HPC@LR - CYCLE DE FORMATION AUTOUR DU HPC 2016/2017
DECEMBER 12TH 2016

Comment accélérer une application R avec des GPUs :
une application aux réseaux de neurones profonds('Deep Learning')

R, GPU & (D)NN

François Courteille | Principal Solutions Architect, NVIDIA | fcourteille@nvidia.com

NVIDIA

AGENDA

1. Introduction
2. Background of Deep Neural Networks
3. Overview of Deep Neural Networks
4. Build DNN with R
5. GPU Acceleration
6. Scaling out with multi-GPUs
7. Summary
8. NVIDIA Deep Learning Platform

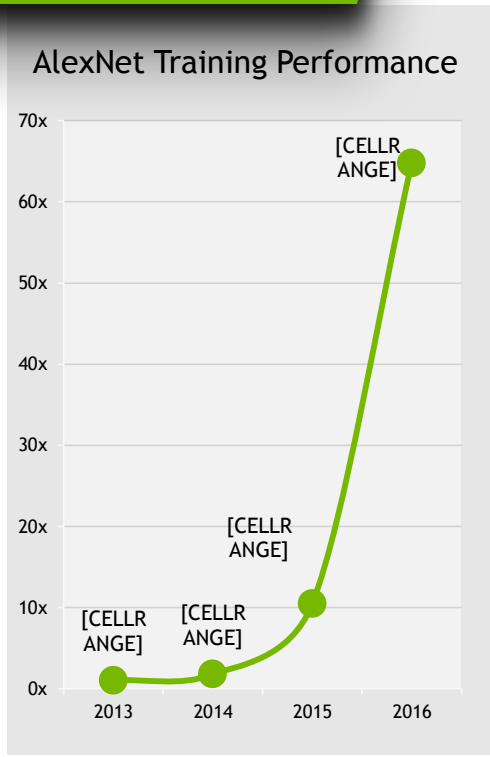
INTRODUCTION

ONE ARCHITECTURE BUILT FOR BOTH DATA SCIENCE & COMPUTATIONAL SCIENCE

Pascal- 5 Miracles

NVIDIA DGX-1

NVIDIA DGX SATURNV



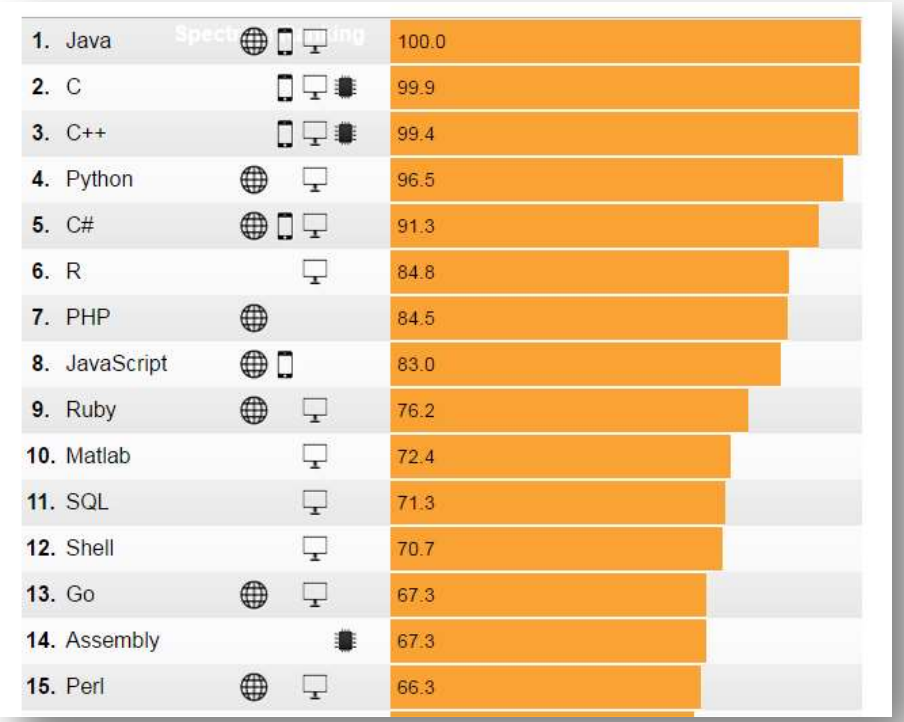
65x in 3 Years

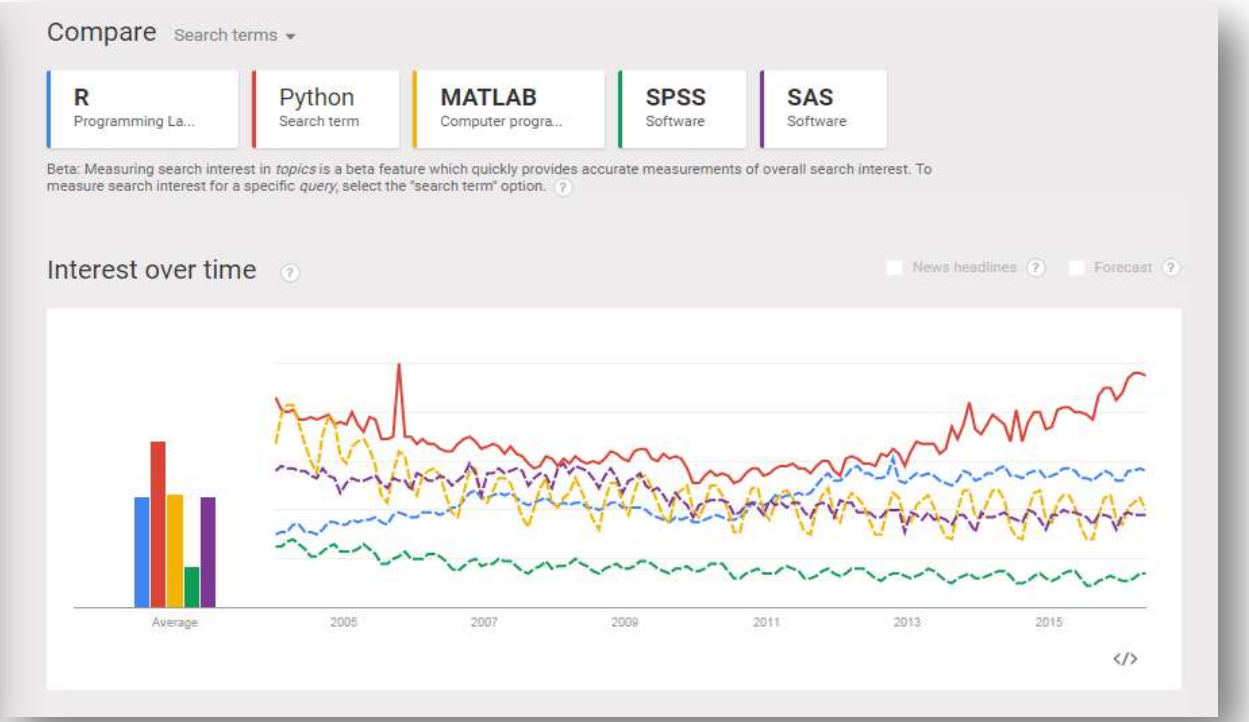
CURRENT TREND IN DATA ANALYTICS

Is R out of date ?

*IEEE Spectrum (http://spectrum.ieee.org/static/interactive-the-top-programming-languages-2015)

*data from google trend shows how often a particular search-term is entered.



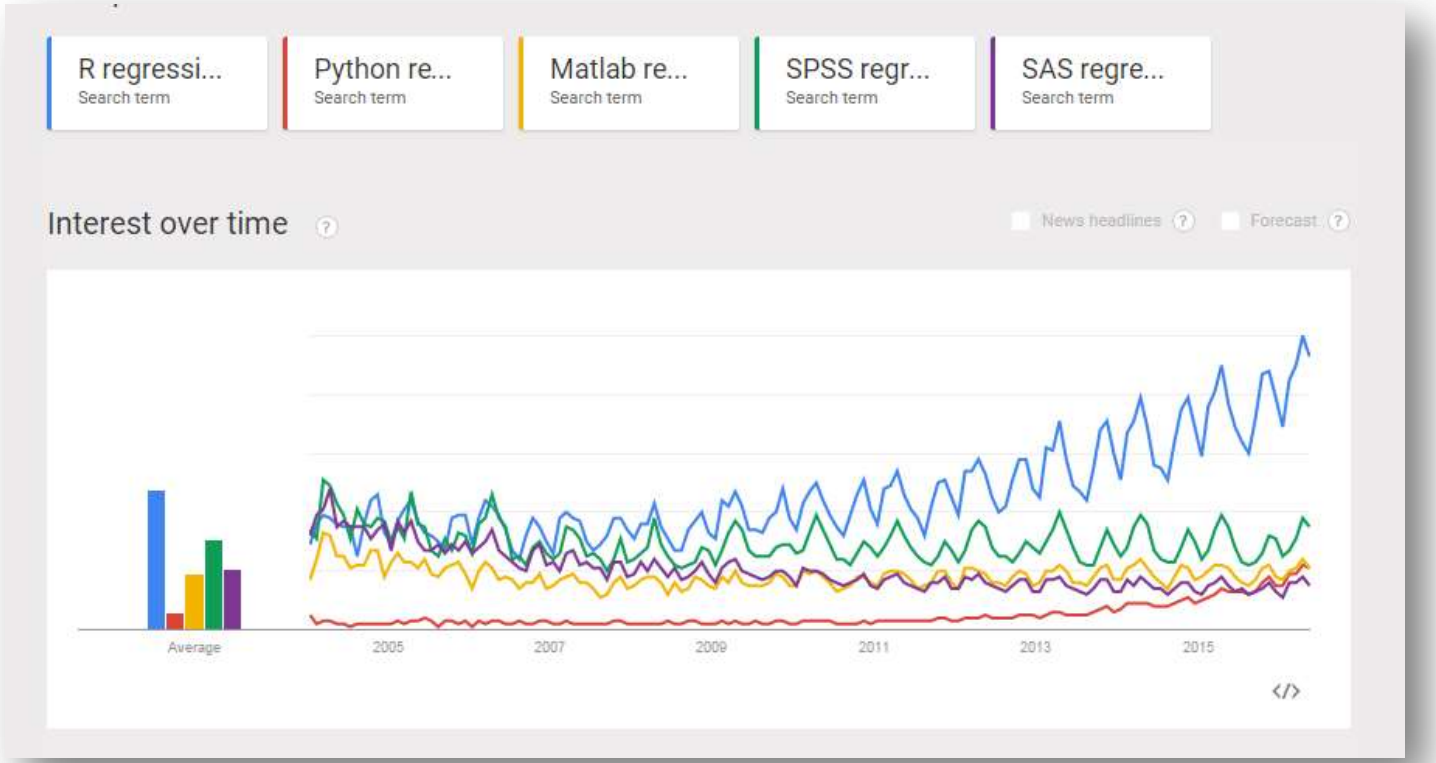


NVIDIA

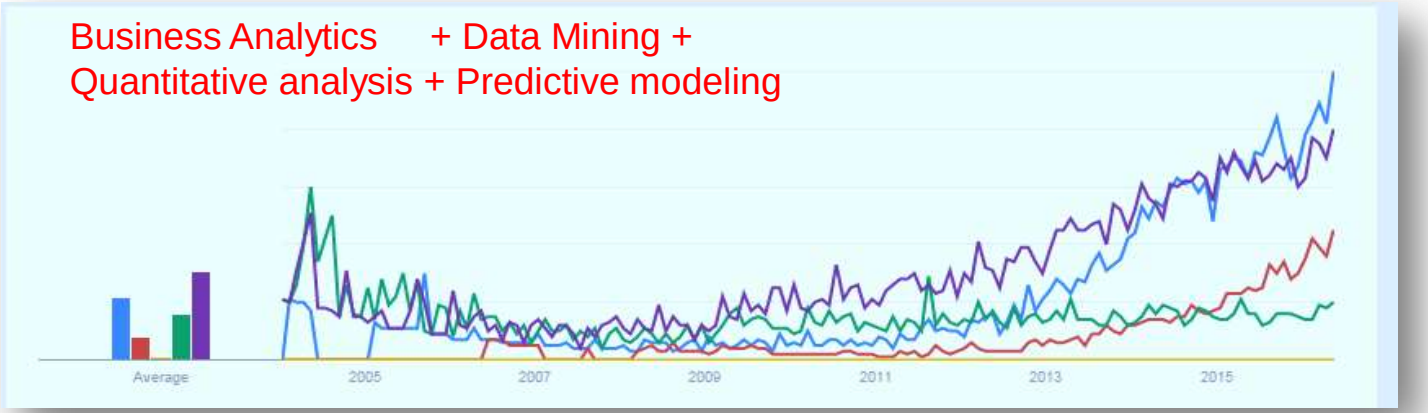
R is still very popular in data analytics fields：

[R | Python | Matlab | SPSS | SAS] +

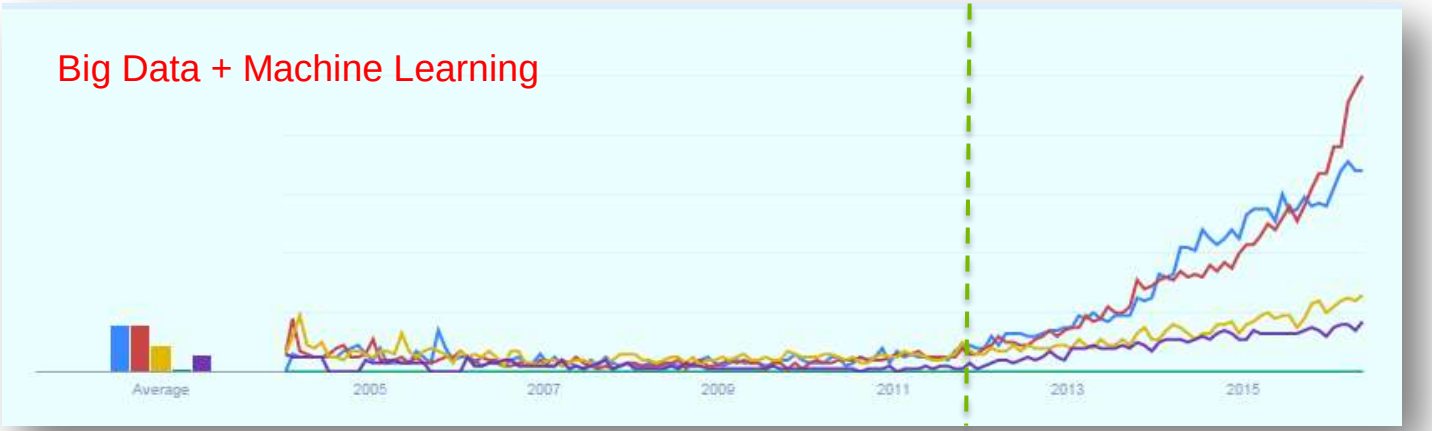
[regression | classification | clustering | time series]



R and **SAS** are widely known on business analytics fields

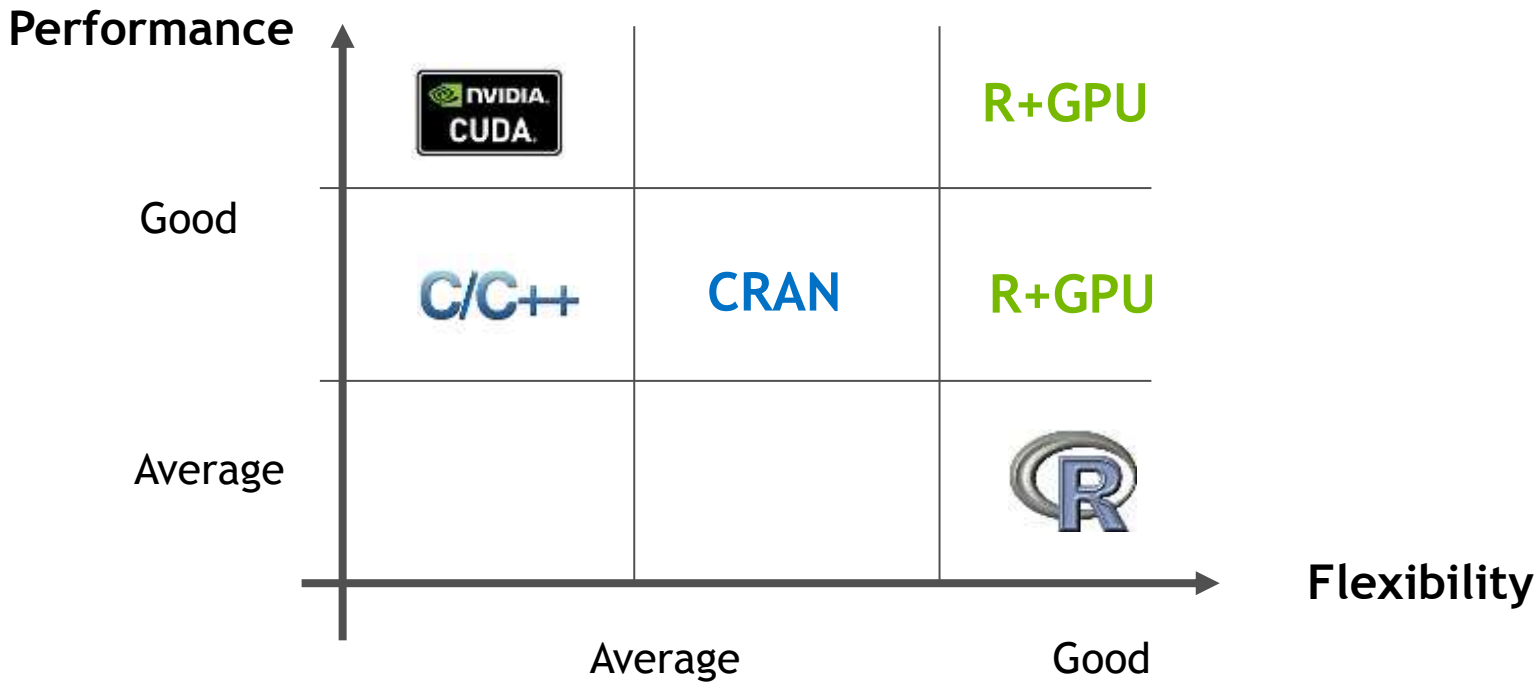


R and **Python** are widely used in big data and deep learning



In this talk, I am going to introduce how to:

- Build DNN network with native R
- Accelerate R code under CUDA ecosystem
- Make a solution as simple as possible



GENERAL R/GPU TOOLS

What's out there now for R/GPU:

gputools

(Buckner *et al.*) The oldest major package. Matrix multiply; matrix of distances between rows; linear model fit; QR decomposition; correlation matrix; hierarchical clustering.

HiPLAR

(Montana *et al.*) R wrapper for **MAGMA** and **PLASMA**. Linear algebra routines, e.g. Cholesky.

rpud

(Yau.) Similar to **gputools**, but has SVM.

Rth

(Matloff.) R interfaces to some various algorithms coded in Thrust. Matrix of distances between rows; histogram; column sums; Kendall's Tau; contingency table.

CURRENT TOOLS (CONT'D.)

gmatrix

(Morris.) Matrix multiply, matrix subsetting, Kronecker product, row/col sums, Hamiltonian MCMC, Cholesky.

RCUDA

(Baines and Temple Lang, currently not under active development.) Enables calling GPU kernels directly from R. (Kernels still written in CUDA.)

rgpu

(Kempenaar, no longer under active development.) "Compiles" simple expressions to GPU.

various OpenCL interfaces

ROpenCL, **gpuR**. Similar to **RCUDA**, but via OpenCL interface.

EXAMPLE: LINEAR REGRESSION VIA GPUTOOLS

```
>  test <— function (n , p) {
        x <— matrix ( runif (n*p ) , nrow=n)
        r e g v a l s +— x %*% rep (1.0 , p)
        y <— r e g v a l s + 0.2 * runif (n)
        xy <— cbind (x , y )
        print (” gputools method” )
        print ( system . time (gpuLm.f i t (x , y )))
        print (” ordinary method” )
        print ( system . time (lm.fit (x , y )))
}
> t e s t (100000 ,1500)
[ 1 ] ” gputools method”
        user system elapsed
      6.280 2.878 17.902

[ 1 ] ” ordinary method”
        user system elapsed
142.282 0.669 142.912
```

KEEPING OBJECTS ON THE DEVICE

Some packages, notably **gputools**, do not take arguments on the device.

So, cannot store intermediate results on the device, thus requiring needless copying.

Some packages remedy this, e.g. **gmatrix**.

EXAMPLE

```
library(gputools)
library(gmatrix)
n <- 5000
z <- matrix(runif(n^2), nrow=n)
# plain R:
system.time(z %*% z %*% z)
# user system elapsed
#138.757 0.322 139.081
system.time(gpuMatMult(gpuMatMult(z, z), z))
# user system elapsed
#6.607 1.170 10.059
zm <- gmatrix(z, nrow=n, ncol=n) # zm2, zm3 not shown
system.time({gmm(zm,zm, zm2); gmm(zm, zm2, zm3)})
# user system elapsed
#6.258 1.031 7.285
```

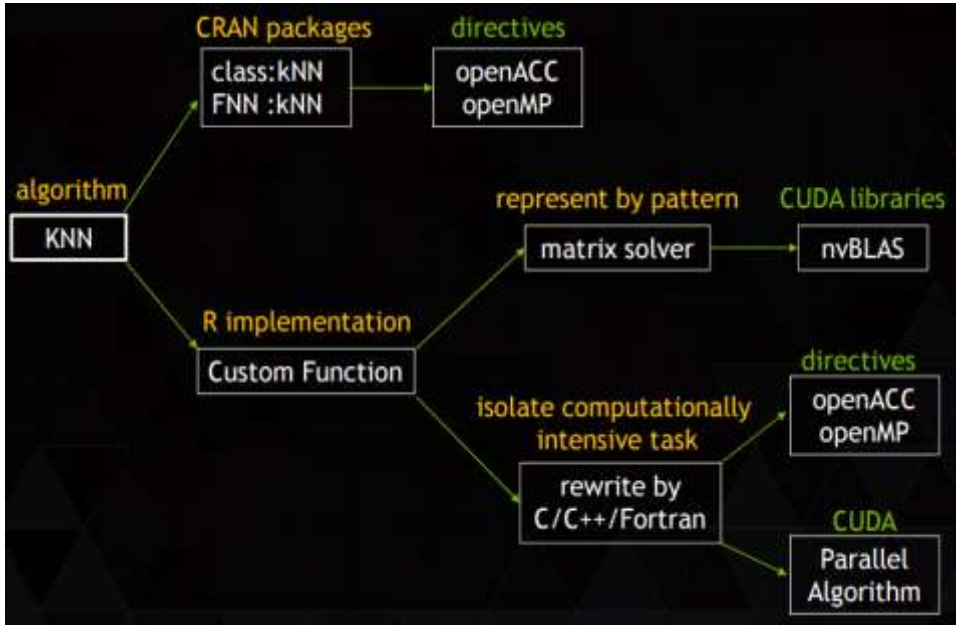
SOME R/GPU INTEGRATION EXAMPLES

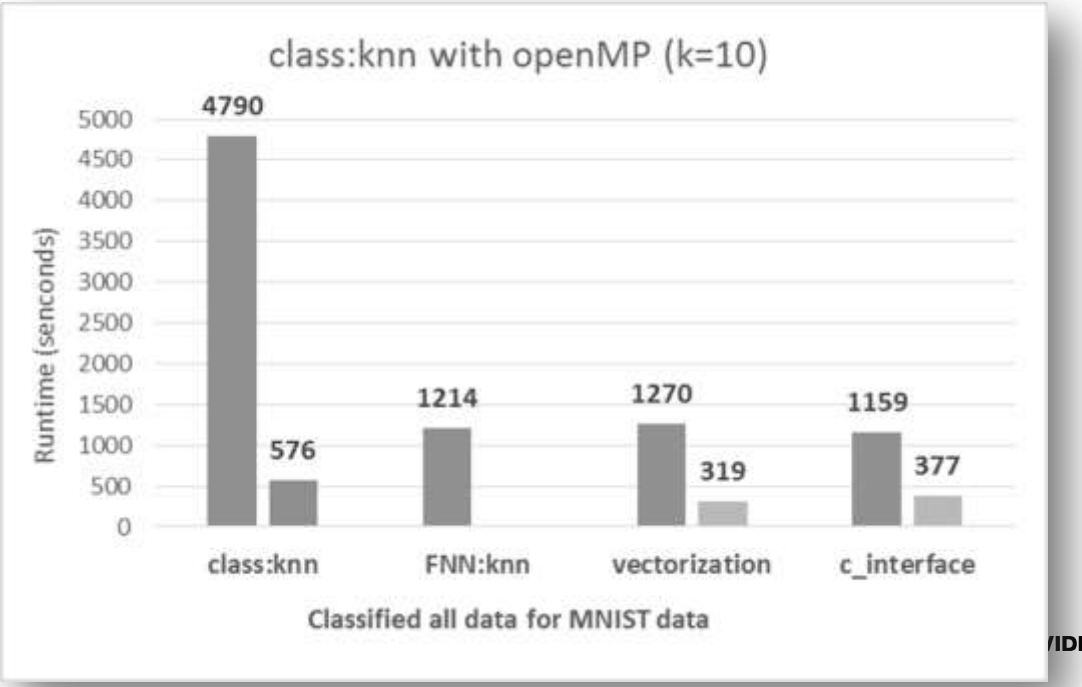
- **Parallel Forall: Accelerating R Applications with CUDA** (blog)

  Step by step guides to build CUDA with R in both Linux and Windows

- **GTC2015: Accelerating R Applications with CUDA** (slide)

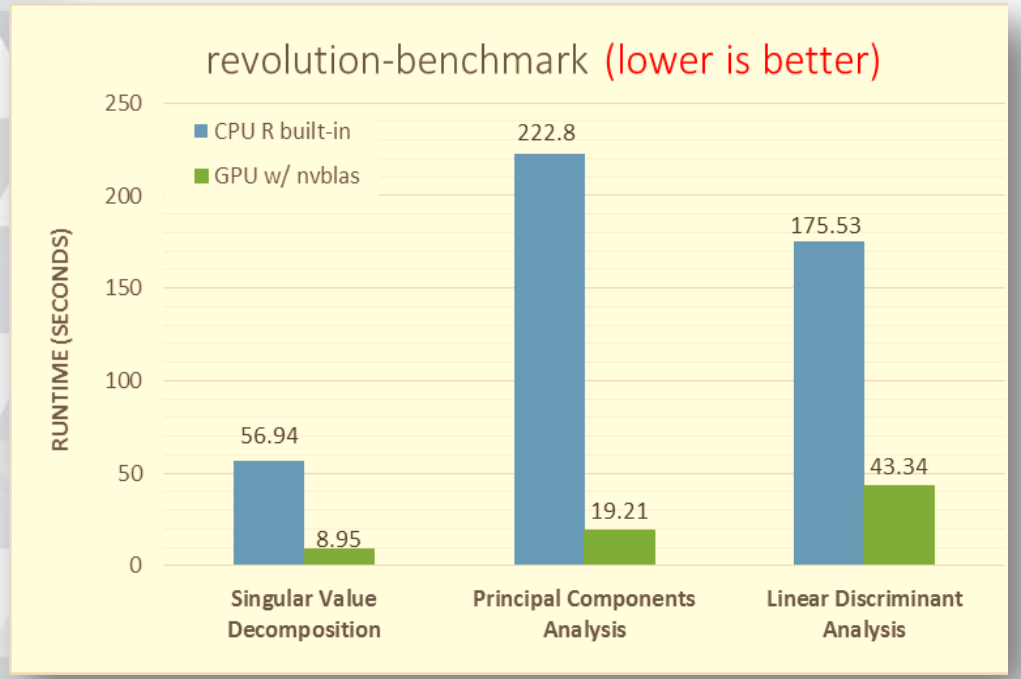
- show parallel strategies in R by CUDA with KNN for example

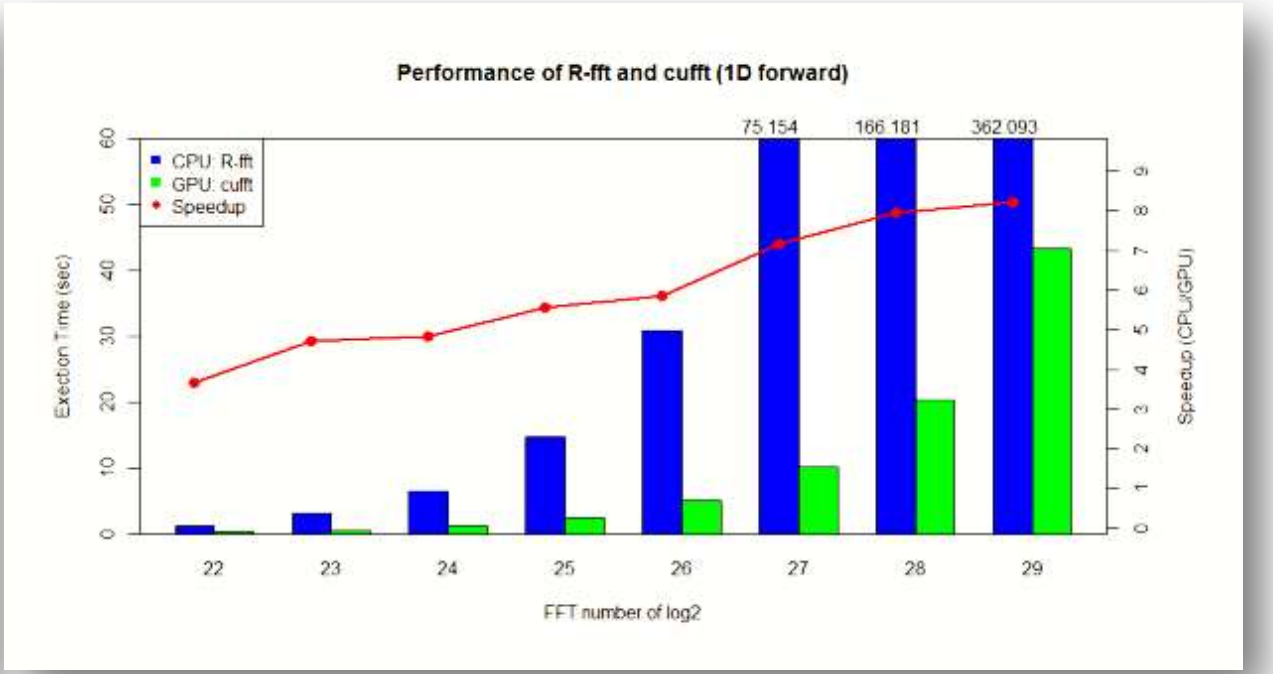




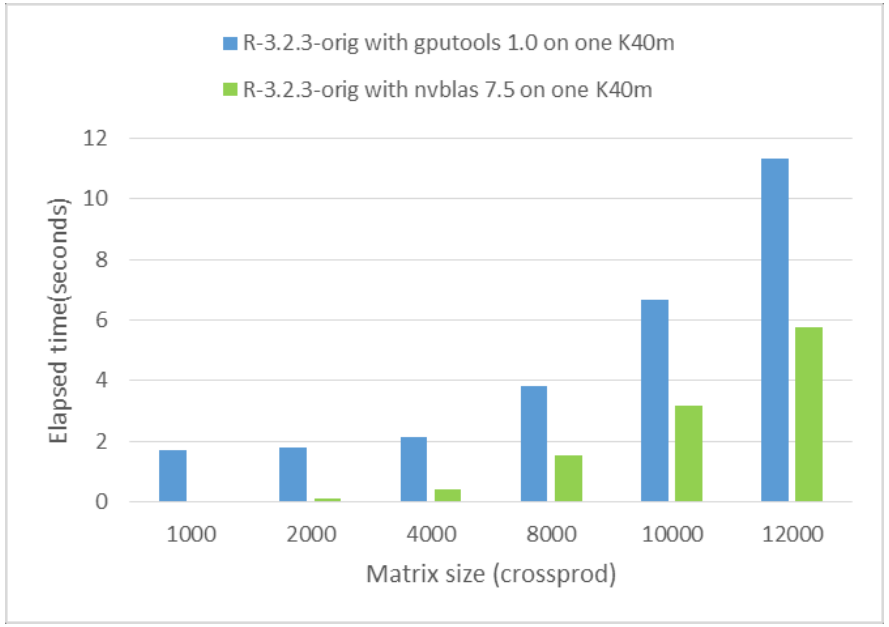
LEVERAGE GPU PERFORMANCE WITH R

- BLAS3 with NVBLAS
- cuFFT with R (blog)





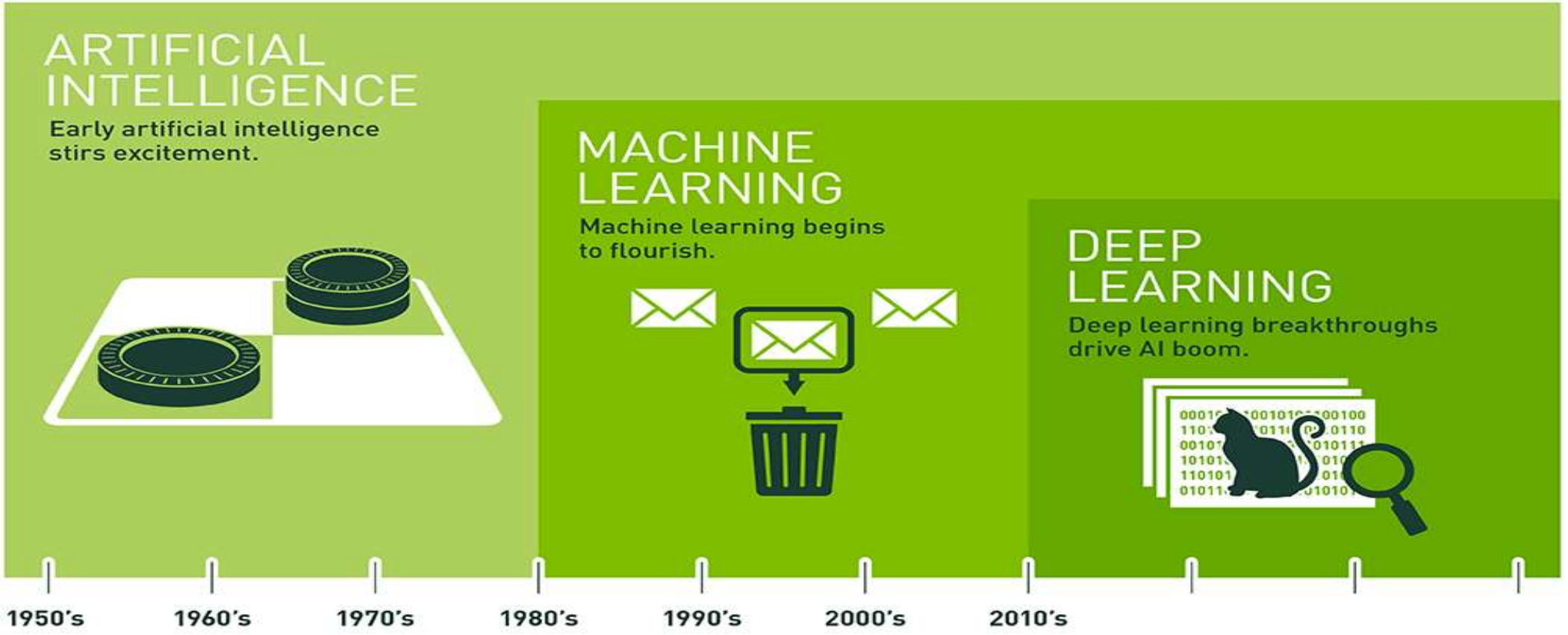
gputools .vs. nvblas




```
Total seconds: time spent in function and callees.
Self seconds: time spent in function alone.

   %       total       %        self
 total    seconds     self    seconds    name
 100.0      4.64       0.0      0.00     "eval"
 100.0      4.64       0.0      0.00     "gpuCrossprod"
 100.0      4.64       0.0      0.00     "source"
 100.0      4.64       0.0      0.00     "system.time"
 100.0      4.64       0.0      0.00     "withVisible"
  84.5      3.92      84.5      3.92     ".Call"
  15.5      0.72      15.5      0.72     "t.default"
  15.5      0.72       0.0      0.00     "t"


   %        self       %       total
 self     seconds    total    seconds    name
 84.5       3.92     84.5       3.92     ".Call"
 15.5       0.72     15.5       0.72     "t.default"


Total seconds: time spent in function and callees.
Self seconds: time spent in function alone.

   %       total       %        self
 total    seconds     self    seconds    name
 100.0      2.18     100.0      2.18     "<Anonymous>"
 100.0      2.18       0.0      0.00     "crossprod"
 100.0      2.18       0.0      0.00     "eval"
 100.0      2.18       0.0      0.00     "source"
 100.0      2.18       0.0      0.00     "standardGeneric"
 100.0      2.18       0.0      0.00     "system.time"
 100.0      2.18       0.0      0.00     "withVisible"


   %        self       %       total
 self     seconds    total    seconds    name
 100.0      2.18     100.0      2.18     "<Anonymous>"
```

BACKGROUND OF DEEP NEURAL NETWORKS

DEFINITIONS



A NEW COMPUTING MODEL

Algorithms that Learn from Examples

Expert Written Computer Program

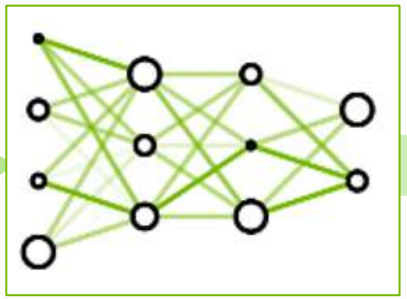
car

vehicle

coupe

Traditional Approach

- Requires domain experts
- Time consuming
- Error prone
- Not scalable to new problems



Deep Neural Network

car

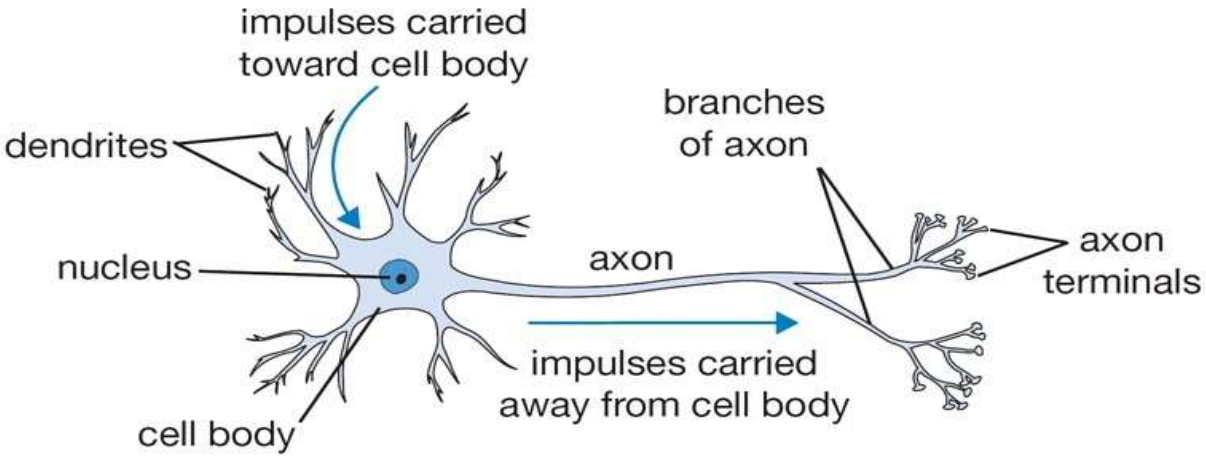
vehicle

coupe

Deep Learning Approach

- ✓ Learn from data
- ✓ Easily to extend
- ✓ Speedup with GPUs

How Does the Brain Work?



- The basic computational unit of the brain is a **neuron**
  ~86B neurons in the brain
- Neurons are connected with nearly **$10^{14} – 10^{15}$ synapses**
- Neurons receive input signal from **dendrites** and produce output signal along **axon**, which interact with the dendrites of other neurons via **synaptic weights**
- Synaptic weights – learnable & control influence strength

Image Source: Stanford

Neural Networks: Weighted Sum

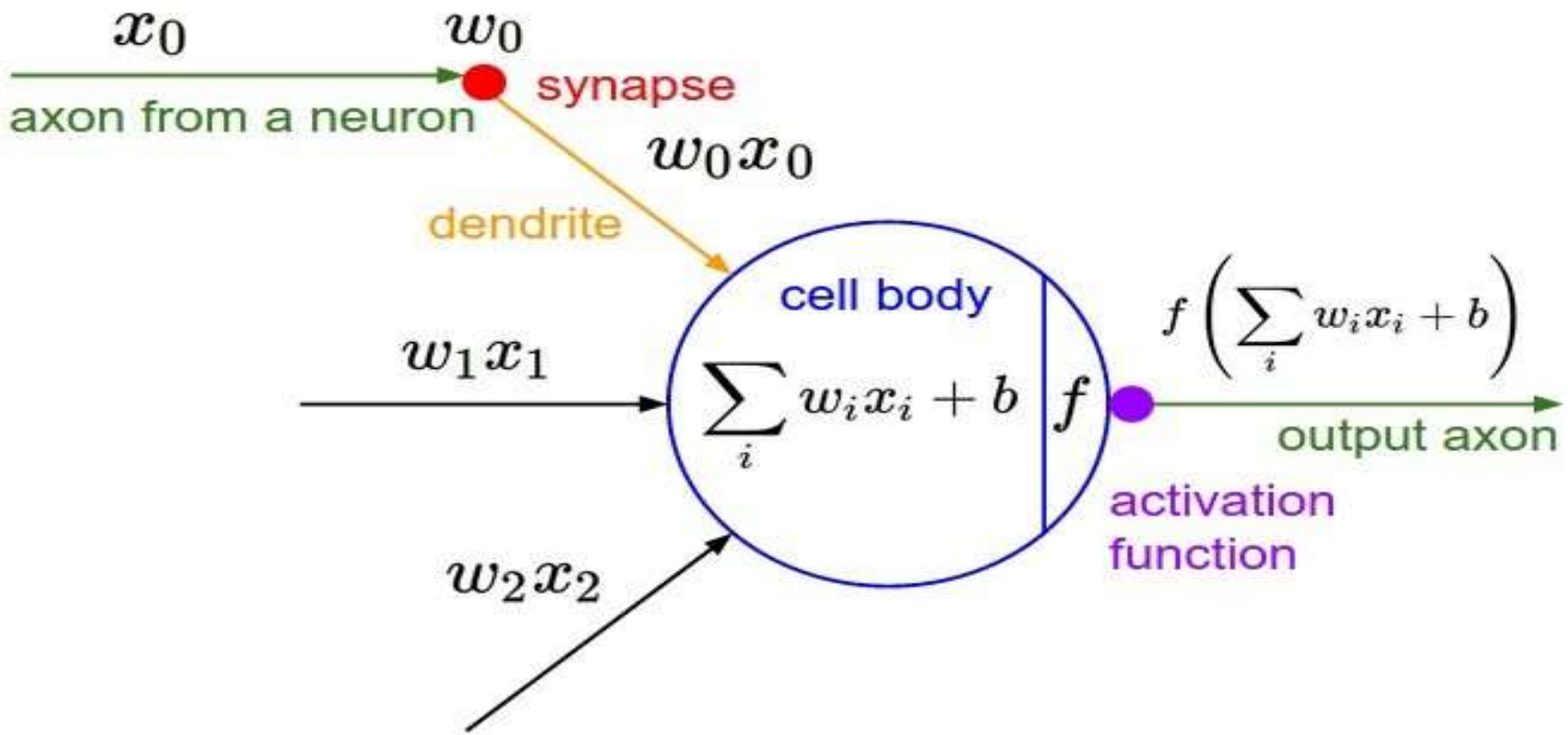


Image Source: Stanford

Many Weighted Sums

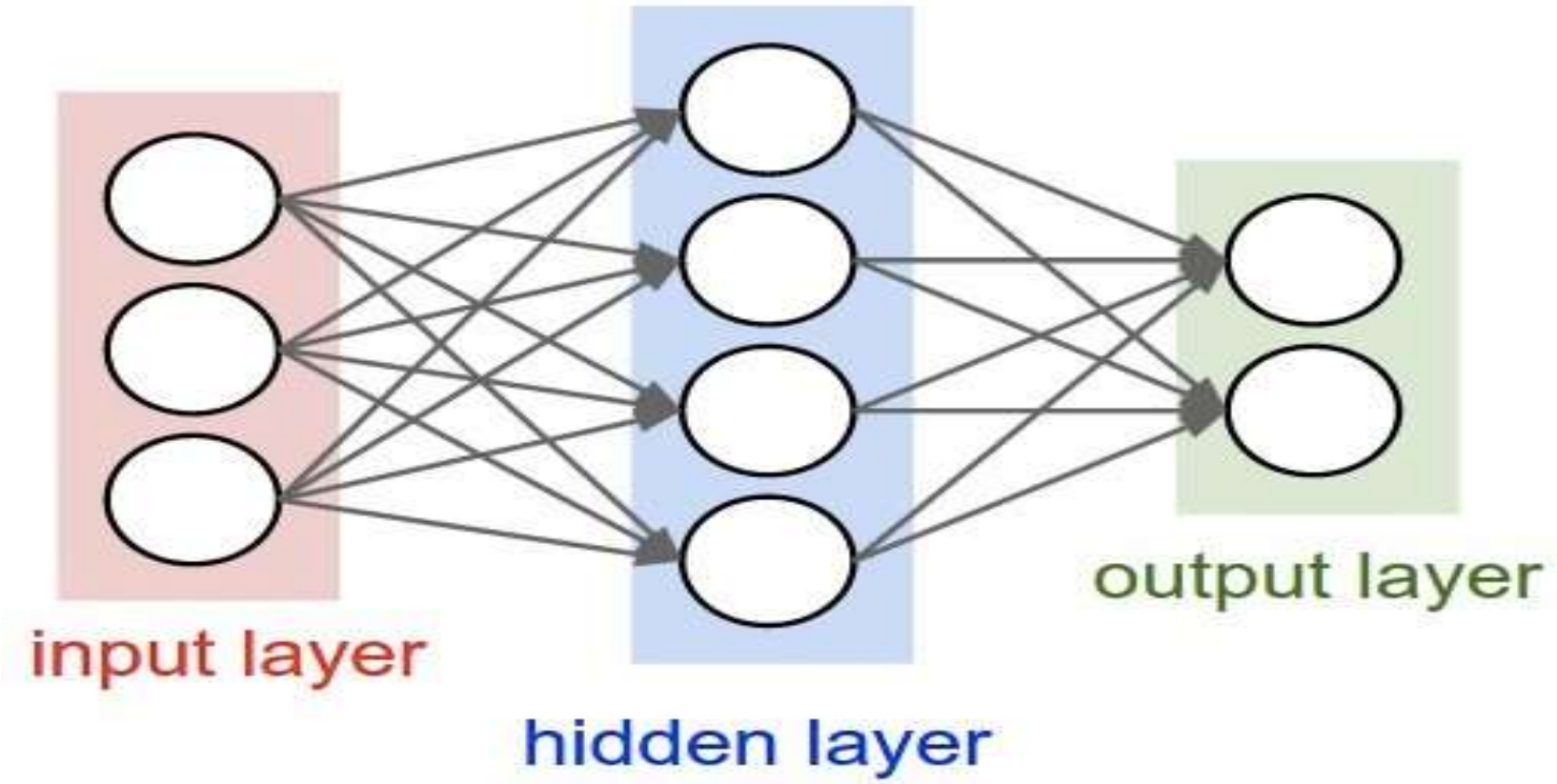


Image Source: Stanford

What is Deep Learning?

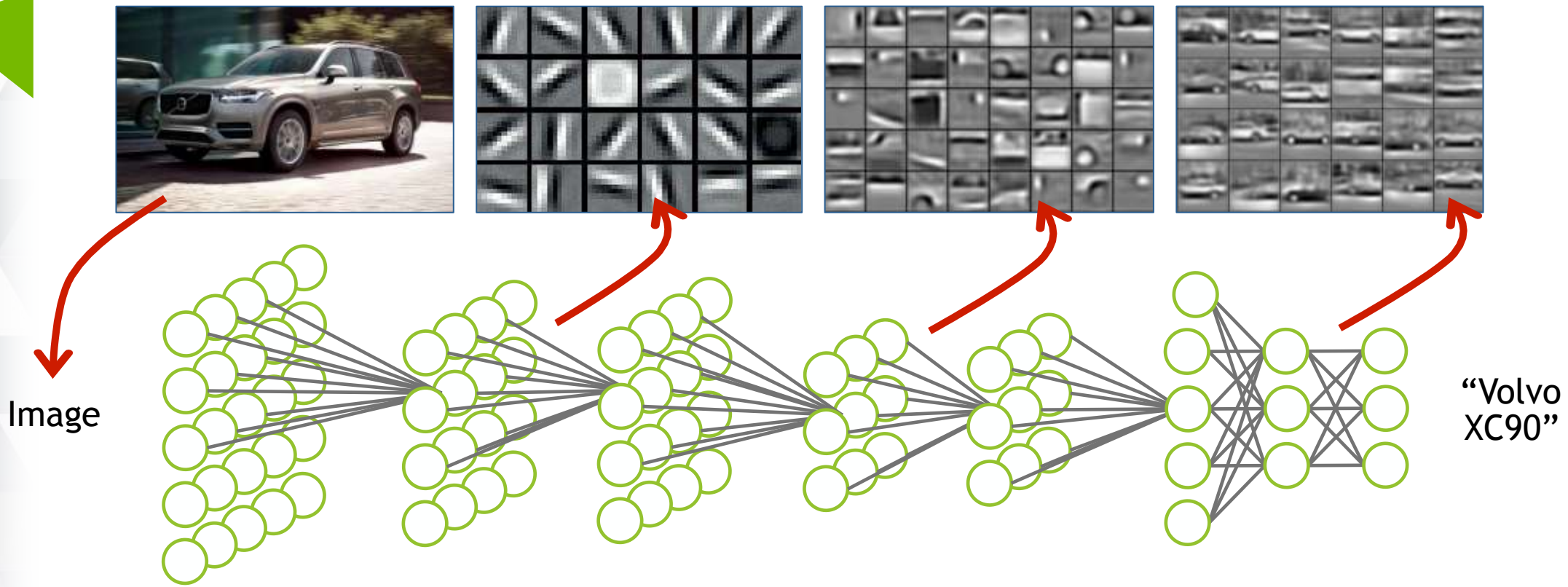


Image Source: [Lee et al., Comm. ACM 2011]

DEEP LEARNING

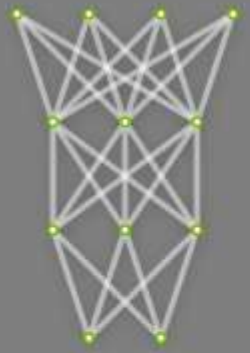
TRAINING

Learning a new capability
from existing data

INFERENCE

Applying this capability
to new data

Untrained
Neural Network
Model



Why is Deep Learning Hot Now?

Big Data Availability

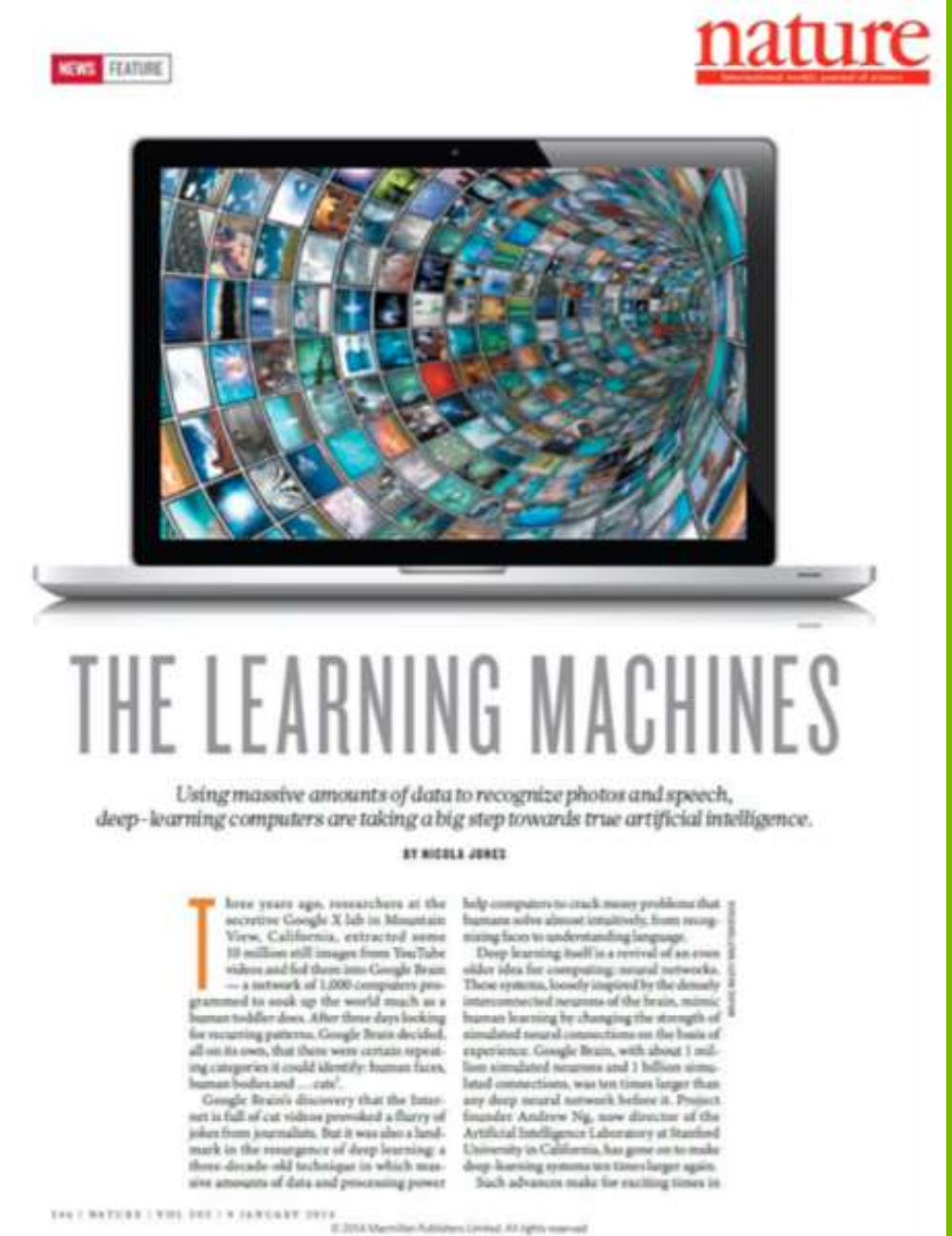
GPU Acceleration

New ML Techniques

facebook **350M** images uploaded per day

Walmart **2.5 Petabytes** of customer data hourly

YouTube **300 hours** of video uploaded every minute



NVIDIA TESLA GPUS REVOLUTIONIZE DEEP LEARNING

GOOGLE BRAIN APPLICATION – DEEP LEARNING

	BEFORE TESLA	AFTER TESLA
Cost	$5,000K	$200K
Servers	1,000 Servers	16 Tesla Servers
Energy	600 KW	4 KW
Performance	1x	6x

ImageNet Challenge

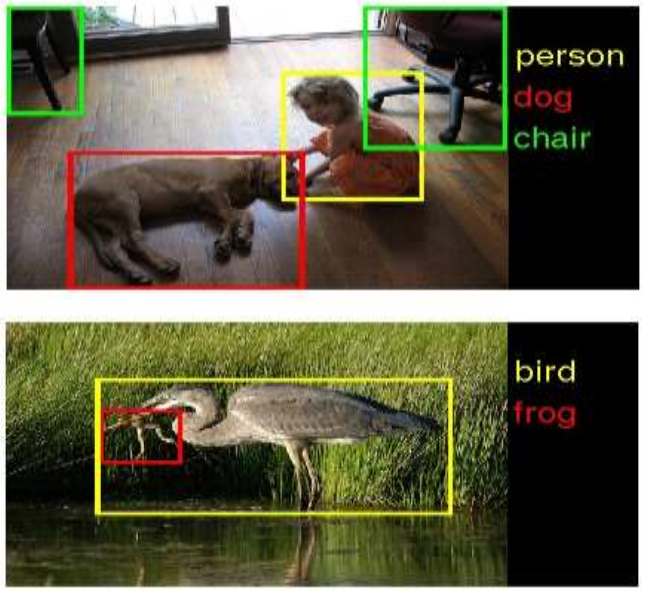
IMAGENET

Image Classification Task:

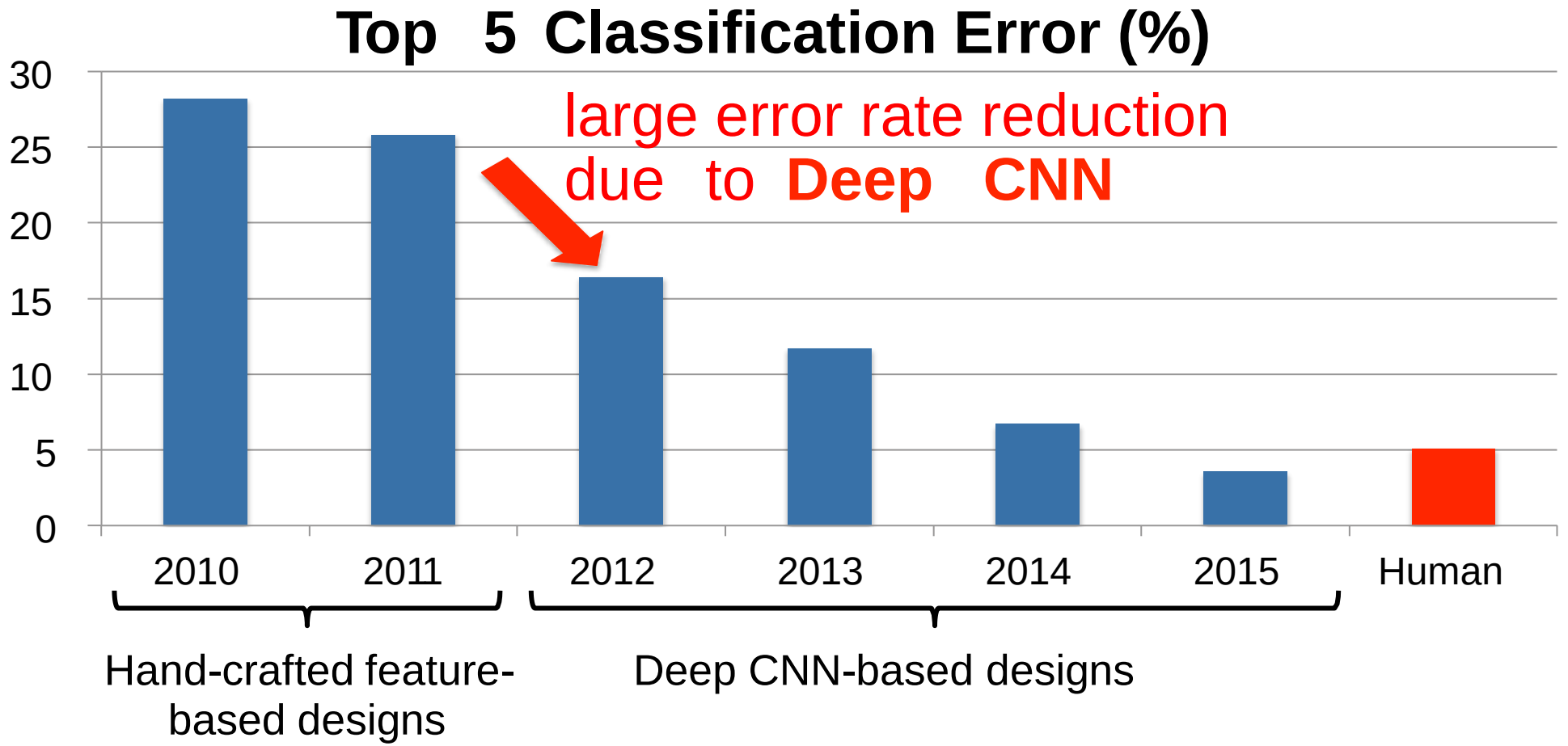
1.2M training images • 1000 object categories

Object Detection Task:

456k training images • *200 object categories*



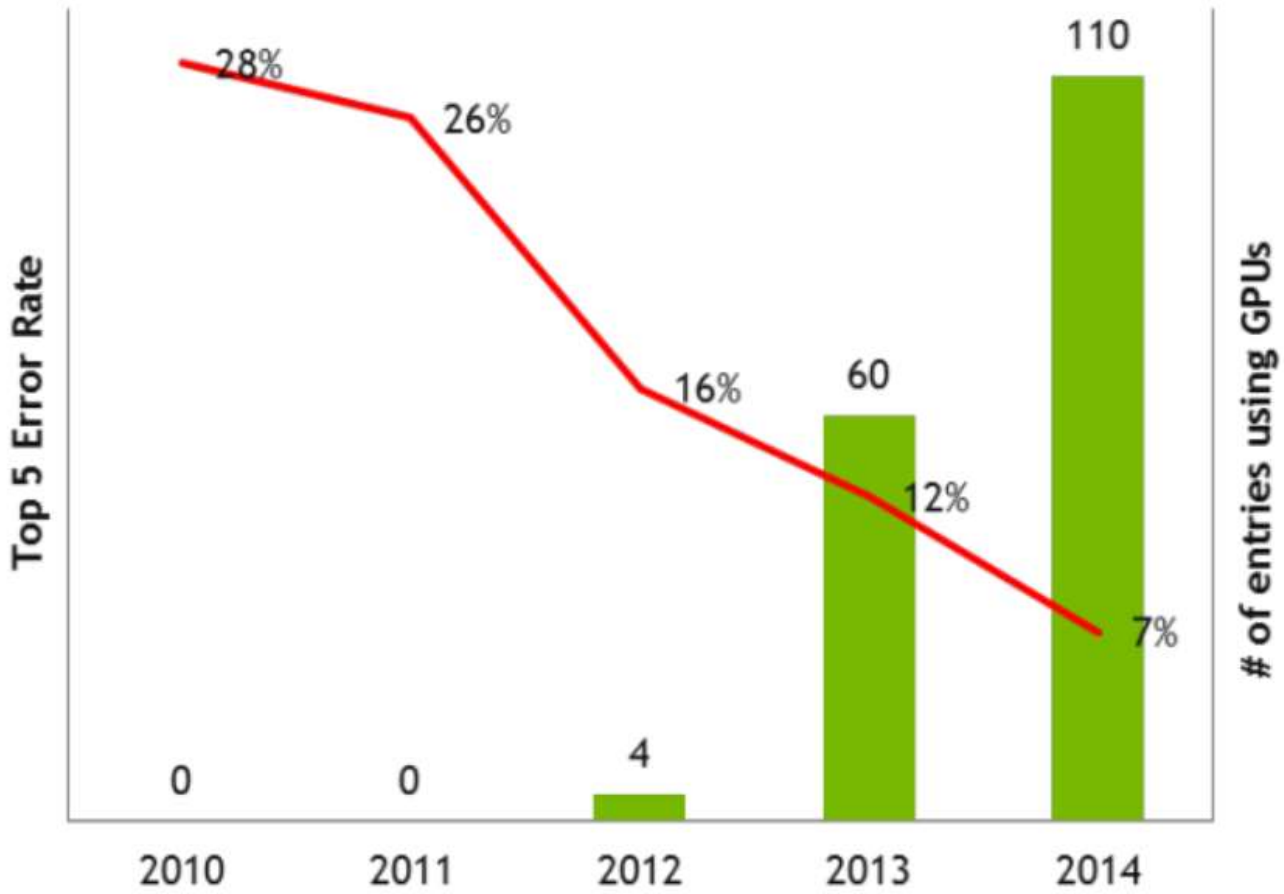
ImageNet: Image Classification Task



[Russakovsky et al., IJCV 2015]

NVIDIA

GPU Usage for ImageNet Challenge



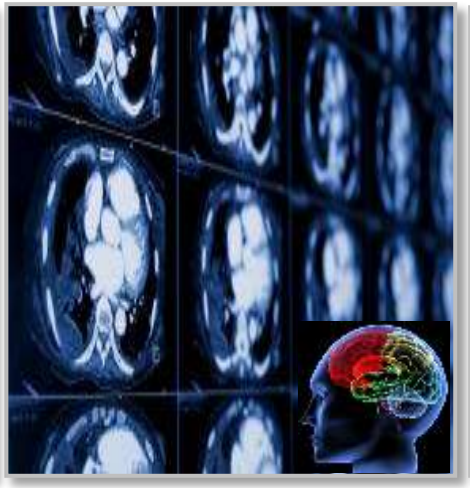
NVIDIA

DEEP LEARNING IS SWEEPING ACROSS INDUSTRIES

Internet Services

- Image/Video classification
- Speech recognition
- Natural language processing

Medicine



- Cancer cell detection
- Diabetic grading
- Drug discovery

Media & Entertainment

- Video captioning
- Content based search
- Real time translation

Security & Defense

- Face recognition
- Video surveillance
- Cyber security

Autonomous Machines

- Pedestrian detection
- Lane tracking
- Recognize traffic sign

clarifai

Amplifying Intelligence

Video analytics demo @ www.clarifai.com/#demo

PayPal

AI Powered Fraud Detection System

CHALLENGE

Build an accurate fraud detection system that can detect changing fraudulent behavior patterns

SOLUTION

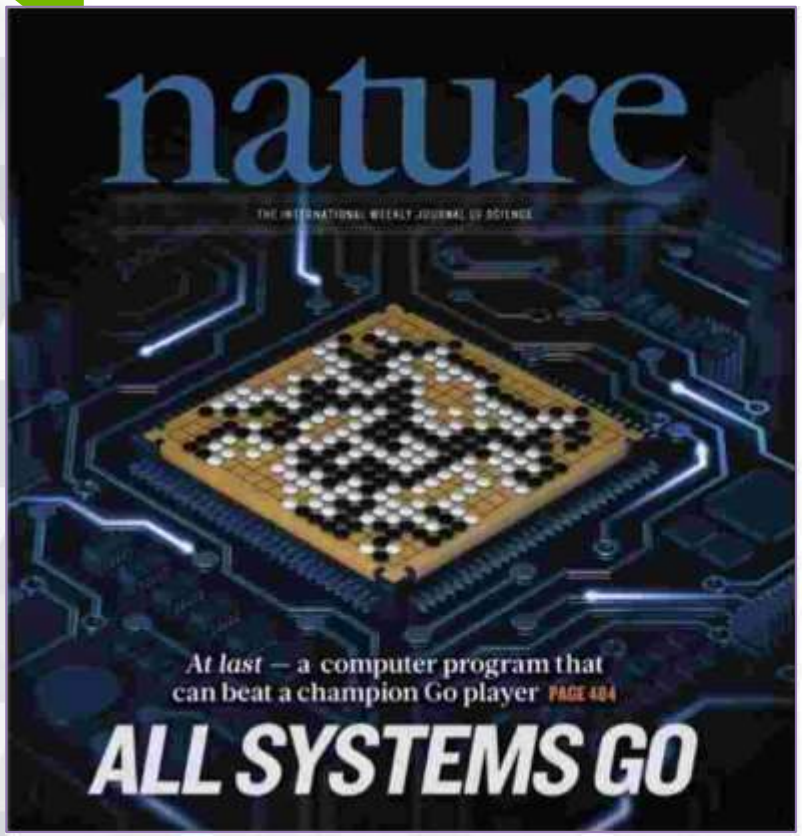
Use deep learning to re-implement legacy rule based fraud detection system to tremendously improve fraud detection accuracy.

"10,000s of features make up todays fraudulent behavior. AI can detect patterns faster and more accurate than humans"

-Hui Wang, Senior Director of Global Risk Sciences, Pay Pal

Deep Learning on Games

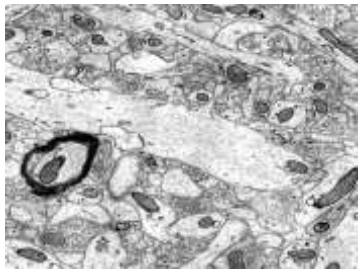
Google DeepMind AlphaGo



Connectomics – Finding Synapses

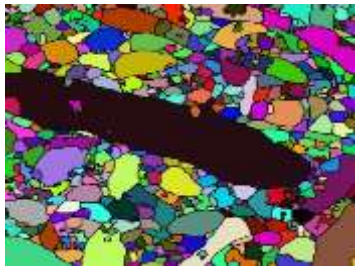
Machine Learning requires orders of Magnitude more computation than other parts

(1) EM

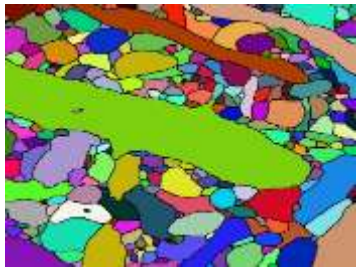


(2) ML Membrane Detection

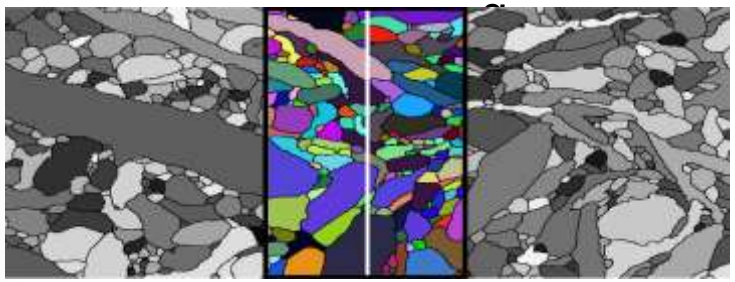
(3) Watershed



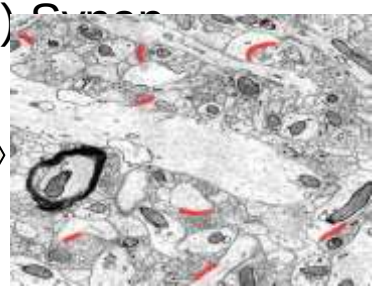
(4) Agglomeration



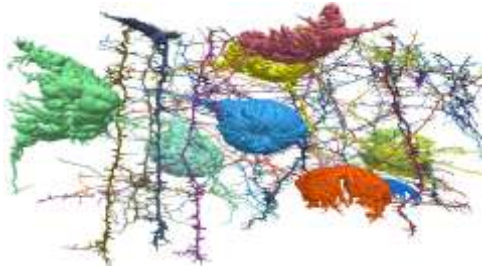
(5) Mergin ses



(6) Synap



(7) Skeletons



(8) Graph

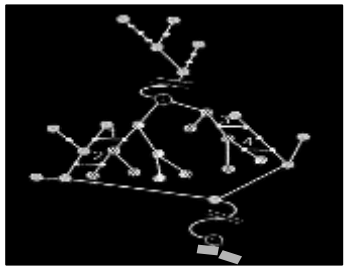


Image Source: MIT

DEEP-LEARNING RESEARCH ROVER

TURBO 2.0

github.org/dusty-nv

NVIDIA

GitHub

Mature Applications

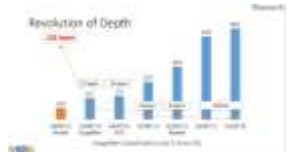
- **Image**
  - Classification: image to object class
  - Recognition: same as classification (except for faces)
  - Detection: assigning bounding boxes to objects
  - Segmentation: assigning object class to every pixel
- **Speech & Language**
  - Speech Recognition: audio to text
  - Translation
  - Natural Language Processing: text to meaning
  - Audio Generation: text to audio
- **Games**

NVIDIA

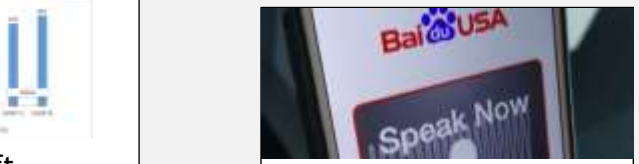
MILESTONES IN AI

IMAGENET

Microsoft & Google
“Superhuman” Image
Recognition



Microsoft
“Super Deep Network”



Deep Speech 2
One network, 2 languages

Berkeley’s Brett
End-to-End
Reinforcement Learning

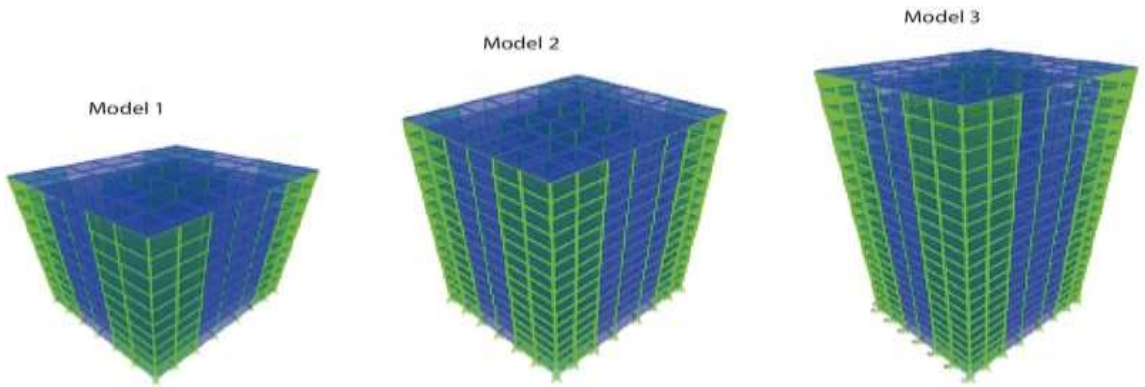
A New Computing Model
Hits Pop Culture

AlphaGo
Rivals a World Champion

TU Delft Deep-Learning
Amazon Picking Champion

Emerging Applications

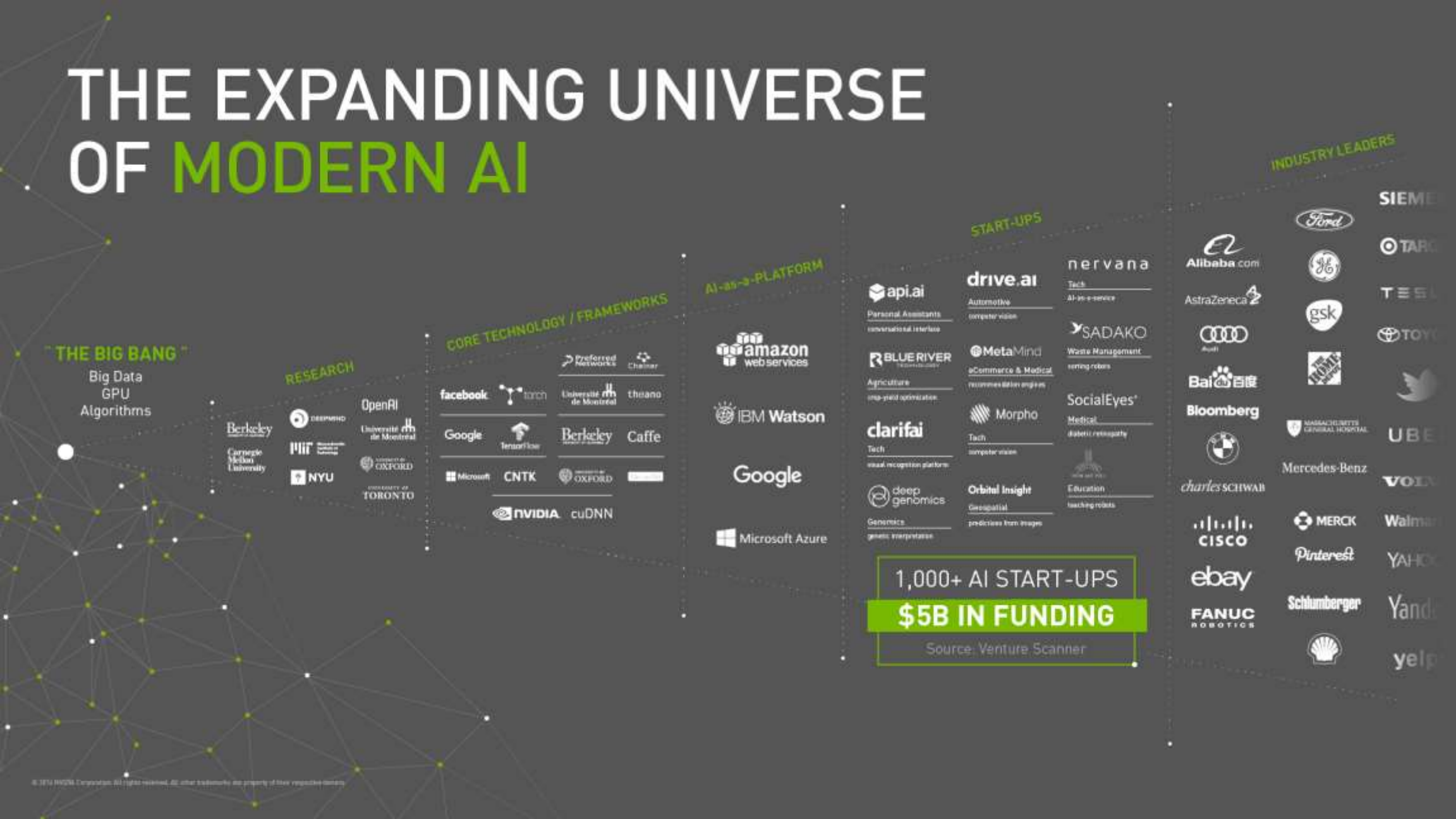
- **Medical** (Cancer Detection, Pre-Natal)
- **Finance** (Trading, Energy Forecasting, Risk)
- **Infrastructure** (Structure Safety and Traffic)
- Weather Forecasting and Event Detection



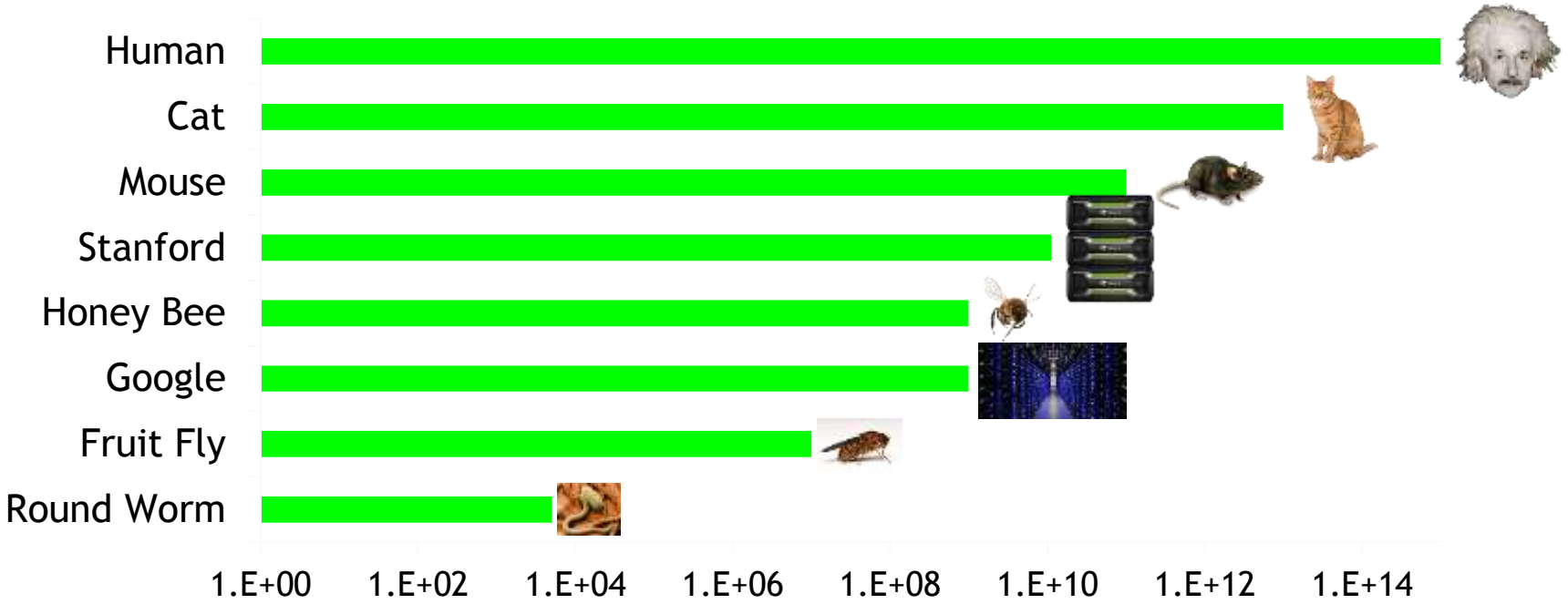
This tutorial will focus on image classification

http://www.nextplatform.com/2016/09/14/next-wave-deep-learning-applications/

THE EXPANDING UNIVERSE OF MODERN AI



NUMBER OF SYNAPSES



Larger Training Sets

More Training Iterations

Concept courtesy of Ian Goodfellow, University of Montreal

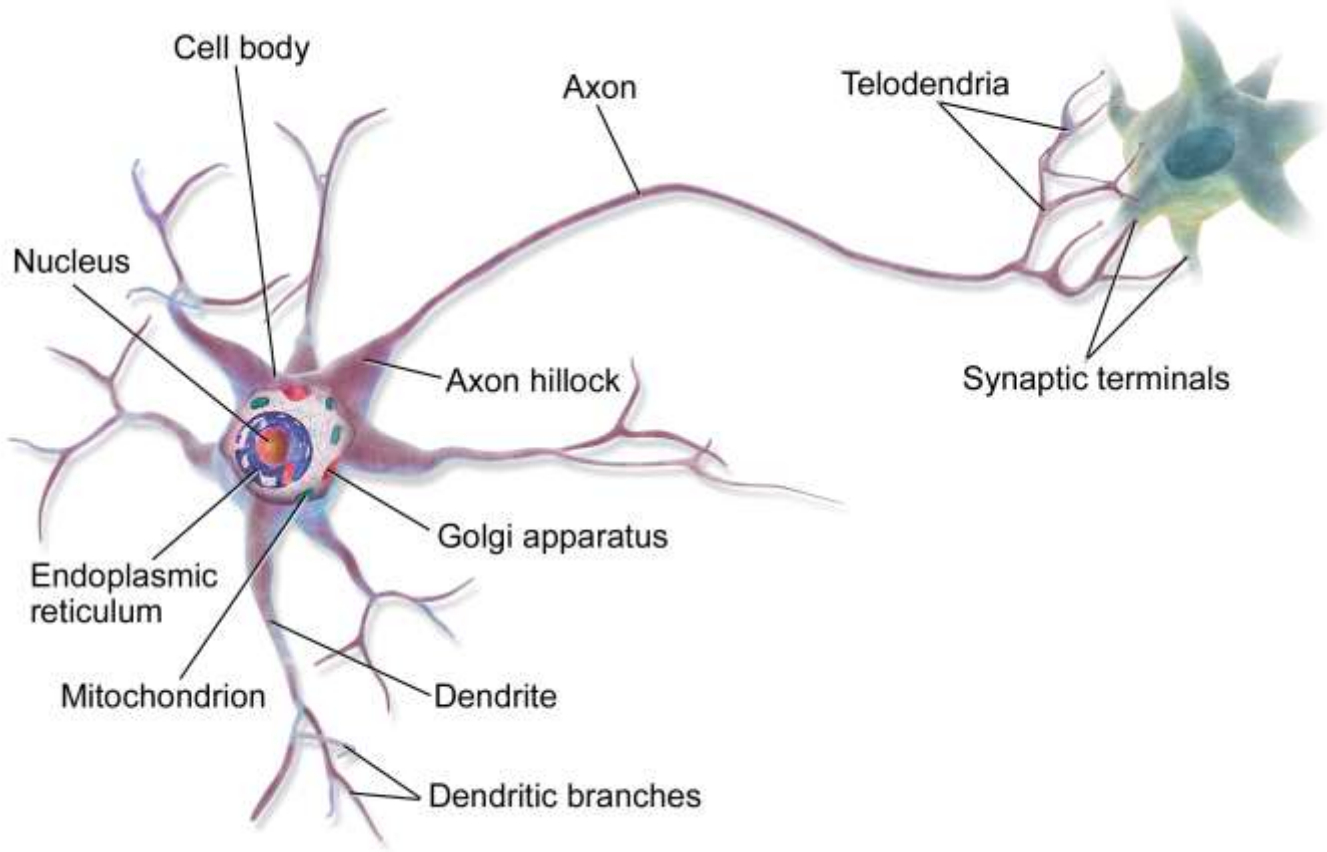
NVIDIA

CHALLENGES

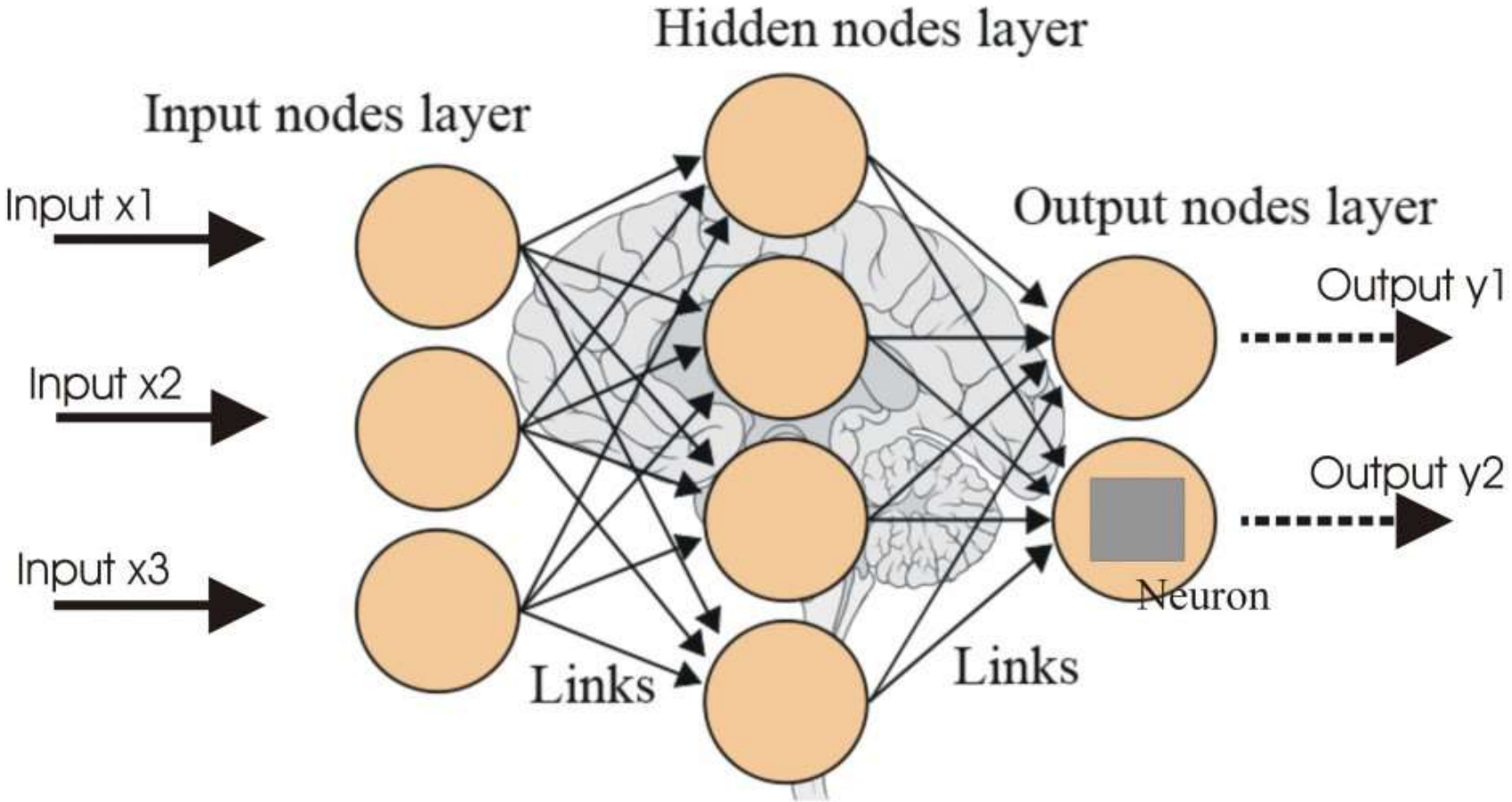
Deep Learning Needs	Why
Data Scientists	Demand far exceeds supply
Latest Algorithms	Rapidly evolving
Fast Training	Impossible -> Practical
Deployment Platform	Must be available everywhere

OVERVIEW OF DEEP NEURAL NETWORKS

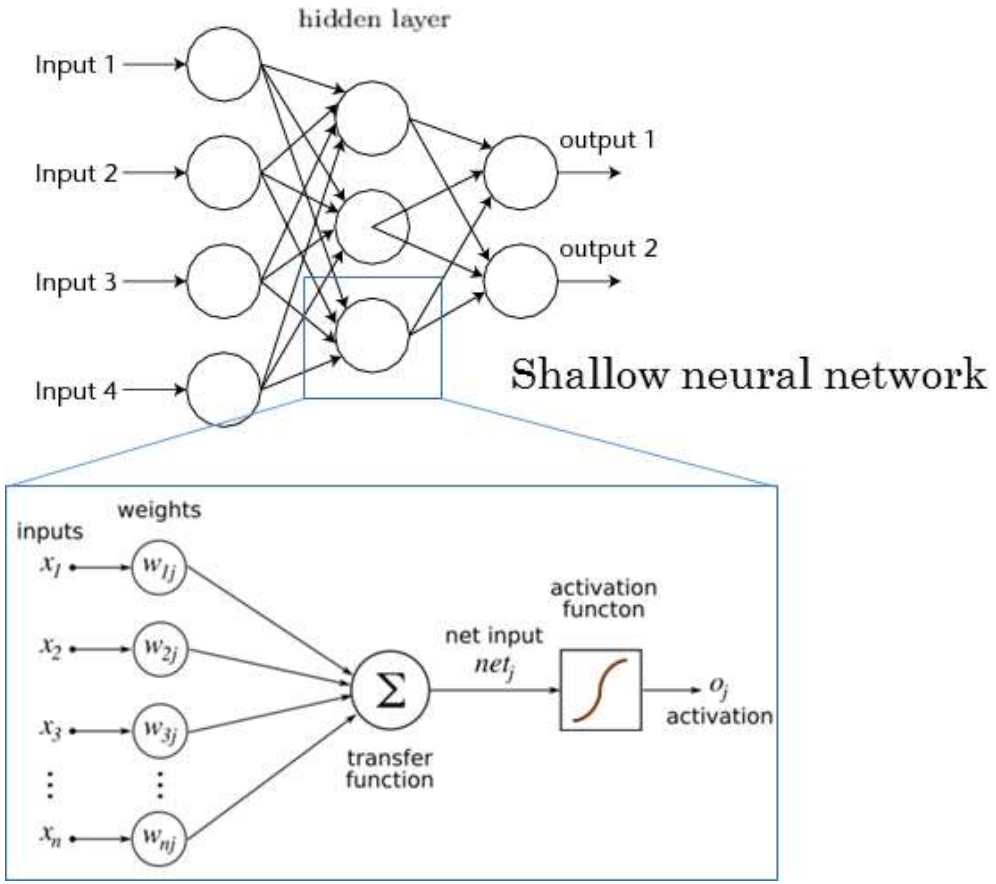
BIOLOGICAL NEURAL NETWORK

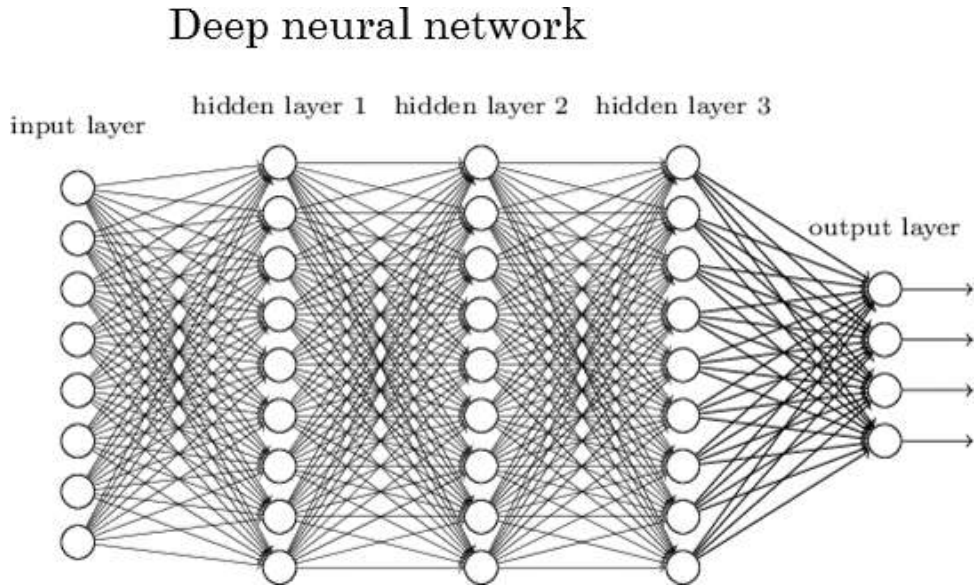


COMPUTER REPRESENTATION

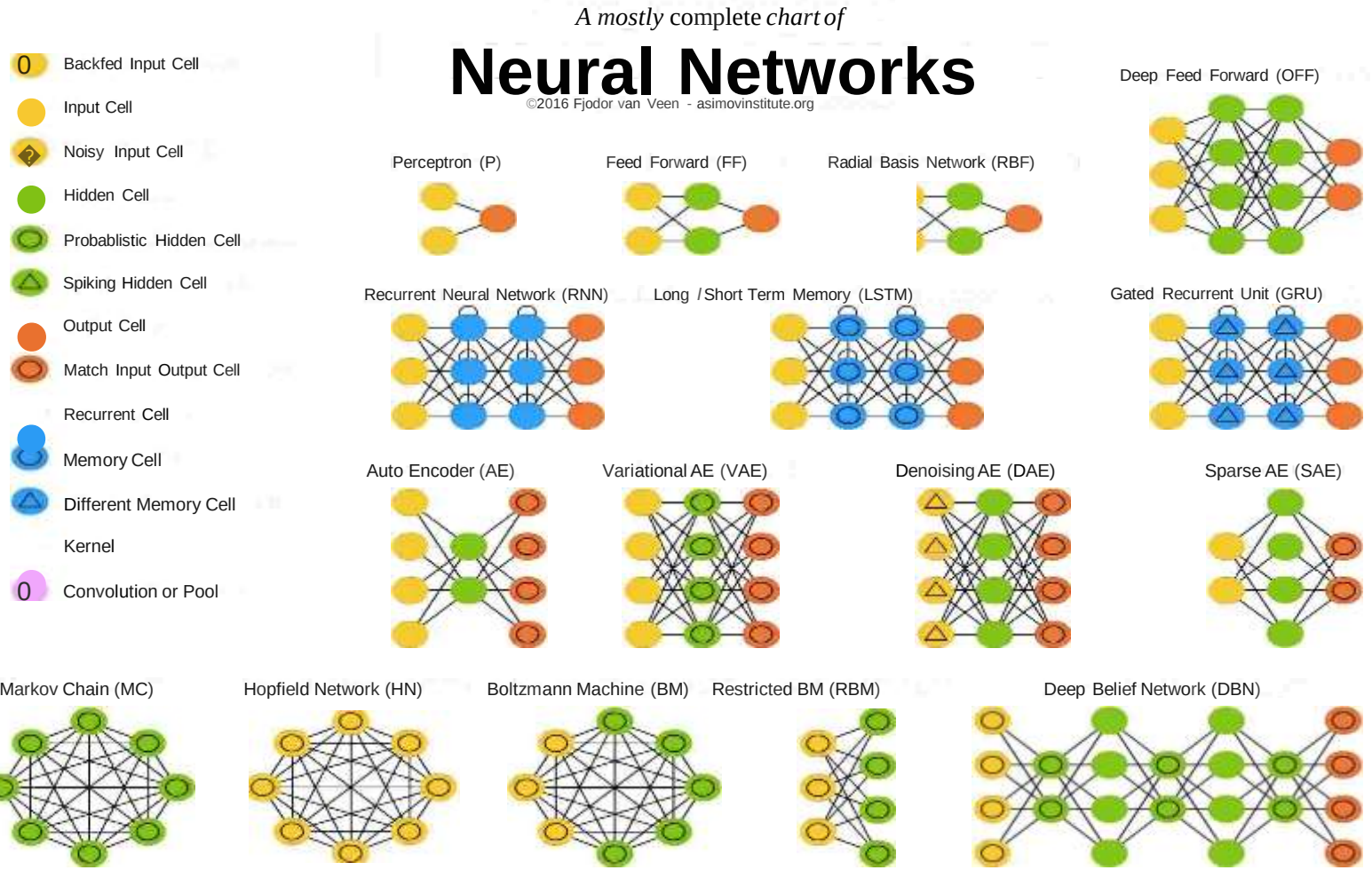


SHALLOW VS DEEP NEURAL NETWORK





So Many Neural Networks!



http://www.asimovinstitute.org/neural-network-zoo/

DNN Terminology 101

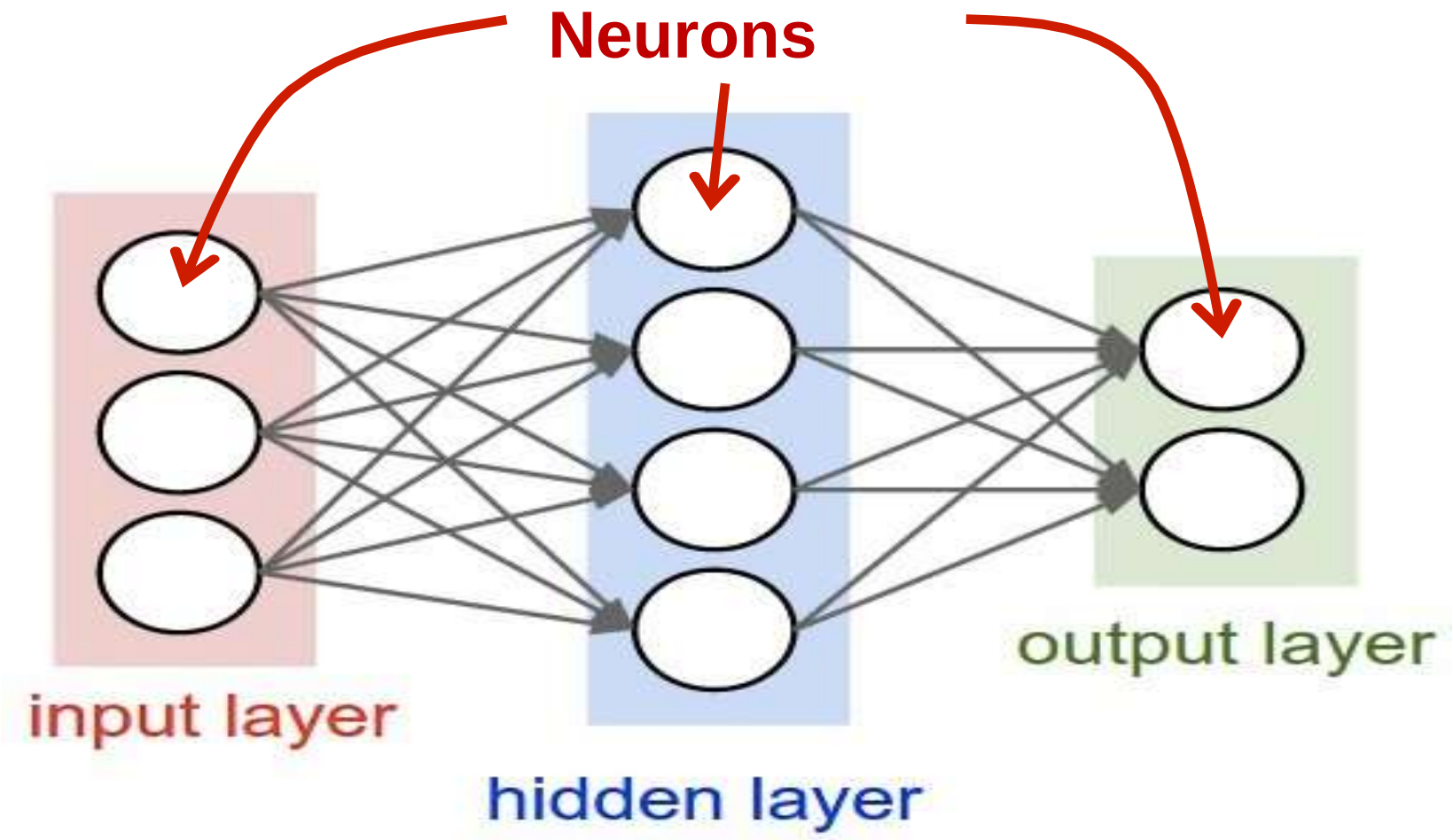


Image Source: Stanford

DNN Terminology 101

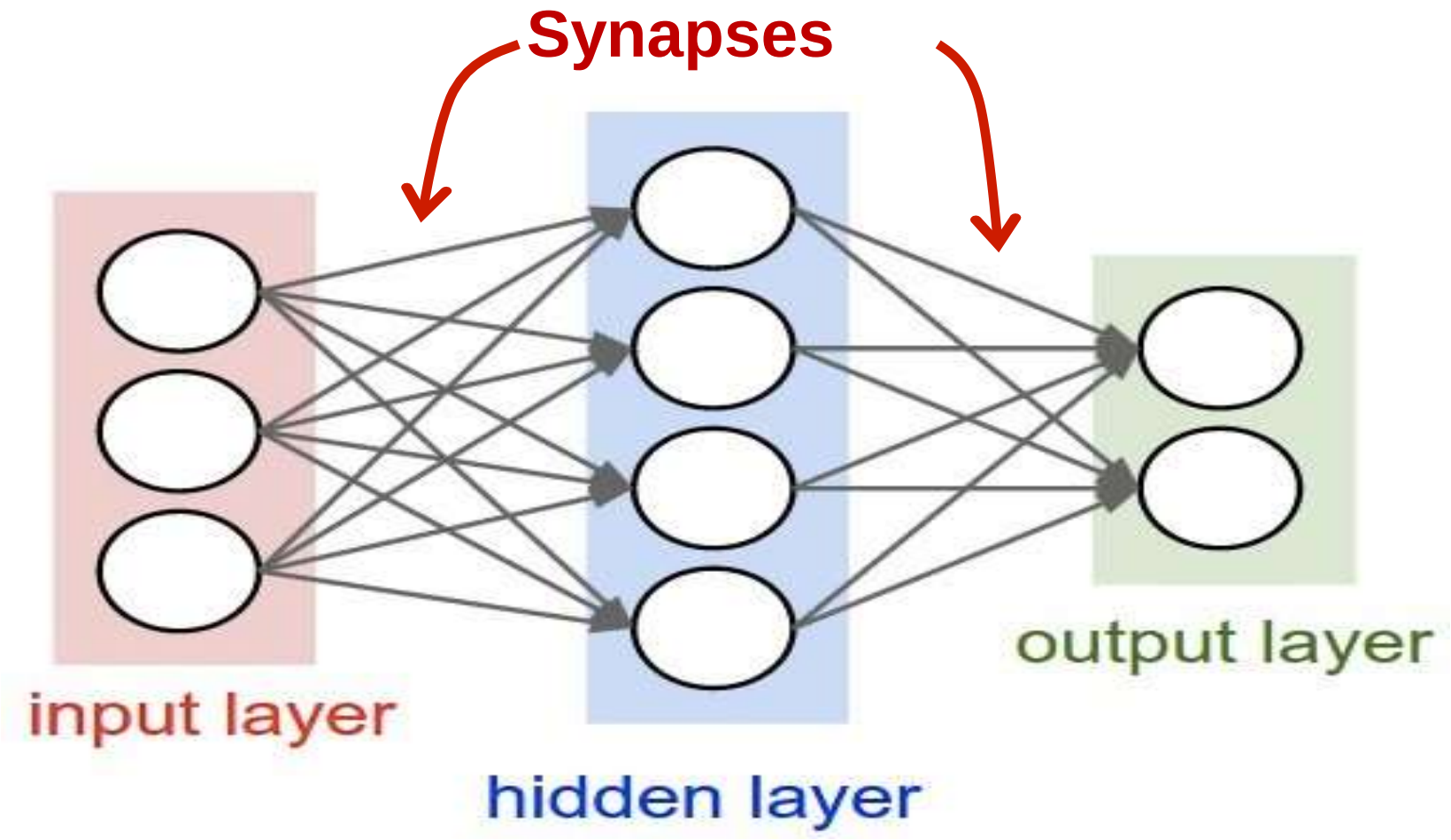


Image Source: Stanford

DNN Terminology 101

Each synapse has a **weight** for neuron **activation**

$$Y_j = activation\left(\sum_{i=1}^{3} W_{ij} * X_i\right)$$

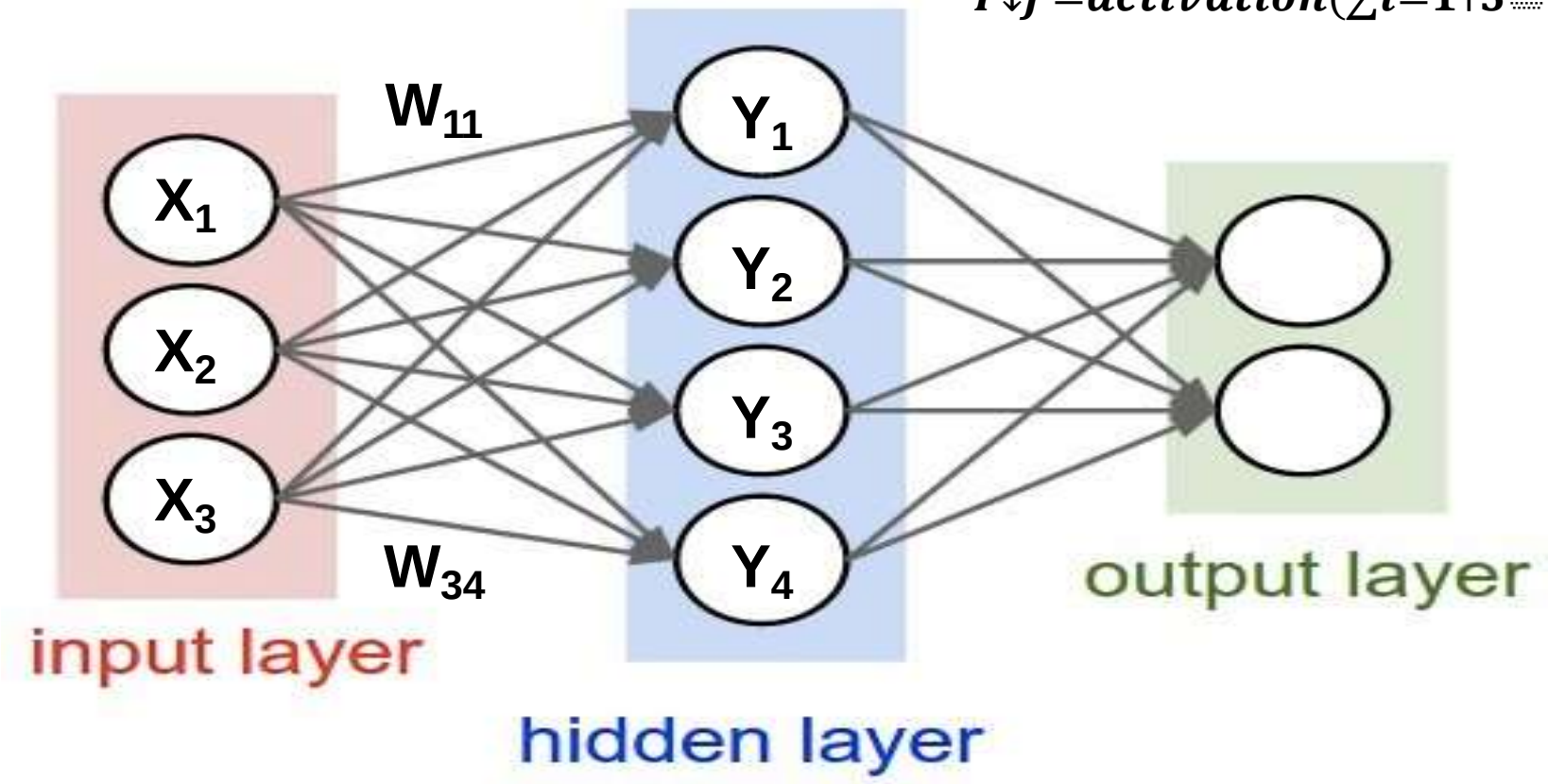


Image Source: Stanford

DNN Terminology 101

Weight Sharing: multiple synapses use the **same weight value**

$$Y\downarrow j = activation(\sum i=1\uparrow 3 \blacksquare W\downarrow ij * X\downarrow$$

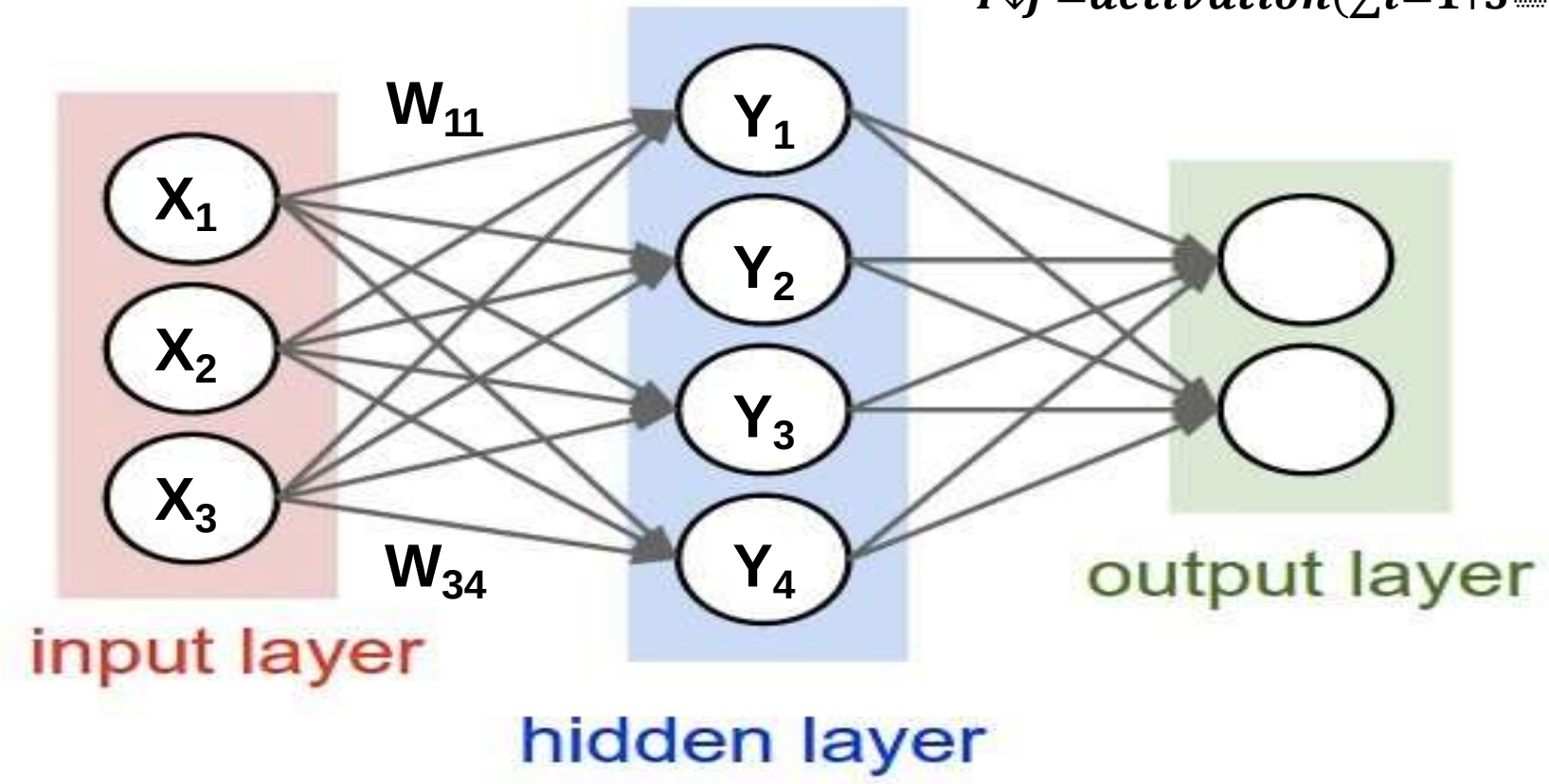


Image Source: Stanford

DNN Terminology 101

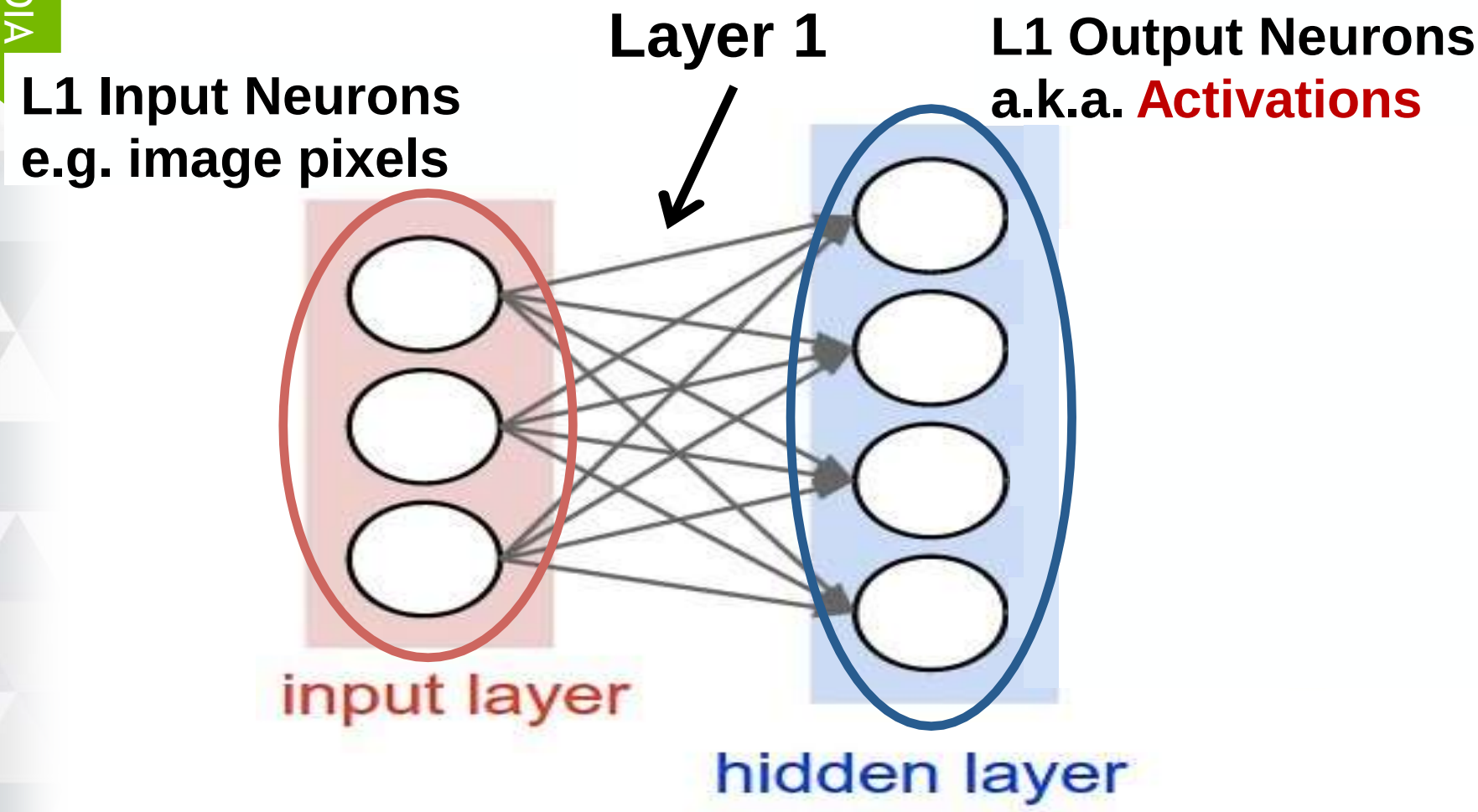


Image Source: Stanford

DNN Terminology 101

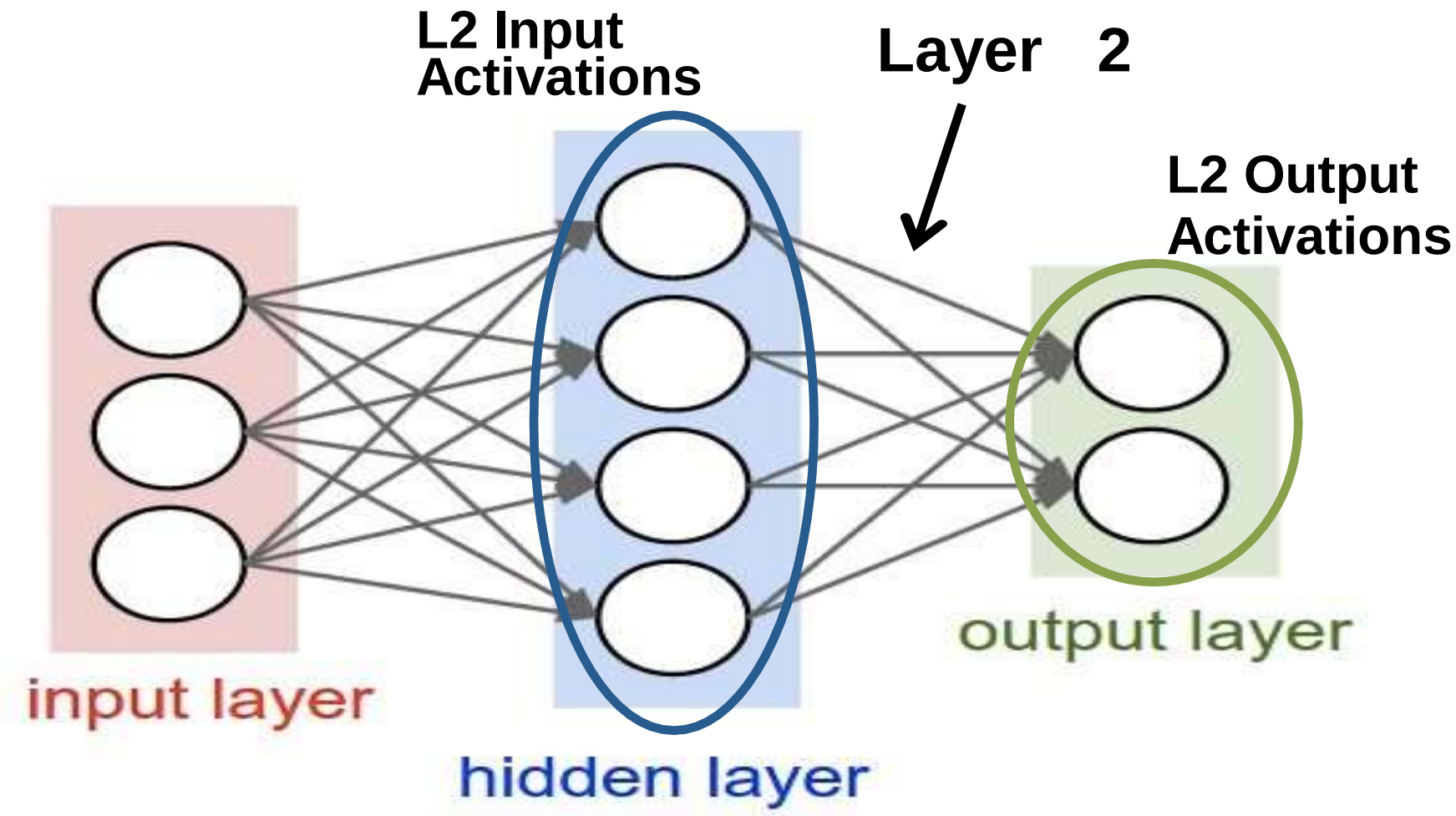


Image Source: Stanford

DNN Terminology 101

Fully-Connected: all i/p neurons connected to all o/p neurons

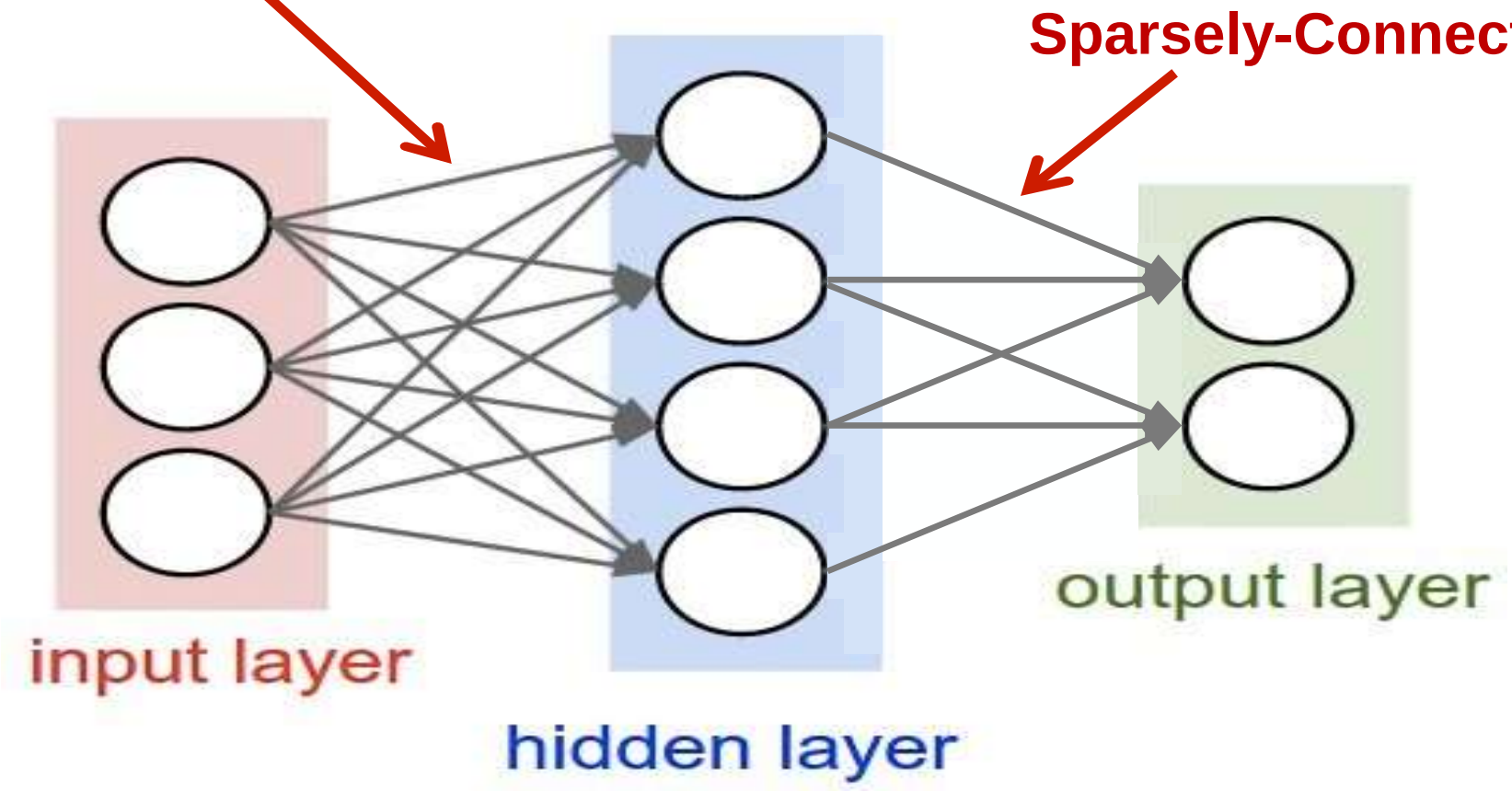


Image Source: Stanford

DNN Terminology 101

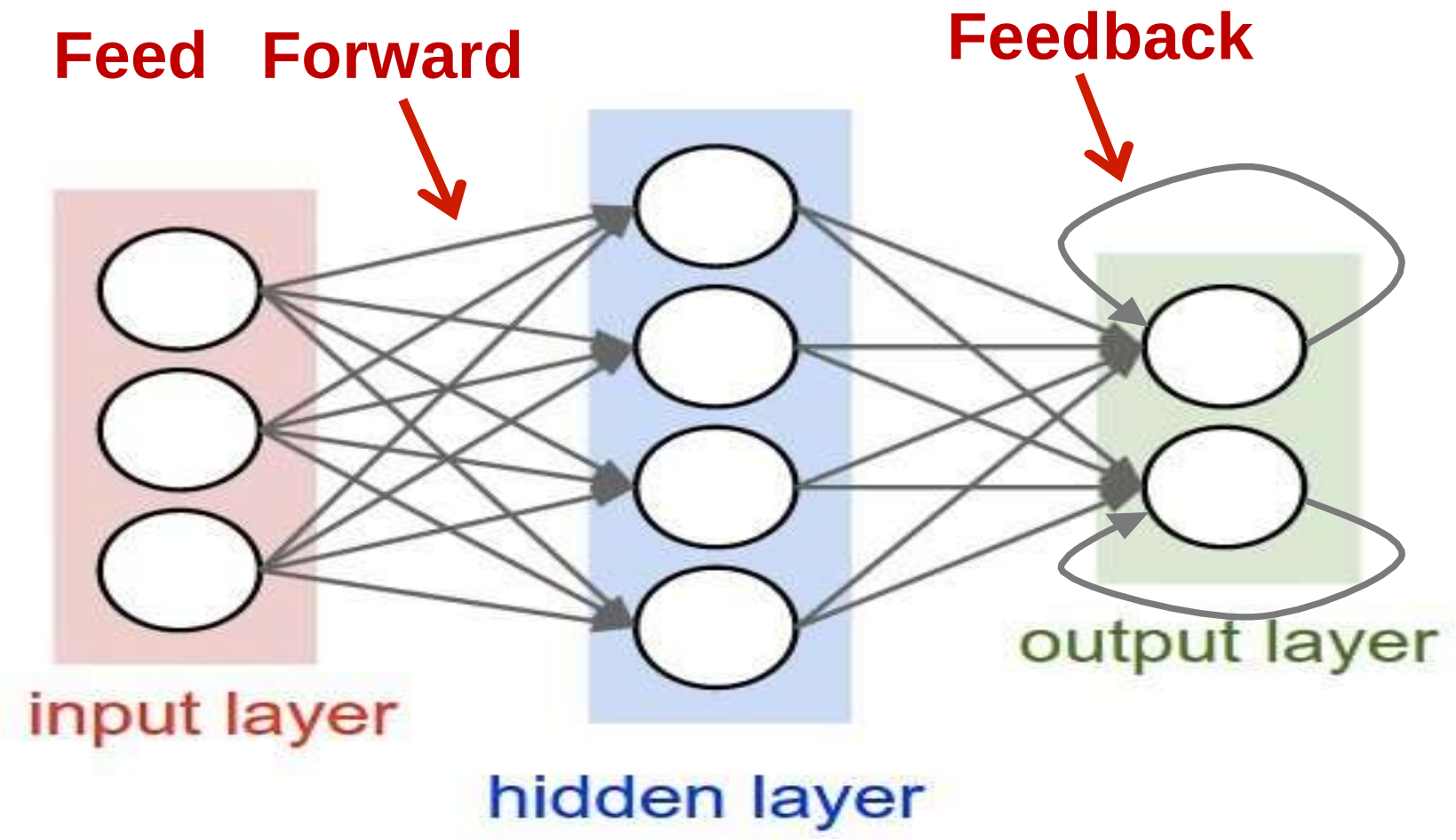


Image Source: Stanford

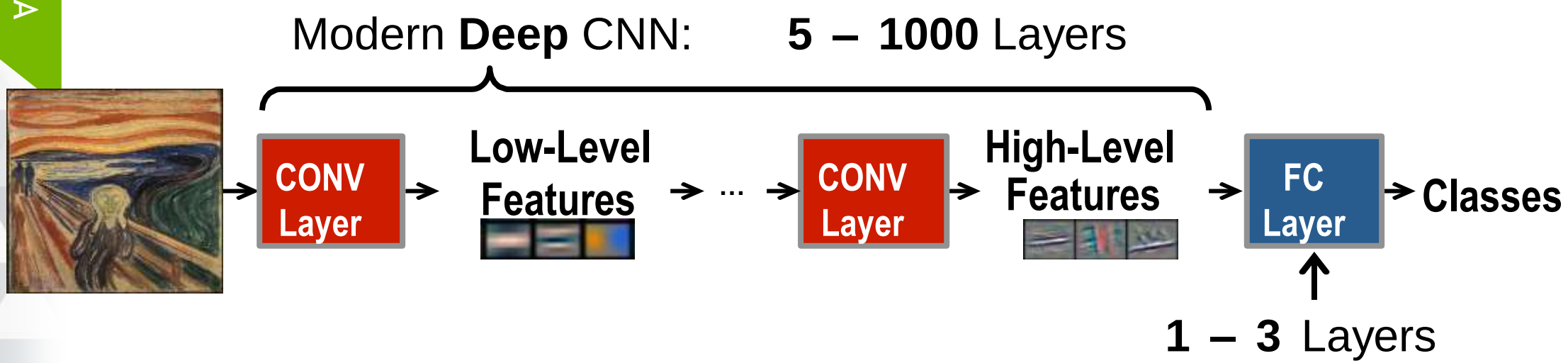
Popular Types of DNNs

- **Fully-Connected NN**
  - feed forward, a.k.a. multilayer perceptron (MLP)
- **Convolutional NN (CNN)**
  - feed forward, sparsely-connected w/ weight sharing
- **Recurrent NN (RNN)**
  - feedback
- **Long Short-Term Memory (LSTM)**
  - feedback + Storage

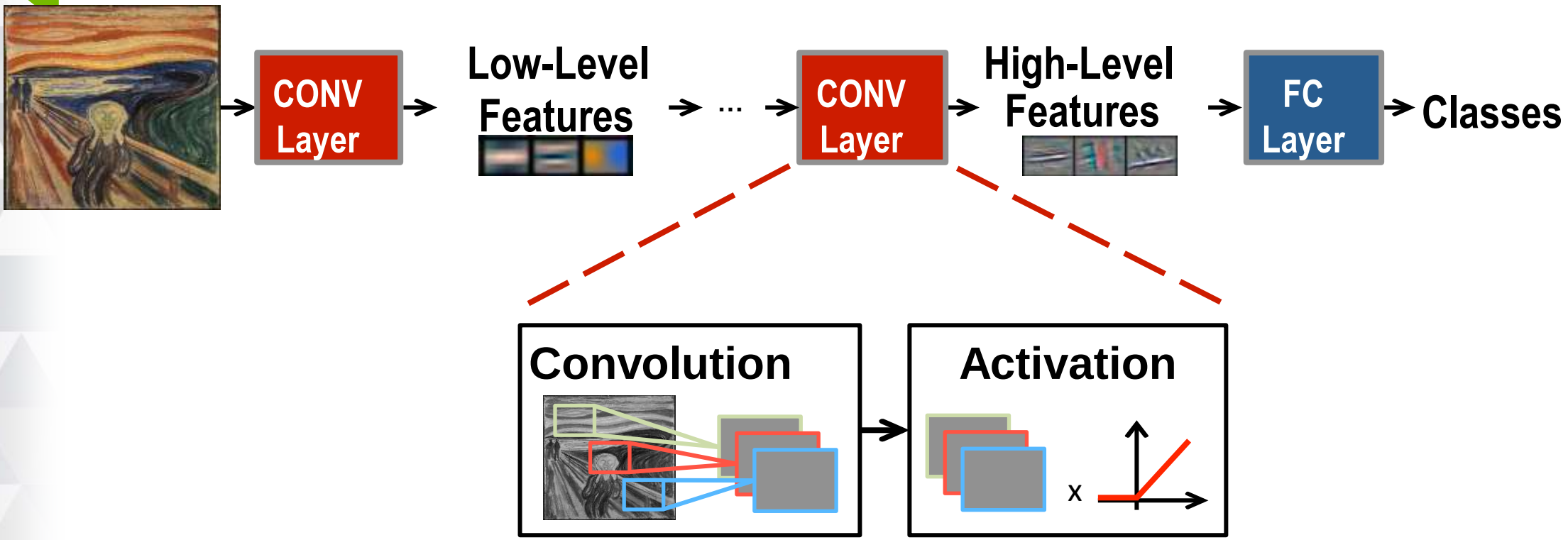
Inference vs. Training

- **Training: Determine weights**
  - **Supervised:**
    - **Training set has inputs and outputs, i.e., labeled**
  - **Reinforcement:**
    - **Output assessed via rewards and punishments**
  - **Unsupervised:**
    - **Training set is unlabeled**
  - **Semi-supervised:**
    - **Training set is partially labeled**
- **Inference: Apply weights to determine output**

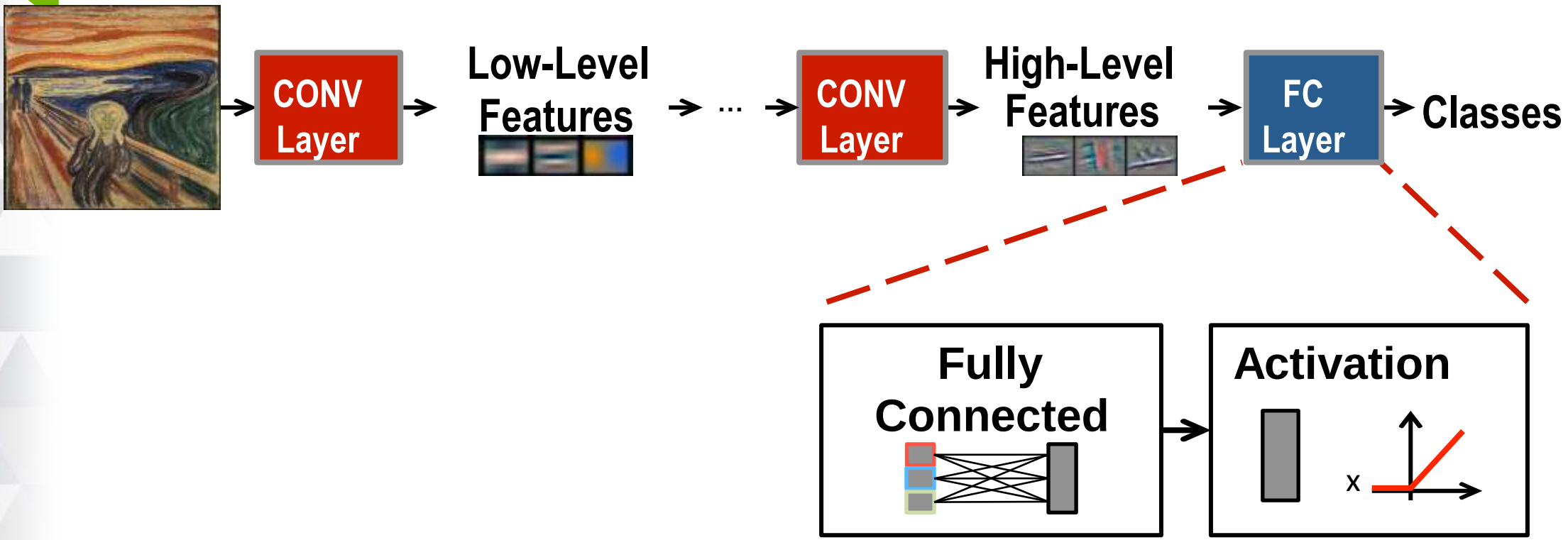
Deep Convolutional Neural Networks



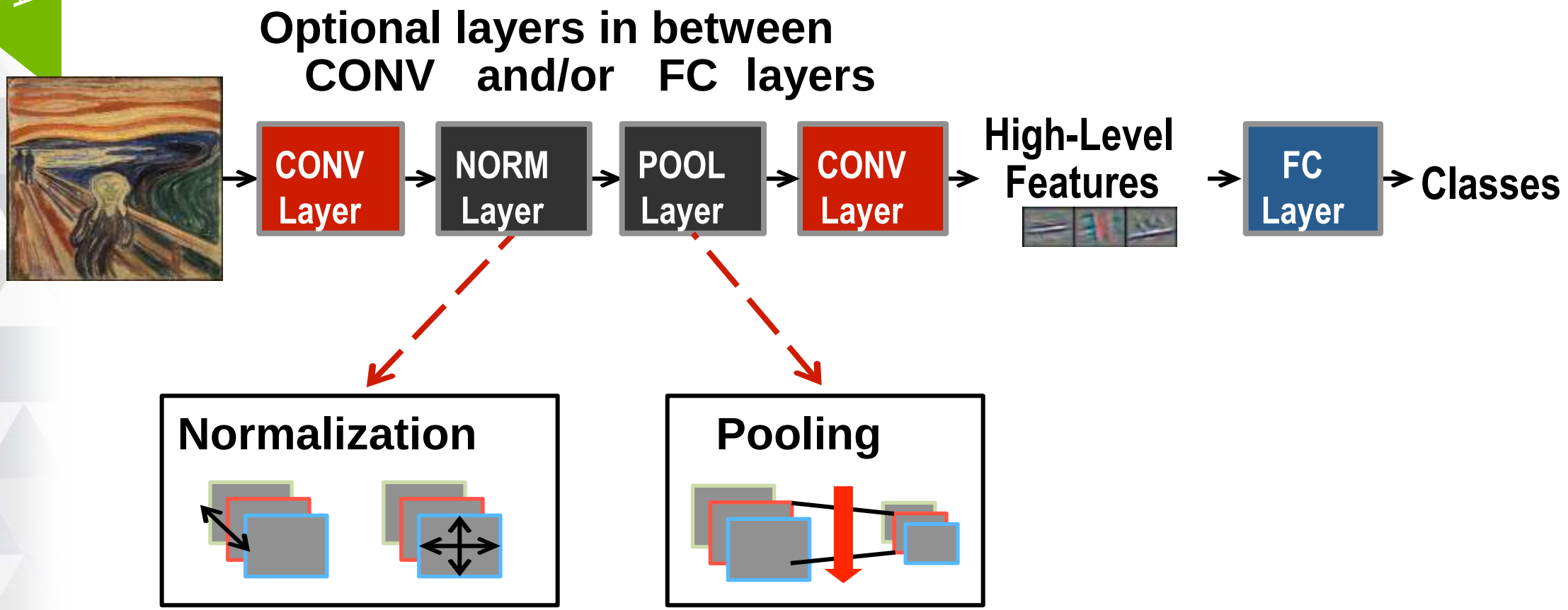
Deep Convolutional Neural Networks



Deep Convolutional Neural Networks

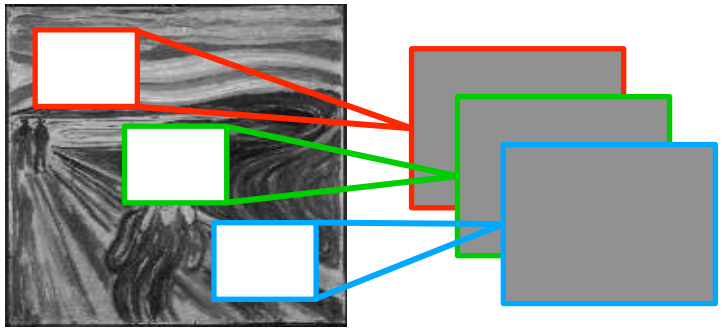


Deep Convolutional Neural Networks



Deep Convolutional Neural Networks

CONV Layer → NORM Layer → POOL Layer → CONV Layer → FC Layer



Convolutions account for more than 90% of overall computation, dominating **runtime** and **energy consumption**

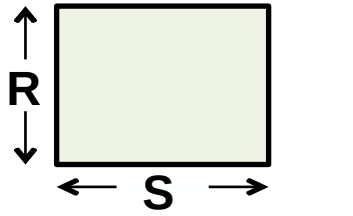
Deep Convolutional Neural Networks

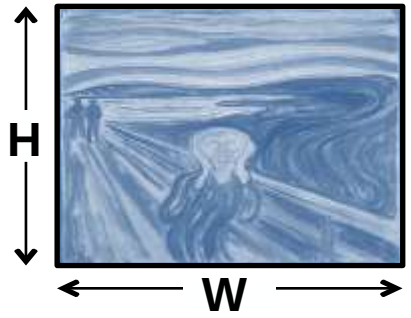
Understanding deep convolutional networks
http://rsta.royalsocietypublishing.org/content/374/2065/20150203
Generic Deep Networks with Wavelet Scattering
https://arxiv.org/pdf/1312.5940v2.pdf
Stéphane Mallat

Convolution (CONV) Layer

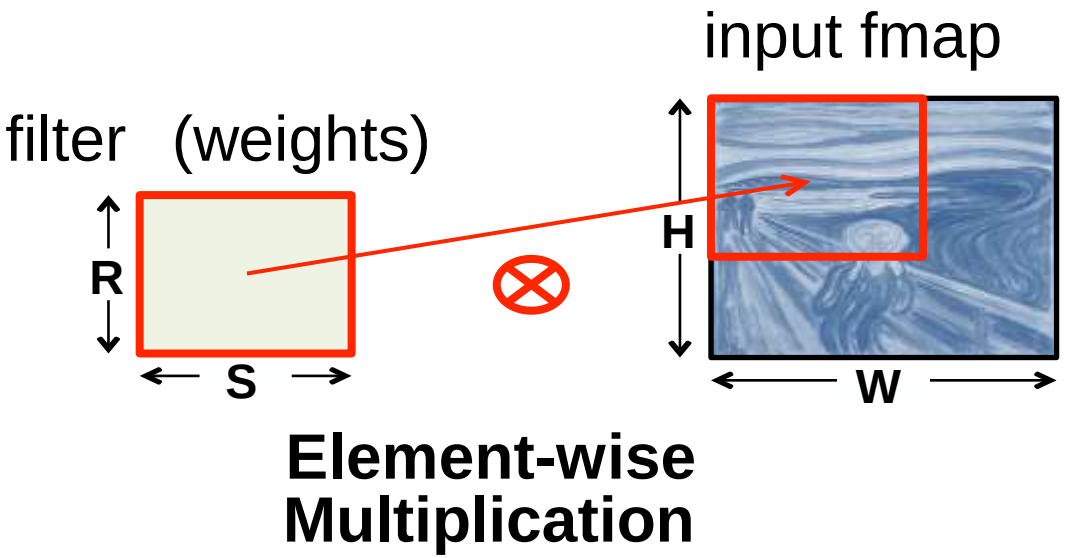
a plane of input activations
a.k.a. **input feature map (fmap)**

filter (weights)

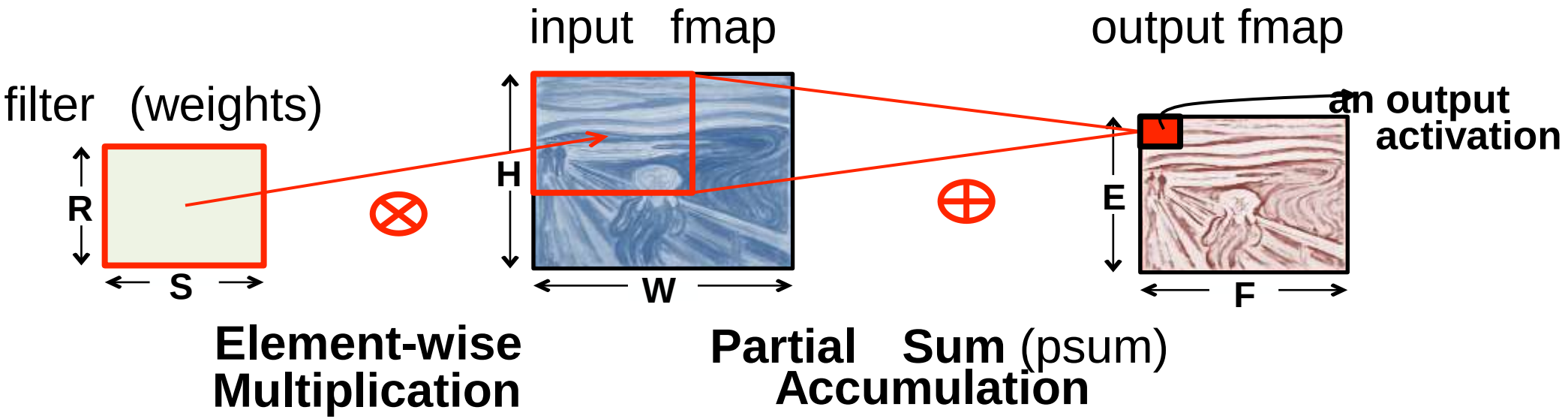




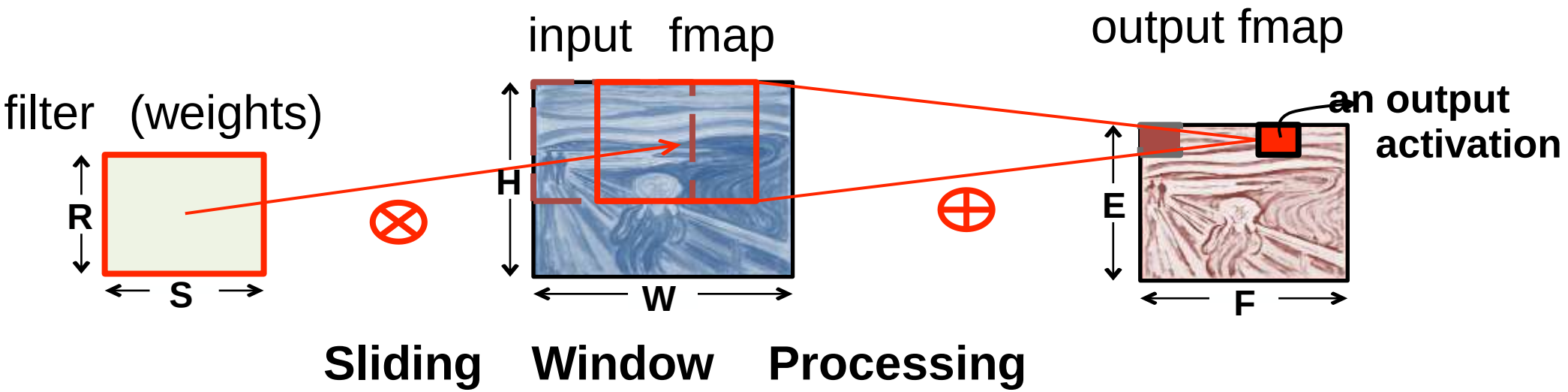
Convolution (CONV) Layer



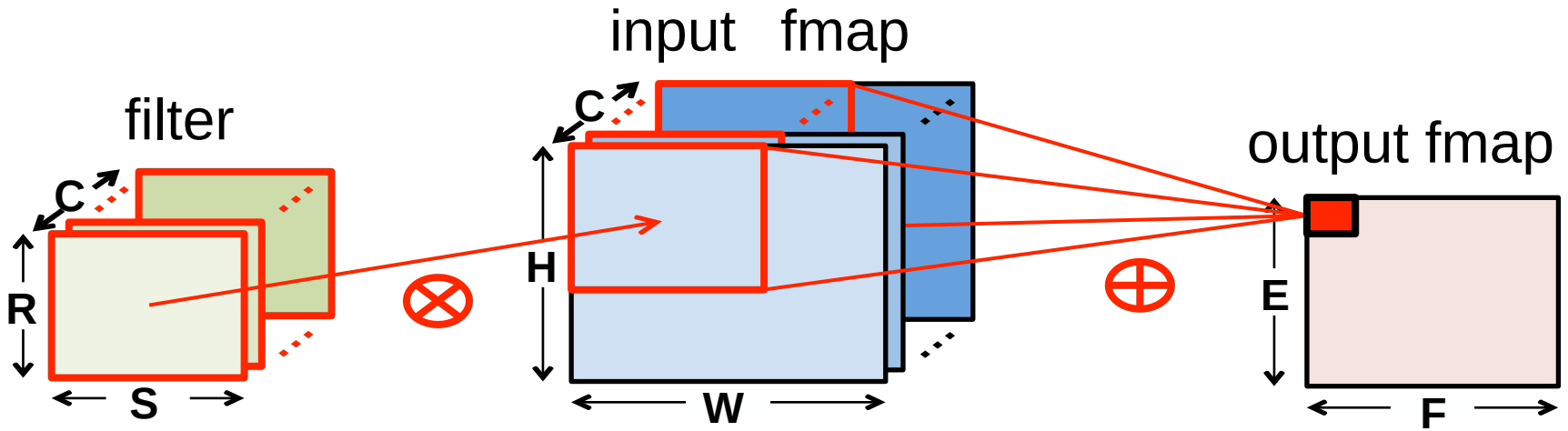
Convolution (CONV) Layer



Convolution (CONV) Layer

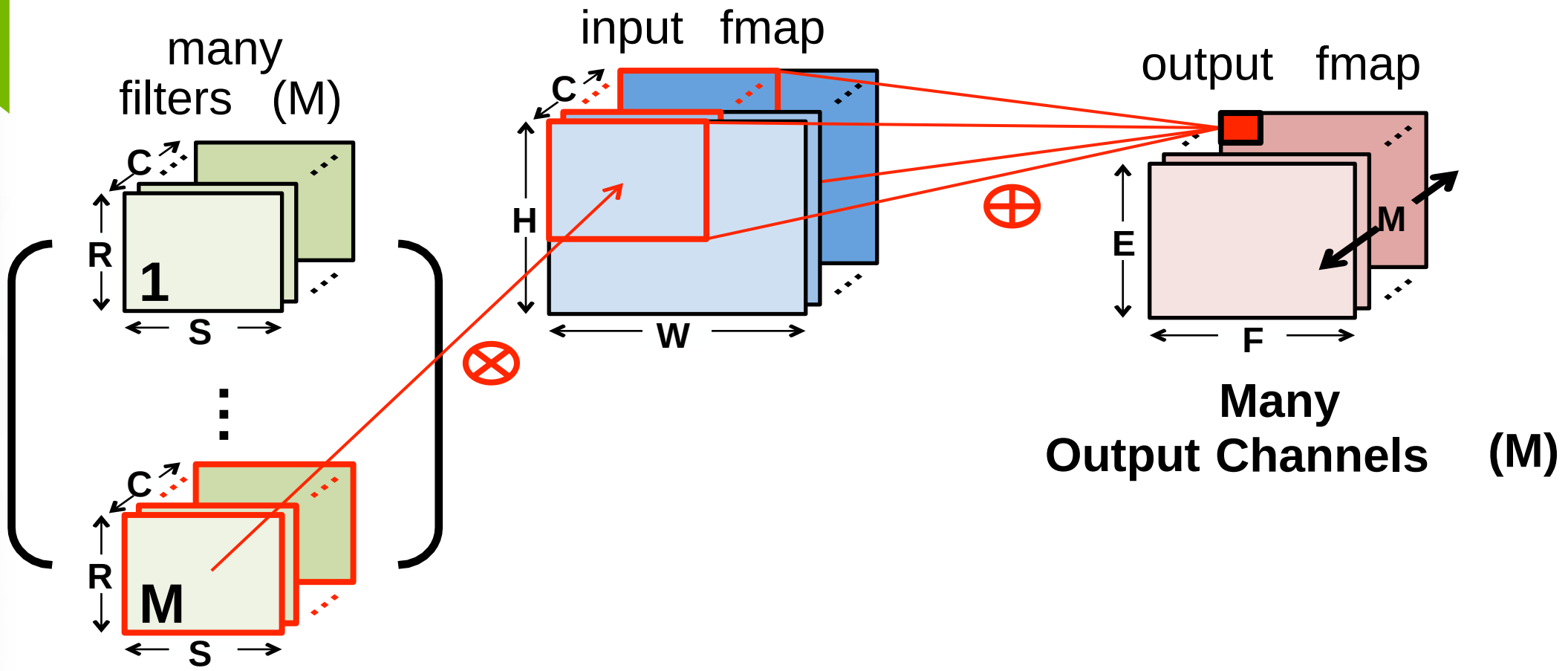


Convolution (CONV) Layer

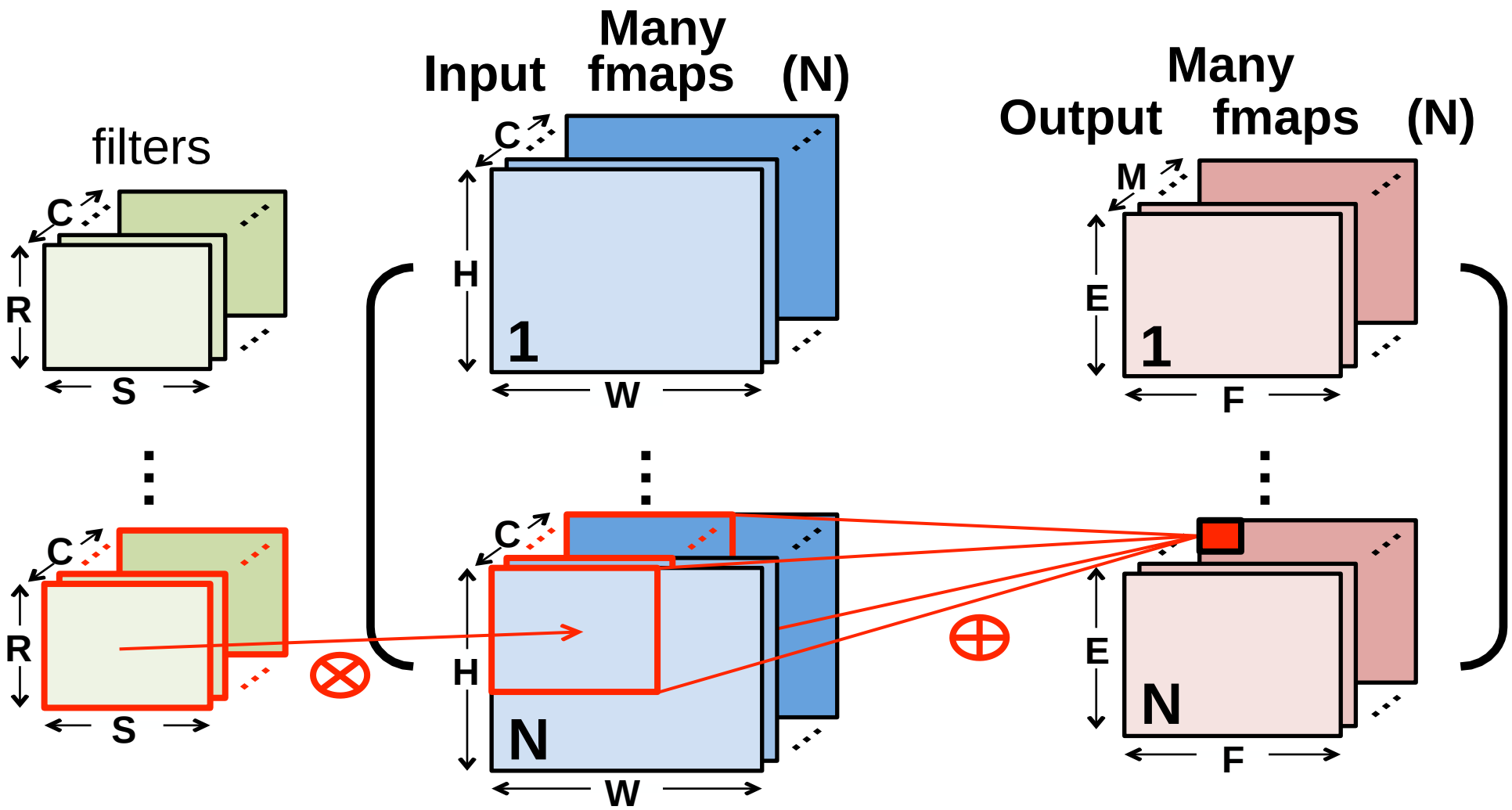


Many Input Channels (C)

Convolution (CONV) Layer



Convolution (CONV) Layer



CONV Layer Implementation

Output fmaps Biases Input fmaps Filter weights

$$\mathbf{O}[n][m][x][y] = \text{Activation}(\mathbf{B}[m] + \sum_{i=0}^{R-1}\sum_{j=0}^{S-1}\sum_{k=0}^{C-1} \mathbf{I}[n][k][Ux+i][Uy+j] \times \mathbf{W}[m][k][i][j]),$$

$$0 \le n < N, 0 \le m < M, 0 \le y < E, 0 \le x < F,$$

$$E = (H - R + U)/U, F = (W - S + U)/U.$$

Shape Parameter	Description
N	fmap batch size
M	# of filters / # of output fmap channels
C	# of input fmap/filter channels
H/W	input fmap height/width
R/S	filter height/width
E/F	output fmap height/width
U	convolution stride

CONV Layer Implementation

Naïve 7-layer for-loop implementation:

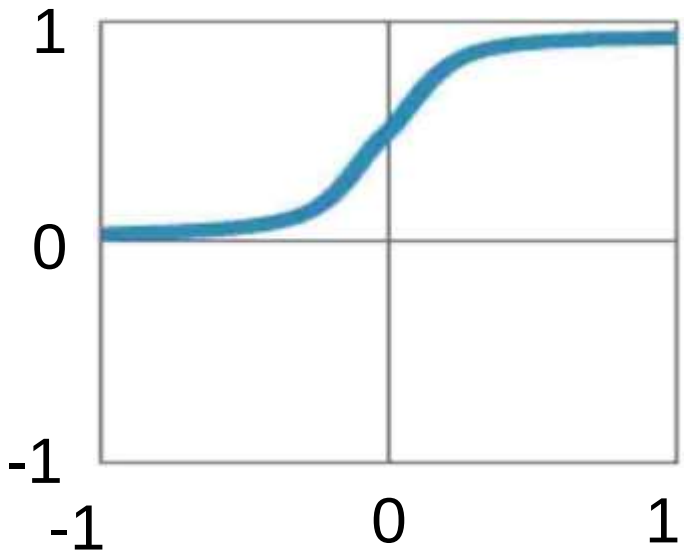
```
for (n=0; n<N; n++) {
    for (m=0; m<M; m++) {
        for (x=0; x<F; x++) {
            for (y=0; y<E; y++) {
                O[n][m][x][y] = B[m];
                for (i=0; i<R; i++) {  for (j=0; j<S; j++) {
                    for (k=0; k<c; k++) {
                O[n][m][x][y] += I[n][k][Ux+i][Uy+j] x W[m][k][i][j];
                }          }
                }
                O[n][m][x][y] = Activation(O[n][m][x][y]);
            }
        }
    }
}
```

for each output fmap value

convolve a window and apply activation

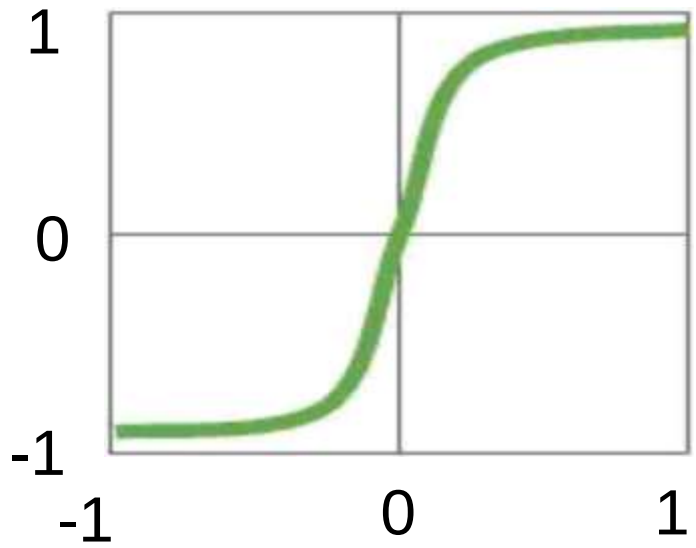
Traditional Activation Functions

Sigmoid



$y=1/(1+e^{-x})$

Hyperbolic Tangent

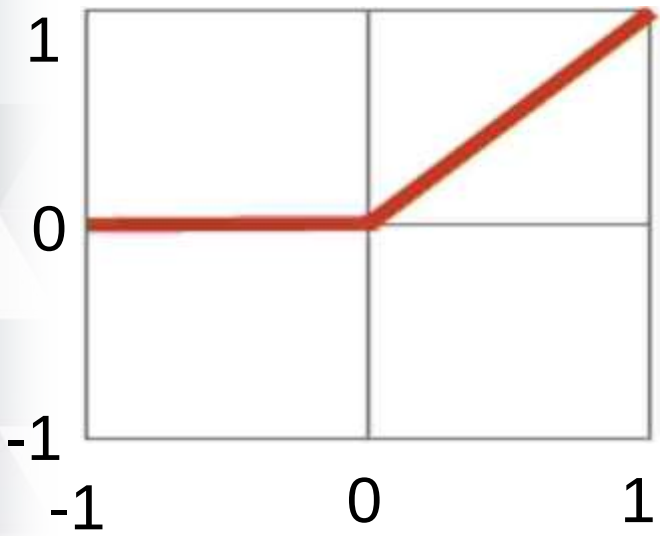


$y=(e^{x}-e^{-x})/(e^{x}+e^{-x})$

Image Source: Caffe Tutorial

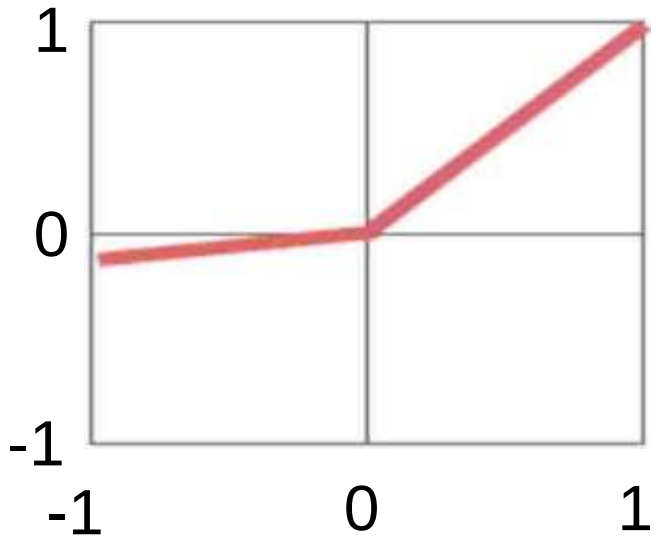
Modern Activation Functions

Rectified Linear Unit (ReLU)



$y=\max(0,x)$

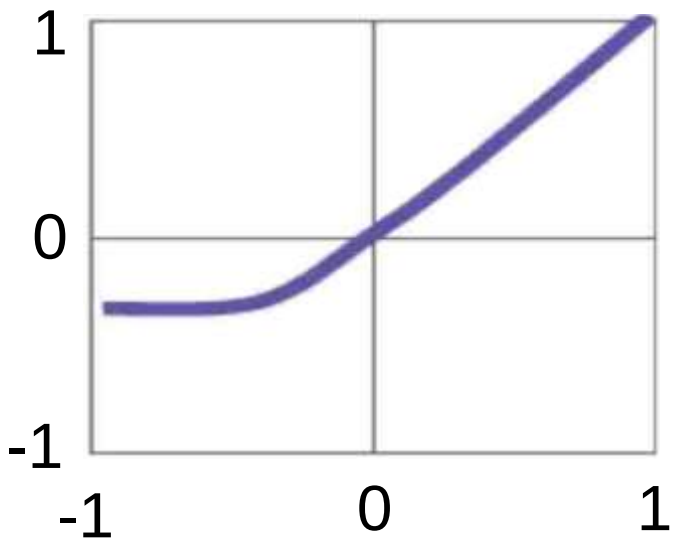
Leaky ReLU



$y=\max(\alpha x,x)$

α = small const. (e.g. 0.1)

Exponential LU

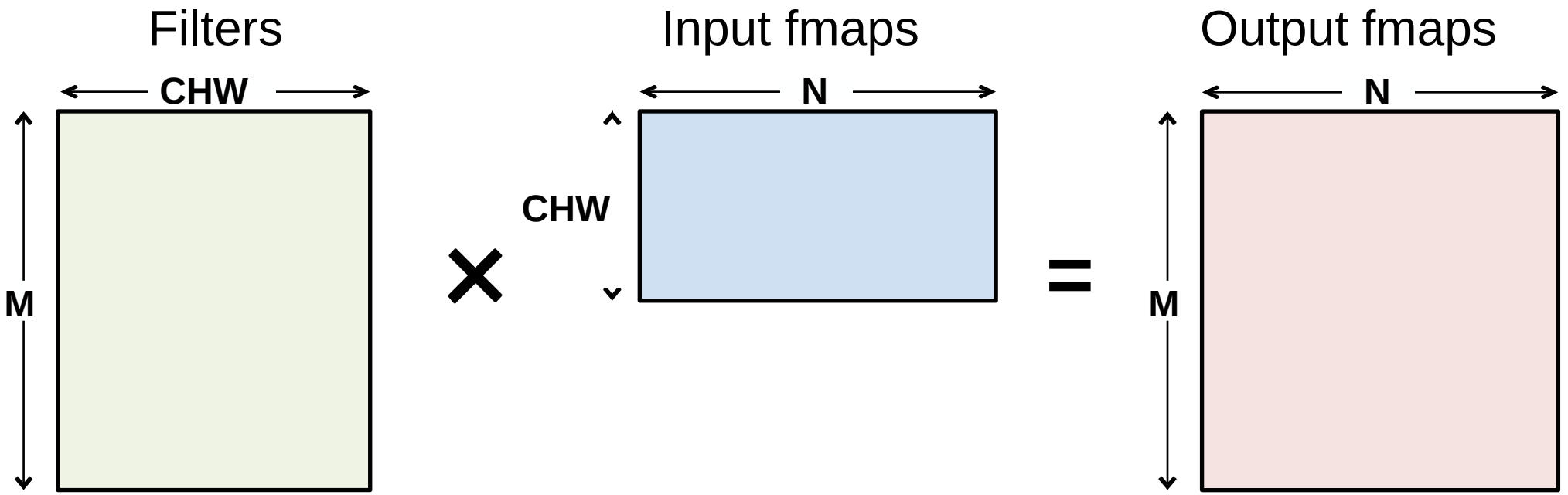


$$y=\begin{cases} x, & x\geq 0 \\ \alpha(e^{x}-1), & x<0 \end{cases}$$

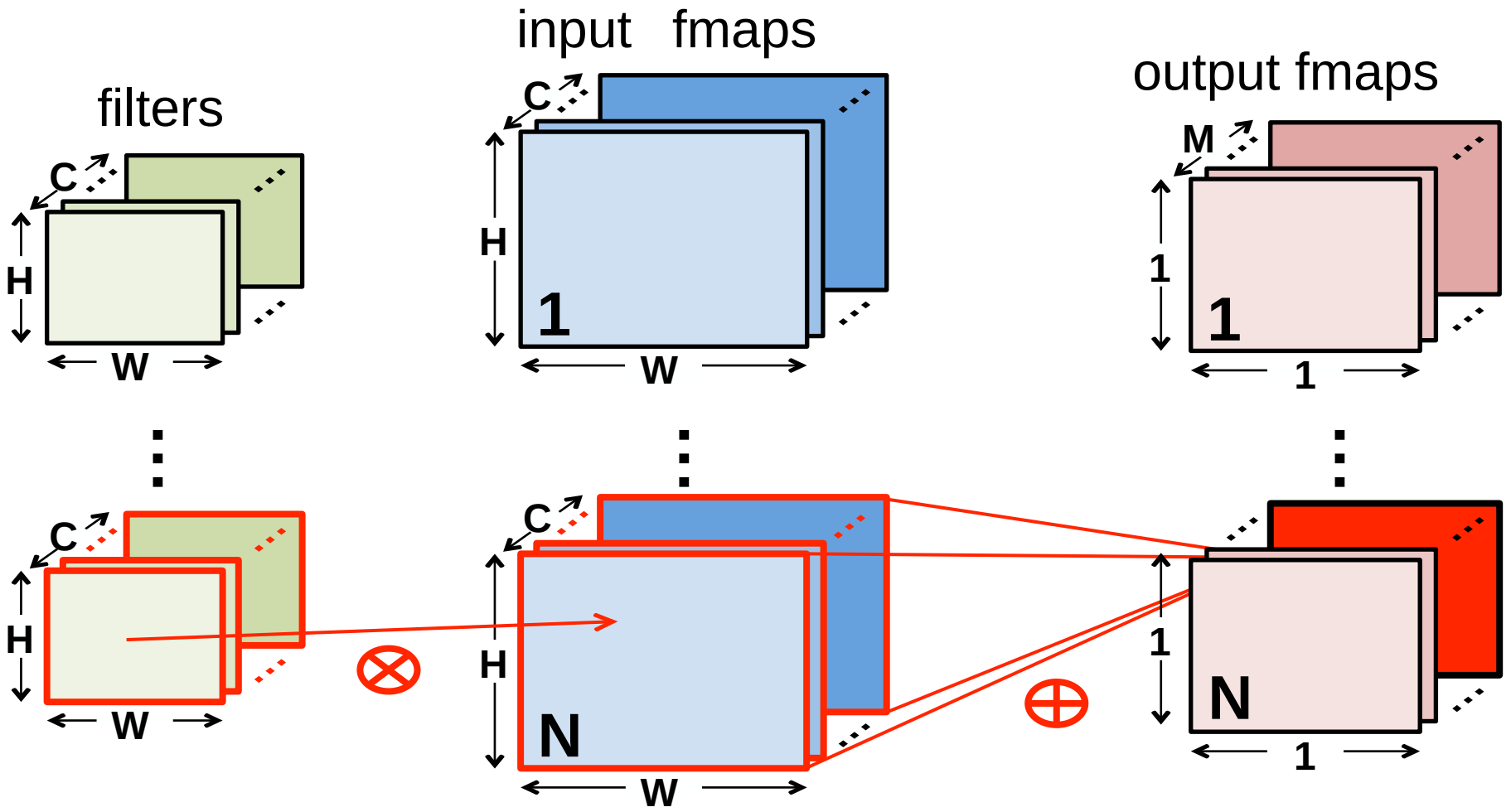
Image Source: Caffe Tutorial

Fully-Connected (FC) Layer

- Height and width of output fmaps are 1 (E = F = 1)
- Filters as large as input fmaps (R = H, S = W)
- Implementation: **Matrix Multiplication**



FC Layer – from CONV Layer POV



Pooling (POOL) Layer

- Reduce resolution of each channel independently
- Increase translation-invariance and noise-resilience
- Overlapping or non-overlapping à depending on stride

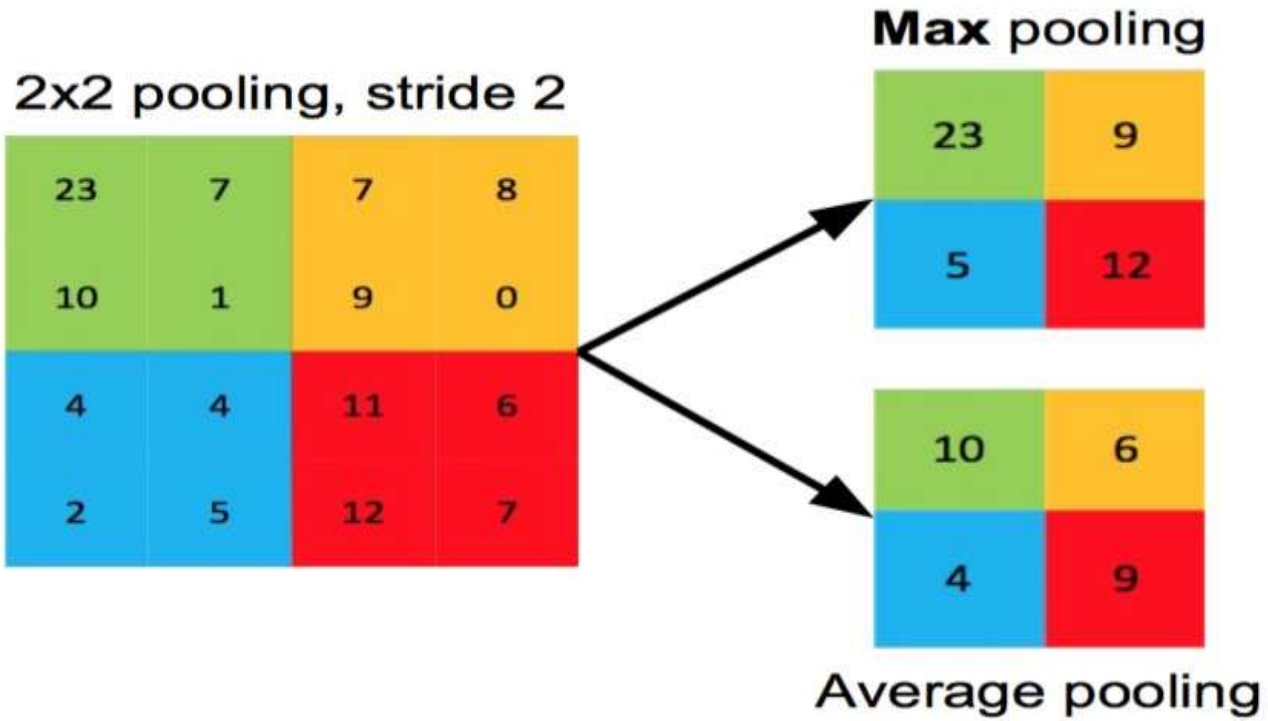


Image Source: Caffe Tutorial

POOL Layer Implementation

Naïve 6-layer for-loop max-pooling implementation:

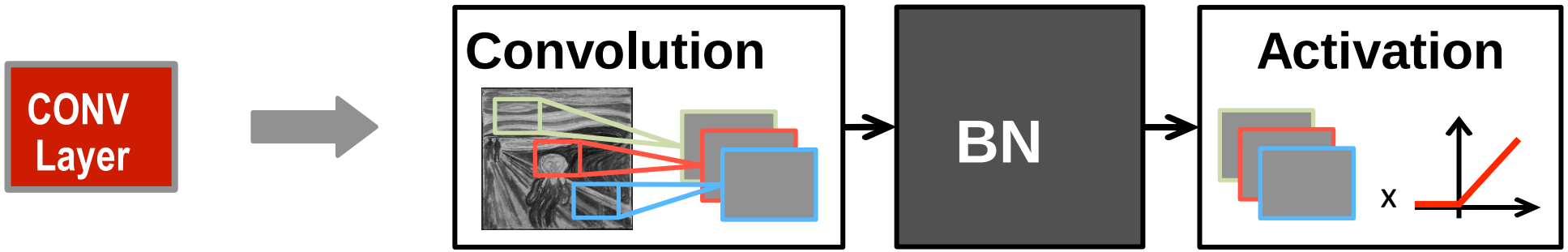
```
for  (n=0; n<N; n++) {
     for  (m=0; m<M; m++) {
          for  (x=0; x<F; x++) {
               for  (y=0; y<E; y++) {

                    max = -Inf;
                    for (i=0; i<R; i++) {
                        for (j=0; j<S; j++) {
                            if (I[n][m][Ux+i][Uy+j] > max)
                                max = I[n][m][Ux+i][Uy+j];
                        }
                    }

                    O[n][m][x][y]    = max;
               }
          }
     }
}
```

for each pooled value

{ find the max with in a window

Normalization (NORM) Layer

- **Batch Normalization (BN)**
  - Normalize activations towards mean=0 and std. dev.=1 based on the statistics of the training dataset
  - put **in between** **CONV/FC** and **Activation function**

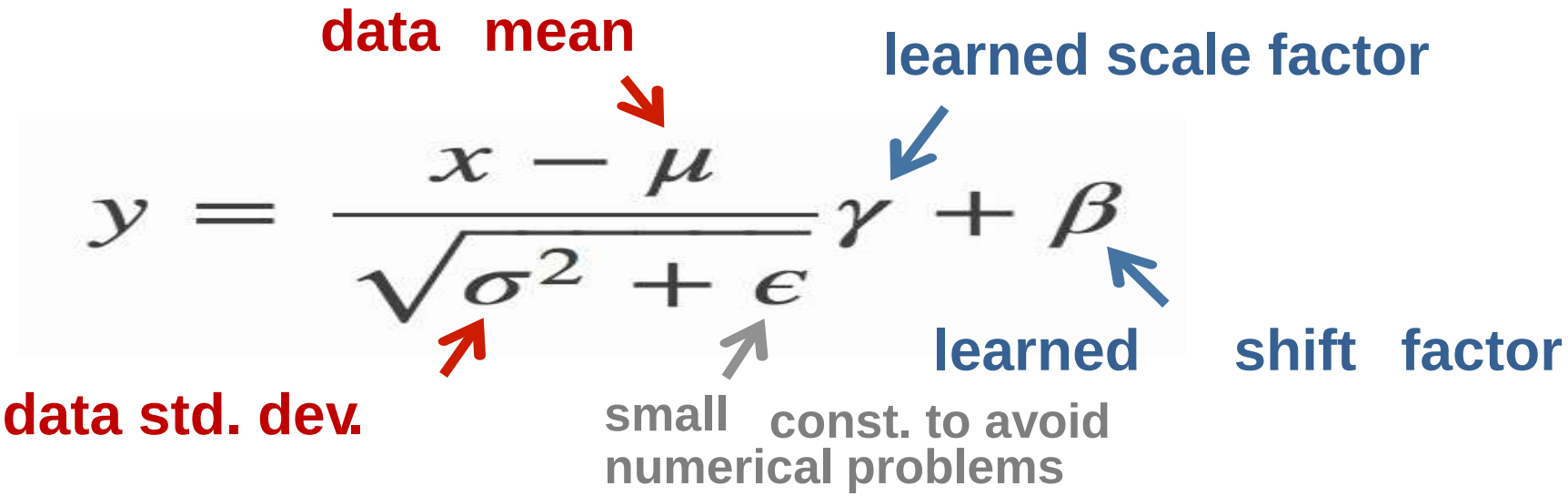


Believed to be key to getting high accuracy and faster training on very deep neural networks.

[Ioffe et al., ICML 2015]

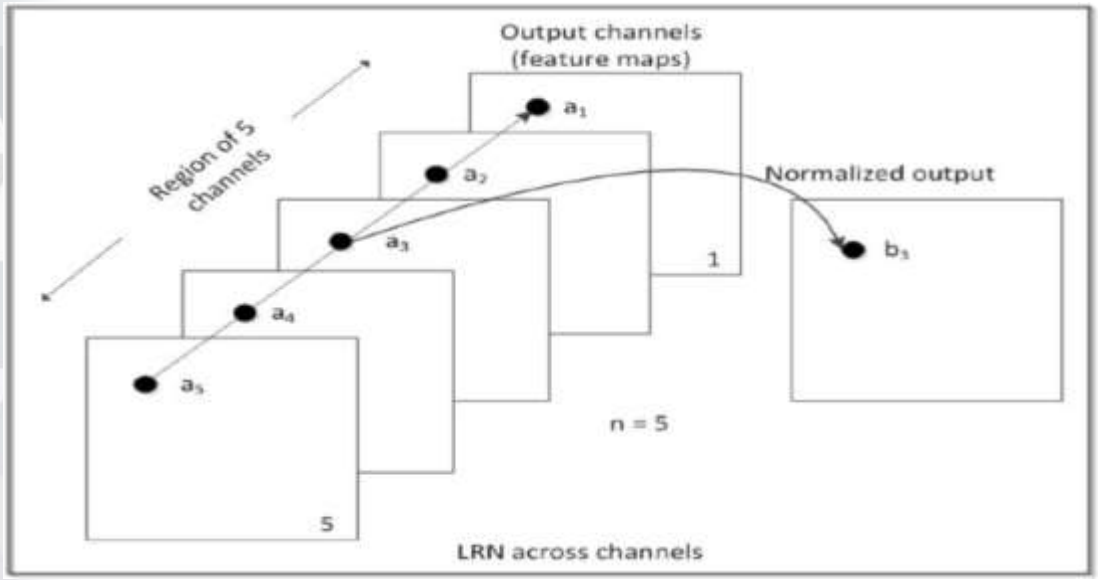
BN Layer Implementation

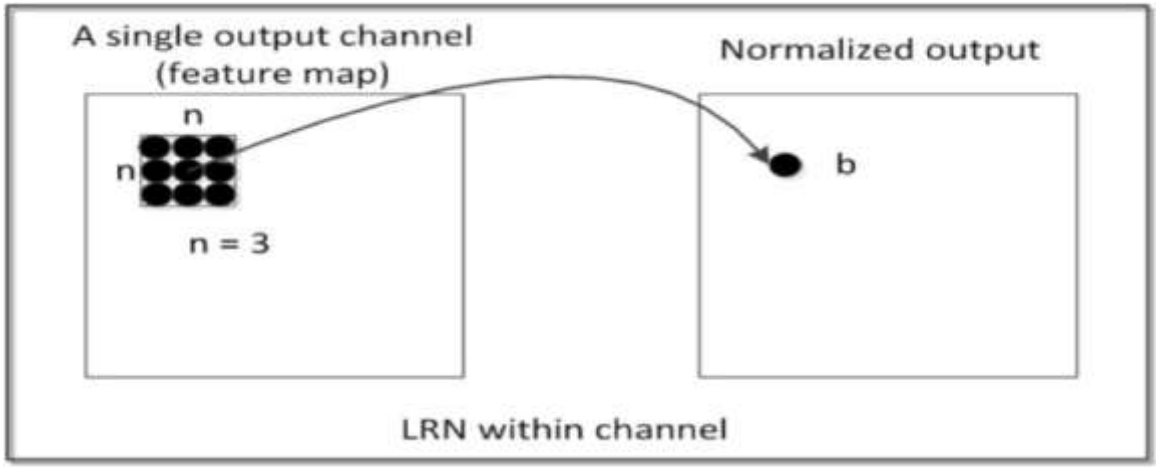
- The normalized value is further scaled and shifted, the parameters of which are learned from training



Normalization (NORM) Layer

- **Local Response Normalization (LRN)**
  - Tries to mimic the inhibition scheme in the brain





Now deprecated!

Image Source: Caffe Tutorial

Relevant Components for Tutorial

- **Typical operations that we will discuss:**
  - **Convolution (CONV)**
  - **Fully-Connected (FC)**
  - **Max Pooling**
  - **ReLU**

BUILD DNN WITH R

Mature Packages in/through R:

Packages	Backend	Compute Resources
nnet	C/C++	Single thread
nerualnet	C/C++	Single thread
DARCH	C/C++	Single thread
deepnet	C/C++	Single thread
H2O	JAVA	Multi-threads, multi-nodes
mxnet	C/C++/CUDA	Multi-threads, GPUs, multi-nodes
Tensorflow	C/C++/CUDA	Multi-threads, GPUs, multi-nodes

https://www.r-bloggers.com/deep-learning-with-mxnetr/

GTC 2016 (https://on-demand-gtc.gputechconf.com/gtcnew/on-demand-gtc.php)
S6853 - MXNet: Flexible Deep Learning Framework from Distributed GPU Clusters to Embedded Systems
L6143 - Train and Deploy Deep Learning for Vision, Natural Language and Speech Using MXNet

R with DL frameworks

MxNetR :

Tensor Flow : represents algorithm by data flow, deep learning

R interface discussion in here , GitHub implementation (here, here)

Caffe: Image Processing

Theano: Nature Language Processing, Speech Reorganization

In general, don't have official/mature supports for these frameworks

- don't have technical barriers but time-consuming for writing interfaces
- data scientists don't do image, speech and language translation too much

R with data analytics

Spark → SparkR; Hadoop → RHadoop; …

In general, R is relative mature for data analytics ranging from traditional methods to latest software and hardware architectures.

NVIDIA

- Code Frame: Stanford Open Course, CS231n, 2015

  Convolutional Neural Networks for Visual Recognition

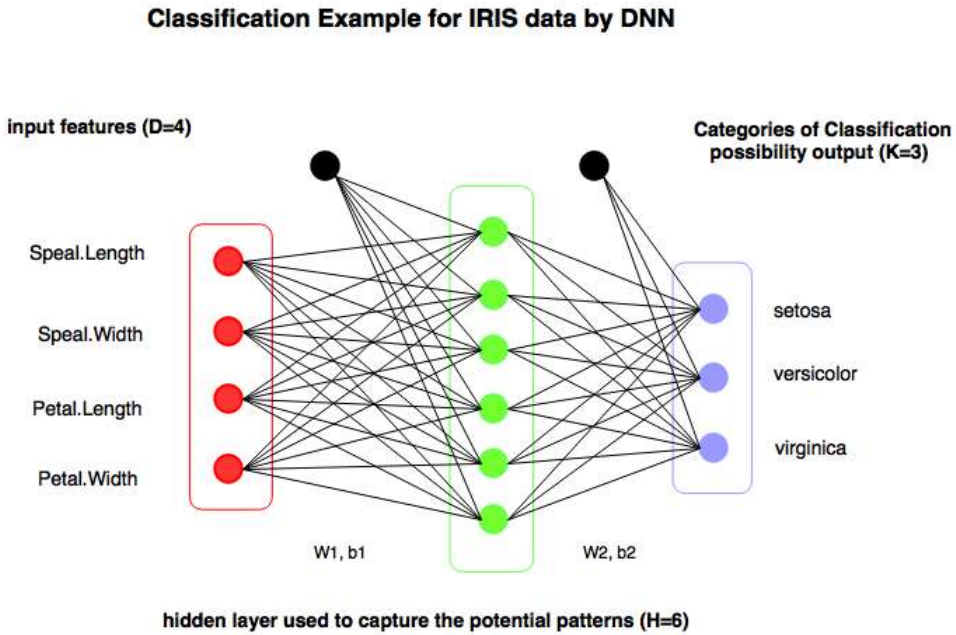
- Classification Network

  1 hidden layer w/ softmax, ReLu

- Vectorization Representation

- Fully connected network

- Python to R translations



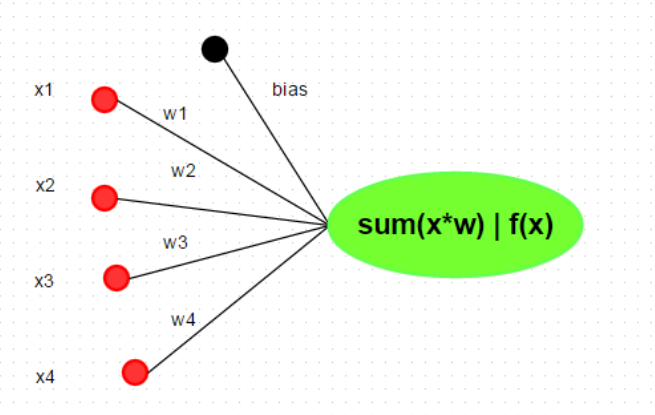
Core complements of DNN in R:

Weights and Bias : matrix representation

```
weight <- 0.01*matrix(rnorm(h*k), nrow=h, ncol=k)
bias   <- matrix(0, nrow=1, ncol=H)
```

Neuron : computation parts

```
neuron <- sweep(input %*% weights ,2, bias, '+')
neuron <- pmax(neuraon, 0)   # ReLu
```



Cost function : Softmax

```
score.exp <- exp(score)
probs     <-sweep(score.exp, 1, rowSums(score.exp), '/')
```

Prediction: Feed Forward

```
predict <- function(model, data = X.test) {
  new.data <- data.matrix(data)
  # Feed Forwad
  hidden.layer <- sweep(new.data %*% model$W1 ,2, model$b1, '+')
  # neurons : Rectified Linear
  hidden.layer <- pmax(hidden.layer, 0)
  score <- sweep(hidden.layer %*% model$W2, 2, model$b2, '+')
  # Loss Function: softmax
  score.exp <- exp(score)
  probs     <- sweep(score.exp, 1, rowSums(score.exp), '/')
  labels.predicted <- max.col(probs)
  return(labels.predicted)
}
```

Training : Feed Forward + Back propagation

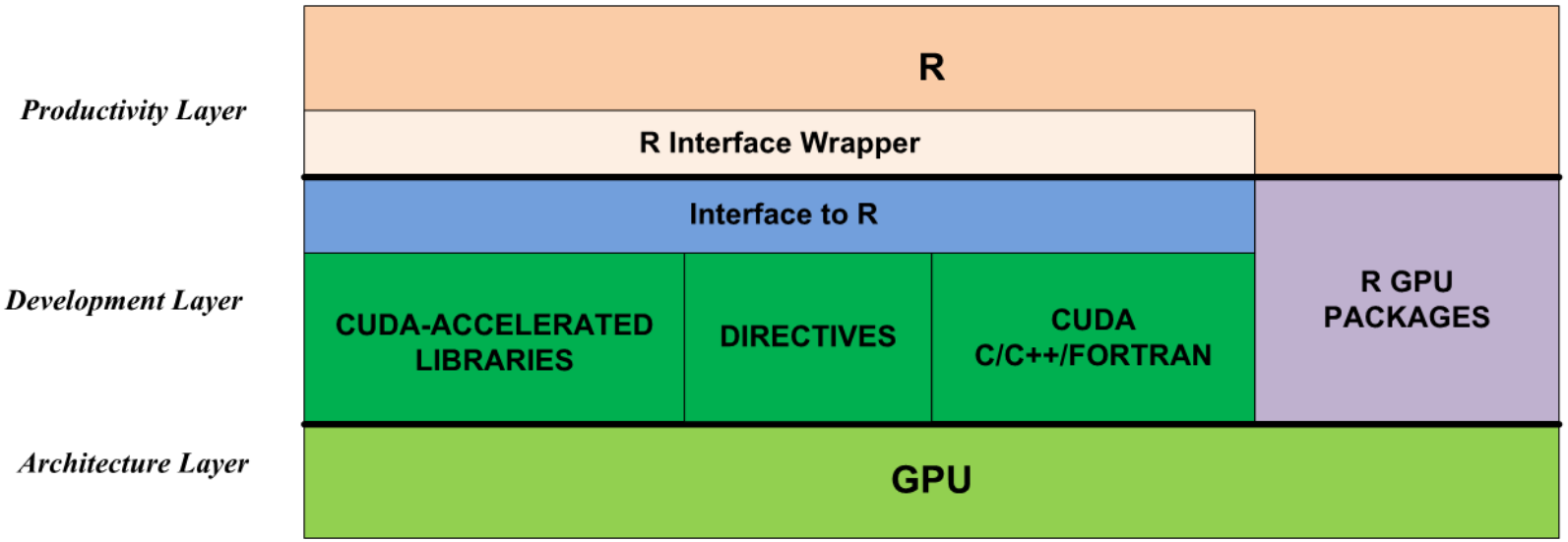
```
train <- function(x, y, model, traindata, hidden,…) {
     # 1. Feed Forwad . . .
     # 2. Compute the loss . . .
     # 3. Backward
       dscores <- probs
       dscores[Y.index] <- dscores[Y.index] -1
       dscores <- dscores / batchsize
       dW2 <- t(hidden.layer) %*% dscores
       db2 <- colSums(dscores)
       dhidden <- dscores %*% t(W2)
       dhidden[hidden.layer <= 0] <- 0
       dW1 <- t(X) %*% dhidden
       db1 <- colSums(dhidden)
       # update ....
 }
```

GPU ACCELERATION

NVIDIA

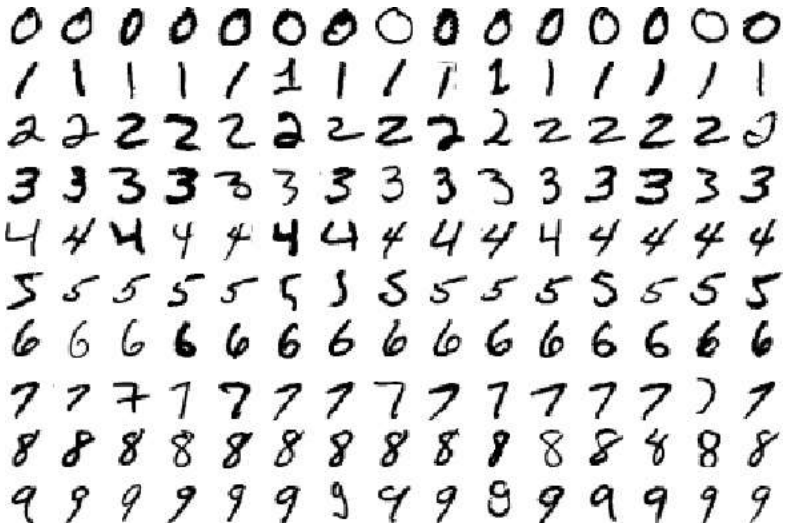
RECAP : How to accelerate R with GPU ?

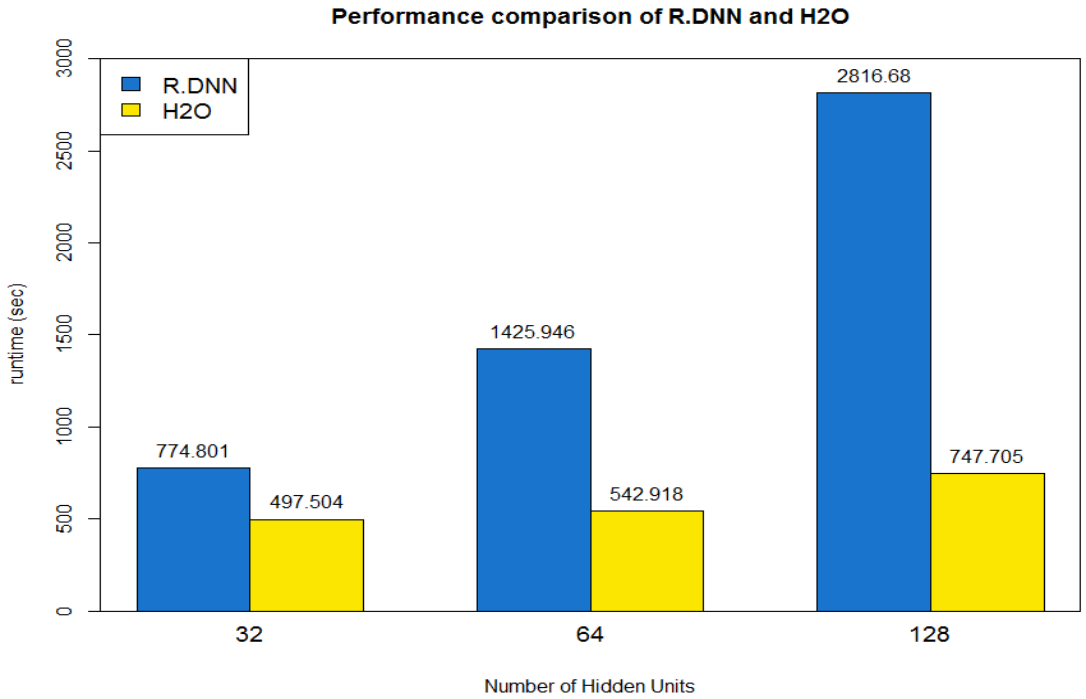
P.Zhao GTC15 talk: Accelerate R Applications with CUDA



Benchmark : MNIST handwritten digit dataset

- Input features: 28X28 = 784, Output classes: 10 (0-9);
- Training Set 60,000, testing set: 1 0,000
- DNN Architecture: 2-layers fully connected neural network



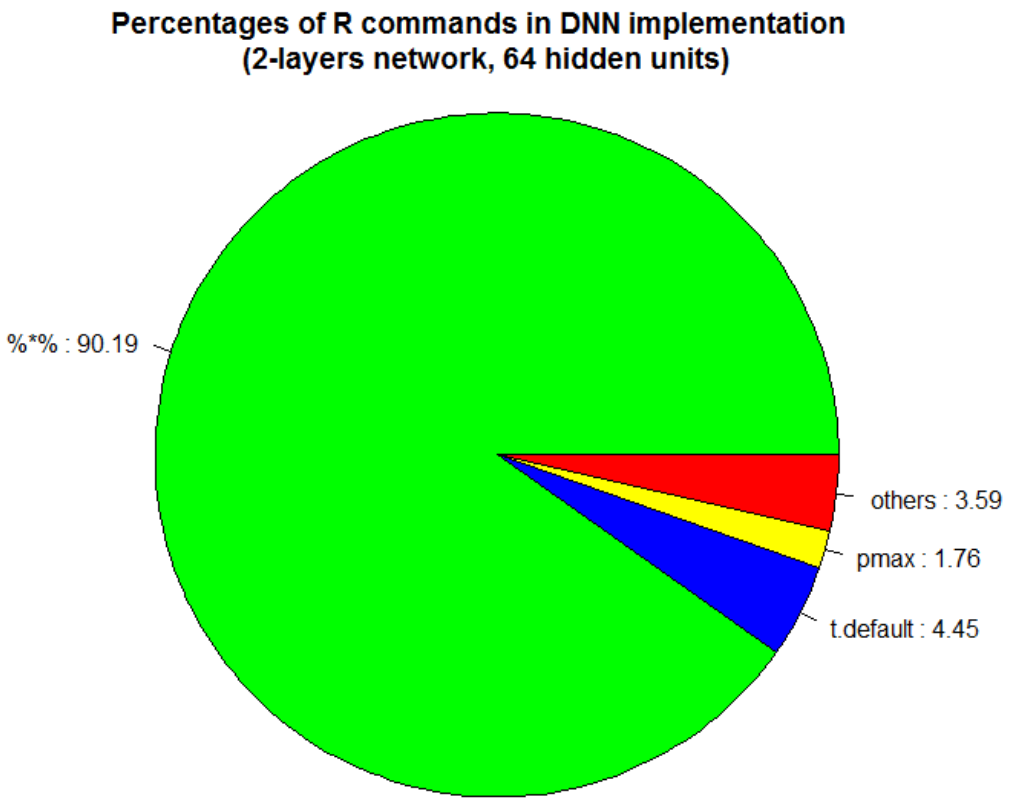


CPU: Ivy Bridge E5-2690 v2 @ 3.00GHz, dual socket 10-core, 128G RAM;

Profiling

Rprof(), summaryRprof()

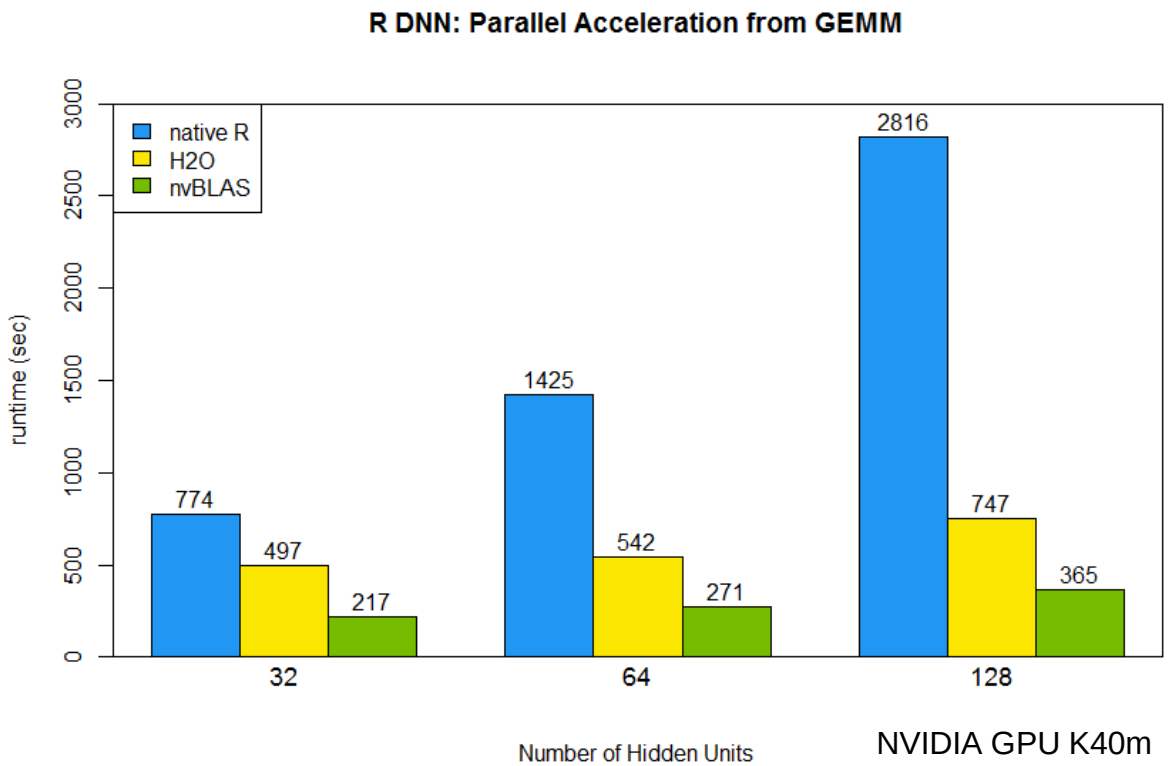
Break Donw R DNN Runtime				
	total.time	total.pct	self.time	self.pct
train.dnn	1386	100	9.74	0.7
%*%	1250.08	90.19	1250.08	90.19
sweep	676.32	48.8	1.58	0.11
t	61.64	4.45	0.02	0
t.default	61.62	4.45	61.62	4.45
pmax	28.42	2.05	24.4	1.76
aperm	21.96	1.58	0	0
aperm.default	11.6	0.84	11.6	0.84
array	10.36	0.75	10.36	0.75
<=	5.72	0.41	5.72	0.41
mostattributes<-	4.02	0.29	4.02	0.29
exp	3.6	0.26	3.6	0.26
unname	1.46	0.11	0.18	0.01
is.data.frame	1.28	0.09	1.28	0.09
data.matrix	1.28	0.09	0	0
colSums	0.86	0.06	0.86	0.06
/	0.52	0.04	0.52	0.04
rowSums	0.36	0.03	0.36	0.03
-	0.04	0	0.04	0
sum	0.02	0	0.02	0



DROP-IN ACCELERATION

By nvBLAS Library on Linux

> env LD_PRELOAD=libnvblas.so R CMD BATCH MNIST_DNN.R



NVIDIA GPU K40m

OPTIMIZATIONS

- Profiling again after NVIDIA GPU acceleration

Break Donw R DNN Runtime (nvBLAS, HU=64)				
function	total.time	total.pct	self.time	self.pct
train.dnn	274	100	10.74	3.92
%*%	114.58	41.82	114.58	41.82
sweep	**87.28**	31.85	1.8	0.66
t.default	**73.42**	26.8	73.42	26.8
t	**73.42**	26.8	0	0
pmax	30.9	11.28	24.62	8.99
aperm	29.74	10.85	0.04	0.01
aperm.default	19.08	6.96	19.04	6.95
array	10.62	3.88	10.62	3.88
mostattributes<-	6.28	2.29	6.26	2.28

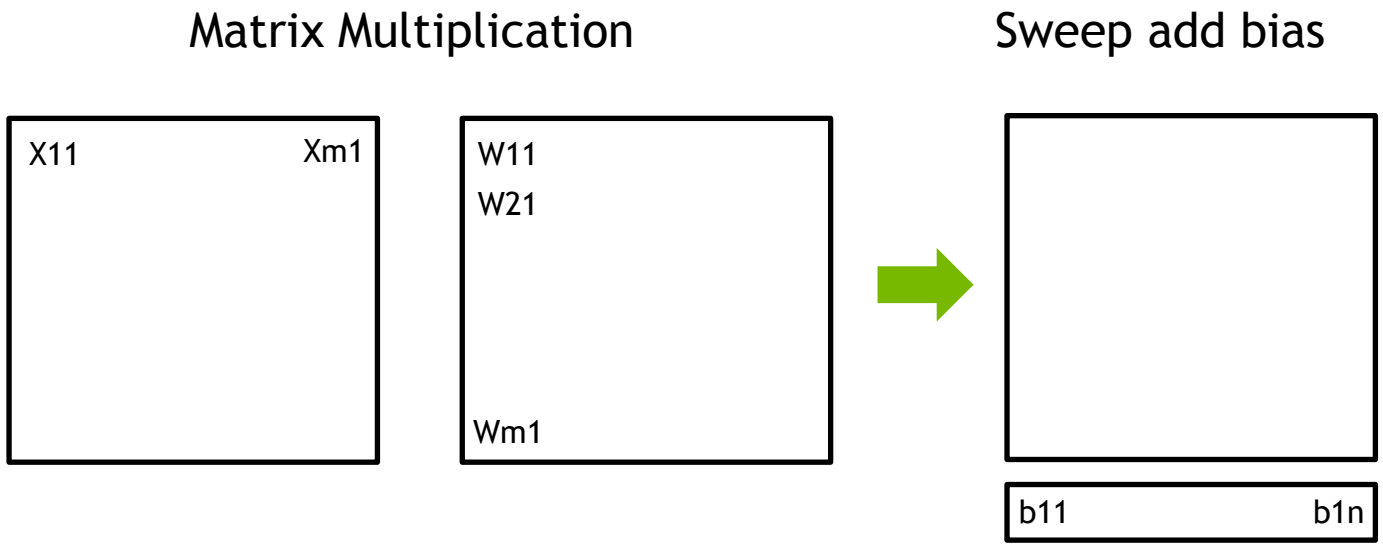
NVIDIA

Opt.1 : replace *t(X) %*% matrix* and *matrix %*% t(X)* with R internal function

```
# original:  t() with matrix multiplication
    dW2     <- t(hidden.layer) %*% dscores
    dhidden <- dscores %*% t(W2)

# Opt1: use builtin function
   dW2     <- crossprod(hidden.layer, dscores)
   dhidden <- tcrossprod(dscores, W2)
```

Opt.2 : replace *sweep()* by matrix multiplication

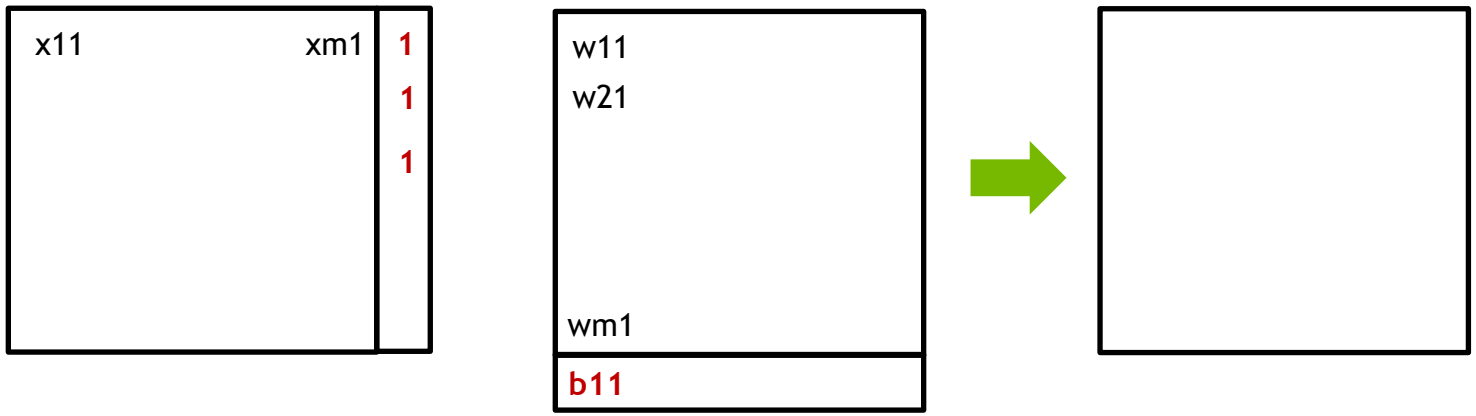
```
# Opt2: original code
 hidden.layer <- sweep(X %*% W1 ,2, b1, '+')
```



Opt.2 : replace *sweep()* by matrix multiplication

```
# Opt2: remove `sweep`
 hidden.layer <- X1 %*% W1b1
```

Matrix Multiplication



X1 <- cbind(X, rep(1, nrow(X)))

W1b1 <- rbind(W1, b1)

Optimization for R DNN (nvBLAS, HU=64)			
by.self	original	Opt1: replace t()	Opt2: remove sweep()
%*%	**112.02**	**53.28**	**53.72**
sweep	**90.46**	**86.7**	-
t	**73.98**	-	-
t.default	**73.96**	-	-
aperm	33.34	30.78	-
pmax	32.52	31.44	**31.58**
aperm.default	22.36	19.84	-
array	10.98	10.92	-
crossprod	-	**23.06**	**23.4**
tcrossprod	-	**2.52**	**2.54**
cbind	-	-	8.9
Total (sec)	266.28	166.76	144.71
Speedup	**1X**	**1.60X**	**1.84X**

GREEN: GPU accelerated parts
RED: Performance limiters

NVIDIA

Opt.3 : implement *pmax()* by CUDA

- *.Call* function in R with simple CUDA implementation of pmax()

(w/ similar method calling cuBLAS API to run R with GPU on Windows Platform)

```
# preload static object file
dyn.load("cudaR.so")

# GPU version of pmax
pmax.cuda <- function(A, threshold, devID=0L)
{
  rst <- .Call("pmax_cuda", A, threshold, as.integer(devID))
  dim(rst) <- dim(A)
  return(rst)
}
```

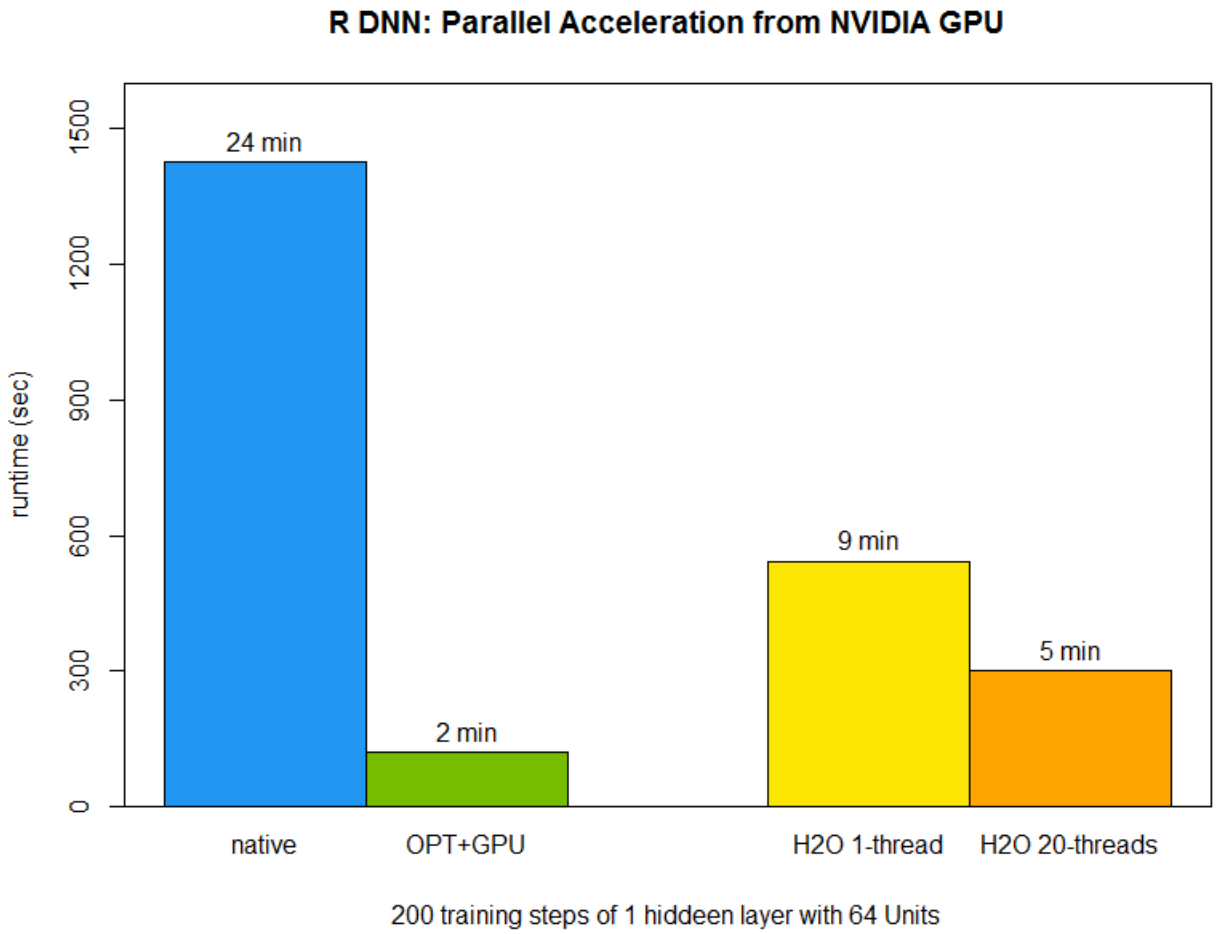
```
// CUDA: simple implementation of pmax
__global__ void pmax_kernel(double *A, const int M, const int N, const double threshold){
   int tid = blockIdx.x * blockDim.x + threadIdx.x;
   if(tid<M*N){ A[tid] = (A[tid] > threshold)?A[tid]:0; }
   return;
}
// Specified for DNN by .CAll format
SEXP pmax_cuda(SEXP A, SEXP threshold) {
   // Initialization including R to C data transfer, CUDA preparations
    . . .
   pmax_kernel<<<(mm*nn-1)/512+1, 512>>>(A_d, mm, nn, gw);
   cudaMemcpy(REAL(Rval), A_d, mm*nn*sizeof(double),  cudaMemcpyDeviceToHost);
   cudaDeviceSynchronize();
   // Free data, unprotect …
   return Rval;
}
```

Final Profiling:

Optimizations for R DNN (HU=64)					
by.total	**native R**	**nvBLAS + R**			**CUDA**
	base code	base code	Opt1: replace t()	Opt2: replace sweep()	Opt3: pmax.cuda
train.dnn	1386	278.7	167.7	144.66	119.86
%*%	1250.08	112.02	53.28	53.72	**53.72**
sweep	676.32	90.46	86.7	-	-
t	61.64	73.98	-	-	-
t.default	61.62	73.96	-	-	-
aperm	21.96	33.34	30.78	-	-
pmax	28.42	32.52	31.44	31.58	**6.76**
aperm.default	11.6	22.36	19.84	-	-
array	10.36	10.98	10.92	-	-
crossprod	-	-	23.06	23.4	**23.26**
tcrossprod	-	-	2.52	2.54	**2.62**
cbind	-	-	-	8.9	8.92
Total	**1425.946**	**266.28**	**166.76**	**144.71**	**119.40**
		1X	**1.60X**	**1.84X**	**2.24X**
Speedup	**1X**	**5.36X**	**8.55X**	**9.85X**	**11.94X**

GPU TECHNOLOGY CONFERENCE

Performance on Linux



SCALING OUT WITH MULTI-GPUS

NVIDIA

DATA PARALLEL BY HOGWILD!

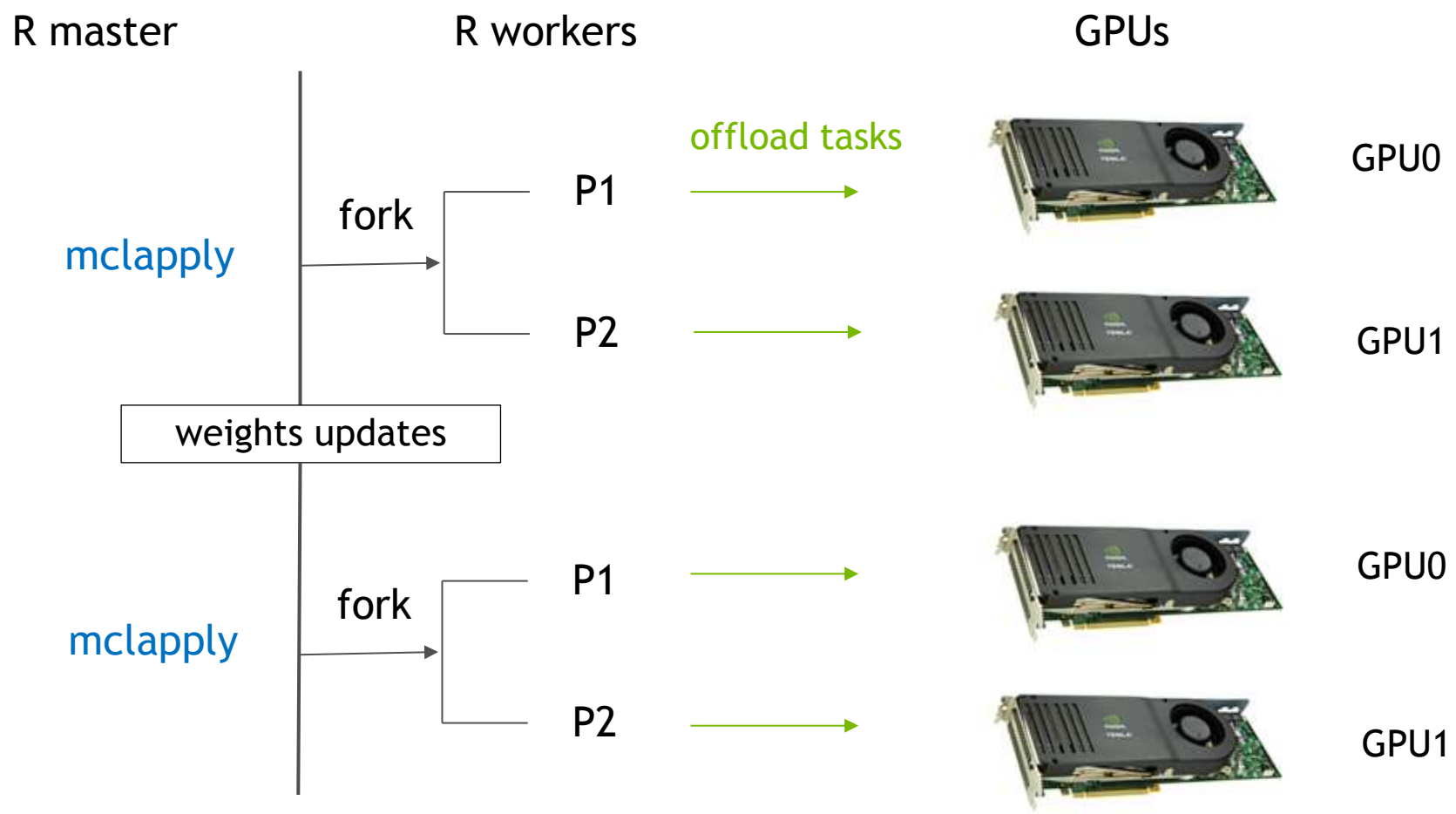
HOGWILD!

- A lock-free approach to parallelizing stochastic gradient descent
- MapReduce-like parallel-processing framework

DNN Training

- Each worker updates local weights/bias based on parts (1/N) of data
- Master collects and average all weights/bias from each worker
- Each worker update its weights/bias

Extend ‘multicores’ solution to multiGPUs



DATA DECOMPOSITION

- mclapply function to map data into each R processor

```
# Parallel Training
res <- mclapply(1:devNum, function(id) {
        train.dnn.cublas(x, y, omodel=para.model,
                         taindata=traindata[N.start[id]:N.end[id],],
                         devType="GPU", devID=(id-1), . . .) },
         mc.cores=devNum, mc.preschedule=TRUE)

# Model Updata
para.model <- list( W1= W1.sum/devNum, b1= b1.sum/devNum,
                    W2= W2/si,/devNum, b2= b2.sum/devNum)
```

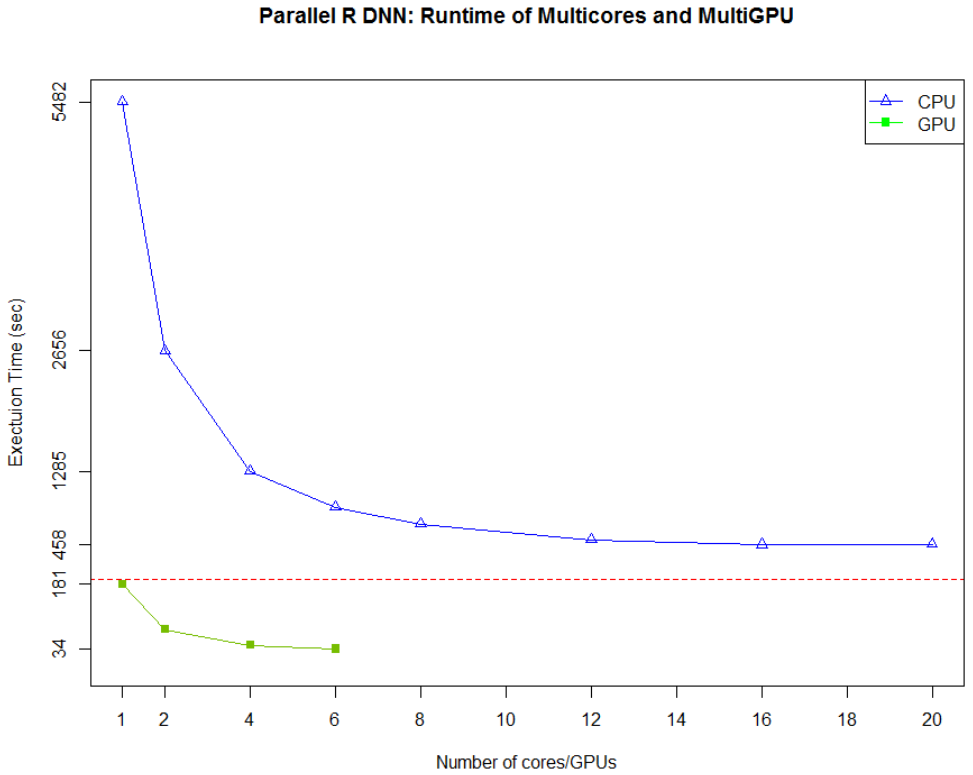
NVIDIA

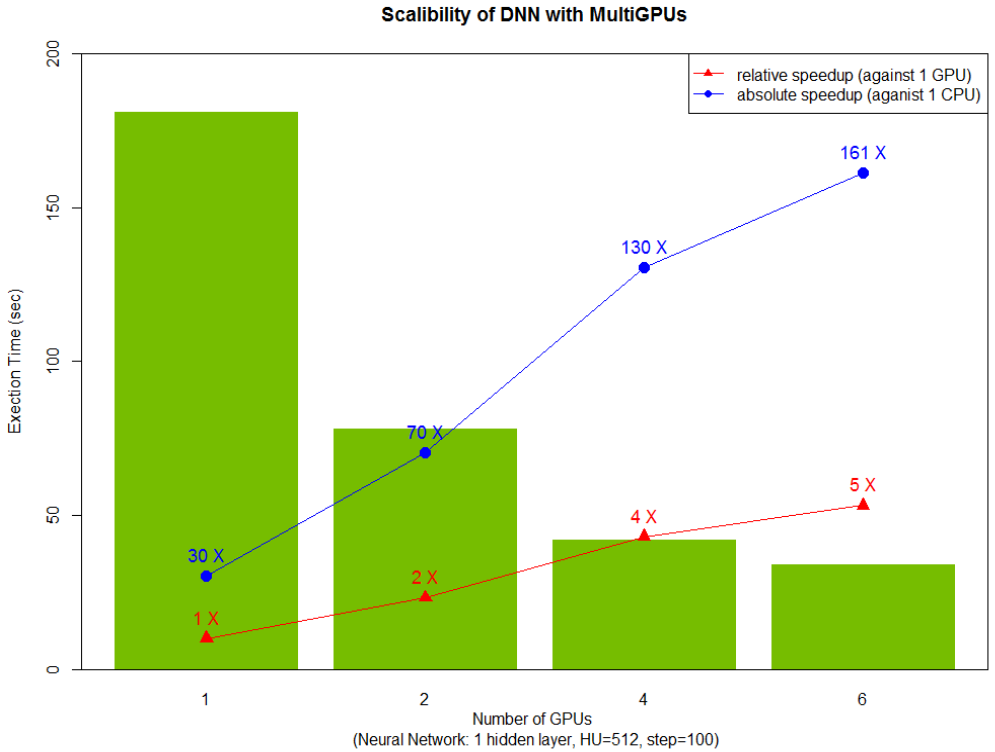
OFFLOAD TASKS TO GPUS

- Explicitly call cuBLAS API and pmax.cuda functions
- Set the GPU ID based on R's thread ID

```
# R level function call
res <- cuBLAS(hidden.layer, dscores, transA=T, devID=devID)

// GEMM cuda call by .CAll format and simplified for DNN
SEXP gemm_cuda(SEXP A, SEXP B, SEXP transA, SEXP transB, SEXP devID)
{   // init  . . .
    cudaSetDevice(gpuID);
   // cuBLAS: double precision matrix multiplication, DGEMM
   cublasDgemm(handle, cuTransA, cuTransB, mt, nt, kt, . . .);
   . . .
}
```

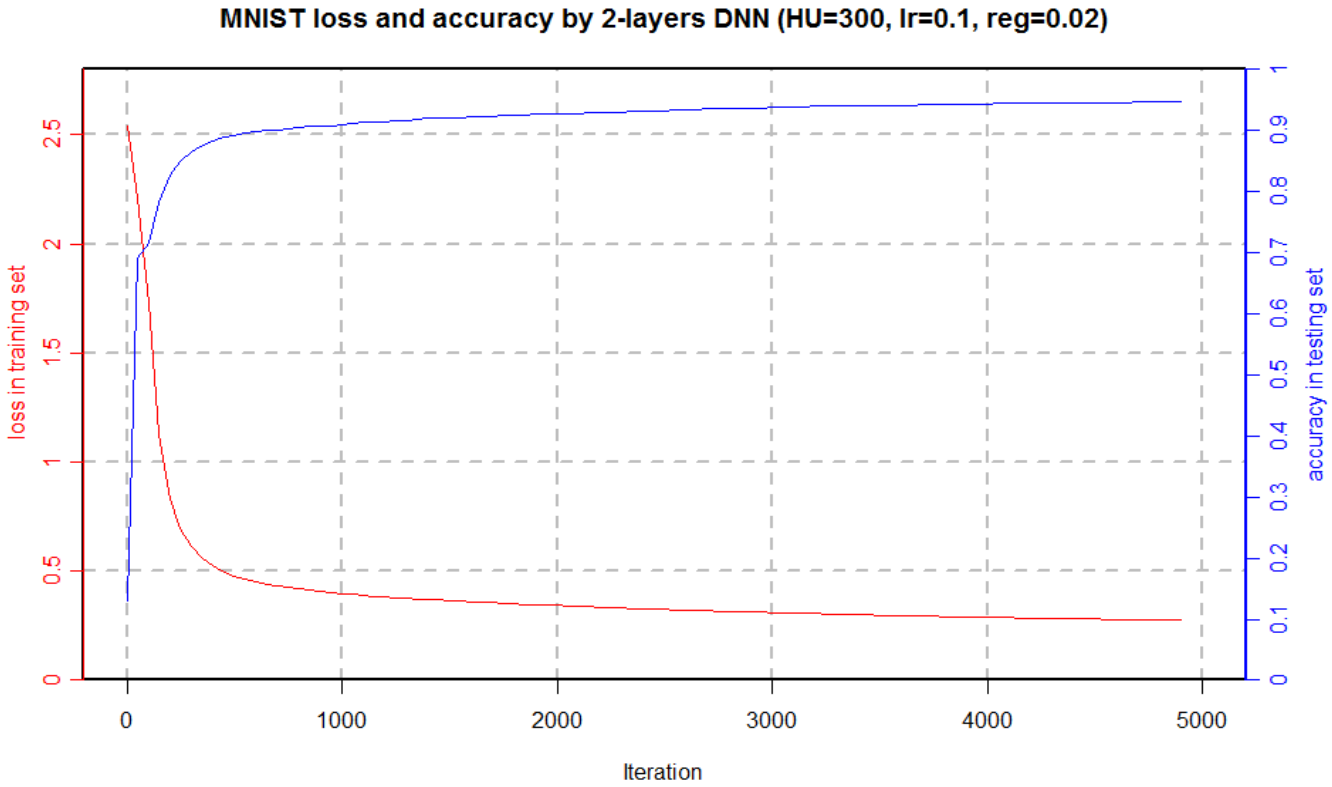
PERFORMANCE IMPROVEMENTS





CPU: Ivy Bridge E5-2690 v2 @ 3.00GHz, dual socket 10-core, 128G RAM; GPU: NVIDIA K40m, 12G RAM

TRAINING RESULTS



NOTE: Just to show the correctness of our codes and methods rather than achieve high accuracy of MNIST

SUMMARY

Related Materials:

CODES:

All codes, scripts and window templates in this talk in here

TALKS:

- GTC15: Accelerate R by CUDA, slide
- GTC16: Data Science Applications of GPUs in the R Language

BLOG:

- Parallel FORALL, post
- ParallelR, R For Deep Learning:

(I) Build Fully Connected Neural Network From Scratch

(II) Achieve High-Performance DNN With Parallel Acceleration

(III) CUDA Acceleration And MultiGPUs Training

R – RPUD DEEP LEARNING TUTORIAL

RPUD INSTALLATION

http://www.r-tutor.com/gpu-computing
http://www.r-tutor.com/content/download
http://www.r-tutor.com/gpu-computing/rpud-installation

RPUD DEEP LEARNING TUTORIAL

http://www.r-tutor.com/deep-learning/introduction

NVIDIA DEEP LEARNING PLATFORM

NVIDIA

POWERING THE DEEP LEARNING ECOSYSTEM

NVIDIA SDK accelerates every major framework

COMPUTER VISION

OBJECT DETECTION

IMAGE CLASSIFICATION

SPEECH & AUDIO

VOICE RECOGNITION

LANGUAGE TRANSLATION

NATURAL LANGUAGE PROCESSING

RECOMMENDATION ENGINES

SENTIMENT ANALYSIS

DEEP LEARNING FRAMEWORKS

Caffe
Chainer
DL4J
Deeplearning4j
Mocha.jl
julia
K
KERAS
Microsoft
CNTK
mxnet
Purine
TensorFlow
torch
MatConvNet
MINERVA
OpenDeep
Pylearn2
theano

NVIDIA DEEP LEARNING SDK

cuDNN

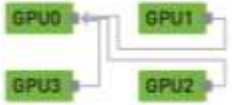
TensorRT

DeepStream SDK

cuBLAS

cuSPARSE

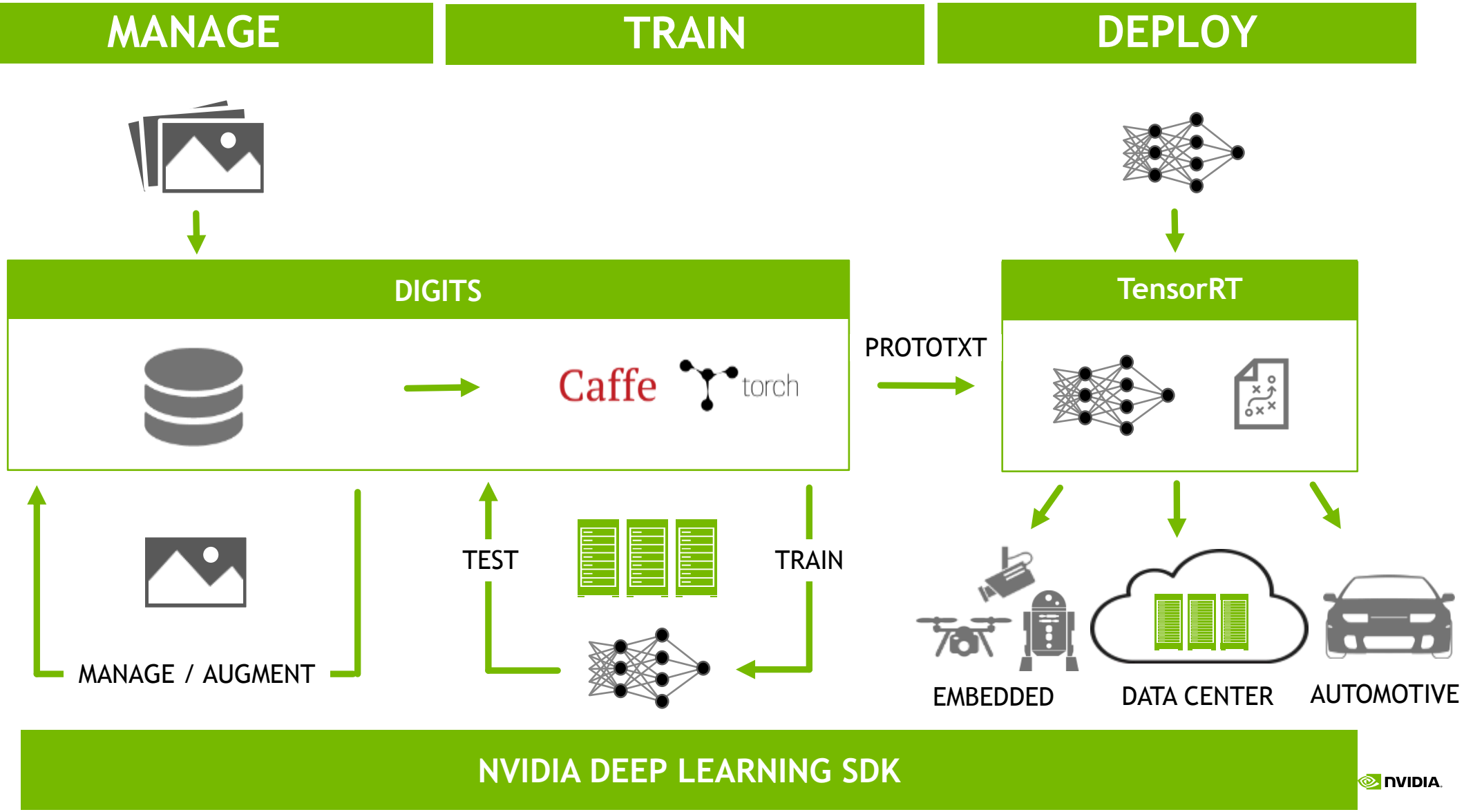
NCCL



developer.nvidia.com/deep-learning-software

A COMPLETE DEEP LEARNING PLATFORM

NVIDIA



GPU TECHNOLOGY CONFERENCE

NVIDIA cuDNN

Accelerating Deep Learning

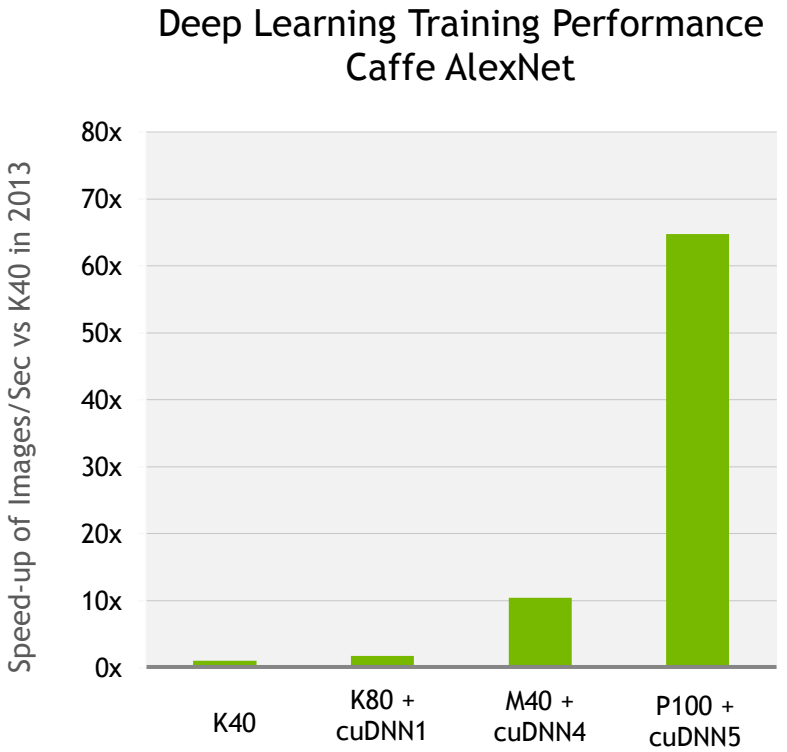
High performance building blocks for deep learning frameworks

Drop-in acceleration for widely used deep learning frameworks such as Caffe, CNTK, Tensorflow, Theano, Torch and others

Accelerates industry vetted deep learning algorithms, such as convolutions, LSTM, fully connected, and pooling layers

Fast deep learning training performance tuned for NVIDIA GPUs

developer.nvidia.com/cudnn



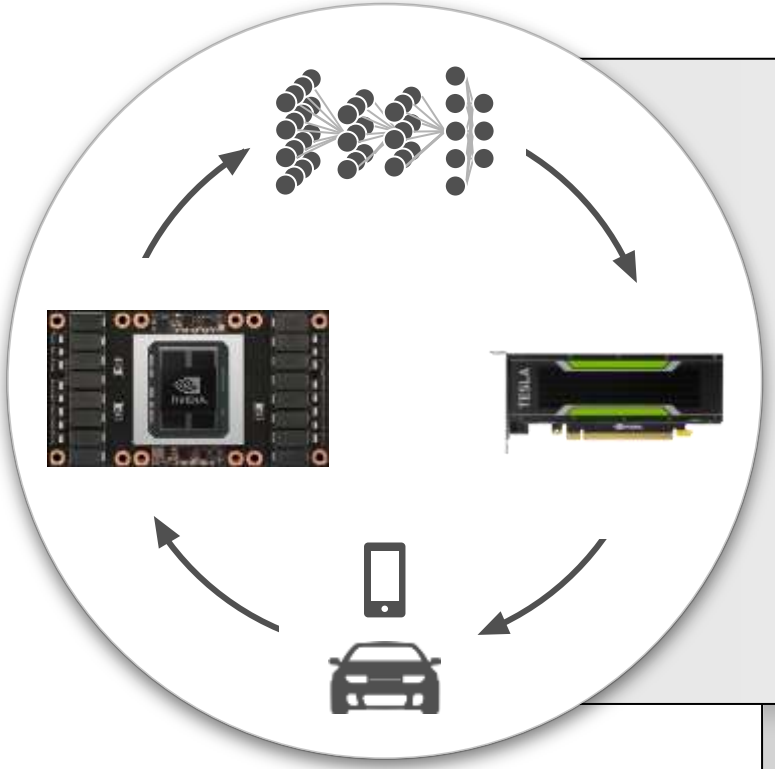
AlexNet training throughput on CPU: 1x E5-2680v3 12 Core 2.5GHz. 128GB System Memory, Ubuntu 14.04
M40 bar: 8x M40 GPUs in a node, P100: 8x P100 NVLink-enabled

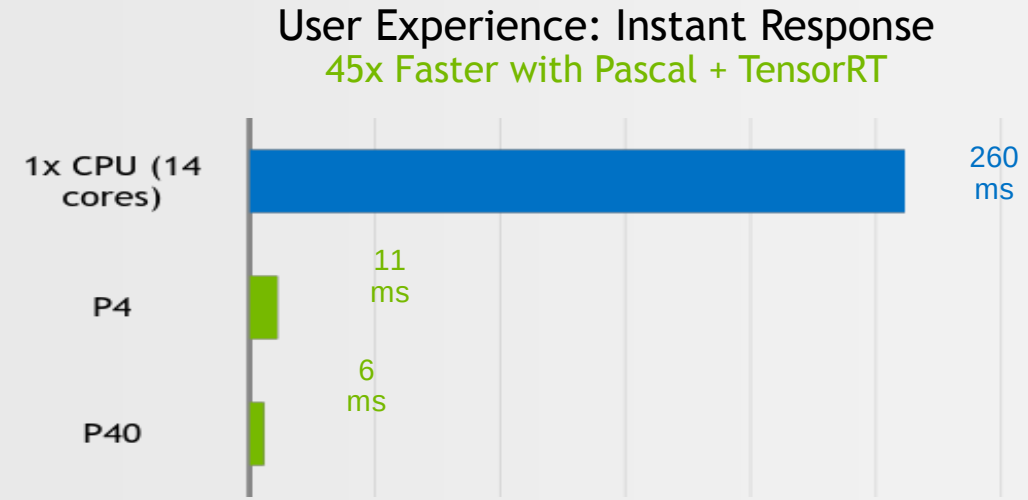
"NVIDIA has improved the speed of cuDNN with each release while extending the interface to more operations and devices at the same time."

— Evan Shelhamer, Lead Caffe Developer, UC Berkeley

INTRODUCING NVIDIA TensorRT

High Performance Inference Engine





Faster, more responsive AI-powered services such as voice recognition, speech translation

Efficient inference on images, video, & other data in hyperscale production data centers

Based on VGG-19 from IntelCaffe Github: https://github.com/intel/caffe/tree/master/models/mkl2017_vgg_19
CPU: IntelCaffe, batch size = 4, Intel E5-2690v4, using Intel MKL 2017 | GPU: Caffe, batch size = 4, using TensorRT internal version

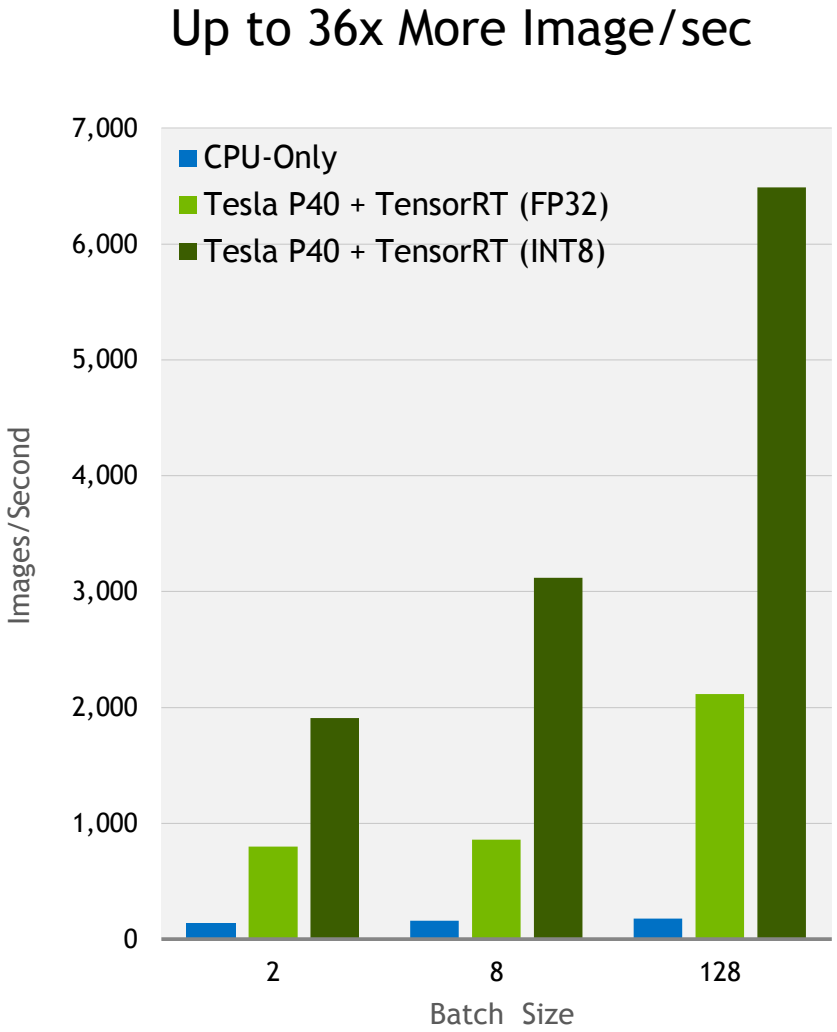
NVIDIA TensorRT

High-performance deep learning inference for production deployment

High performance neural network inference engine for production deployment

- Generate optimized and deployment-ready models for datacenter, embedded and automotive platforms
- Deliver high-performance, low-latency inference demanded by real-time services
- Deploy faster, more responsive and memory efficient deep learning applications with INT8 and FP16 optimized precision support

developer.nvidia.com/tensorrt



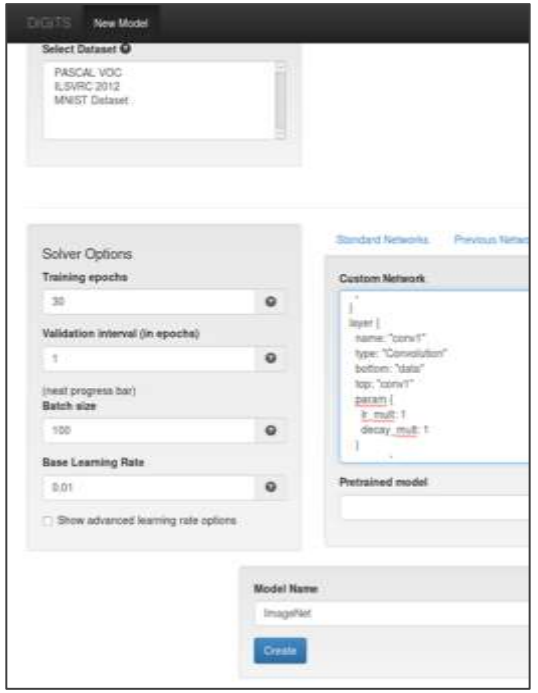
GoogLenet, CPU-only vs Tesla P40 + TensorRT
CPU: 1 socket E4 2690 v4 @2.6 GHz, HT-on
GPU: 2 socket E5-2698 v3 @2.3 GHz, HT off, 1 P40 card in the box

NVIDIA DIGITS

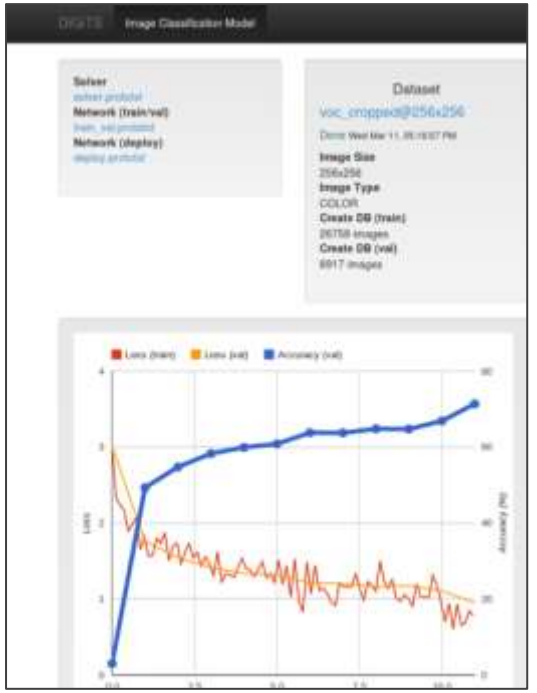
Interactive Deep Learning GPU Training System

Process Data

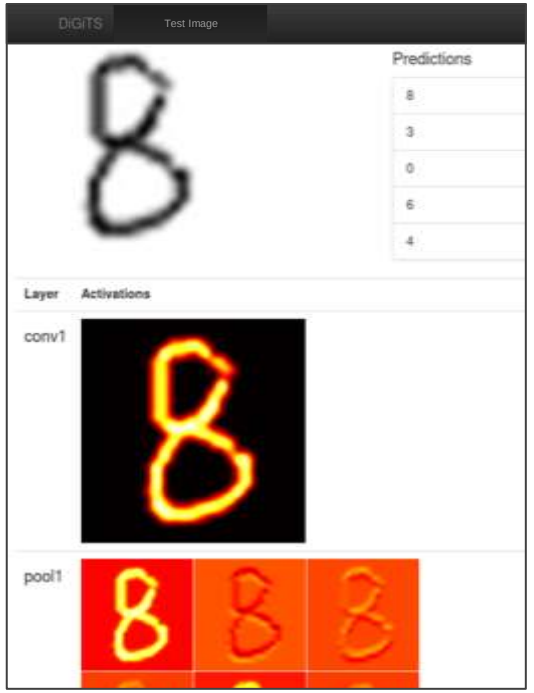
Configure DNN



Monitor Progress



Visualize Layers



developer.nvidia.com/digits

GitHub

NVIDIA

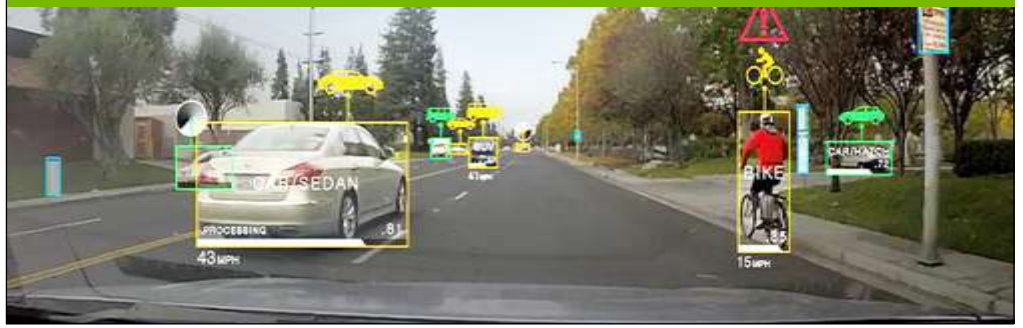
OBJECT DETECTION

New in DIGITS 4

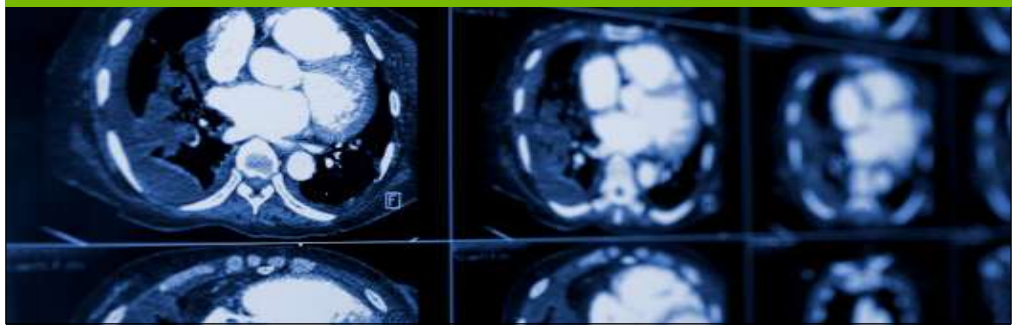
REMOTE SENSING

INTELLIGENT VIDEO ANALYTICS

ADVANCED DRIVER ASSISTANCE SYSTEMS (ADAS)

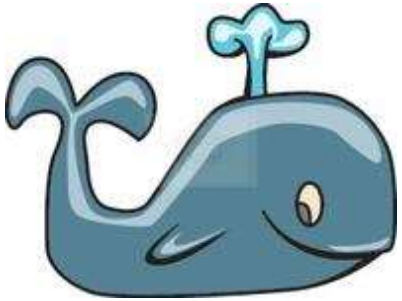


MEDICAL DIAGNOSTICS

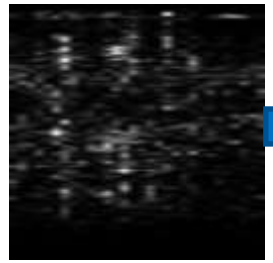


SONAR DATA

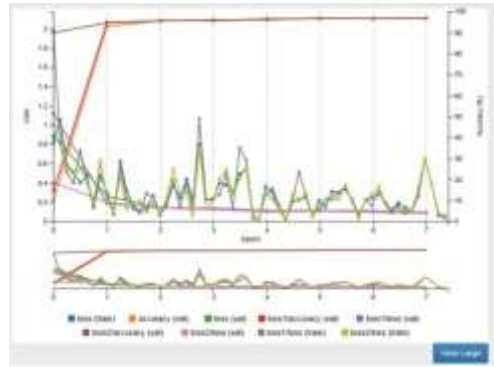
Classification



Convert to a spectrogram



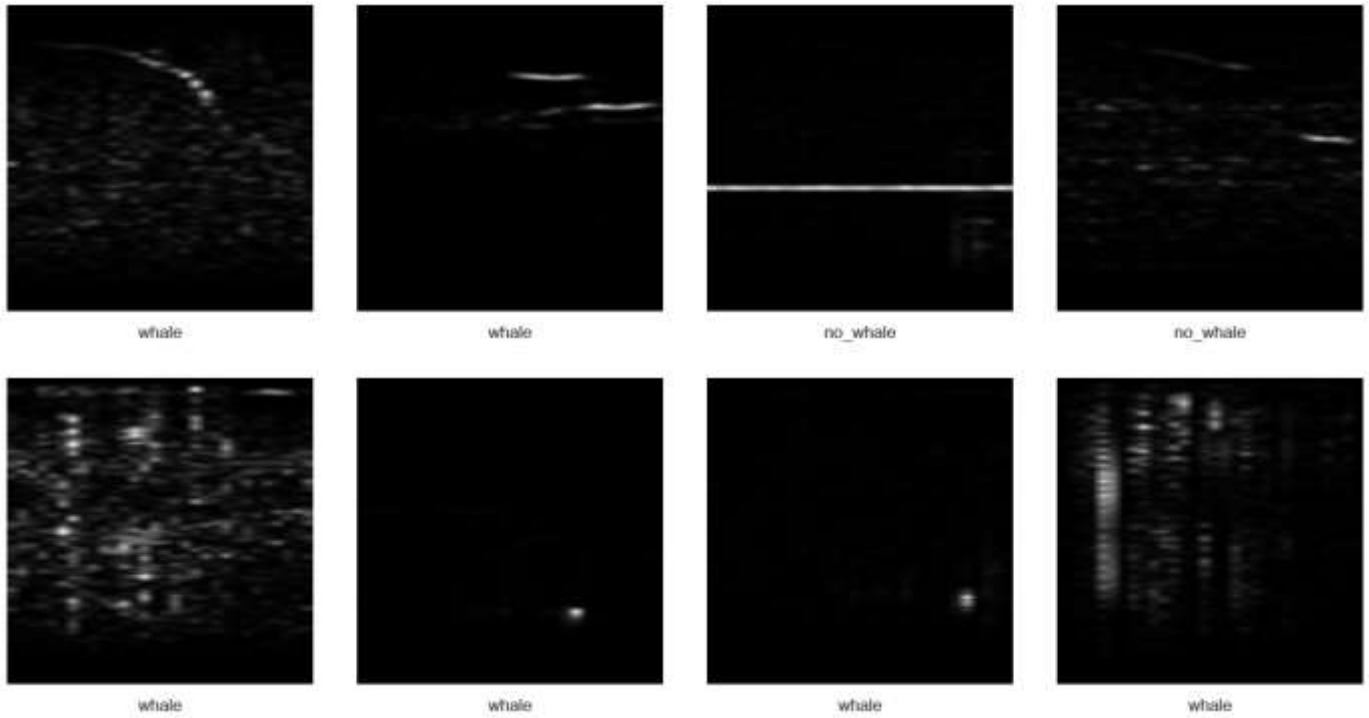
Whale or no whale



DIGITS training

Right Whale dataset, GoogLeNet network, https://www.kaggle.com/c/whale-detection-challenge/

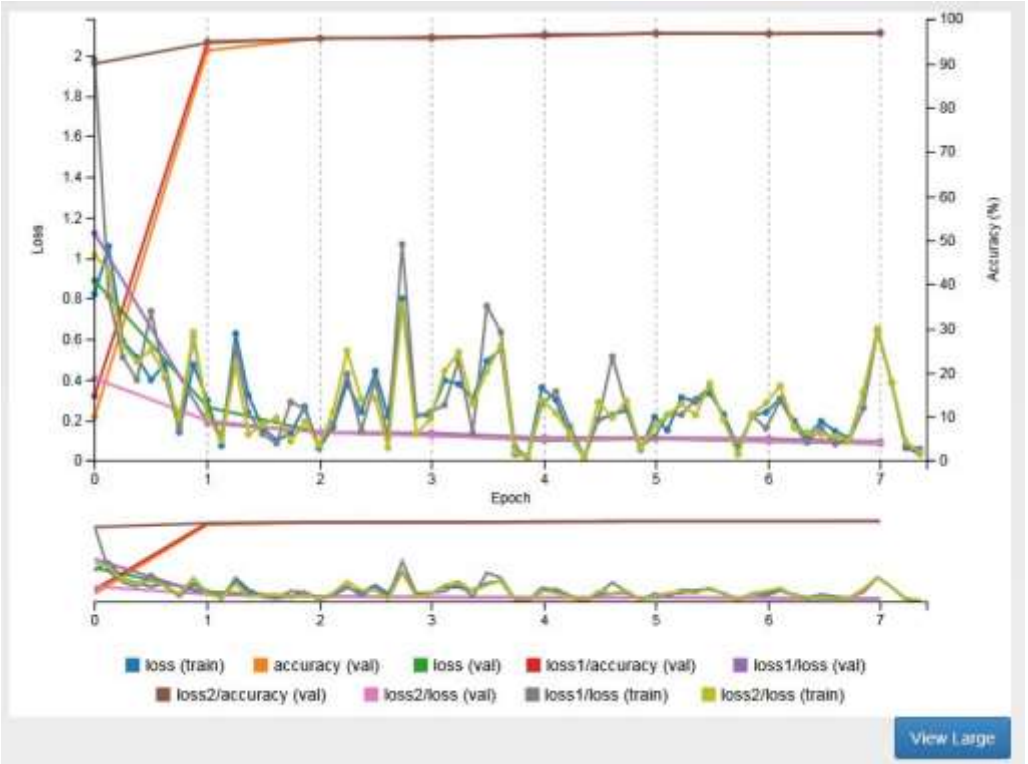
VISUALIZATION OF SOUNDS



Right Whale dataset, https://www.kaggle.com/c/whale-detection-challenge/

TRAINING WITH SONAR DATA

Classification with non-image data



Right Whale dataset, GoogLeNet network, https://www.kaggle.com/c/whale-detection-challenge/

NVIDIA

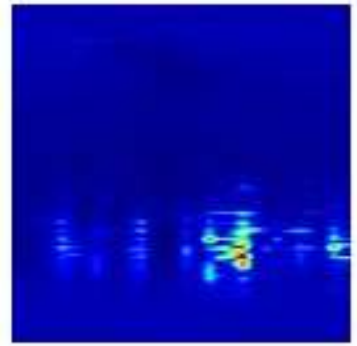
SONAR DATA

Auto Encoder

Unsupervised learning approach

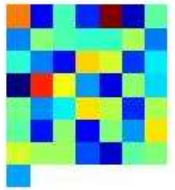
Compress input data

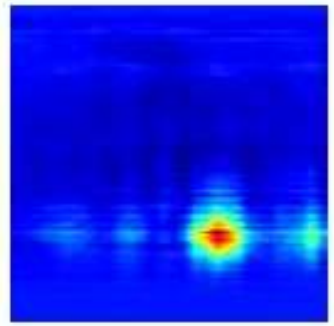
Similarity searching



Input image

Compressed vector

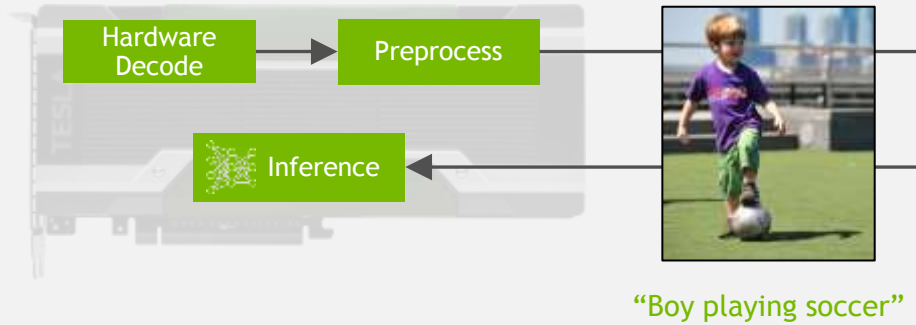




Regenerated input

NVIDIA DEEPSTREAM SDK

Delivering Video Analytics at Scale

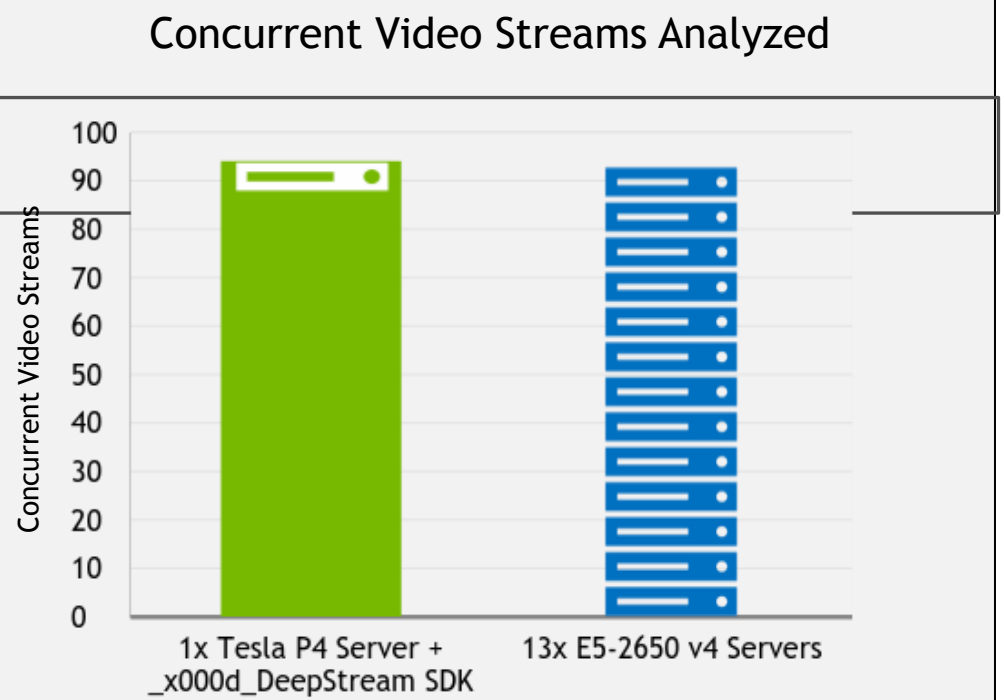


Simple, high performance API for analyzing video

Decode H.264, HEVC, MPEG-2, MPEG-4, VP9

CUDA-optimized resize and scale

TensorRT



720p30 decode | IntelCaffe using dual socket E5-2650 v4 CPU servers, Intel MKL 2017
Based on GoogLeNet optimized by Intel: https://github.com/intel/caffe/tree/master/models/mkl2017_googlenet_v2

NVIDIA

END-TO-END PRODUCT FAMILY

TRAINING

FULLY INTEGRATED DL SUPERCOMPUTER

DGX-1

DESKTOP

Titan X

HYPERSCALE

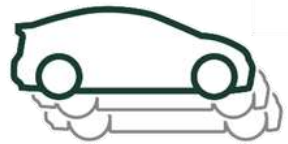
Tesla M40

INFERENCE

DATA CENTER

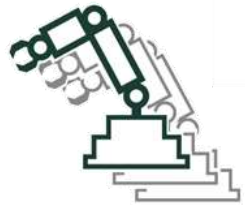
Tesla M4

AUTOMOTIVE



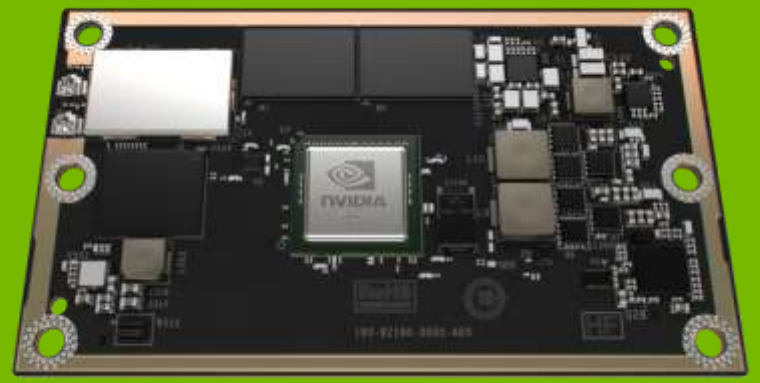
Drive PX

EMBEDDED



Jetson TX1

Jetson TX1



	JETSON TX1
GPU	1024 GFLOPS 256-core Maxwell
CPU	4x 64-bit ARM A57 CPUs | 1.6 GHz
Memory	4 GB LPDDR4 | 25.6 GB/s
Video decode	4K 60Hz H.264 / H.265
Video encode	4K 30Hz H.264 / H.265
CSI	Up to 6 cameras | 1400 Mpix/s
Display	2x DSI, 1x eDP 1.4, 1x DP 1.2/HDMI
Wi-Fi	802.11 2x2 ac
Networking	1 Gigabit Ethernet
PCI-E	Gen 2 1x1 + 1x4
Storage	16 GB eMMC, SDIO, SATA
Other	3x UART, 3x SPI, 4x I2C, 4x I2S, GPIOs
Power	10-15W, 6.6V-19.5VDC
Size	50mm x 87mm

NVIDIA

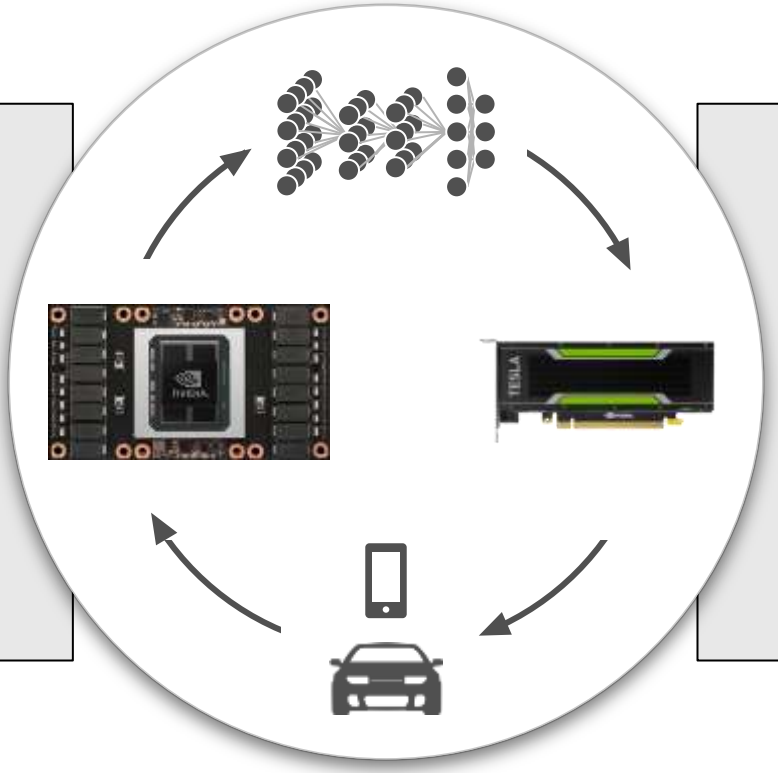
TESLA END-TO-END DEEP LEARNING

TRAINING

Tesla P100

65X

in 3 years



INFERENCING

Tesla P4

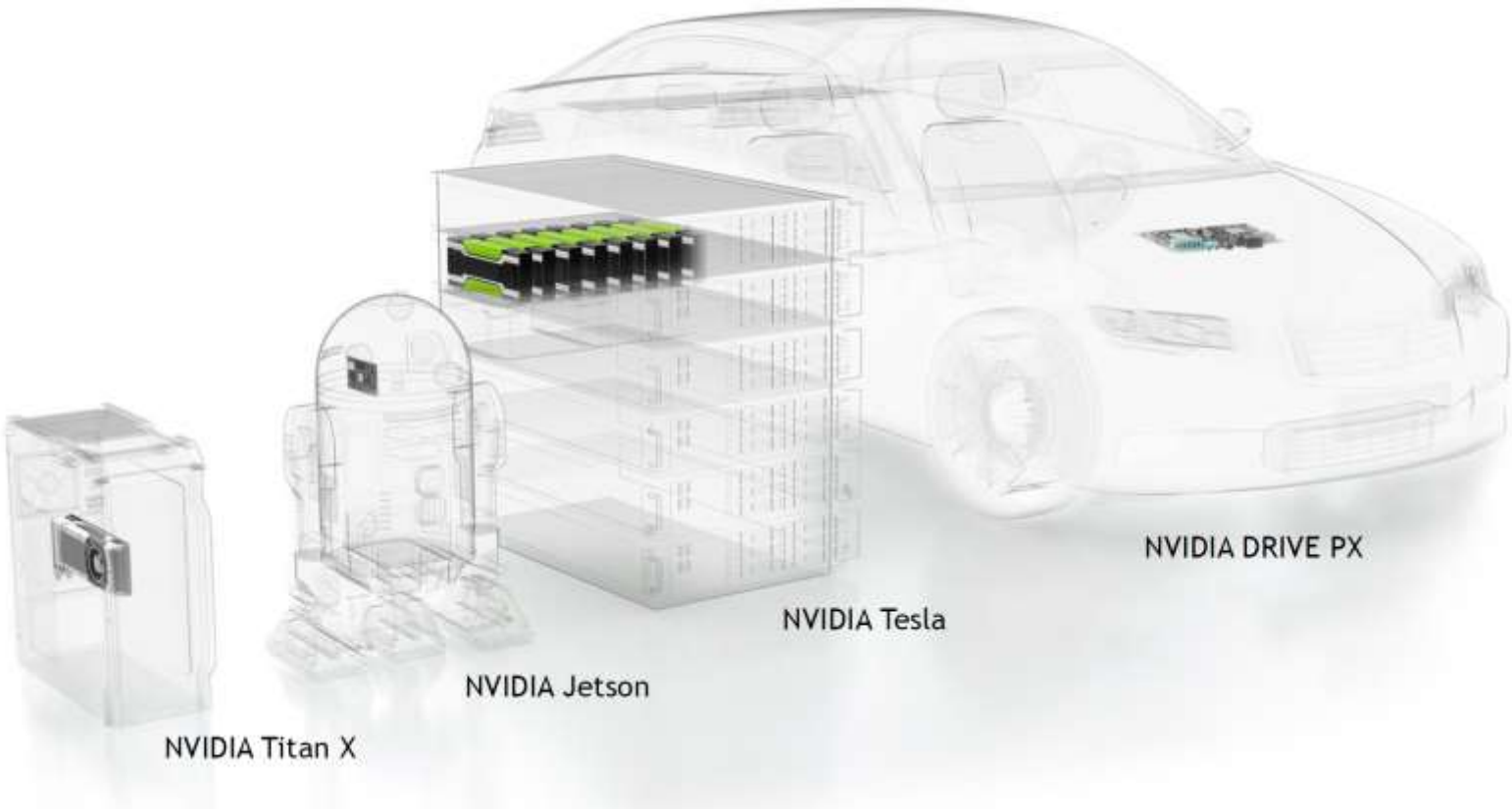
40X

vs CPU

Training: comparing to Kepler GPU in 2013 using Caffe, Inference: comparing img/sec/watt to CPU: Intel E5-2697v4 using AlexNet

NVIDIA

ONE ARCHITECTURE — END-TO-END AI



CUDA + Linux throughout the stack.

NVIDIA DEEP LEARNING INSTITUTE

Hands-on Training for Data Scientists and Software Engineers

Training organizations and individuals to solve challenging problems using Deep Learning

On-site workshops and online courses presented by certified experts

Covering complete workflows for proven application use cases

Self-driving cars, recommendation engines, medical image classification, intelligent video analytics and more

www.nvidia.com/dli

NVIDIA

CHALLENGES

Deep Learning Needs	NVIDIA Delivers
Data Scientists	DIGITS, DLI Training
Latest Algorithms	DL SDK-Accelerated Frameworks
Fast Training	DGX, P100, P40, TITAN X
Deployment Platform	TensorRT, P4, Jetson, Drive PX2

NVIDIA

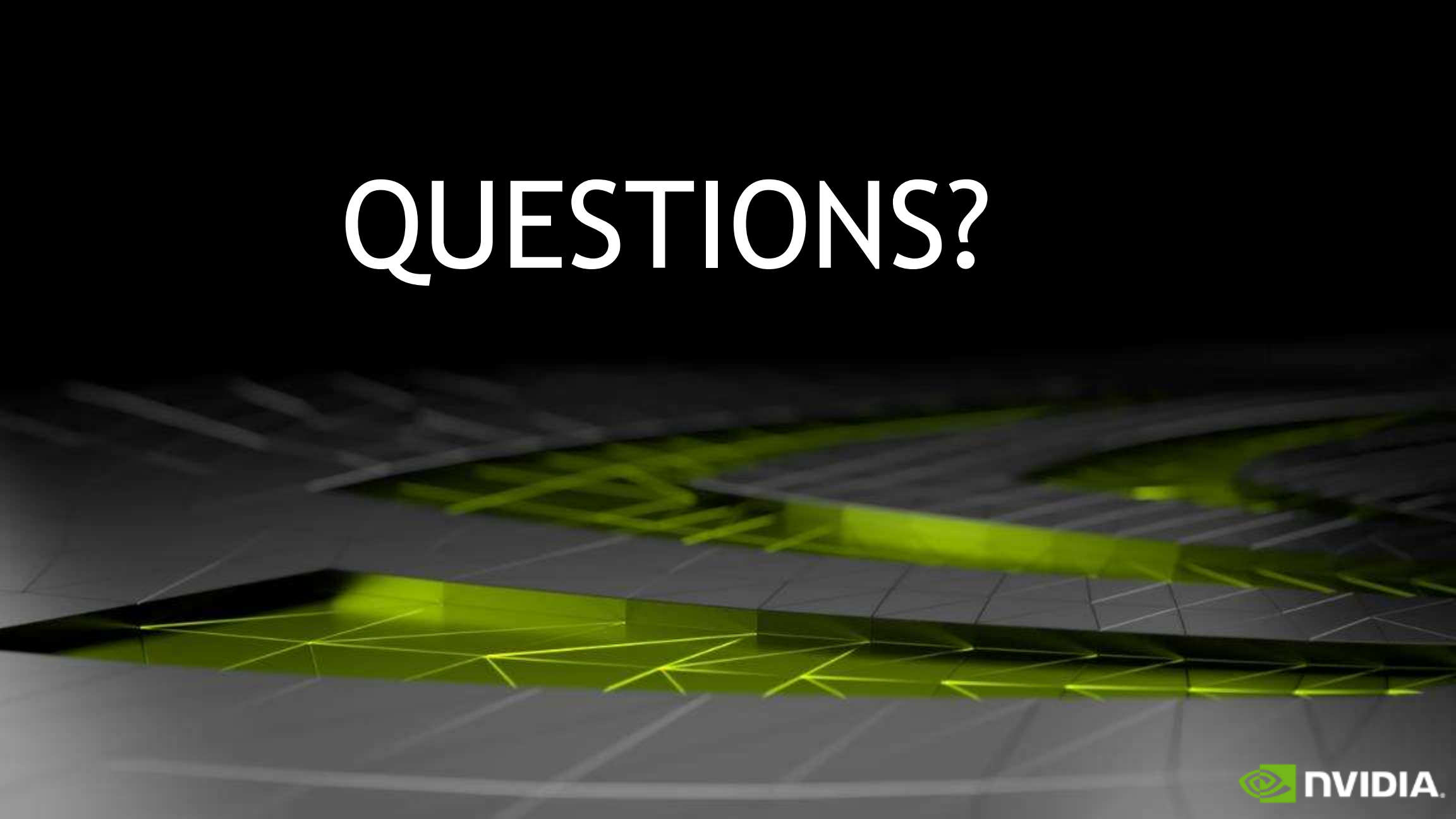
ACKNOWLEDGEMENTS

Patrick Zhao, NVIDIA

Will Ramey, NVIDIA

Joel Emer, MIT, NVIDIA/Vivienne Sze,MIT/Yu-Hsin Chen, MIT (http://eyeriss.mit.edu/tutorial.html)

Stephane Mallat ENS Paris, France

Questions: fcourteille@nvidia.com

QUESTIONS?