Hardware Architectures for Deep Neural Networks

CICS/MTL Tutorial

March 27, 2017

Website: http://eyeriss.mit.edu/tutorial.html

Massachusetts Institute of Technology

Speakers and Contributors

Joel Emer

Senior Distinguished Research Scientist

NVIDIA

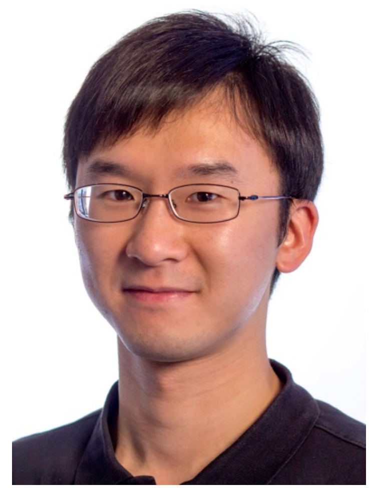
Professor

MIT

Vivienne Sze

Professor

MIT



Yu-Hsin Chen

PhD Candidate

MIT

Tien-Ju Yang

PhD Candidate

MIT

Outline

- Overview of Deep Neural Networks
- DNN Development Resources
- Survey of DNN Hardware
- DNN Accelerators
- DNN Model and Hardware Co-Design

Participant Takeaways

- Understand the key design considerations for DNNs
- Be able to evaluate different implementations of DNN with benchmarks and comparison metrics
- Understand the tradeoffs between various architectures and platforms
- Assess the utility of various optimization approaches
- Understand recent implementation trends and opportunities

Resources

- **Eyeriss Project: http://eyeriss.mit.edu**
  - Tutorial Slides
  - Benchmarking
  - Energy modeling
  - Mailing List for updates   Follow @eems_mit
    - **http://mailman.mit.edu/mailman/listinfo/eems-news**
  - **Paper based on today's tutorial:**
    - V. Sze, Y.-H. Chen, T-J. Yang, J. Emer, "*Efficient Processing of Deep Neural Networks: A Tutorial and Survey*", arXiv, 2017

Background of
Deep Neural Networks

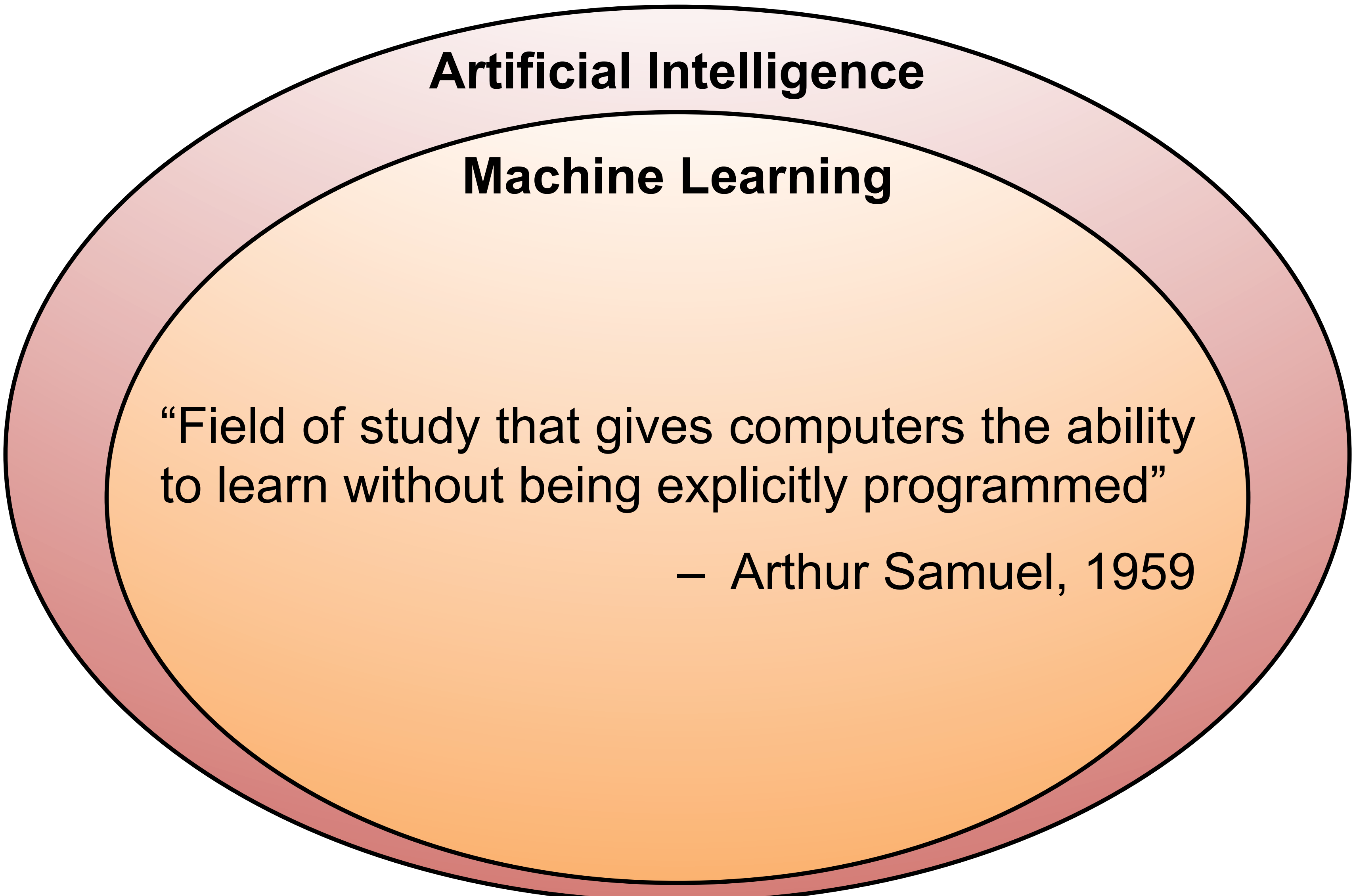
Artificial Intelligence

Artificial Intelligence

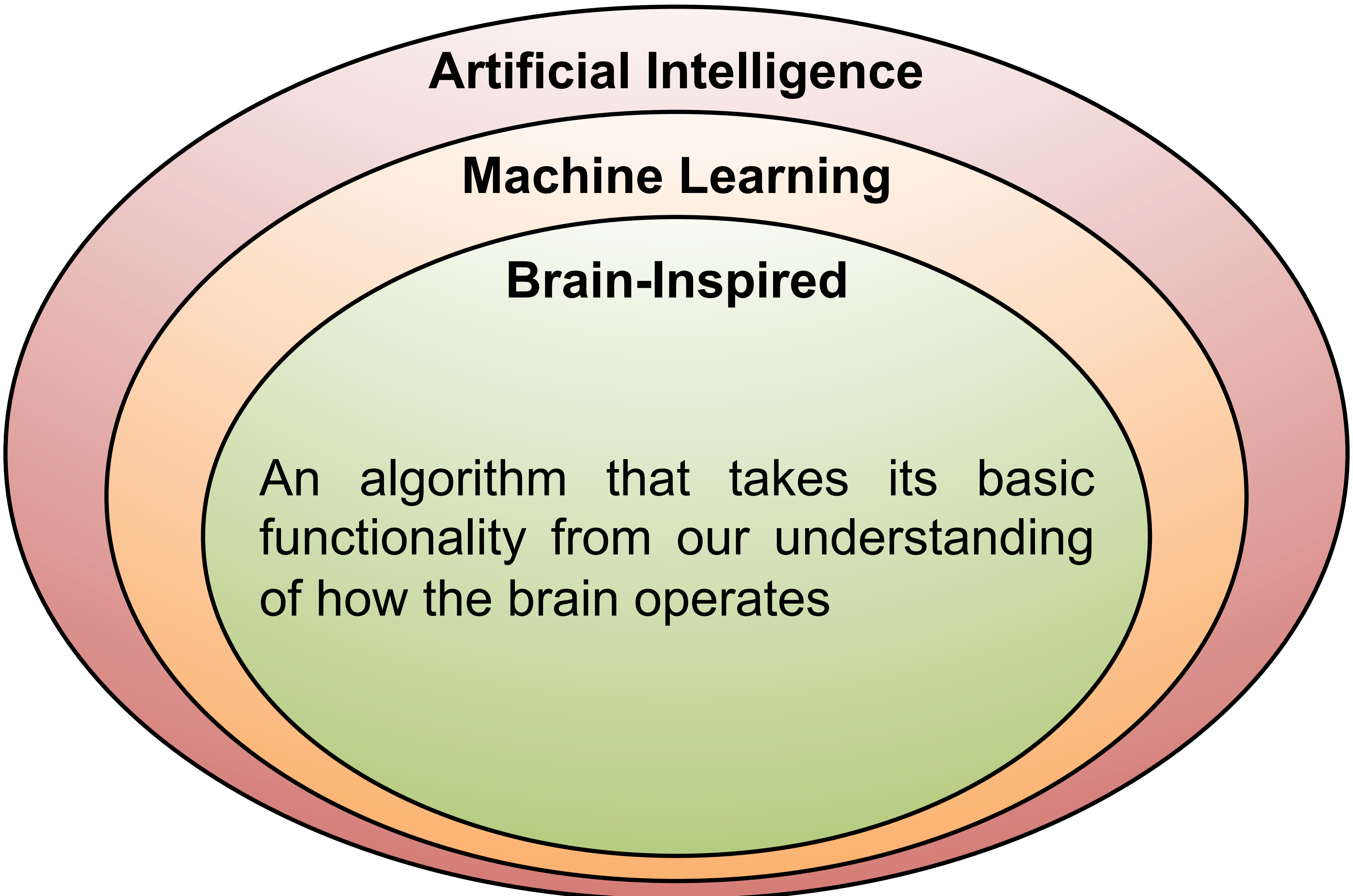
"The science and engineering of creating intelligent machines"

- John McCarthy, 1956

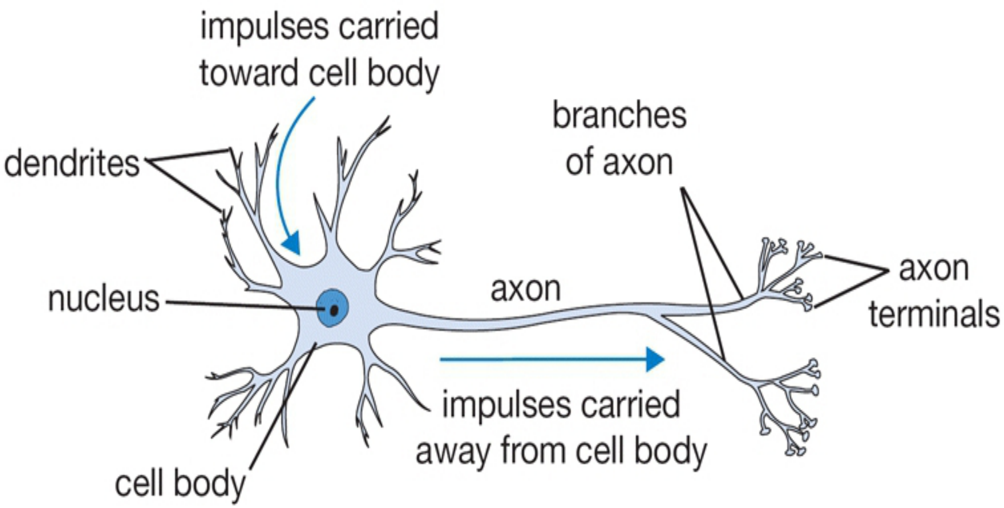
AI and Machine Learning

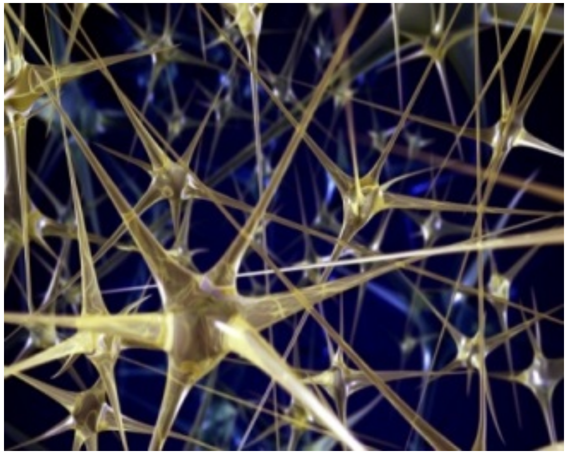


Brain-Inspired Machine Learning



How Does the Brain Work?

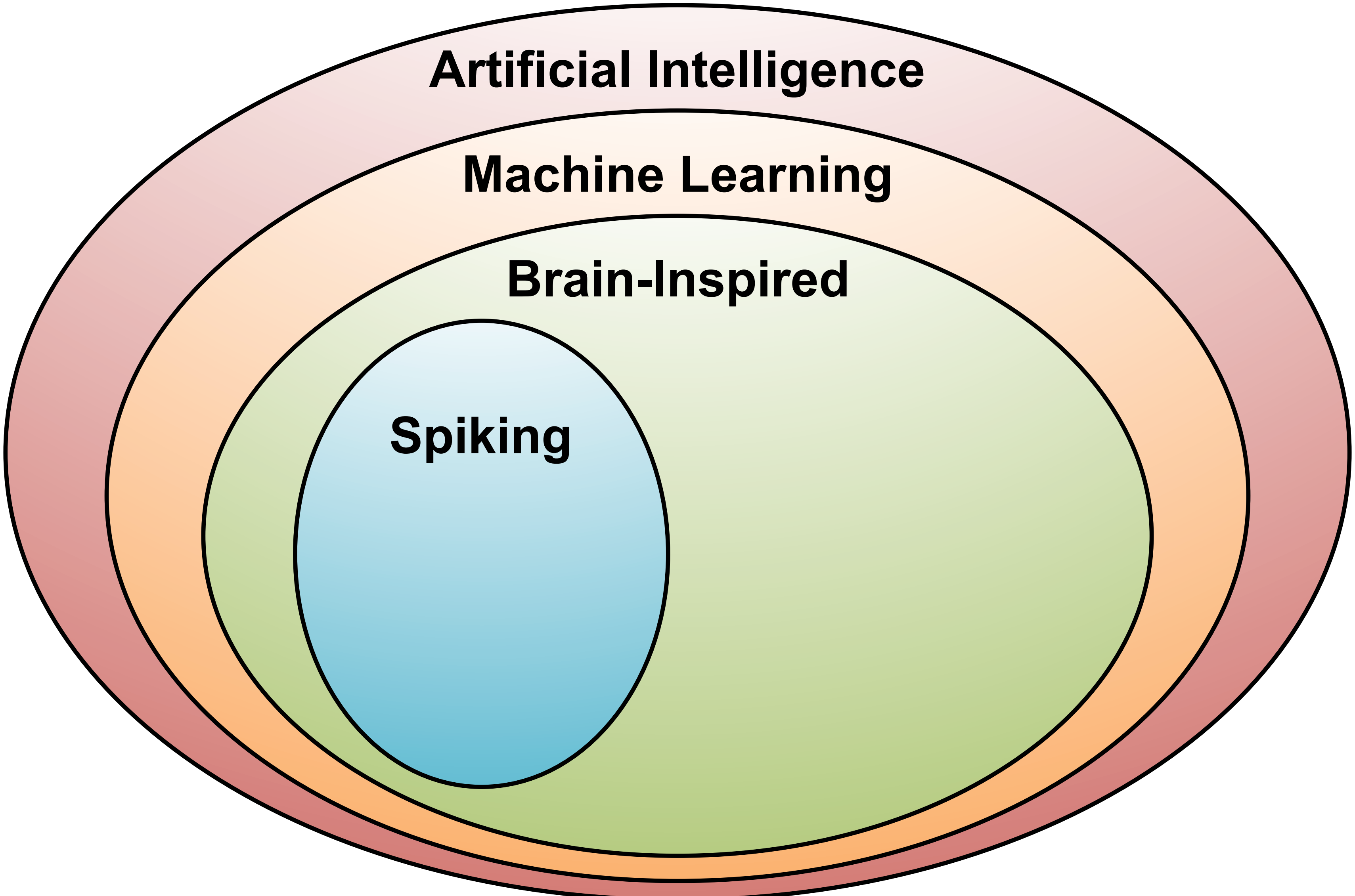




- The basic computational unit of the brain is a **neuron**
  → 86B neurons in the brain
- Neurons are connected with nearly **10^{14} – 10^{15} synapses**
- Neurons receive input signal from **dendrites** and produce output signal along **axon**, which interact with the dendrites of other neurons via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

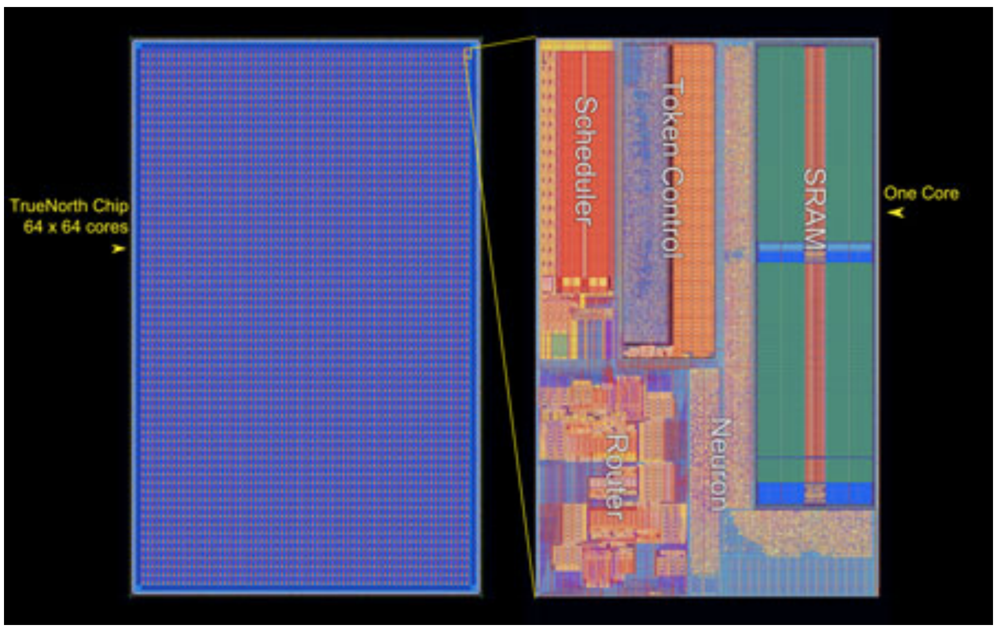
Image Source: Stanford

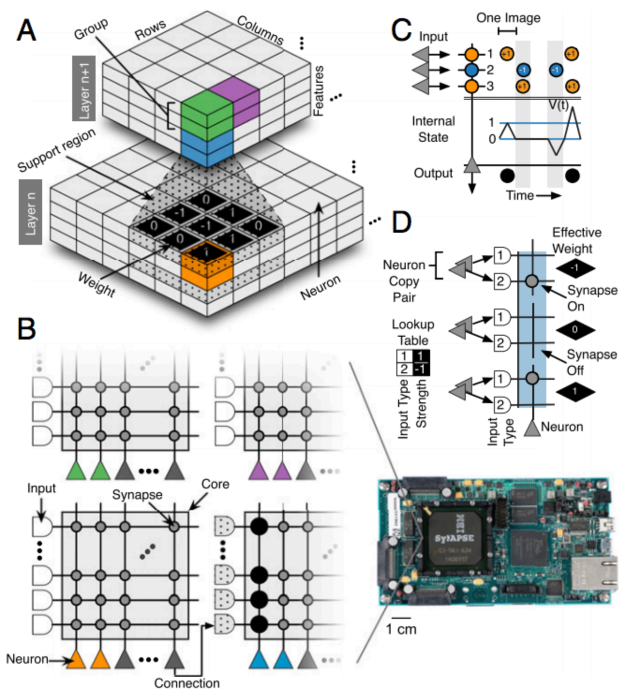
Spiking-based Machine Learning



Spiking Architecture

- Brain-inspired
- Integrate and fire
- Example: IBM TrueNorth

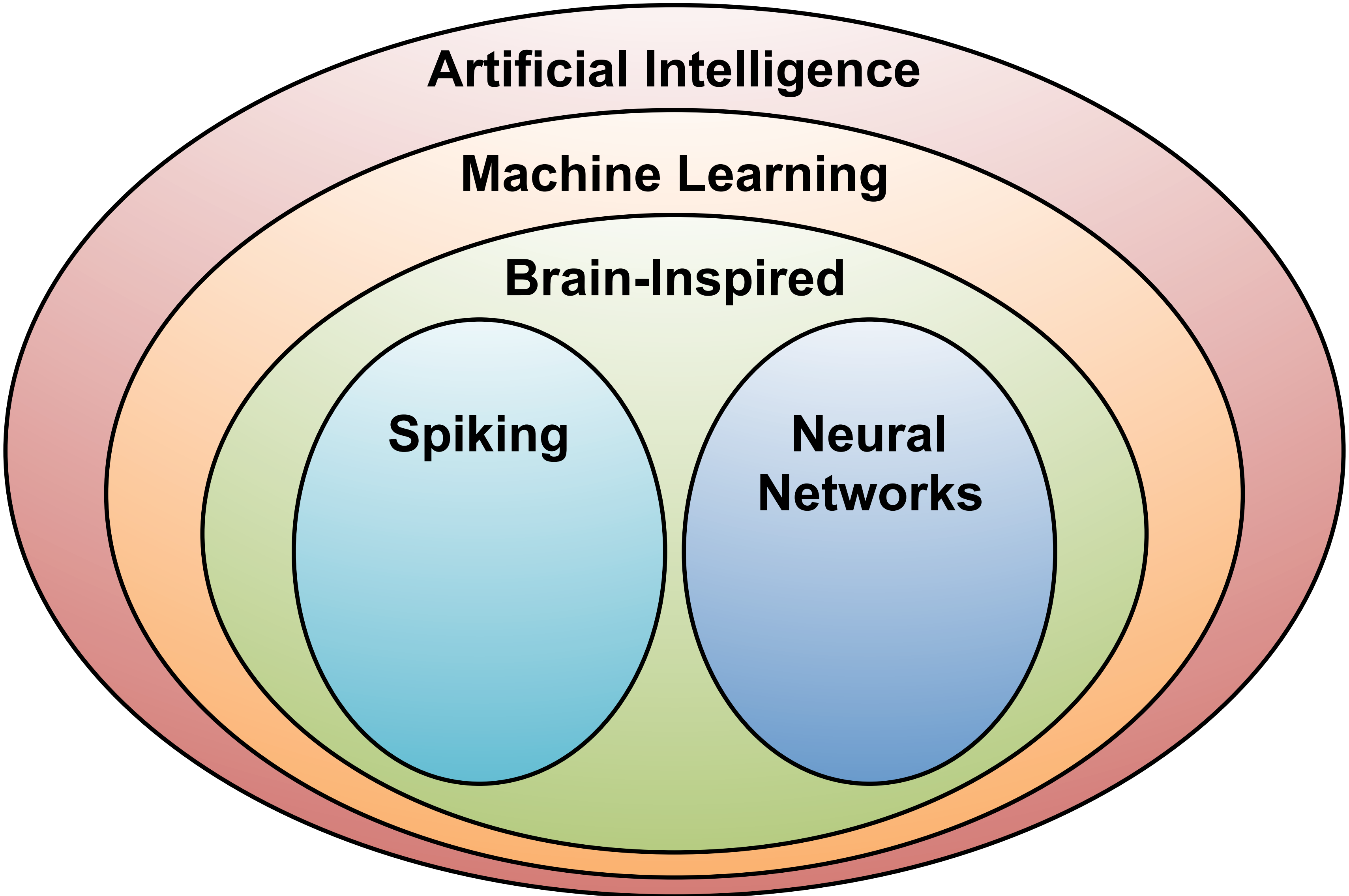




[Merolla et al., Science 2014; Esser et al., PNAS 2016]

http://www.research.ibm.com/articles/brain-chip.shtml

Machine Learning with Neural Networks



Neural Networks: Weighted Sum

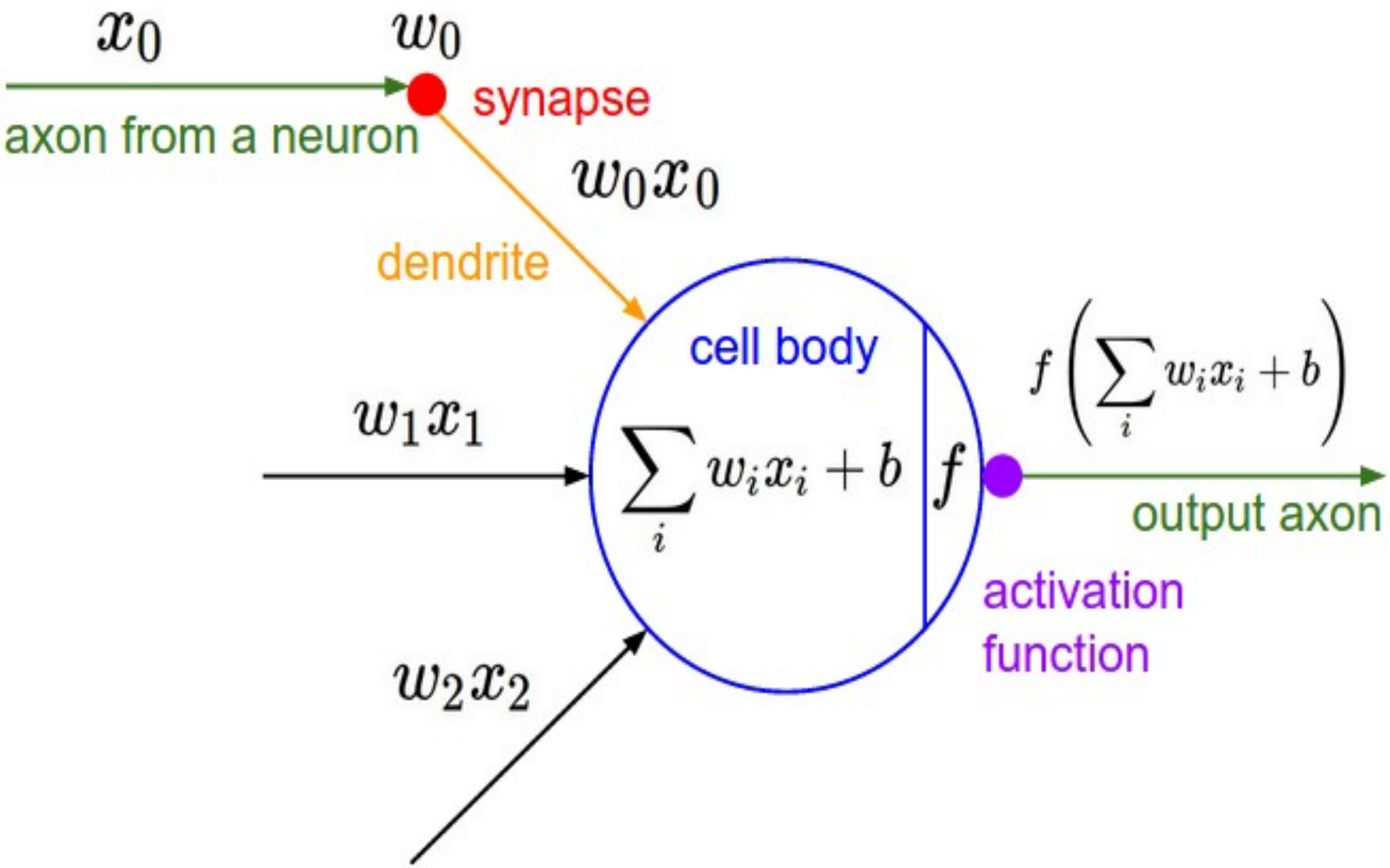


Image Source: Stanford

Many Weighted Sums

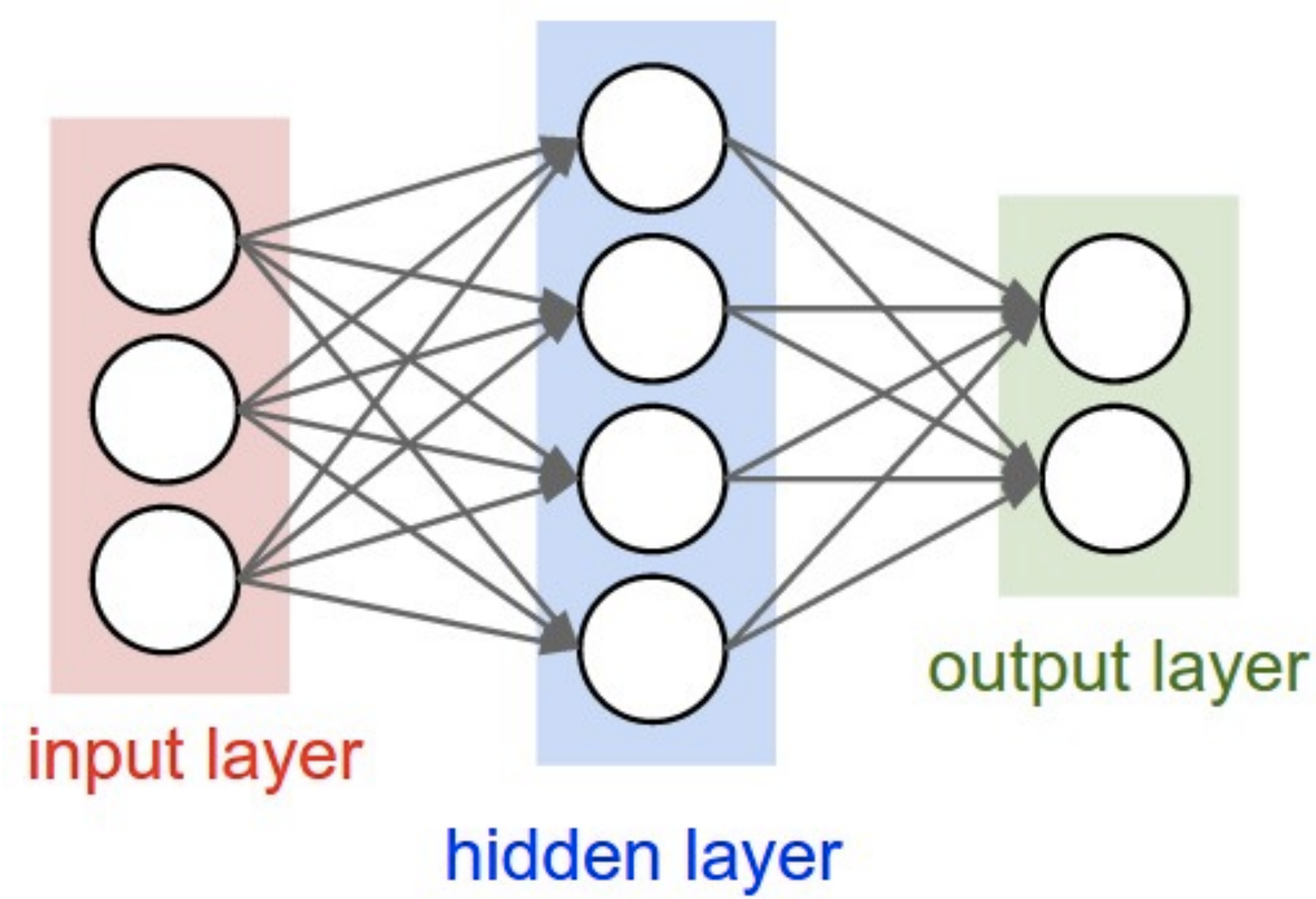
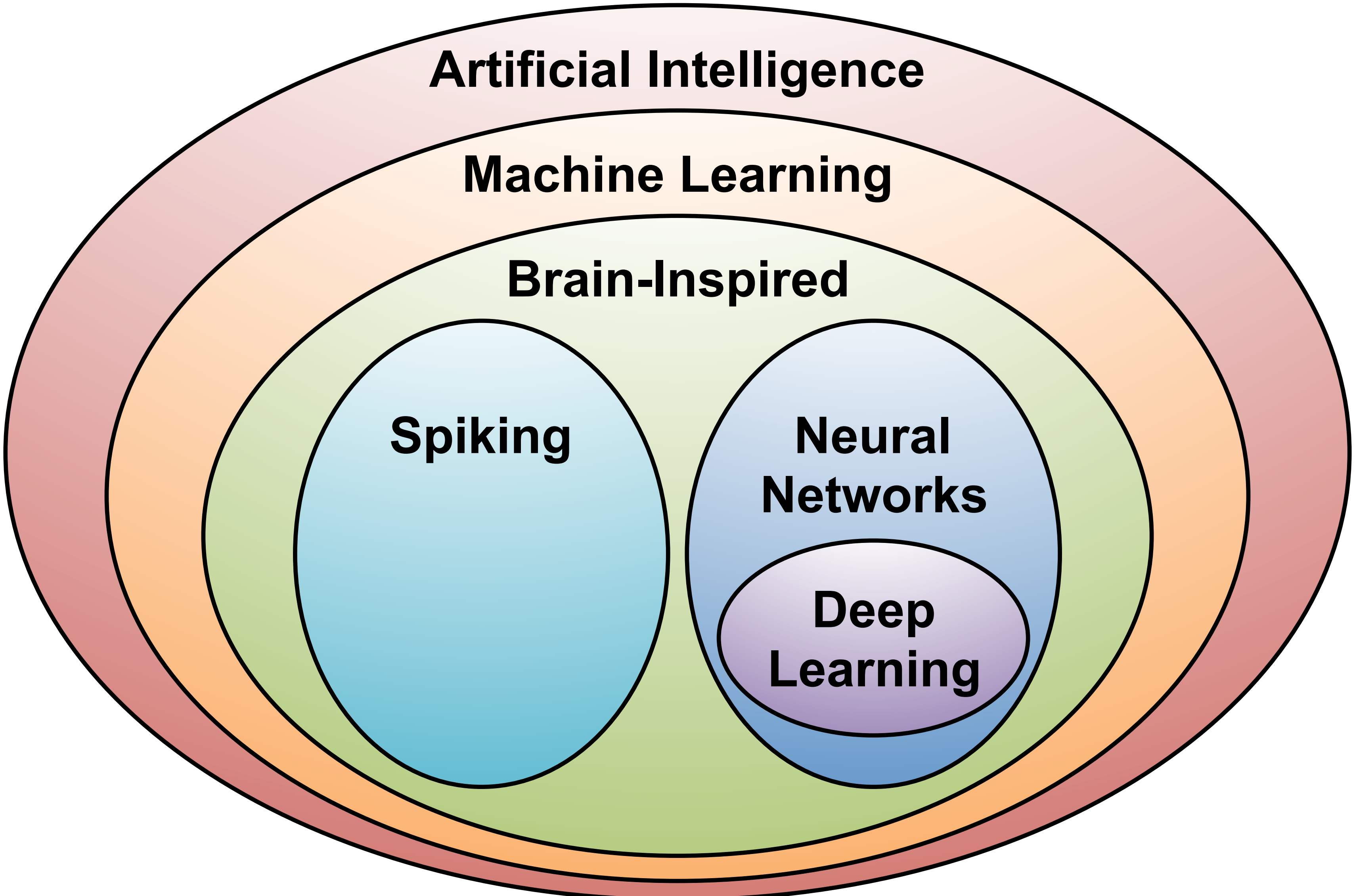


Image Source: Stanford

Deep Learning



What is Deep Learning?

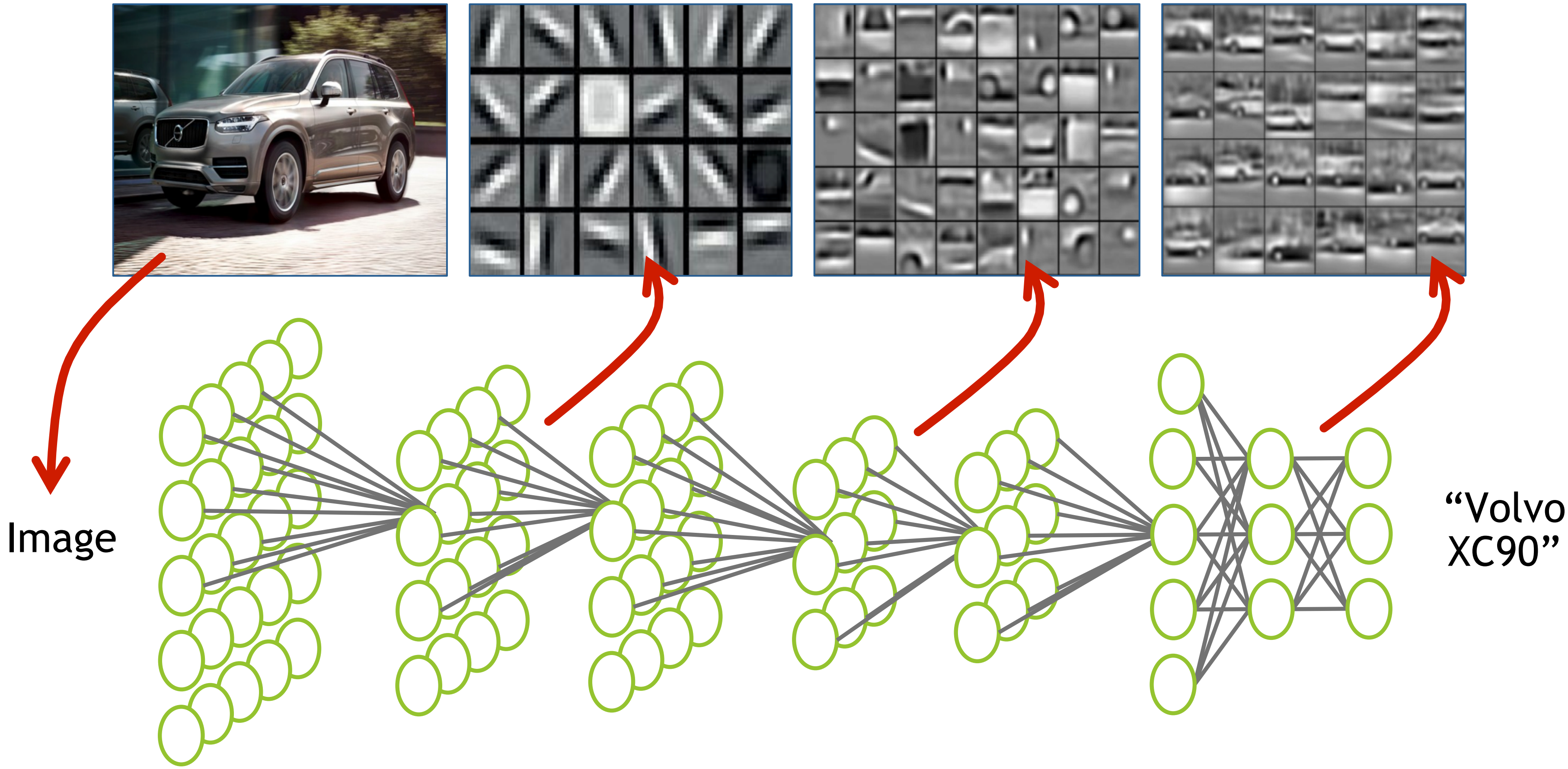


Image Source: [Lee et al., Comm. ACM 2011]

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

ImageNet Challenge

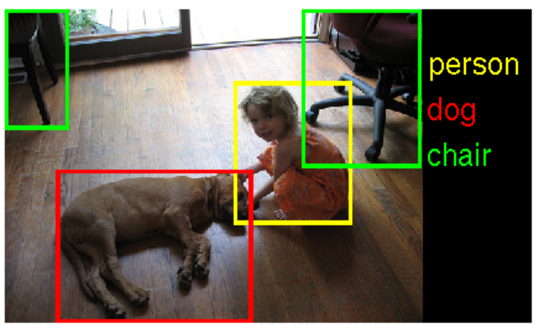
IMAGENET

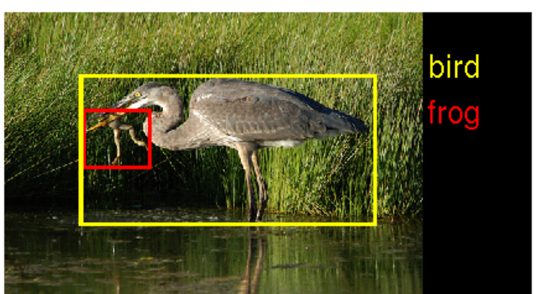
<u>Image Classification Task</u>:

1.2M training images • 1000 object categories

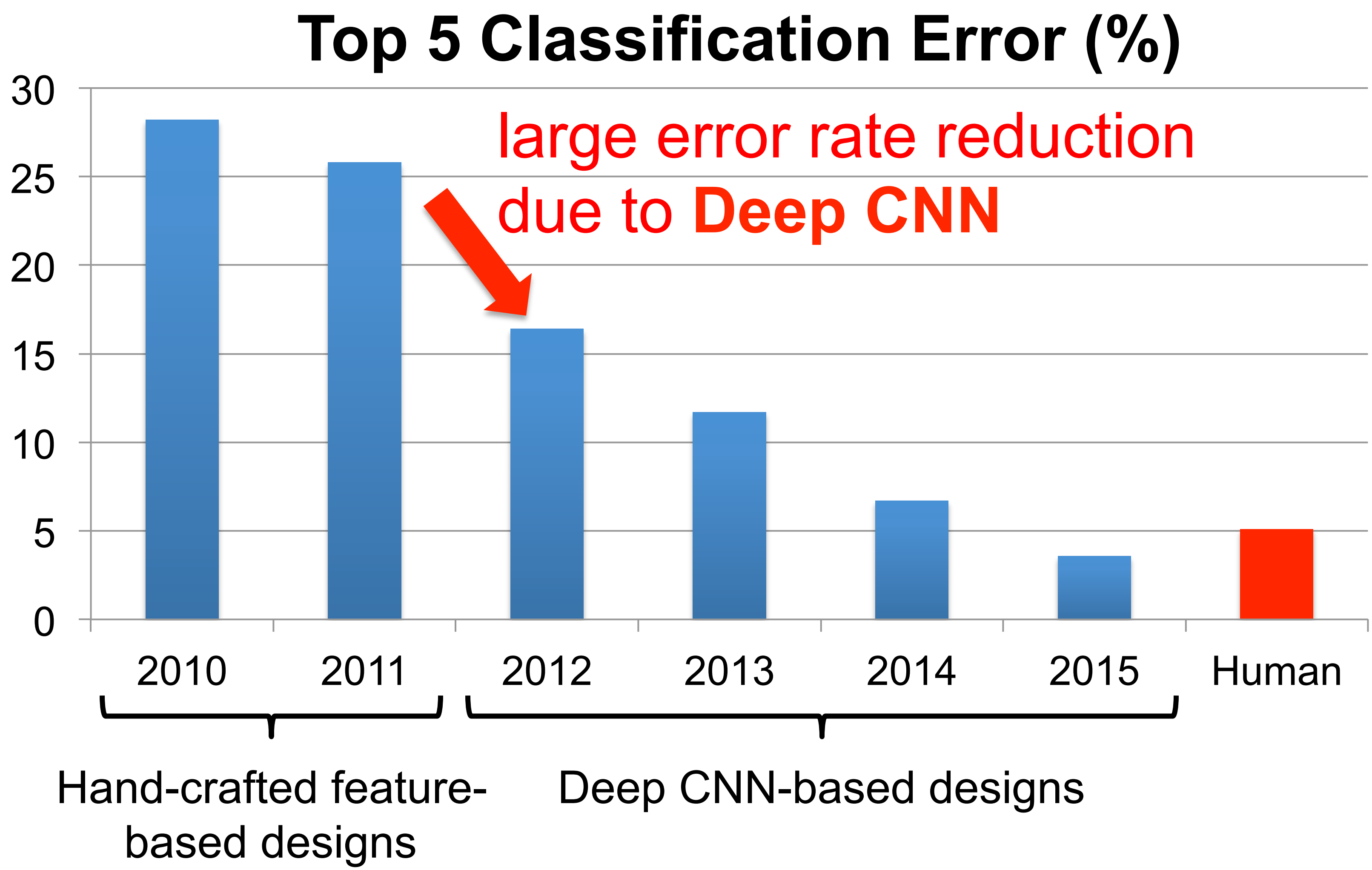
<u>Object Detection Task</u>:

456k training images • *200 object categories*



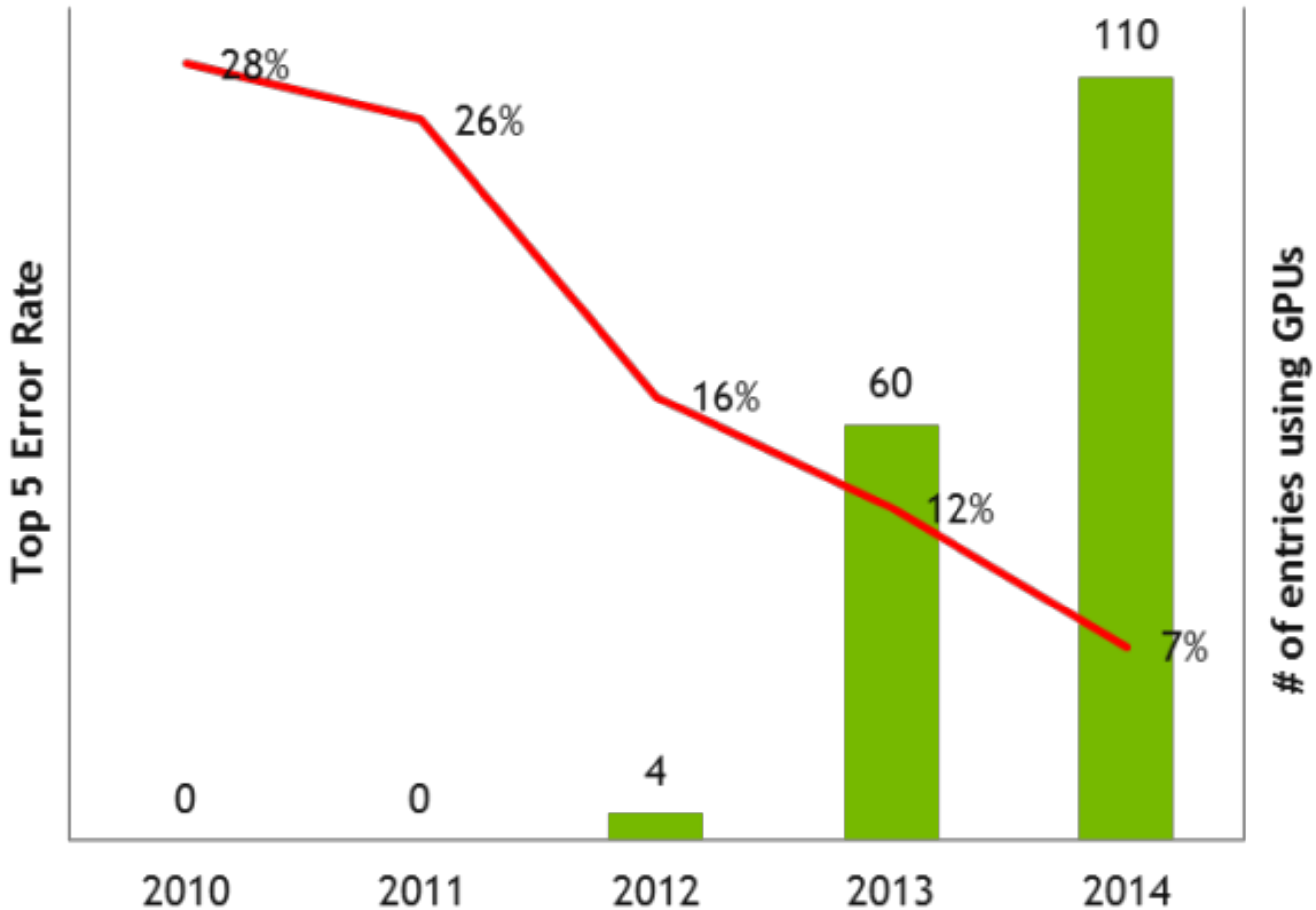


ImageNet: Image Classification Task



[Russakovsky et al., IJCV 2015]

GPU Usage for ImageNet Challenge

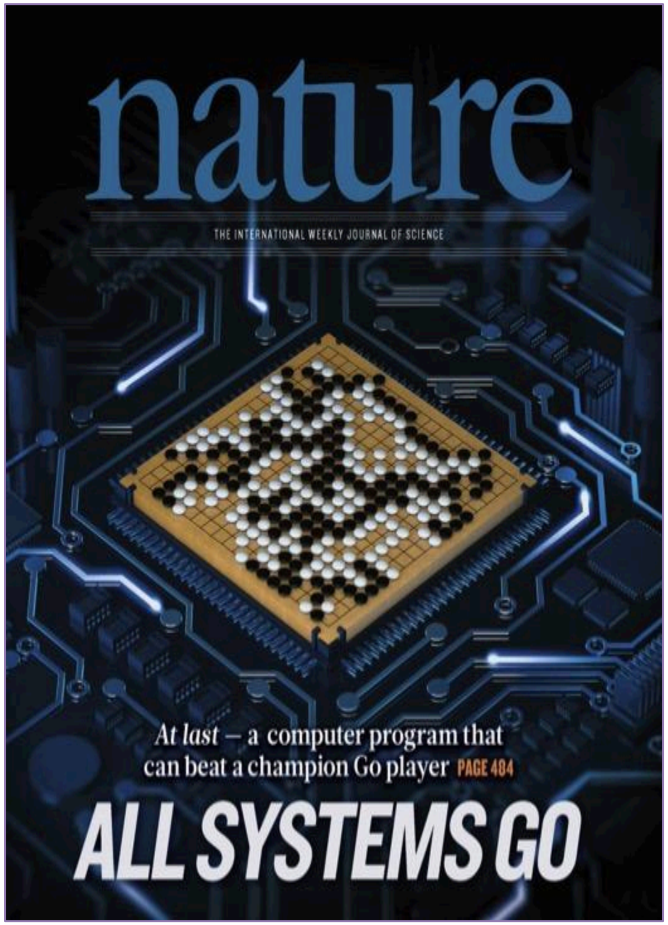


Established Applications

- **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**

Deep Learning on Games

Google DeepMind AlphaGo

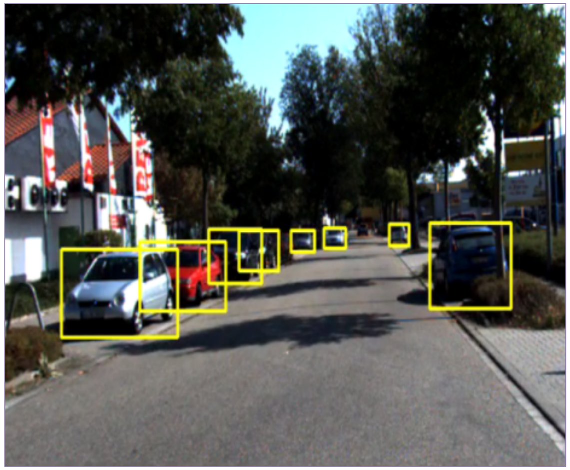


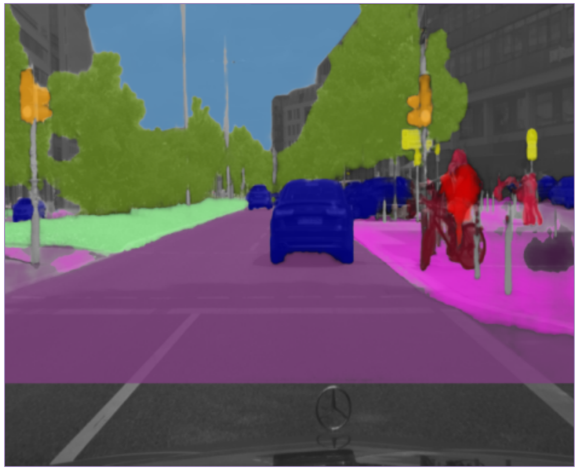
Emerging Applications

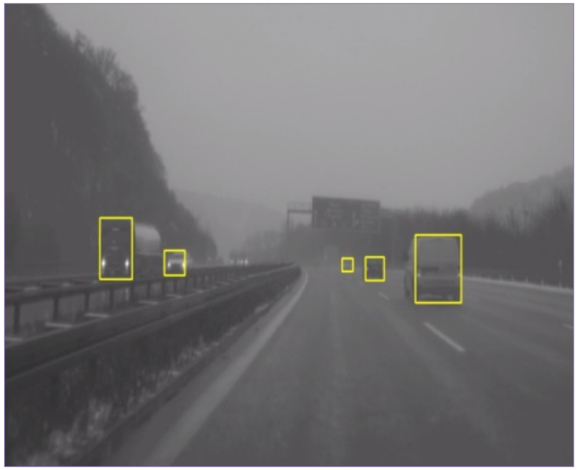
- **Medical** (Cancer Detection, Pre-Natal)
- **Finance** (Trading, Energy Forecasting, Risk)
- **Infrastructure** (Structure Safety and Traffic)
- Weather Forecasting and Event Detection

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

Deep Learning for Self-driving Cars







Opportunities

From EE Times – September 27, 2016

”Today the job of training machine learning models is limited by compute, if we had faster processors we’d run bigger models…in practice we train on a reasonable subset of data that can finish in a matter of months. We could use improvements of several orders of magnitude – 100x or greater.”

– Greg Diamos, Senior Researcher, SVAIL, Baidu

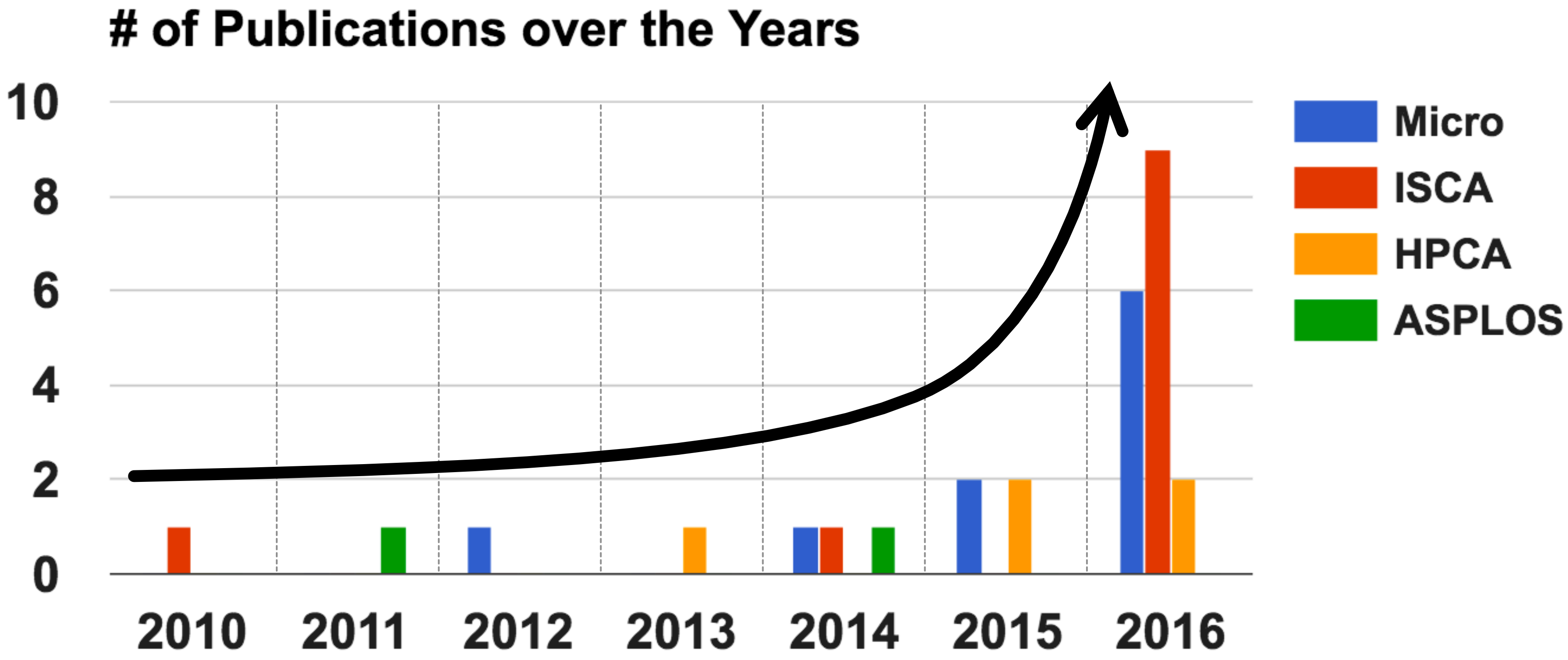
Overview of Deep Neural Networks

DNN Timeline

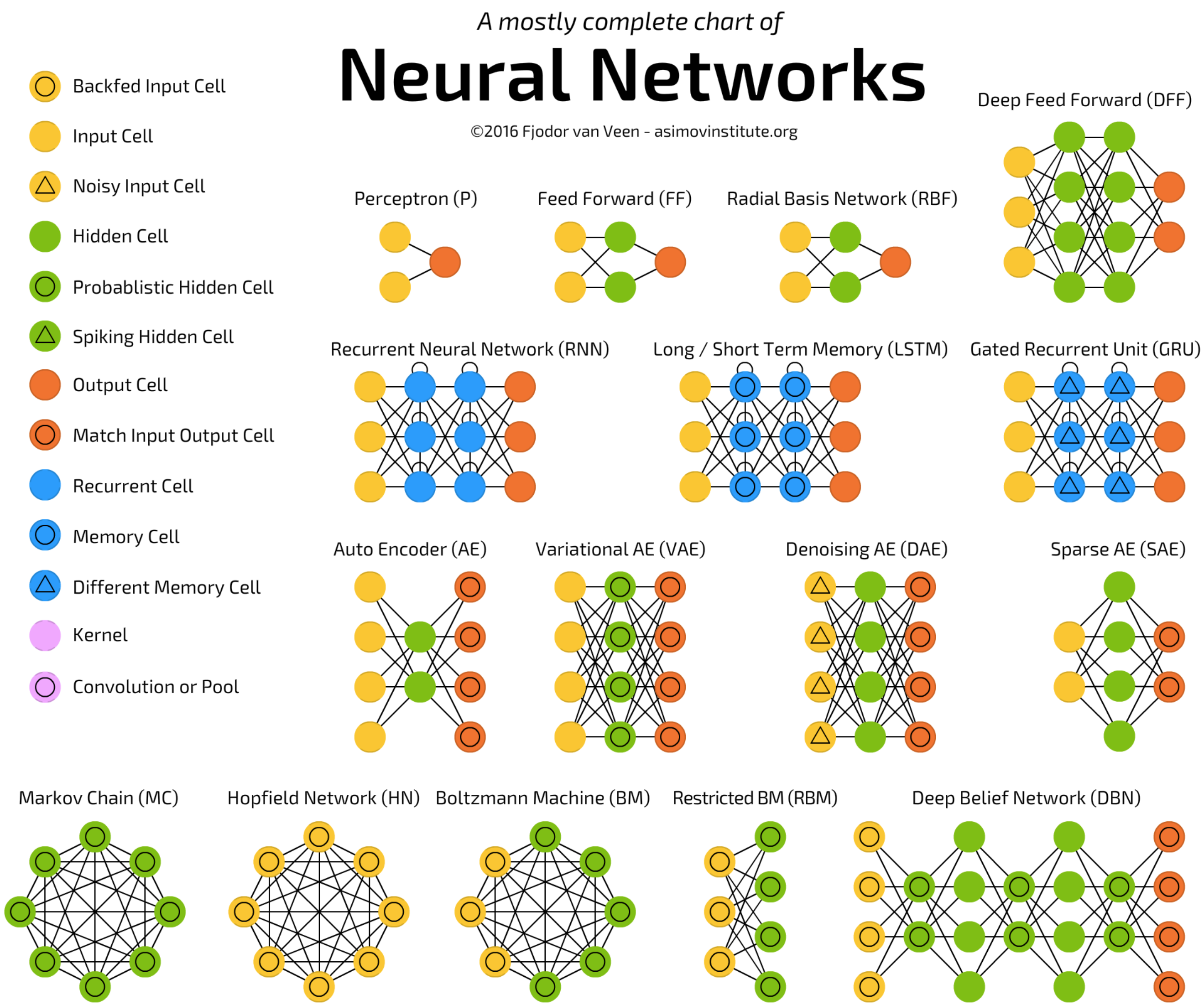
- **1940s: Neural networks were proposed**
- **1960s: Deep neural networks were proposed**
- **1989: Neural network for recognizing digits (LeNet)**
- **1990s: Hardware for shallow neural nets**
  - Example: Intel ETANN (1992)
- **2011: Breakthrough DNN-based speech recognition**
  - Microsoft real-time speech translation
- **2012: DNNs for vision supplanting traditional ML**
  - AlexNet for image classification
- **2014+: Rise of DNN accelerator research**
  - Examples: Neuflow, DianNao, etc.

Publications at Architecture Conferences

- **MICRO, ISCA, HPCA, ASPLOS**



So Many Neural Networks!



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

DNN Terminology 101

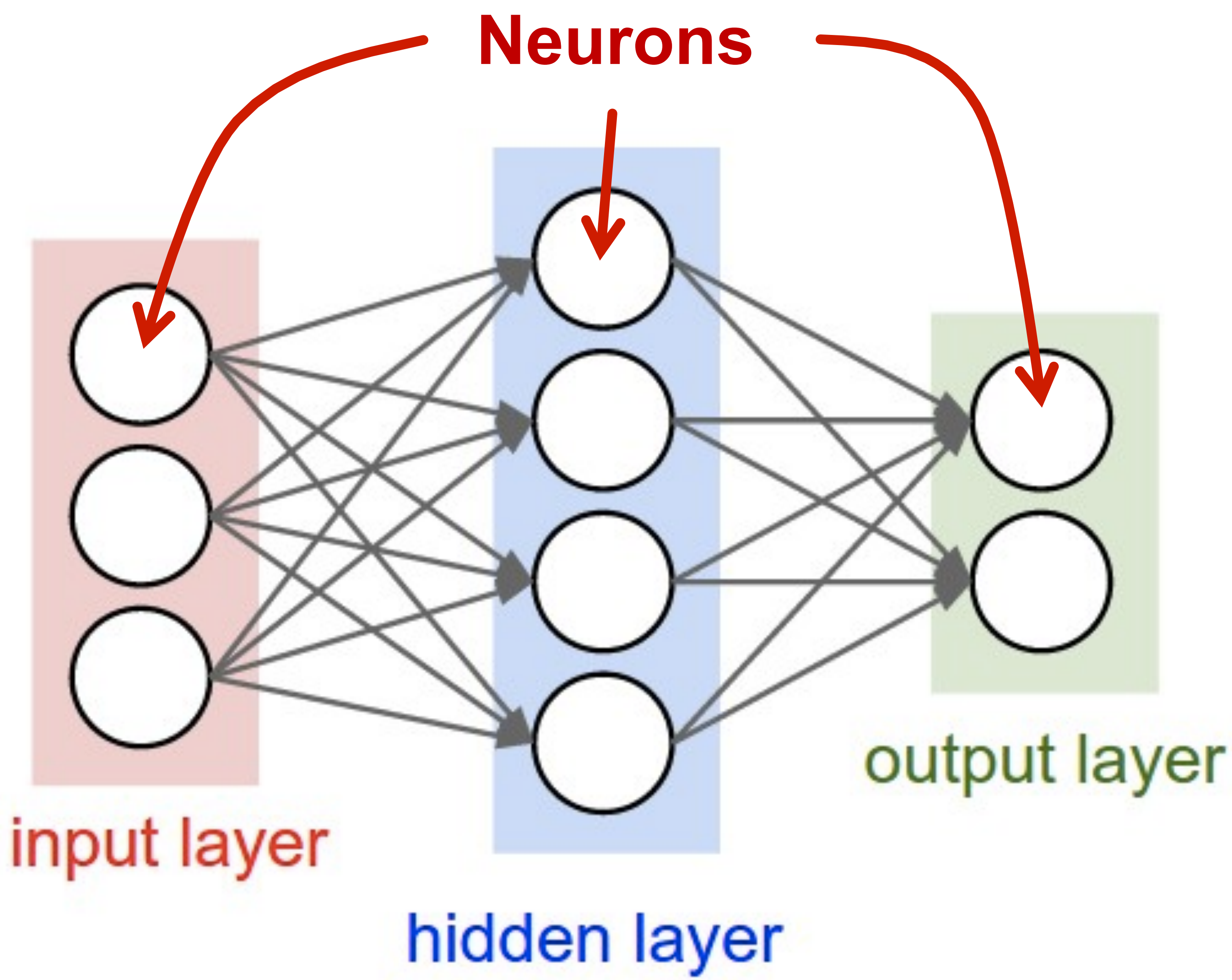


Image Source: Stanford

DNN Terminology 101

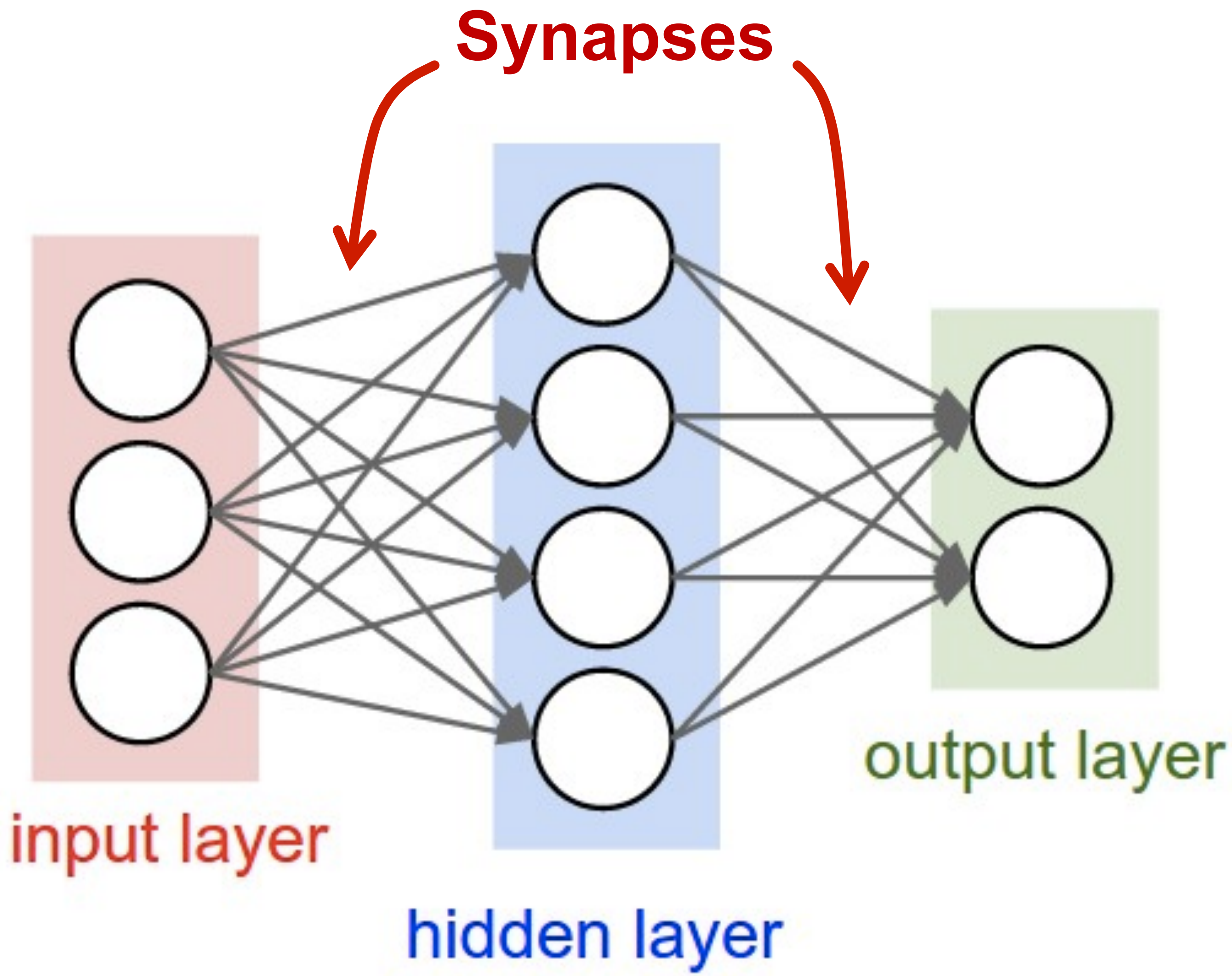


Image Source: Stanford

DNN Terminology 101

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

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

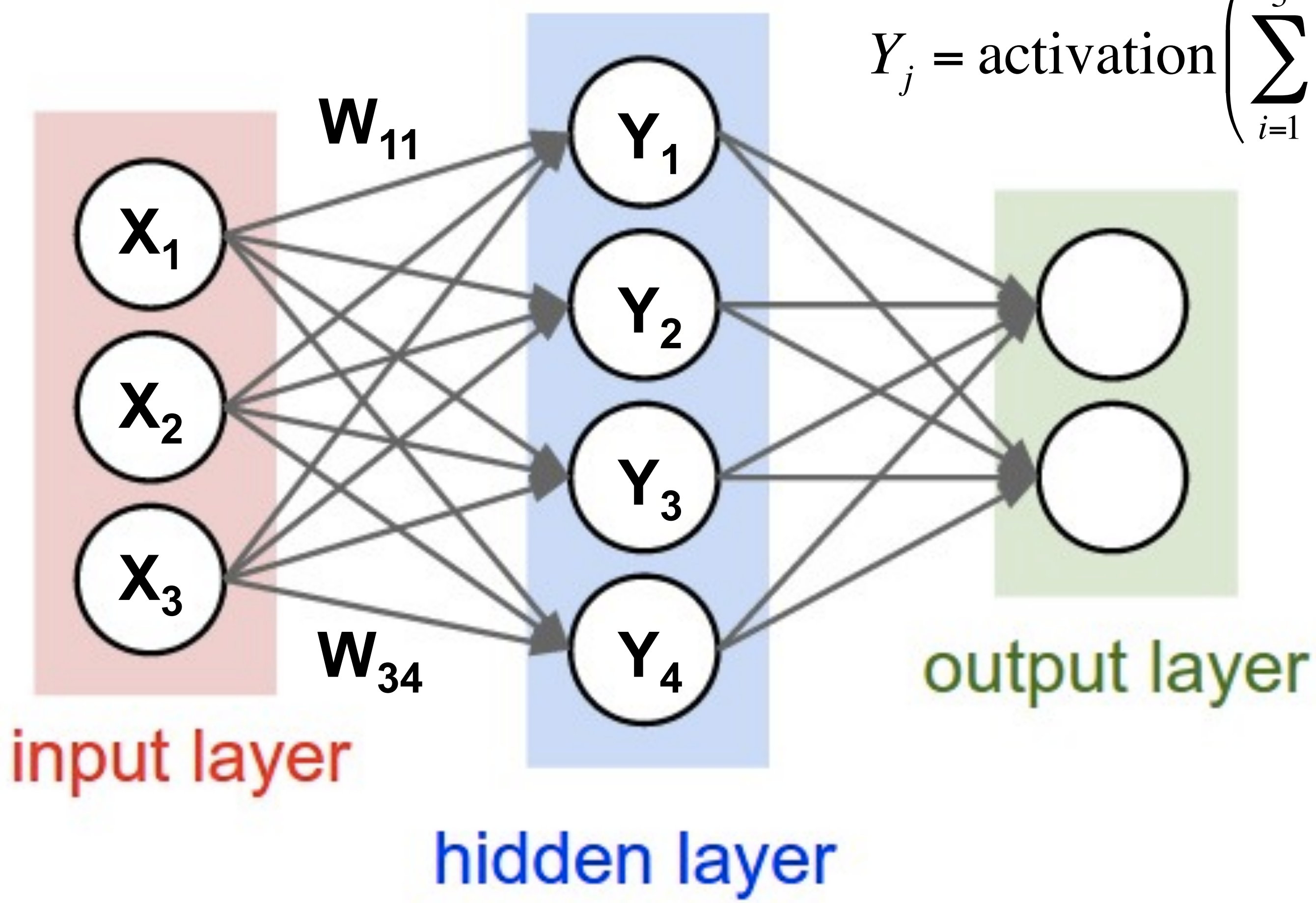


Image Source: Stanford

DNN Terminology 101

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

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

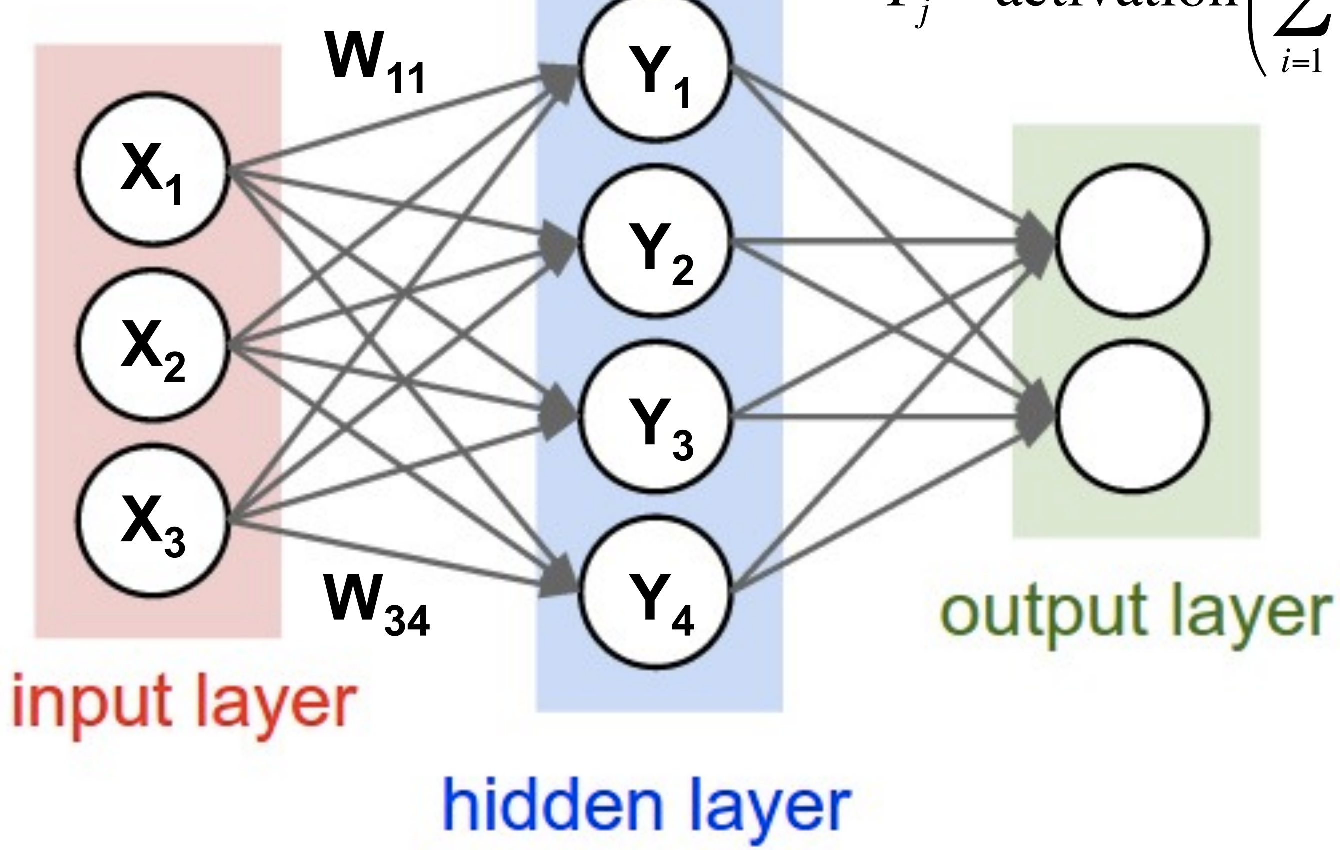


Image Source: Stanford

DNN Terminology 101

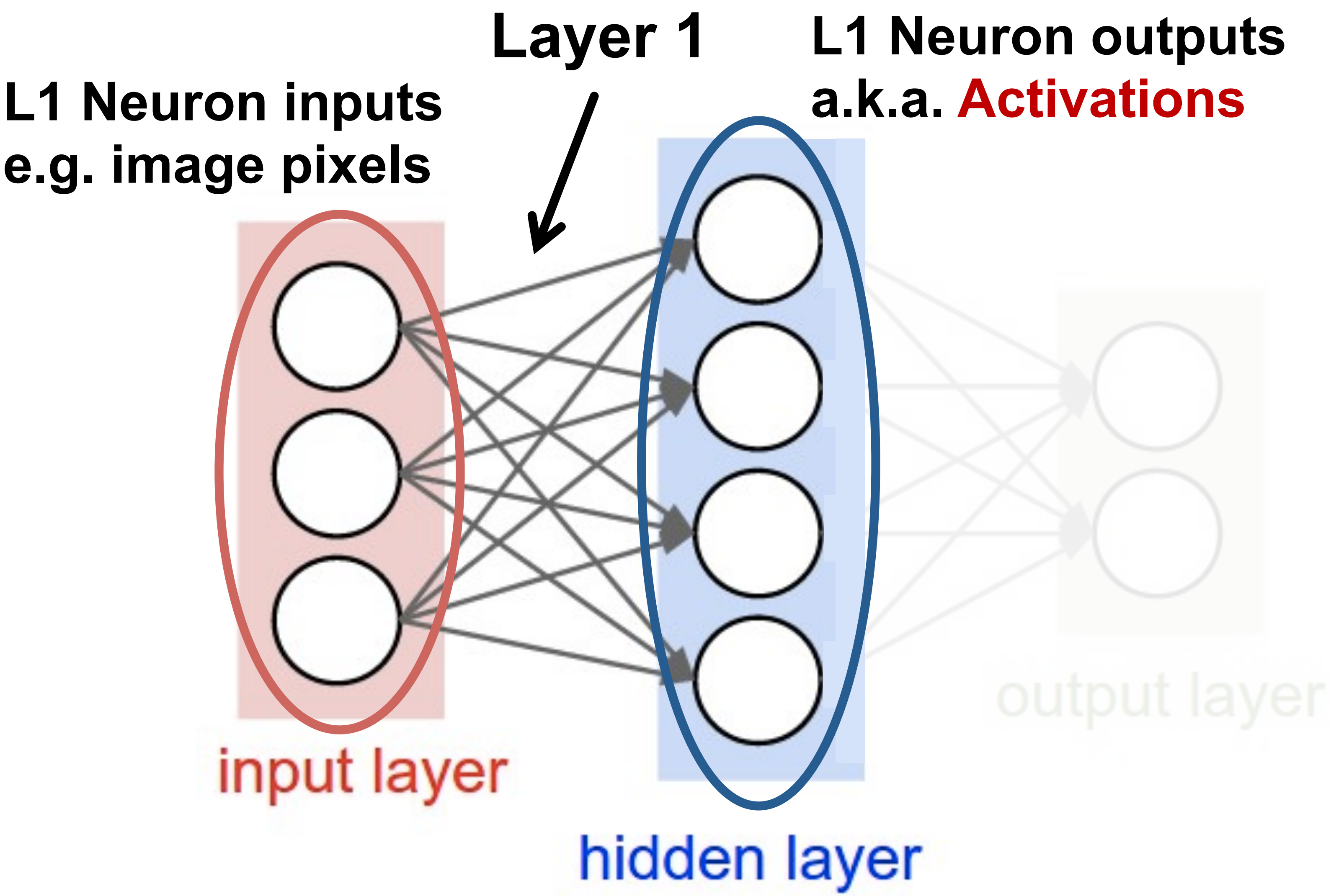


Image Source: Stanford

DNN Terminology 101

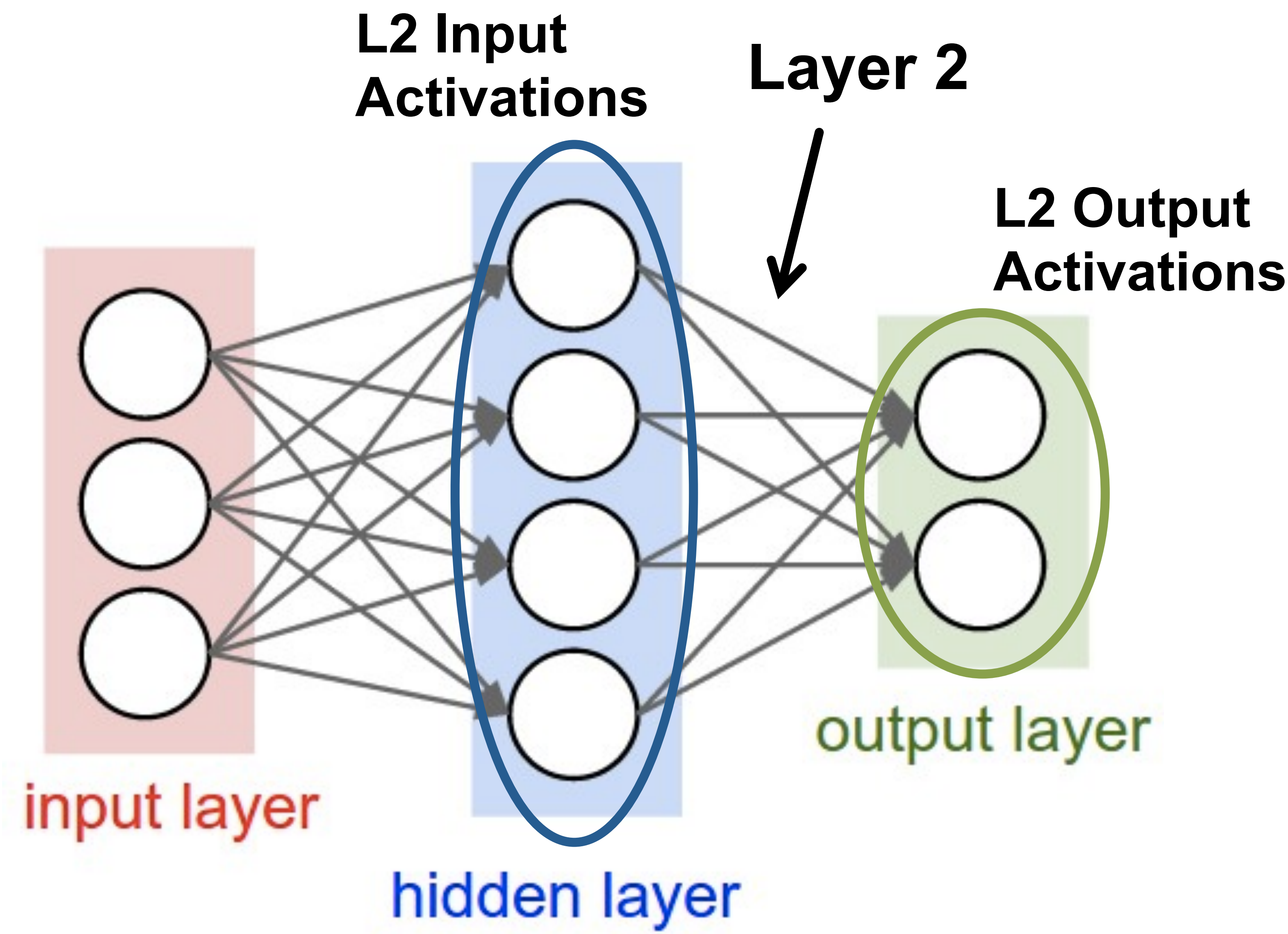


Image Source: Stanford

DNN Terminology 101

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

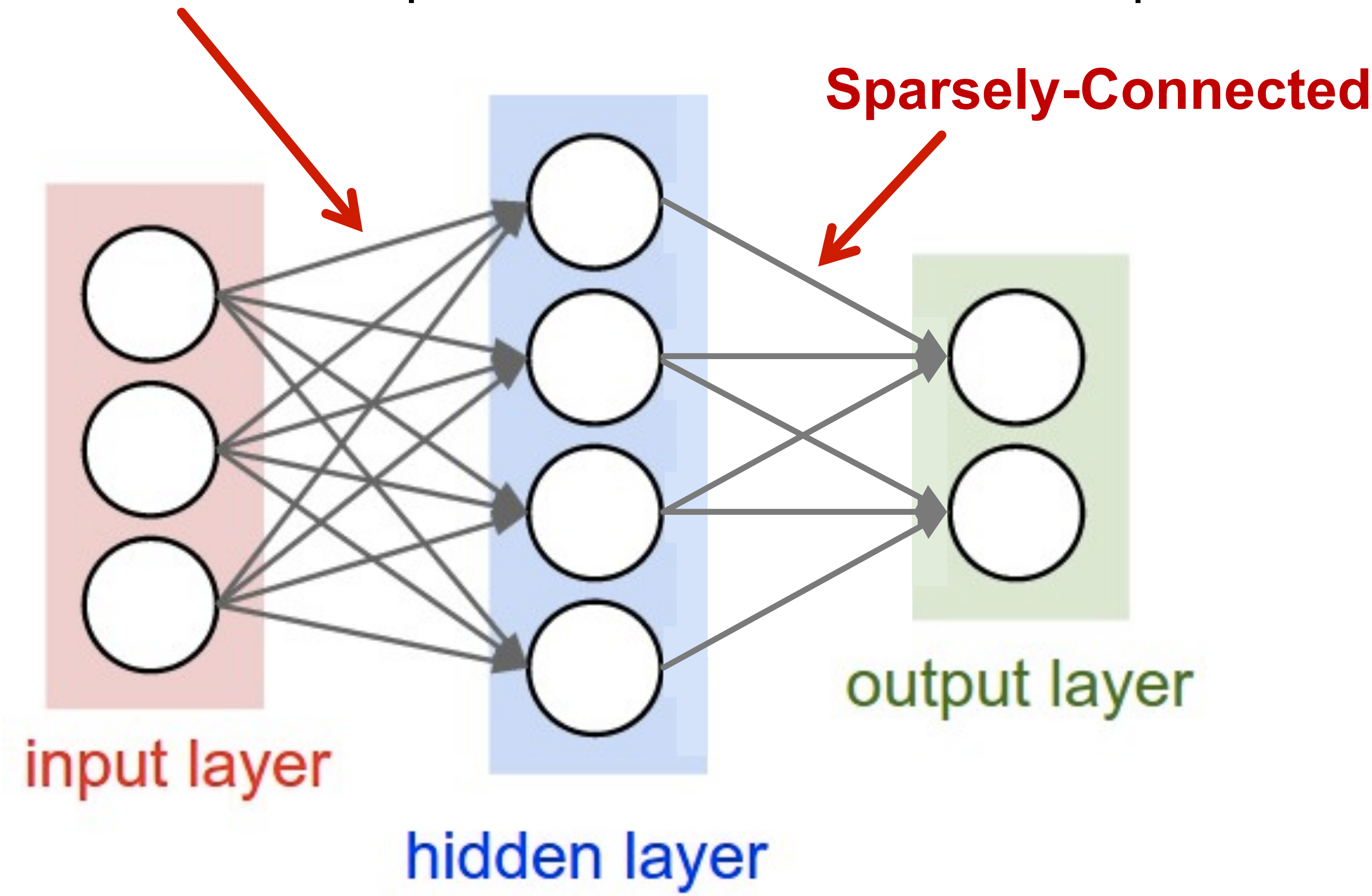


Image Source: Stanford

DNN Terminology 101

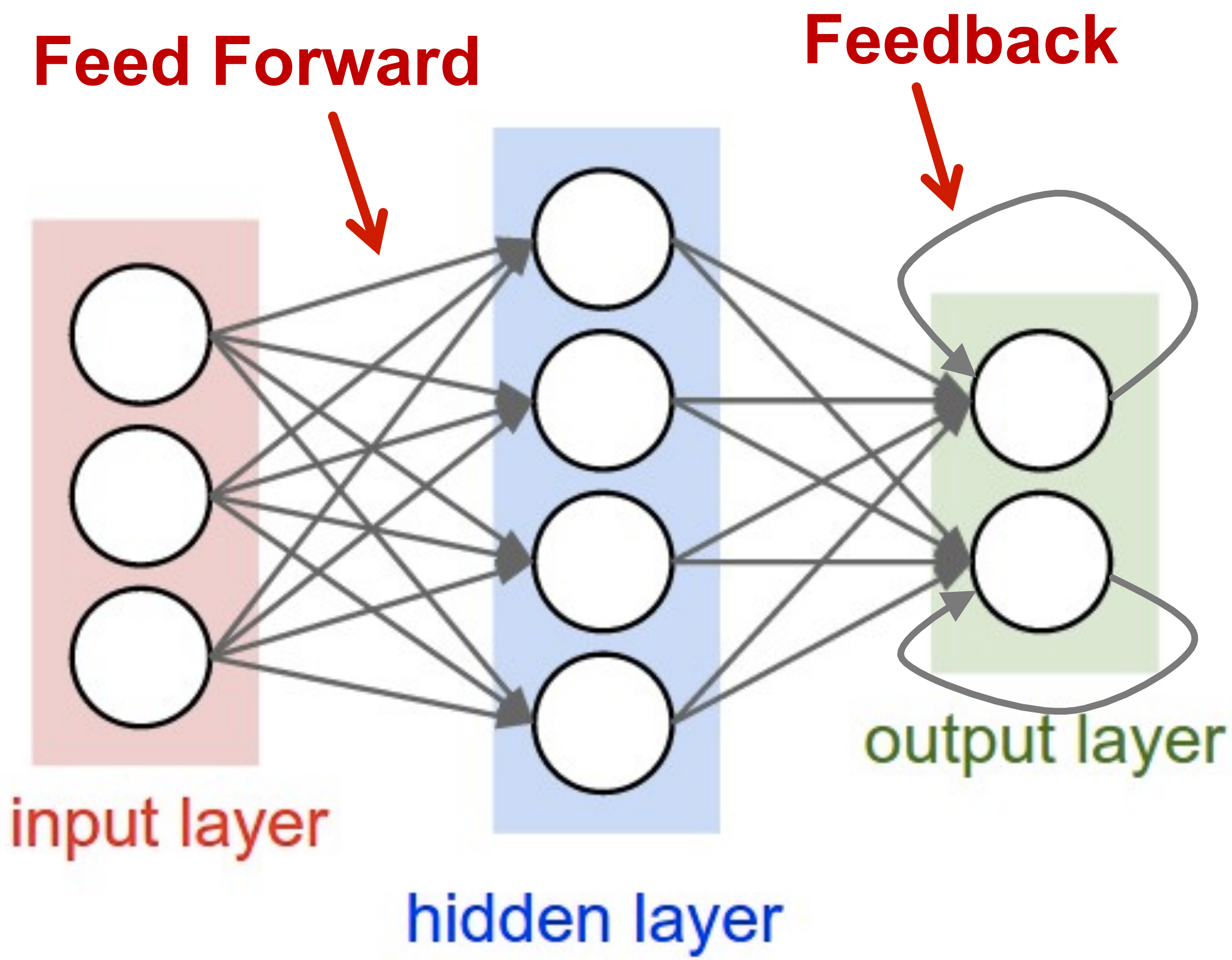


Image Source: Stanford

Popular Types of DNNs

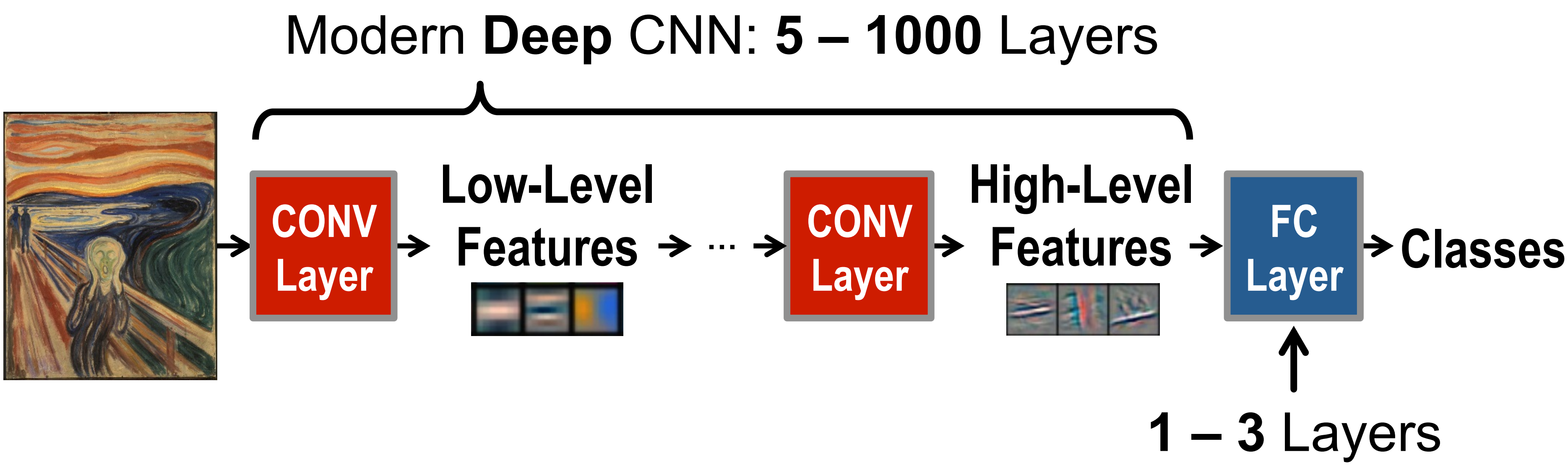
- **Fully-Connected NN**
  – feed forward, a.k.a. multilayer perceptron (MLP)
- **Convolutional NN (CNN)**
  – feed forward, sparsely-connected w/ weight sharing
- **Recurrent NN (RNN)**
  – feedback
- **Long Short-Term Memory (LSTM)**
  – feedback + storage

Inference vs. Training

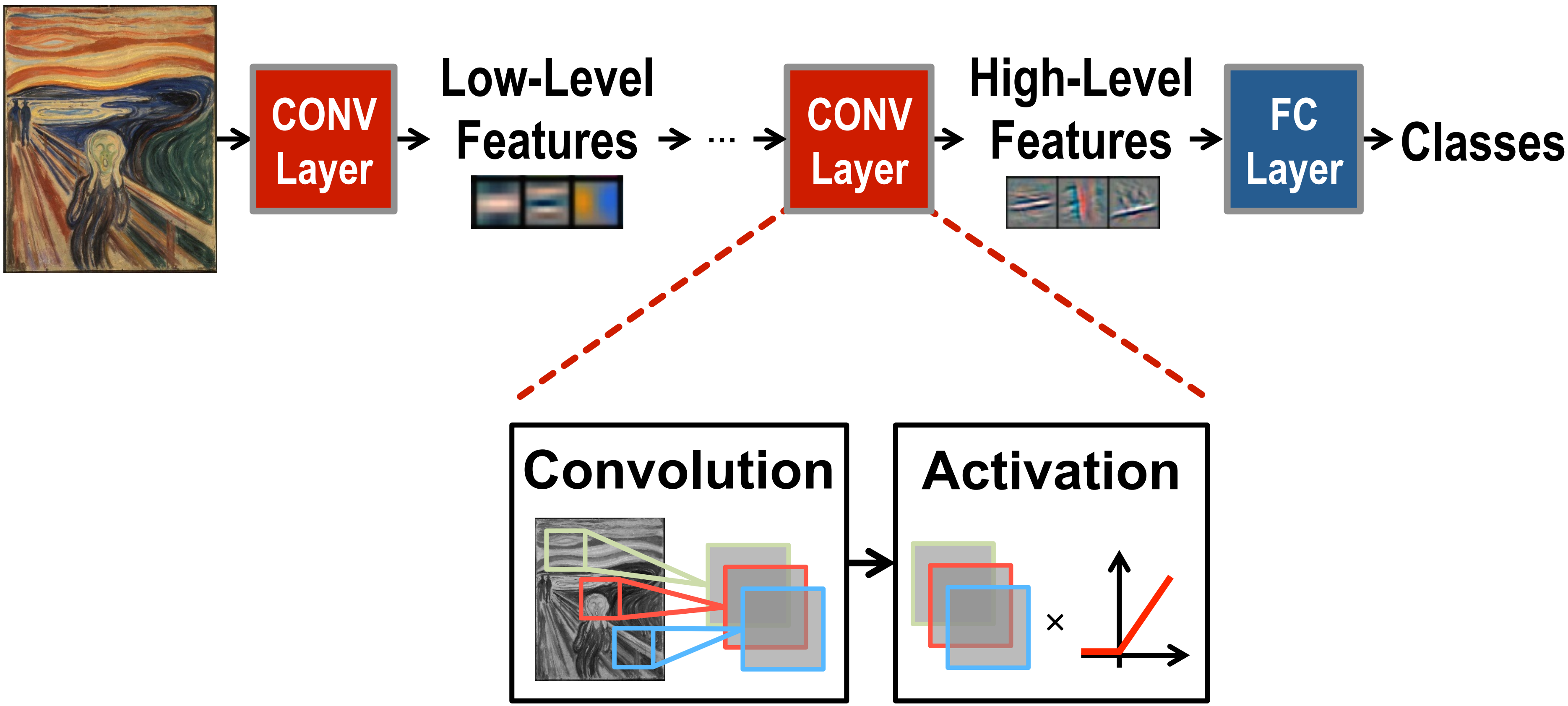
- **Training:** Determine weights
  - **Supervised:**
    - Training set has inputs and outputs, i.e., labeled
  - **Unsupervised:**
    - Training set is unlabeled
  - **Semi-supervised:**
    - Training set is partially labeled
  - **Reinforcement:**
    - Output assessed via rewards and punishments

- **Inference:** Apply weights to determine output

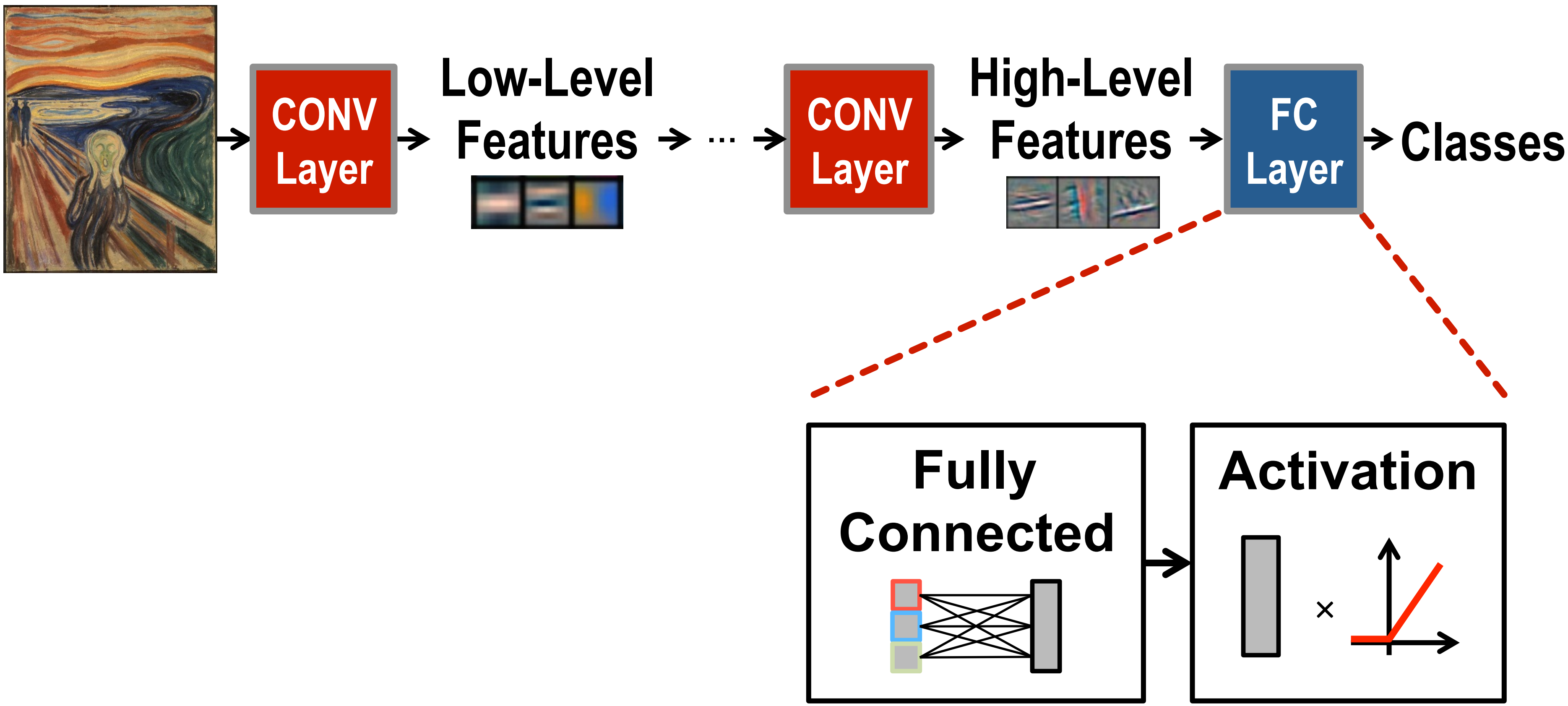
Deep Convolutional Neural Networks



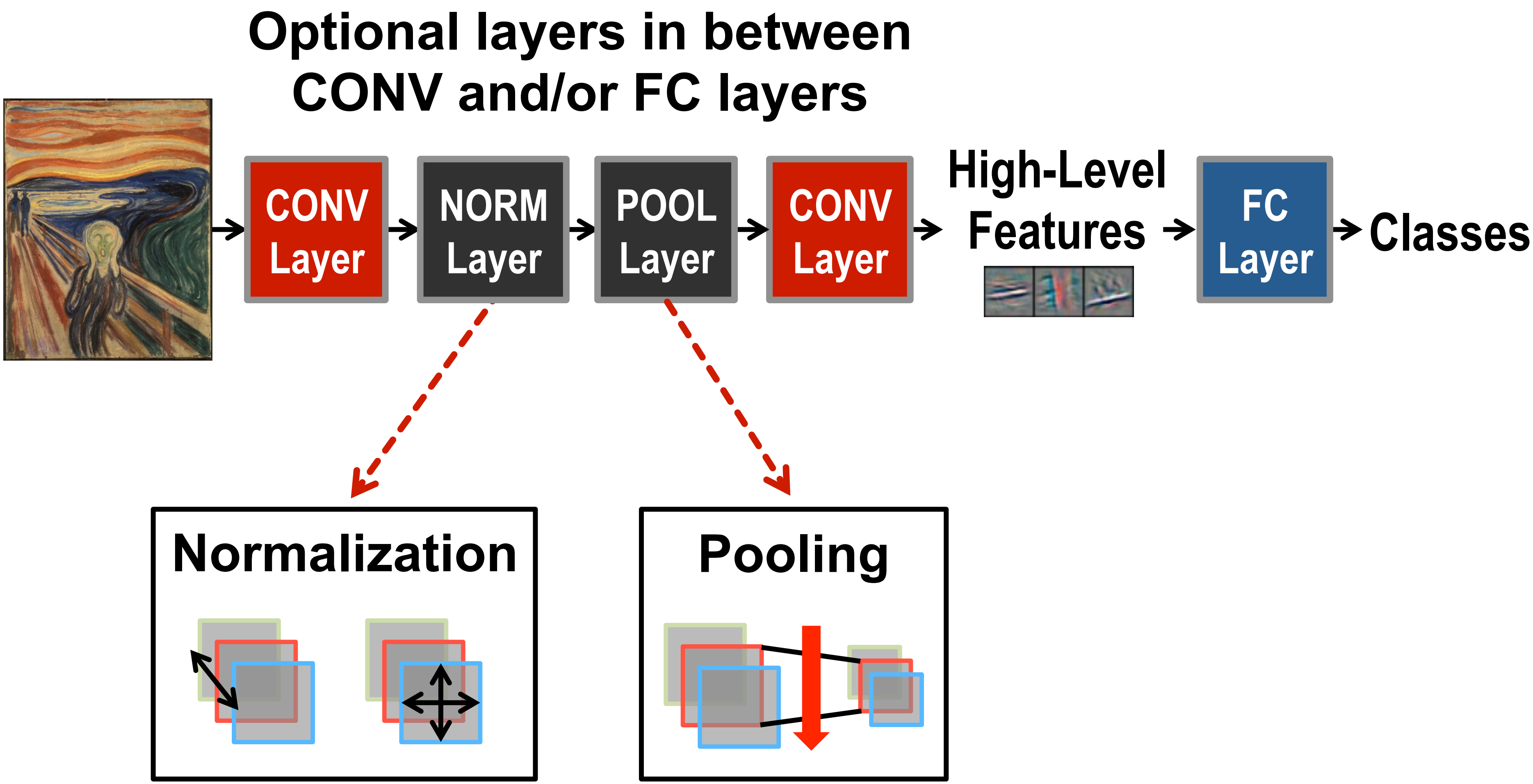
Deep Convolutional Neural Networks



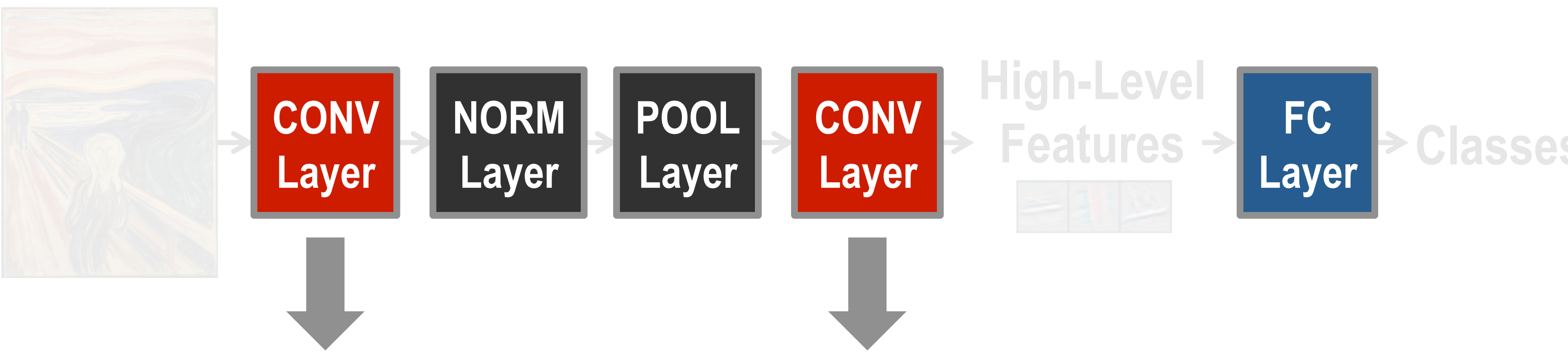
Deep Convolutional Neural Networks

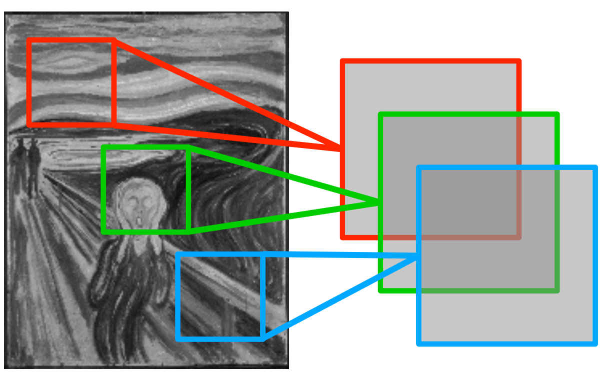


Deep Convolutional Neural Networks



Deep Convolutional Neural Networks



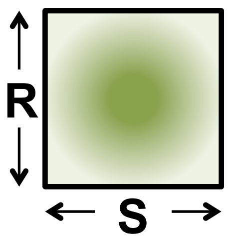


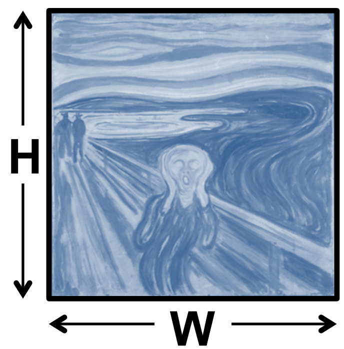
Convolutions account for more than 90% of overall computation, dominating **runtime** and **energy consumption**

Convolution (CONV) Layer

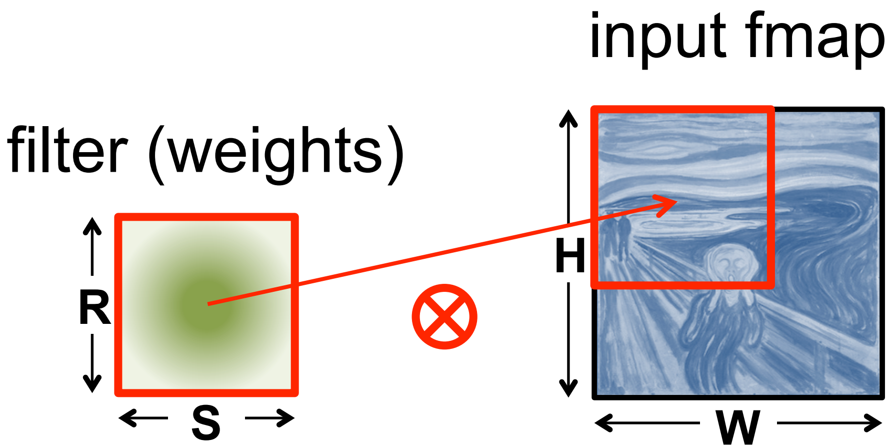
a plane of input activations
a.k.a. **input feature map (fmap)**

filter (weights)



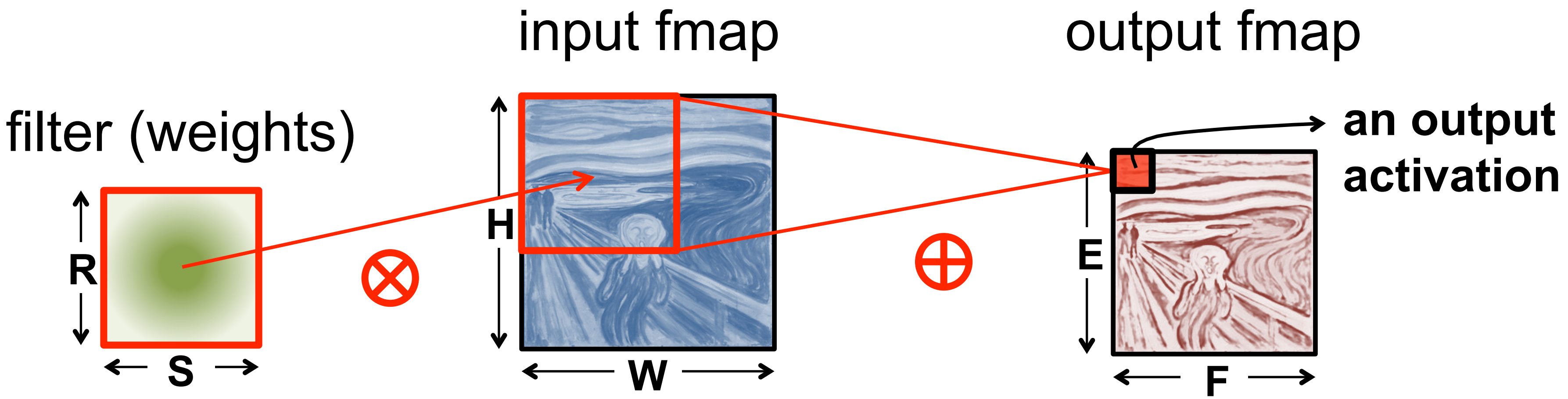


Convolution (CONV) Layer



Element-wise Multiplication

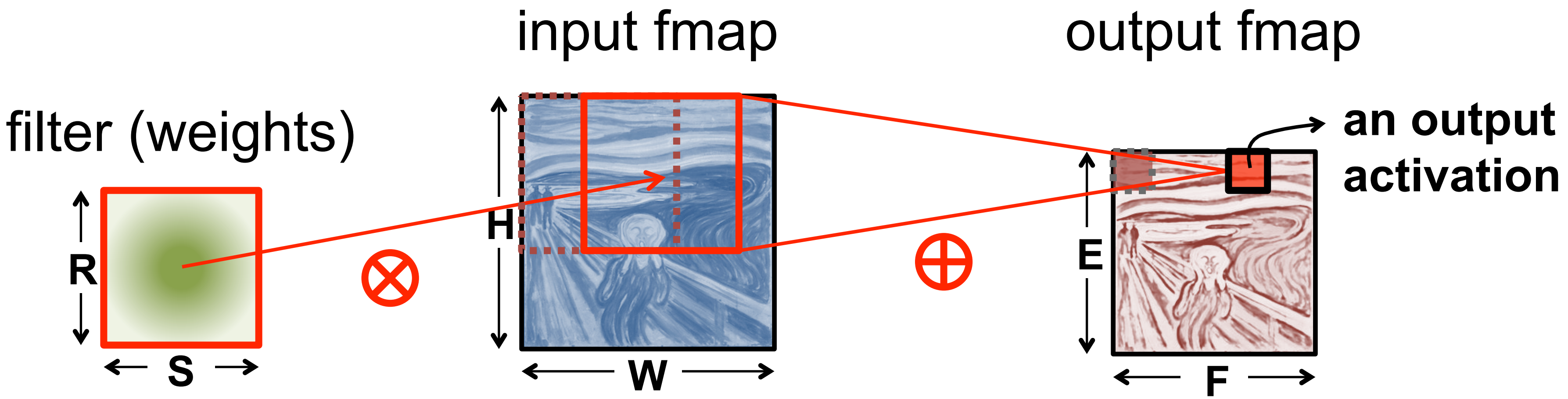
Convolution (CONV) Layer



Element-wise Multiplication

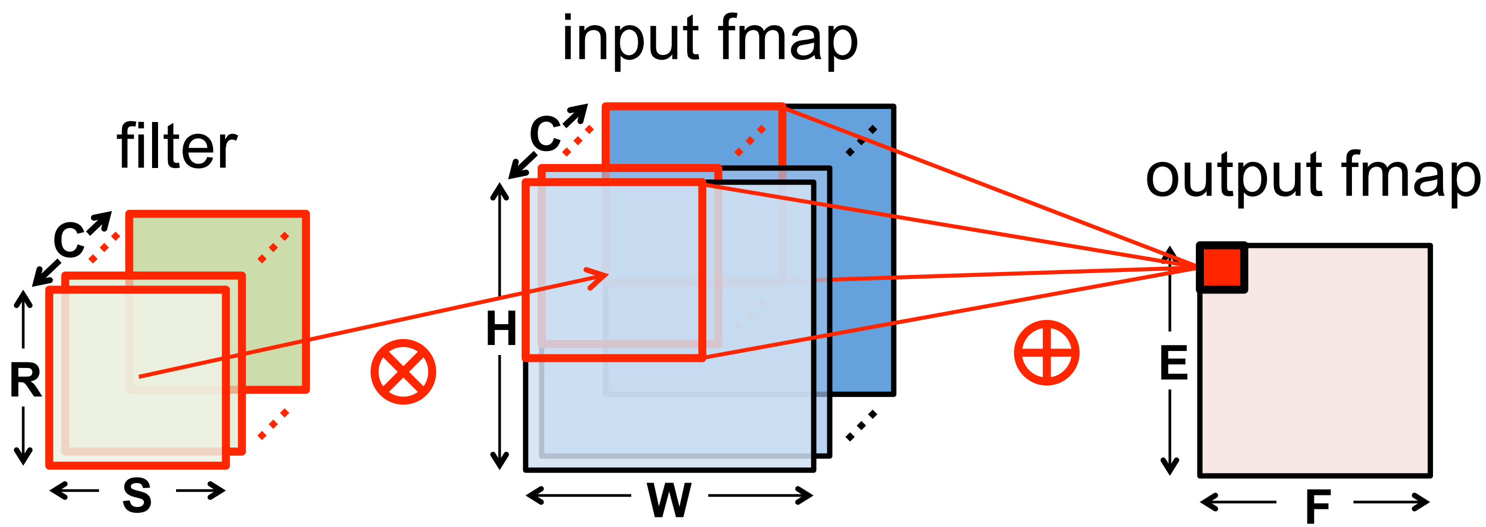
Partial Sum (psum) **Accumulation**

Convolution (CONV) Layer



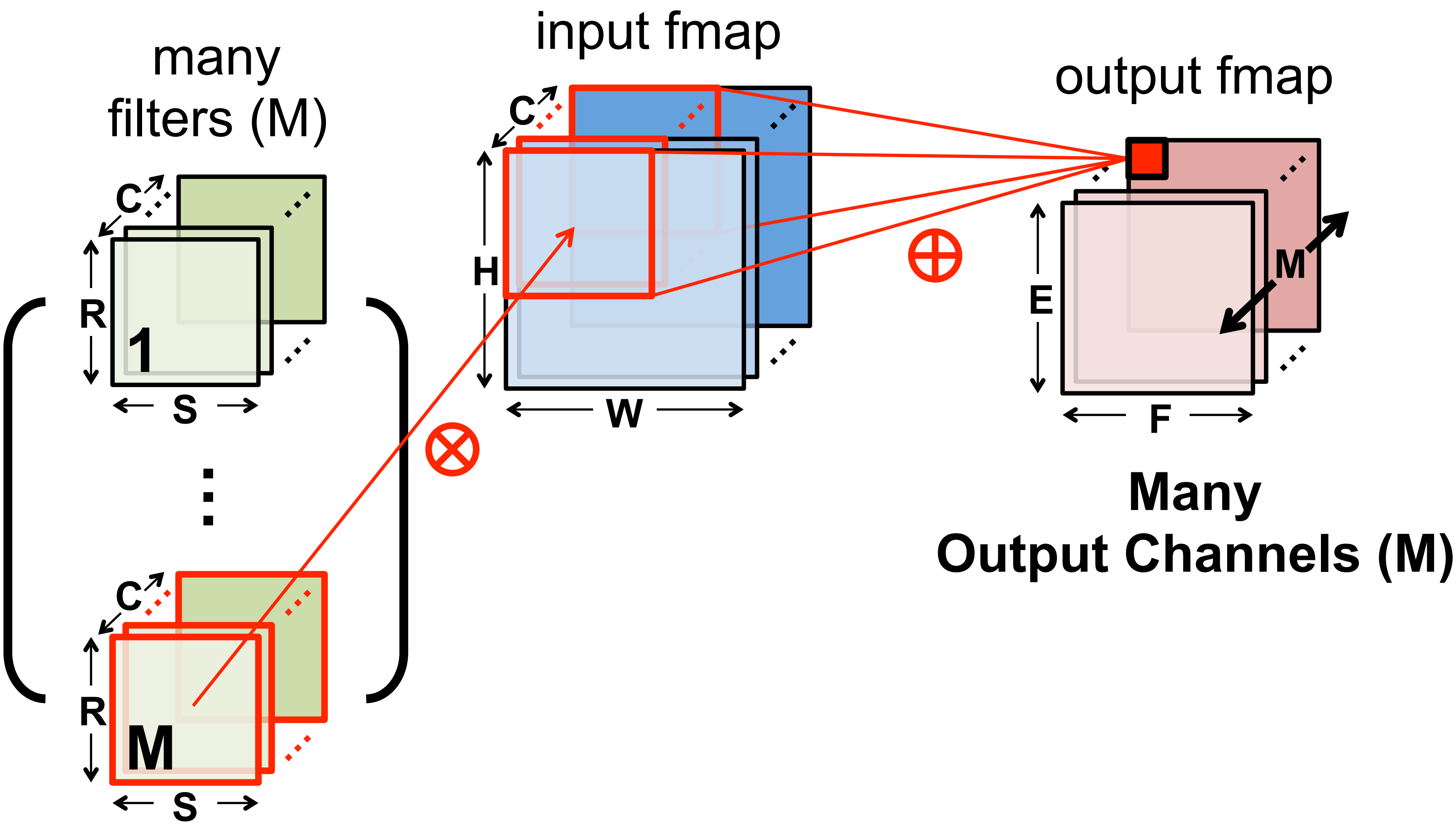
Sliding Window Processing

Convolution (CONV) Layer

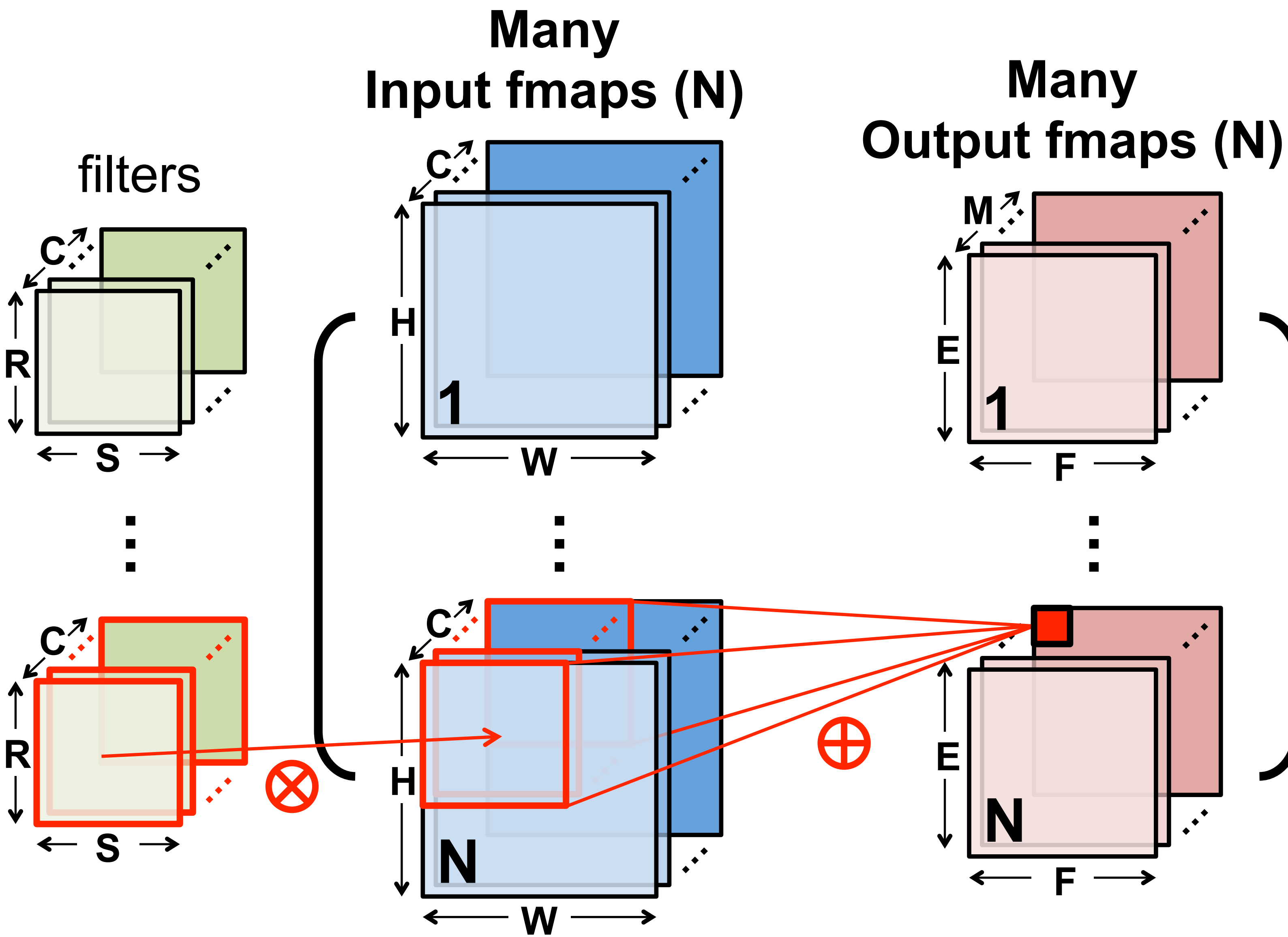


Many Input Channels (C)

Convolution (CONV) Layer



Convolution (CONV) Layer



CNN Decoder Ring

- **N – Number of input fmaps/output fmaps (batch size)**
- **C – Number of 2-D input fmaps /filters (channels)**
- **H – Height of input fmap (activations)**
- **W – Width of input fmap (activations)**
- **R – Height of 2-D filter (weights)**
- **S – Width of 2-D filter (weights)**
- **M – Number of 2-D output fmaps (channels)**
- **E – Height of output fmap (activations)**
- **F – Width of output fmap (activations)**

CONV Layer Tensor Computation

Output fmaps (O)

Biases (B)

Input fmaps (I)

Filter weights (W)

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

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

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

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:

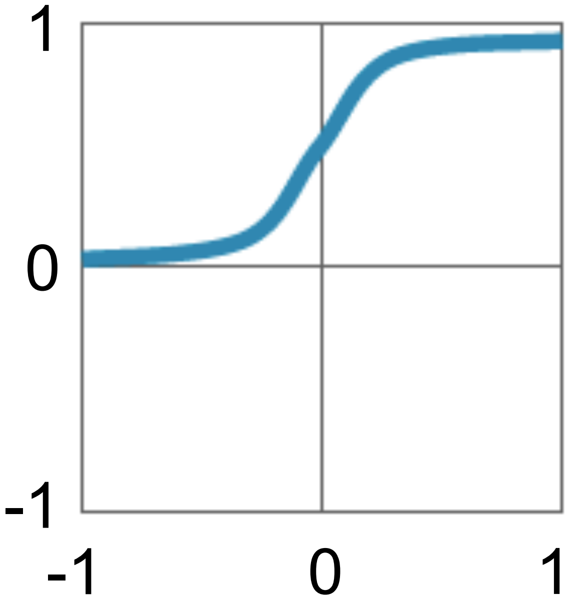
for each output fmap value

convolve a window and apply activation

```
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] × W[m][k][i][j];
                        }
                    }
                }

                O[n][m][x][y] = Activation(O[n][m][x][y]);
            }
        }
    }
}
```

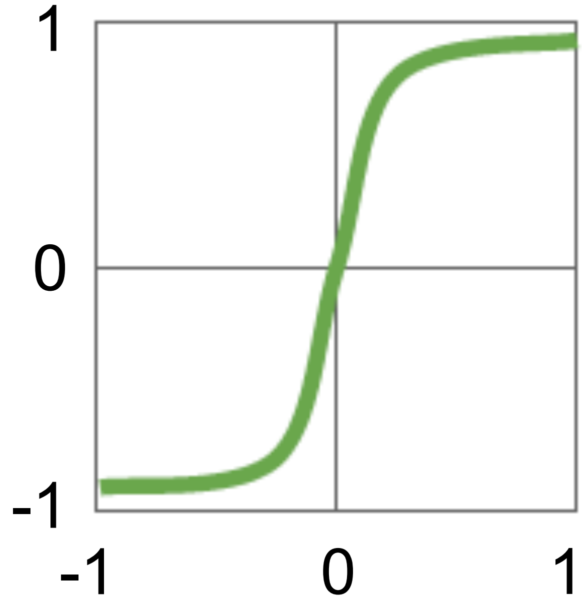
Traditional Activation Functions

Sigmoid



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

Hyperbolic Tangent

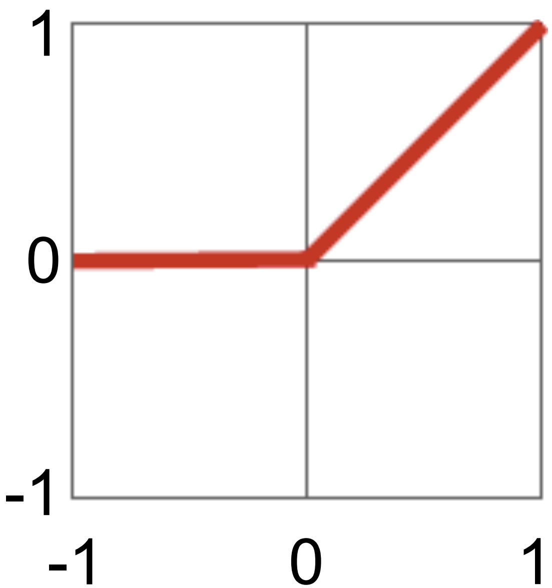


$$y=(e^{x}-e^{-x})/(e^{x}+e^{-x})$$

Image Source: Caffe Tutorial

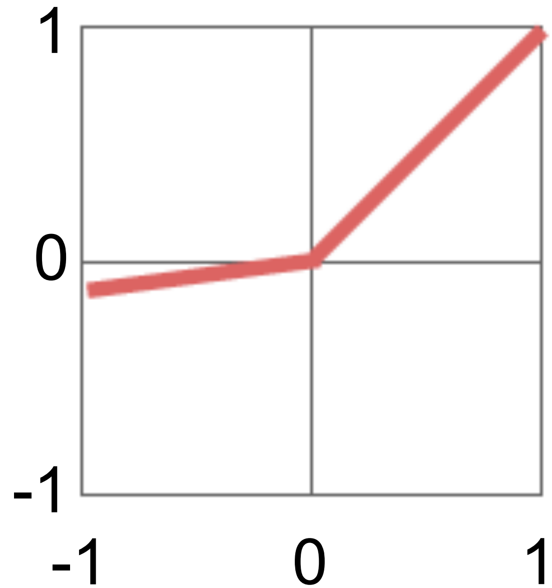
Modern Activation Functions

Rectified Linear Unit (ReLU)



$$y=\max(0,x)$$

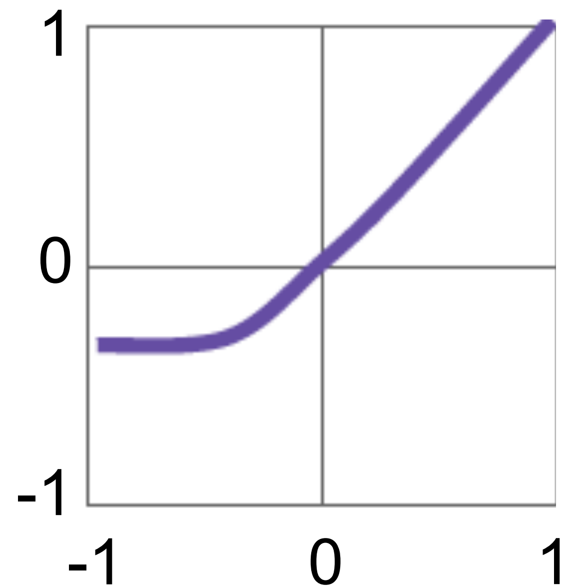
Leaky ReLU



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

α = small const. (e.g. 0.1)

Exponential LU

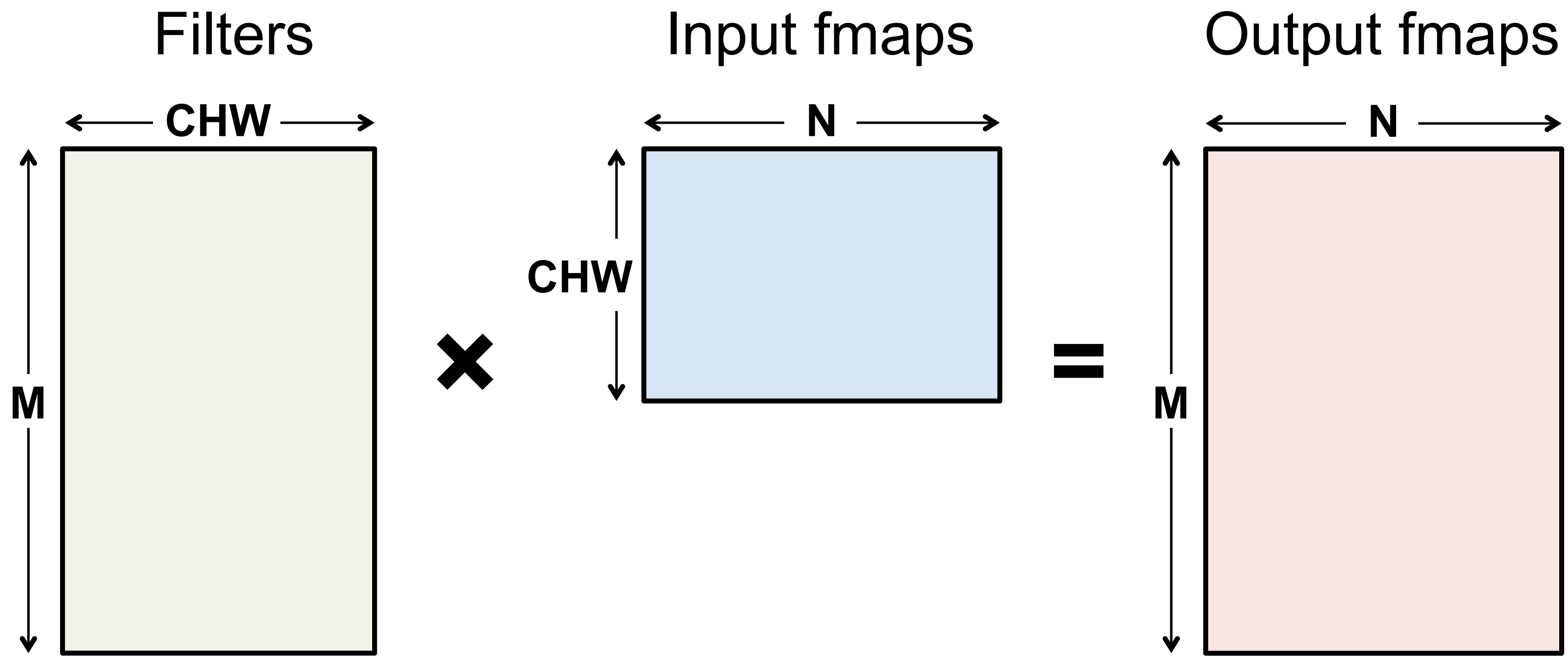


$$y=\begin{cases} x, & x\geq 0 \\ \alpha(e^x-1), & x<0 \end{cases}$$

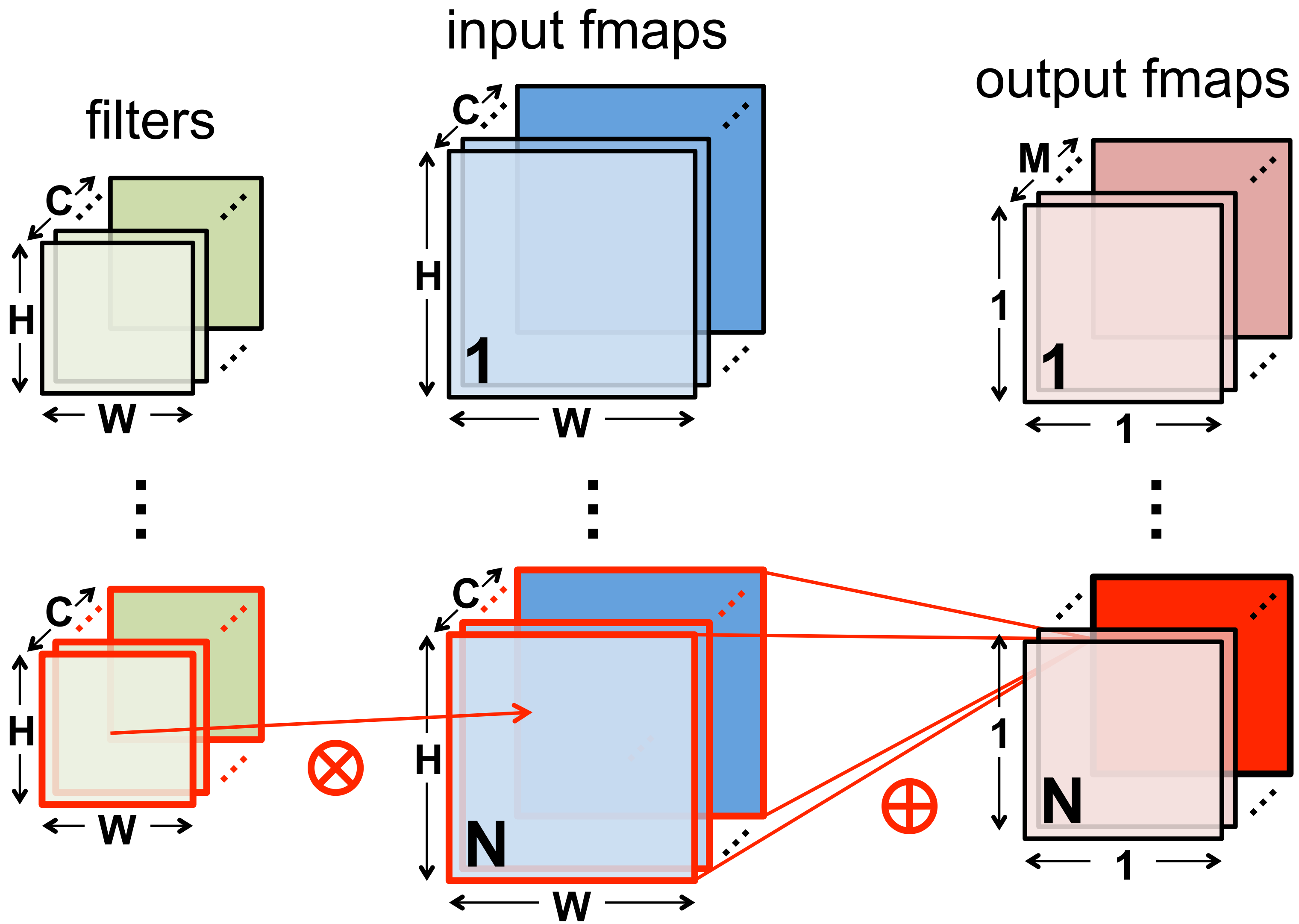
Image Source: Caffe Tutorial

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**

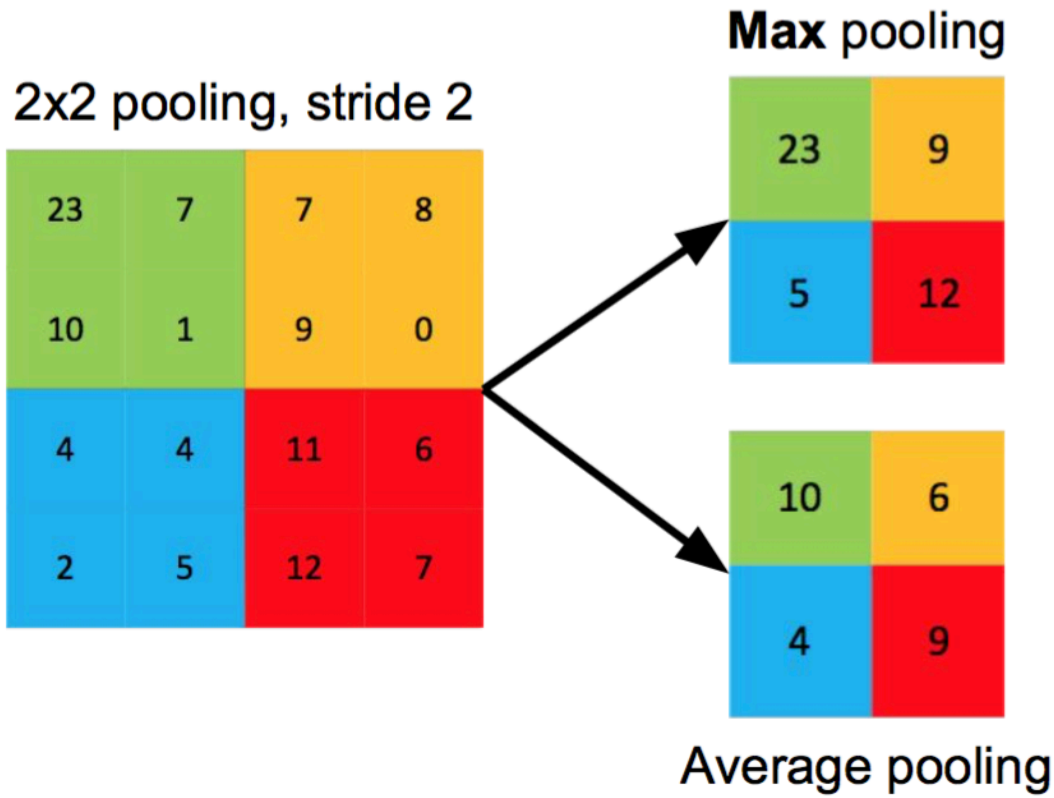


FC Layer – from CONV Layer POV



Pooling (POOL) Layer

- Reduce resolution of each channel independently
- Overlapping or non-overlapping → depending on stride



Increases translation-invariance and noise-resilience

Image Source: Caffe Tutorial

POOL Layer Implementation

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

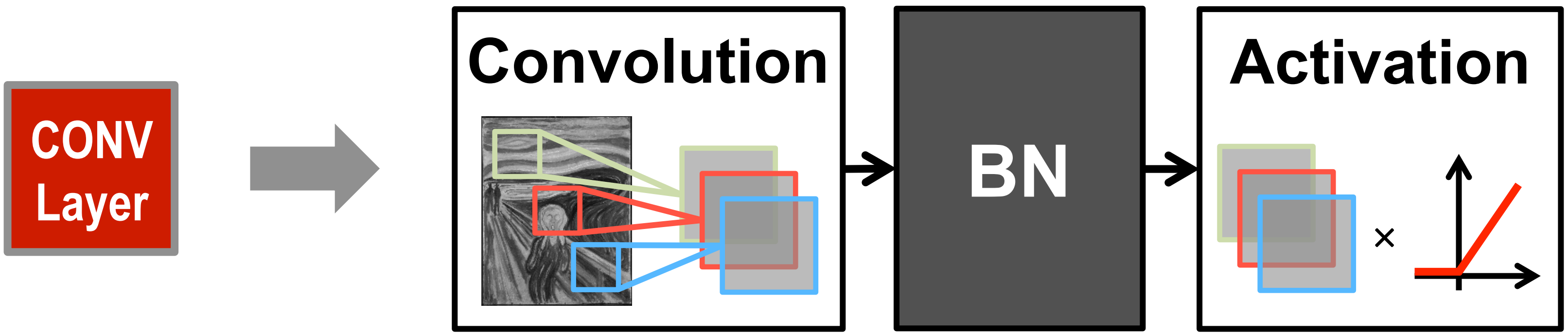
```
for (n=0; n<N; n++) {
    for (m=0; m<M; m++) {
        for (x=0; x<F; x++) {
            for (y=0; y<E; y++) {

                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

data mean

learned scale factor

$$y = \frac{x - \mu}{\sqrt{\sigma^2 + \epsilon}}\gamma + \beta$$

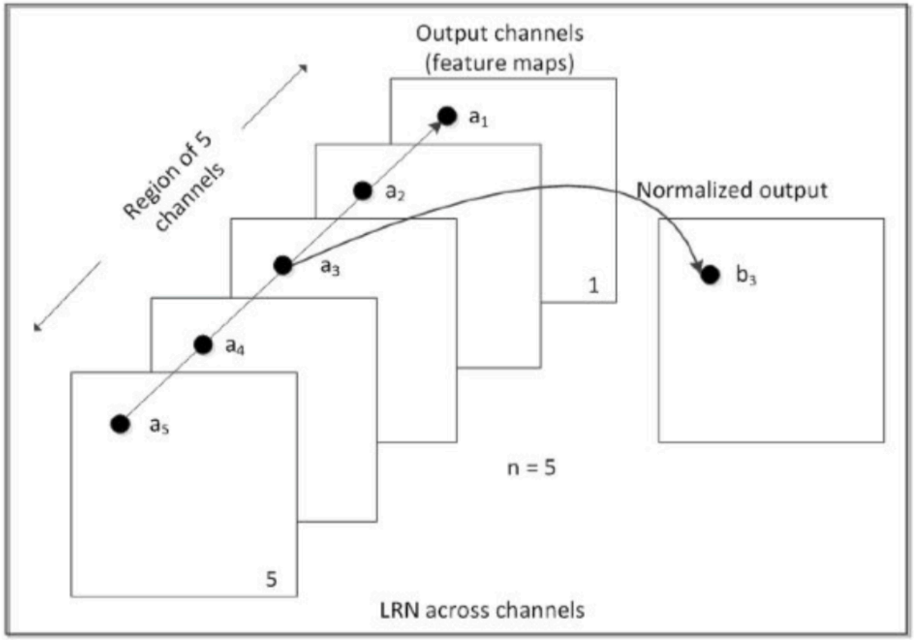
learned shift factor

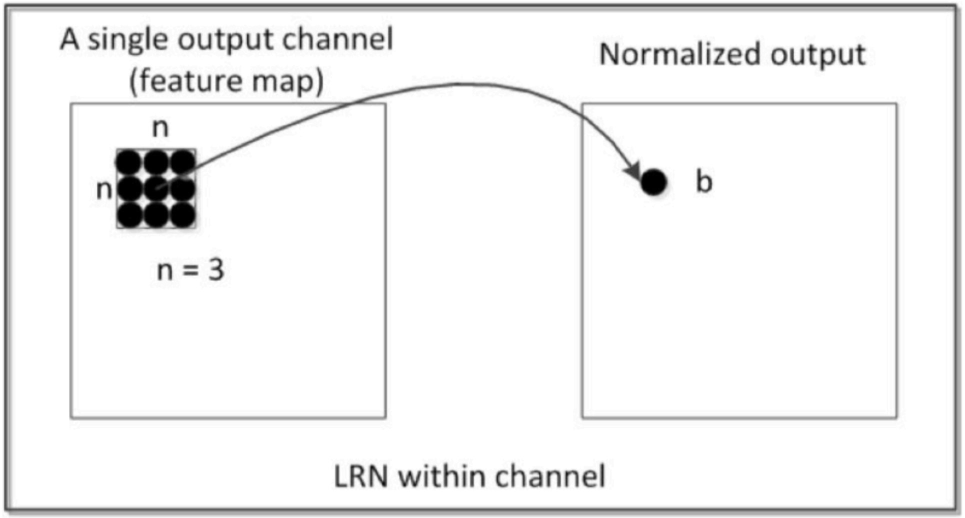
data std. dev.

small const. to avoid numerical problems

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