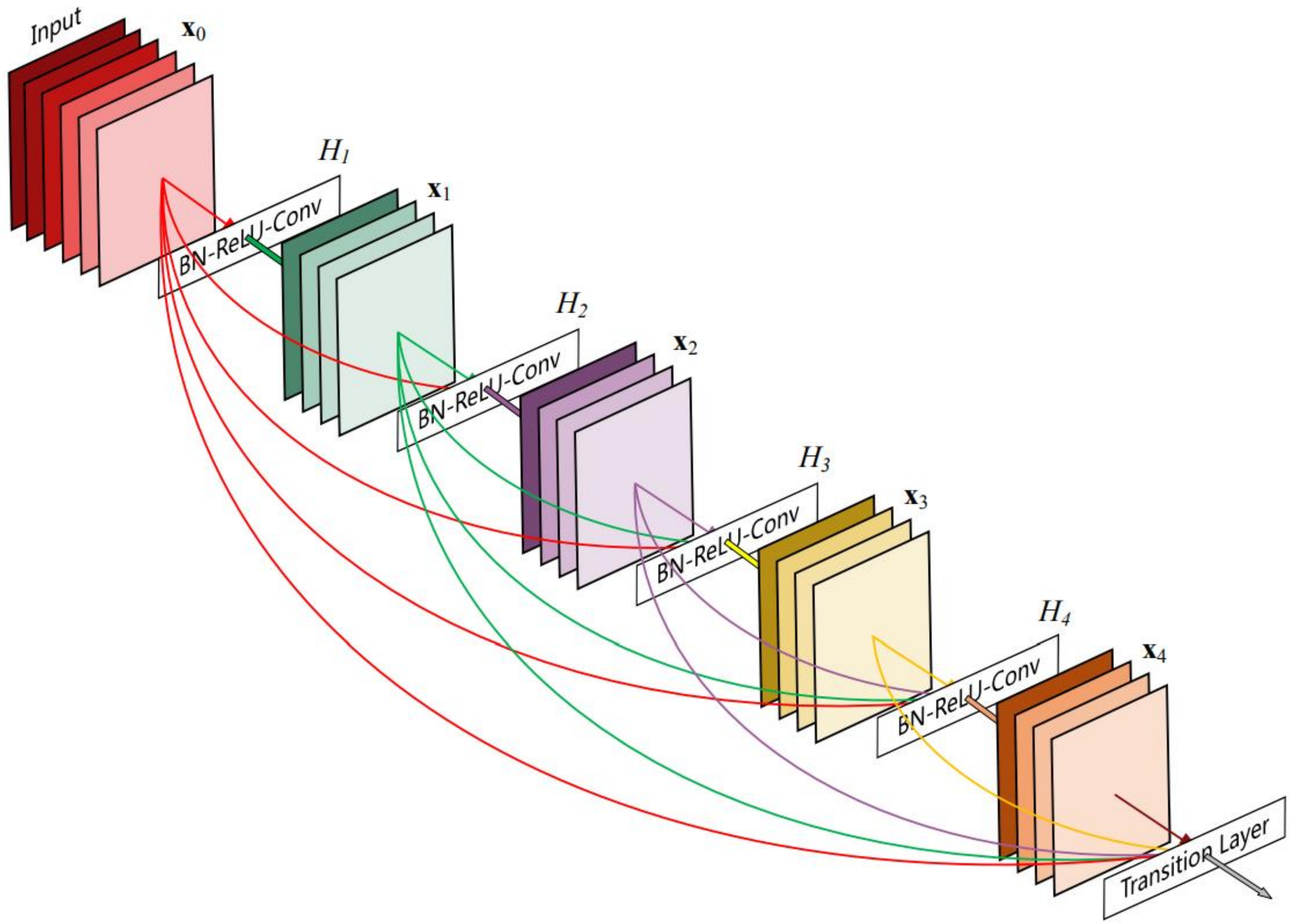
Kaiming He Xiangyu Zhang Shaoqing Ren Jian Sun
Microsoft Research

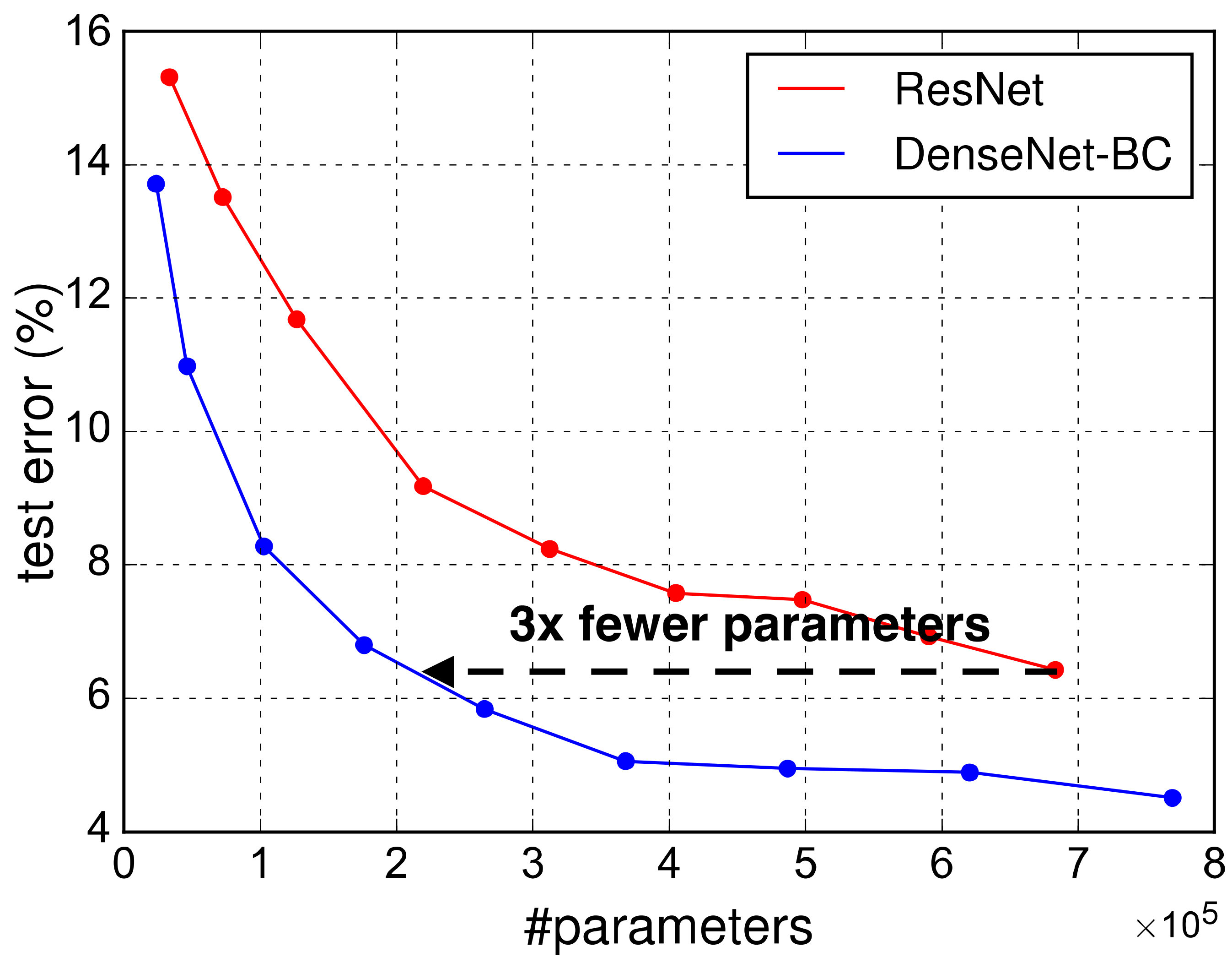


https://arxiv.org/pdf/1409.1556.pdf

DENSENET

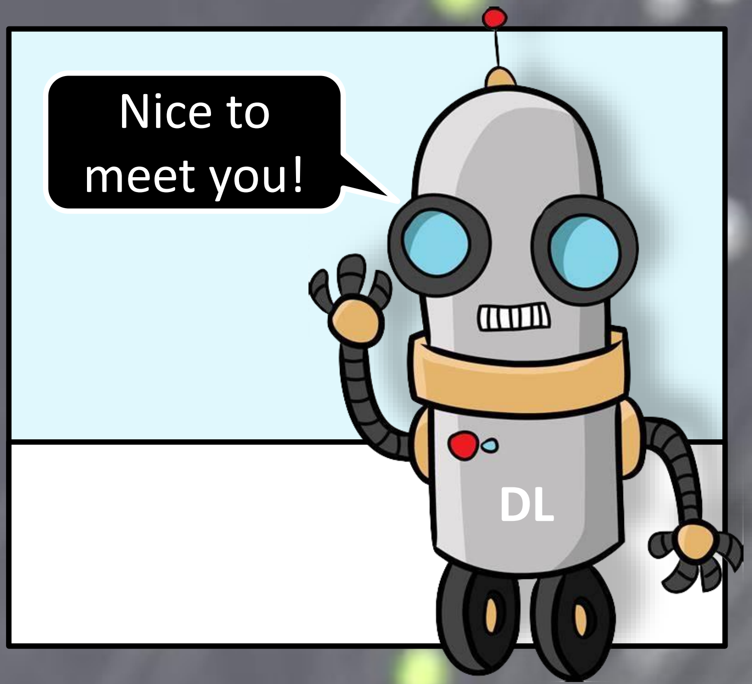
2017



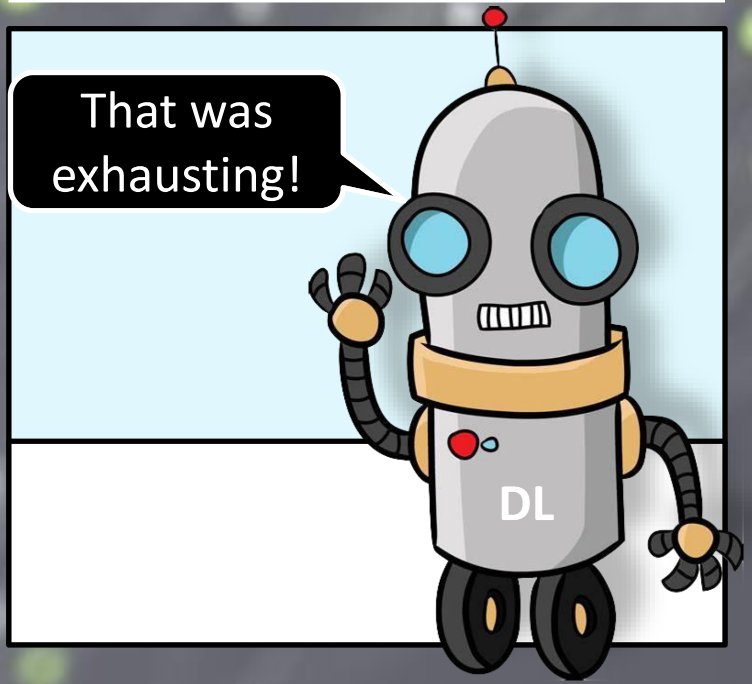


https://arxiv.org/abs/1608.06993

INTRO TO DL, PART 1

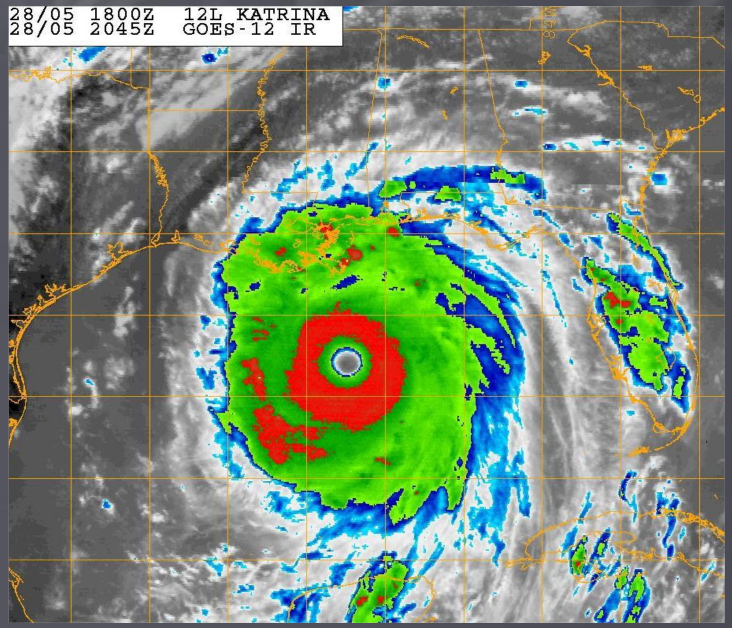


INTRO TO DL, PART 2

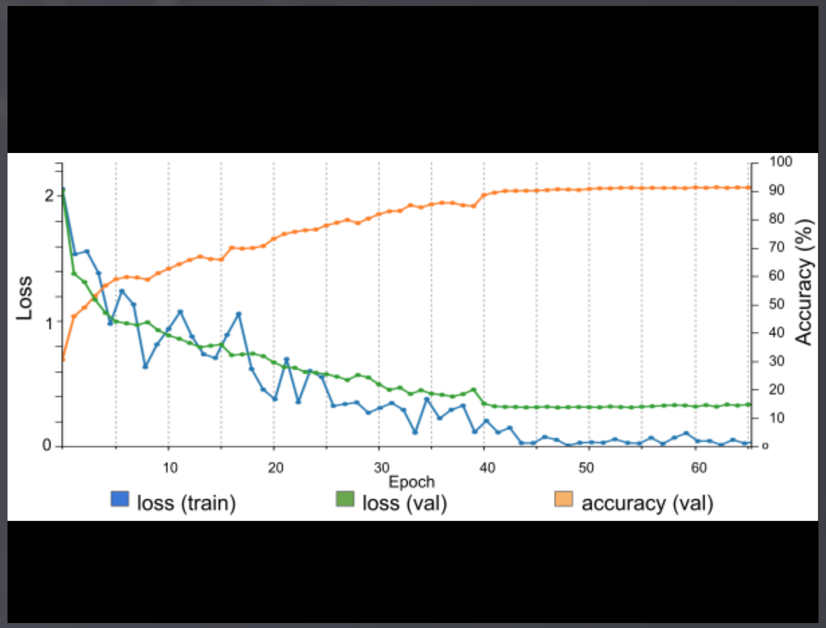


LAB 1: CNNs AND KERAS

LAB 2: TROPICAL CYCLONES



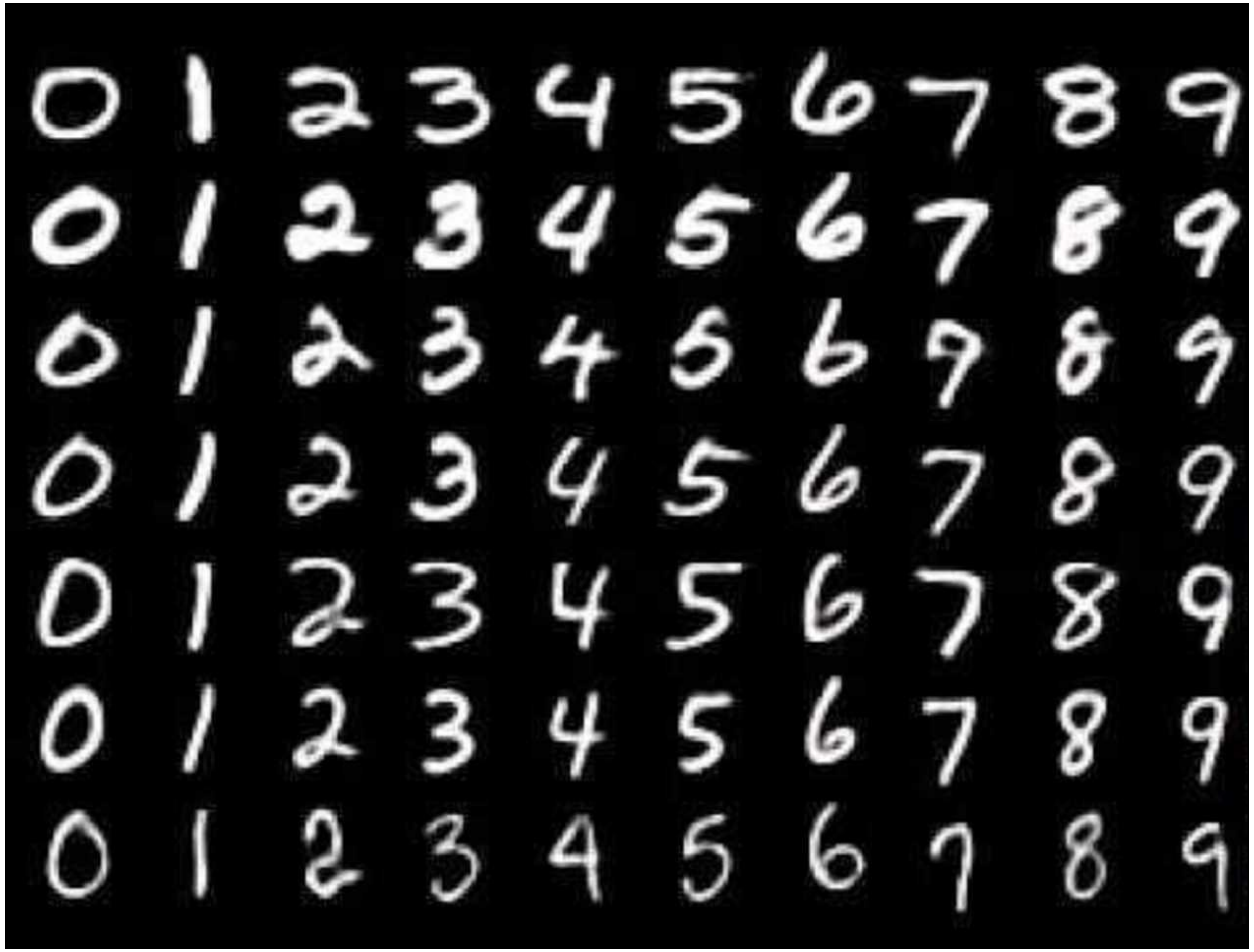
LAB 3: CFD STEADY FLOW



LAB PART 1
CNN AND KERAS 101

MNIST

The standard 'hello world' problem for deep learning



MNIST

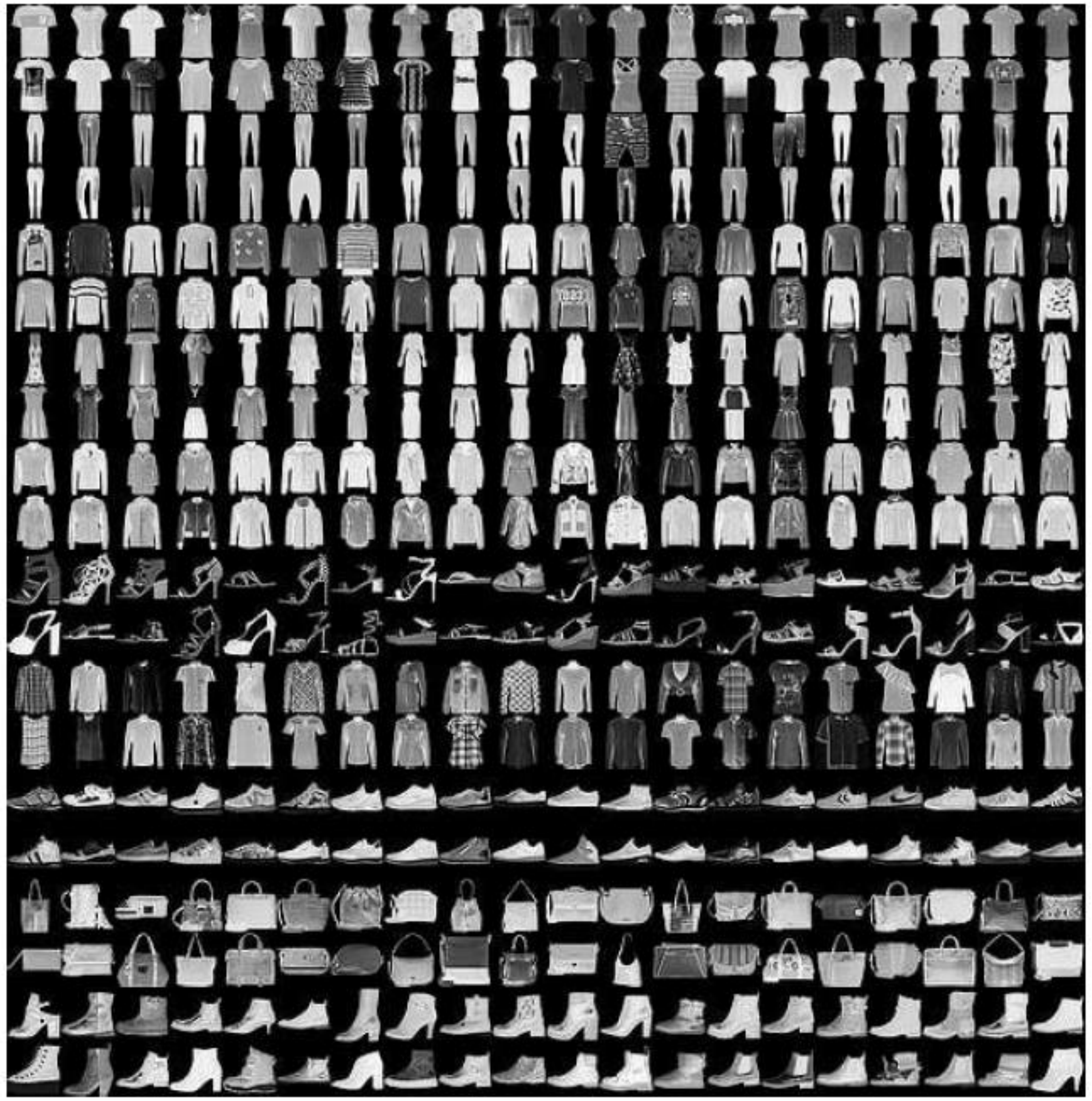
Keras implementation

```python
from tensorflow import keras
from tensorflow.keras.datasets import mnist
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense,Dropout,Flatten,Conv2D,MaxPooling2D
from tensorflow.keras import backend as K

num_classes = 10
img_rows, img_cols = 28,28

# DATA
(x_train, y_train), (x_test, y_test) = mnist.load_data()

x_train = x_train.reshape(x_train.shape[0], img_rows, img_cols, 1)
x_test  = x_test.reshape (x_test.shape[0] , img_rows, img_cols, 1)
y_train = keras.utils.to_categorical(y_train, num_classes)
y_test  = keras.utils.to_categorical(y_test,  num_classes)

# MODEL
input_shape = (img_rows, img_cols, 1)
model = Sequential()
model.add(Conv2D(32, kernel_size=(3, 3), activation='relu', input_shape=input_shape))
model.add(Conv2D(64, (3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))
model.add(Flatten())
model.add(Dense(128, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(num_classes, activation='softmax'))
model.compile(loss      = keras.losses.categorical_crossentropy,
              optimizer= keras.optimizers.Adadelta(),
              metrics   = ['accuracy'])
# TRAIN
model.fit(x_train, y_train,batch_size=128,epochs=12,
          verbose=1, validation_data=(x_test, y_test))

# TEST
score = model.evaluate(x_test, y_test, verbose=0)
print('Test loss:',      score[0])
print('Test accuracy:', score[1])
```

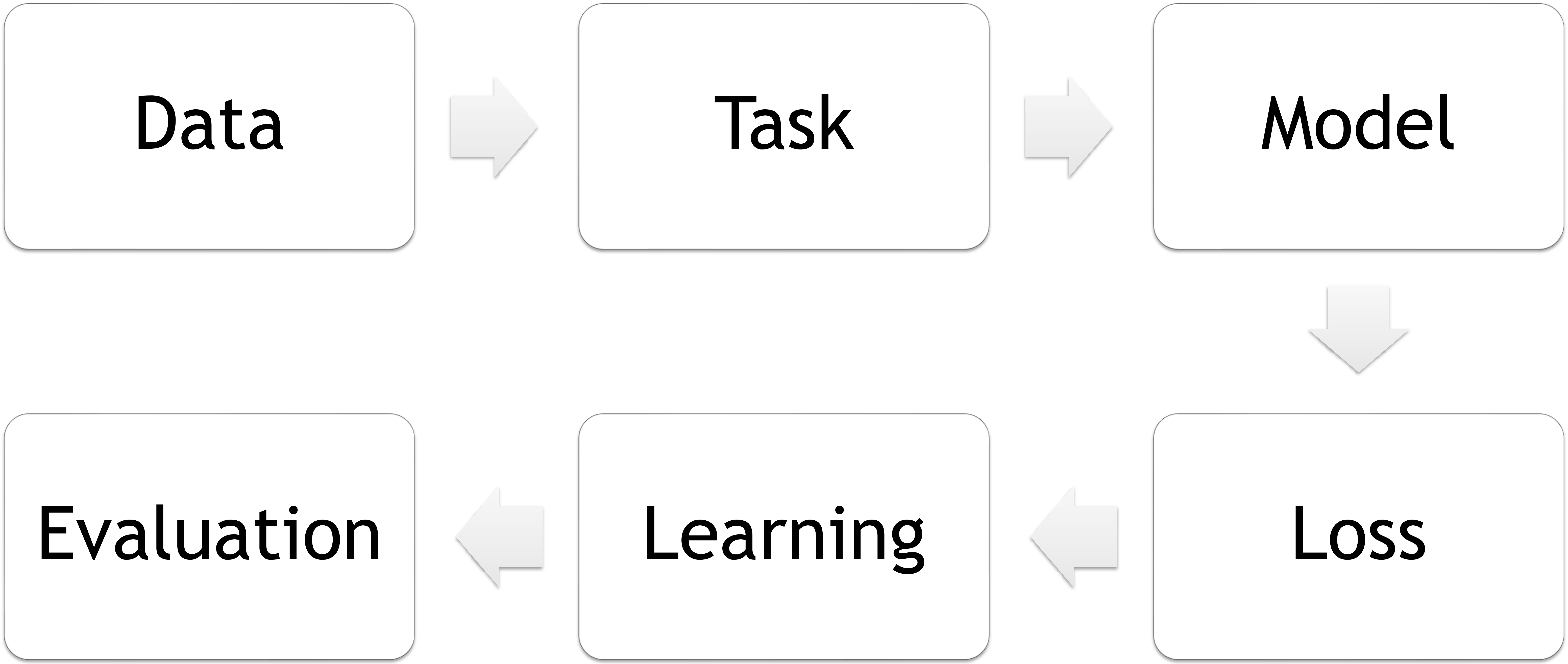
FASHION MNIST

A slightly more interesting version of MNIST

Label	Description	Examples
0	T-Shirt/Top	
1	Trouser	
2	Pullover	
3	Dress	
4	Coat	
5	Sandals	
6	Shirt	
7	Sneaker	
8	Bag	
9	Ankle boots	



6 STEPS APPROACH



LAUNCH CNN PRIMER AND KERAS 101

12:30-1:00 ET

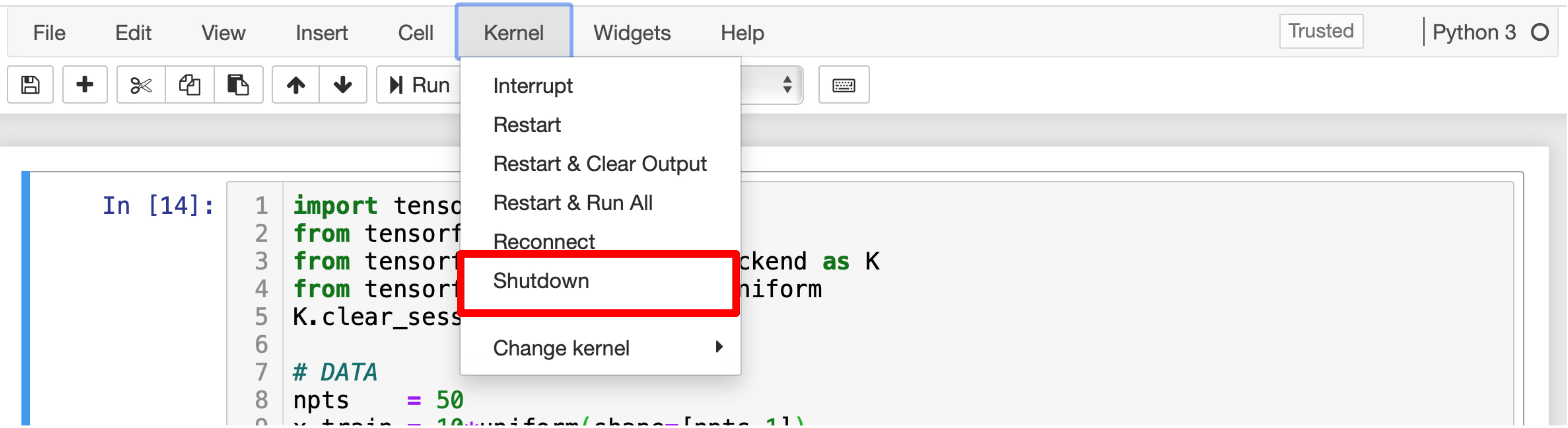
Step through the primer on your own (shift + enter on each cell)

The following contents will be covered during the Bootcamp :

- CNN Primer and Keras 101 (Intro_to_DL/Part_2.ipynb)
- Tropical Cyclone Intensity Estimation using Deep Convolution Neural Networks.

CLICK HERE

Shutdown the kernel before clicking on “Next Notebook” to free up the GPU memory

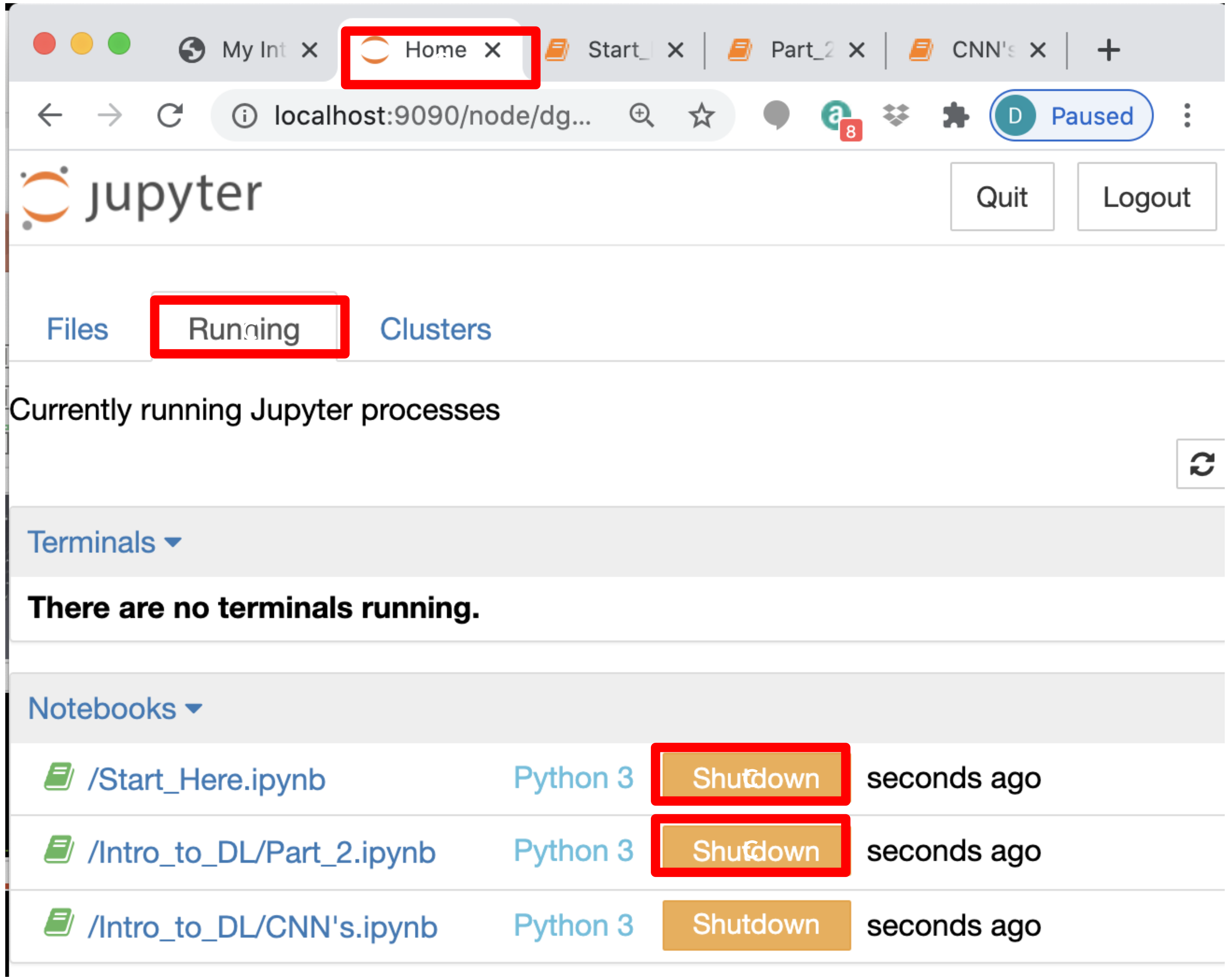


FORGOT TO SHUTDOWN YOUR KERNELS?

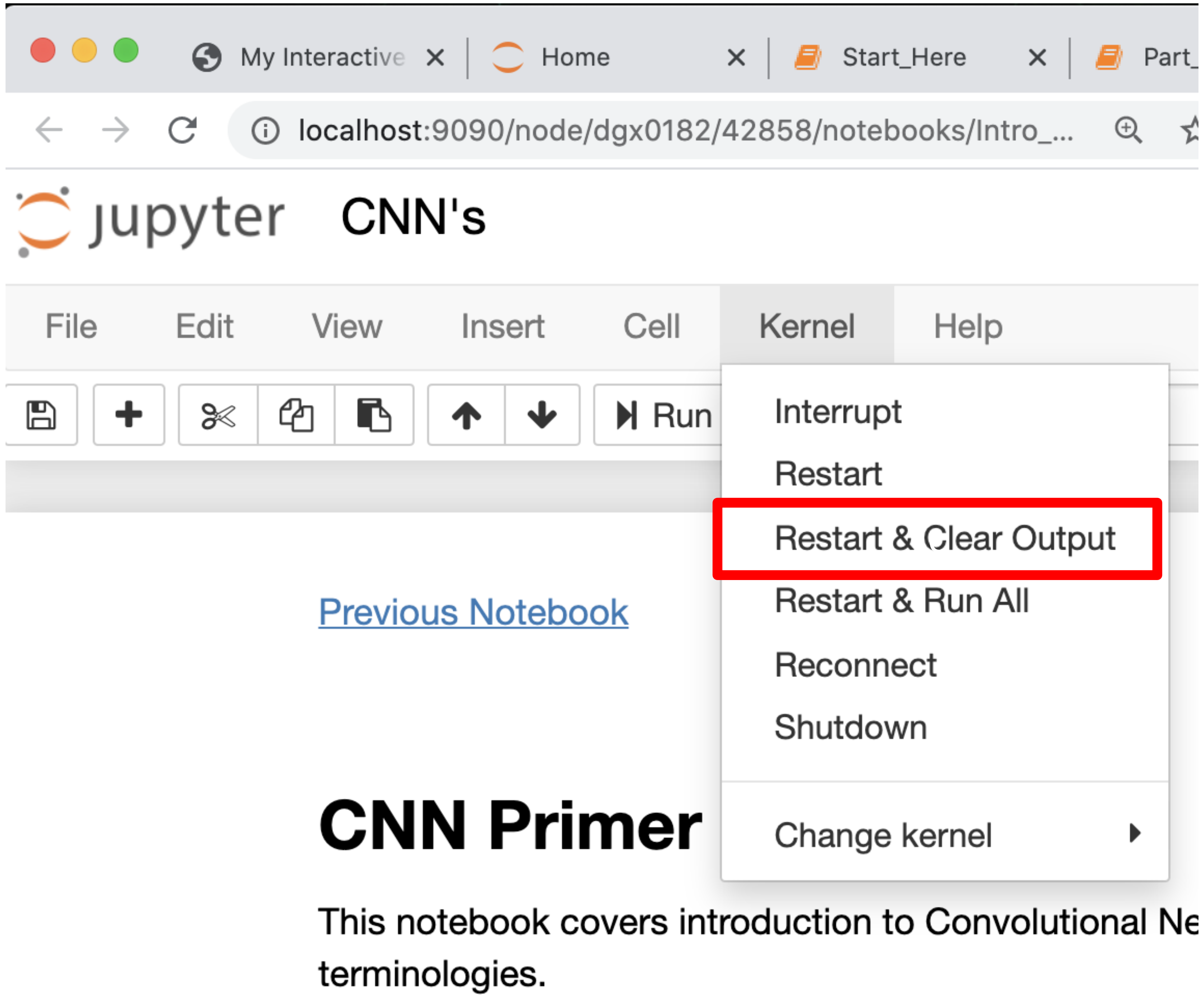
Don't worry, you can fix it.

```
UnknownError:  Failed to get convolution algorithm. This is probably because cuDNN failed to
initialize, so try looking to see if a warning log message was printed above.
```

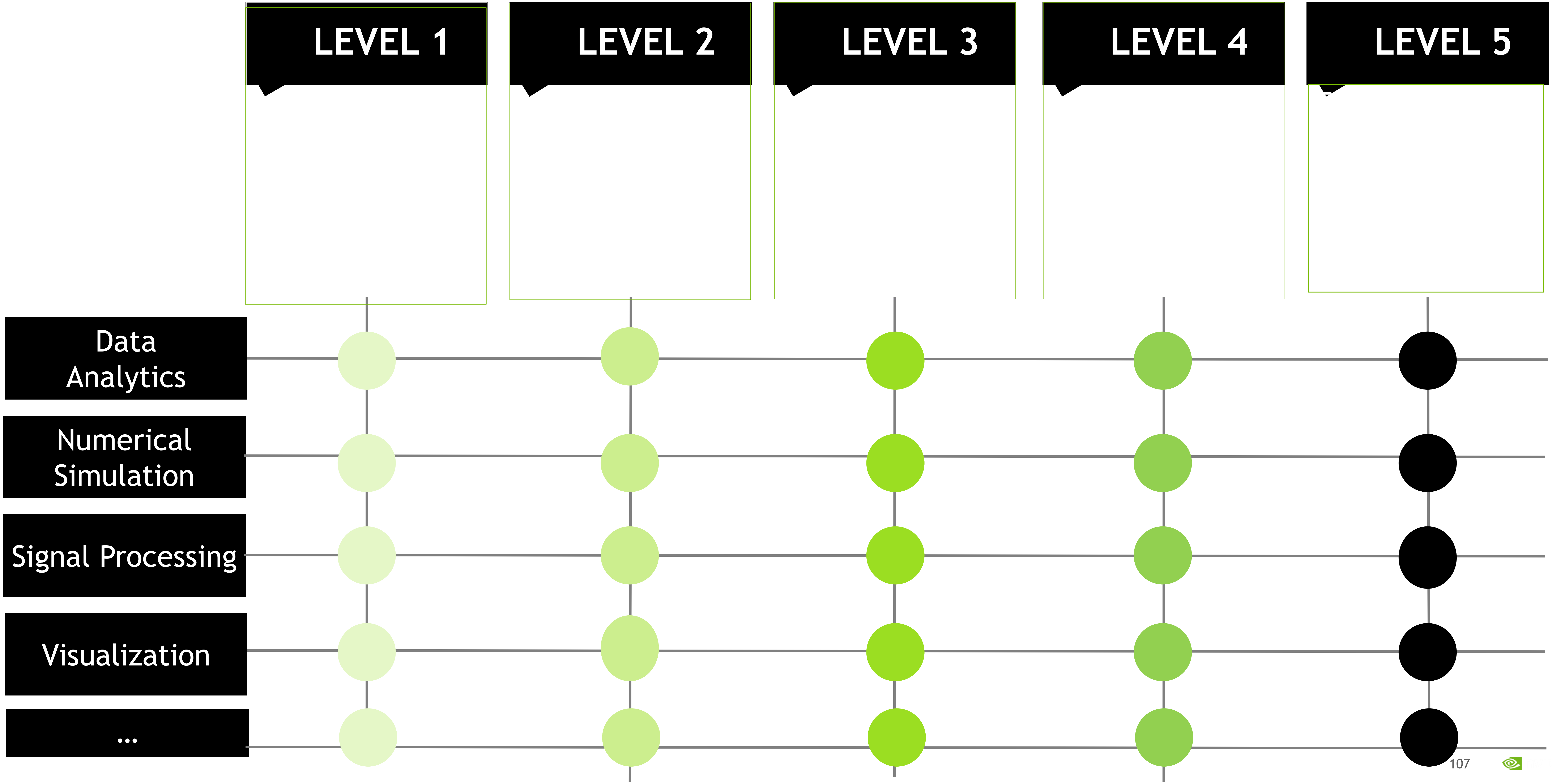
Go to Home Tab, Click Running Tab, Kill notebooks you aren't using



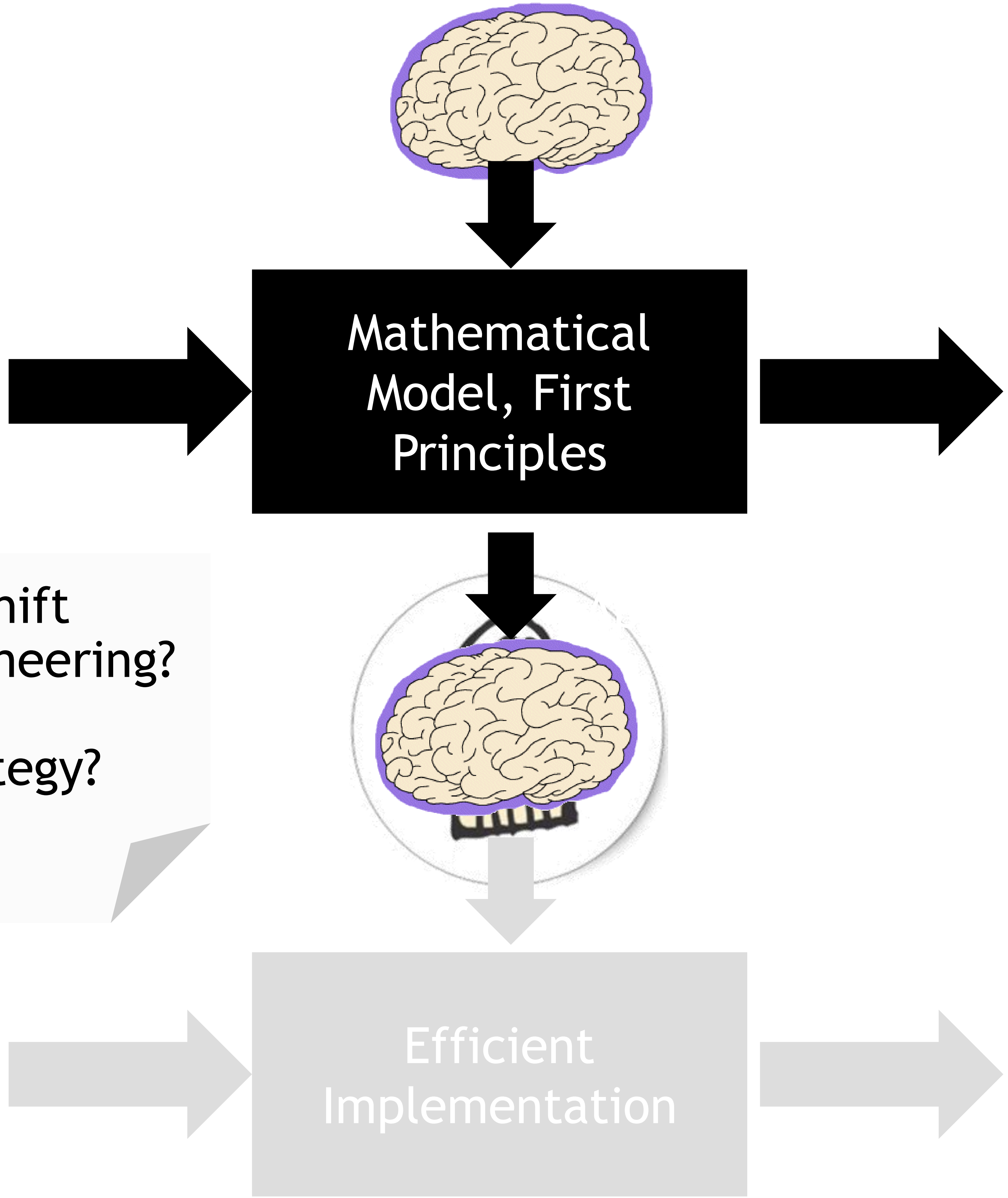
Restart & Clear Output on the Kernel you are using



LEVELS OF AI ENGAGEMENT



COMPUTATIONAL SCIENCES

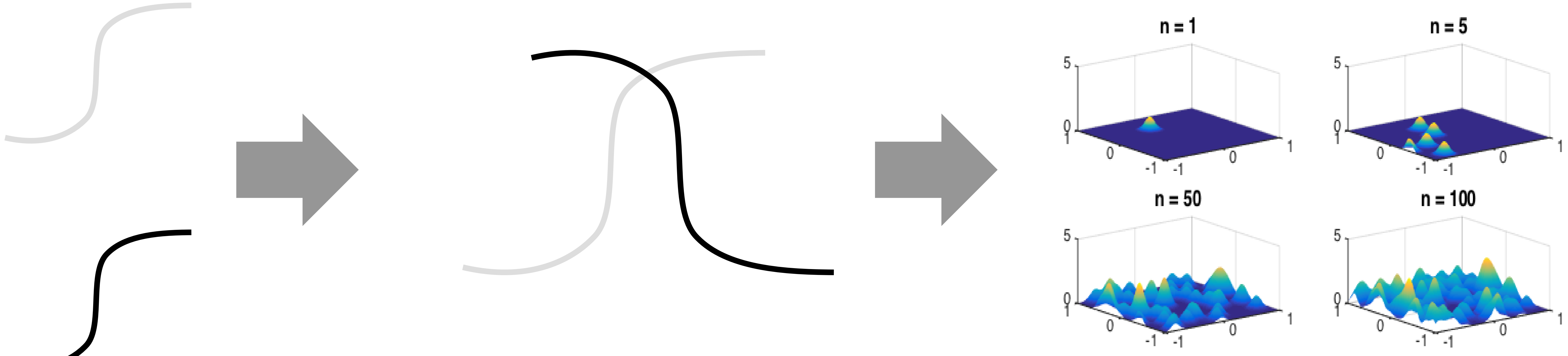


Similarities to the shift
Feature → Network Engineering?

NNs as a Porting Strategy?

<u>CAN</u> THIS WORK ∀? ABOLUTELY, YES!

Proof: Universal Approximation Theorem



Problem: this is an essentially useless theorem for practical purposes

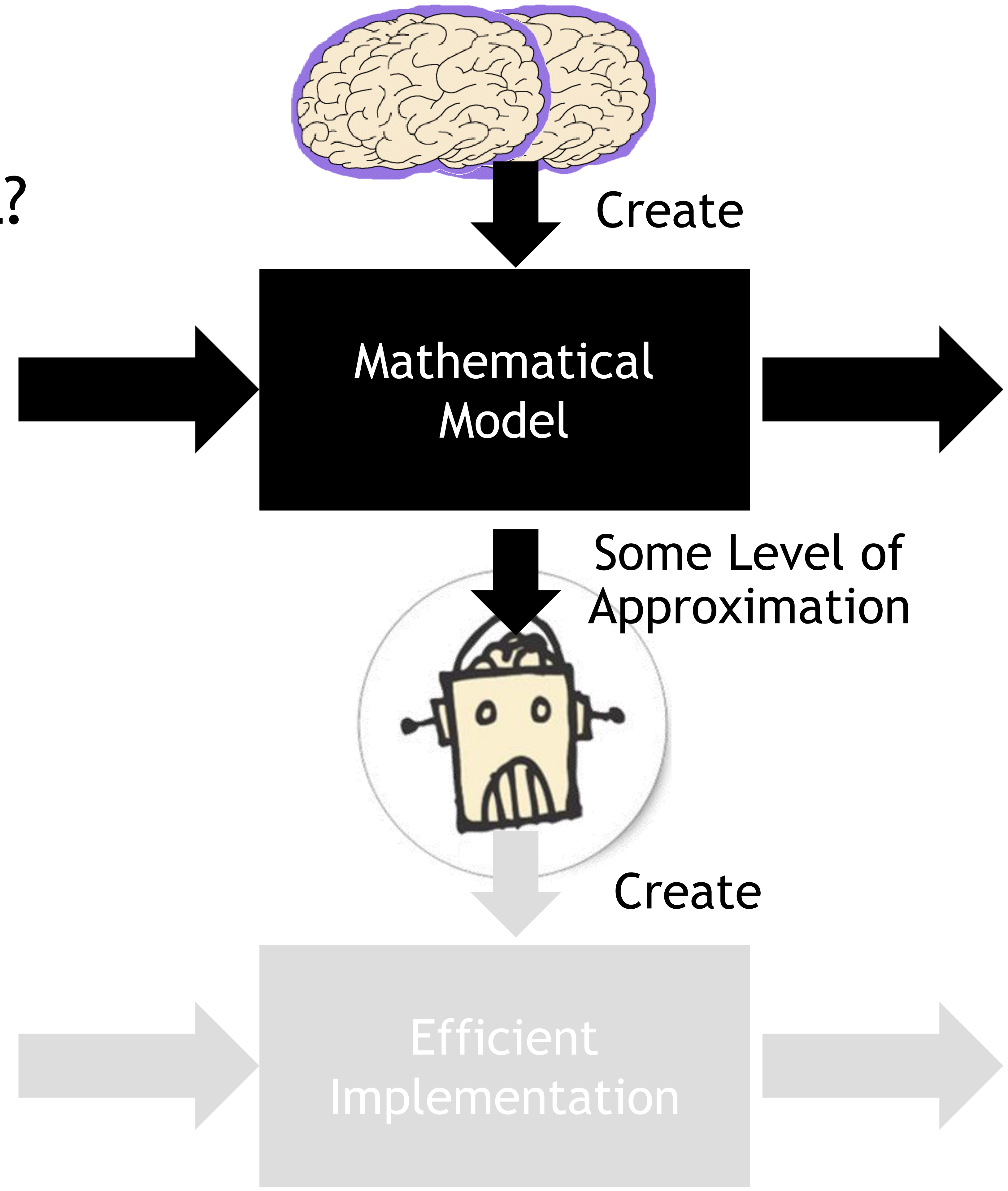
<u>WILL</u> THIS WORK ∀?

Considering pesky practical constraints, like memory and performance

- Anecdotal Evidence: ∃ scientific cases where NNs seem to do work extremely well
- Save bet: it will not work for ∀
- Therefore, by induction (sort of):
  - There exists ∃ a subspace in ∀ HPC applications, for which AI works well
  - Need to explore the **size** and **shape** of this subspace
  - Currently I think it is fair to say we don't understand this domain very well
  - **But:** Each individual case promising 10x, 100x,1000x performance improvement is probably worth exploring; those can be groundbreaking!

But Intuition is Misleading

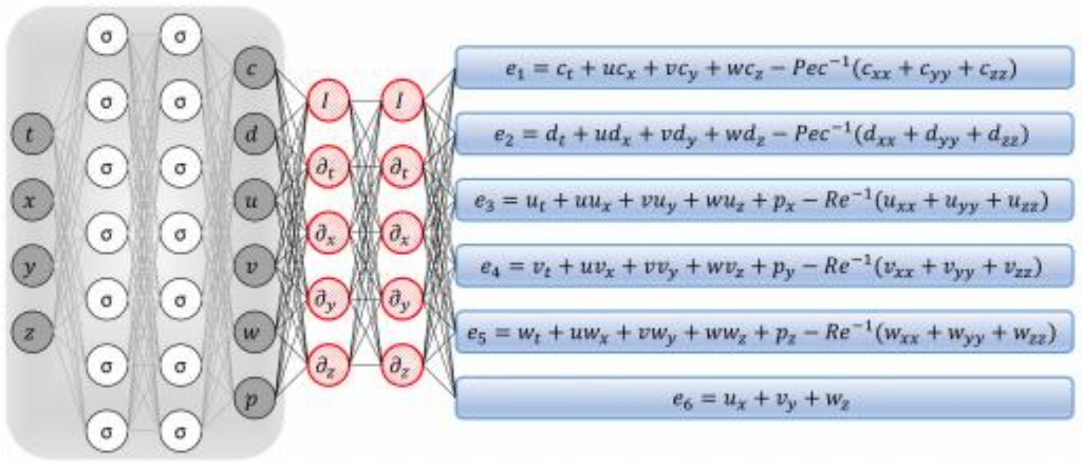
WHAT MAKES AI * HPC SPECIAL?



HOW TO FILL IN THE

Guided Design

New Approaches



SCIENTIFIC CHALLENGES

Barriers to acceptance of deep learning as a tool for science

- **Interpretability:** **Can I understand what the neural-net is doing?**
- **Robustness:** **Will it always give me the right answer?**
- **Coverage:** **How much training data do I need?**
- **Convergence:** **How can I ensure that training will converge?**
- **Uncertainty:** **How certain can I be of the answers?**

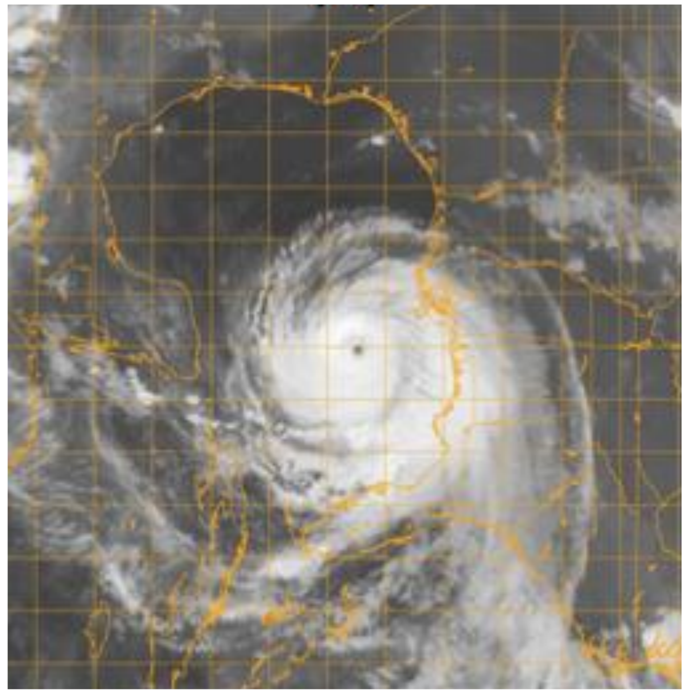
ESTIMATING TROPICAL CYCLONE INTENSITY

Paper Overview

Tropical Cyclone Intensity Estimation Using a Deep Convolutional Neural Network

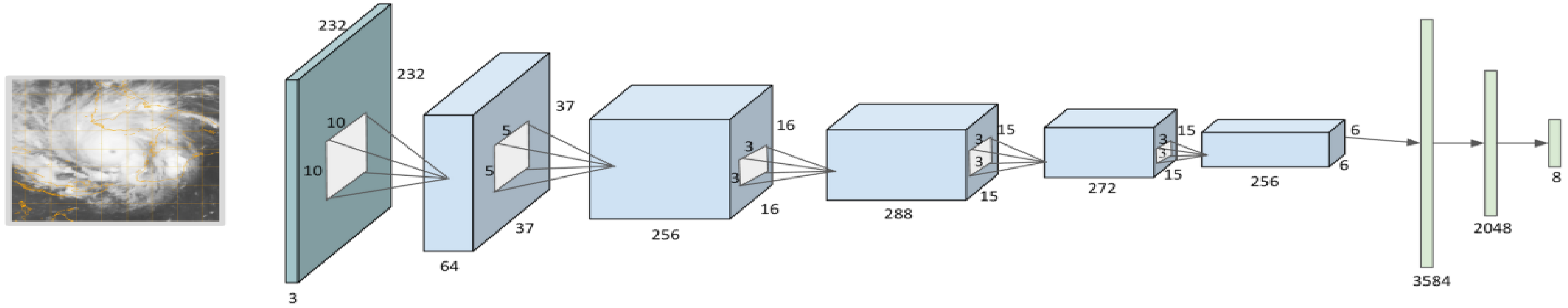
Ritesh Pradhan, Ramazan Aygun, *Senior Member, IEEE,* Manil Maskey, *Member, IEEE,* Rahul Ramachandran, *Senior Member, IEEE,* and Daniel Cecil

INPUT: 232 x 232 pixels



OUTPUT: 8 CLASSES

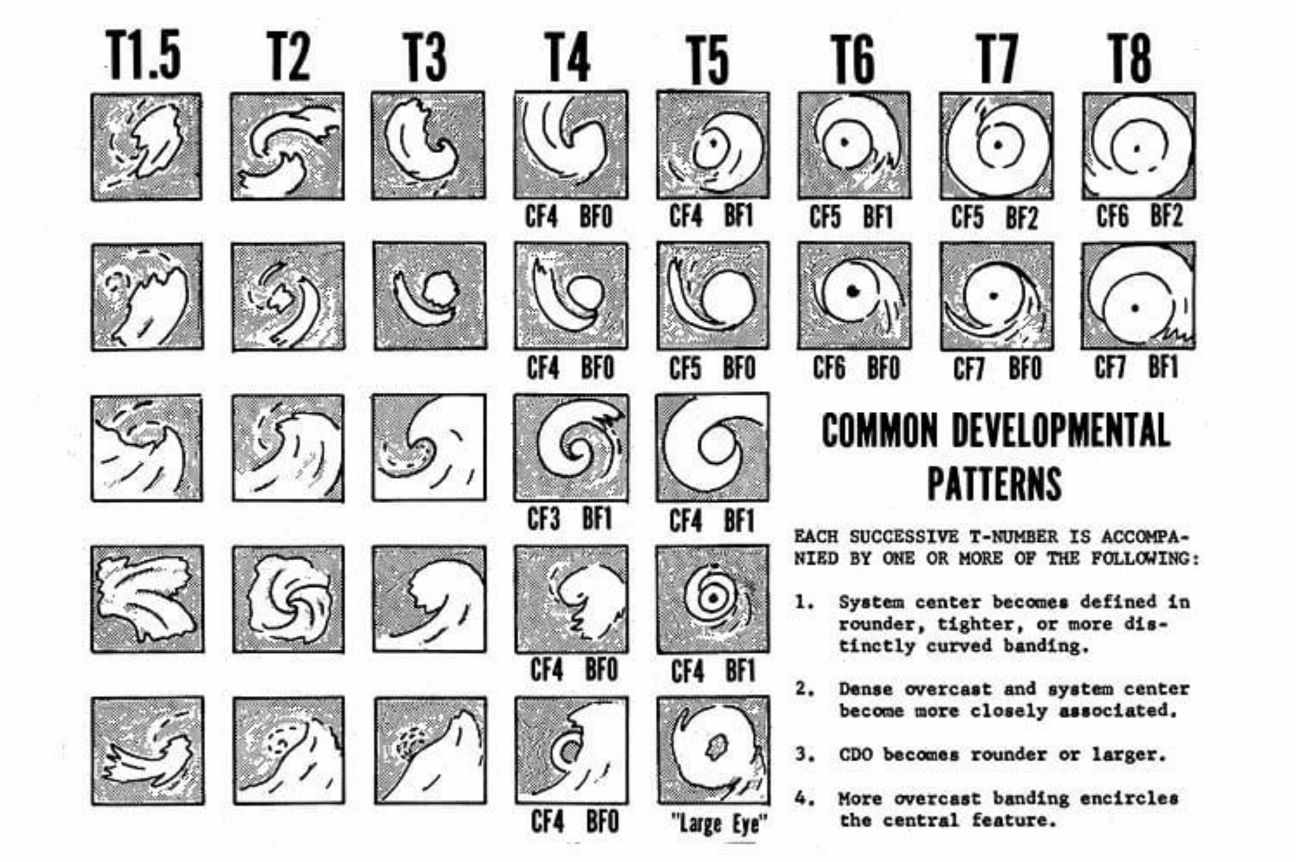
Category	Symbol	Wind speeds	Damage
Five	H5	$\geq$ 137 knots	Catastrophic
Four	H4	113-136 knots	Catastrophic
Three	H3	96-112 knots	Devastating
Two	H2	83-95 knots	Extensive
One	H1	64-82 knots	Significant
Tropical storm	TS	34-63 knots	Significant
Tropical depression	TD	20-33 knots	Small
No Category	NC	$\leq$ 20 knots	–



ESTIMATING TROPICAL CYCLONE INTENSITY

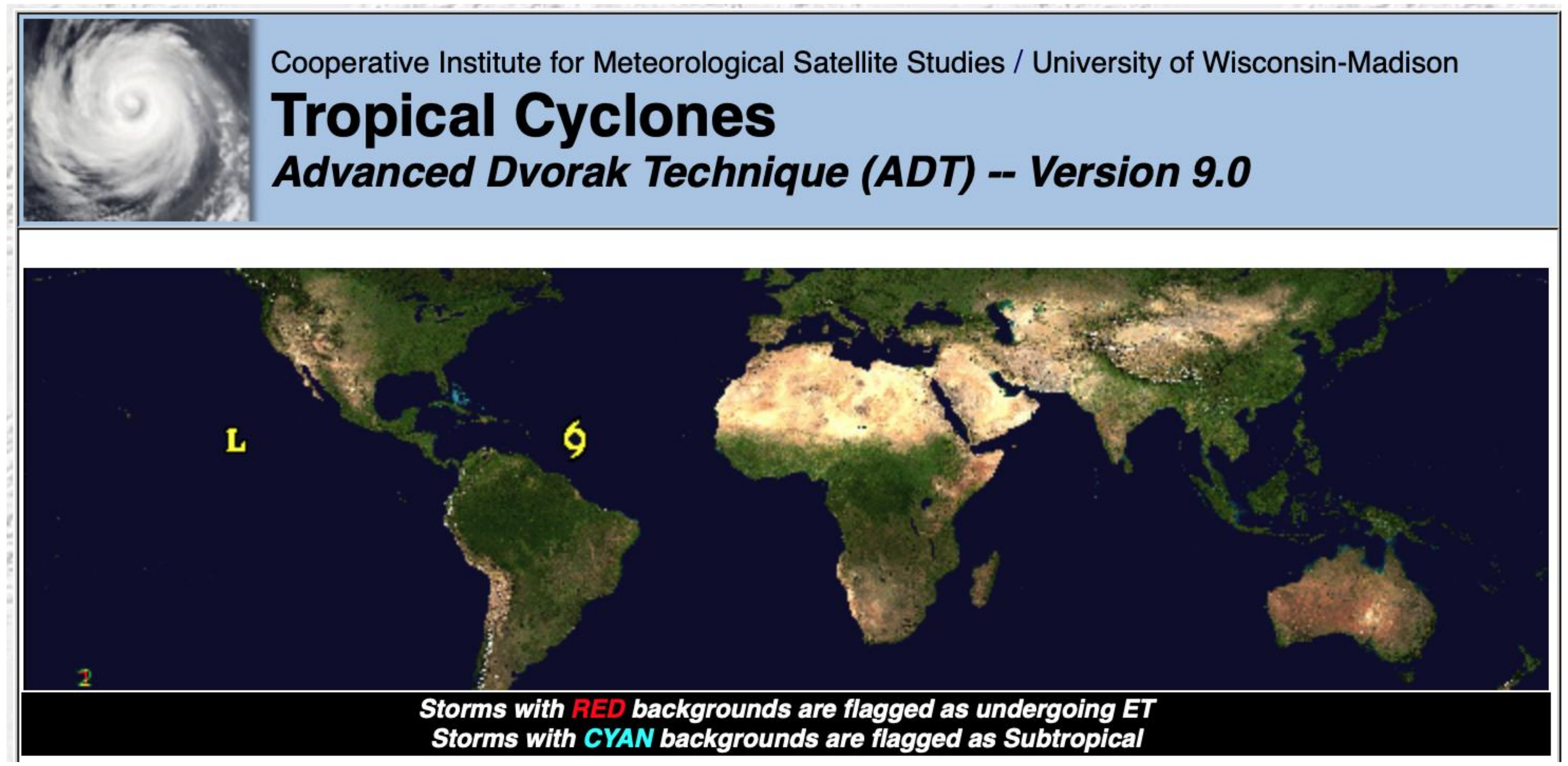
Background: Dvorak technique

Dvorak Technique (1974)



https://doi.org/10.1175/1520-0493(1975)103%3C0420:TCIAAF%3E2.0.CO;2

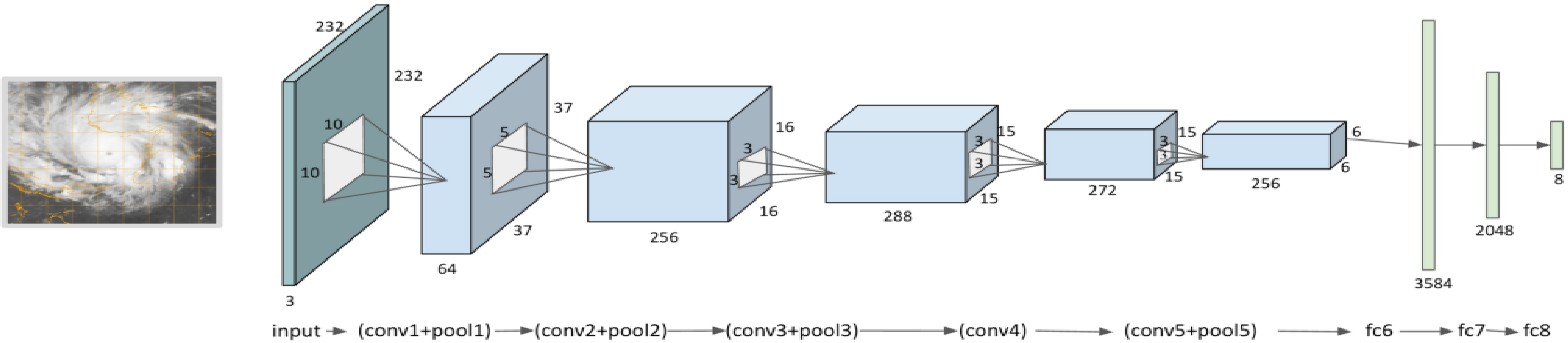
Advanced Dvorak Technique- version 9 (2019)



https://doi.org/10.1175/WAF-D-19-0007.1

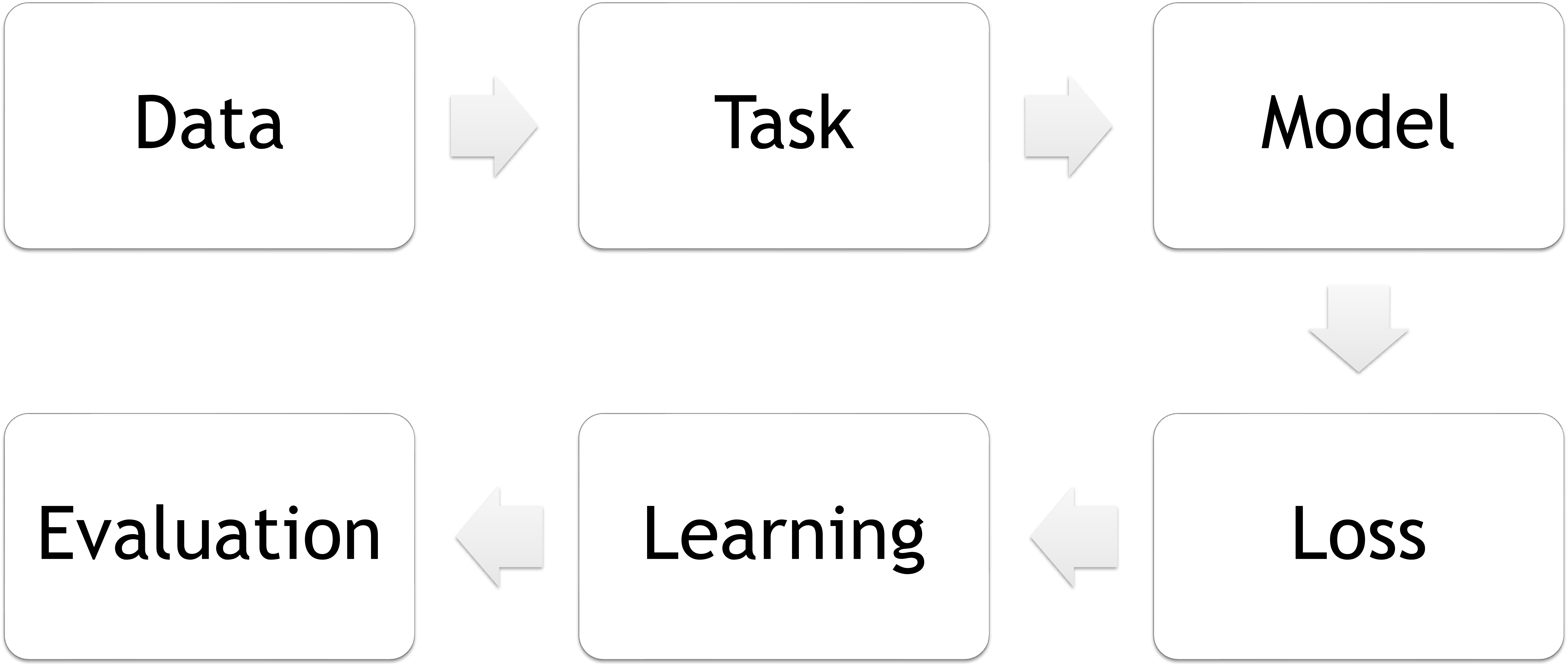
ESTIMATING TROPICAL CYCLONE INTENSITY

CNN Model

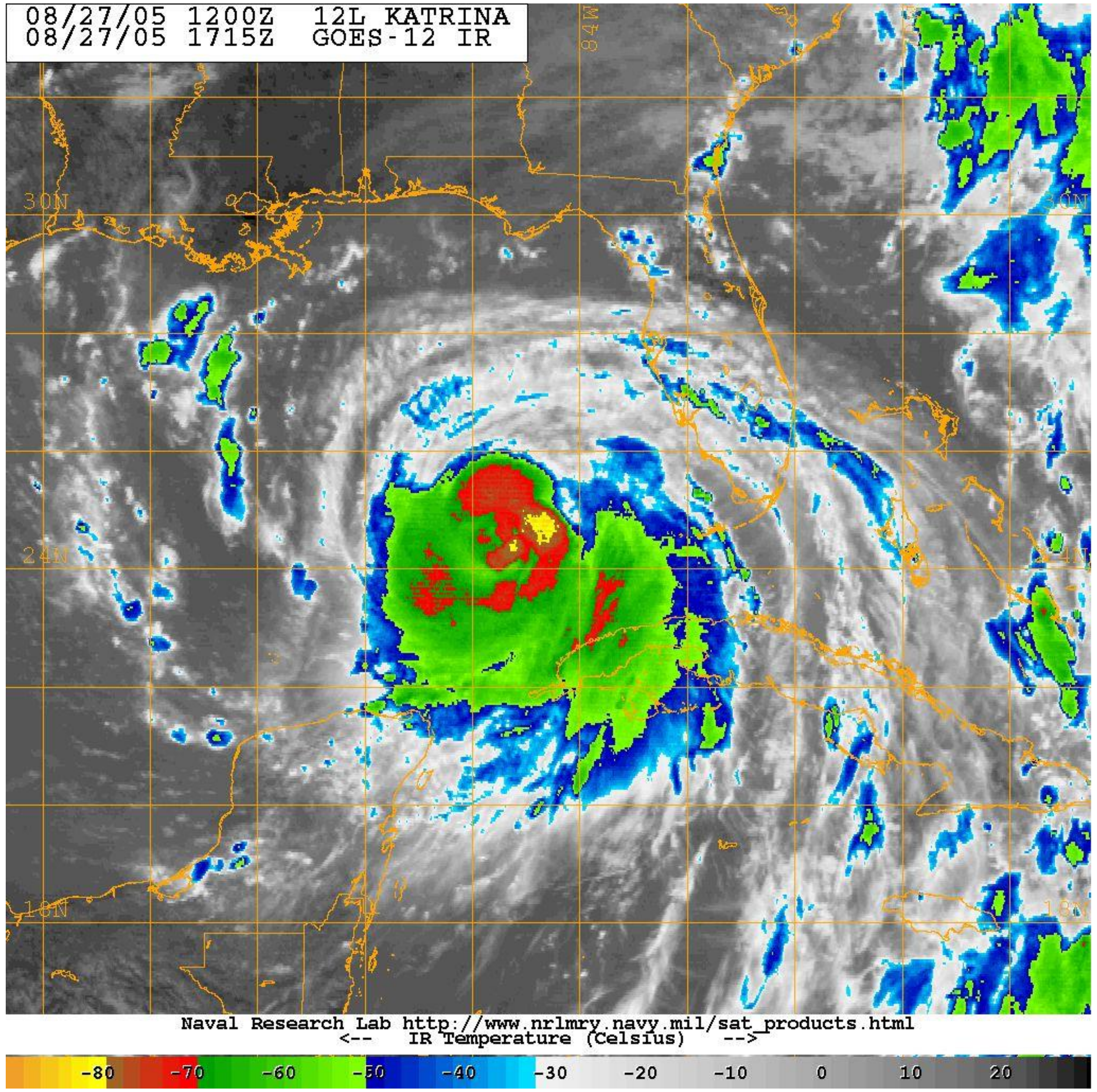


6 STEPS APPROACH

Steps to follow while solving a Machine Learning problem



DATA



SAFFIR-SIMPSON HURRICANE WIND SCALE AND RELATED CLASSIFICATIONS

Category	Symbol	Wind speeds	Damage
Five	H5	$\geq$ 137 knots	Catastrophic
Four	H4	113–136 knots	Catastrophic
Three	H3	96–112 knots	Devastating
Two	H2	83–95 knots	Extensive
One	H1	64–82 knots	Significant
Tropical storm	TS	34–63 knots	Significant
Tropical depression	TD	20–33 knots	Small
No Category	NC	$\leq$ 20 knots	-

TASK

Multi-class Classification.

NC (No Category , ≤ 20 knots)

TD (Tropical Depression , $20-33$ knots)

TS (Topical Storm , $34-63$ knots)

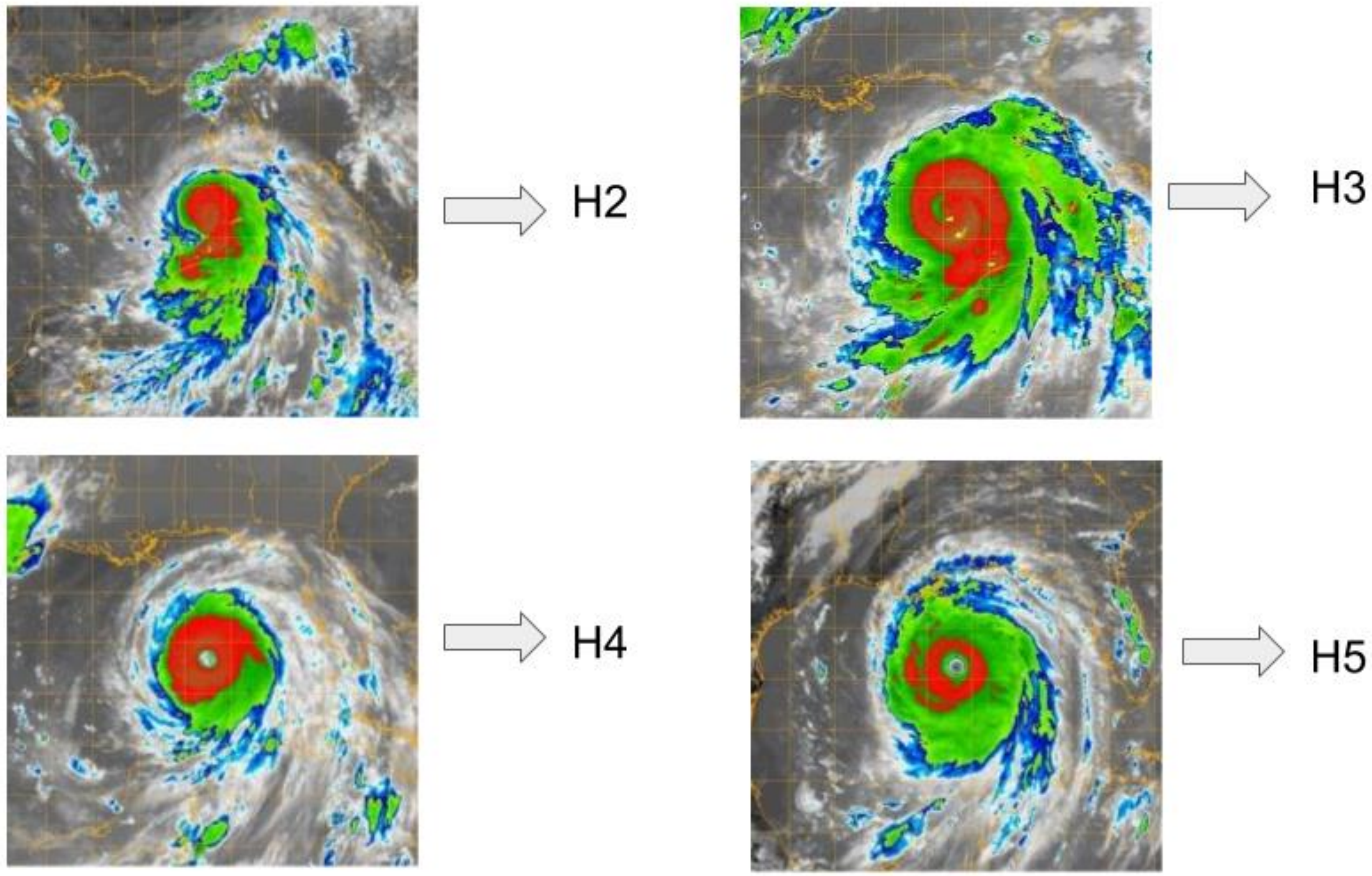
H1 (Category One , $64-82$ knots)

H2 (Category Two , $83-95$ knots)

H3 (Category Three , $96-112$ knots)

H4 (Category Four , $113-136$ knots)

H5 (Category Five , ≥ 137 knots)



MODEL 1

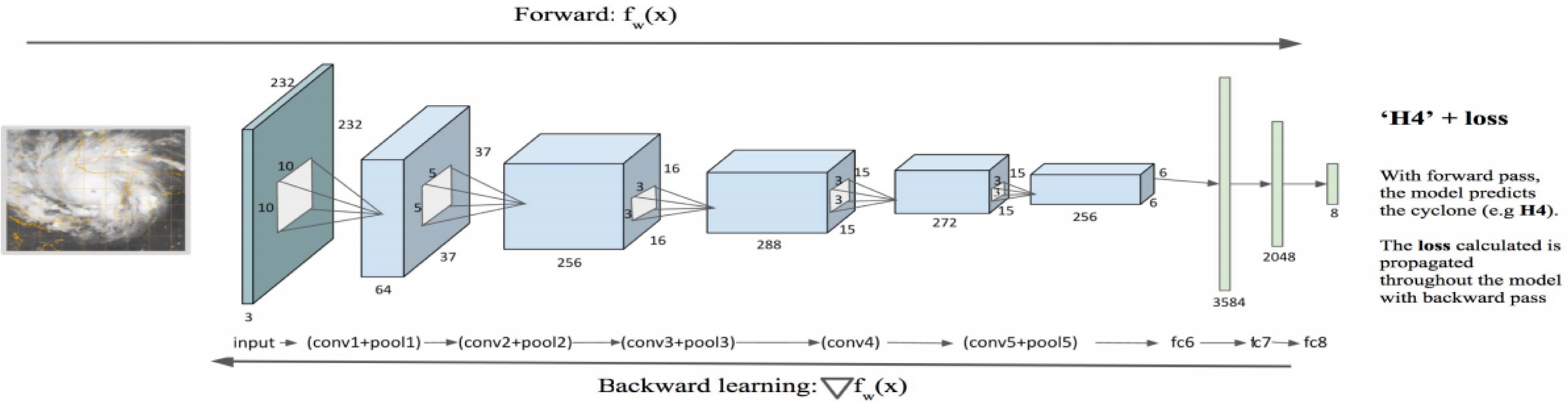


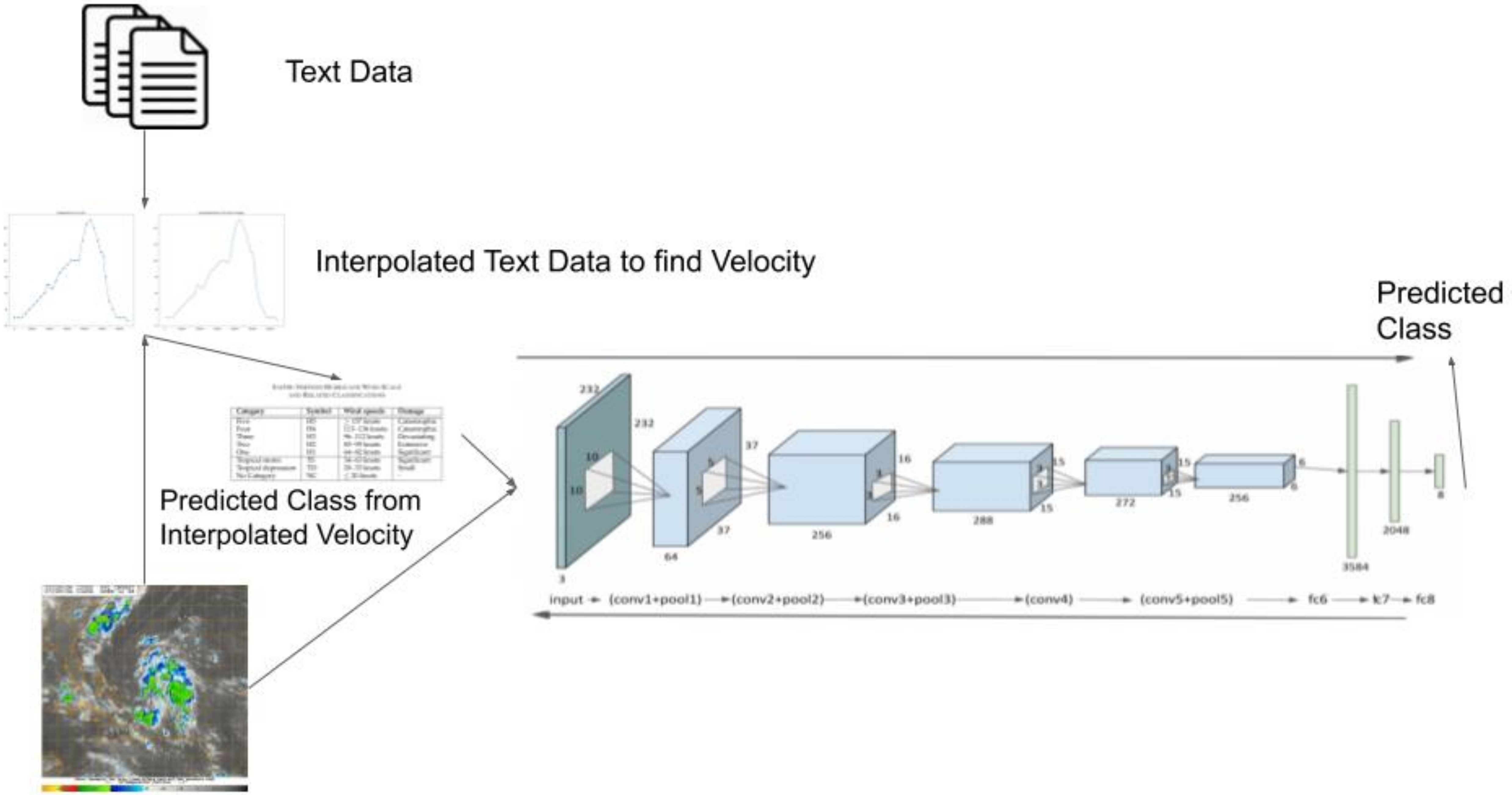
Fig. 2. Network architecture for hurricane intensity estimation showing different steps of convolution and pooling.

Loss Function: Multi-class Cross-Entropy loss functions

Optimizer SGD (Stochastic Gradient Descent)

Training and Evaluation: Training Set 72 % , Test Set, 8 %, Validation Set 10%

SUMMARY OF APPROACH



PREPROCESSING DATA

Pre-Processing Data:

Step 1 : Resize Image from (1024, 1024 ,3) to (256 , 256 ,3)

Step 2 : Choose a random (232 , 232 , 3) patch from the (256 , 256 , 3) and feed into our model.

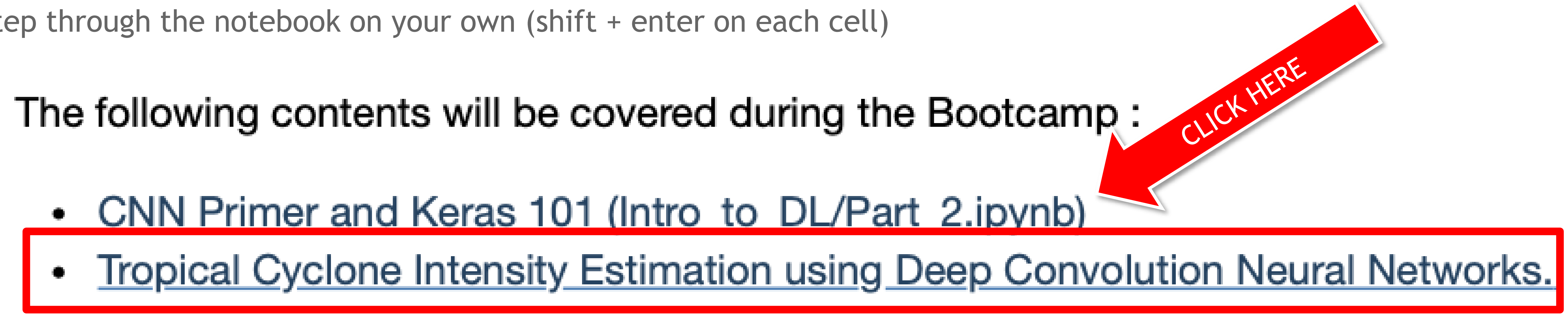
There are different types of Resizing:

- cv2.INTER_AREA (Preferable for Shrinking)
- cv2.INTER_CUBIC (Preferable for Zooming but slow)
- cv2.INTER_LINEAR (Preferable for Zooming and the default option)

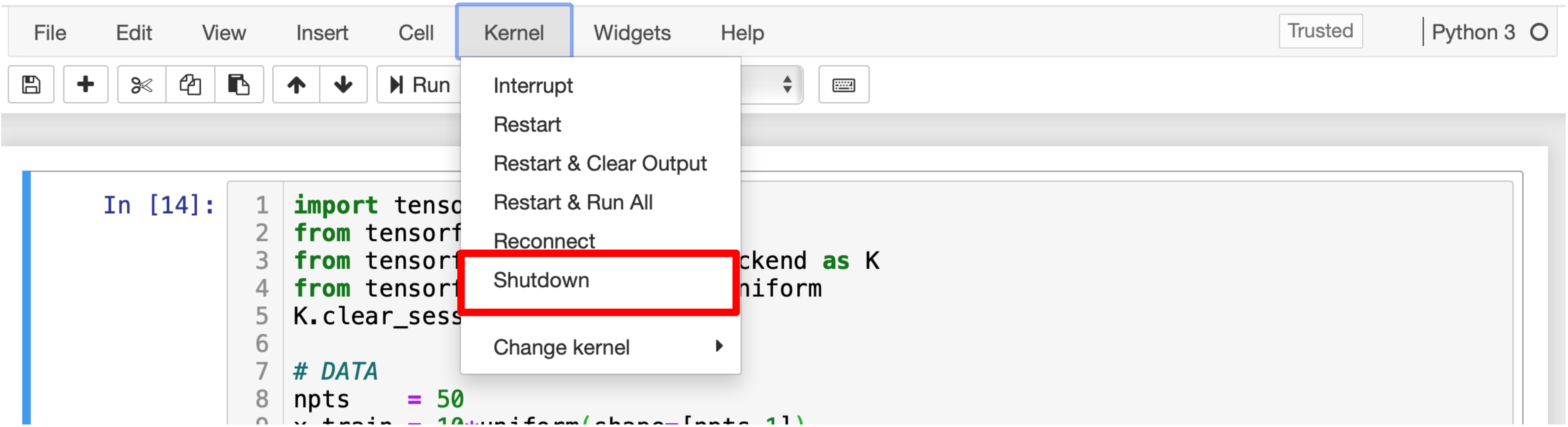
LAUNCH TROPICAL CYCLONE NOTEBOOK

1:00-2:00 ET

Step through the notebook on your own (shift + enter on each cell)



Shutdown the kernel before clicking on “Next Notebook” to free up the GPU memory



TROPICAL CYCLONE COMPETITION

Can you make a better prediction?

Go to last notebook in the TC section

The following contents will be covered during the Bootcamp :

- CNN Primer and Keras 101 (Intro_to_DL/Part_2.ipynb)
- Tropical Cyclone Intensity Estimation using Deep Convolution Neural Networks.

See if you can improve the accuracy.

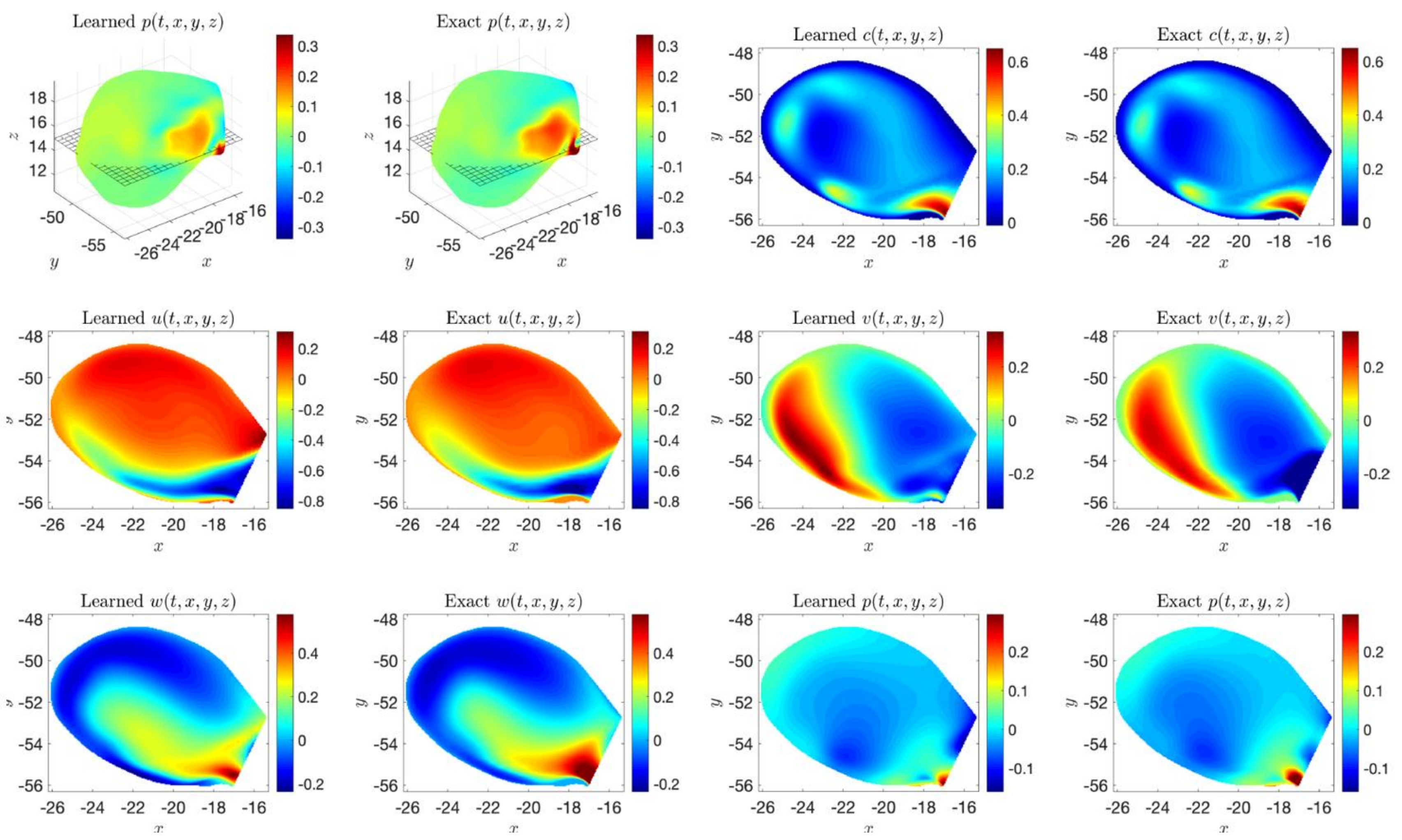
Suggestions:
- Improve the data balance
- Tweak the model hyperparamters
- Try different optimizers

Bug in the lab: when doing the test/train split on time-series data, the data should not be shuffled!

- Try maximizing validation accuracy on both shuffled and un-shuffled validation sets

STEADY STATE FLOW WITH NEURAL NETWORKS

Flow fields are simulated using computational fluid dynamics (CFD) solvers

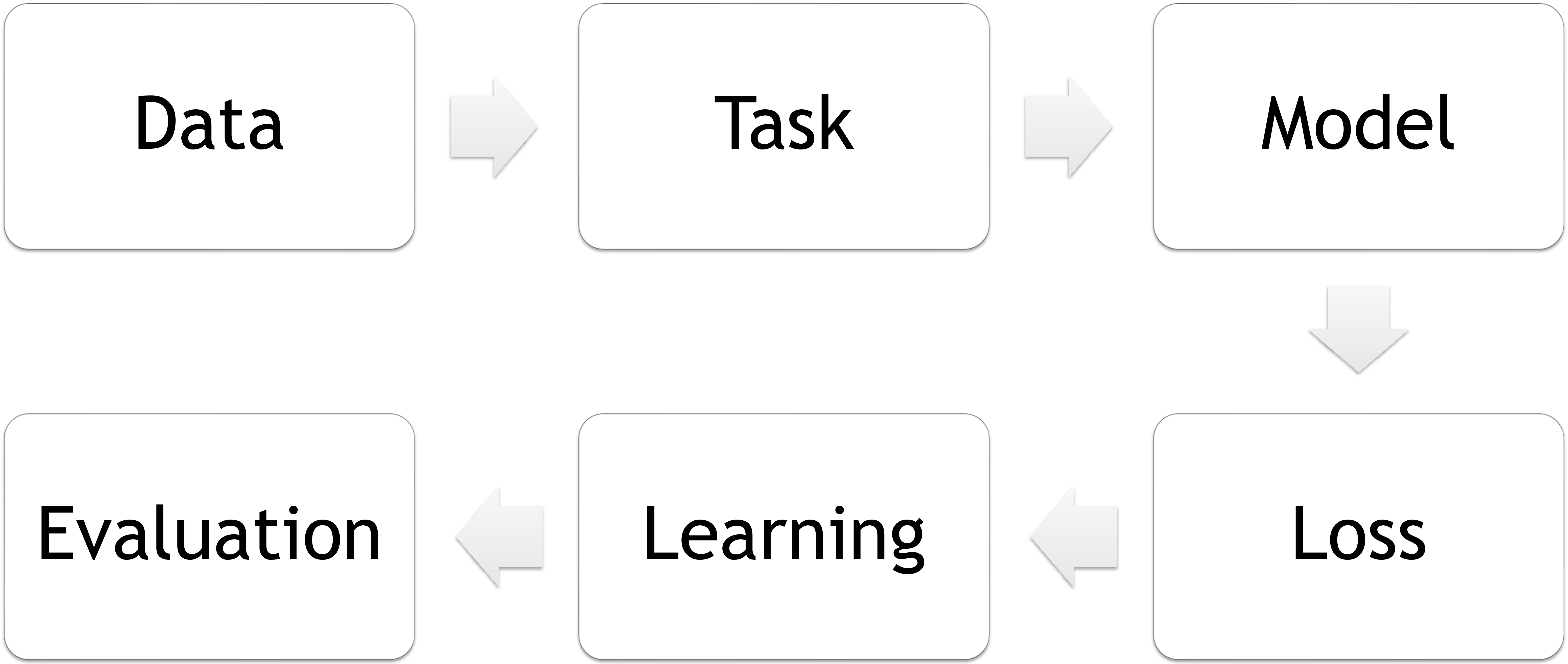


STEADY STATE FLOW WITH NEURAL NETWORKS

Our aim is to predict 2D flow around objects. The input is the boundary around which we want to calculate the flow. Here is an example of input data and the corresponding flow that was calculated using the Lattice Boltzmann method. (Mechsys).

6 STEPS APPROACH

Steps to follow while solving a Machine Learning problem



predict the velocity vectors of both the xx and yy channels from our model.

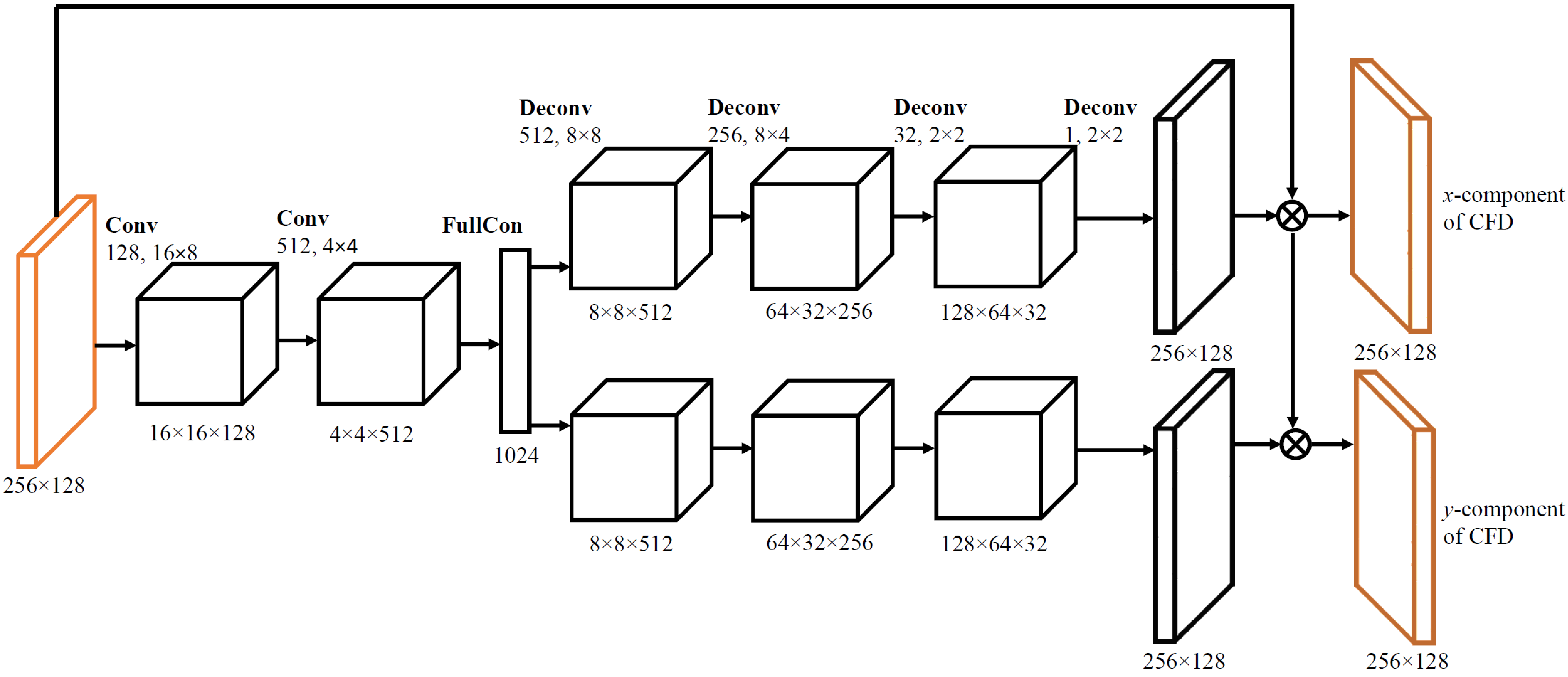
DATA AND TASK

MODEL

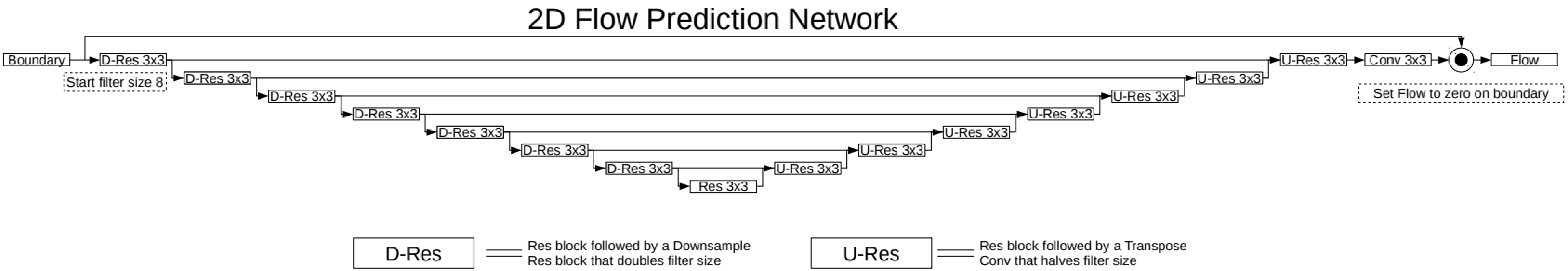
We will be building the following Models and benchmarking them as we proceed :

- Simple Fully Connected Networks
    - *3 Layer Network*
    - *5 Layer Network*
- Convolution Neural Networks
    - *Binary Boundary*
    - *Signed Distance Function*
- Advanced Networks
    - *Gated Residual Network*
    - *Non-Gated Residual Network*

SUMMARY OF APPROACH



U GATED NETWORK



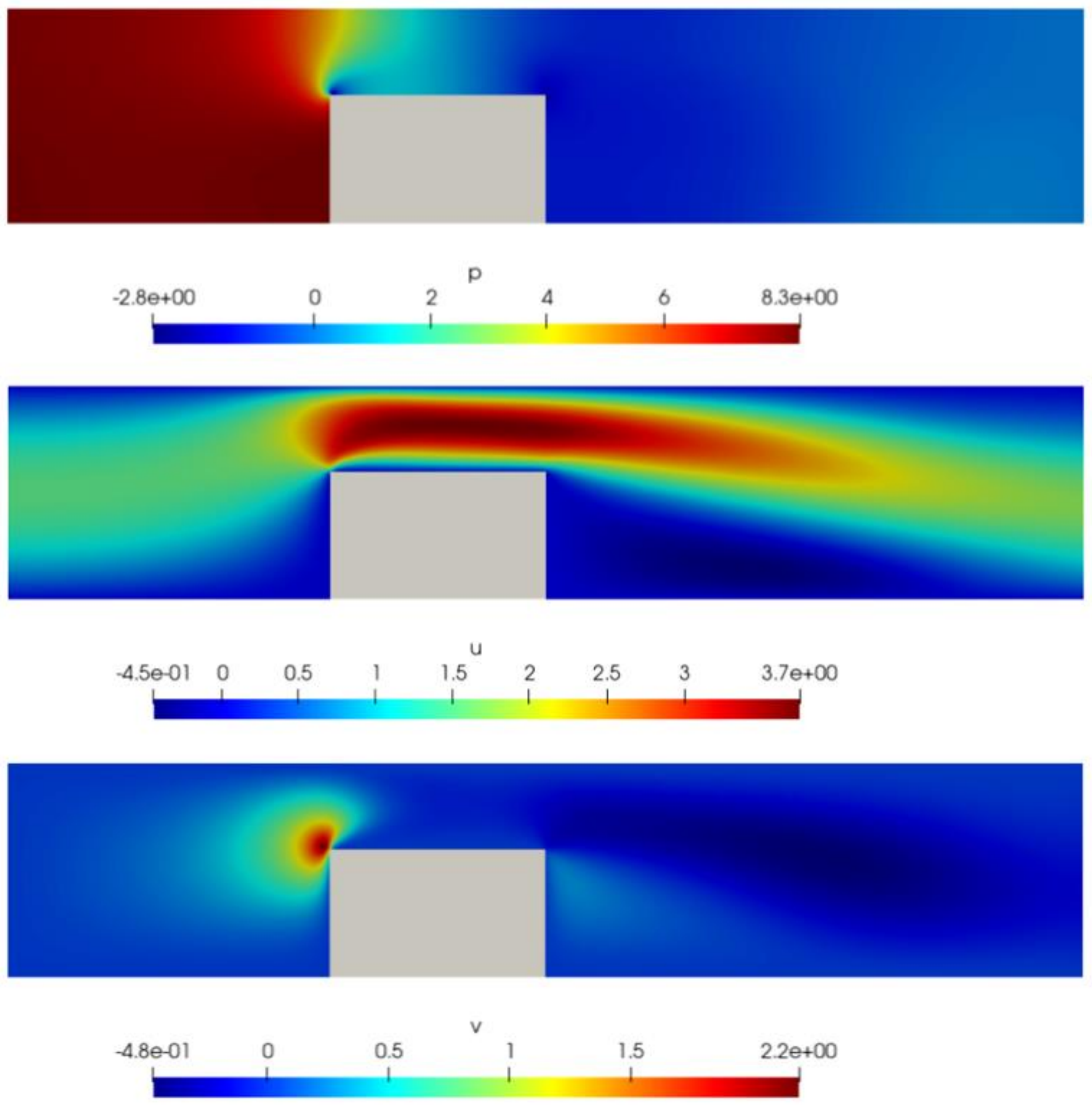
NVIDIA

AI FOR SCIENCE BOOTCAMP USING SIMNET

This material is released by NVIDIA Corporation under the Creative Commons Attribution 4.0 International (CC BY 4.0)

OUTLINE

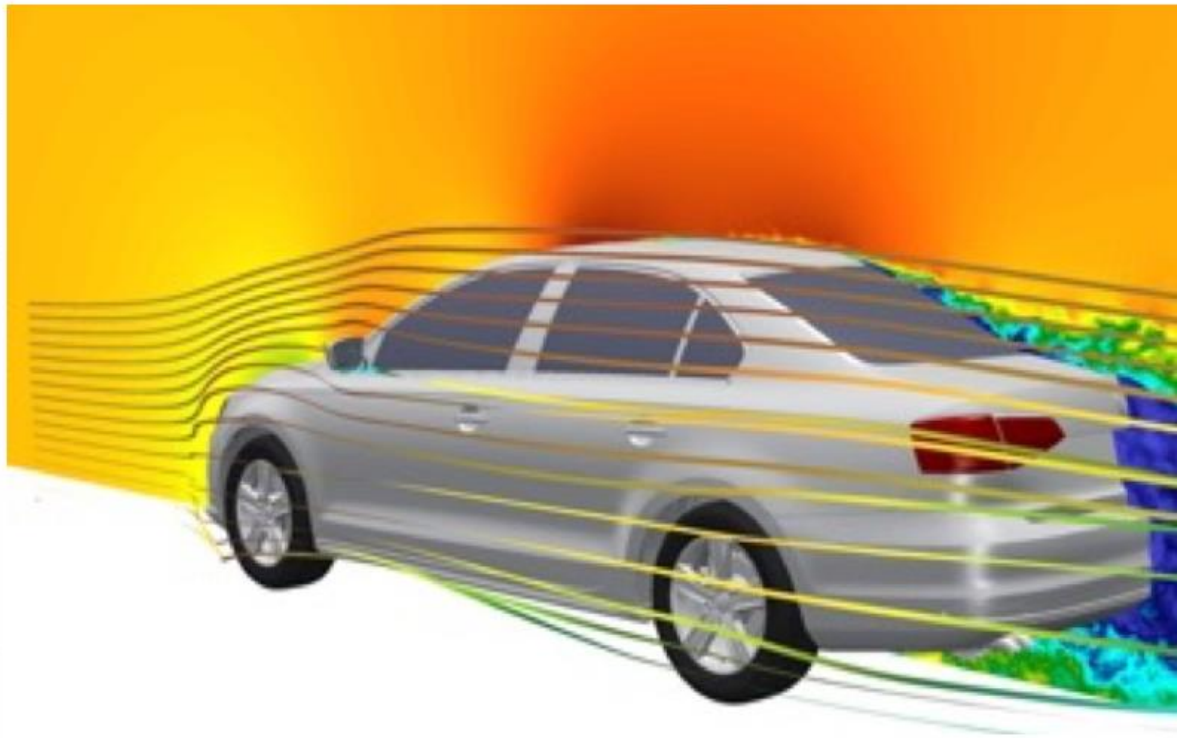
- Introduction to Physics Informed Neural Networks (PINN)
- Solving Partial Differential Equation system using SimNet toolkit
- Solving parameterized PDEs
- Solving transient problems
- Solving inverse problems
- Challenge CFD problem – Flow over a 2D chip



DATA DRIVEN METHODS

Pros
Not dependent on Physics

Cons
No physics awareness; Generalization ability may be limited
Need to generate a lot of simulations (accuracy dependent on the simulation code)
Not very efficient for complex 3D geometries/curved surfaces
Interpolation/extrapolation errors



NEURAL NETWORK SOLVER THEORY

Goal: Train a neural network to satisfy the boundary conditions and differential equations by constructing an appropriate loss function

- Consider an example problem:

- We construct a neural network $u_{net}(x)$ which has a single value input $x \in \mathbb{R}$ and single value output $u_{net}(x) \in \mathbb{R}$.

- We assume the neural network is infinitely differentiable $u_{net} \in C^{\infty}$ - Use activation functions that are infinitely differentiable

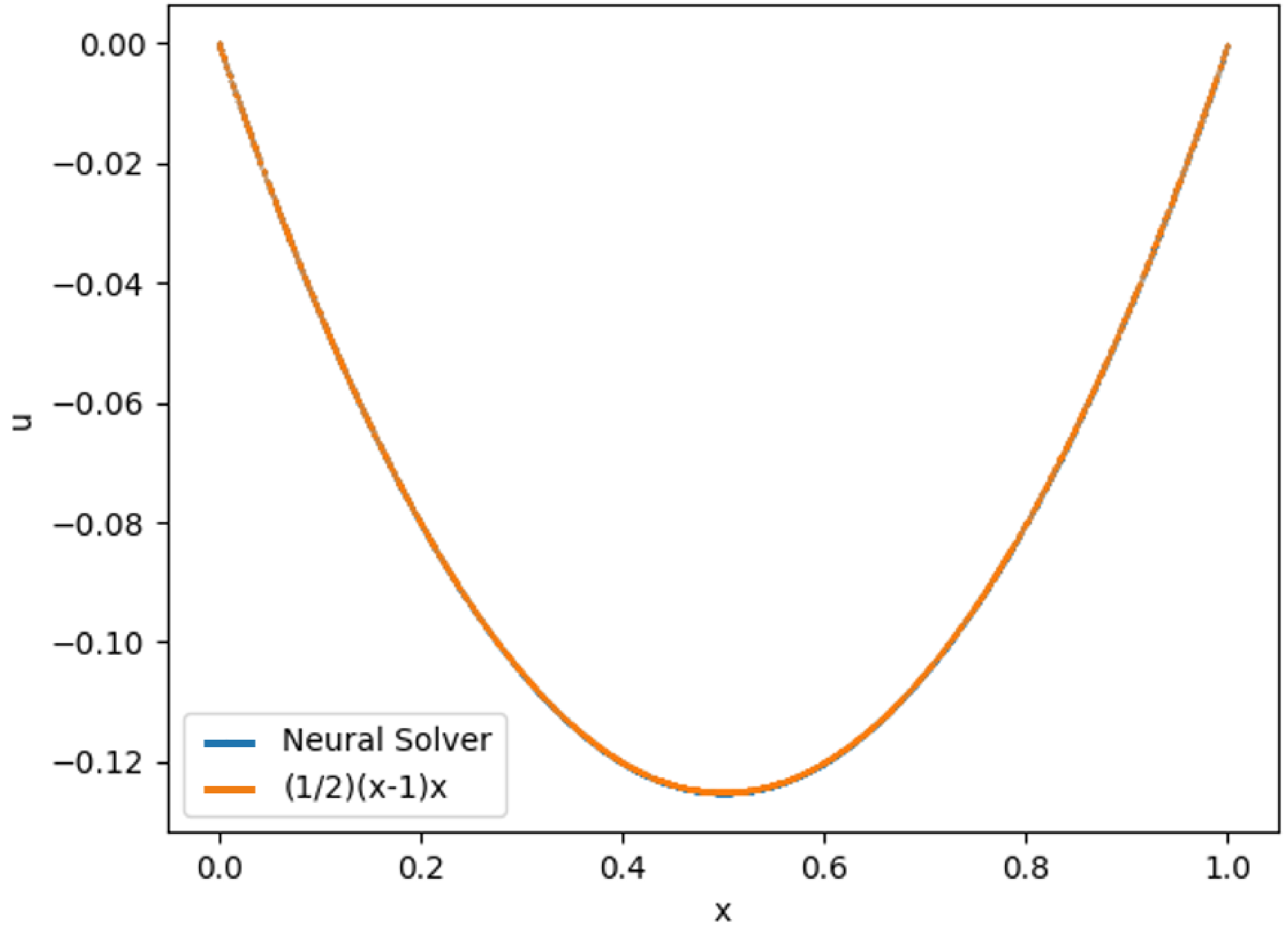
NEURAL NETWORK SOLVER THEORY

- Construct the loss function. We can compute the second order derivatives $\left(\frac{\delta^2 u_{net}}{\delta x^2}(x)\right)$ using Automatic differentiation

- Where x_i are a batch of points in the interior $x_i \in (0, 1)$. Total loss becomes $L = L_{BC} + L_{Residual}$
- Minimize the loss using optimizers like Adam

NEURAL NETWORK SOLVER THEORY

- For $f(x) = 1$, the true solution is $\frac{1}{2}(x-1)x$. After sufficient training we have,



SOLVING PARAMETERIZED PROBLEMS

- Consider the parameterized version of the same problem as before. Suppose we want to determine how the solution changes as we move the position on the boundary condition $u(l) = 0$

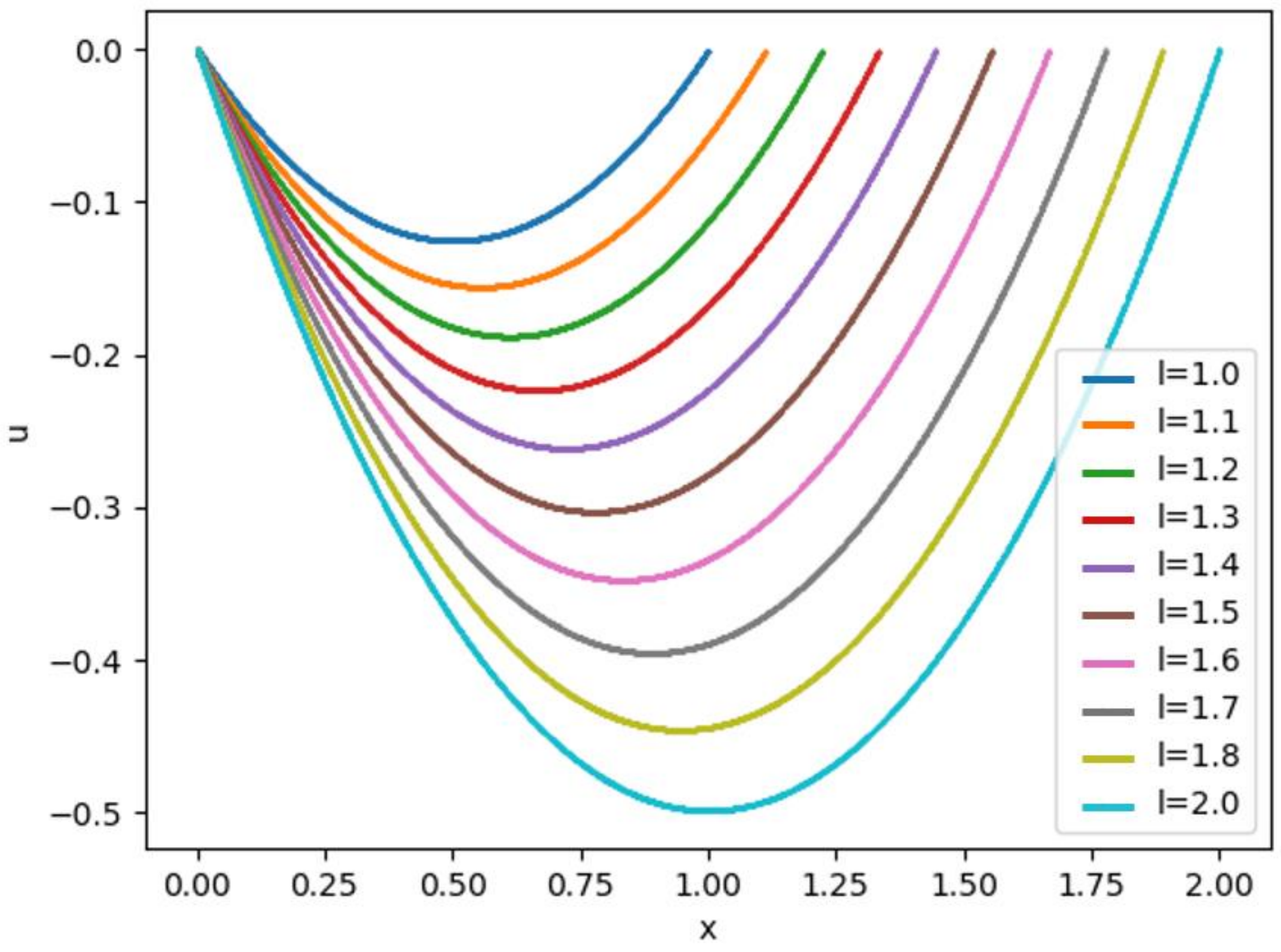
- Parameterize the position by variable $l \in [1, 2]$ and the problem now becomes:

- This time, we construct a neural network $u_{net}(x, l)$ which has x and l as input and single value output $u_{net}(x, l) \in \mathbb{R}$.

- The losses become

SOLVING PARAMETERIZED PROBLEMS

- For $f(x) = 1$, for different values of l we have different solutions

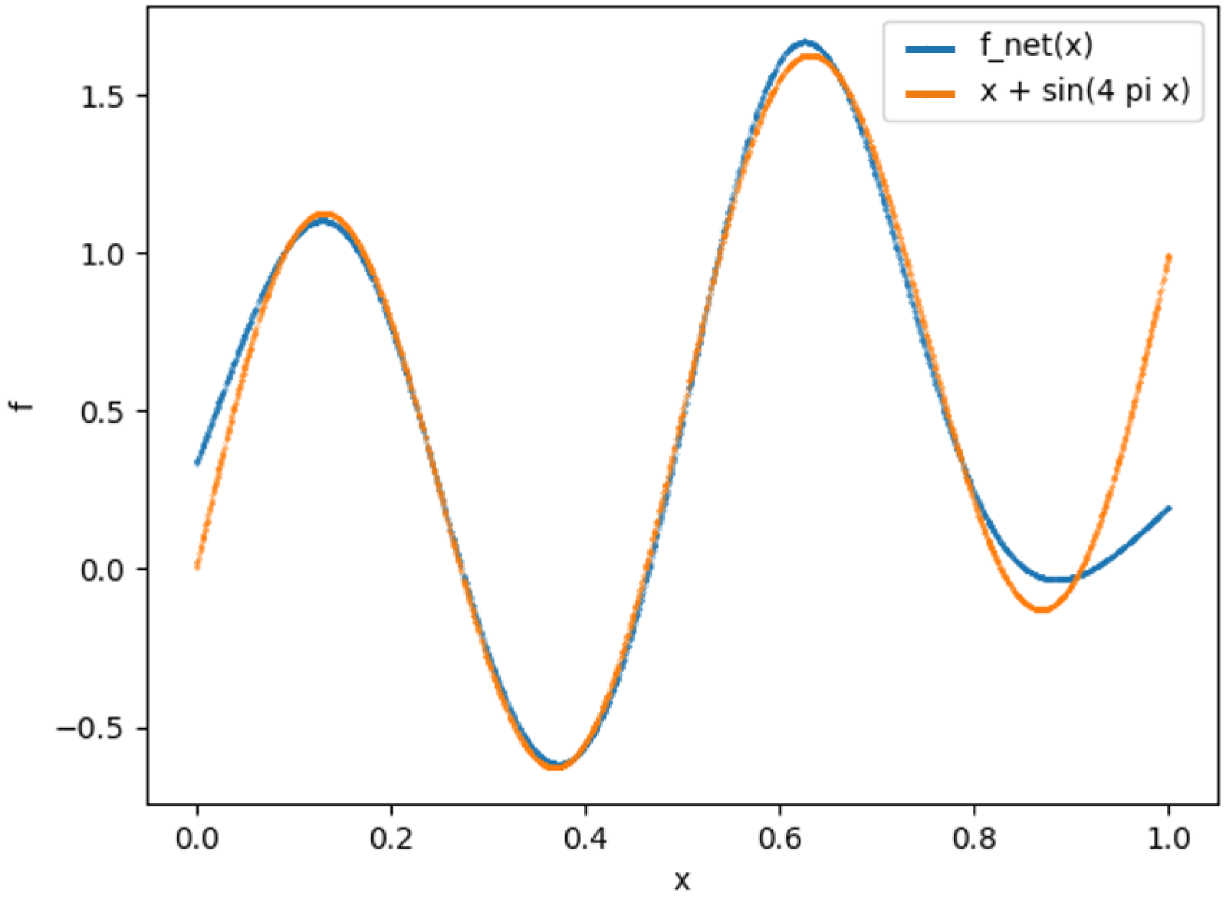


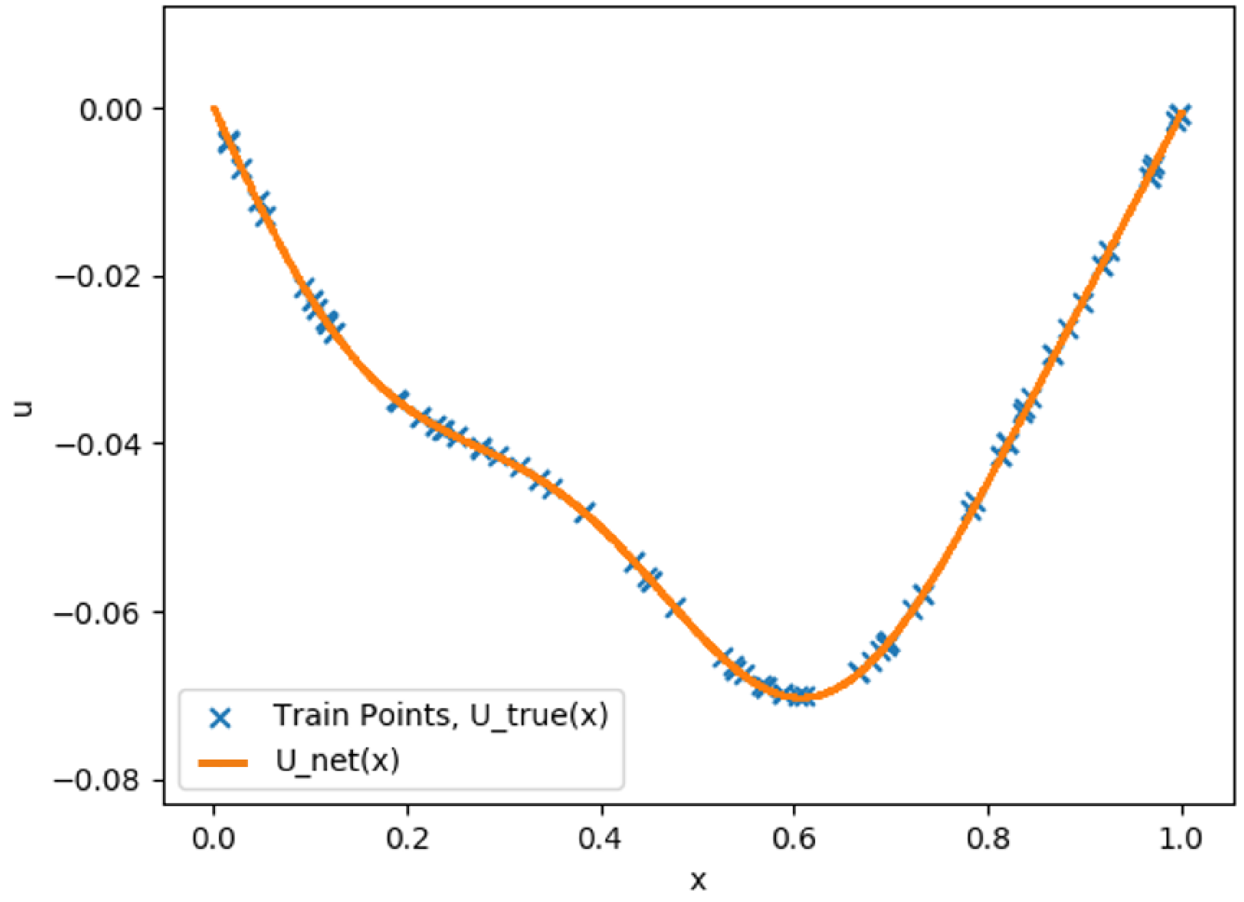
SOLVING INVERSE PROBLEMS

- For inverse problems, we start with a set of observations and then calculate the causal factors that produced them
- For example, suppose we are given the solution $u_{true}(x)$ at 100 random points between 0 and 1 and we want to determine the $f(x)$ that is causing it
- Train two networks $u_{net}(x)$ and $f_{net}(x)$ to approximate $u(x)$ and $f(x)$

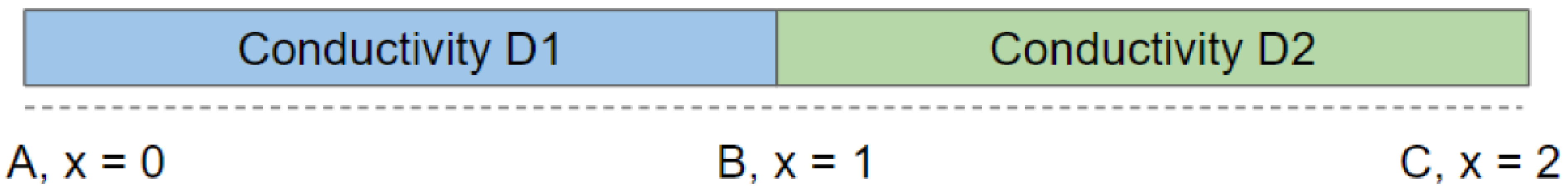
SOLVING INVERSE PROBLEMS

- For $u_{true}(x) = \frac{1}{48}\left(8x(-1+x^2) - \frac{3\sin(4\pi x)}{\pi^2}\right)$ the solution for $f(x)$ is $x + \sin(4\pi x)$





SOLUTION TO PDES- 1D DIFFUSION

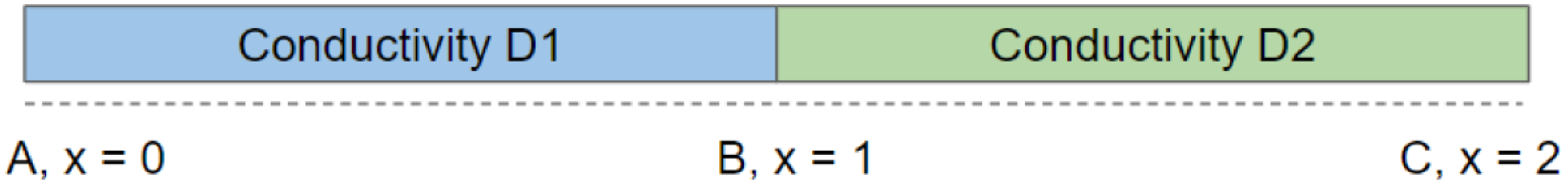


- Composite bar with material of conductivity $D_1 = 10$ for $x \in (0,1)$ and $D_2 = 0.1$ for $x \in (1,2)$. Point A and C are maintained at temperatures of 0 and 100 respectively

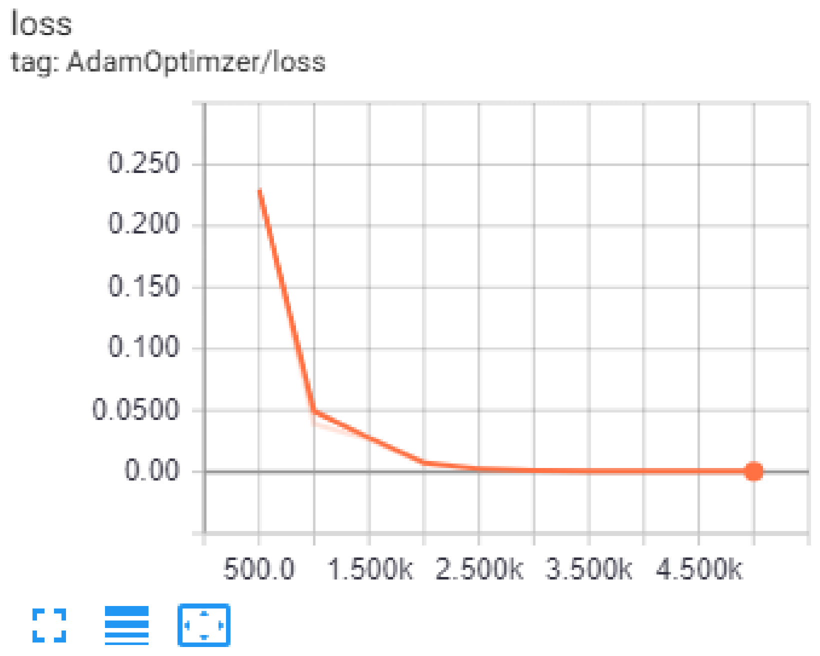
- Equations: Diffusion equation in 1D

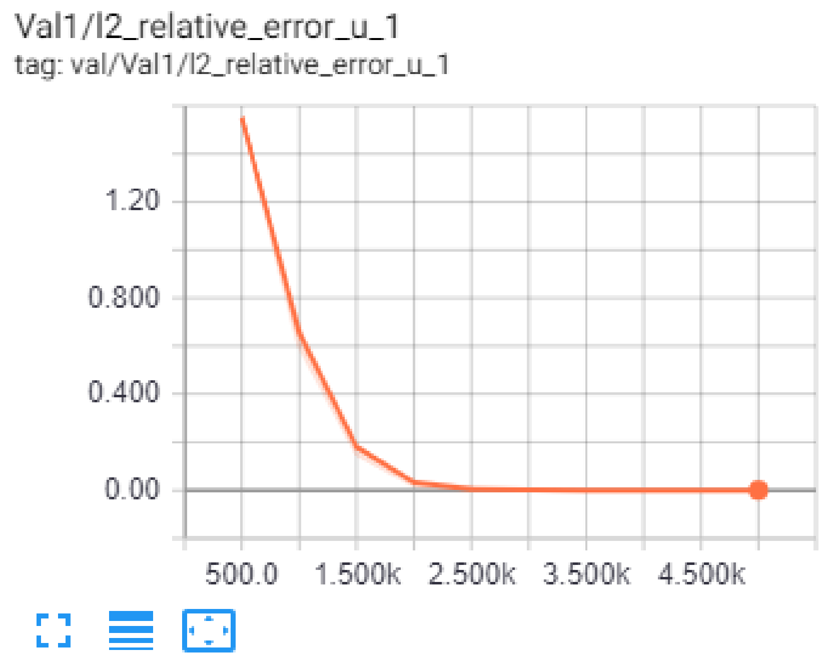
- Flux and field continuity at interface $(x = 1)$

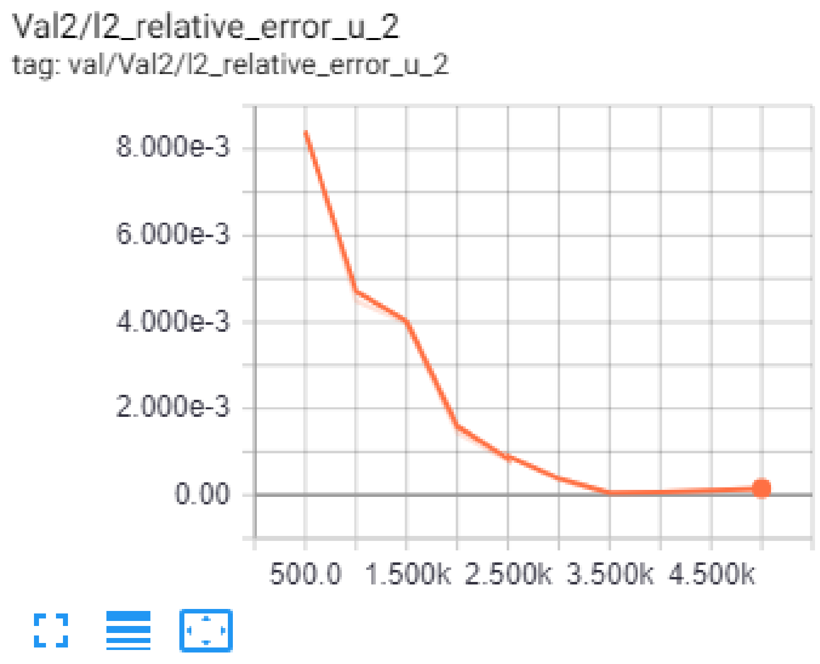
SOLUTION TO PDES- 1D DIFFUSION

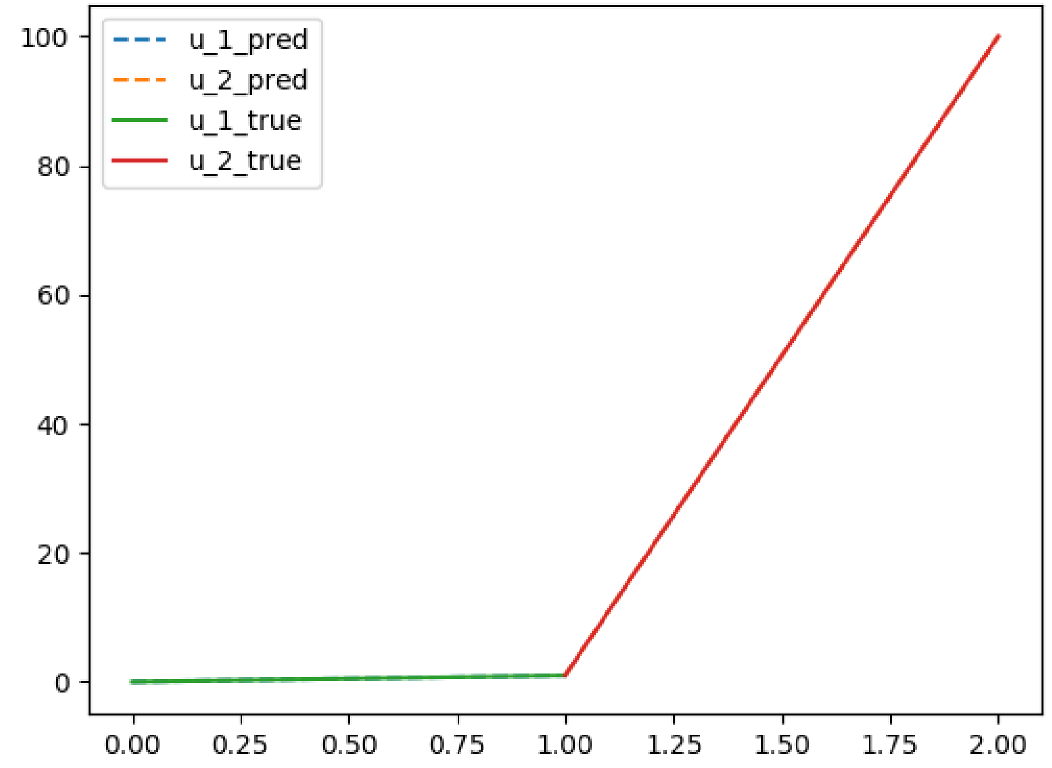


- Define the problem and train the neural network to obtain the temperature distribution in the bar
- Compare the results with analytical solution

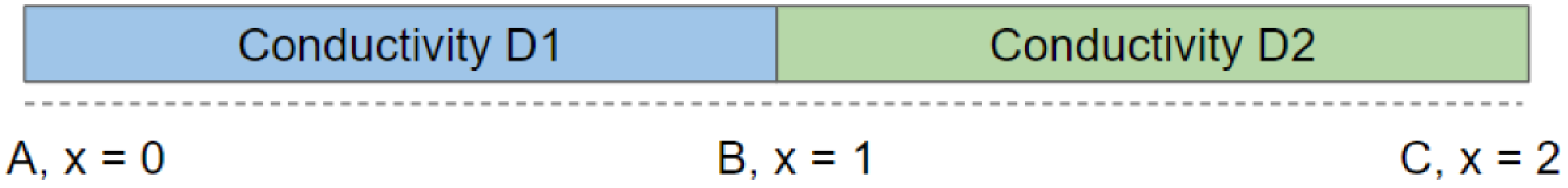




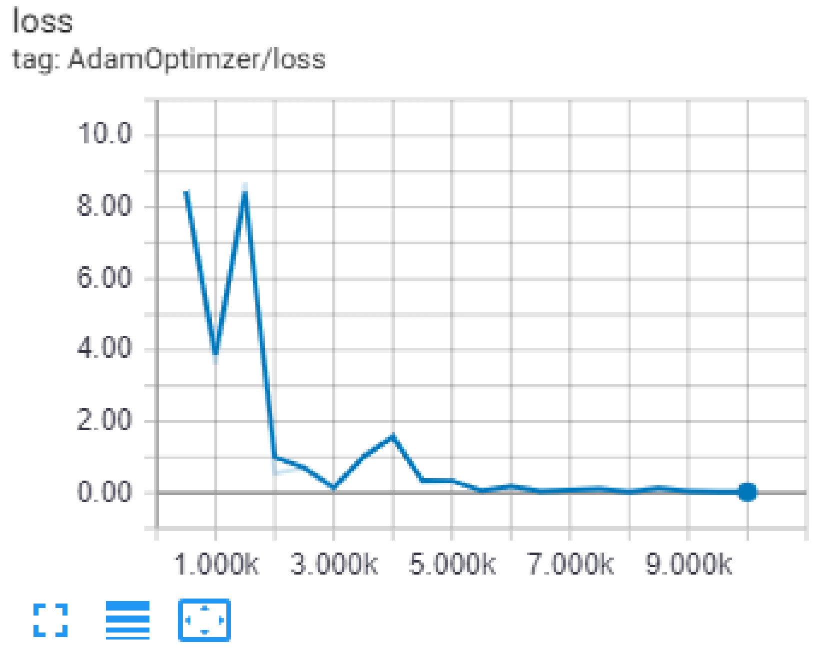


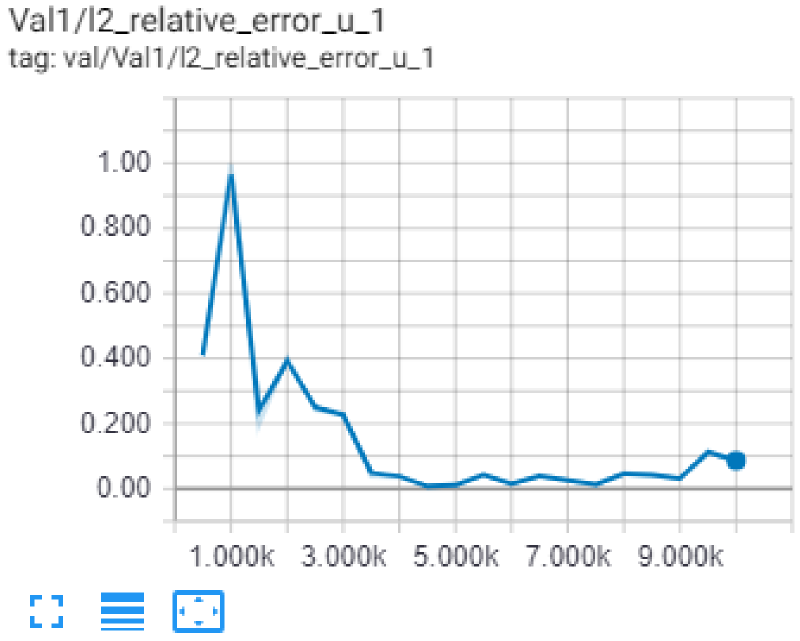


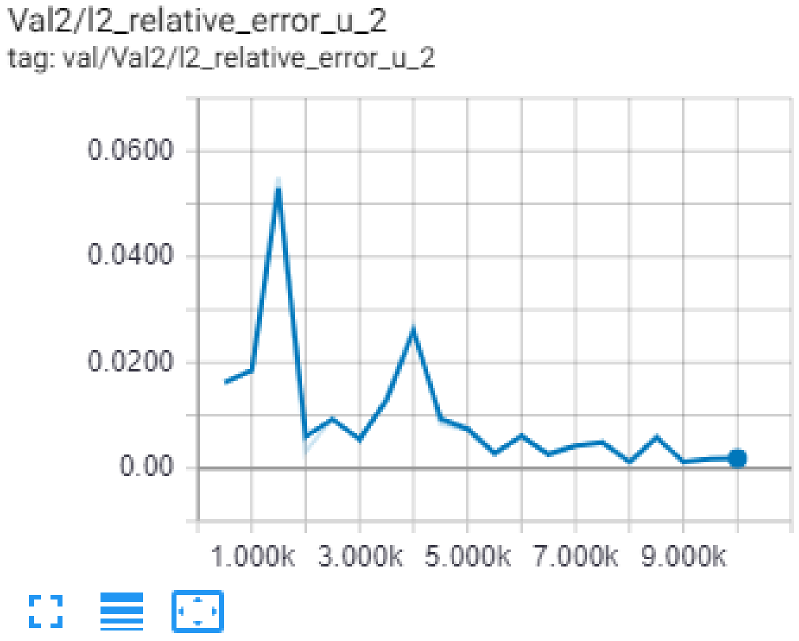
SOLUTION TO PARAMETERIZED PDES- 1D DIFFUSION



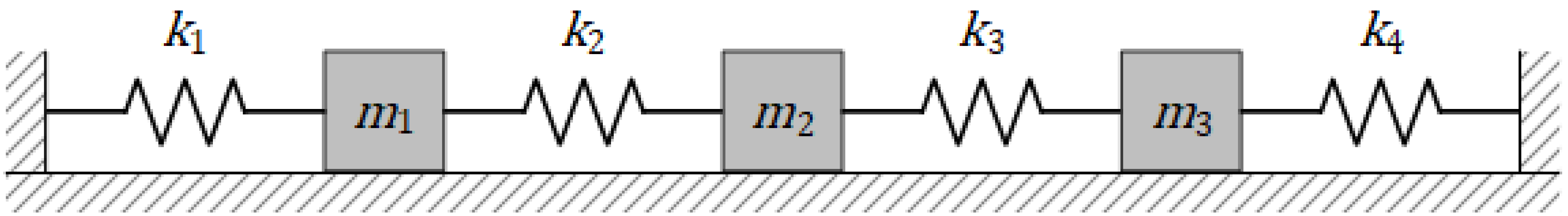
- Composite bar with material of conductivity D_1 for $x \in (0,1)$ and $D_2 = 0.1$ for $x \in (1,2)$.
- Solve the problem for multiple values of D_1 in the range $(5, 25)$ in a single training
- Same boundary and interface conditions as before





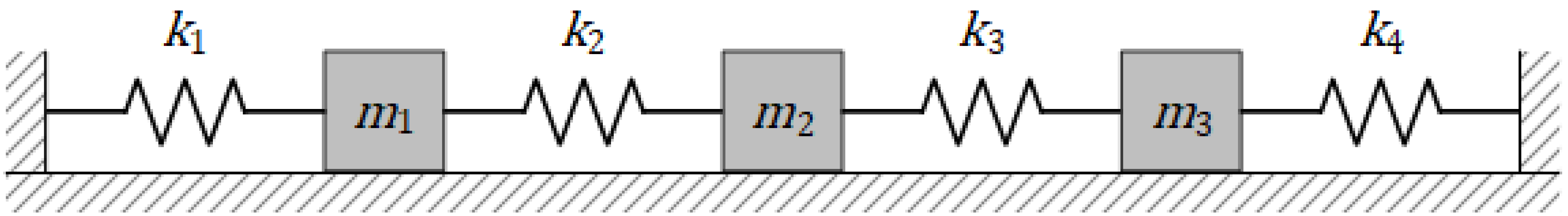


SOLUTION TO ODES- COUPLED SPRING MASS SYSTEM



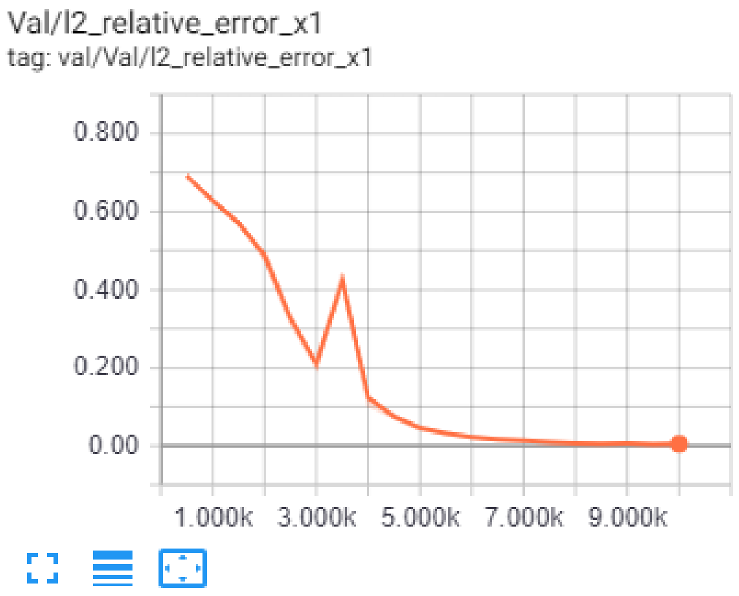
- Three masses connected by four springs
- System's equations (ordinary differential equations):
- For given values masses, spring constants and boundary conditions

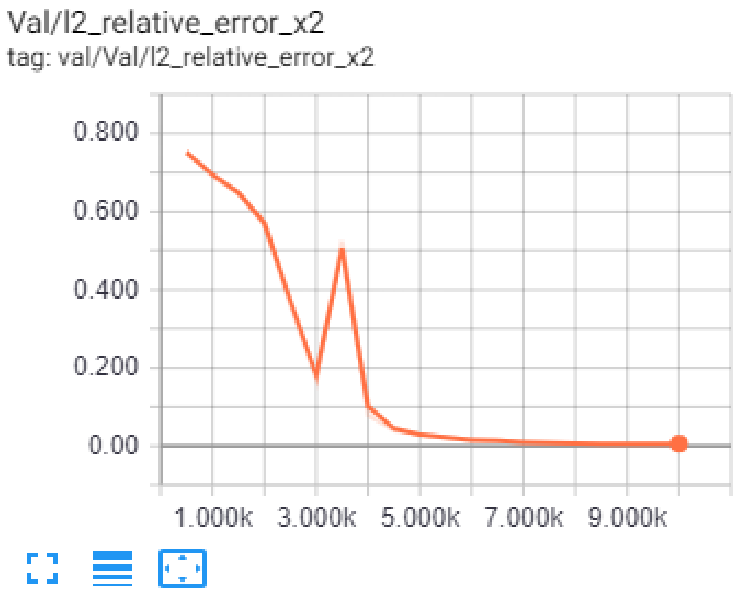
SOLUTION TO ODES- COUPLED SPRING MASS SYSTEM

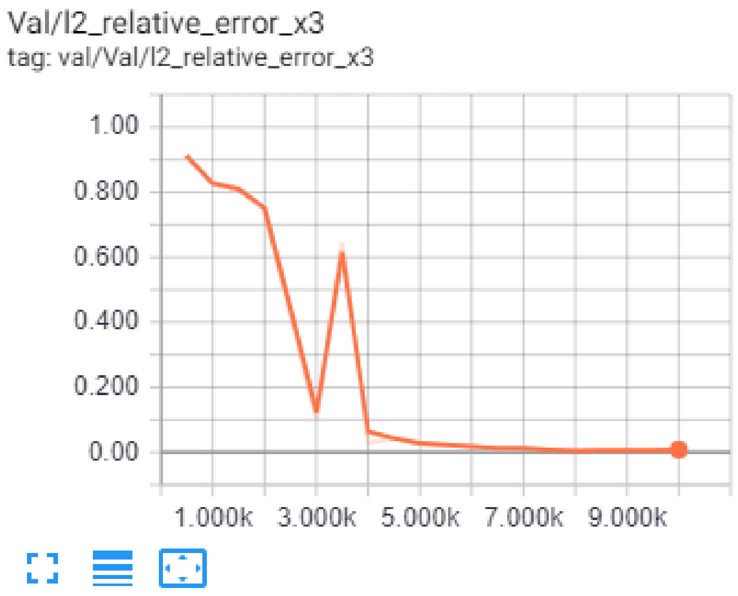


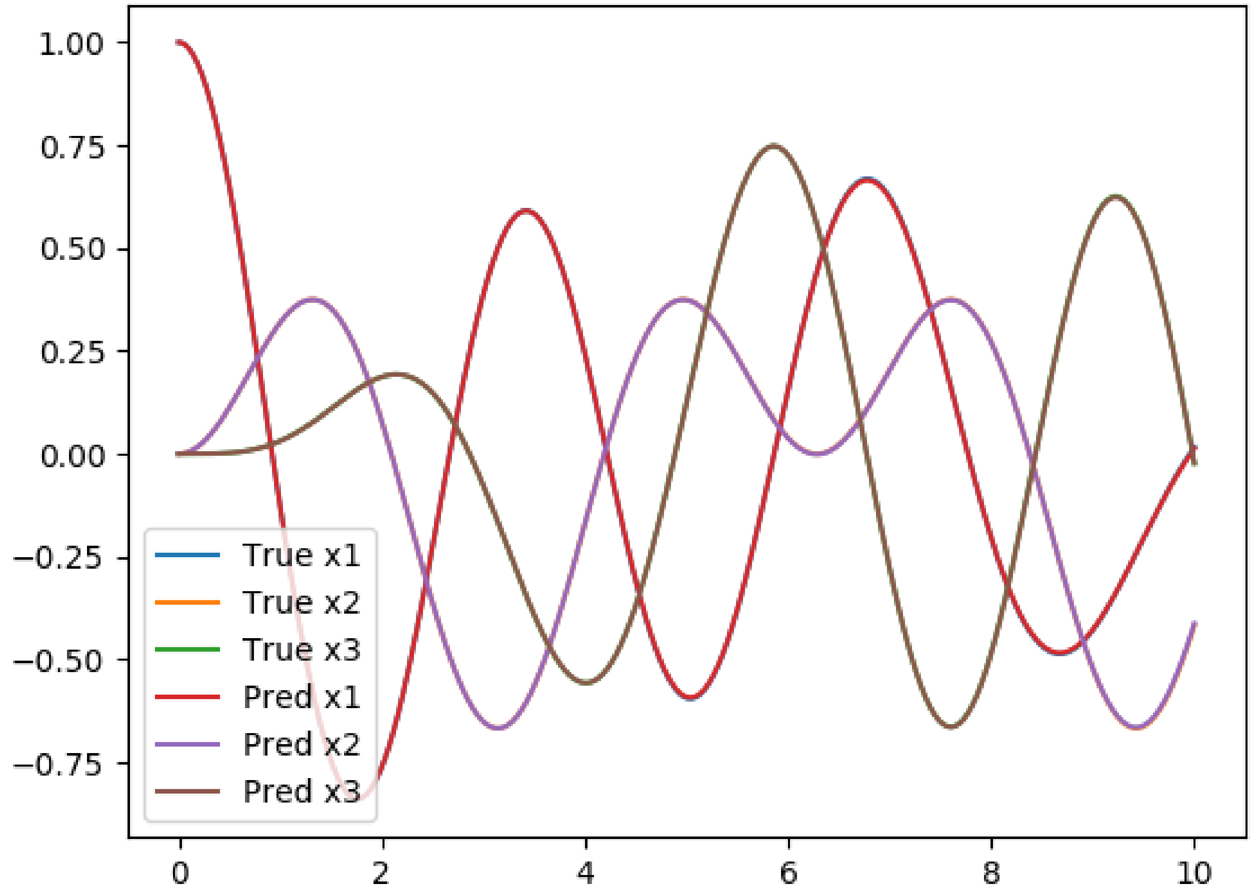
- Define the transient problem for time, $t = (0, 10)$ and train the neural network to obtain the displacement of each mass

- Compare the results with analytical solution

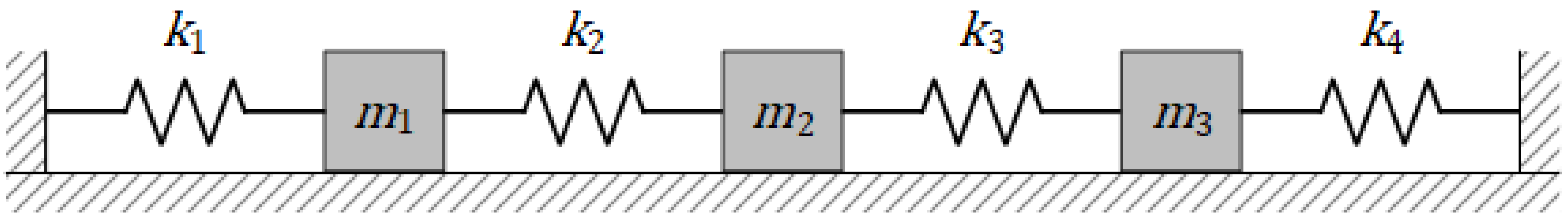






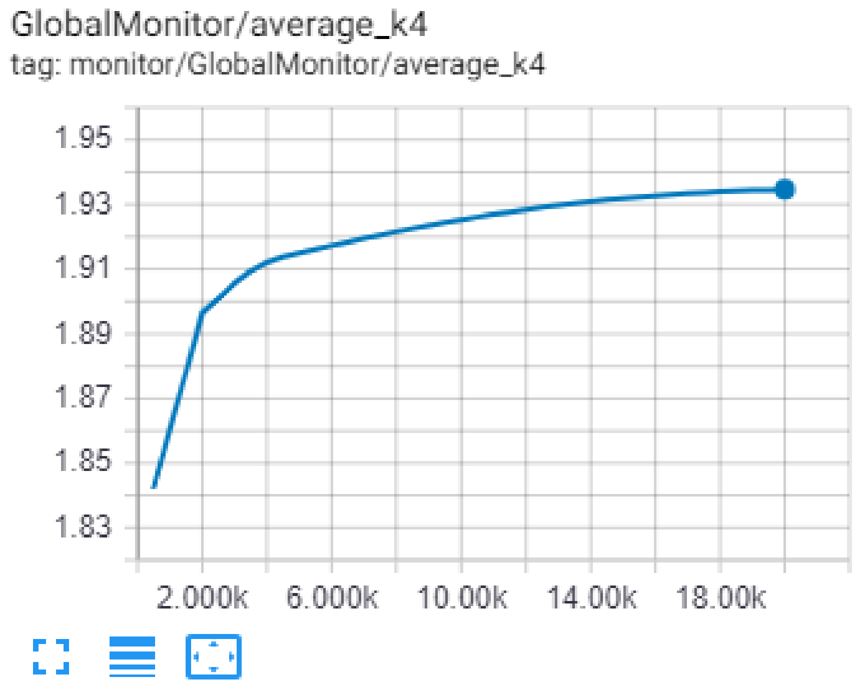


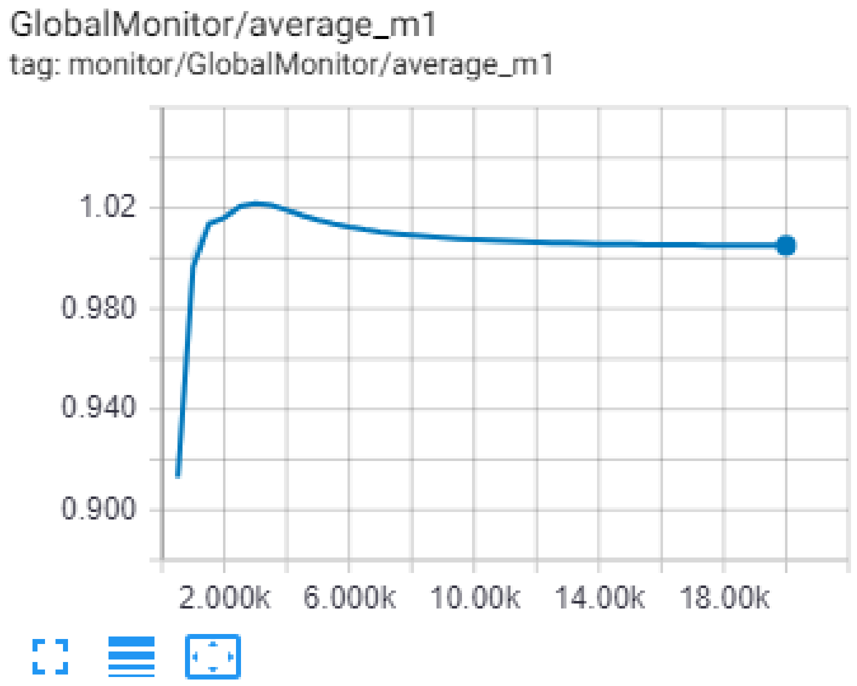
INVERSE PROBLEMS- COUPLED SPRING MASS SYSTEM



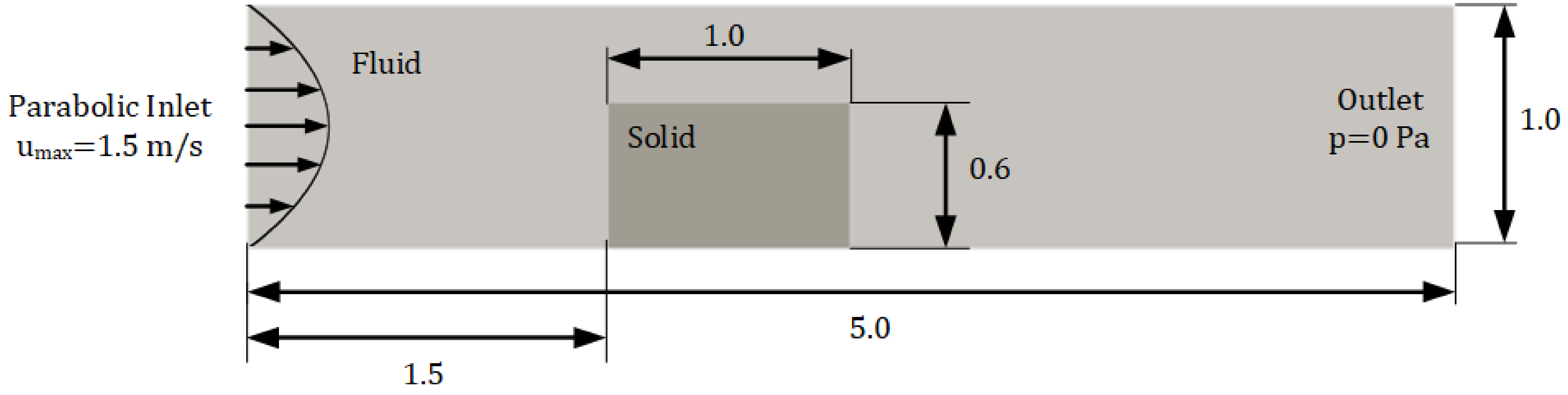
- For the same system, assume we know the analytical solution which is given by:

- With the above data and the values for m_2, m_3, k_1, k_2, k_3 same as before, use the neural network to find the values of m_1 and k_4





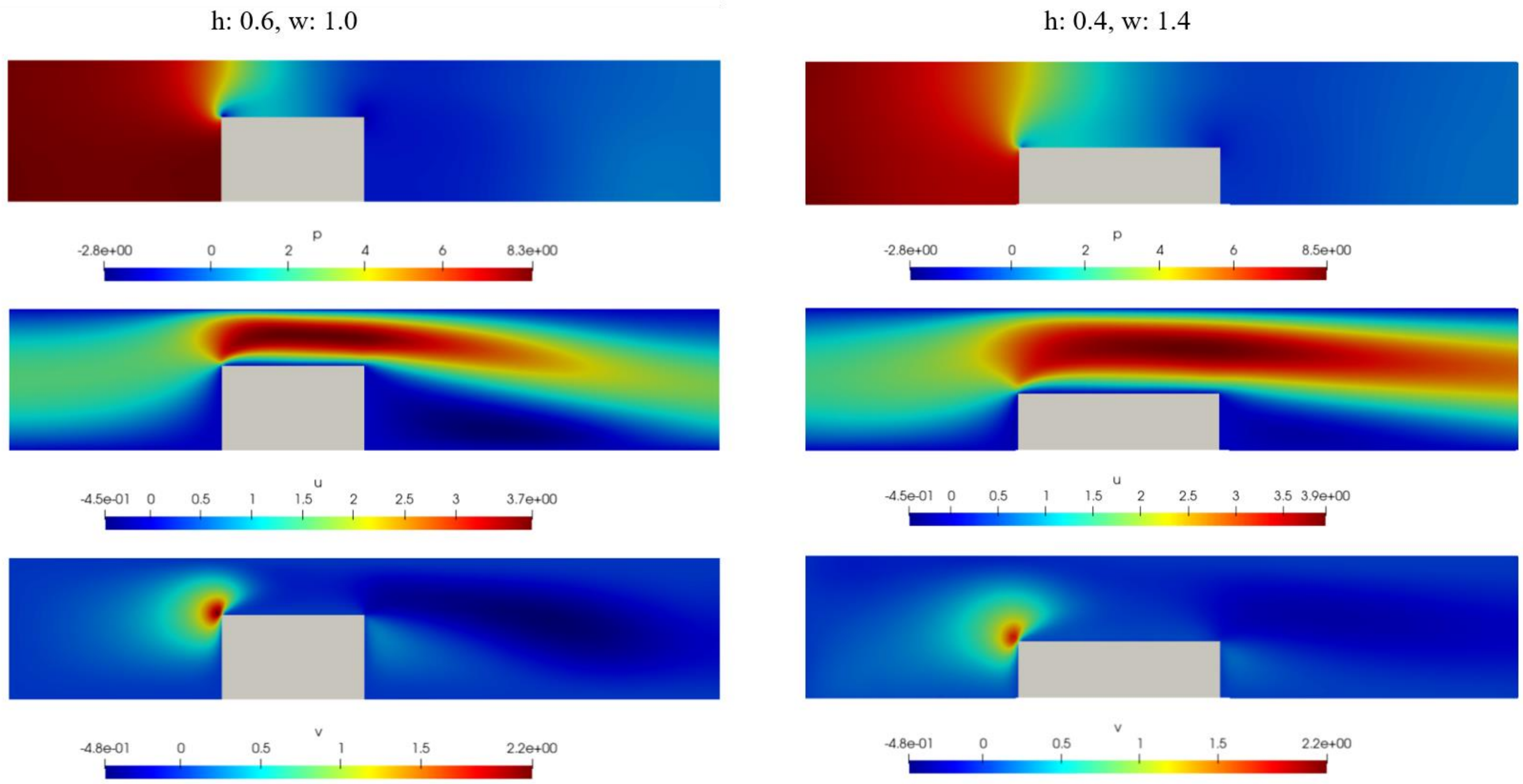
CHALLENGE: FLOW OVER 2D CHIP



- Solve the flow over 2D chip for the given boundary conditions. The challenge problem has 3 parts:

1. Solve the fluid flow for the given boundary conditions and geometry
2. Solve the fluid flow for the parameterized Chip geometry
3. Solve the inverse problem where, given a flow field, use it to invert out the viscosity of the flow

CHALLENGE: FLOW OVER 2D CHIP



CHALLENGE: FLOW OVER 2D CHIP- HINTS AND TIPS

- Use Signed Distance Function to weight the equation losses inside domain for faster convergence (User Guide Section 2.3.2)
- Use Integral Continuity for faster convergence (User Guide Section 8.3.1 and 8.3.2)

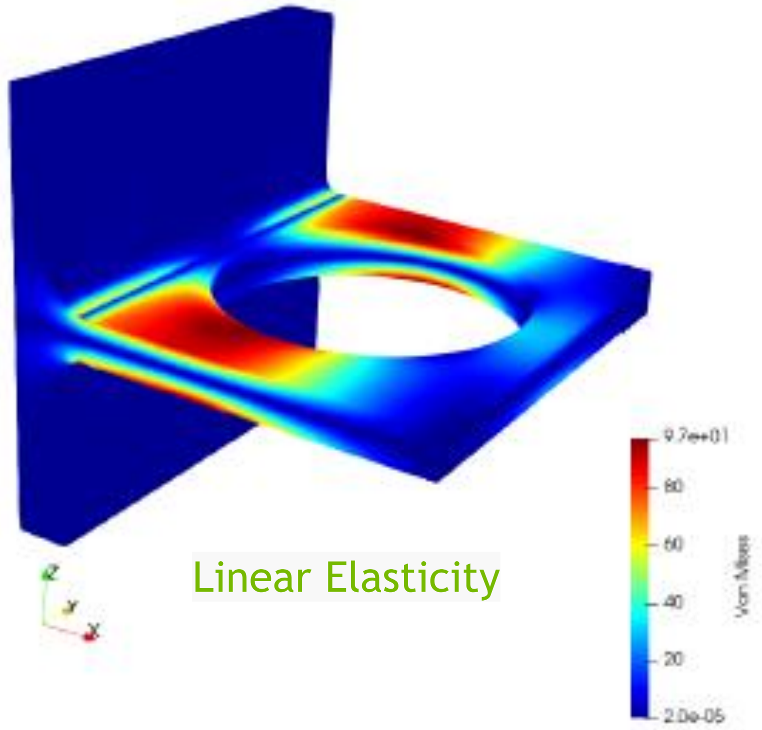
SIMNET FEATURES AND ADVANCEMENTS

Physics types:

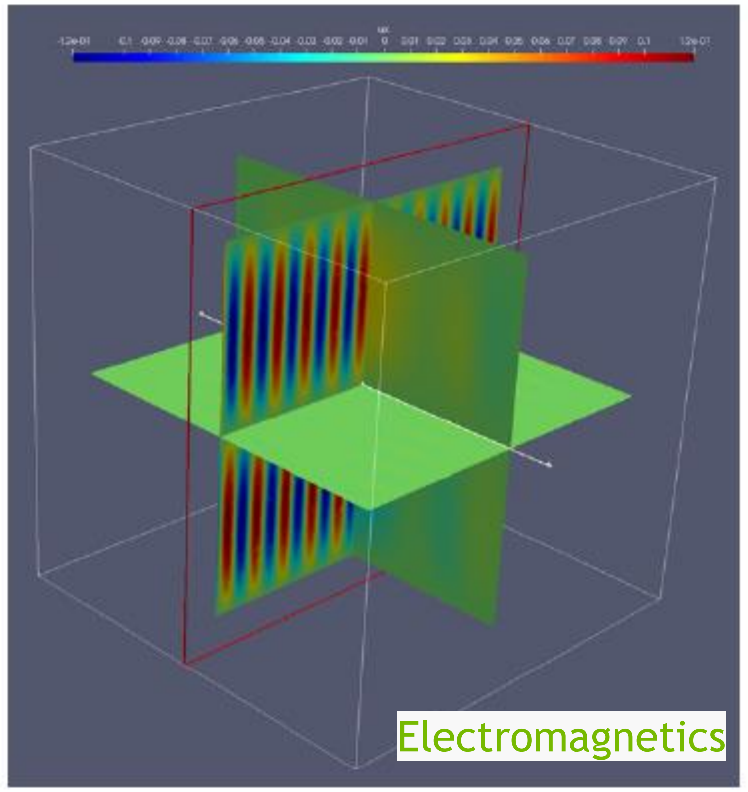
- Linear Elasticity (plane stress, plane strain and 3D)
- Fluid Mechanics
- Heat Transfer
- Coupled Fluid-Thermal
- Electromagnetics
- 2D wave propagation

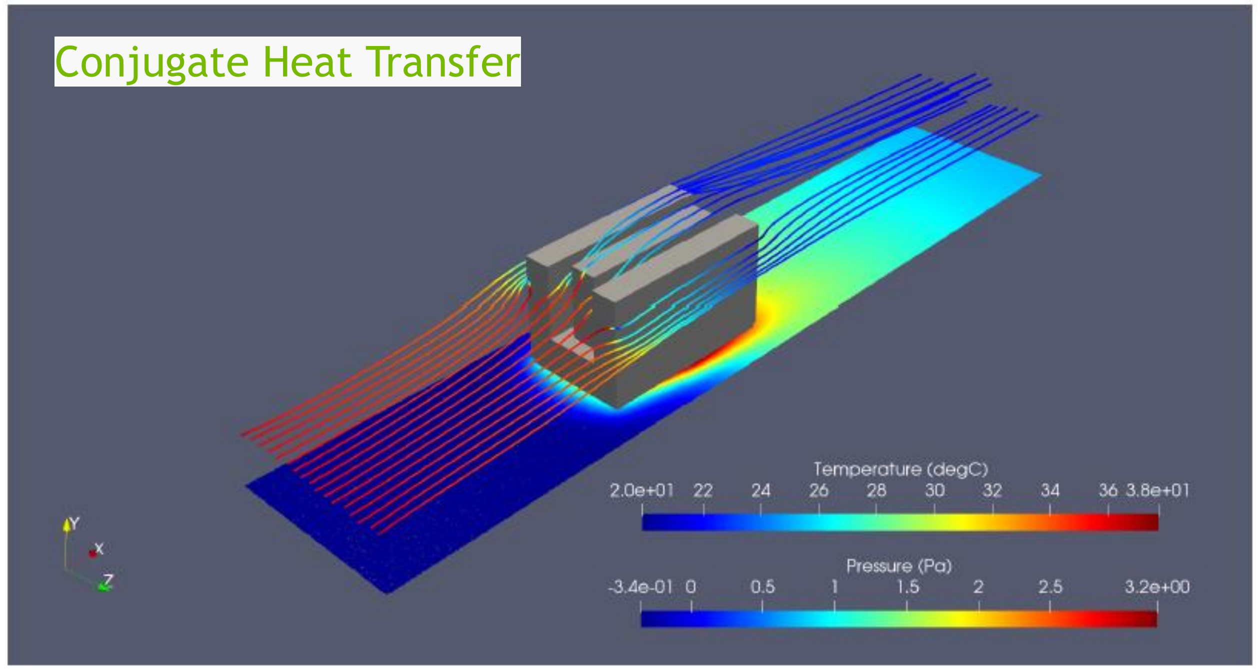
Solution of differential equations:

- Ordinary Differential Equations
- Partial Differential Equations
- Differential (strong) Form
- Integral (weak) form of the PDEs

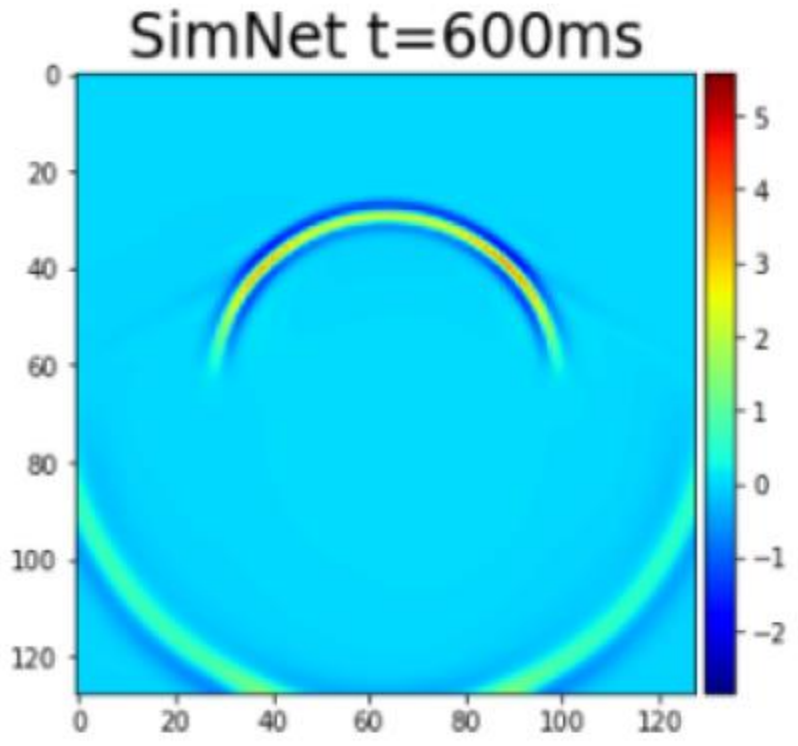


Linear Elasticity

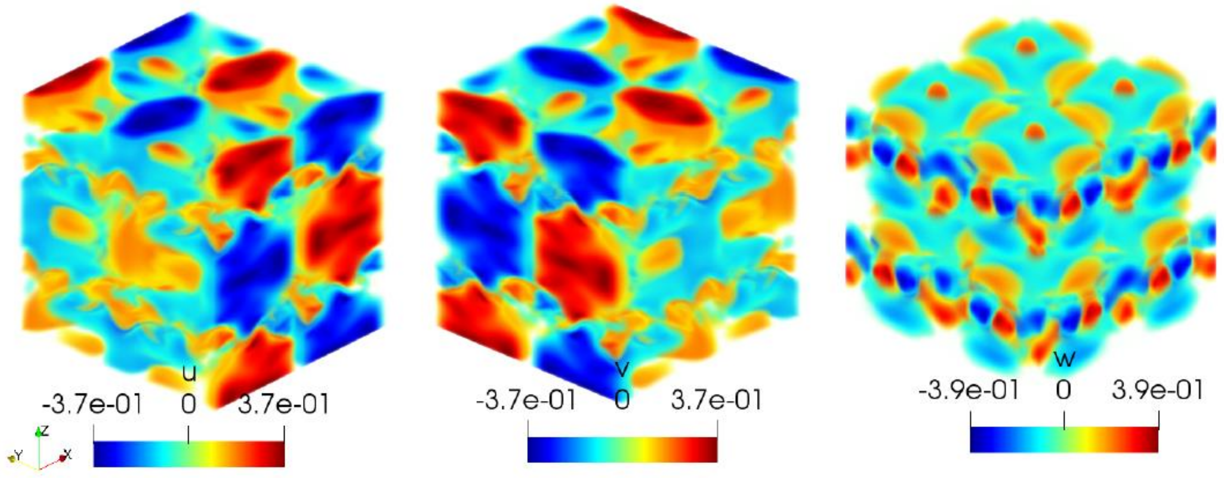

Electromagnetics



Conjugate Heat Transfer



Wave Propagation



Taylor-Green vortex decay

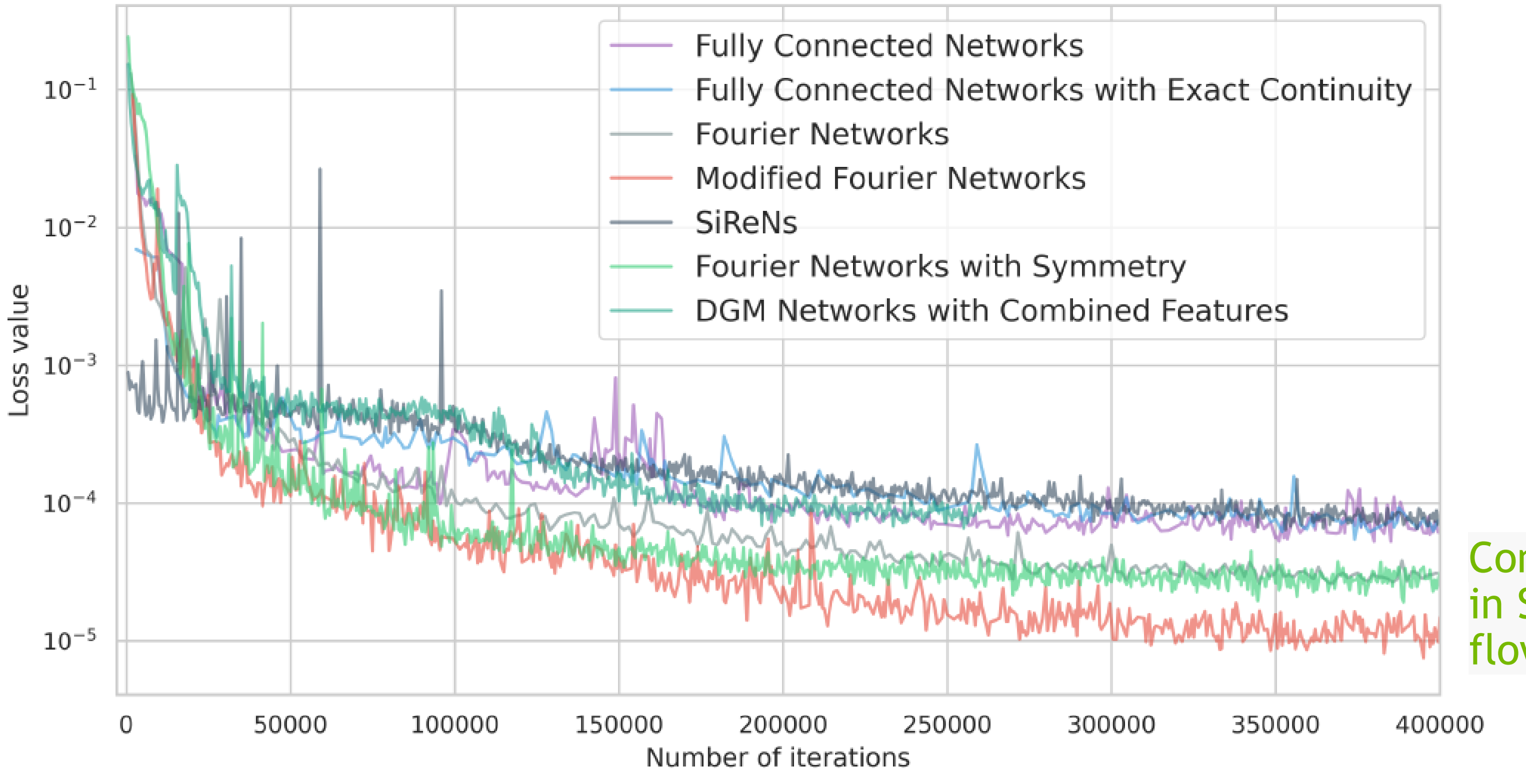
SIMNET FEATURES AND ADVANCEMENTS

Several neural network architectures:

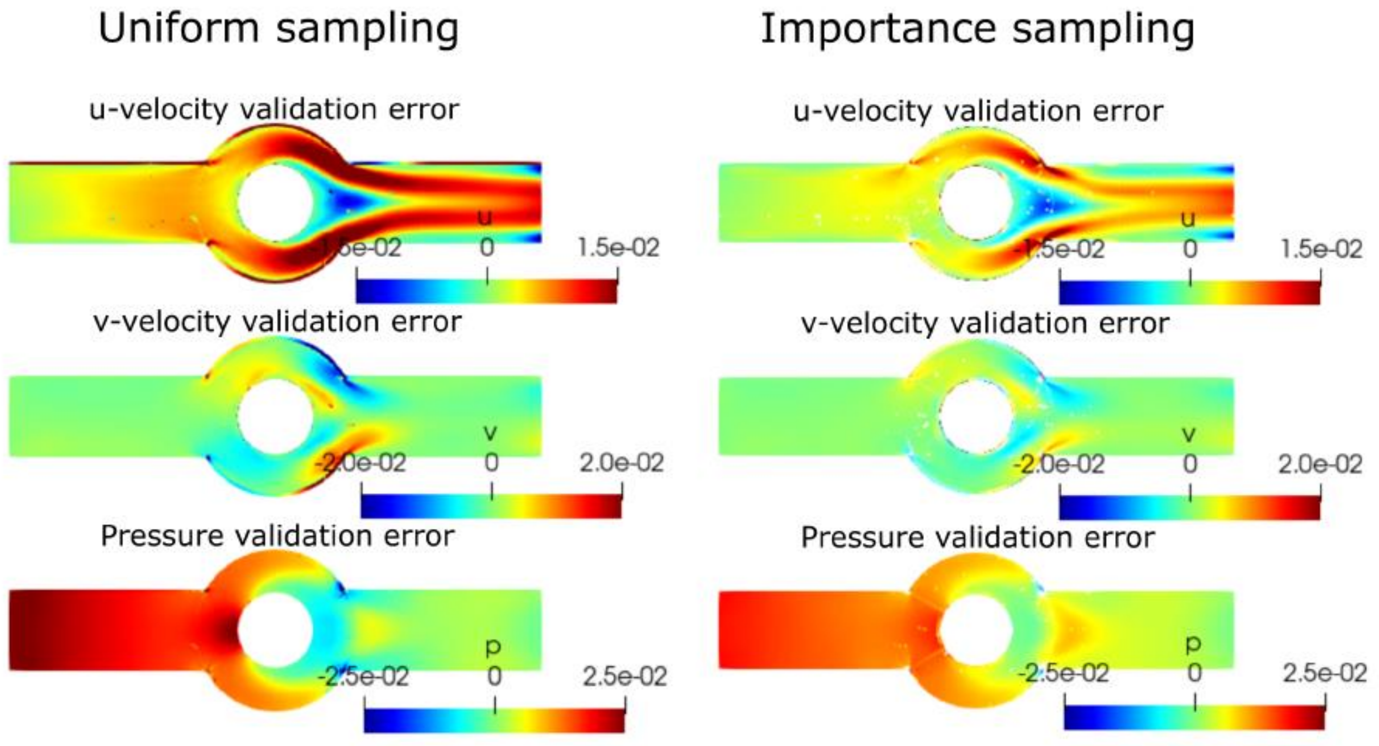
- Fully connected Network
- Fourier Feature Network
- Sinusoidal Representation Network (SiReN)
- Modified Fourier Network
- Deep Galerkin Method Network
- Modified Highway Network
- Multiplicative Filter Networks

Other Features include:

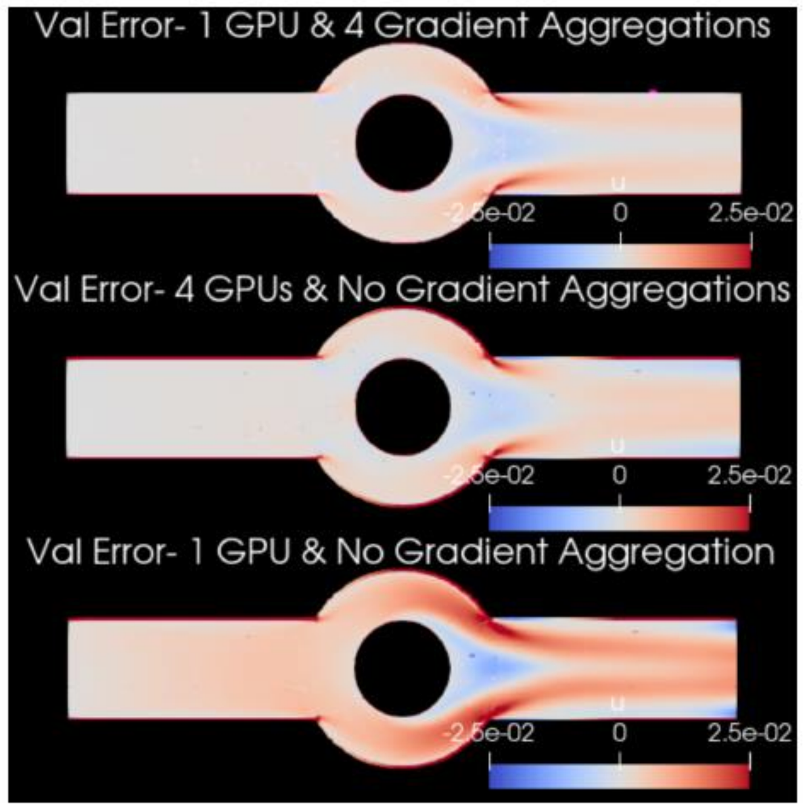
- Global and local learning rate annealing
- Global adaptive activation functions
- Halton sequences for low-discrepancy point cloud creation
- Gradient Accumulation
- Time-stepping schemes for transient problems
- Temporal loss weighting and time marching for the continuous time approach
- Importance sampling
- Homoscedastic task uncertainty quantification for loss weighting



Comparisons of various networks in SimNet applied to solve the flow over a heatsink



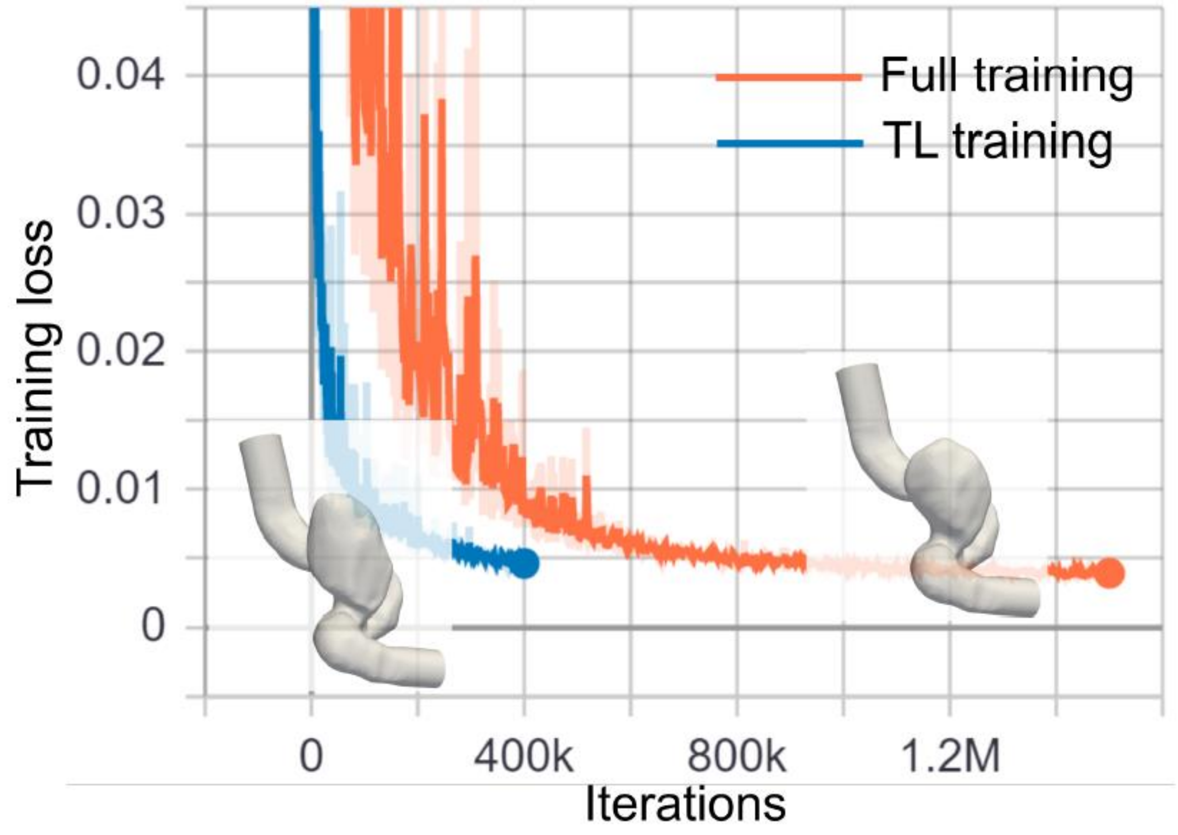
Importance sampling



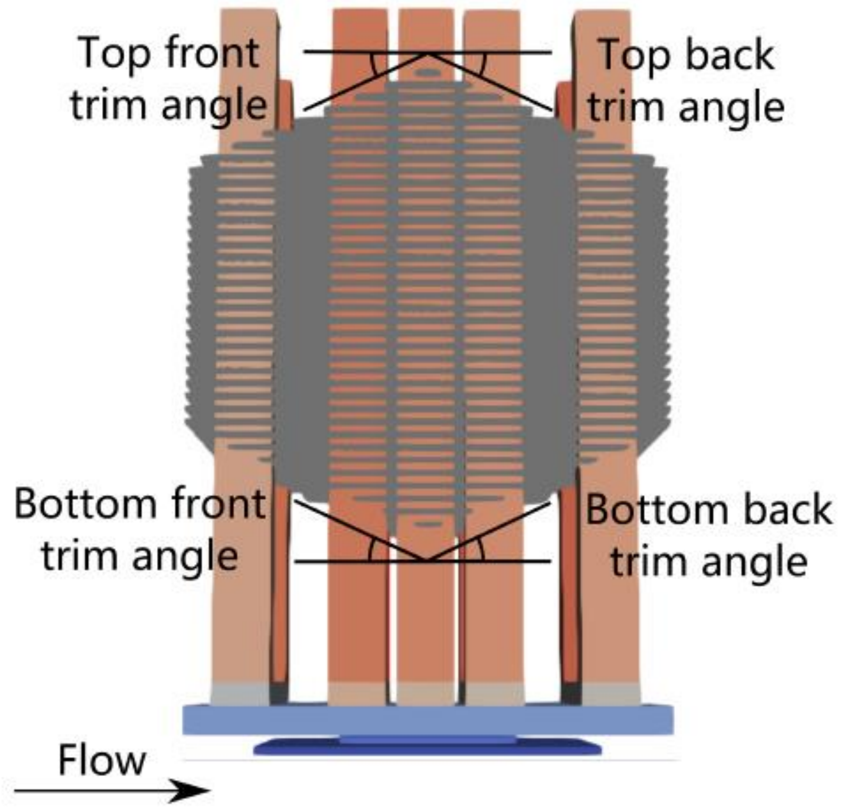
Handling larger batch sizes using Multi-GPU and/or Gradient Aggregation

SIMNET FEATURES AND ADVANCEMENTS

- APIs to automatically generate point clouds from Boolean compositions of geometry primitives or import point cloud for complex geometry (e.g., STL files)
- Parameterized system representation that solves several configurations concurrently for analytical geometry using SimNet CSG module
- Transfer learning for efficient surrogate-based parameterization of STL and constructive solid geometries
- Polynomial Chaos Expansion method for assessing how uncertainties in a model input manifest in its output



STL geometry and Transfer Learning



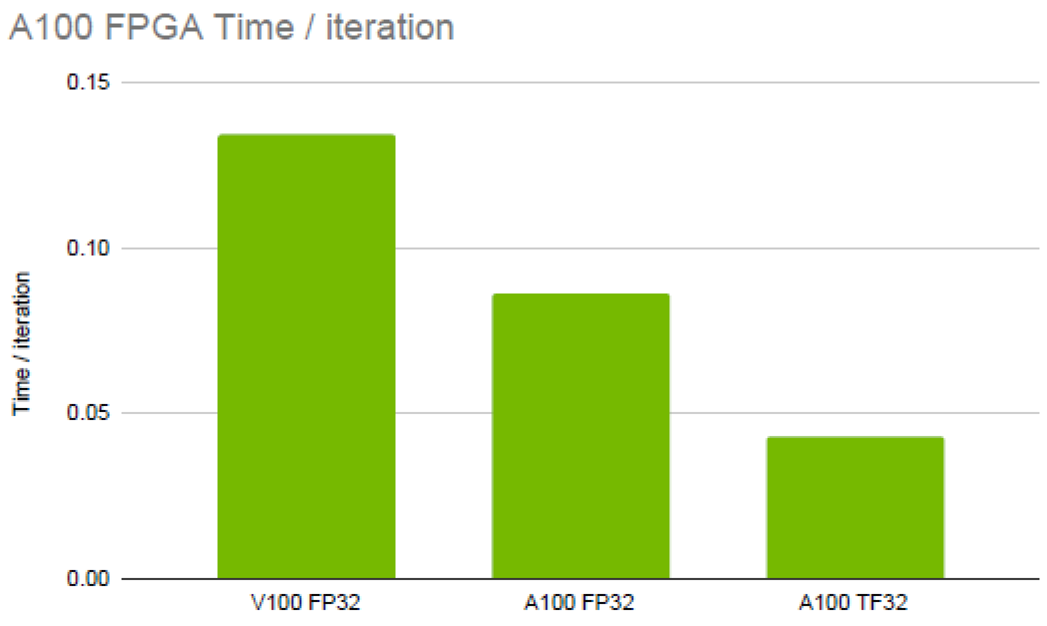
Geometry parameterization using SimNet's CSG module

SIMNET FEATURES AND ADVANCEMENTS

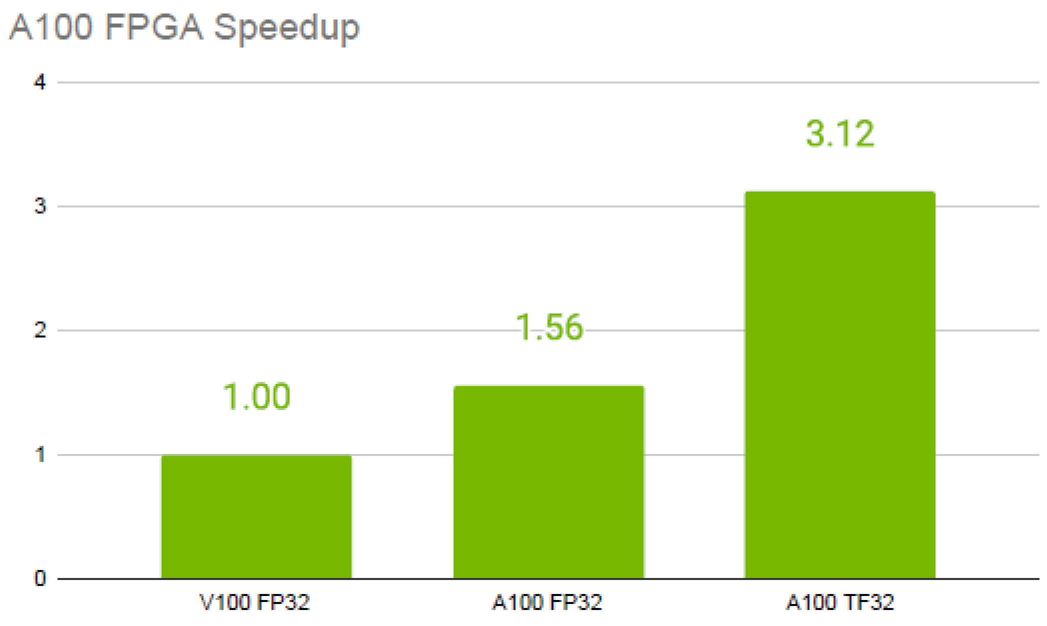
Improved performance with XLA enabled for TensorFlow models and multi-GPU/multi-Node runs

- Accelerated Linear Algebra (XLA)
- Strong scaling with learning rate adjustments

Improved stability in multi-GPU/multi-Node implementations using linear-exponential learning rate and utilization of TF32 precision for A100 GPUs

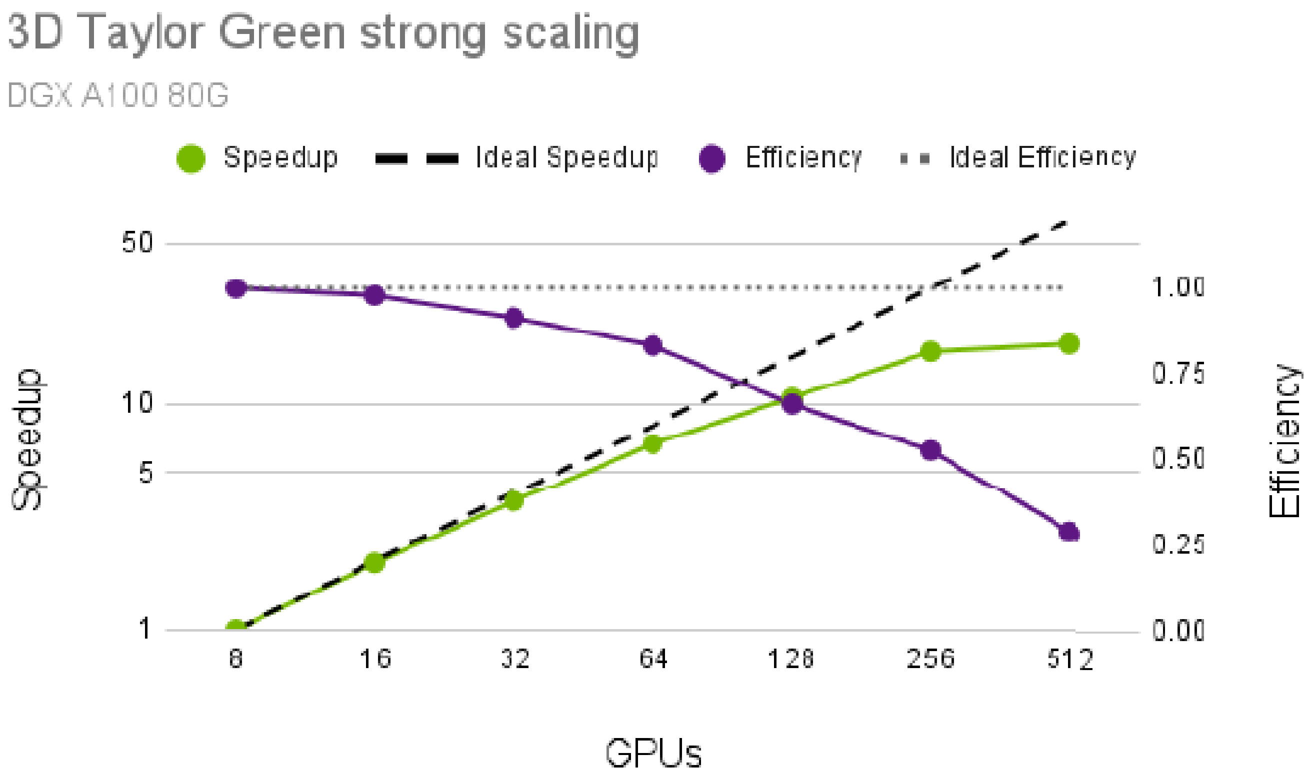


(a) Time per iteration



(b) Speed-up

Accelerated training using TF32 on A100 GPUs



Strong Scaling- Speedup and Scaling Efficiency

Thanks!