ECE 598NSG/498NSU
Deep Learning in Hardware
Fall 2020

Deep Learning – An Introduction

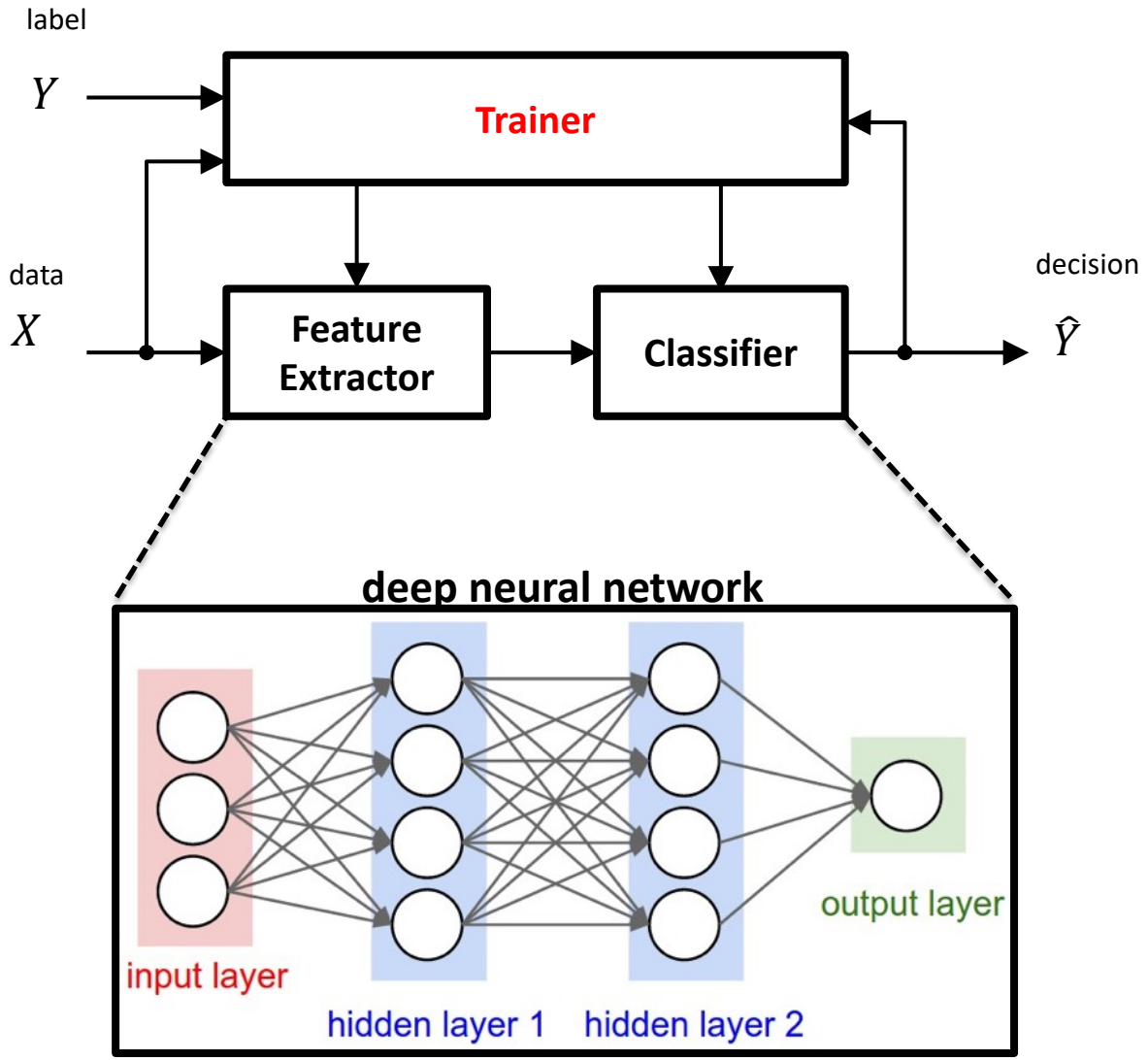
Naresh Shanbhag
Department of Electrical and Computer Engineering
University of Illinois at Urbana-Champaign

http://shanbhag.ece.uiuc.edu

Today

- Fundamental inference tasks
- DNN – structure
- DNN – function
  - linear predictor

A Deep Learning System



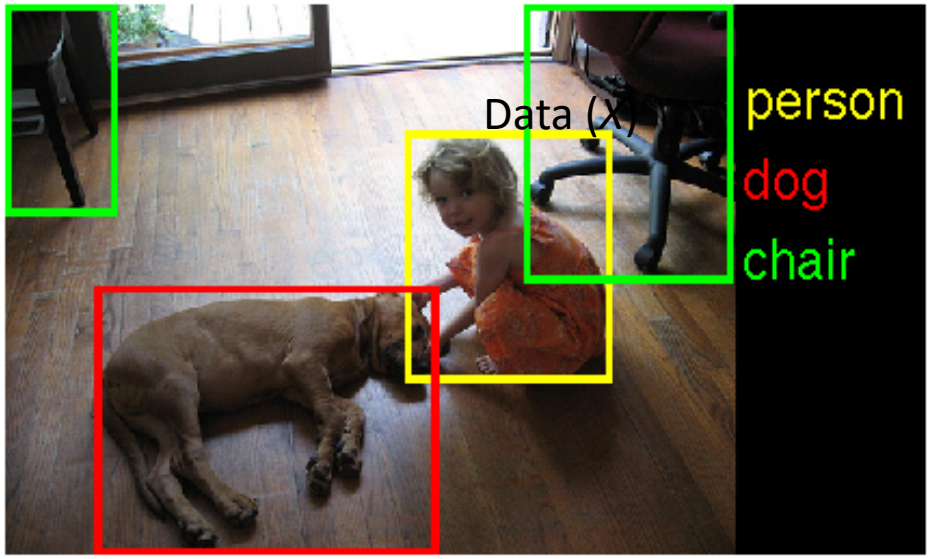
(training phase)

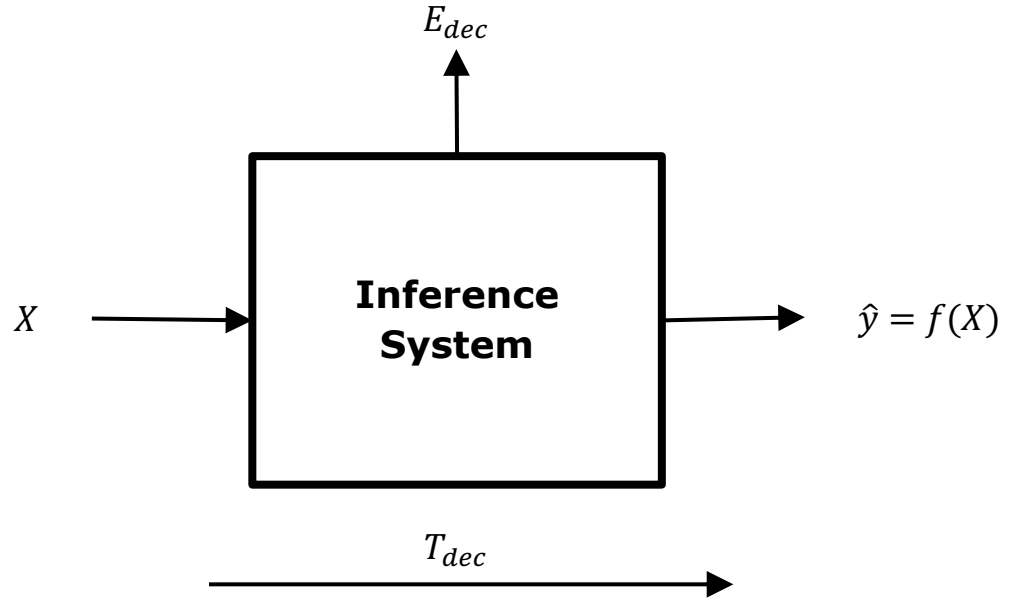
(test phase)

- in spite of its recent popularity → DNNs are old (since 90s)
- precursors to DNNs → MADALINE and multi-layer perceptrons
- resurgence of DNNs due to the availability of large datasets and computational power
- a few deep-rooted drawbacks of DNNs:
  - lack of a solid underlying theory, e.g., no Wiener-Hopf solution
  - interpretability of results is difficult
  - very high complexity
- drawbacks overlooked (in a very 'unengineering' fashion) today due their 'magical' performance on some very complex tasks

Fundamental Inference Tasks

What is Inference?

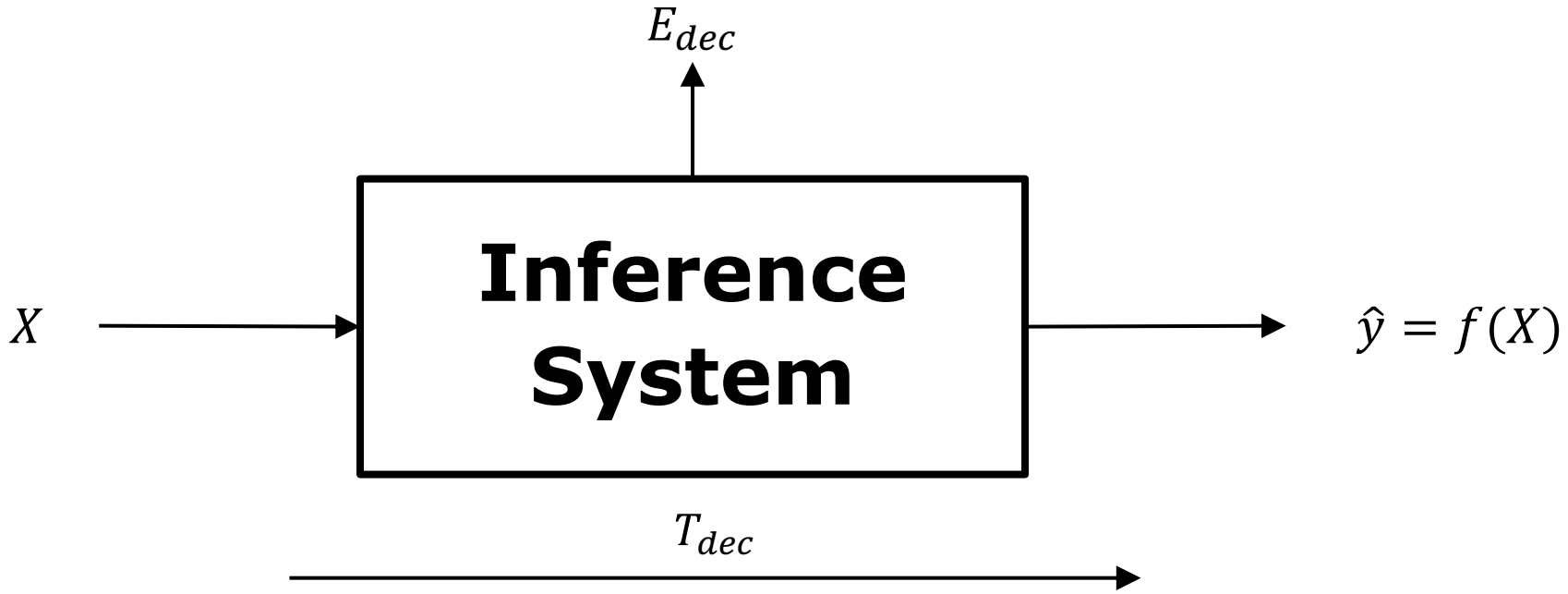




- E.g., identify (box) and recognize (name) objects
- Others?

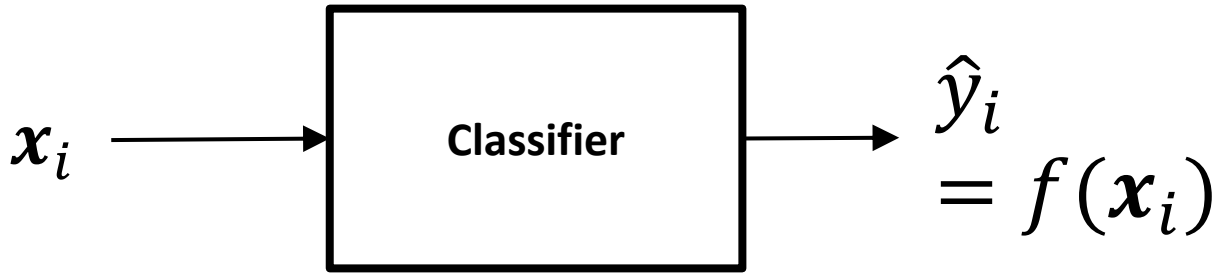
- Energy cost: E_{dec} (pJ/decision)
- Latency cost: T_{dec} (μs/decision)

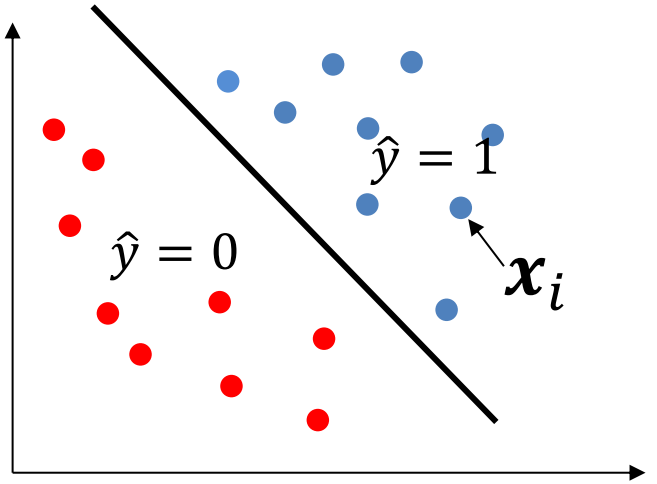
Two Key Inference Tasks



- Classification - $\hat{y}$ is discrete-valued (1-of-M classes)
- Regression - $\hat{y}$ is continuous-valued ($\hat{y} \in \mathbb{R}$)

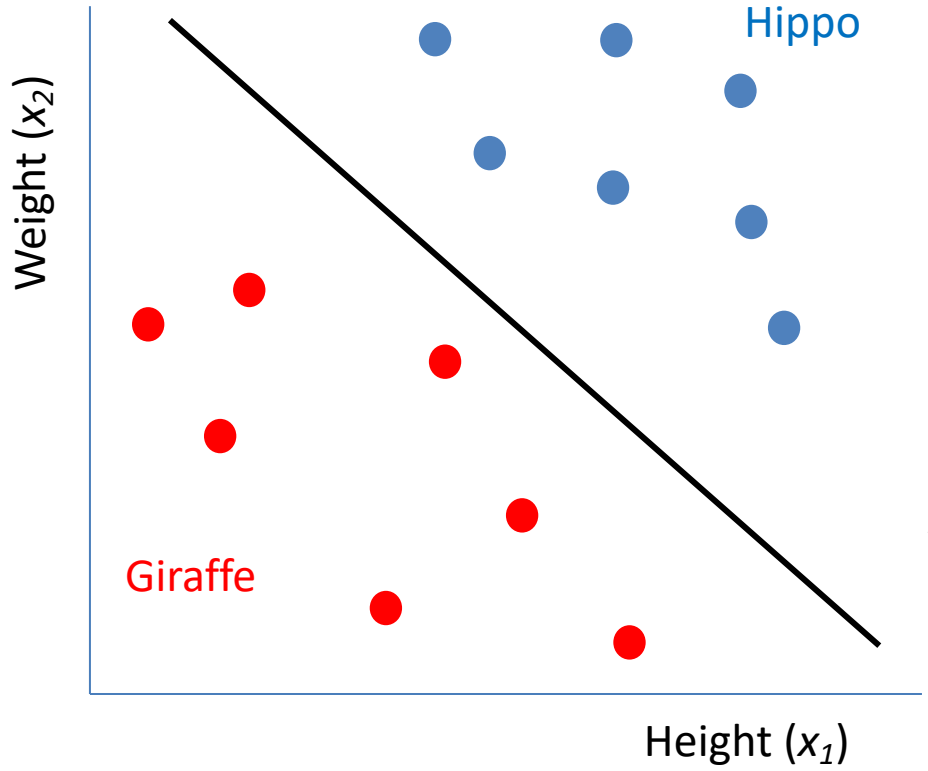
Classification





- assigning a class label $\hat{y}_i \in \{\mathcal{C}_0, \mathcal{C}_1, \ldots, \mathcal{C}_{M-1}\}$ to data $\boldsymbol{x}_i$
- class labels are discrete – binary or multi-class
- data $\boldsymbol{x}_i$ can be discrete or continuous, scalar or vector
- function $f(\boldsymbol{x}_i)$ can be linear or non-linear

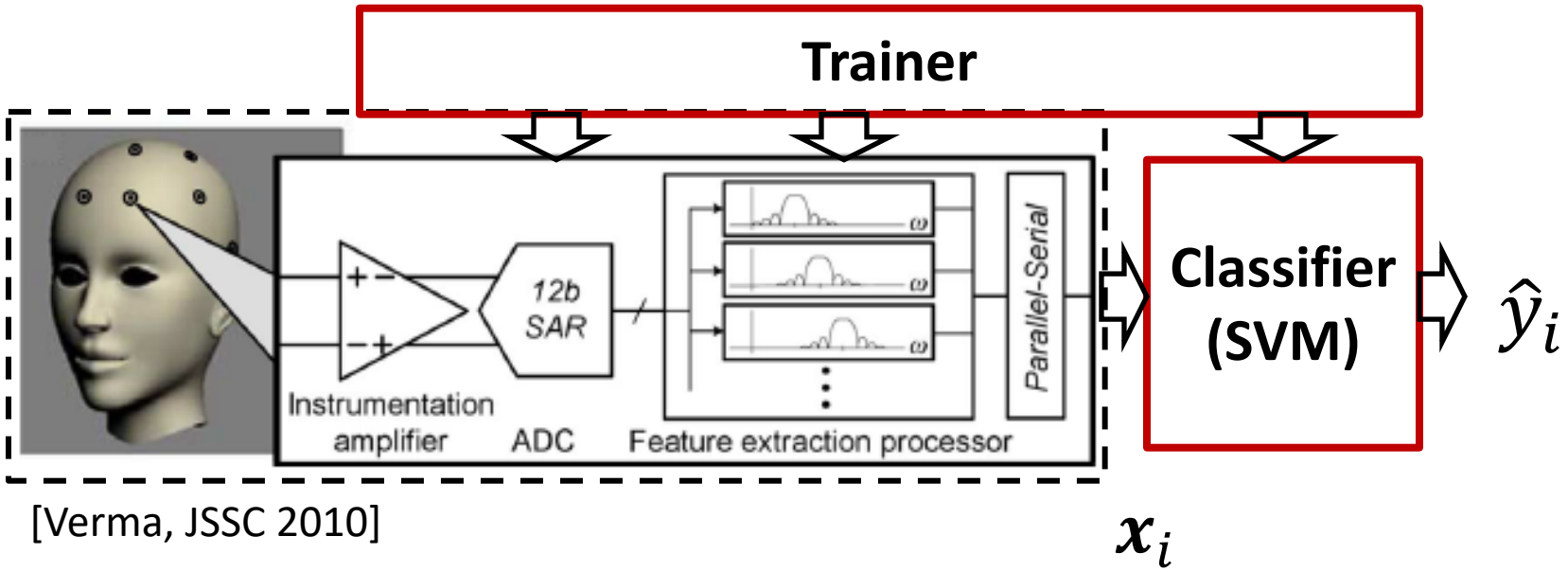
Classification - Example



$$f(\mathbf{x}) = \mathbf{w}^T\mathbf{x} + b$$

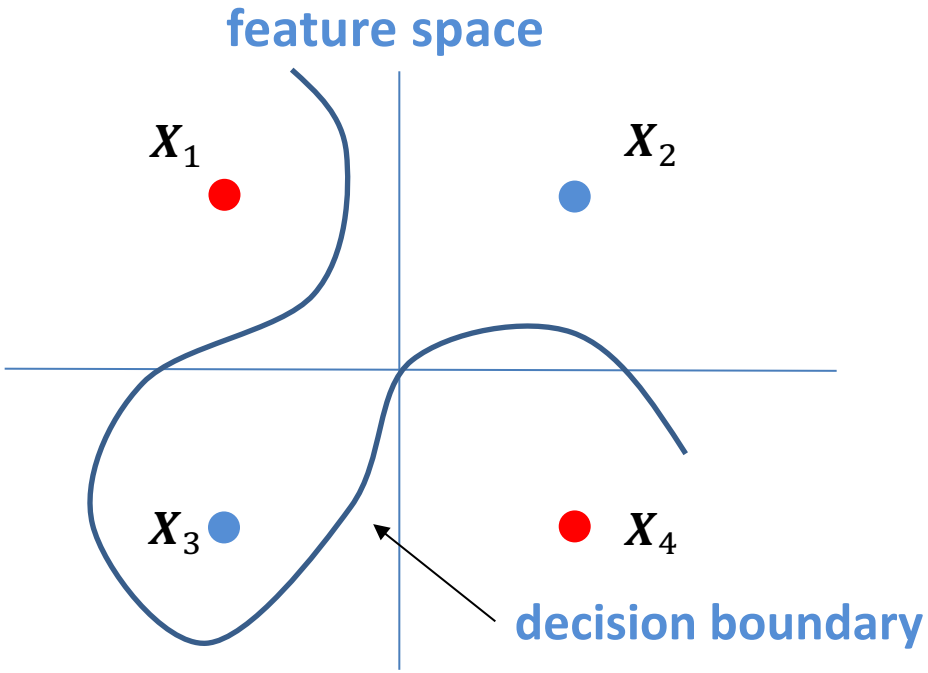
- $\boldsymbol{x}_i = [x_1, x_2]$ is the feature vector;
- assign to $\boldsymbol{x}_i$ a class label $\hat{y}_i \in \{0,1\}$ where '1' → 'Hippo' and '0' → 'Giraffe' class.

Classification – EEG Analysis Example



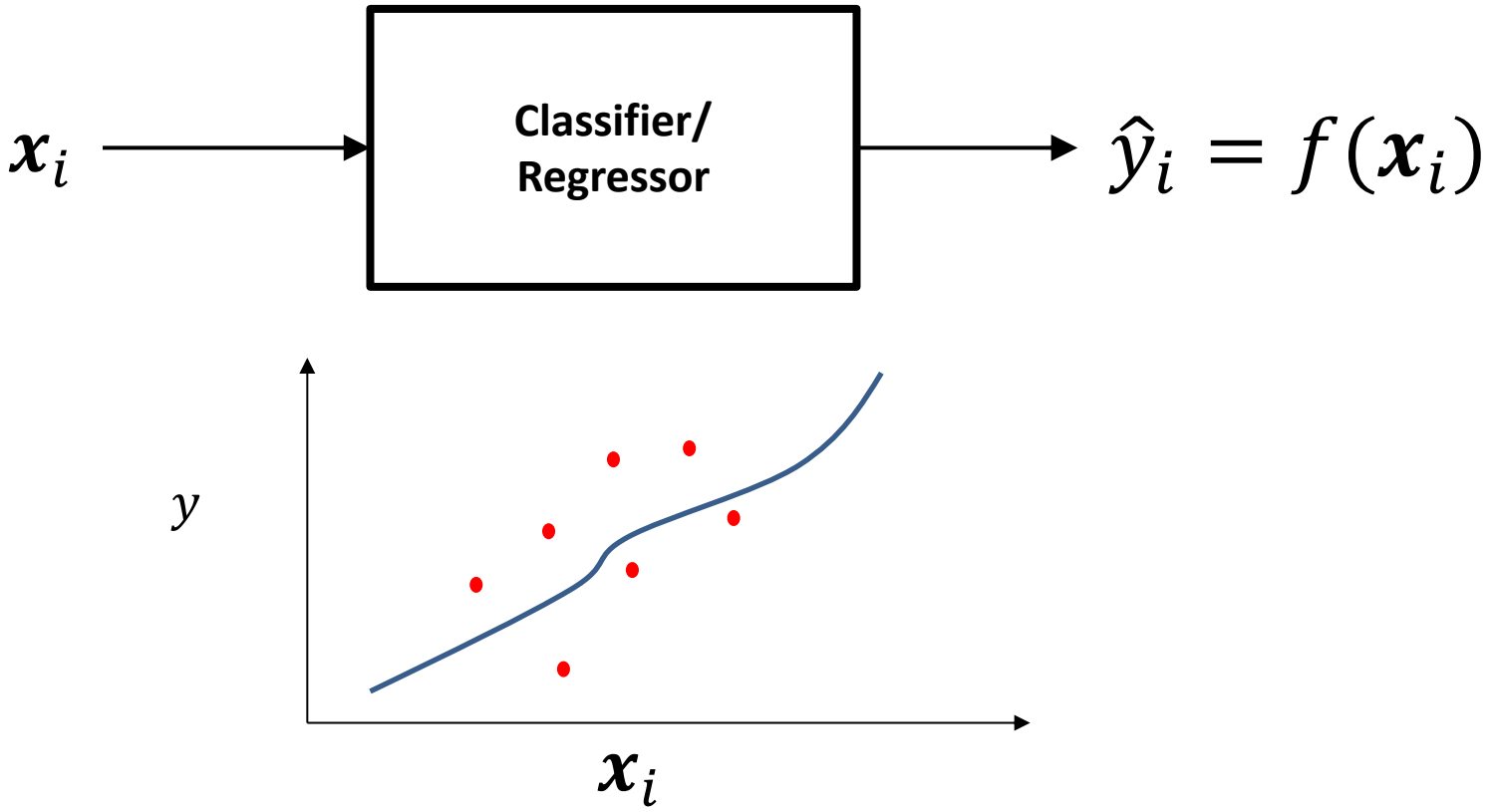
[Verma, JSSC 2010]

- $\boldsymbol{x}_i$ is the feature vector from patient $i's$ EEG record;
- assign to $\boldsymbol{x}_i$ a class label $\hat{y}_i \in \{0,1\}$ where '1' (abnormal/epileptic seizure) and '0' (normal) class.



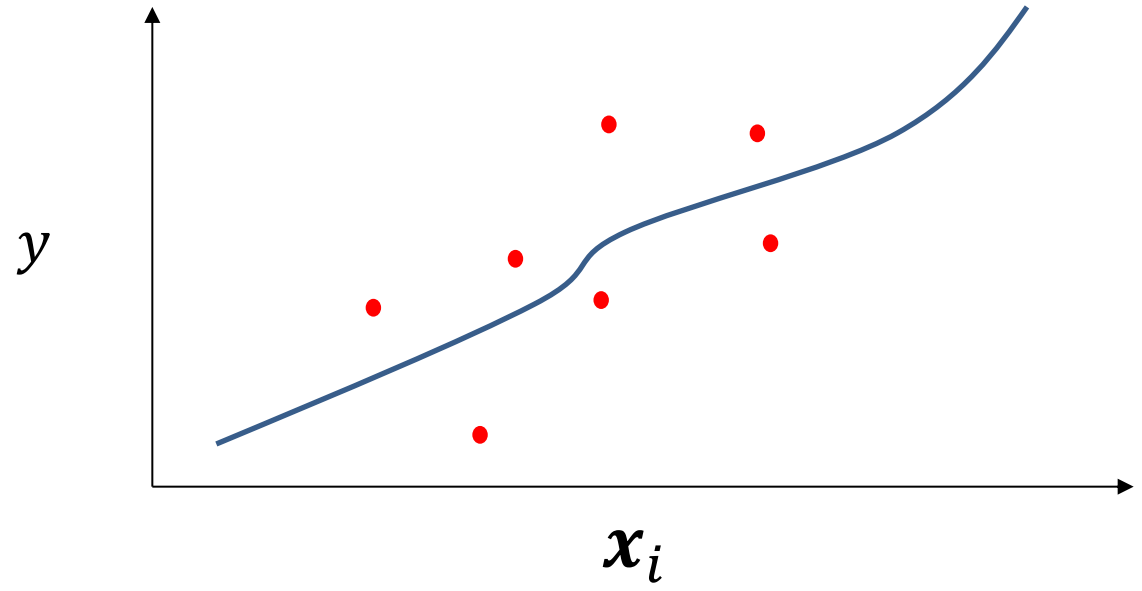
- maps input feature vectors ($\boldsymbol{X}_i$) to 1-of-M classes via a decision boundary

Regression



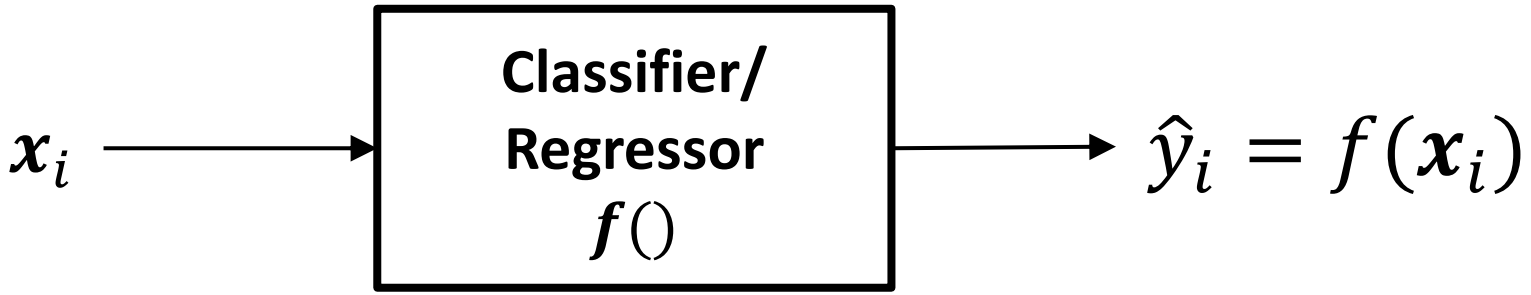
- assigning a continuous valued prediction $\hat{y}_i$ to data x_i
- data x_i can be discrete or continuous, scalar or vector

Regression Task – Predicting Grades



- $\boldsymbol{x}_i = [study\ hours, previous\ grades, ..]$ is a vector of student data
- assign to $\boldsymbol{x}_i$ a scalar value $\hat{y}_i \in [0,100]$ where $\hat{y}_i$ is the predicted grade in ECE 498NSU/598NSG
- Other examples?

Inference



- Learning → determining the function $f()$ from **training dataset** (examples)
- Inference → using the function $f()$ on **test dataset**

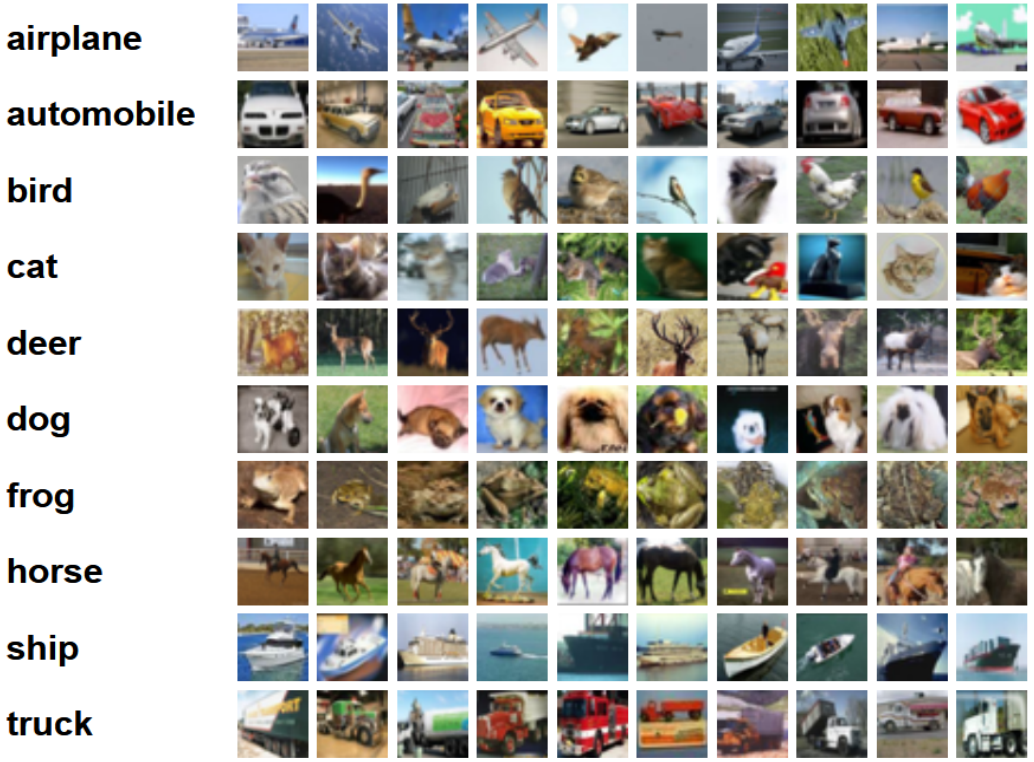
Standard Datasets

MNIST

[LeCun 1998]

- dataset of handwritten digits
- 60k training samples and 10k test samples
- each image is 28x28 grayscale
- ‘old’ dataset with >99% accuracy achieved via several methods

CIFAR-10

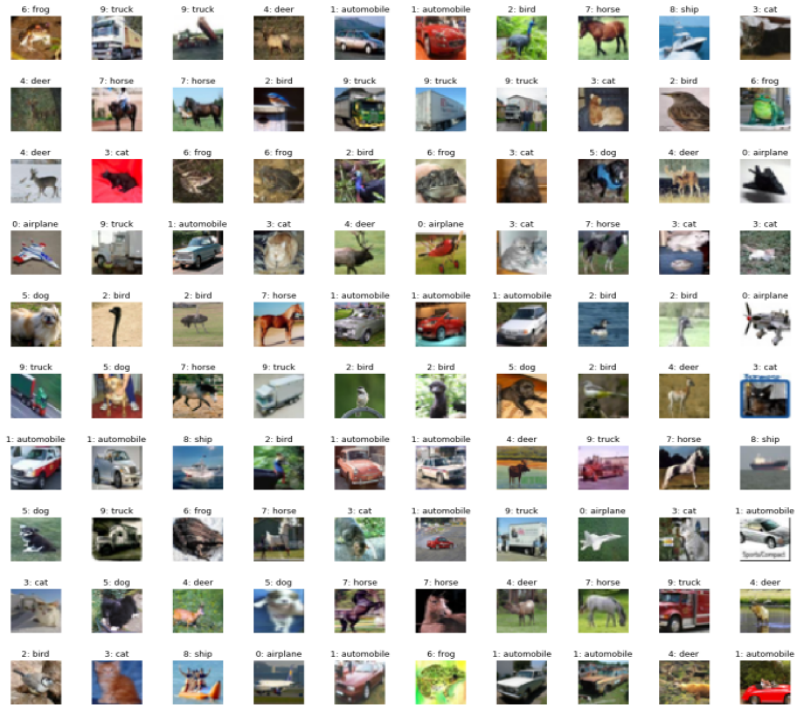


[Krizhevsky 2009]

- small images of objects and animals
- 50k training samples and 10k test samples
- each image is 32x32x3 (with colors)
- de-facto dataset for academic research
  - good for validating ideas and generating insights
  - *if it doesn't work on CIFAR-10 don't even worry about ImageNet*
- typical accuracy ~90%, good accuracy ~95%, SOTA ~98% (changes every few months)

CIFAR-100

- small images similar to CIFAR-10 but having 100 classes instead of 10
- 50k training samples and 10k test samples
- each image is 32x32x3 (with colors)
- good alternative to ImageNet when resources are limited
  - difficult task on a small dataset
- typical accuracy ~70%, good accuracy ~80%, SOTA ~90% (changes every few months)



[Krizhevsky 2009]

SVHN

- dataset of digits
- 600k training samples and 26k test samples (a large dataset)
- each image is 32x32x3 (with colors)
- simple task, but size of dataset makes it interesting
- typical accuracy ~95%, good accuracy ~97%, SOTA ~98% (changes every few months)

[Andrew Ng 2011]

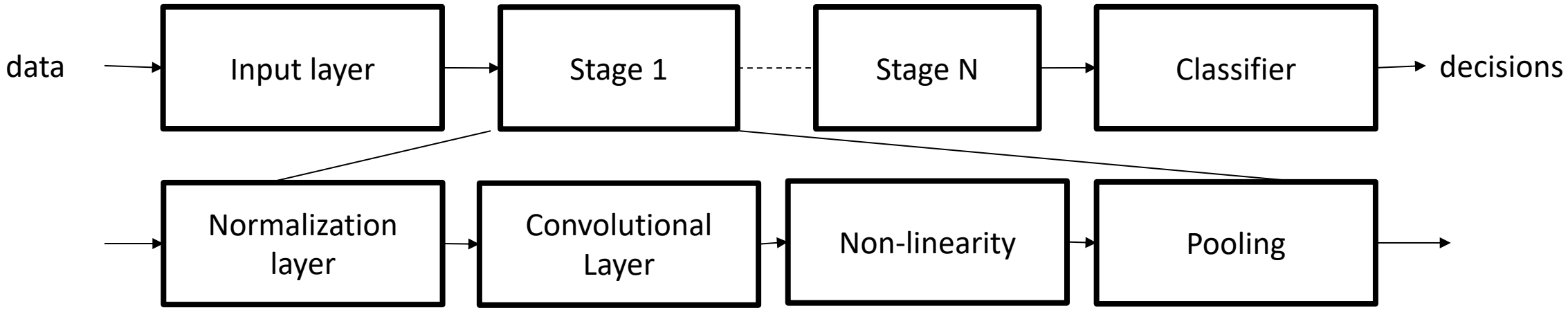
ImageNet

[FeiFei Li 2006]

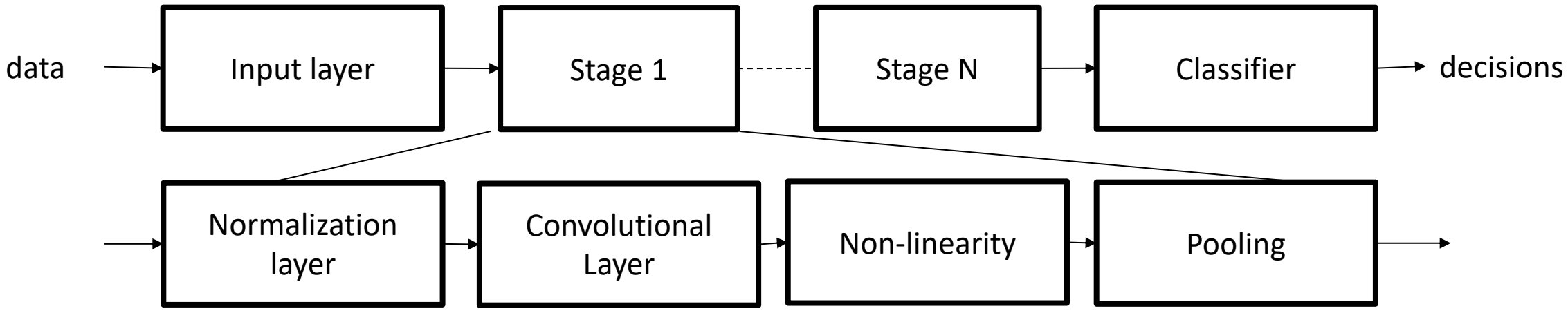
- Gold standard dataset of large images originating from UIUC in 2006
- every year, more data is added to the dataset
- 2012 version used for reporting in publications
- 1.2M training samples and 100k test samples (a HUGE dataset)
- each image is 224x224x3 (with colors) – 1000 classes
- very difficult and complex task → yearly competition
- remainder of this lecture: we'll look at some famous networks to have been deployed on ImageNet

Deep Neural Network - Structure

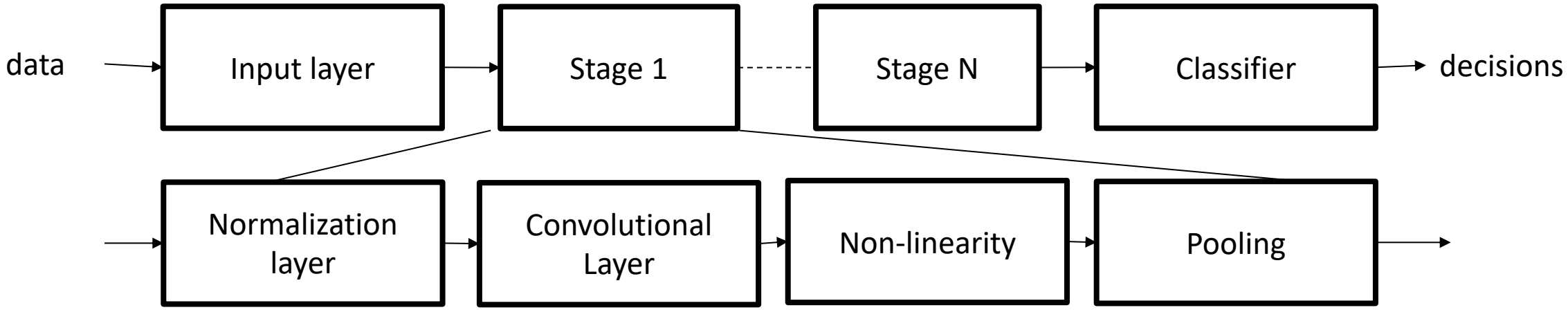
Layered Architecture



- input layer
- convolutional (CONV) – compute bound
- non-linearity
- subsampling/pooling
- fully connected (FC) – memory bound
- classifier



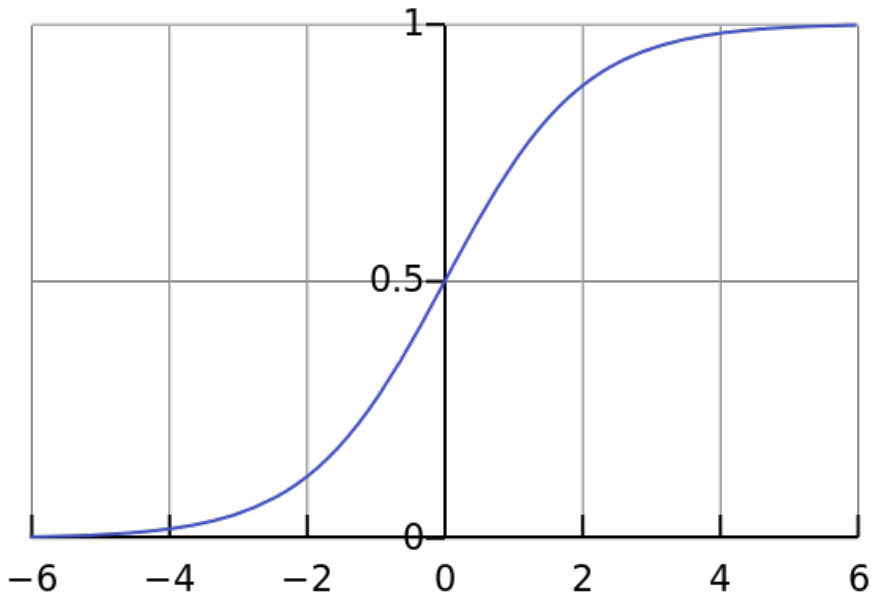
- **Normalization**: optional step →makes feature extraction independent of specific input data attributes such as intensity, e.g., an object should be detected whether it is illuminated or not
- average removal, high-pass filtering, local contrast normalization, variance normalization
- **Convolutional Layer**: extracts features



- **Non-linearity**: accentuates certain aspects of the feature maps required by the application → *soft decision* step
  - E.g., sparsification, saturation, lateral inhibition, ReLU (rectified linear unit), tanh, component-wise shrinkage
- **Pooling/subsampling**: averaging step → reduces variance to geometric transformations → data dimension is reduced, e.g., max, average
- **Fully connected layer**: matrix-vector multiply
- **Classifier**: generates the final decision $\hat{y}$

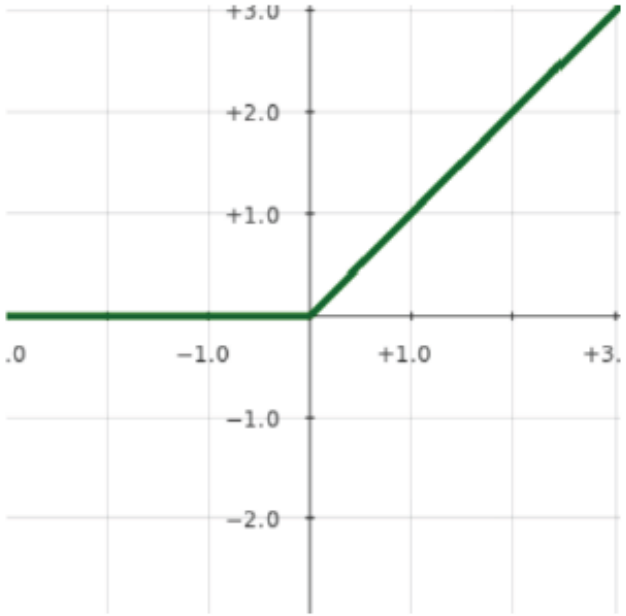
Non-linearity

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



sigmoid

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



ReLU

- ReLU: rectified linear unit
- ReLU enables faster convergence than sigmoid (why?). Easier to implement.

Pooling

- Two commonly choices for pooling:
  - **average pooling** takes arithmetic mean within the pooling window, i.e.,

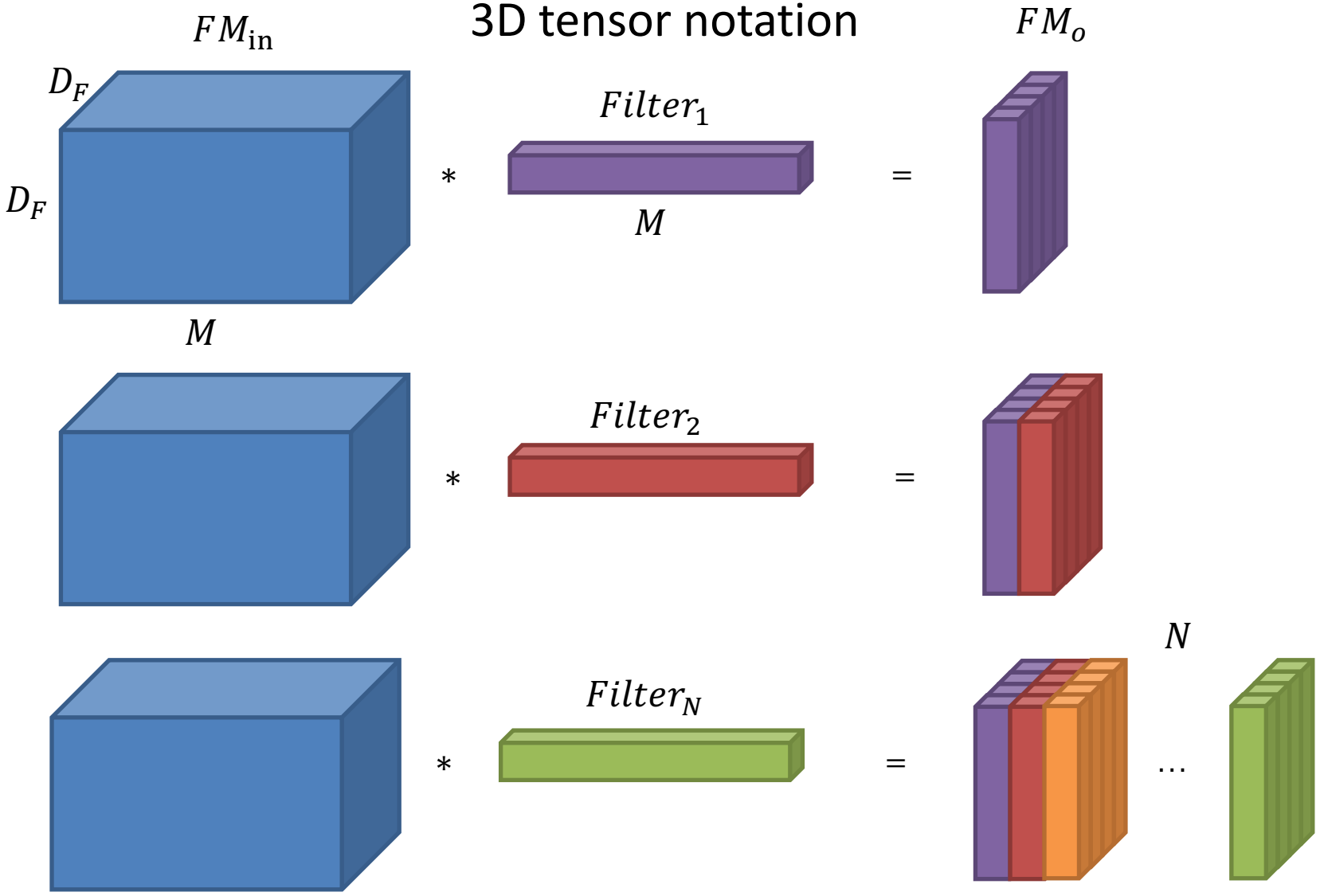
$$s_j = \frac{1}{|R_j|} \sum_{i \in R_j} a_i$$

  - **max pooling** selects the largest element within the pooling window, i.e.,

$$s_j = \max_{i \in R_j} a_i$$

Convolutional Layer (CONV)

3D tensor notation



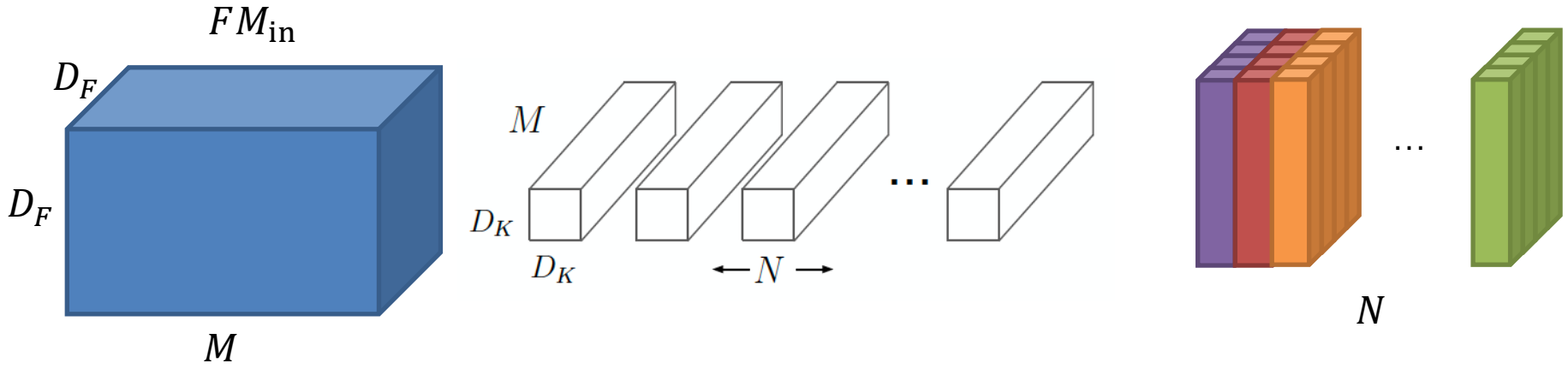
# of input channels = M

# of filter kernels = N

# of output channels = N

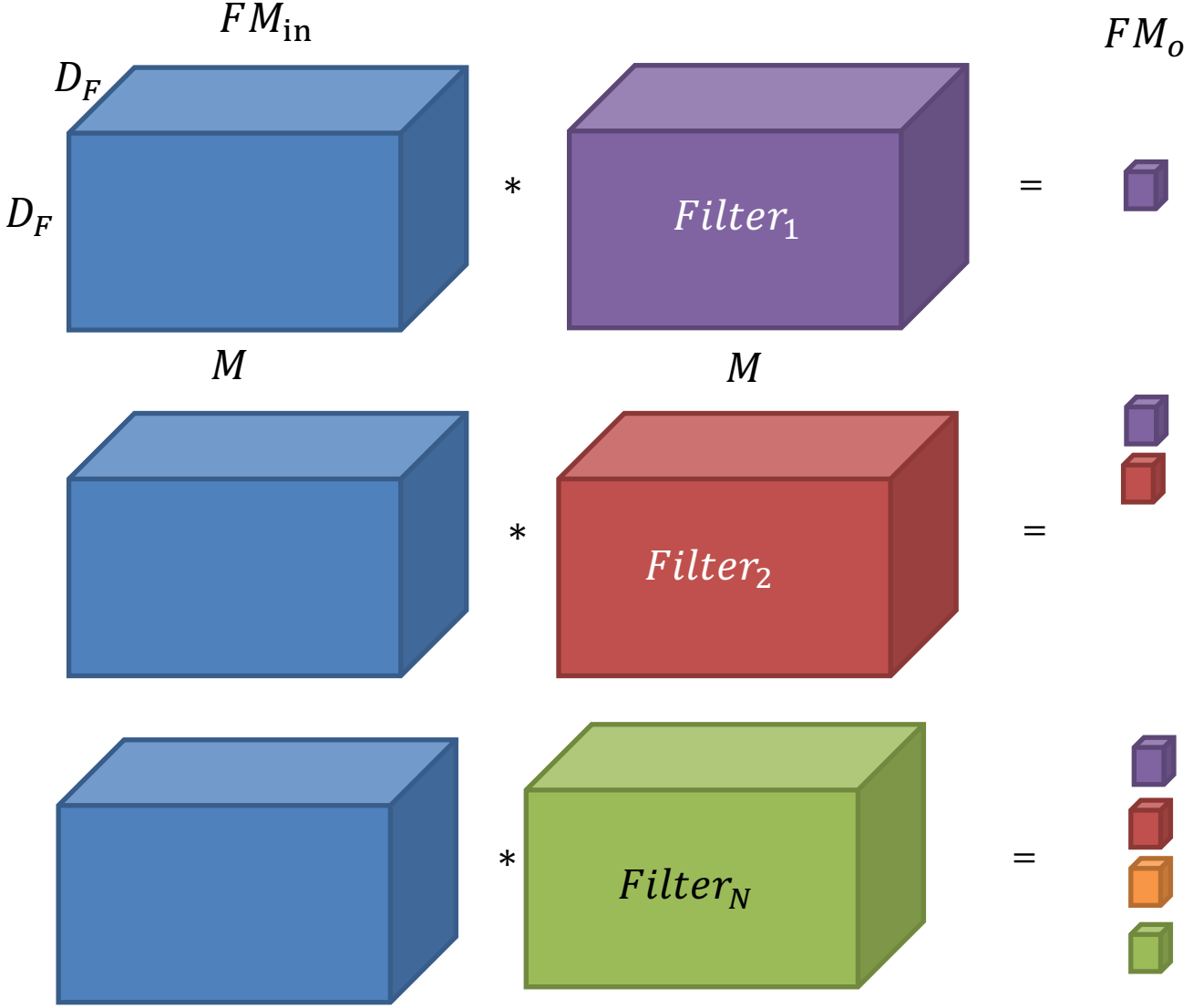
input feature map: FM_{in}

output feature map: FM_o



- Kernel size: $D_K^2 \times M \times N$; Assuming stride of 1 below
- Computational cost in terms of # of MACs (multiply-accumulates) = $D_K^2 \times M \times N \times D_F^2$
- Each weight contributes to D_F^2 outputs = "weight reuse factor"
- Each input contributes to $D_K^2 \times N$ outputs = "data reuse factor"
- higher reuse factors → lower memory bandwidth requirements (compute bound)

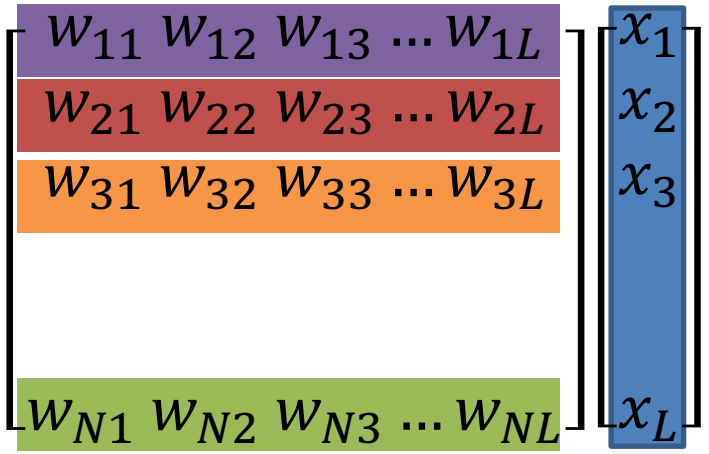
Fully-Connected Layer (FC)



matrix-vector multiply (MVM) notation

$$\begin{bmatrix} w_{11} & w_{12} & w_{13} & \dots & w_{1L} \\ w_{21} & w_{22} & w_{23} & \dots & w_{2L} \\ w_{31} & w_{32} & w_{33} & \dots & w_{3L} \\ & & & & \\ w_{N1} & w_{N2} & w_{N3} & \dots & w_{NL} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ \\ \\ x_L \end{bmatrix}$$

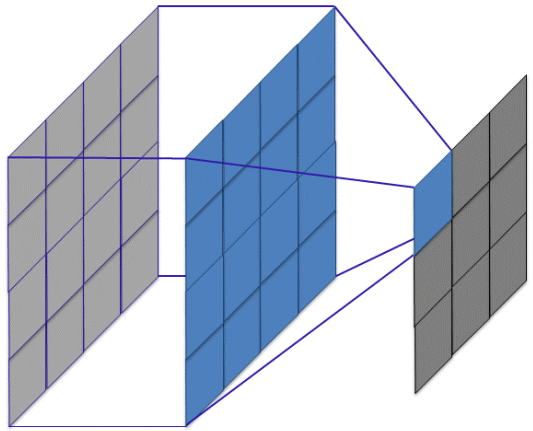
$$L = D_F^2 M$$



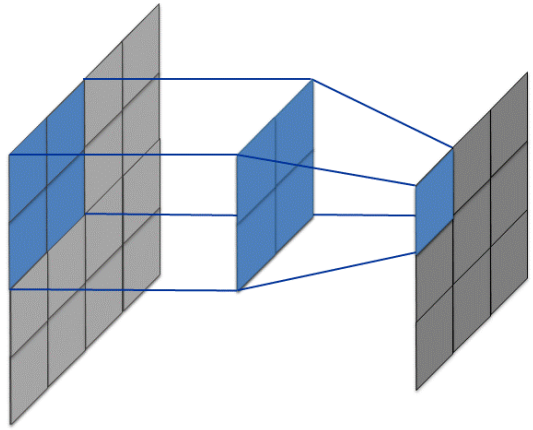
- Computational cost in terms of # of MACs (multiply-accumulates) = $D_F^2 \times M \times N$
- Each weight contributes to 1 output = no weight reuse
- Each input contributes to N outputs = small data reuse factor
- FC layers will have highest storage and memory bandwidth requirements (memory bound)

FC vs. CONV

4x4 input 4x4 kernel 3x3 output



4x4 input 2x2 kernel 3x3 output



fully connected (FC):

each output pixel connects to all the pixels of the input FM

distinct weights (no weight reuse):

each output pixel is processed by distinct kernel

total number of weights: 9x4x4=144

locally connected:

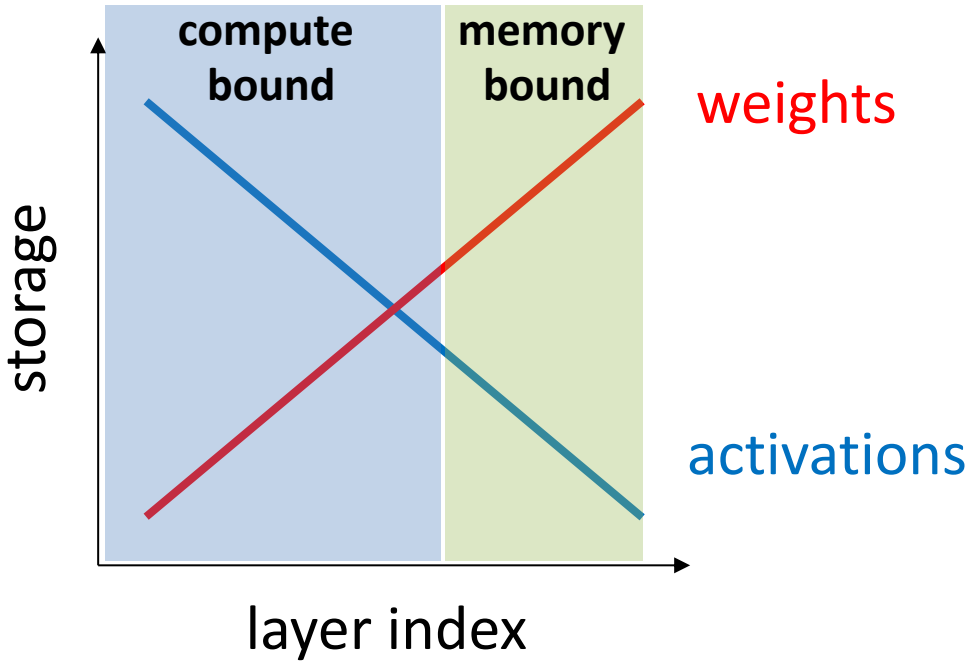
each output pixel only connects to a subset of pixels (receptive field) in the input FM

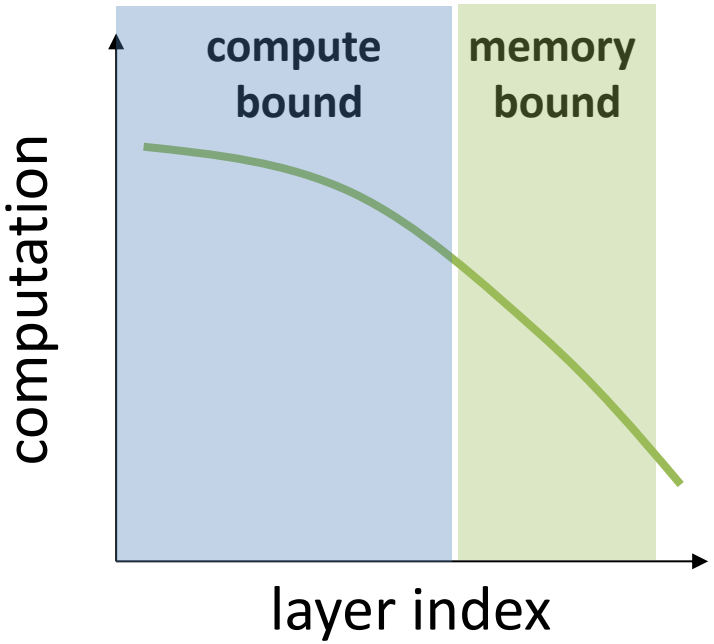
shared weights (high weight reuse):

every output pixel is computed with the same kernel

total number of weights: 4

Storage vs. Computational Costs





- networks layers tend to be compressive – large input image → M softmax values
- layers near the input - compute bound (convolutional)….activation storage cost is high
- layers near the output – memory bound (FC)….weight storage is much higher

Popular DNNs

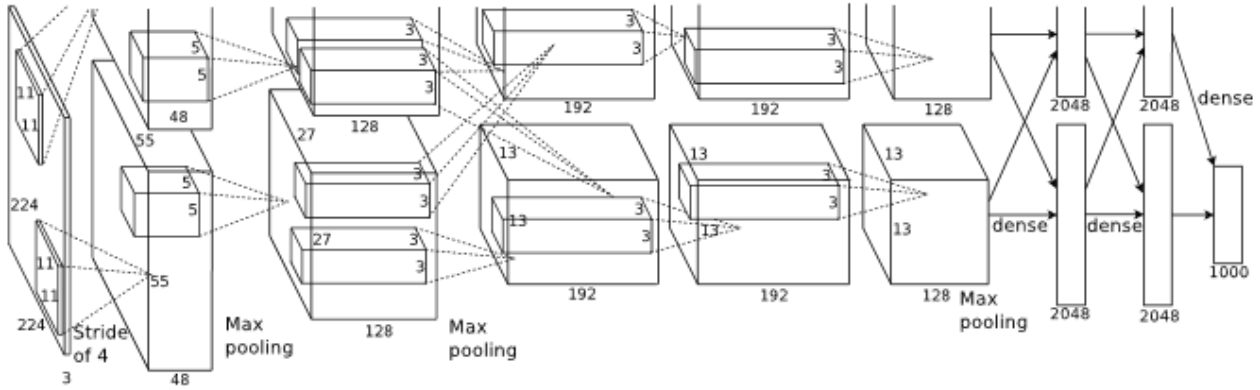
AlexNet

ImageNet Classification with Deep Convolutional Neural Networks

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

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

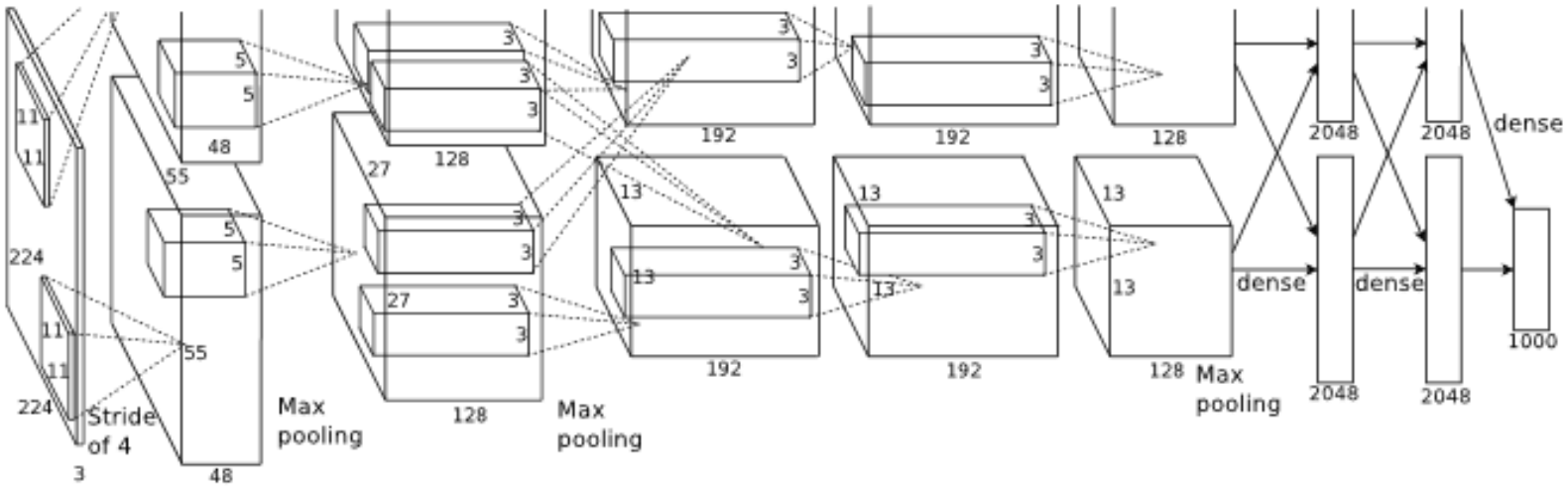
Geoffrey E. Hinton
University of Toronto
hinton@cs.utoronto.ca



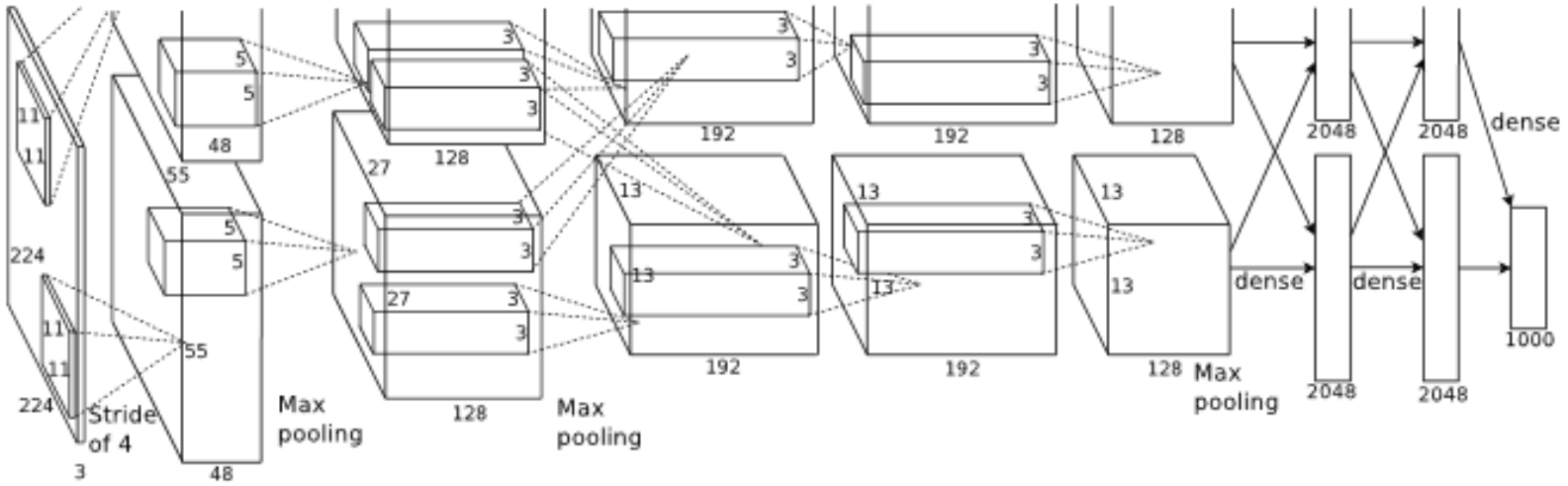
[NIPS 2012]

- famous for winning the 2012 ImageNet competition by a large margin (Top 5: 15.3% vs 26.2% (second place))
- key innovations:
  - use of ReLU instead of tanh non-linearity
  - use of dropout for regularization
  - overlap pooling to reduce size of network

AlexNet: Network Topology



- 5 convolutional layers & 3 fully connected layers
- 62.3 million parameters – 6% in convolutional layers
- 1.1 billion MACs in a forward path – 95% in convolutional layers



CONV

FC

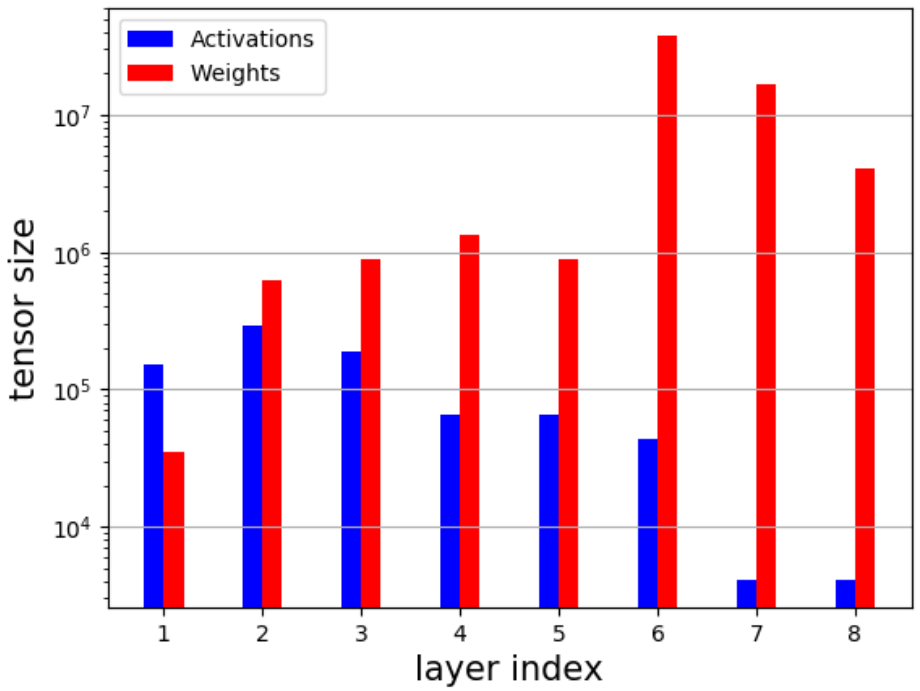
```
class AlexNet(nn.Module):

    def __init__(self, num_classes=1000):
        super(AlexNet, self).__init__()
        self.features = nn.Sequential(
            nn.Conv2d(3, 64, kernel_size=11, stride=4, padding
            nn.ReLU(inplace=True),
            nn.MaxPool2d(kernel_size=3, stride=2),
            nn.Conv2d(64, 192, kernel_size=5, padding=2),
            nn.ReLU(inplace=True),
            nn.MaxPool2d(kernel_size=3, stride=2),
            nn.Conv2d(192, 384, kernel_size=3, padding=1),
            nn.ReLU(inplace=True),
            nn.Conv2d(384, 256, kernel_size=3, padding=1),
            nn.ReLU(inplace=True),
            nn.Conv2d(256, 256, kernel_size=3, padding=1),
            nn.ReLU(inplace=True),
            nn.MaxPool2d(kernel_size=3, stride=2),
        )
```

```
self.avgpool = nn.AdaptiveAvgPool2d((6, 6))
self.classifier = nn.Sequential(
    nn.Dropout(),
    nn.Linear(256 * 6 * 6, 4096),
    nn.ReLU(inplace=True),
    nn.Dropout(),
    nn.Linear(4096, 4096),
    nn.ReLU(inplace=True),
    nn.Linear(4096, num_classes),
)
```

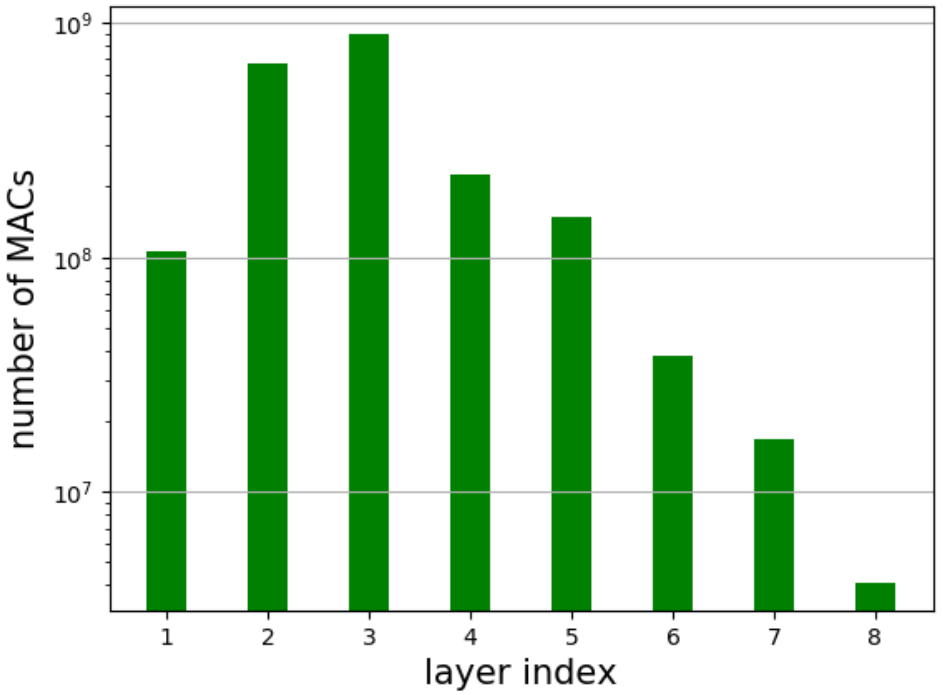
- 5 convolutional layers & 3 fully connected layers
- 62.3 million parameters – 6% in convolutional layers
- 1.1 billion MACs in a forward path – 95% in convolutional layers

Complexity of AlexNet

Representation



Computation

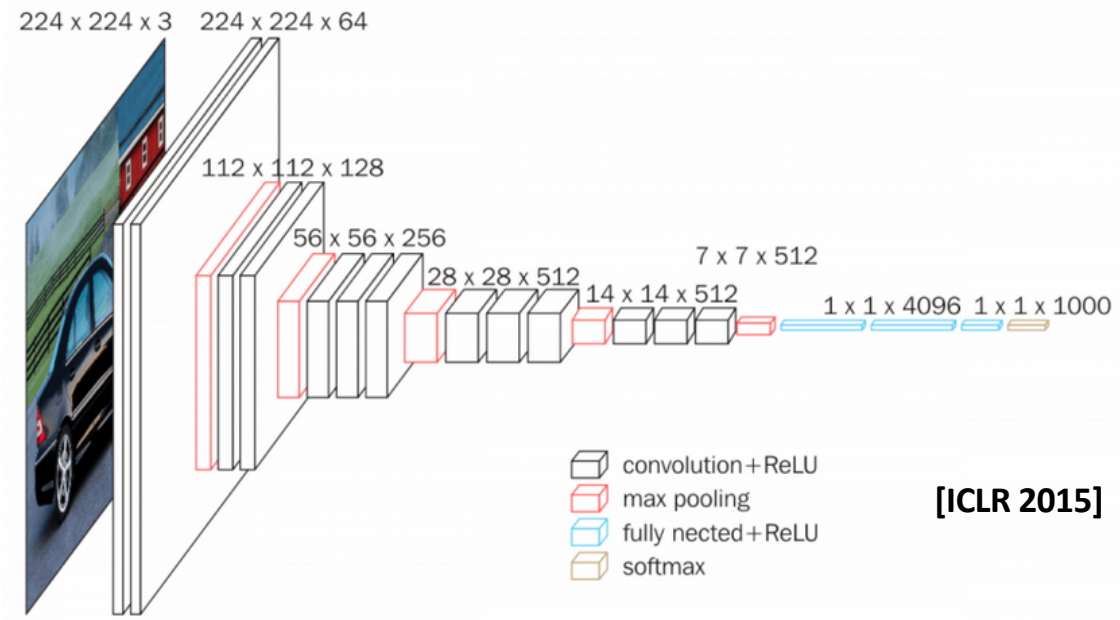


- last three weight layers are an order of magnitude larger than all other tensors
- first few layers are most compute intensive

VGGNet

VERY DEEP CONVOLUTIONAL NETWORKS FOR LARGE-SCALE IMAGE RECOGNITION

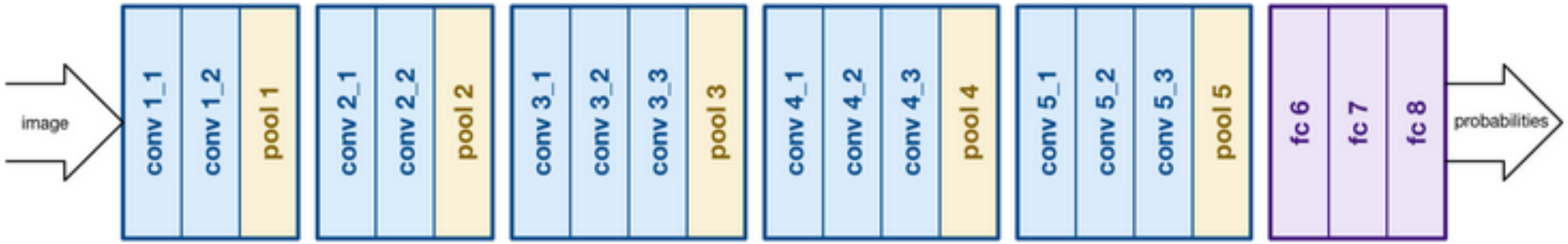
Karen Simonyan* **& Andrew Zisserman+**
Visual Geometry Group, Department of Engineering Science, University of Oxford
{karen,az}@robots.ox.ac.uk



[ICLR 2015]

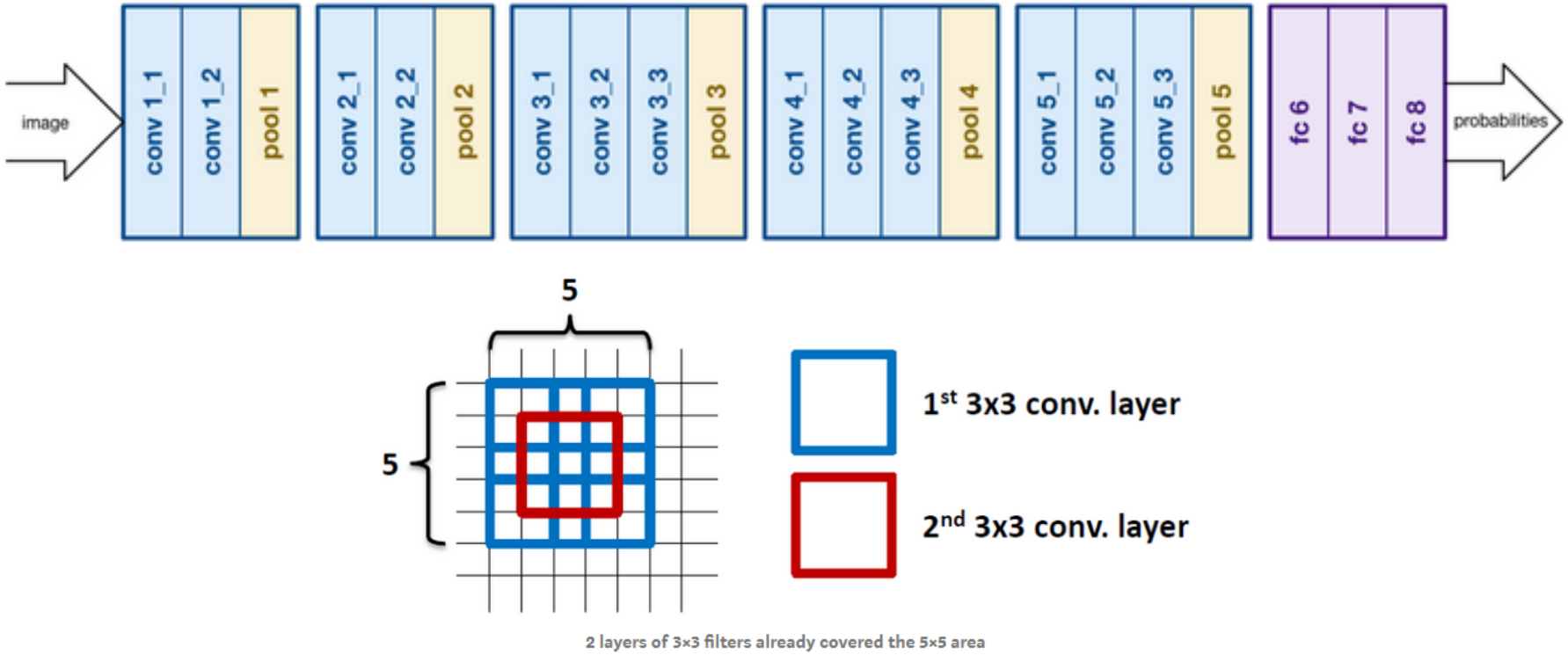
- runner-up at the ImageNet 2014 competition behind GoogleNet
- achieves a top-5 error rate of 8.8% (VGG-16)
- very appealing to practitioners because of its simple and uniform topology – (maxpool = 2 x 2 (stride 2)) + (3x3 filter kernels)

VGGNet: Topological Details (VGG-16)



- 13 convolutional layers cascading 3x3 filters
- 3 fully connected layers
- 138 million parameters: a large network!

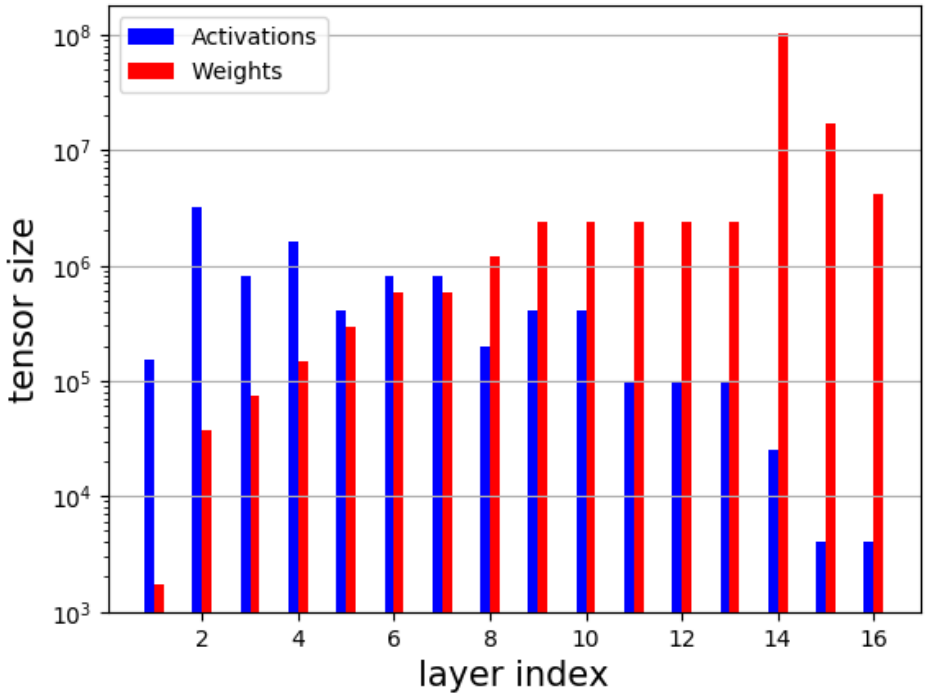
VGGNet (VGG-16)



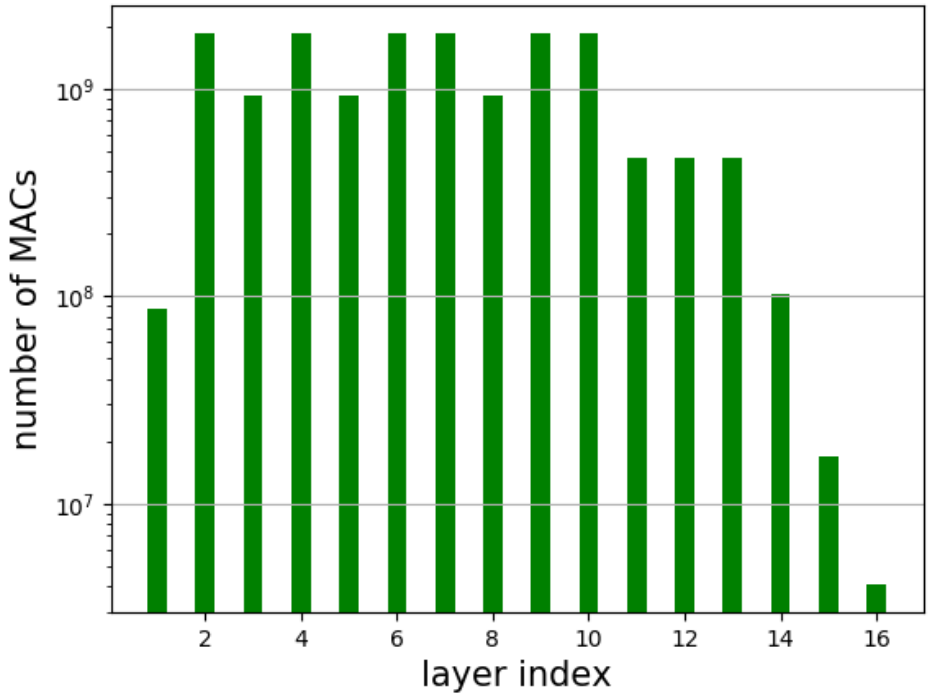
- main idea: focus on 3x3 filters and cascade them instead of using large filters
- e.g.: one 5x5 filter has 25 parameters but two successive 3x3 filters have 18

Complexity of VGG-16

Representation



Computation

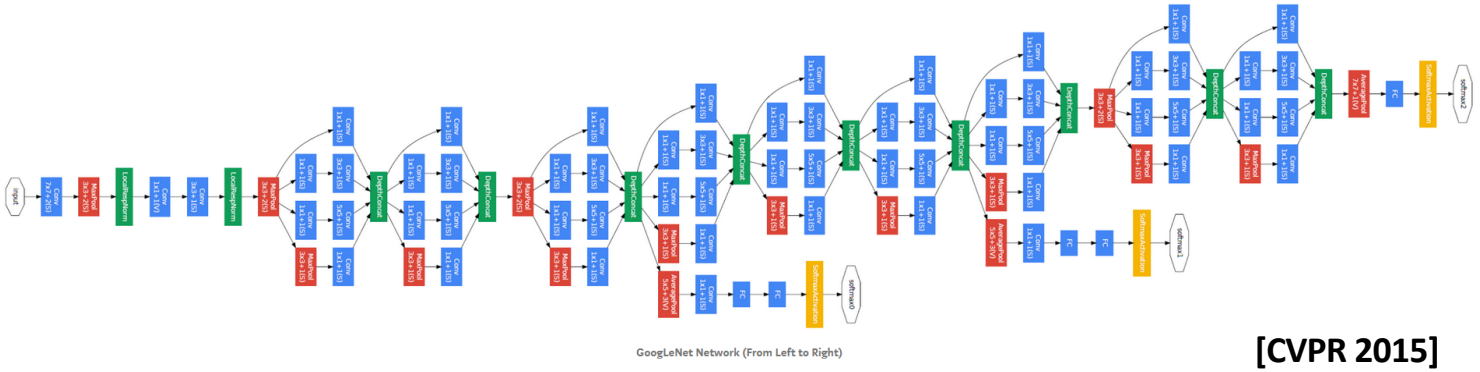


- last three weight layers are an order of magnitude larger than all other tensors
- first half of layers (convolutional) are most compute intensive

GoogLeNet

Going Deeper with Convolutions

Christian Szegedy[1], Wei Liu[2], Yangqing Jia[1], Pierre Sermanet[1], Scott Reed[3],
Dragomir Anguelov[1], Dumitru Erhan[1], Vincent Vanhoucke[1], Andrew Rabinovich[4]
[1]Google Inc. [2]University of North Carolina, Chapel Hill
[3]University of Michigan, Ann Arbor [4]Magic Leap Inc.
[1]{szegedy,jiayq,sermanet,dragomir,dumitru,vanhoucke}@google.com
[2]wliu@cs.unc.edu, [3]reedscott@umich.edu, [4]arabinovich@magicleap.com

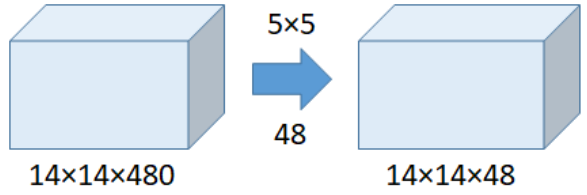


GoogLeNet Network (From Left to Right)

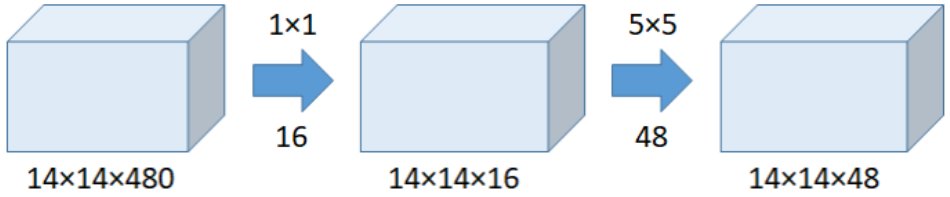
[CVPR 2015]

- winner of the ImageNet 2014 competition
- achieves a top-5 error rate of 6.7%
- key innovations
  - 1x1 convolutional layers → 'network in network'
  - 'Inception' modules

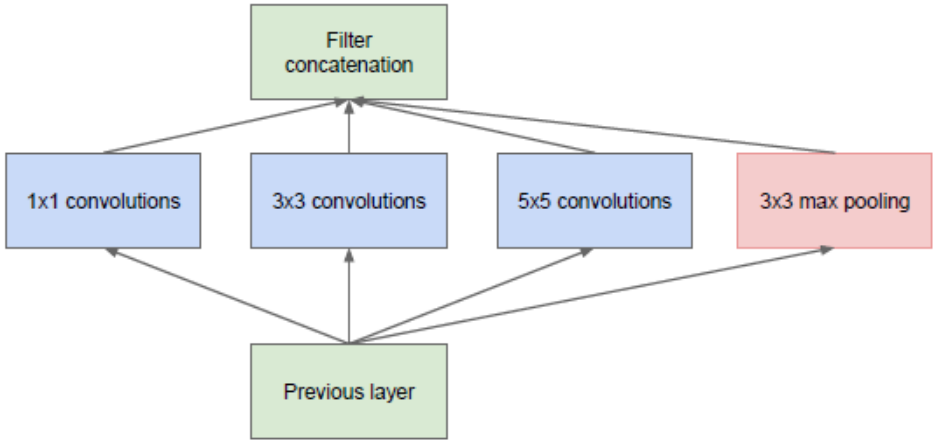
GoogLeNet: Topological Details



total #MACs = 112.9M



total #MACs = 5.3M

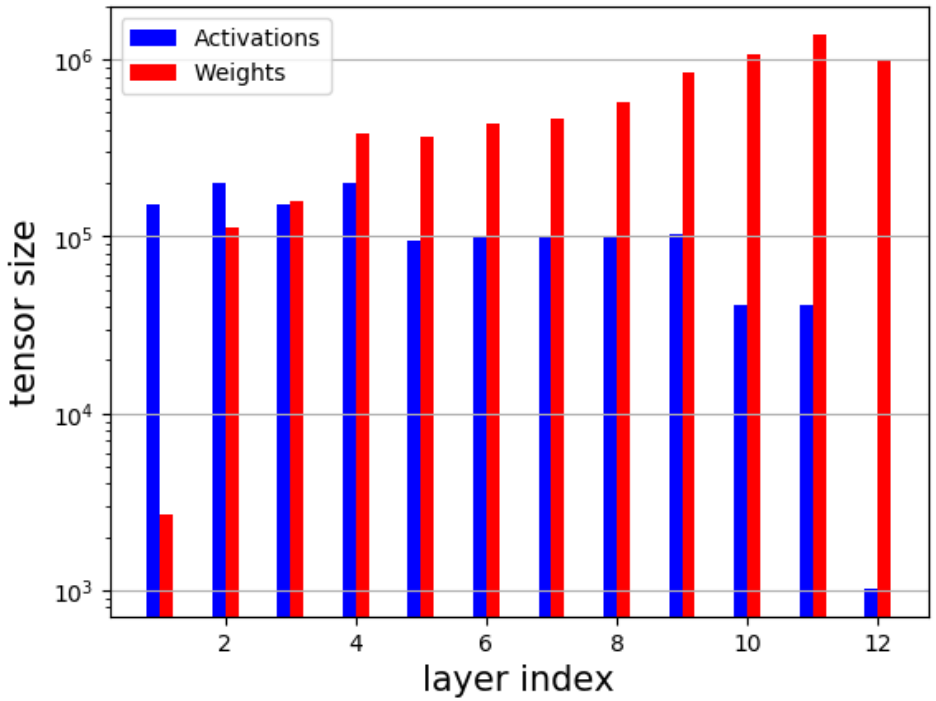


Inception module

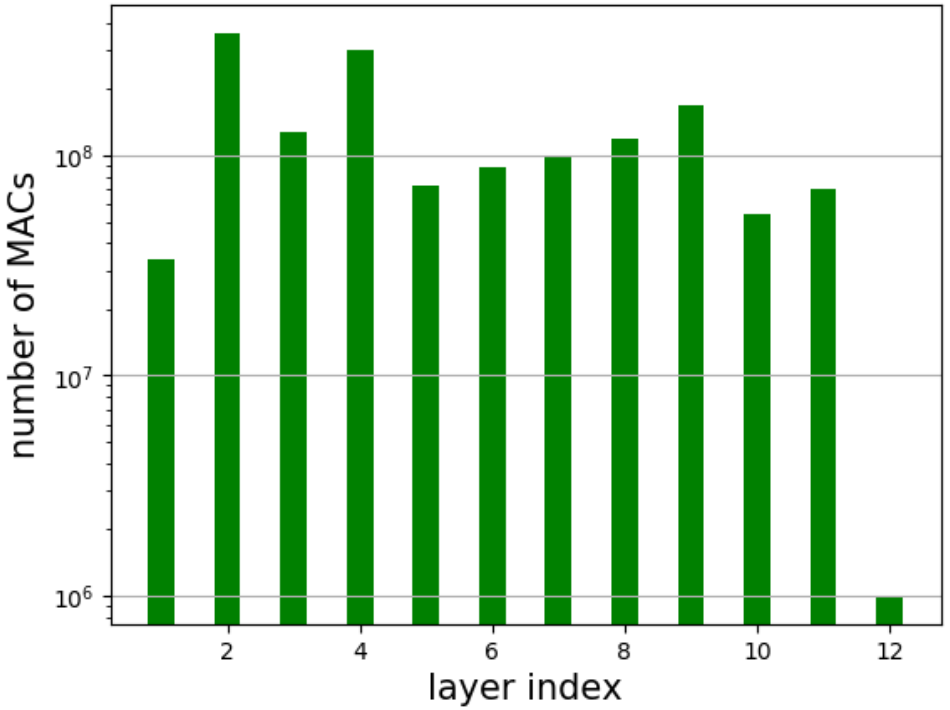
- 1x1 convolutions utilized as intermediate dimensionality reduction
- Inception modules allow for more feature extraction (expressivity)
- 22 layers but only 4 million parameters (compared to 60M for AlexNet)

Complexity of GoogleNet

Representation



Computation



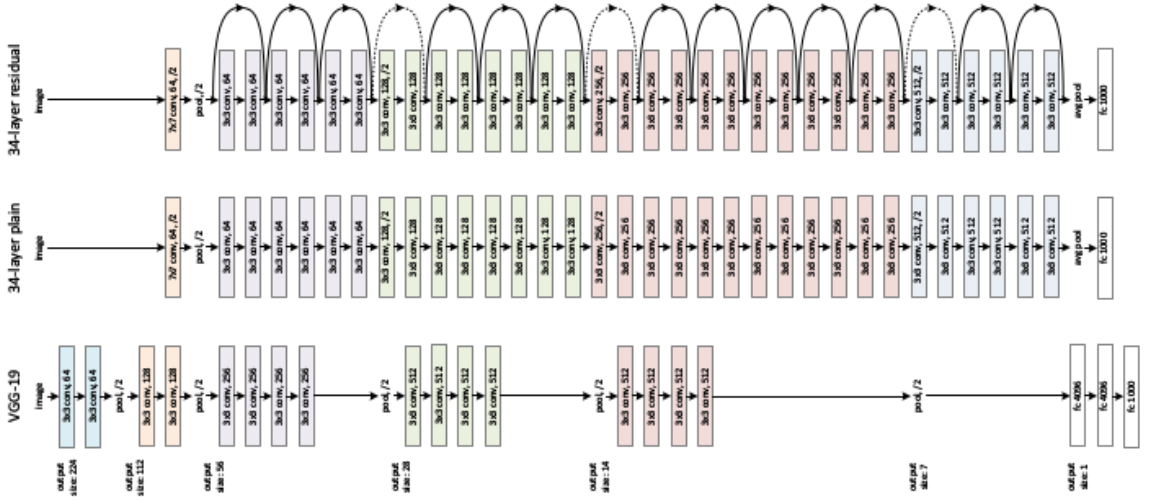
- more uniform weight distribution overall
- better weight-to-activations ratio
- computation concentrated in the 'middle' layers

N.B. each Inception module is considered one layer in the plots

ResNet

Deep Residual Learning for Image Recognition

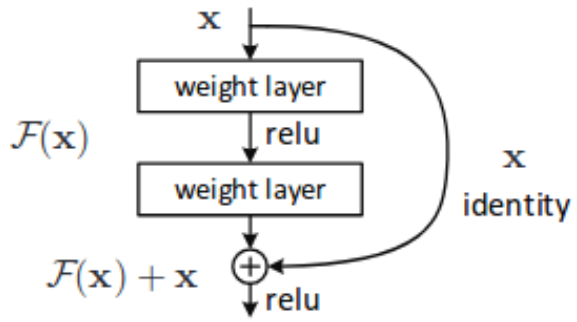
Kaiming He Xiangyu Zhang Shaoqing Ren Jian Sun
Microsoft Research
{kahe, v-xiangz, v-shren, jiansun}@microsoft.com

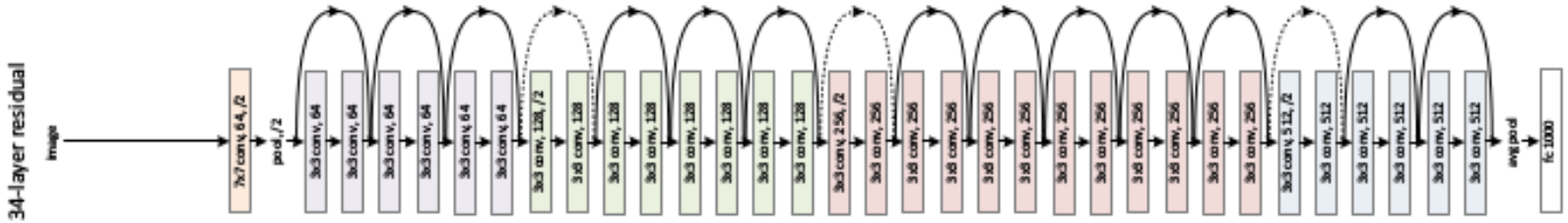


[CVPR 2016]

- winner of the ImageNet 2015 competition
- achieves a top-5 error rate of 3.6% - better than human (ResNet 152)
- key innovations
  - skip connections → allows for very deep nets (more than 100 layers)
  - new weight initialization technique for better gradient handling

ResNet: Topological Details



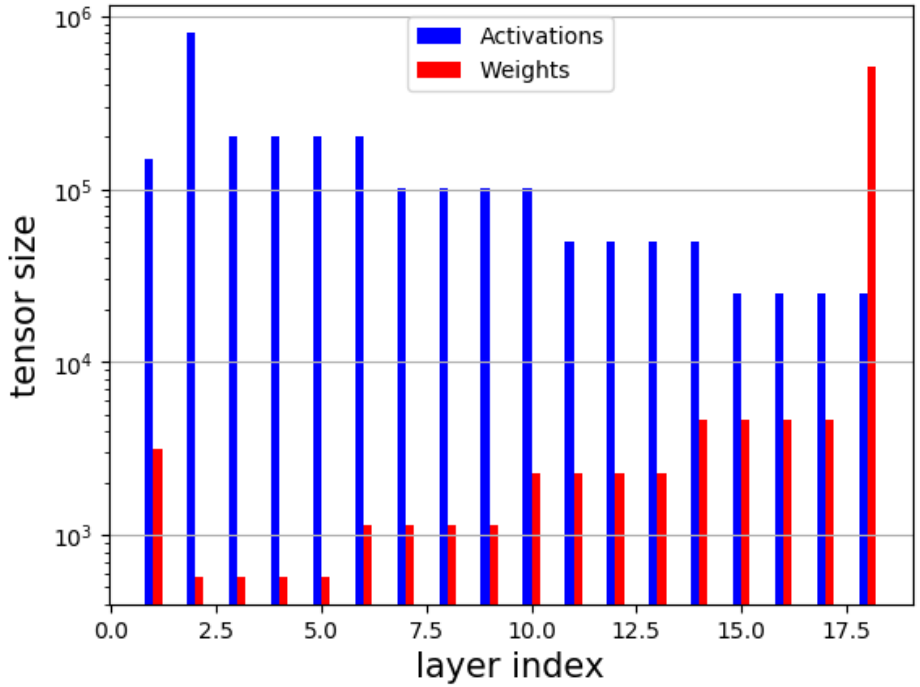


skip connection: network learns deviation from identity function

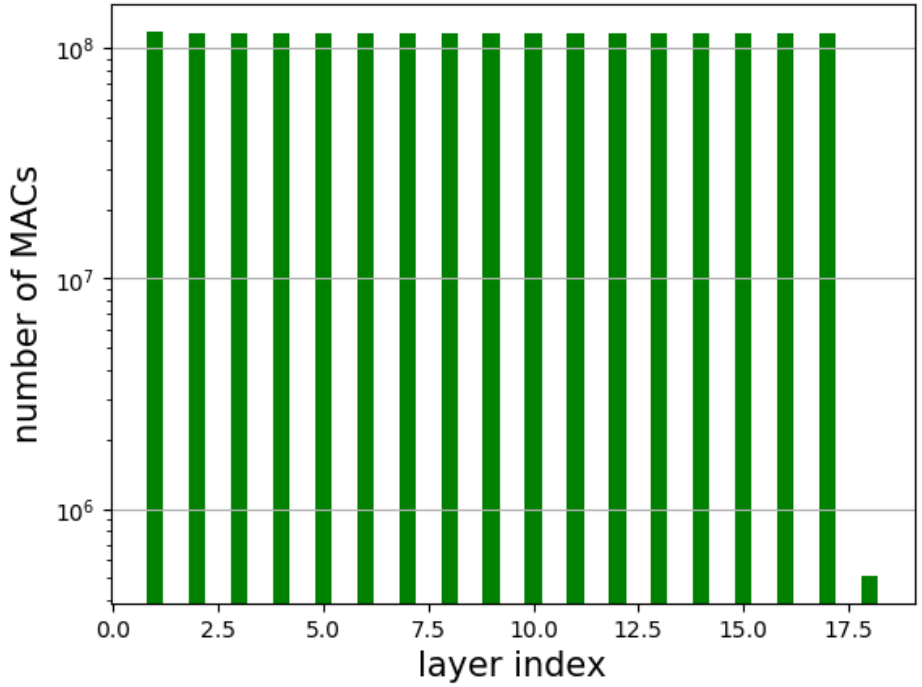
- deeper is better but need to preserve gradients → back-propagate using the identity function
- ResNet 152 has 23 million parameters and 11.3 billion MACs

Complexity of ResNet-18

Representation



Computation



- unlike most ConvNets, ResNets are *activation-dominated*
- equal amount of computation across layers – when number of channels doubles, feature dimension halves

Summary

- Two key inference tasks – classification and inference
- DL structure – layered, non-linear, vectorized (high-dimensional), compute and memory bound layers
- DL function – learning and inference steps
- Inference – making decisions in a data-driven manner (minimal information about data distributions)

Course Web Page

https://courses.grainger.illinois.edu/ece598nsg/fa2020/
https://courses.grainger.illinois.edu/ece498nsu/fa2020/

http://shanbhag.ece.uiuc.edu