Deep Learning & HPC

Valeriu Codreanu
Damian Podareanu
31 October 2016

SURF SARA

About SURFsara

History:

- 1971: Founded by the VU, UvA, and CWI
- 2013: SARA (Stichting Academisch Rekencentrum A'dam) becomes part of SURF

Super/cluster-computing group:

- 8 consultants
- 17 members in total (including admins/system-experts)

Other activities:

- HPC cloud / virtualisation
- Data services / storage
- Visualisation

Our systems (1/2)

Cartesius (Bull supercomputer):

- 40.960 Ivy Bridge / Haswell cores: 1327 TFLOPS
- 56GBit/s Infiniband
- 64 nodes with 2 GPUs each: 210 TFLOPS
  - NVIDIA Tesla K40m GPU
  - 12GB GDDR5
  - GPU-Direct RDMA
- Accelerator island: #4 Green500 (June 2014)
- Broadwell & KNL extension (Nov 2016)
  - 177 BDW and 18 KNL nodes: 284TFLOPS
- 7.7 PB Lustre parallel file-system

THE GREEN 500™

Our systems (2/2)

LISA (Dell cluster):

- 7856 cores (16 cores per node, Xeon E5-2650)
- Peak performance: 149 TFLOPS

HPC cloud:

- Virtual machines
- Up to 64 cores and 2TB RAM

The archive:

- Tape-storage for long-term storage
- Virtually unlimited space

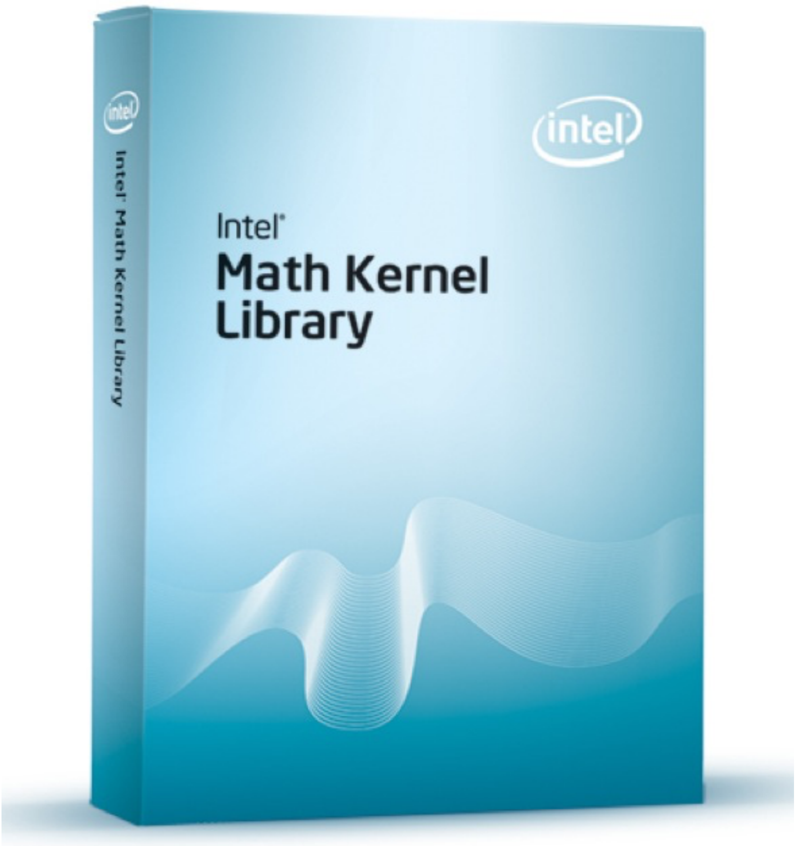
Others:

- Elvis (visualisation/render cluster with 18 GPUs)
- Grid
- Hadoop

Challenges

Typical challenges:

- Bottleneck identification
- MPI/OpenMP parallelisation
- Inter-node communication
- I/O scaling
- GPU/Xeon Phi acceleration
- Algorithm optimization
- Vectorization



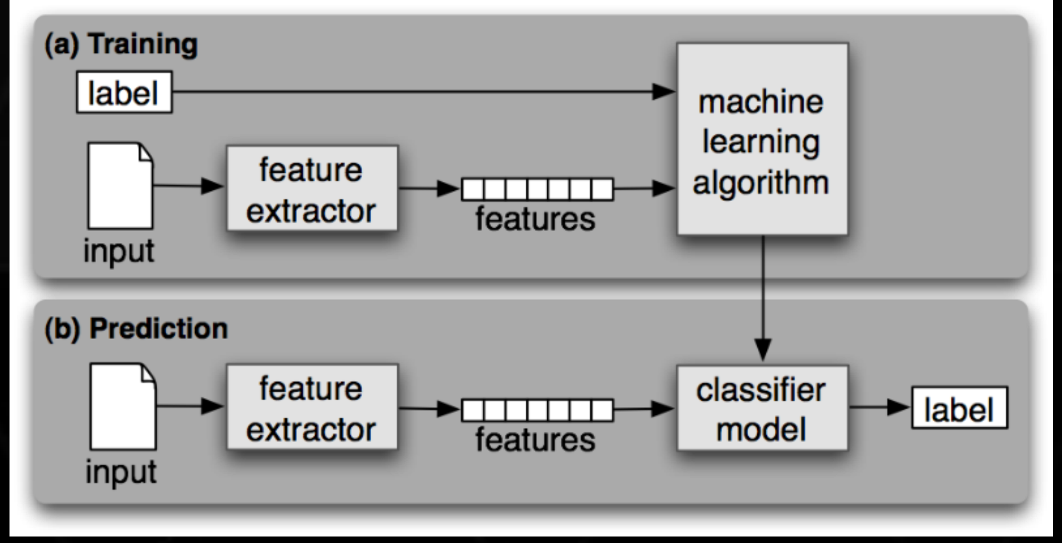
Approach:

- Discussions, scientific papers, manuals
- Hot-spot detection, timing analysis (manual)
- V-tune / Scalasca / score-p / Likwid / nvvp profiling (guided)

Deep Learning & HPC

TRADITIONAL ML – HAND TUNED FEATURES

Images/video

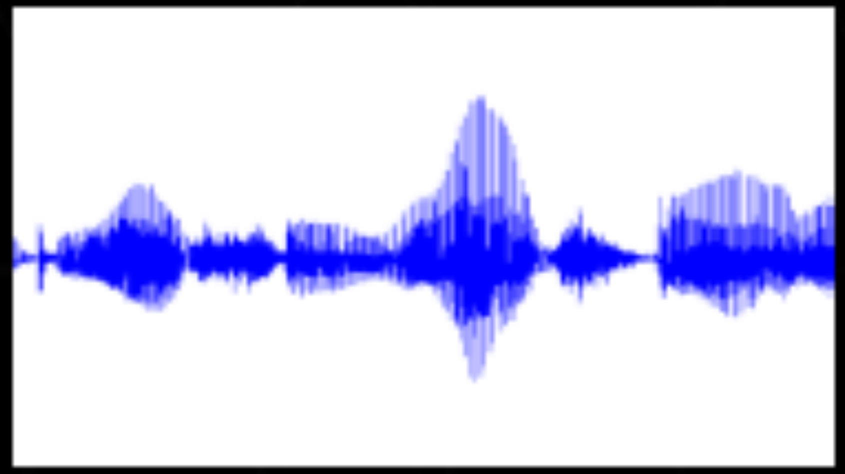


Image

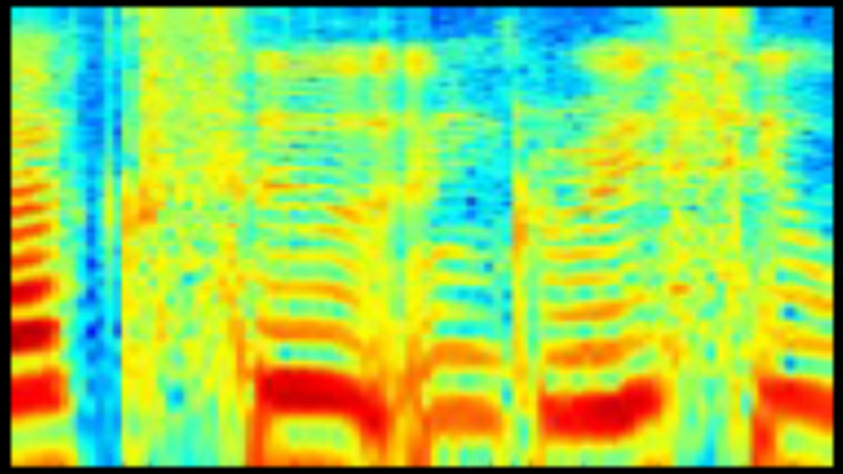
Vision features

Detection

Audio



Audio

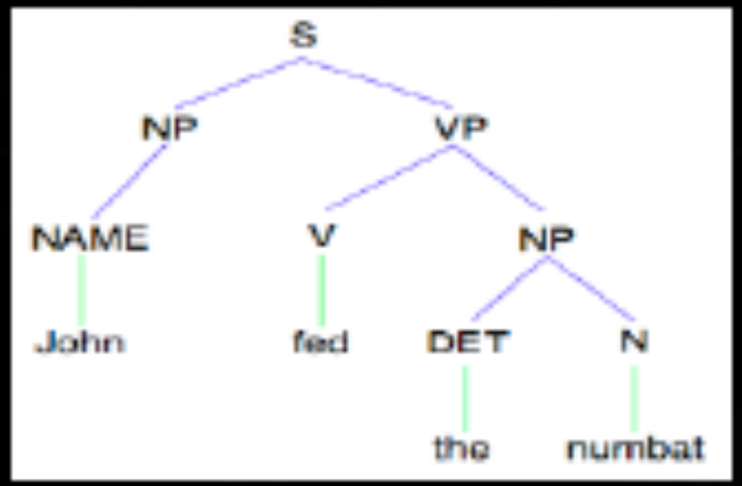


Audio features

Speaker ID

Text

Text

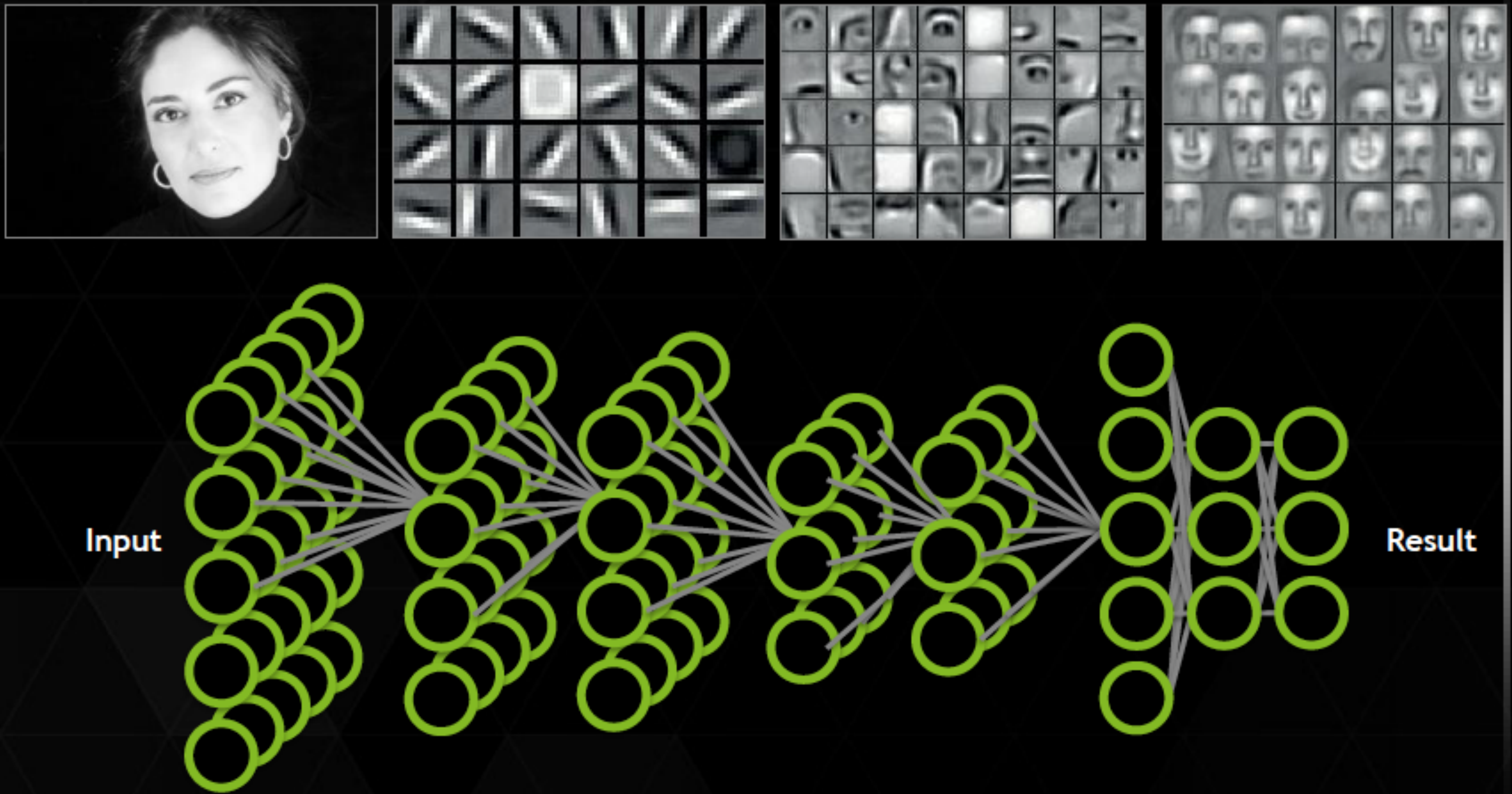


Text features

Text classification, Machine translation, Information retrieval, ...

Slide courtesy of Andrew Ng, Stanford University

WHAT MAKES DEEP LEARNING DEEP?



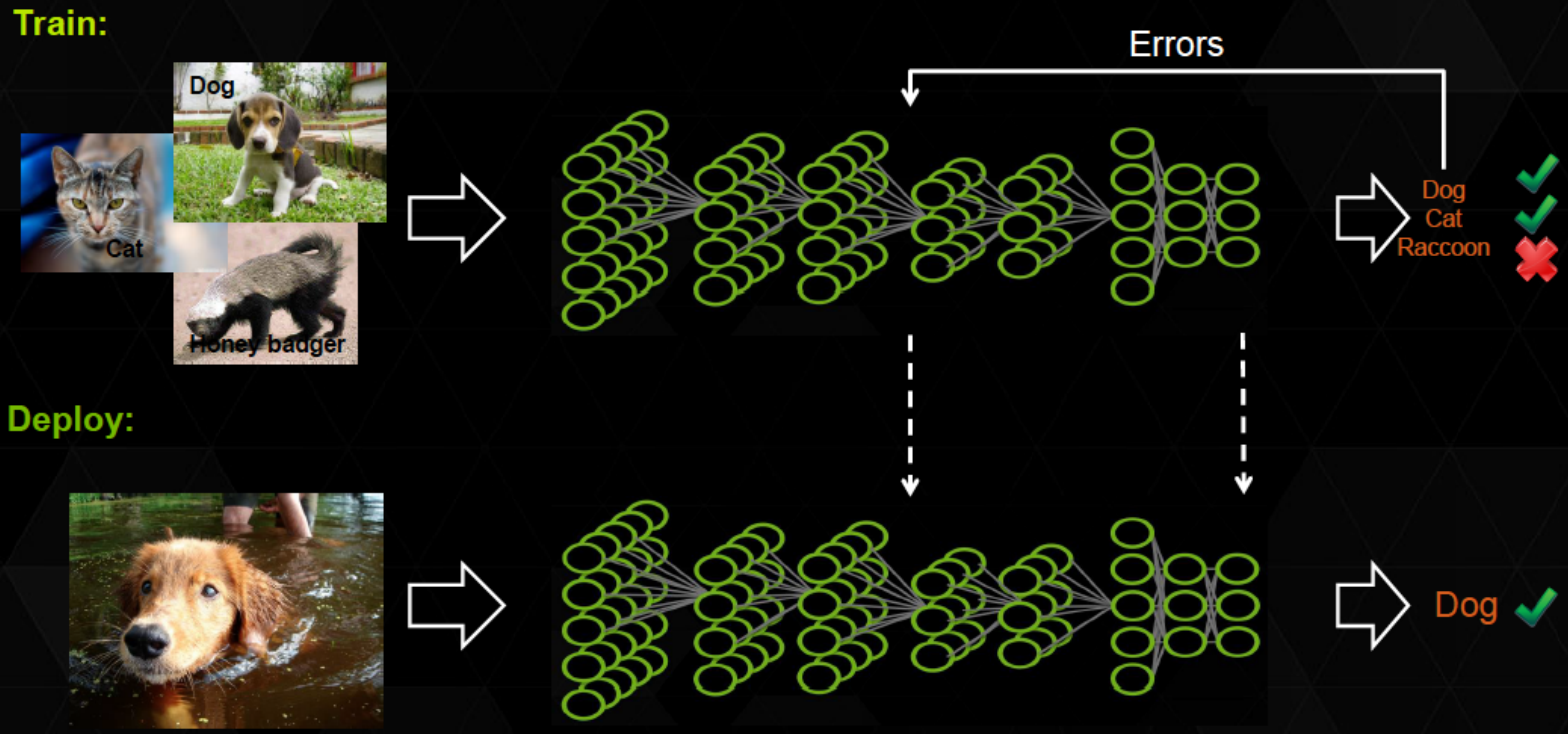
Convolutional Neural Networks are biologically-inspired

The network learns from raw pixels

Each layer learns incrementally complex features

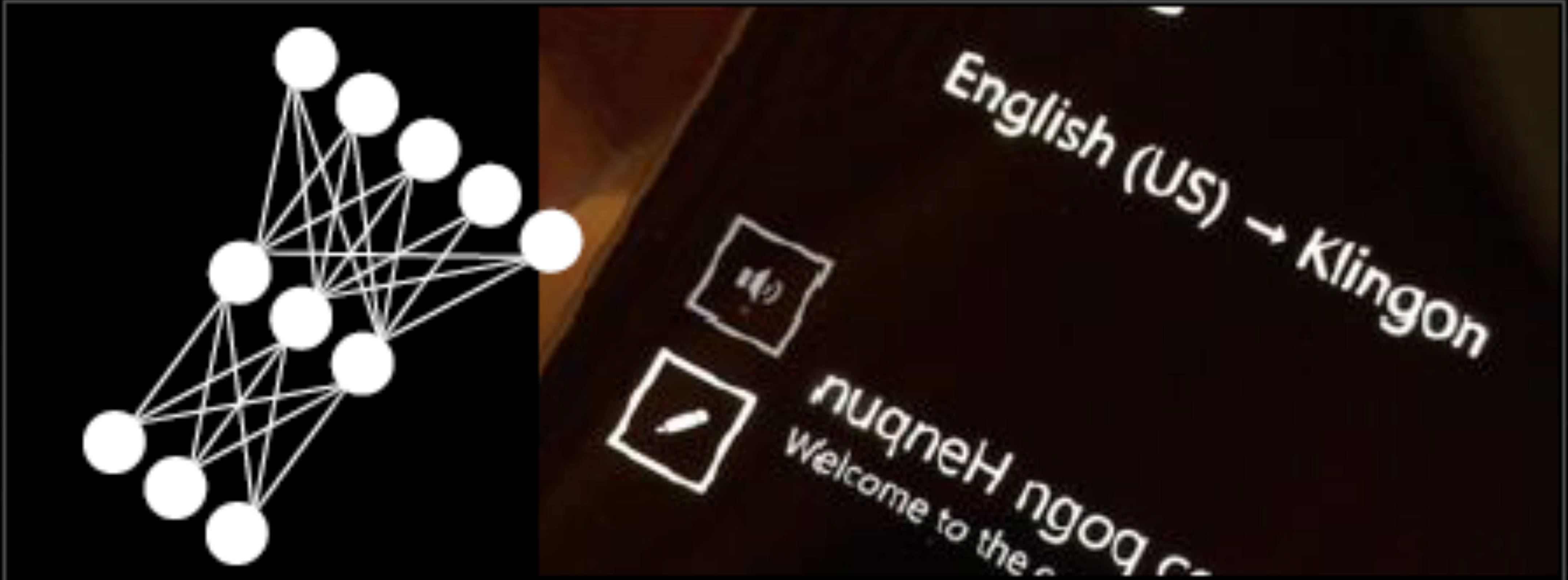
Typically implemented as a series of convolution and max-pooling layers

DEEP LEARNING APPROACH



DEEP LEARNING EXAMPLES

Image Classification, Object Detection, Localization,
Action Recognition, Scene Understanding



Speech Recognition, Speech Translation,
Natural Language Processing

Pedestrian Detection, Traffic Sign Recognition

Breast Cancer Cell Mitosis Detection,
Volumetric Brain Image Segmentation

WHY IS DEEP LEARNING HOT *NOW*?

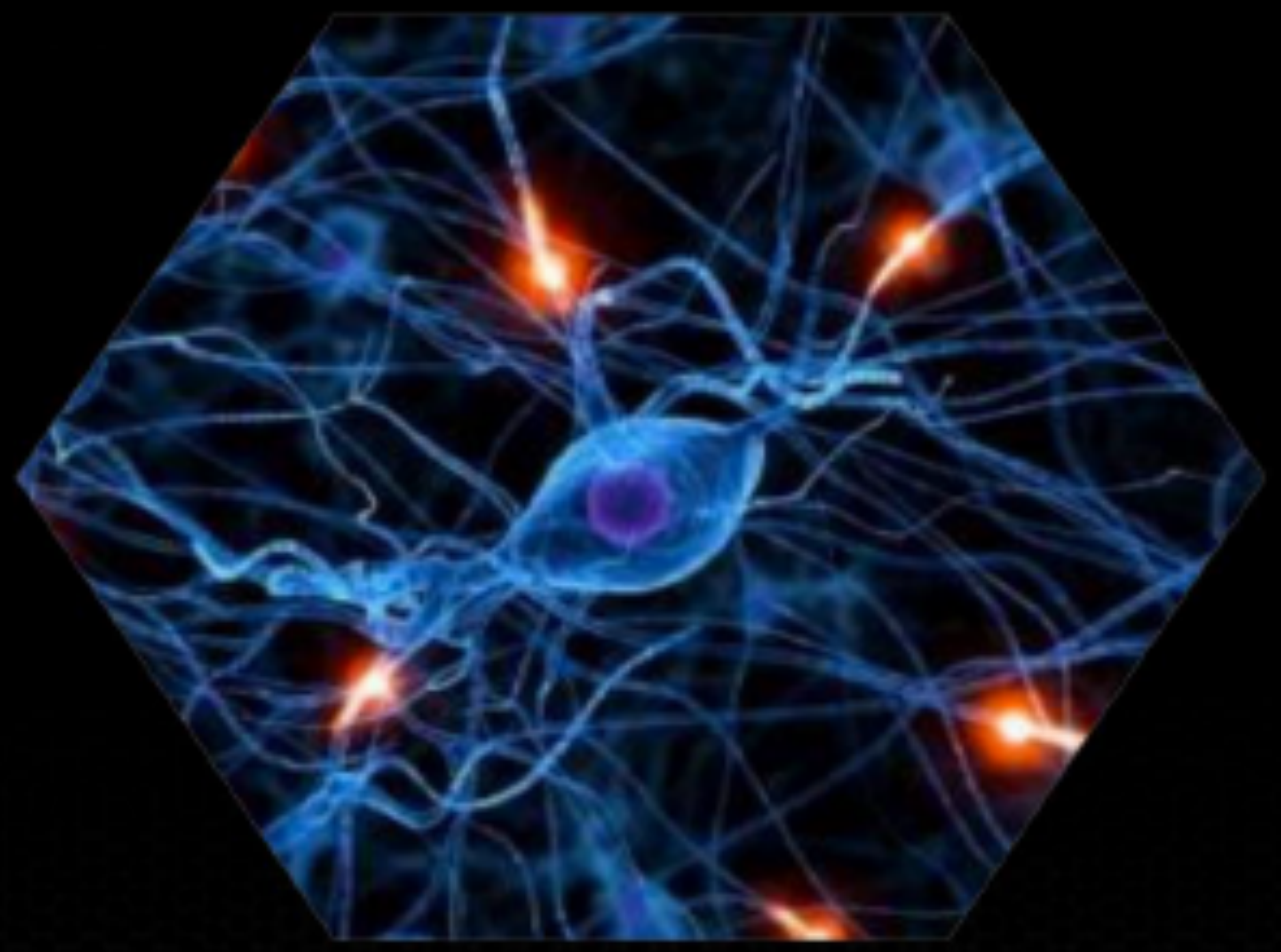
Three Driving Factors…

Big Data Availability	New DL Techniques	GPU acceleration

facebook | 350 millions images uploaded per day

Walmart | 2.5 Petabytes of customer data hourly

YouTube | 100 hours of video uploaded every minute



Deep Learning & GPUs

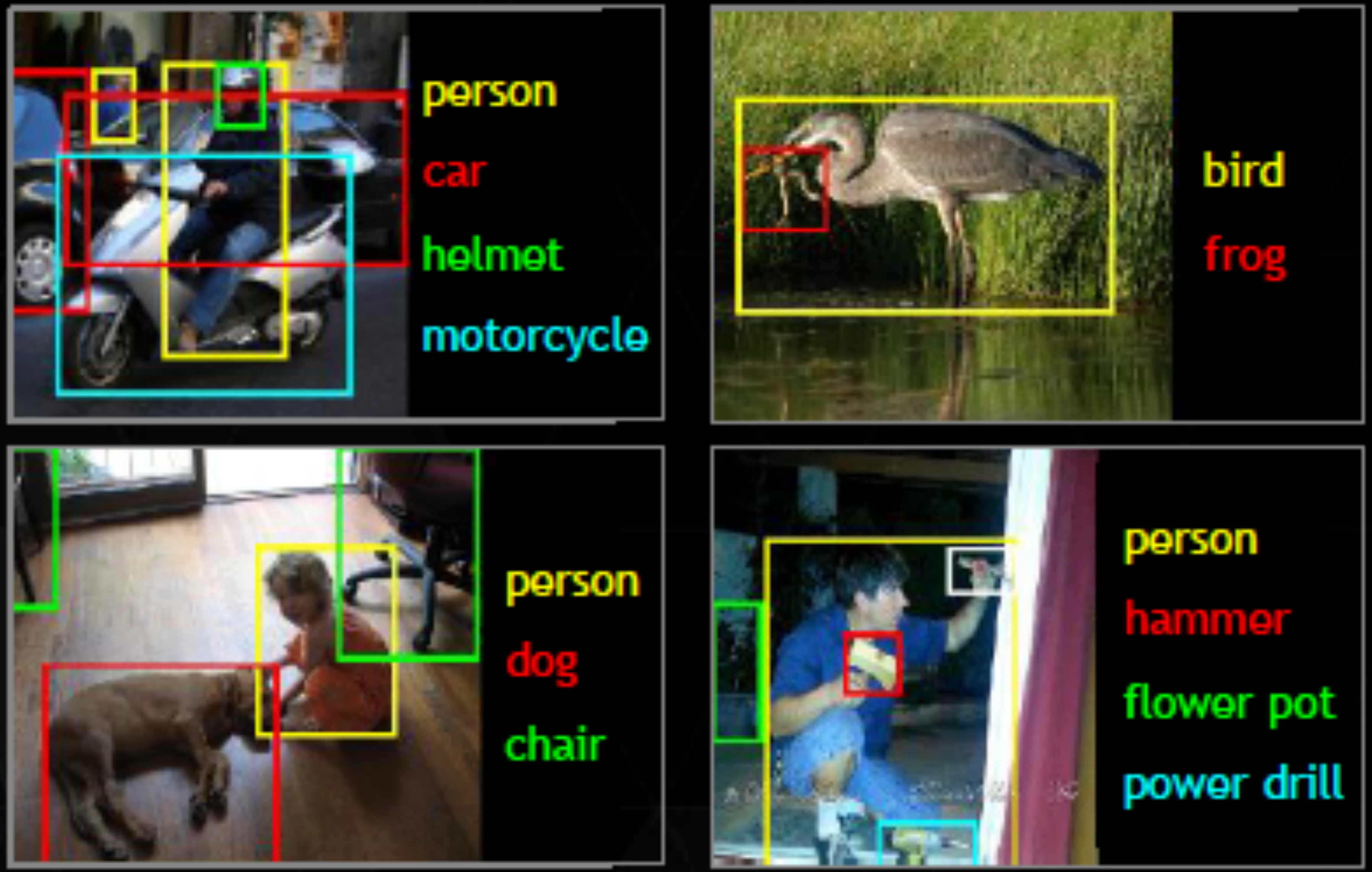
GPUs — THE PLATFORM FOR DEEP LEARNING

Image Recognition Challenge

1.2M training images • 1000 object categories

Hosted by

IMAGENET



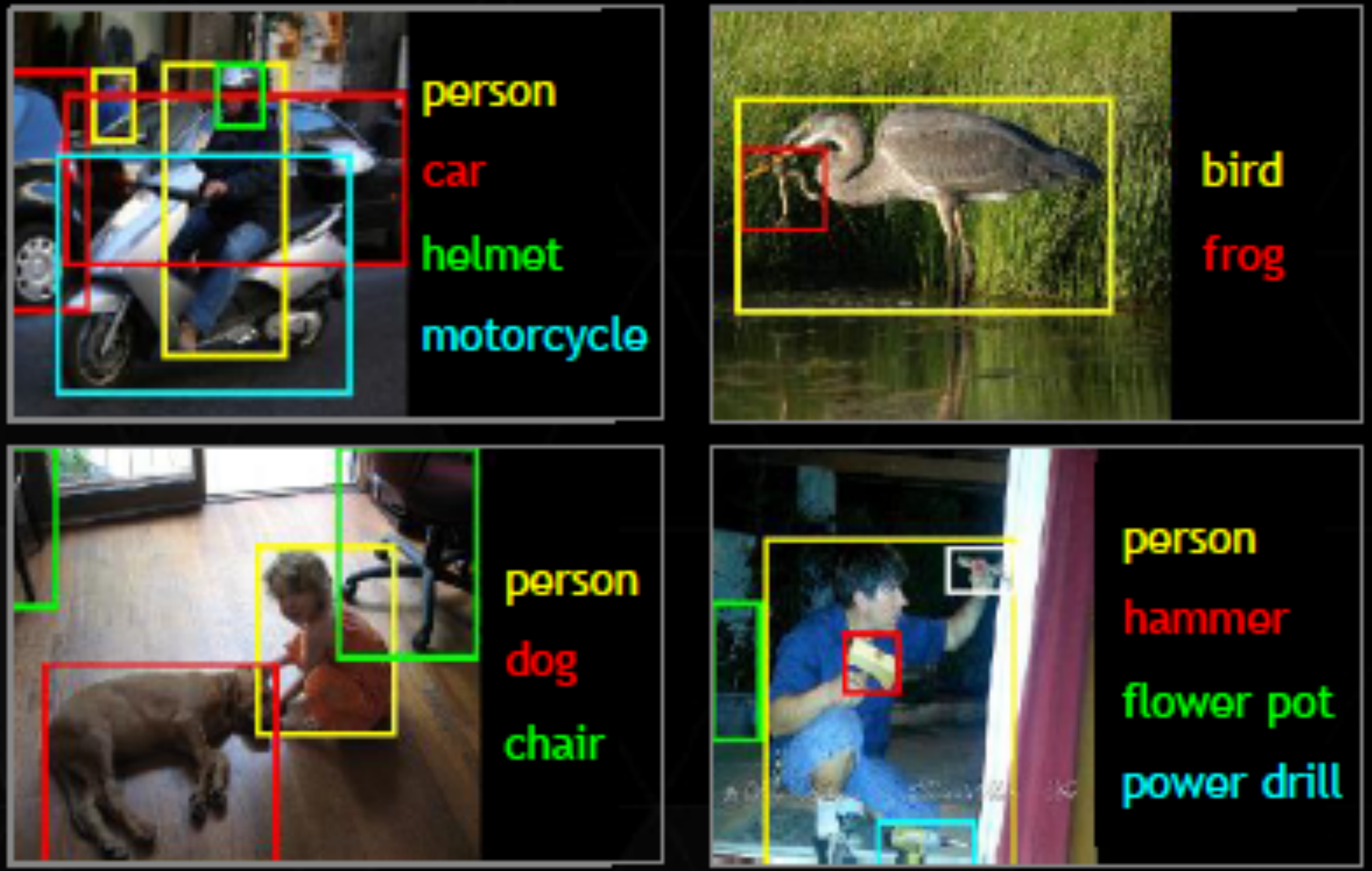
GPUs — THE PLATFORM FOR DEEP LEARNING

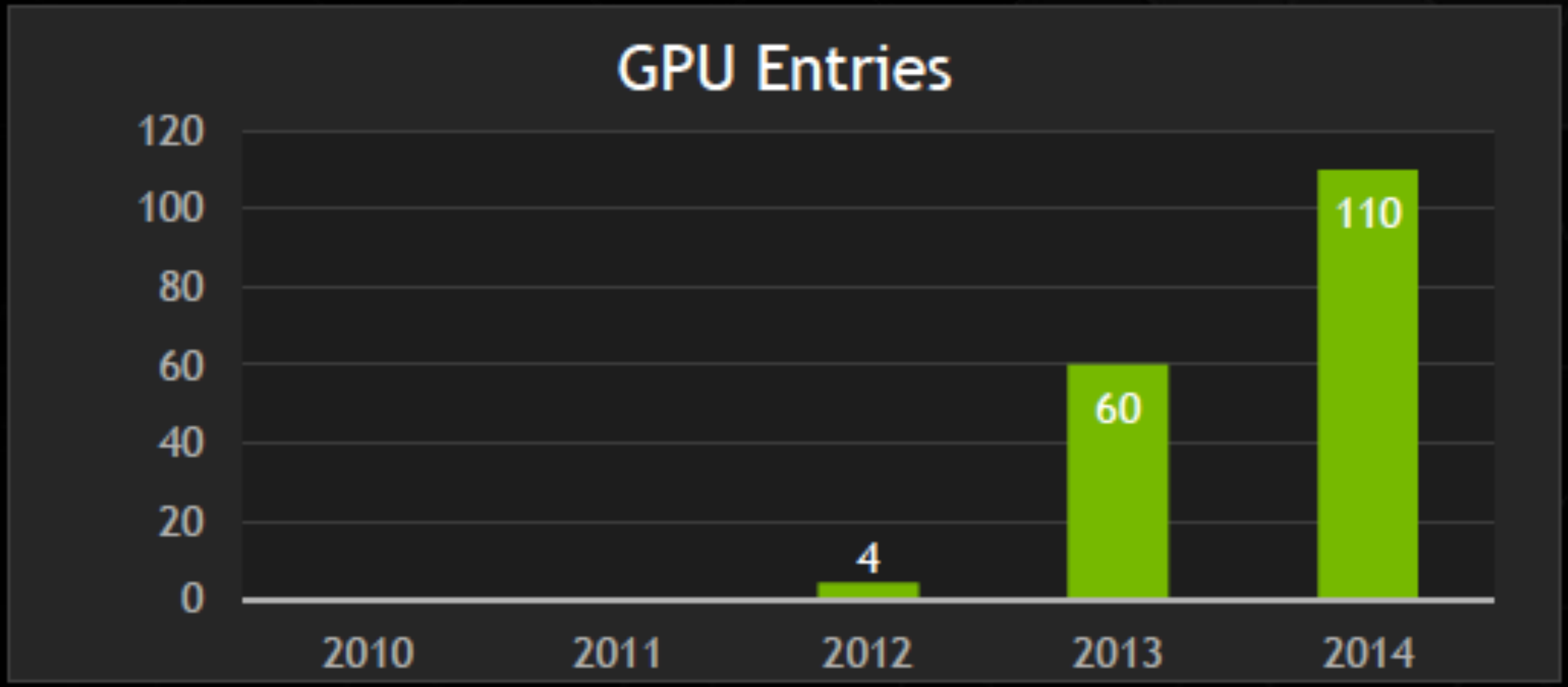
Image Recognition Challenge

1.2M training images • 1000 object categories

Hosted by

IMAGENET





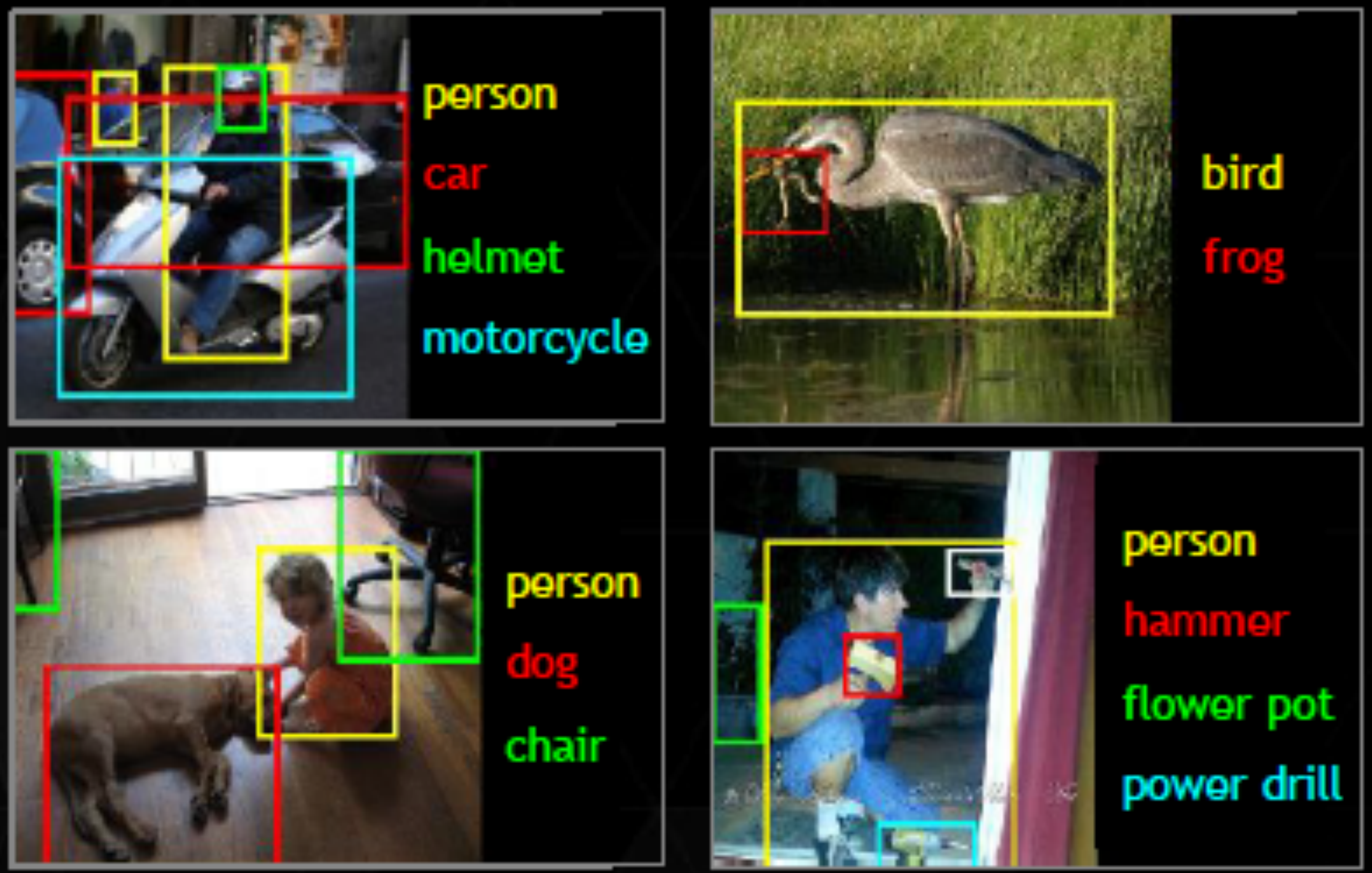
GPUs — THE PLATFORM FOR DEEP LEARNING

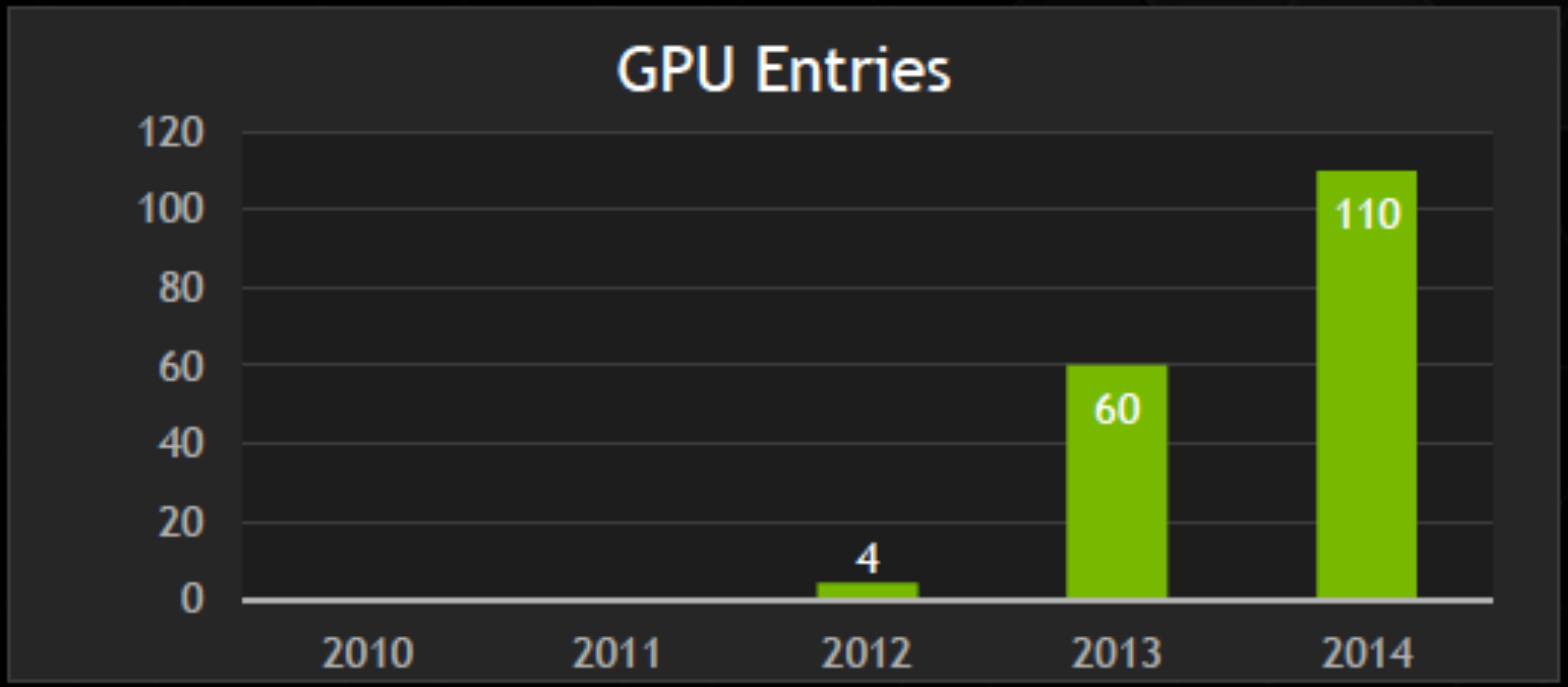
Image Recognition Challenge

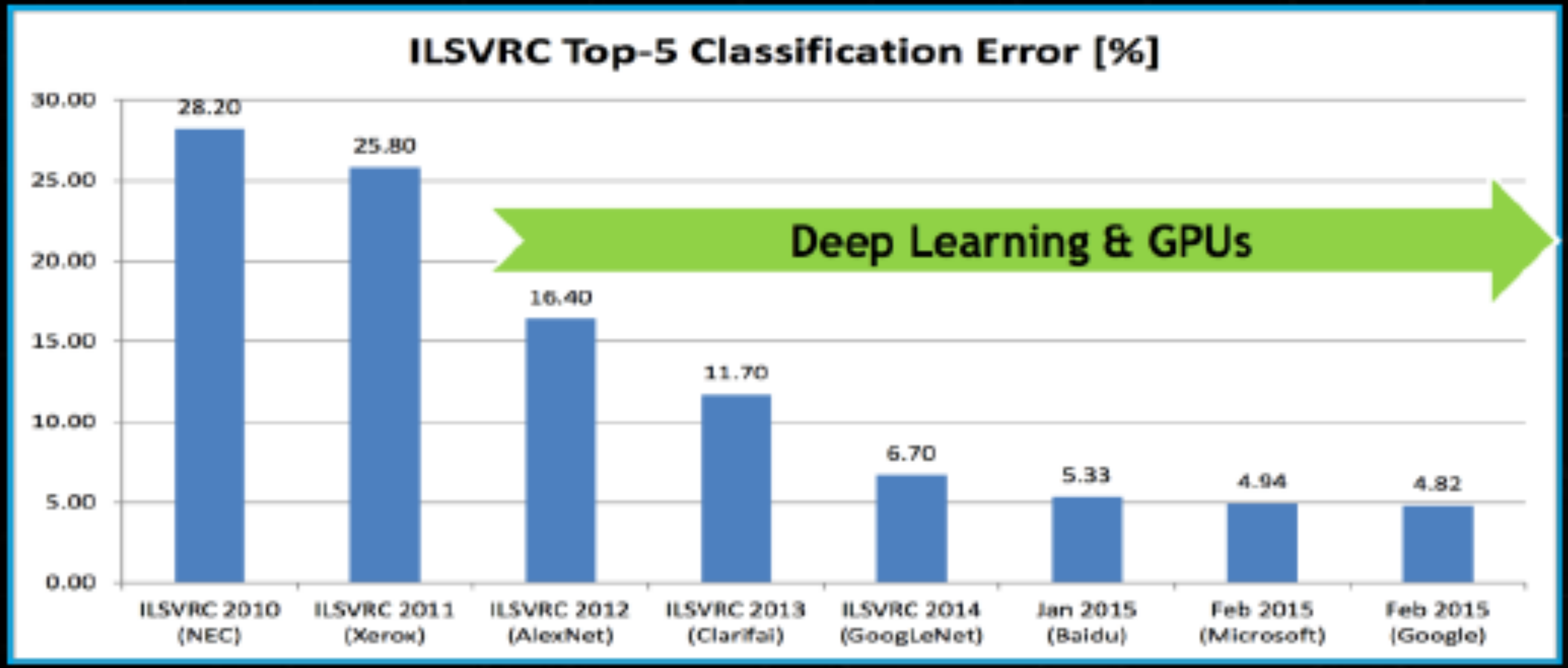
1.2M training images • 1000 object categories

Hosted by

IMAGENET







GPUS MAKE DEEP LEARNING ACCESSIBLE

Deep learning with COTS HPC systems

A. Coates, B. Huval, T. Wang, D. Wu, A. Ng, B. Catanzaro

ICML 2013

“Now You Can Build Google’s $1M Artificial Brain on the Cheap”

WIRED

GOOGLE DATACENTER

1,000 CPU Servers
2,000 CPUs • 16,000 cores

600 kWatts
$5,000,000

STANFORD AI LAB

3 GPU-Accelerated Servers
12 GPUs • 18,432 cores

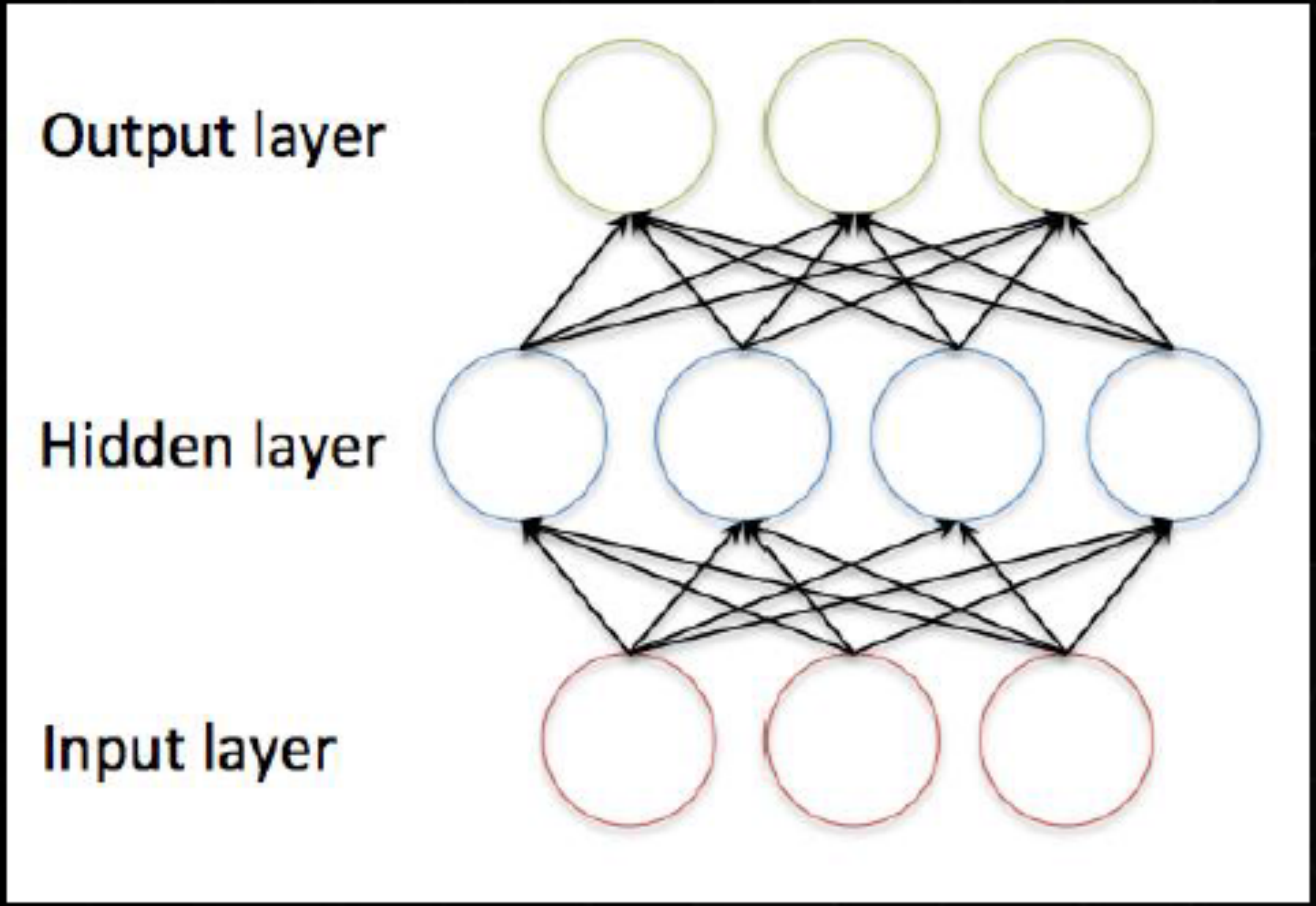
4 kWatts
$33,000

WHY ARE GPUs GOOD FOR DEEP LEARNING?

	Neural Networks	GPUs
Inherently Parallel	✓	✓
Matrix Operations	✓	✓
FLOPS	✓	✓
Bandwidth	✓	✓

GPUs deliver --

- *same or better prediction accuracy*
- *faster results*
- *smaller footprint*
- *lower power*
- *lower cost*



GPU ACCELERATION

Training A Deep, Convolutional Neural Network

Batch Size	Training Time CPU	Training Time GPU	GPU Speed Up
64 images	64 s	7.5 s	8.5X
128 images	124 s	14.5 s	8.5X
256 images	257 s	28.5 s	9.0X

- ILSVRC12 winning model: "Supervision"
- 7 layers
- 5 convolutional layers + 2 fully-connected
- ReLU, pooling, drop-out, response normalization
- Implemented with Caffe

- Dual 10-core Ivy Bridge CPUs
- 1 Tesla K40 GPU
- CPU times utilized Intel MKL BLAS library
- GPU acceleration from CUDA matrix libraries (cuBLAS)

Case Studies

19.6 GFLOPS

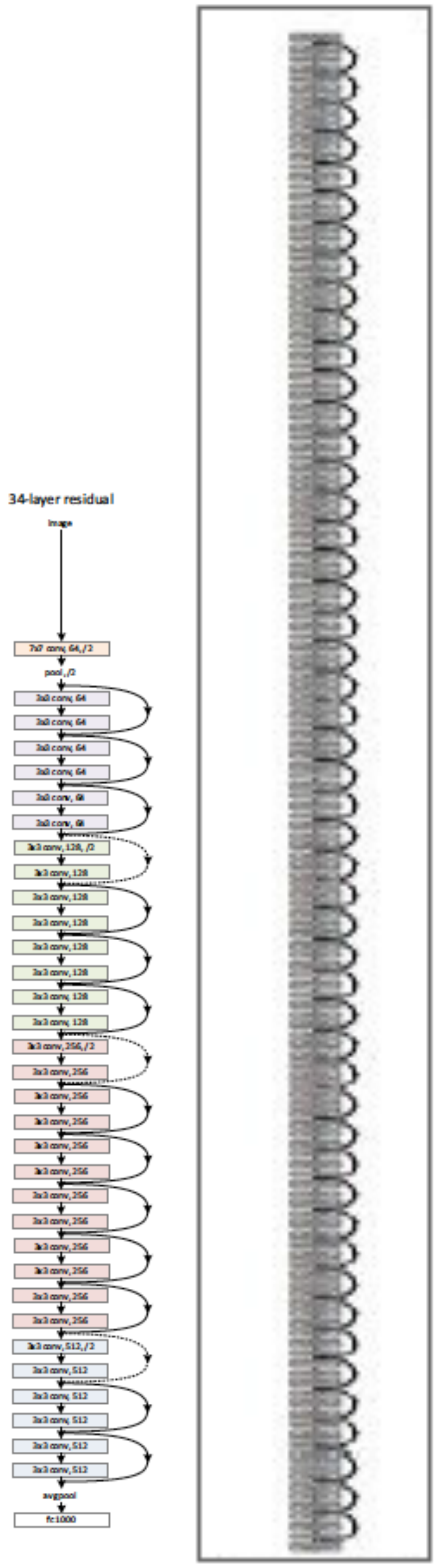
D	E
16 weight layers	19 weight layers
conv3-64 conv3-64	conv3-64 conv3-64
conv3-128 conv3-128	conv3-128 conv3-128
conv3-256 conv3-256 **conv3-256**	conv3-256 conv3-256 conv3-256 **conv3-256**
conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool	
FC-4096	
FC-4096	
FC-1000	
soft-max	

VGG (2014)

1.5 GFLOPS

GoogLeNet (2014)

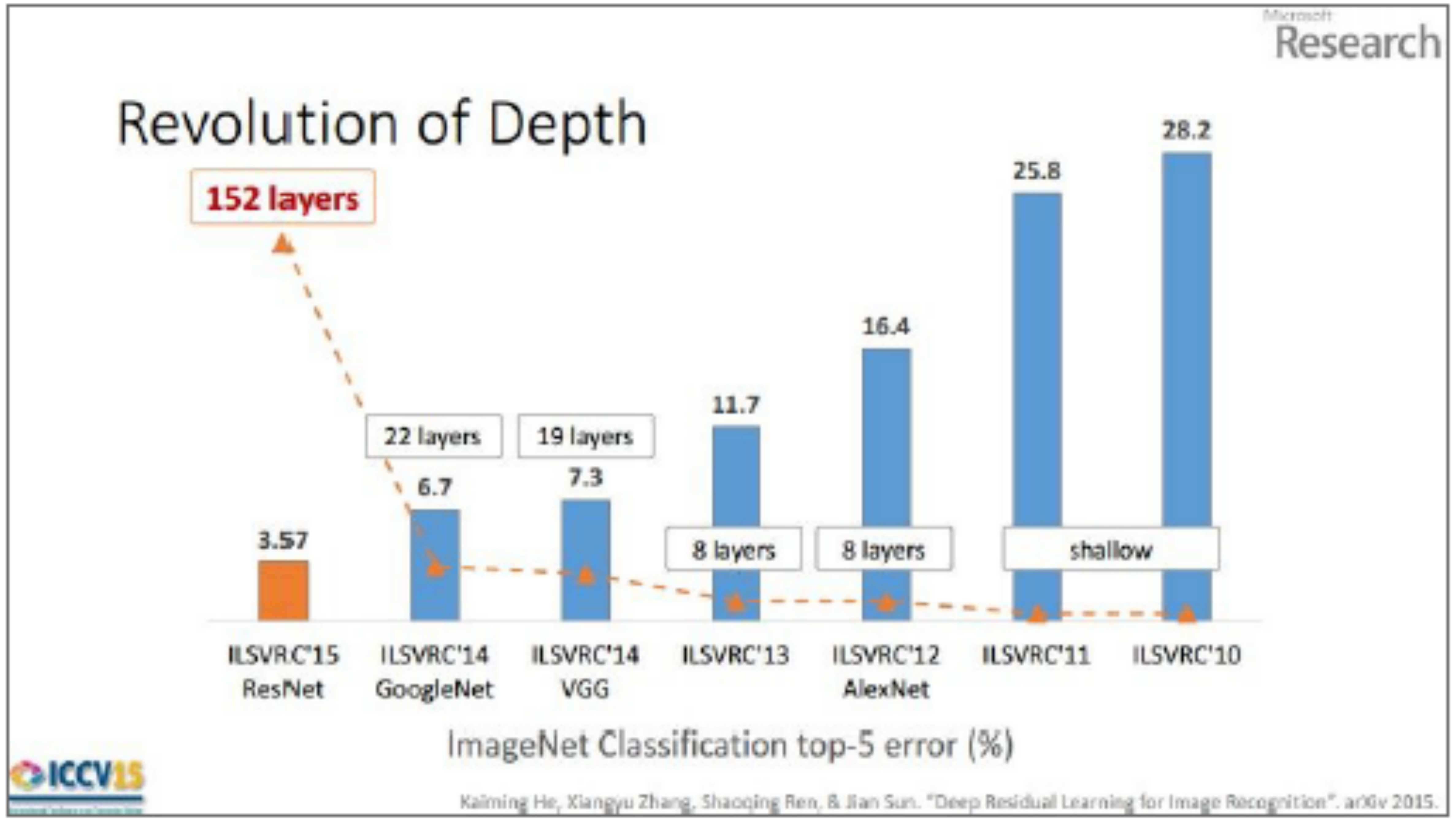
3.6-11.6 GFLOPS



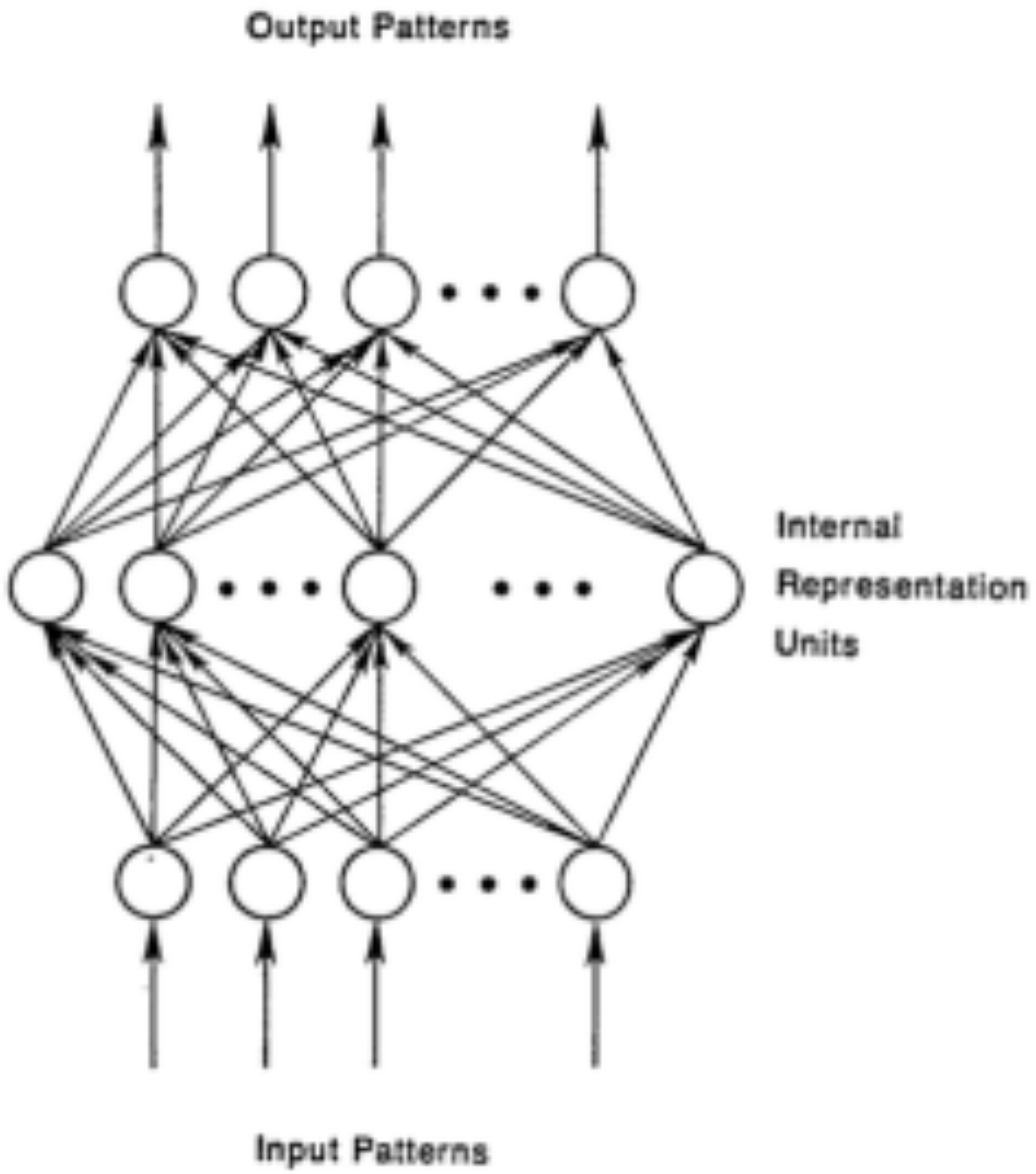
ResNet (2015)

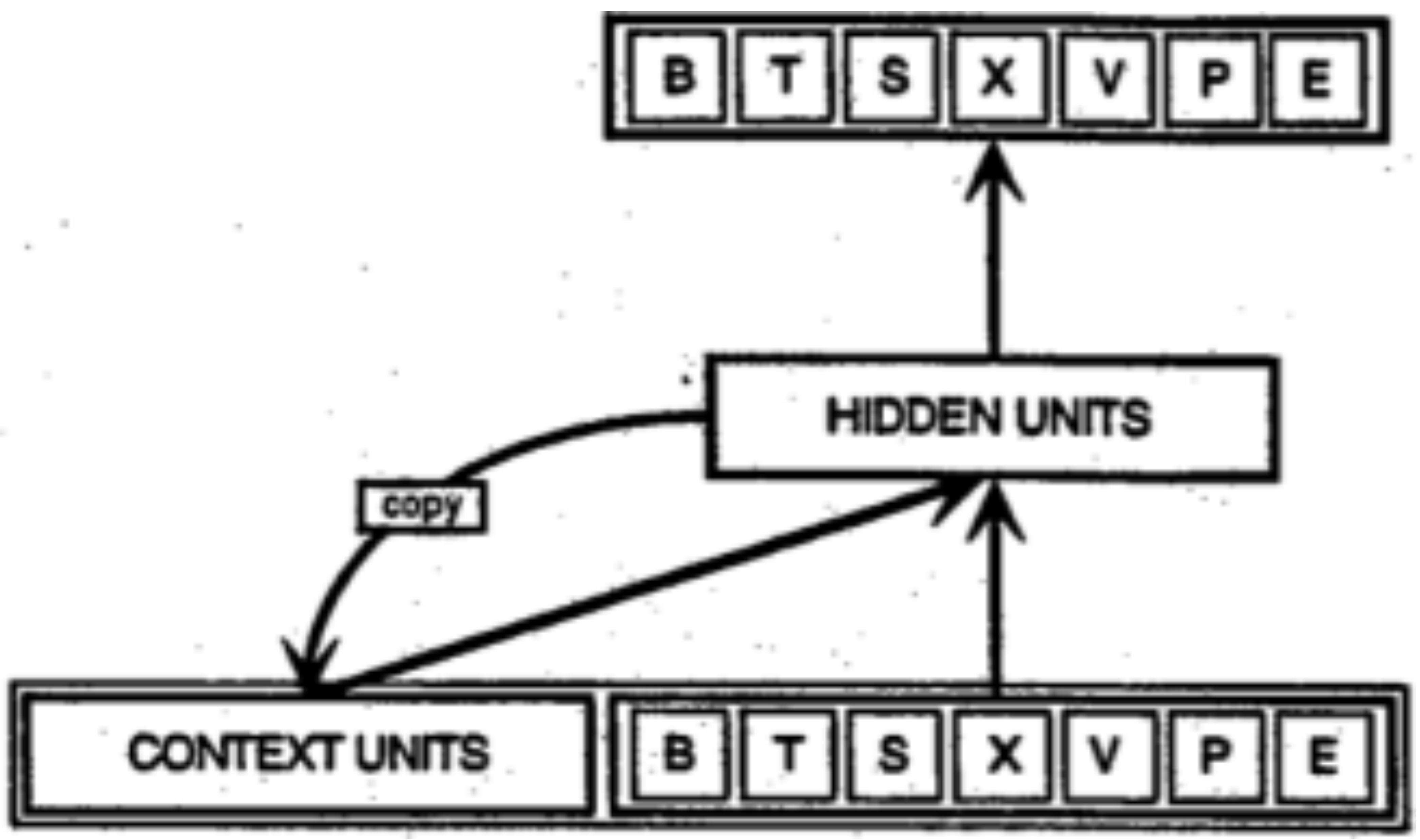
Training VGG for 50 epochs on Imagenet uses more than 1 ExaFlop

True HPC distributed training is needed



From Feed Forward to Recurrent

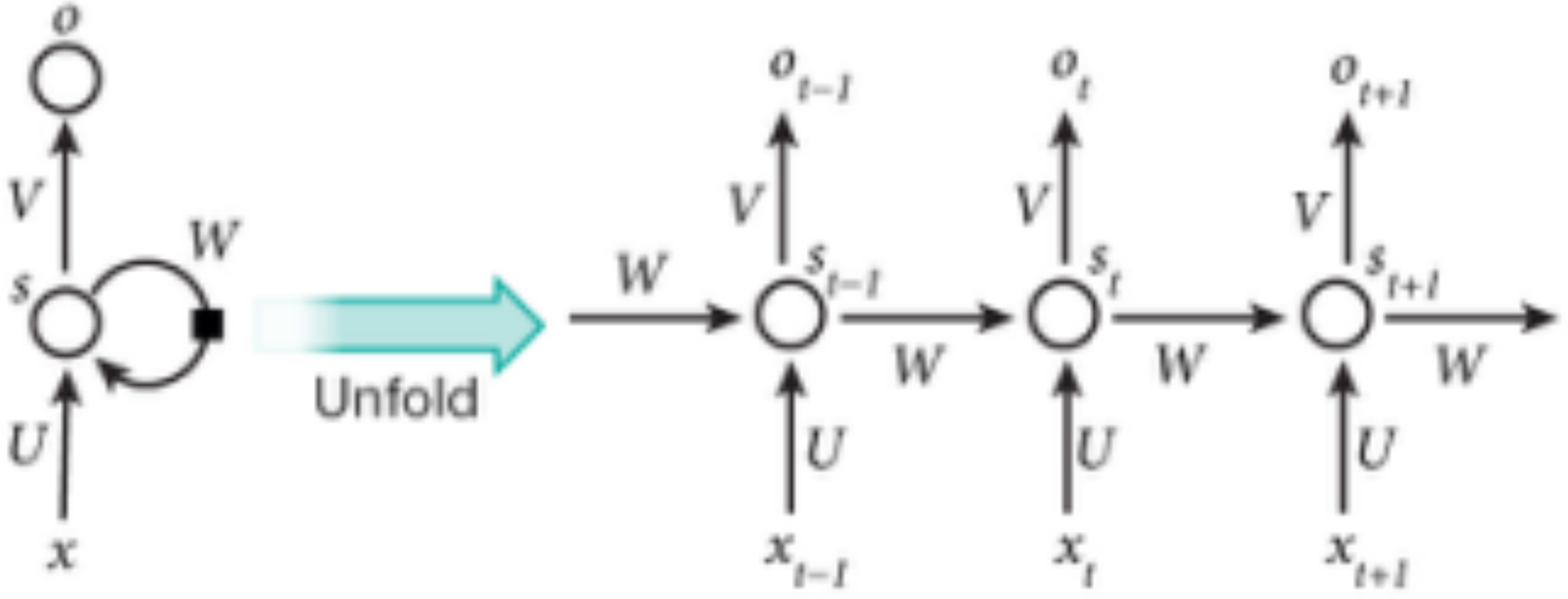




$$\mathbf{h}_t = \phi\left(W\mathbf{x}_t + U\mathbf{h}_{t-1}\right),$$

http://deeplearning4j.org/lstm.html

Back propagation through time



$$s_t = f(Ux_t + Ws_{t-1}) \qquad o_t = \mathrm{softmax}(Vs_t)$$

U, V, W are shared across all steps

Input / output are not necessary for all steps - prediction

$$E_t(y_t, \hat{y}_t) = -y_t \log \hat{y}_t$$

$$E(y, \hat{y}) = \sum_t E_t(y_t, \hat{y}_t) = -\sum_t y_t \log \hat{y}_t$$

$$\frac{\partial E}{\partial W} = \sum_t \frac{\partial E_t}{\partial W}$$

$$\frac{\partial E_3}{\partial W} = \sum_{k=0}^{3} \frac{\partial E_3}{\partial \hat{y}_3} \frac{\partial \hat{y}_3}{\partial s_3} \frac{\partial s_3}{\partial s_k} \frac{\partial s_k}{\partial W}$$

Standard Feedforward backprop

Sum up gradients for W at each timestep

Efficiency

Duration of one epoch on a 24 core Intel Xeon E5 ~ 3000 seconds → an experiment would take ~ 1 day

Duration of one epoch on a Tesla k40 ~ 100 seconds → an experiment would take ~ 1 hour

The speedup is in this case ~ 30x; however, when testing with different parameters, speedups of 10x - 70x were recorded

Results	SGD	BDLSTM	LSTM
Movie Lens	81%	87%	84%
Sentiment 140	78%	84%	82%

Live demos

http://www.cs.toronto.edu/~graves/handwriting.html
http://www.inkposter.com/
http://www.cs.toronto.edu/~ilya/rnn.html
http://104.131.78.120/
http://www.cs.toronto.edu/~hinton/digits.html
http://www.cs.toronto.edu/~hinton/adi/index.htm

Machine learning @ SURFsara

ML Software on Cartesius

Already installed software:

- Caffe
- Torch7
- Tensorflow
- Theano/Lasagne
- cuda-convnet2
- CNTK
- MXNet
- scikit-learn
- cuDNN
- NVIDIA DIGITS

Other software:

- Install yourself…
- …or ask us to install it

DIGITS training for GoogLeNet

Execution on Cartesius

2 options:

- create a sleep job and connect there for interactive usage
- create a “regular” batch job (preferred option)

```
#!/bin/bash
#SBATCH -p gpu
#SBATCH -N 1
#SBATCH -t 5:00:00
sleep 18000
```

sleep_job.sh

```
#!/bin/bash
#SBATCH -p gpu
#SBATCH -N 1
#SBATCH -t 5:00:00
module load cuda
module load cudnn
module load python/2.7.9
THEANO_FLAGS=‘mode=FAST_RUN,device=gpu,floatX=float32,lib.
cnmem=1' srun -u python code/convolutional_mlp.py
```

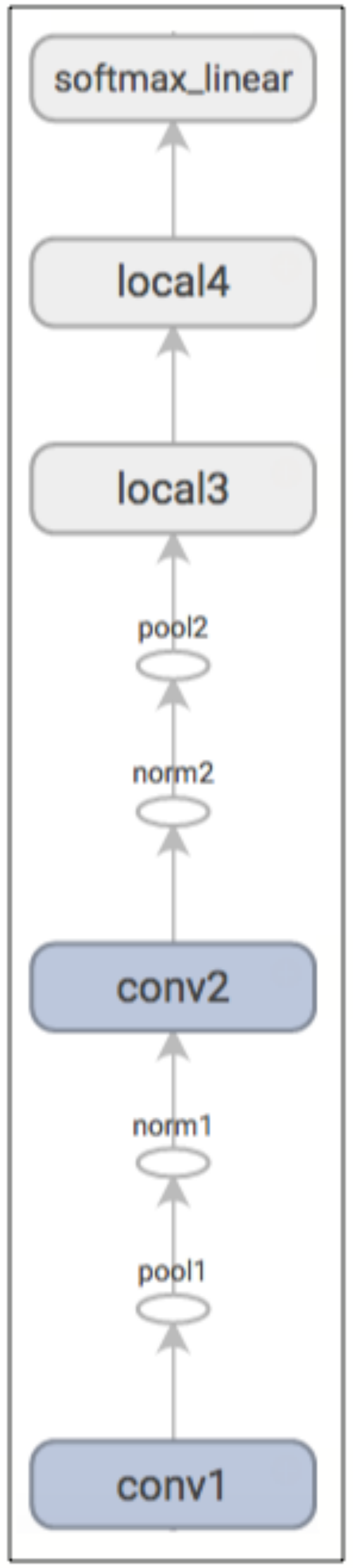
theano_job.sh

Tensorflow basics

- Represents computations as graphs.
- Executes graphs in the context of Sessions.
- Represents data as tensors.
- Maintains state with Variables.
- Uses feeds and fetches to get data into and out of arbitrary operations.

https://www.tensorflow.org/versions/r0.10/get_started/basic_usage.html

Study 1: training CIFAR10

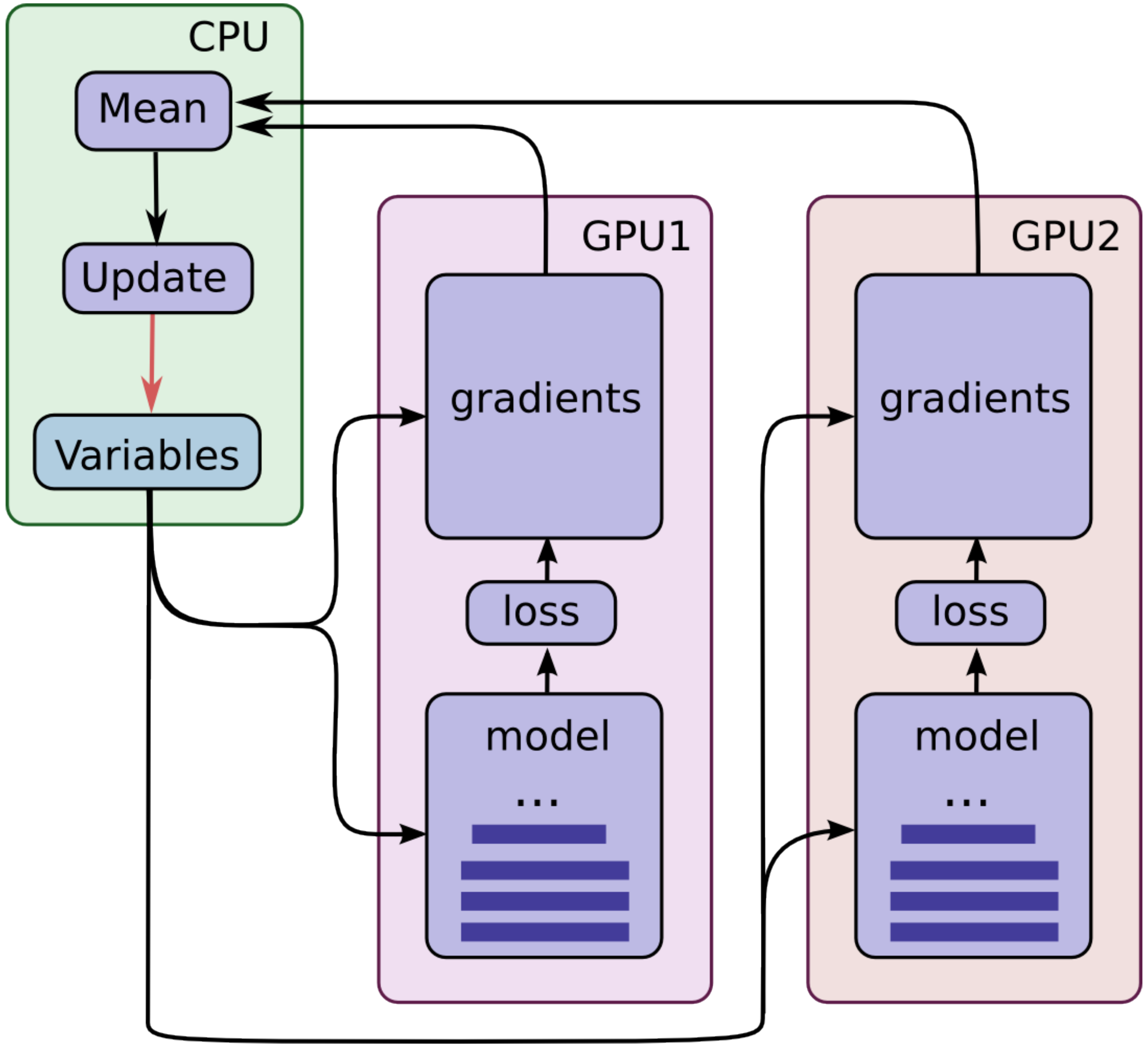


conv	convolution and rectified linear activation.
pool	max pooling.
norm	local response normalization.
conv	convolution and rectified linear activation.
norm	local response normalization.
pool	max pooling.
local	fully connected layer with rectified linear activation.
local	fully connected layer with rectified linear activation.
softmax_linea	linear transformation to produce logits.

https://www.tensorflow.org/versions/r0.11/tutorials/deep_cnn/index.html

Study 1: Data parallelism

```
tower_grads = []
  for i in xrange(FLAGS.num_gpus):
    with tf.device('/gpu:%d' % i):
      with tf.name_scope('%s_%d' % (cifar10.TOWER_NAME, i)) as scope:
        # Calculate the loss for one tower of the CIFAR model. This function
        # constructs the entire CIFAR model but shares the variables across
        # all towers.
        loss = tower_loss(scope)

        # Reuse variables for the next tower.
        tf.get_variable_scope().reuse_variables()

        # Retain the summaries from the final tower.
        summaries = tf.get_collection(tf.GraphKeys.SUMMARIES, scope)

        # Calculate the gradients for the batch of data on this CIFAR tower.
        grads = opt.compute_gradients(loss)

        # Keep track of the gradients across all towers.
        tower_grads.append(grads)

  # We must calculate the mean of each gradient. Note that this is the
  # synchronization point across all towers.
  grads = average_gradients(tower_grads)
```



Study 2: Model parallelism in TF

```
with tf.device('/gpu:0'):
    a = tf.placeholder(tf.float32, [10000, 10000])
    b = tf.placeholder(tf.float32, [10000, 10000])
    # Compute A^n and B^n and store results in c1
    c1.append(matpow(a, n))
    c1.append(matpow(b, n))

with tf.device('/cpu:0'):
  sum = tf.add_n(c1) #Addition of all elements in c1, i.e. A^n + B^n

t1_1 = datetime.datetime.now()
with tf.Session(config=tf.ConfigProto( \
    log_device_placement=log_device_placement)) as sess:
    # Run the op.
    sess.run(sum, {a:A, b:B})
t2_1 = datetime.datetime.now()
```

```
# GPU:0 computes A^n
with tf.device('/gpu:0'):
    # Compute A^n and store result in c2
    a = tf.placeholder(tf.float32, [10000, 10000])
    c2.append(matpow(a, n))

# GPU:1 computes B^n
with tf.device('/gpu:1'):
    # Compute B^n and store result in c2
    b = tf.placeholder(tf.float32, [10000, 10000])
    c2.append(matpow(b, n))

with tf.device('/cpu:0'):
  sum = tf.add_n(c2) #Addition of all elements in c2, i.e. A^n + B^n

t1_2 = datetime.datetime.now()
with tf.Session(config=tf.ConfigProto( \
    log_device_placement=log_device_placement)) as sess:
    # Run the op.
    sess.run(sum, {a:A, b:B})
t2_2 = datetime.datetime.now()
```

Study 3: Simple Keras example

General repository https://github.com/kjw0612/awesome-rnn

KERAS example

```
model = Sequential()
model.add(LSTM(512, return_sequences=True, input_shape=(maxlen, len(chars))))
model.add(Dropout(0.2))
model.add(LSTM(512, return_sequences=False))
model.add(Dropout(0.2))
model.add(Dense(len(chars)))
model.add(Activation('softmax'))

model.compile(loss='categorical_crossentropy', optimizer='rmsprop')
```

Questions?