16-720: COMPUTER VISION

FALL 2017

INSTRUCTORS

SRINIVAS NARASIMHAN AND DAVID HELD

M-W: 4:30p-5:50p

(Slides from Yaser Sheikh)

DEEP LEARNING II

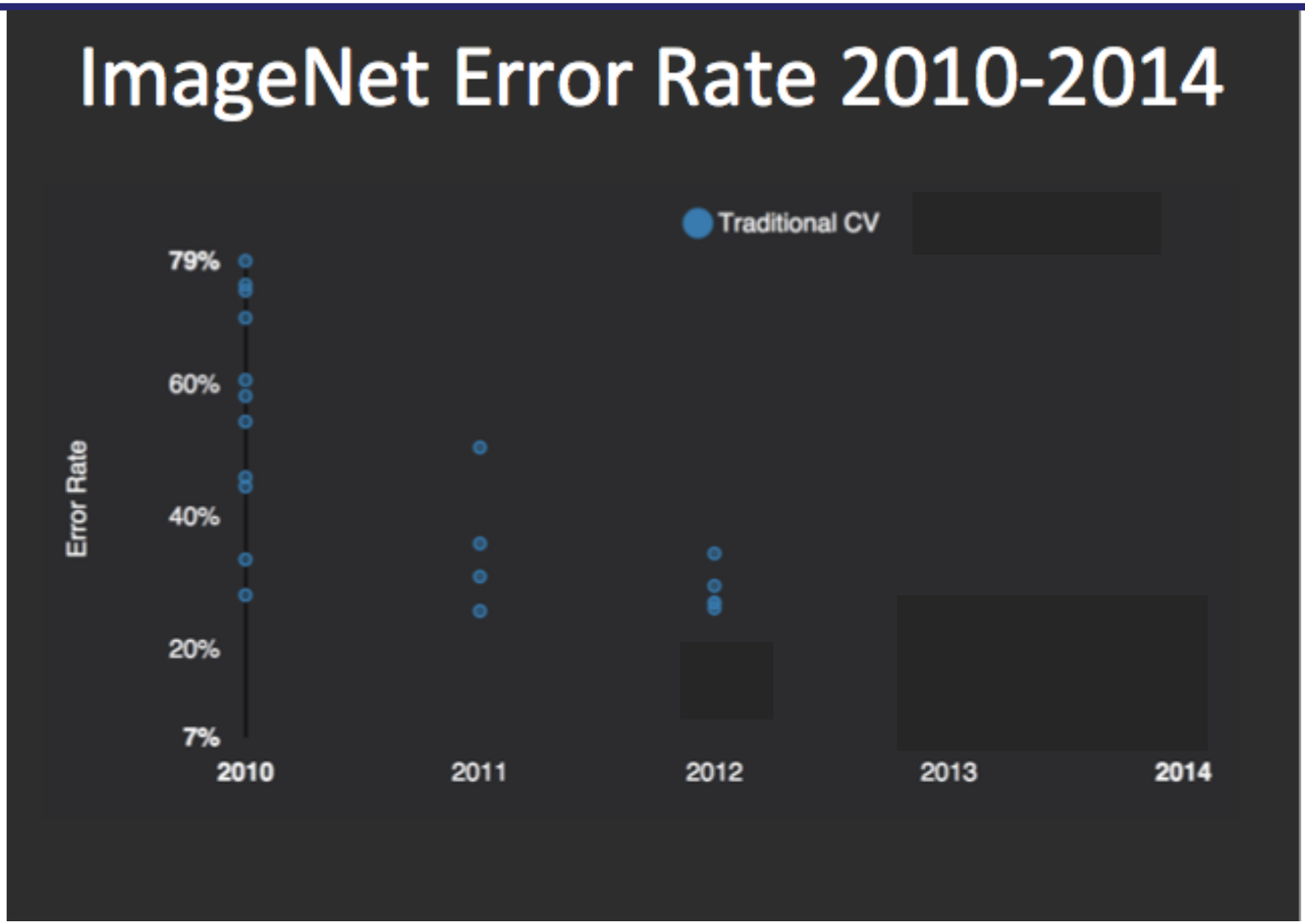
Back Propagation and Convolutional Neural Networks

“THE EYE SEES ONLY WHAT THE MIND IS PREPARED TO COMPREHEND”

- HENRI BERGSON

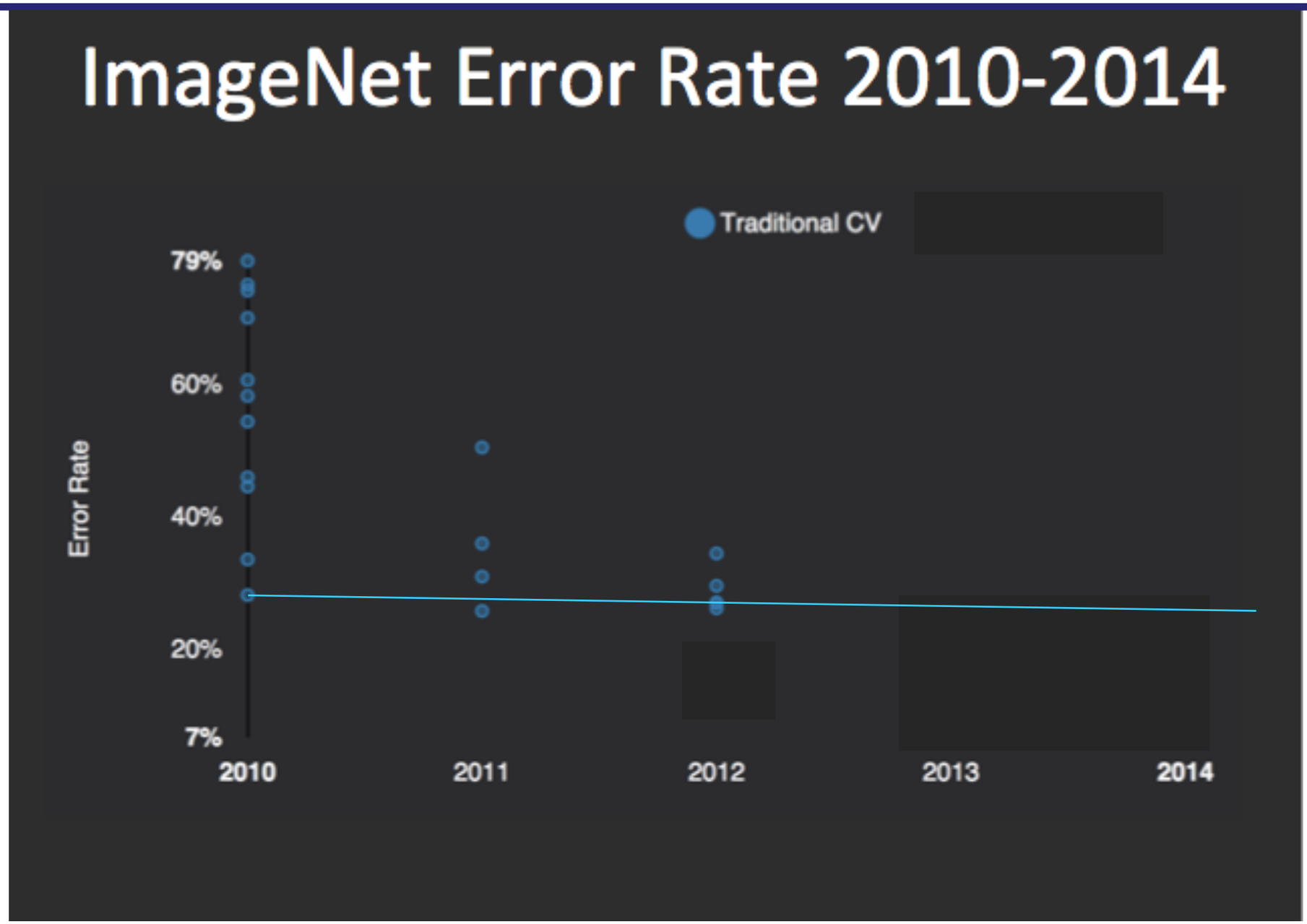

IMAGENET

Performance



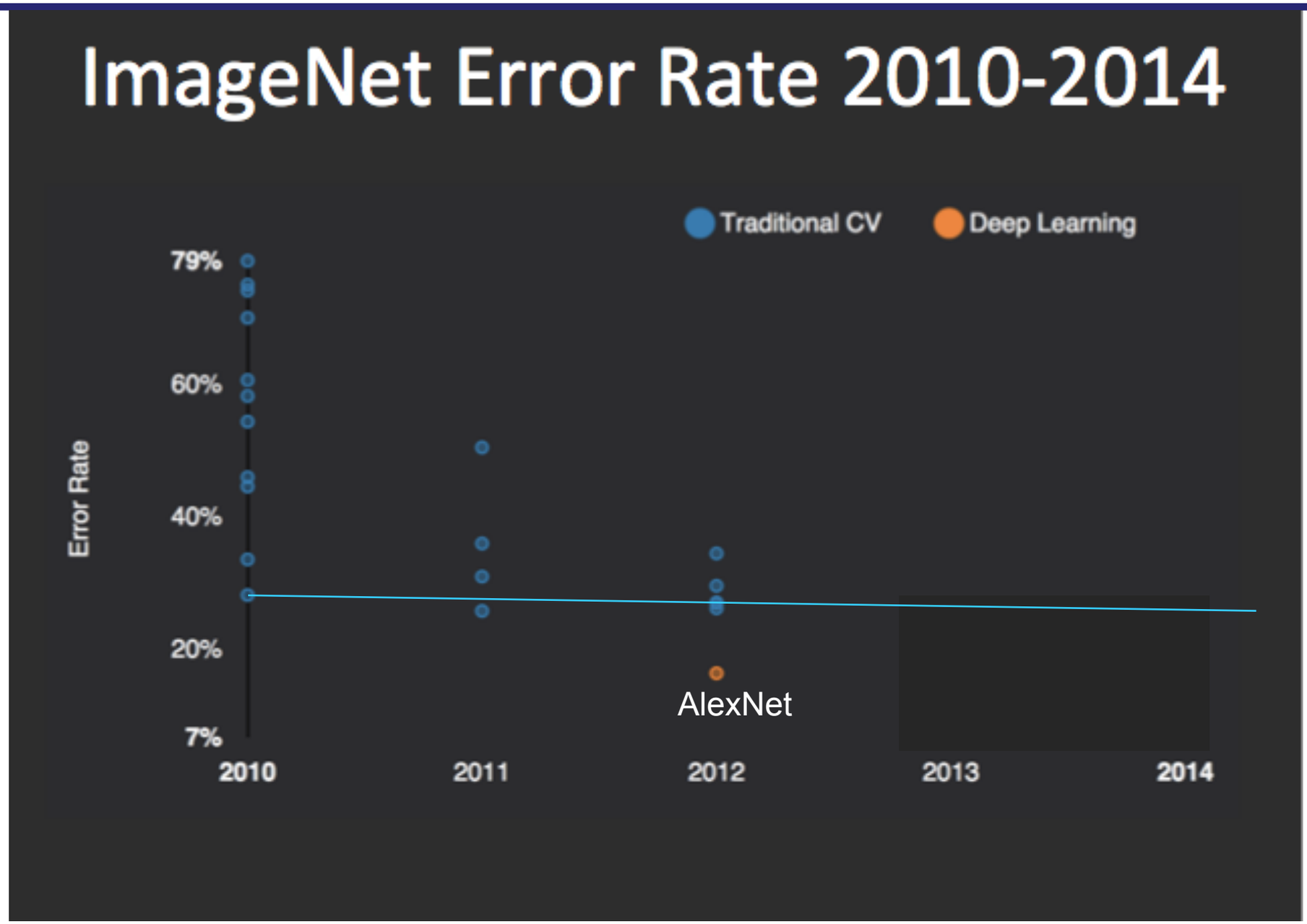
graph credit Matt Zeiler, Clarifai

Performance



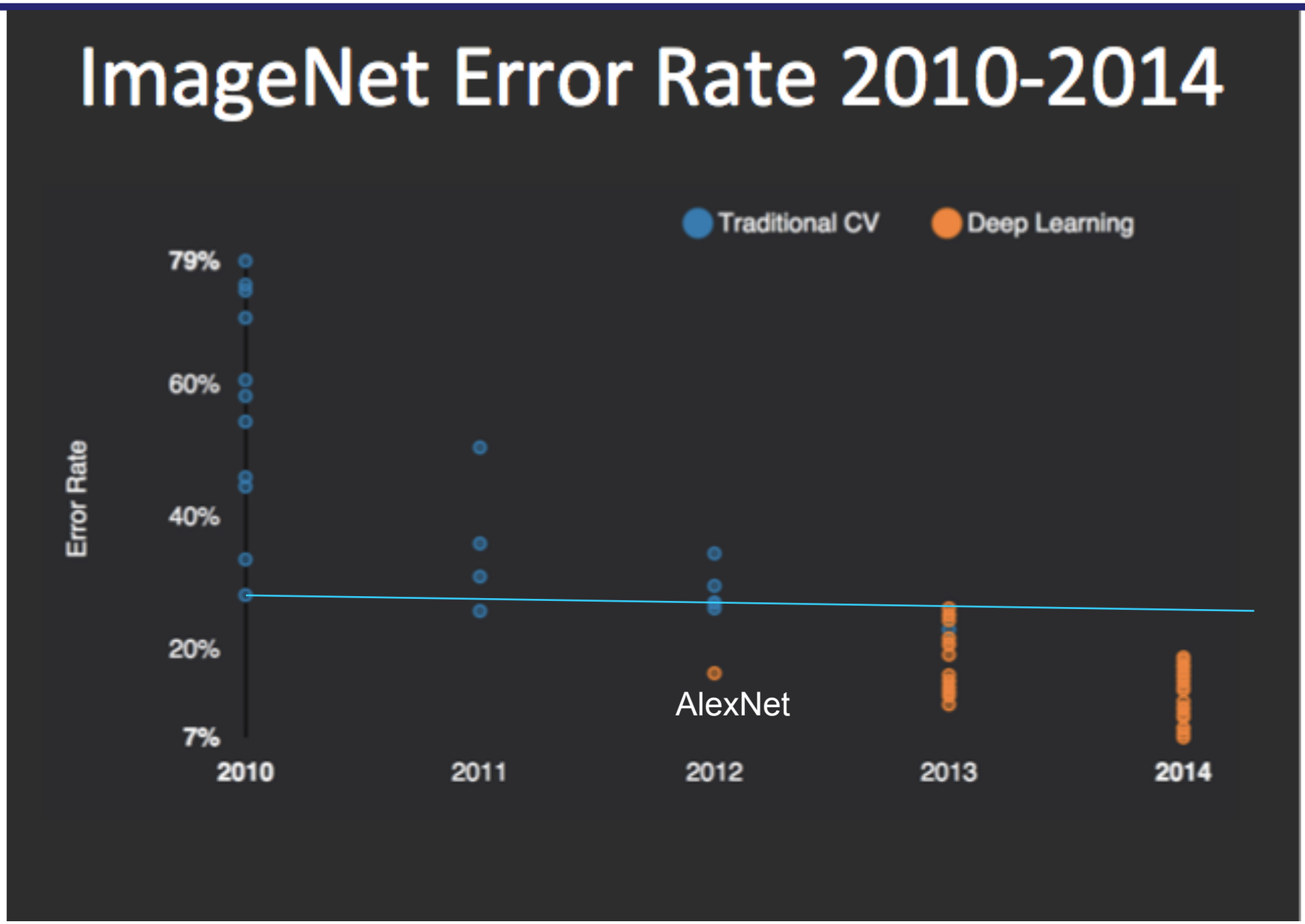
graph credit Matt Zeiler, Clarifai

Performance



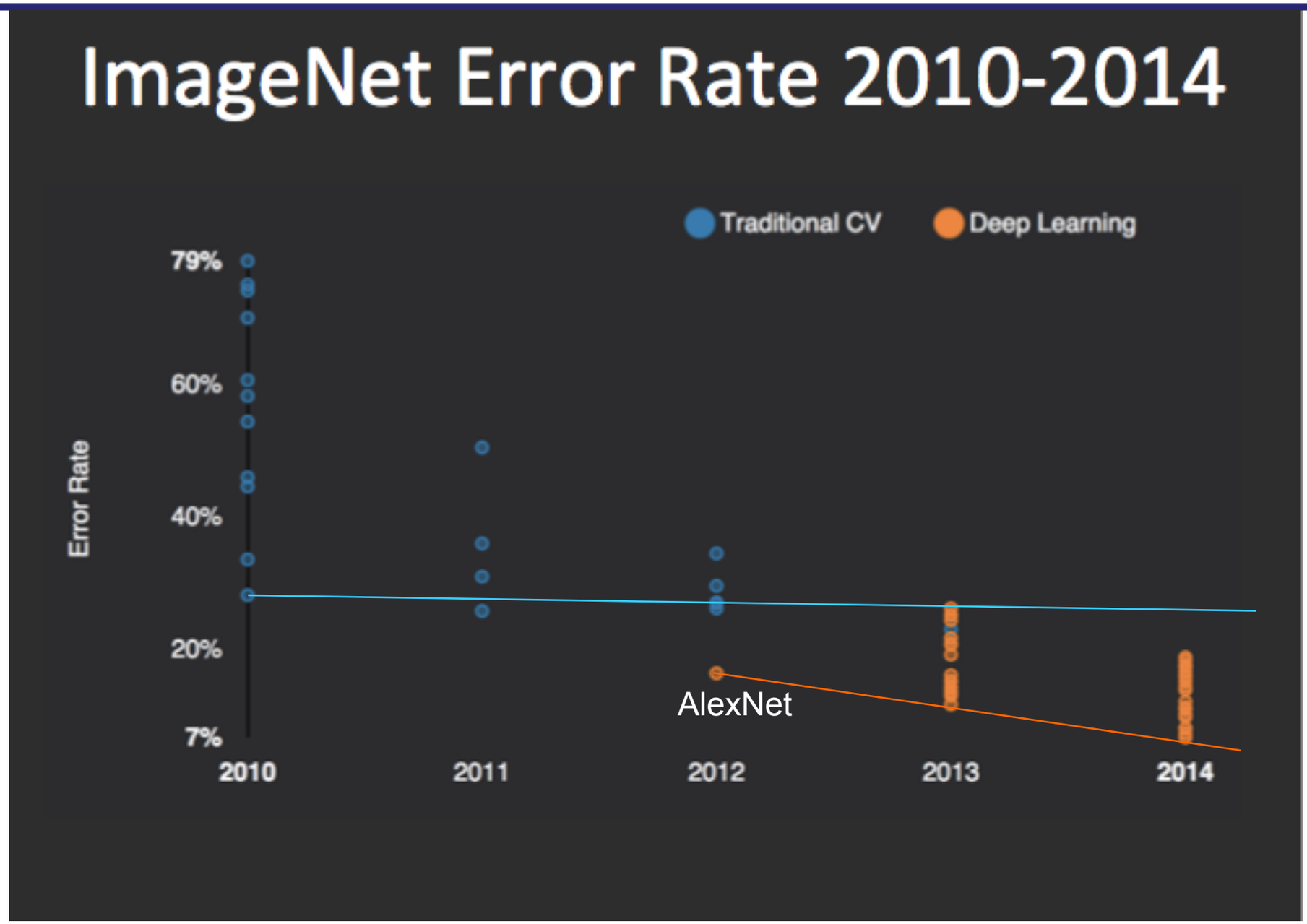
graph credit Matt Zeiler, Clarifai

Performance



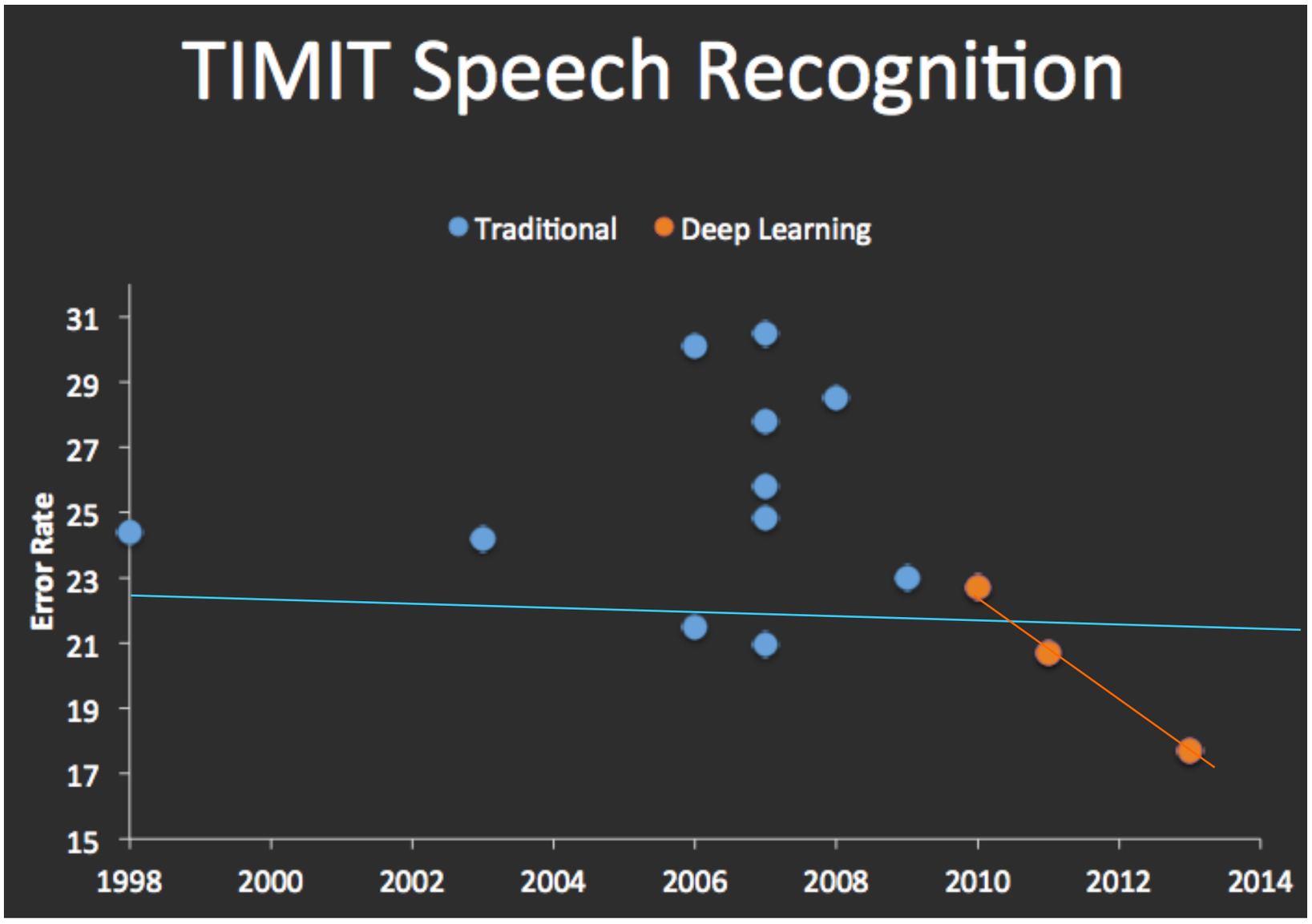
graph credit Matt Zeiler, Clarifai

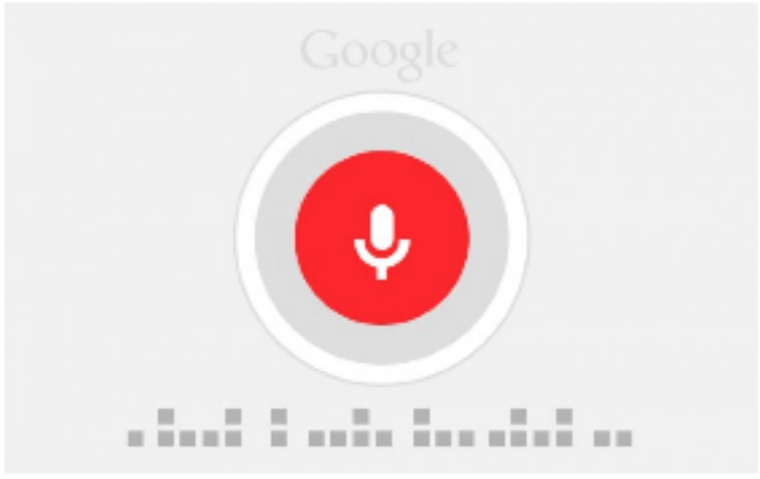
Performance



graph credit Matt Zeiler, Clarifai

Speech Recognition





graph credit Matt Zeiler, Clarifai

MS COCO Image Captioning Challenge

"man in black shirt is playing guitar."

"construction worker in orange safety vest is working on road."

"two young girls are playing with lego toy."

"boy is doing backflip on wakeboard."

"girl in pink dress is jumping in air."

"black and white dog jumps over bar."

"young girl in pink shirt is swinging on swing."

"man in blue wetsuit is surfing on wave."

Karpathy & Fei-Fei, 2015; Donahue et al., 2015; Xu et al, 2015; more

Visual QA Challenge

Stanislaw Antol, Aishwarya Agrawal, Jiasen Lu, Margaret Mitchell, Dhruv Batra, C. Lawrence Zitnick, De

What vegetable is on the plate?
Neural Net: broccoli
Ground Truth: broccoli

What color are the shoes on the person's feet ?
Neural Net: brown
Ground Truth: brown

How many school busses are there?
Neural Net: 2
Ground Truth: 2

What sport is this?
Neural Net: baseball
Ground Truth: baseball

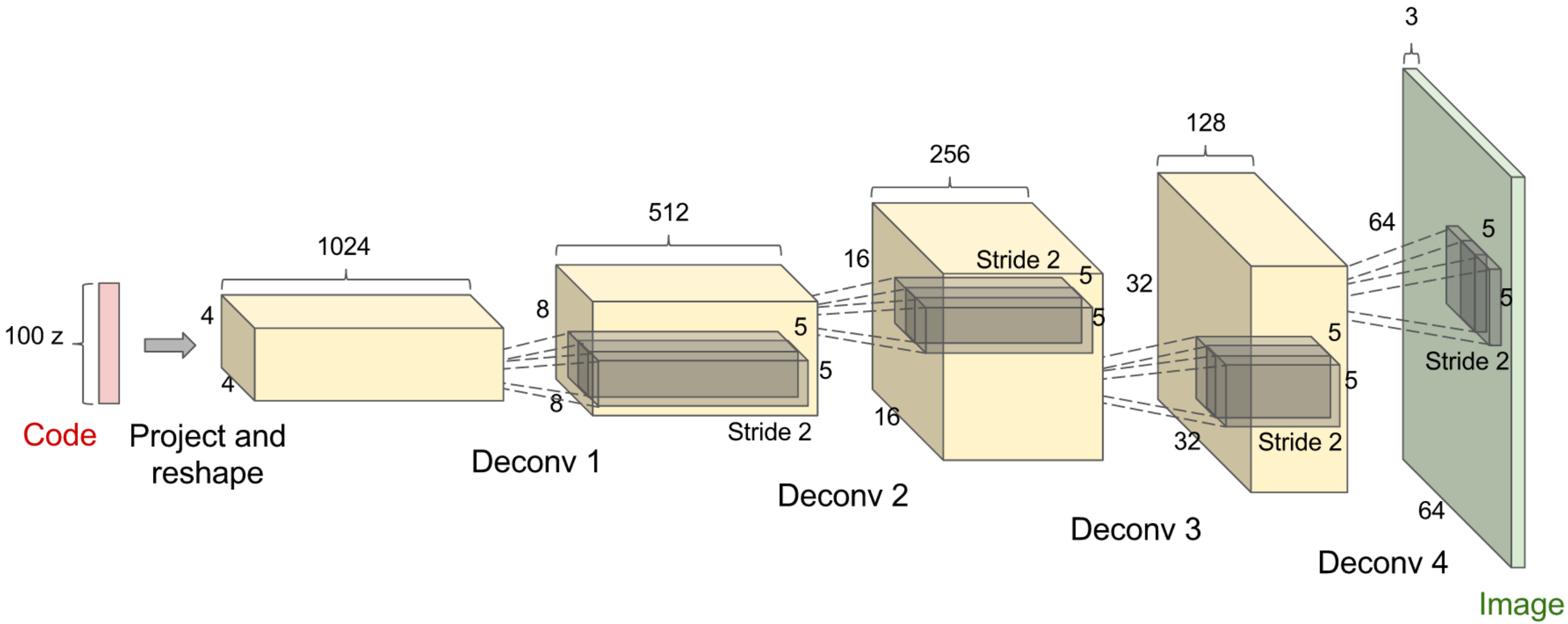
What is on top of the refrigerator?
Neural Net: magnets
Ground Truth: cereal

What uniform is she wearing?
Neural Net: shorts
Ground Truth: girl scout

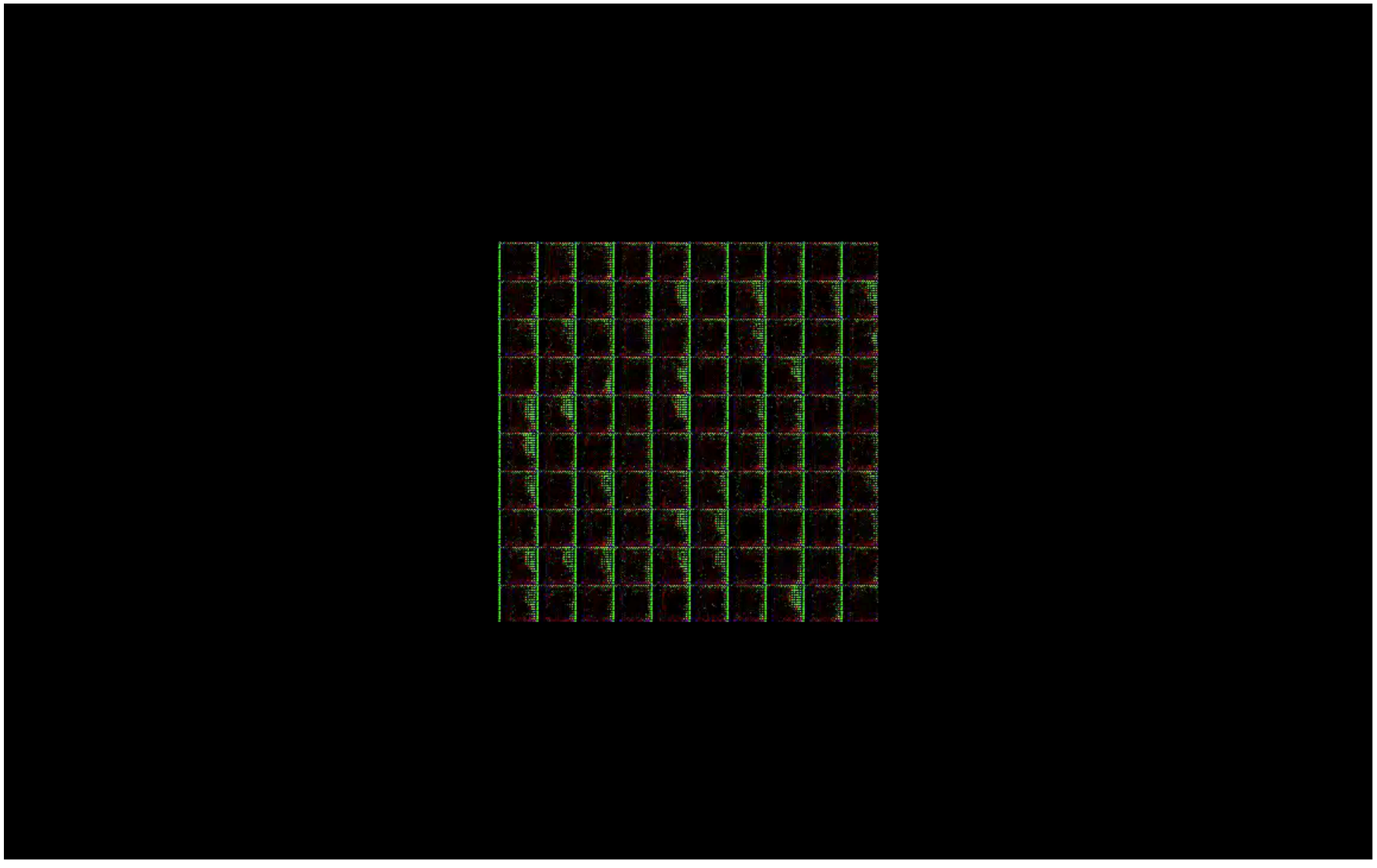
What is the table number?
Neural Net: 4
Ground Truth:40

What are people sitting under in the back?
Neural Net: bench
Ground Truth: tent

Image Generation – DC-GAN

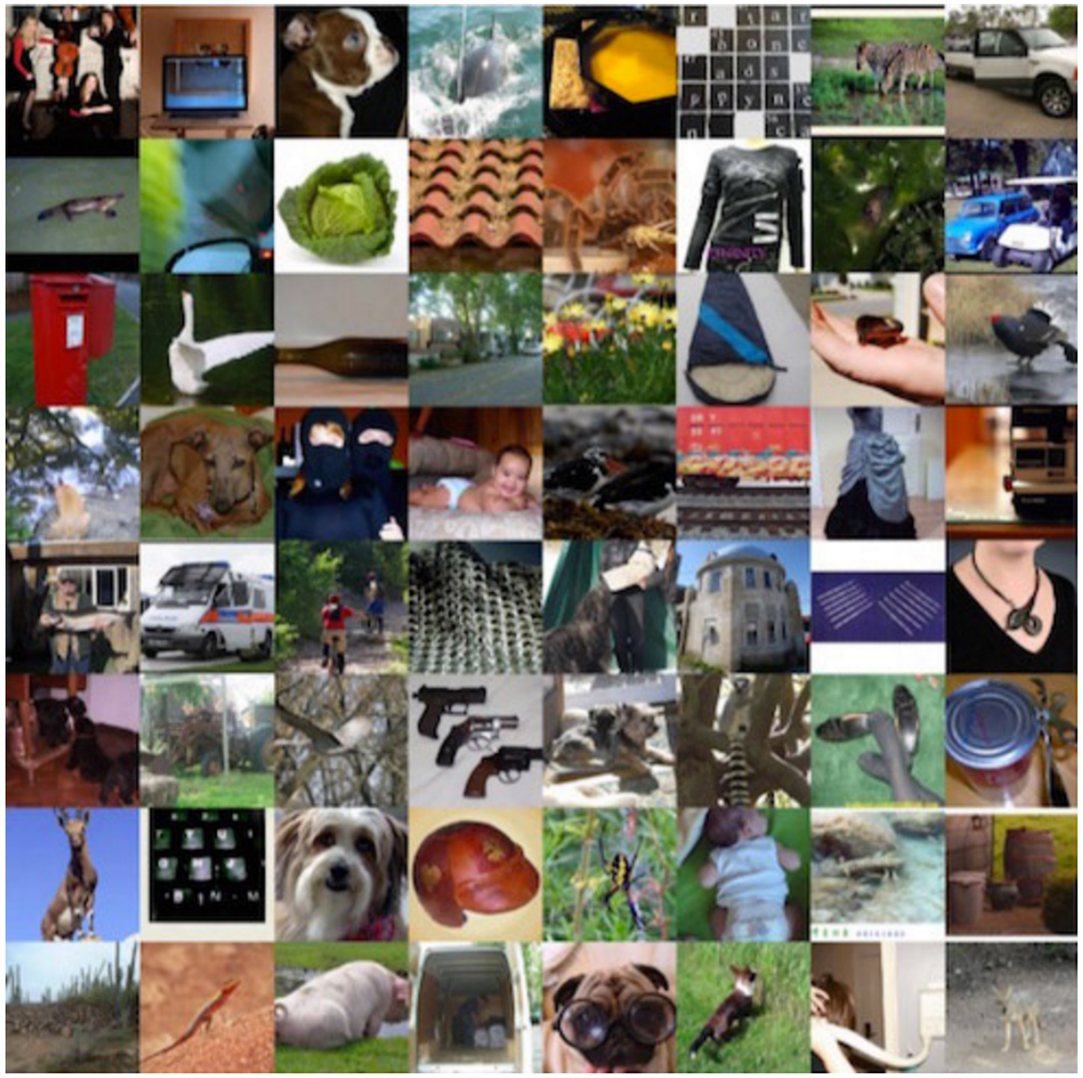


Training



Tim Salimans, Ian Goodfellow, Wojciech Zaremba, Vicki Cheung, Alec Radford, Xi Chen, 2016

Comparison with Real Images

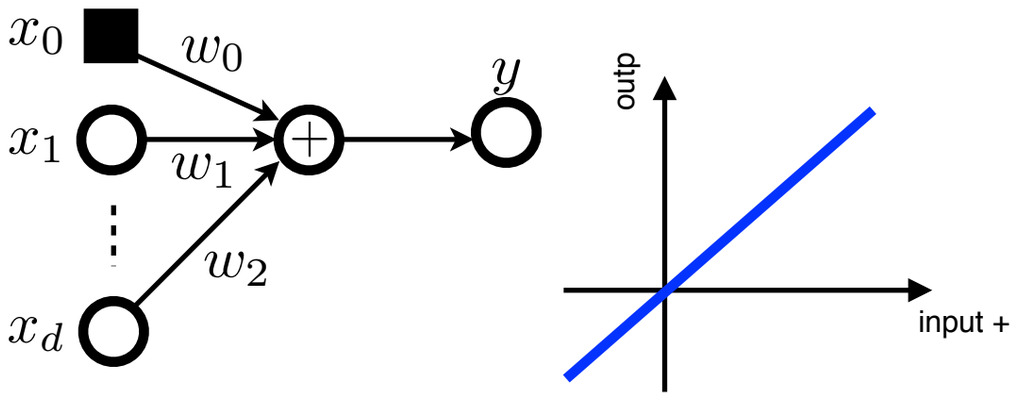


Tim Salimans, Ian Goodfellow, Wojciech Zaremba, Vicki Cheung, Alec Radford, Xi Chen, 2016

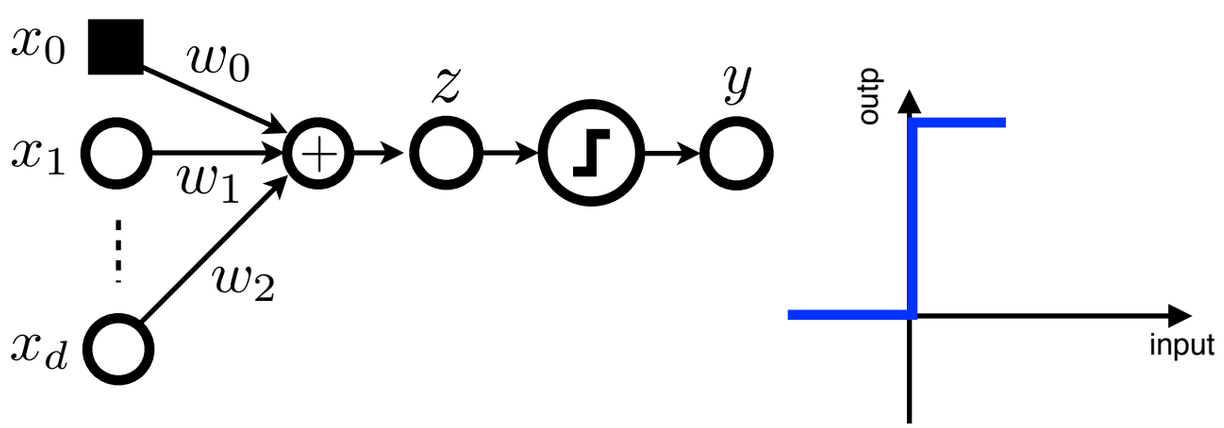
MODELING NEURONS

Perceptron Architecture

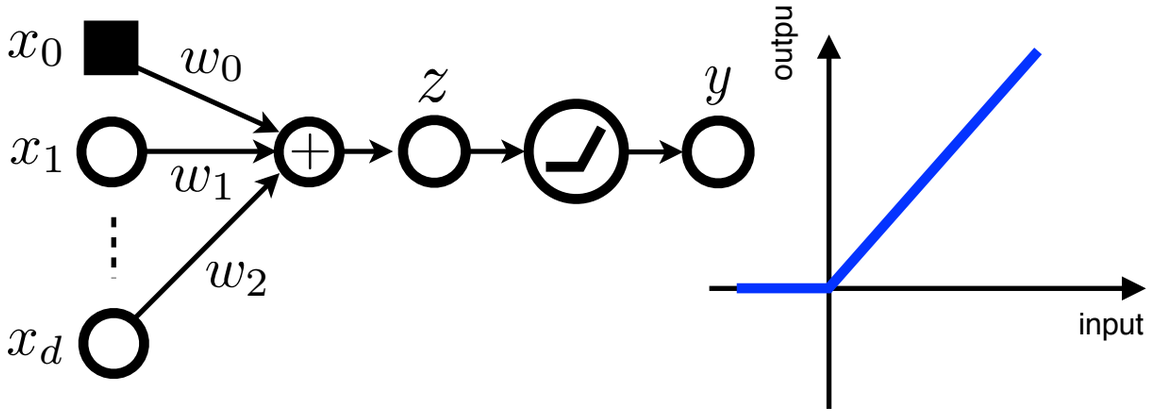
Linear Neuron



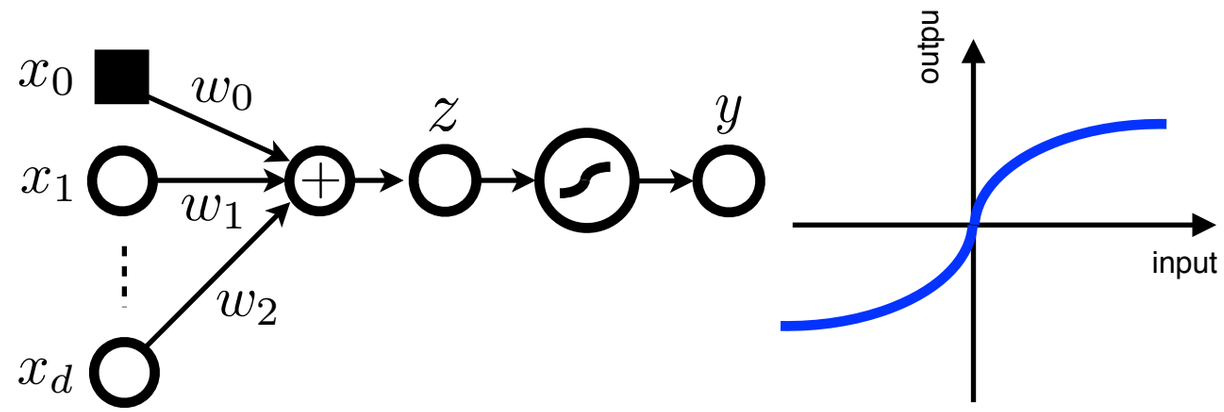
Binary Threshold Neuron



Rectified Linear Neuron



Sigmoid Neuron



MODELING NEURONS

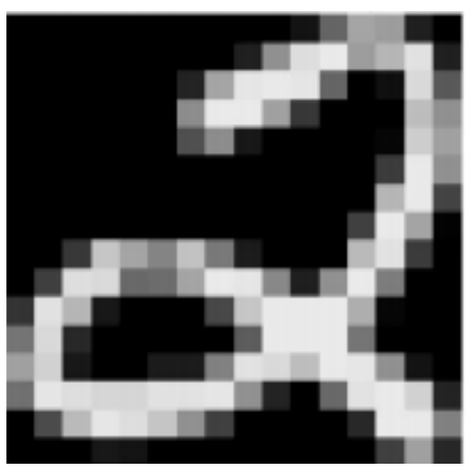
Supervised Learning Example: 2-Category Image Classification

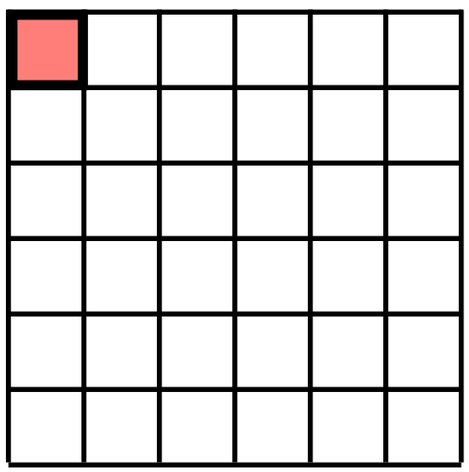
Training Pair: $(\mathbf{x}_k, y_k)_{k=1}^K$

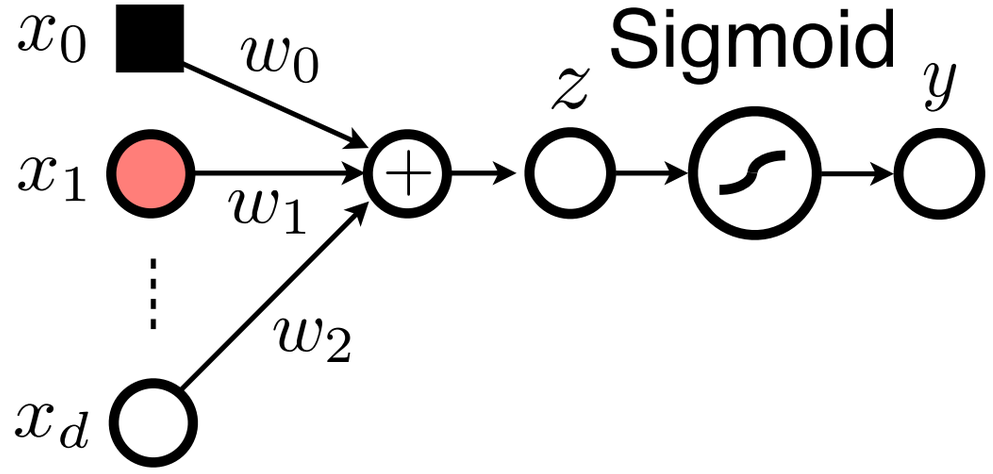
Input: $\mathbf{x}_k \in \mathbb{R}^d$

Output: $\mathbf{y}_k \in \mathbb{R}$

NN Classifier: $y = \sigma\left(\sum_{i=0}^{d} w_i x_{ki}\right) = h(\mathbf{x}_k; \mathbf{w})$







Loss Function: $E(\mathbf{w}) := \sum_{k=1}^{K} \left(h(\mathbf{x}_k; \mathbf{w}) - y_k\right)^2$

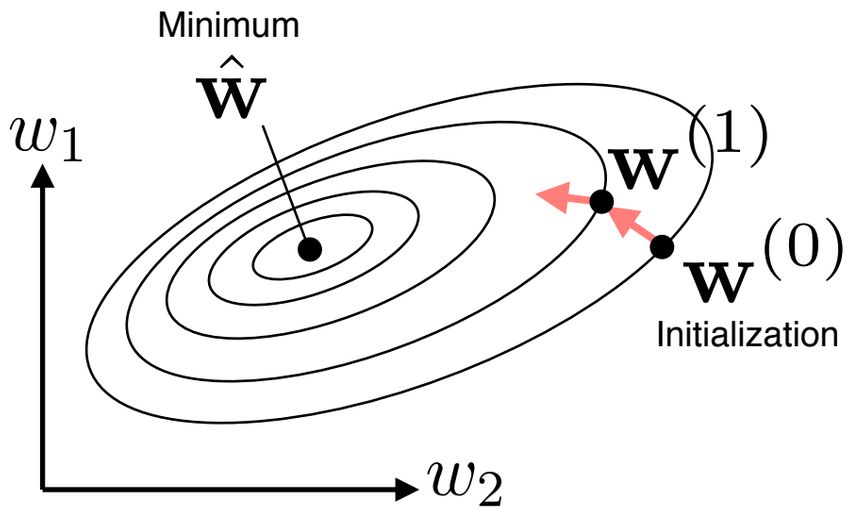
What are the right weights?

TRAINING A NEURAL NETWORK

Gradient Descent: Take a Step Proportional to the Negative of the Gradient Direction

$$y = \sigma\Big(\sum_{i=0}^{d} w_i x_{ki}\Big) = h(\mathbf{x}_k; \mathbf{w}) \qquad E(\mathbf{w}) := \sum_{k=1}^{K} \Big(h(\mathbf{x}_k; \mathbf{w}) - y_k\Big)^2$$

$$\mathbf{w}^* = \arg\min_{\mathbf{w}} E(\mathbf{w})$$



Learning Rate

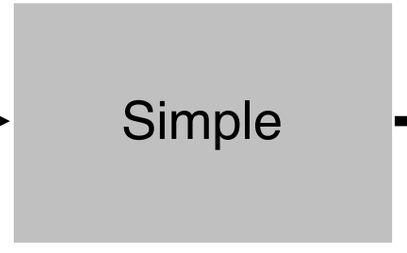
$$\mathbf{w}^{(t+1)} = \mathbf{w}^{(t)} - \eta \frac{\partial E(\mathbf{w})}{\partial \mathbf{w}}\bigg|_{\mathbf{w}^{(t)}}$$

$$\frac{\partial E(\mathbf{w})}{\partial \mathbf{w}} = \left(\frac{\partial E(\mathbf{w})}{\partial w_0} \quad \frac{\partial E(\mathbf{w})}{\partial w_1} \quad \cdots \quad \frac{\partial E(\mathbf{w})}{\partial w_d}\right)$$ Gradient

LIMITATIONS OF THE PERCEPTRON

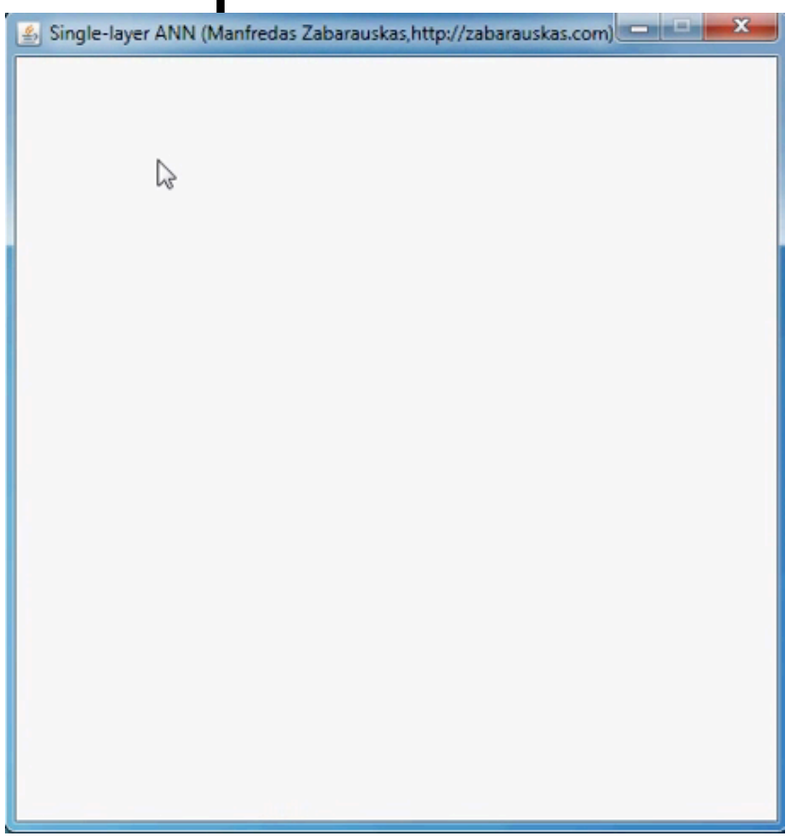
Linear Separability

Fixed



Trainabl

If you craft enough good features, perceptrons can be powerful.



Based on a slide by Marc'Aurelio Ranzato, Yann

"COMPLICATED" CLASSIFIERS

Composition: Functions of Functions

Given a library of simple function

$\sin(x)$

$\log(x)$

x^4

$\tanh(x)$

$\vdots$

Compose into a "complicated" function

Compositionality

Deep Learning

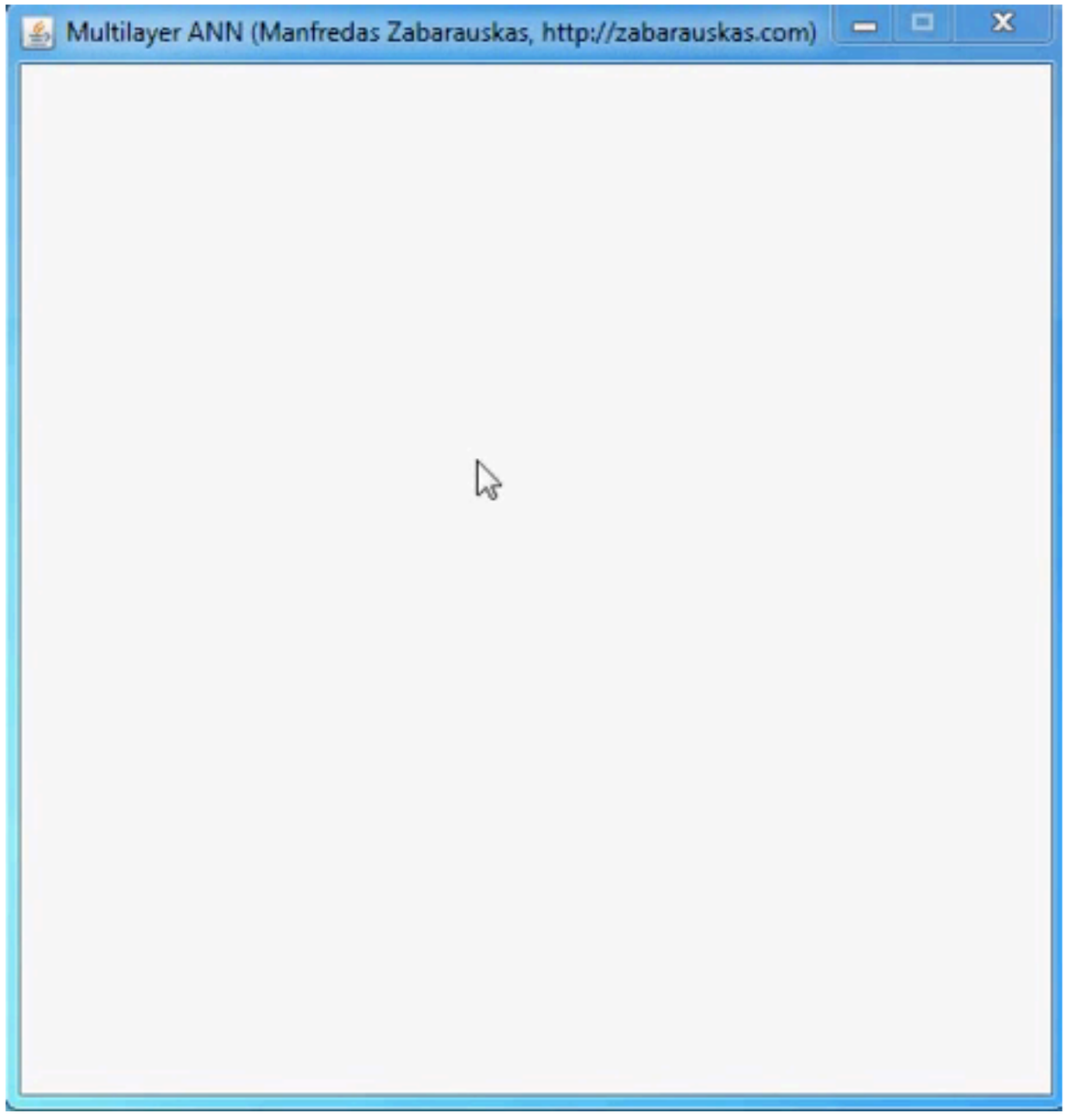
$$f(x) = g_1(g_2(\cdots g_n(x)))$$

$$f(x) = \sin(\log(\cdots \tanh(x))^3))$$

Based on a slide by Marc'Aurelio Ranzato, Yann

TRAINING A MULTILAYER PERCEPTRON

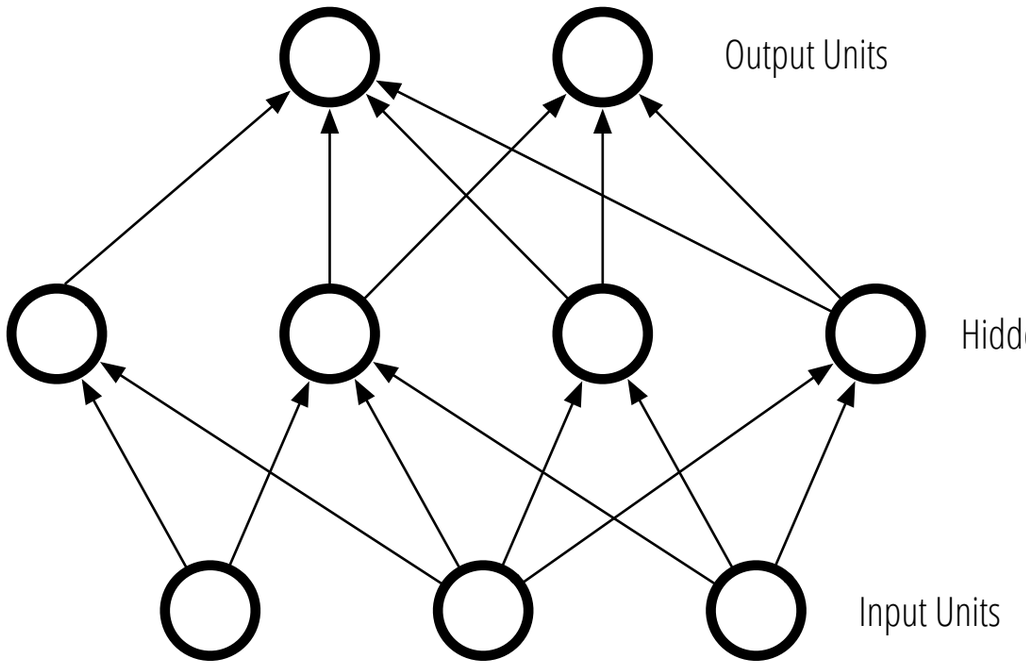
Optimizing Weights for a Multi-Layer Neural Network



Visualization Credit: Manfred Zabarau

NEURAL NETWORK ARCHITECTURES

Feed-forward Neural Networks

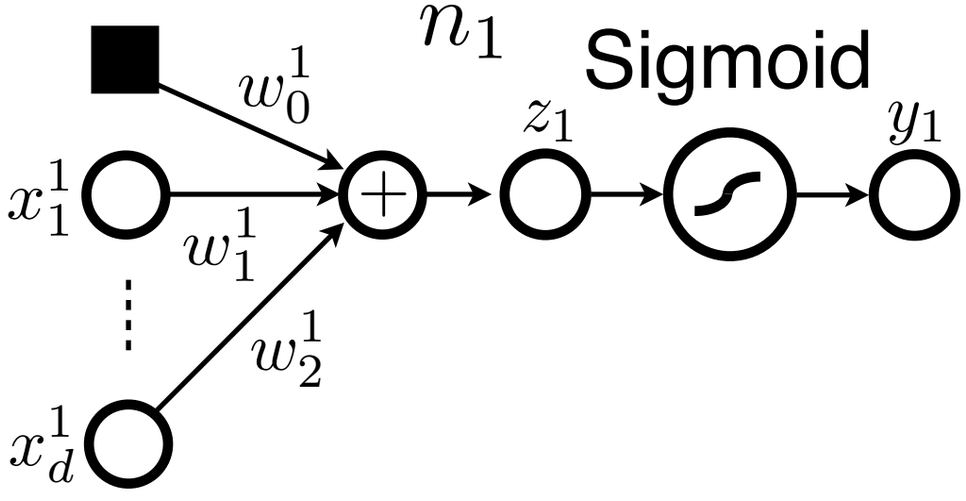


Information enters through the input units and flows in one direction, through hidden layers, to the output

Each hidden layer changes similarity relationships, producing a new representation of the input

TRAINING A MULTILAYER PERCEPTRON

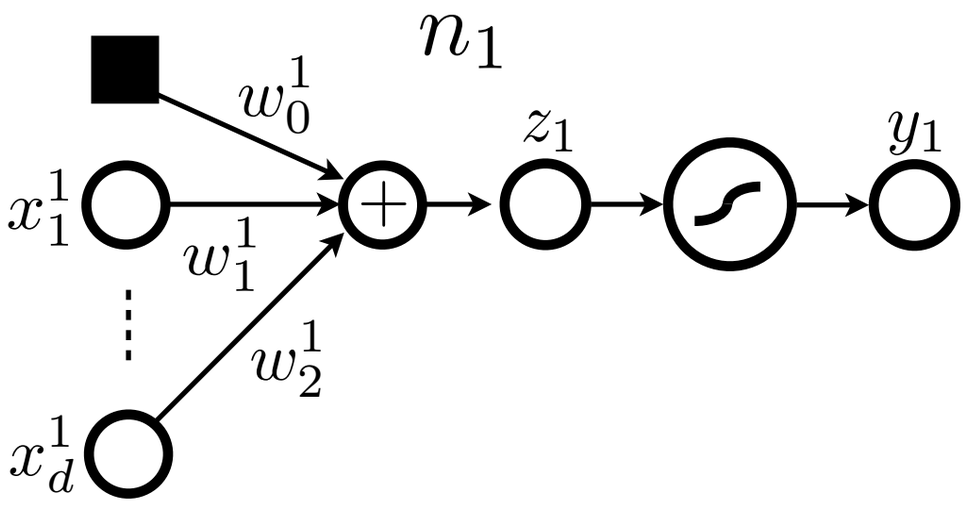
Optimizing Weight for a Multi-layer Neural Network

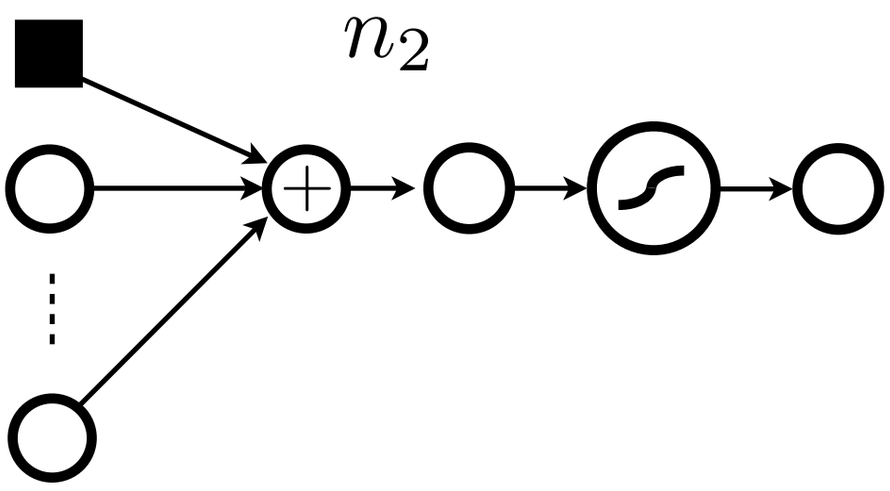


$$z_{kj} = \sum_{i=0}^{d} w_i^j x_{ki}^j$$

TRAINING A MULTILAYER PERCEPTRON

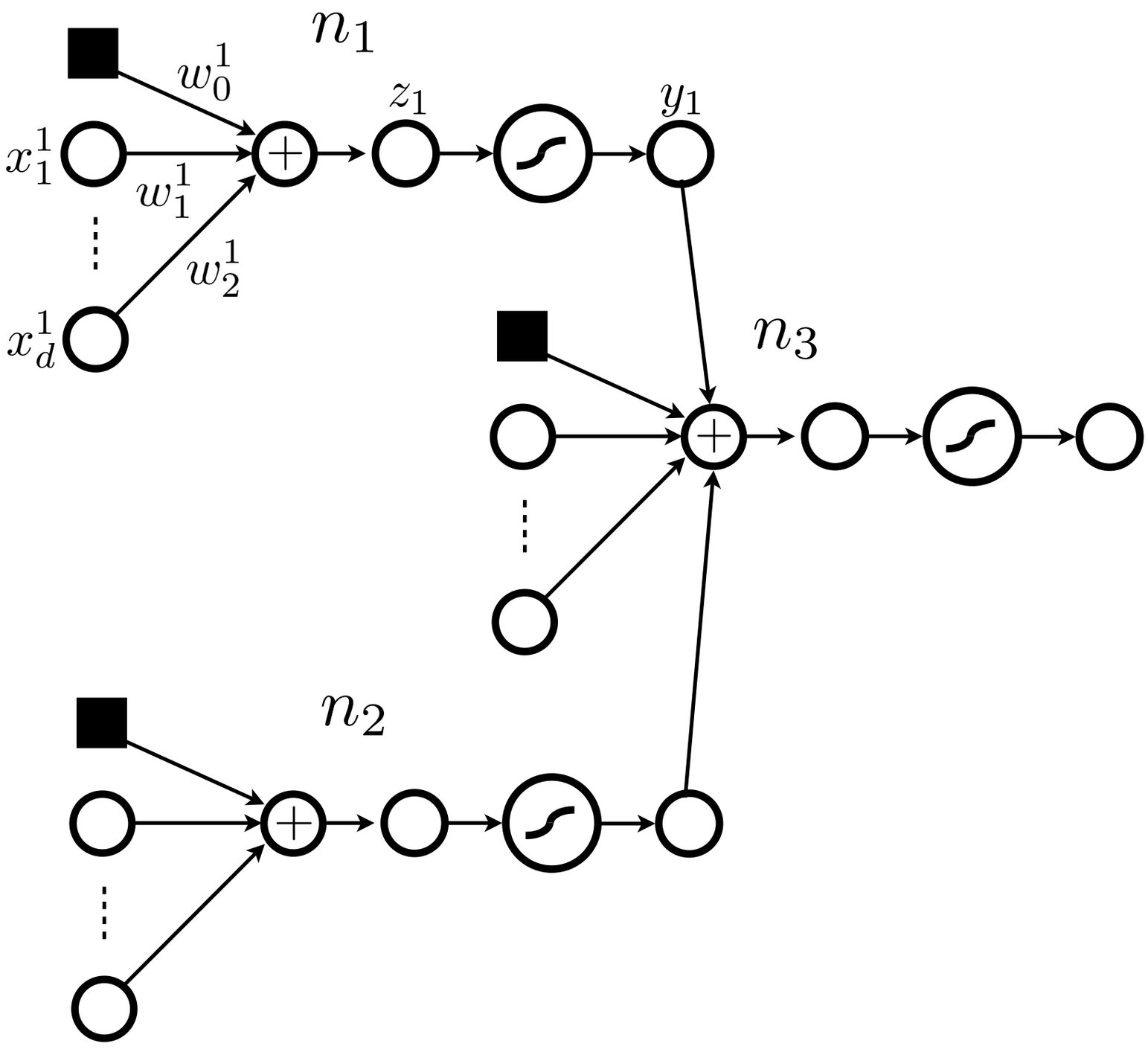
Optimizing Weight for a Multi-layer Neural Network





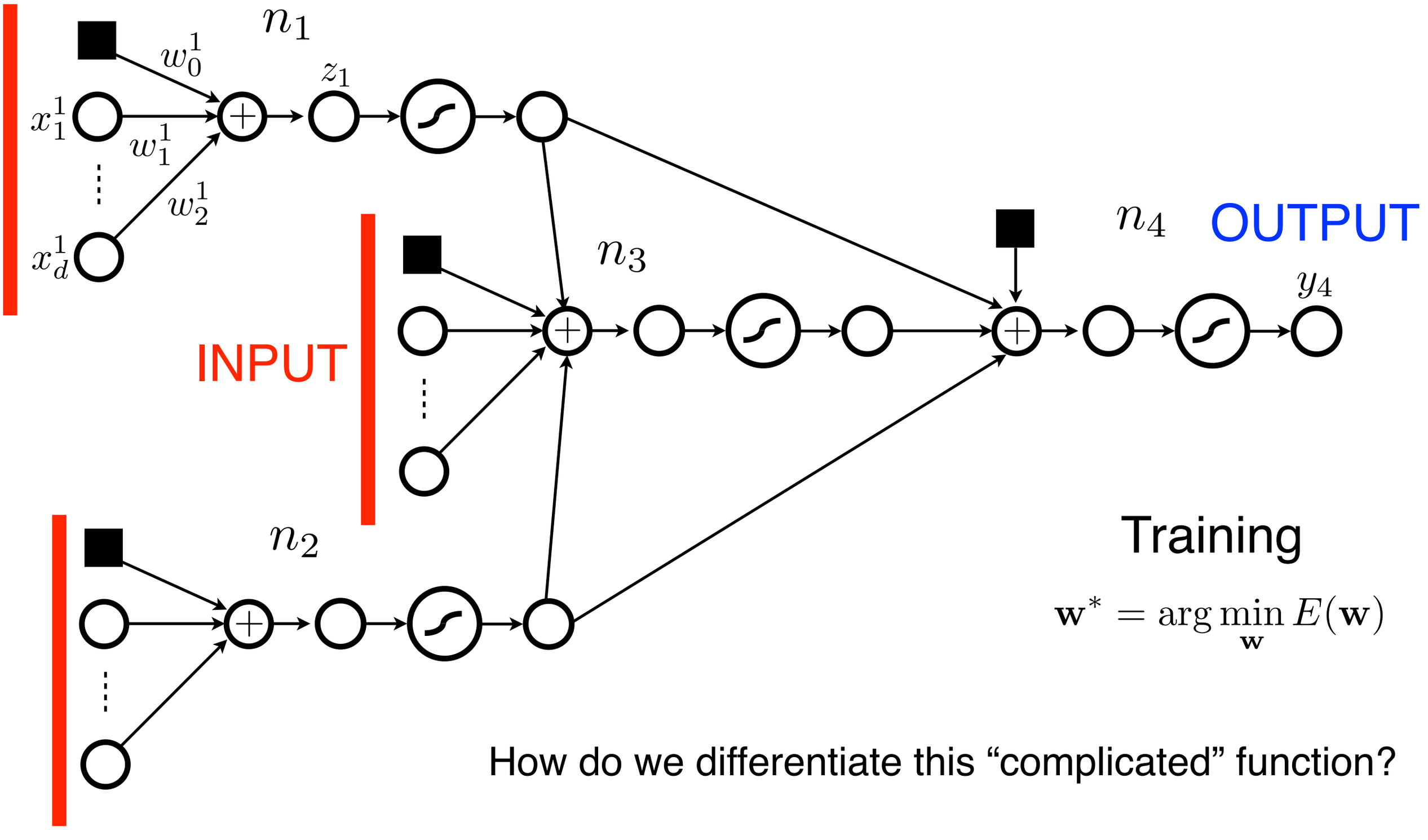
TRAINING A MULTILAYER PERCEPTRON

Optimizing Weight for a Multi-layer Neural Network



TRAINING A MULTILAYER PERCEPTRON

Optimizing Weight for a Multi-layer Neural Network



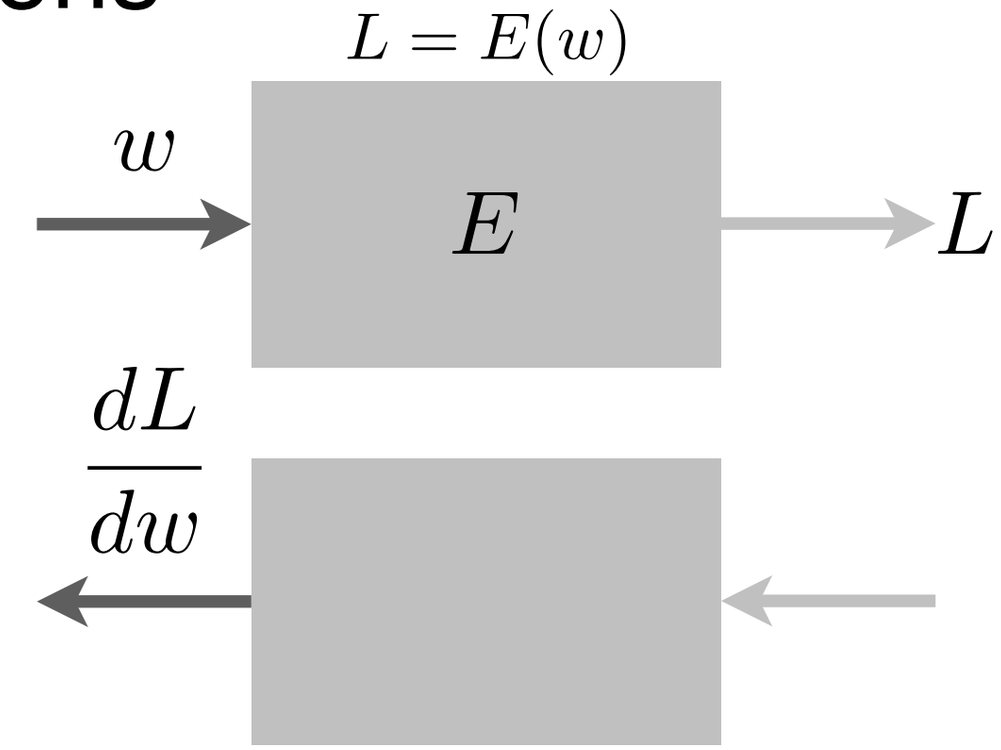
Training

$$\mathbf{w}^* = \arg\min_{\mathbf{w}} E(\mathbf{w})$$

How do we differentiate this “complicated” function?

CHAIN RULE

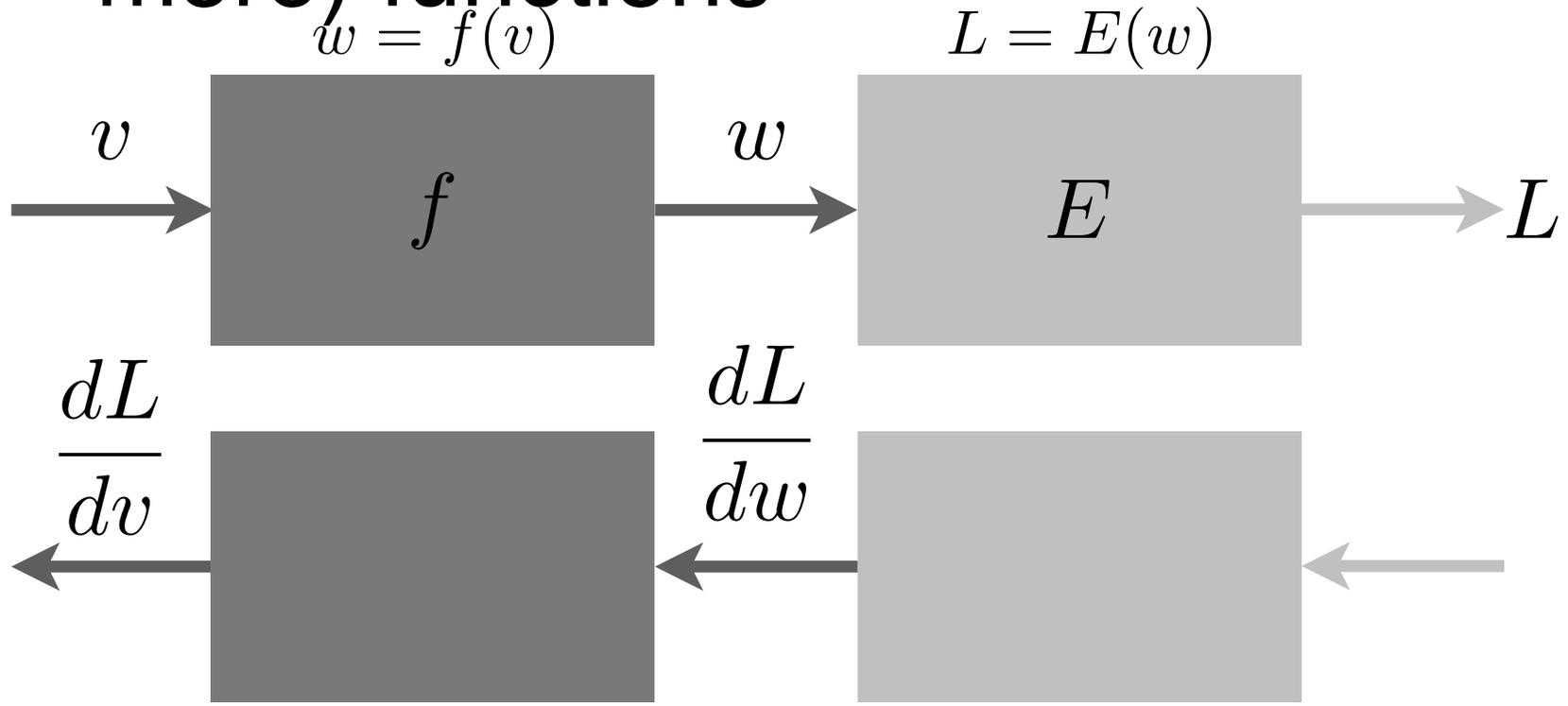
Computing the derivative of a composition of two (or more) functions



Based on slides by Marc'Aurelio Ranzato, Yann

CHAIN RULE

Computing the derivative of a composition of two (or more) functions

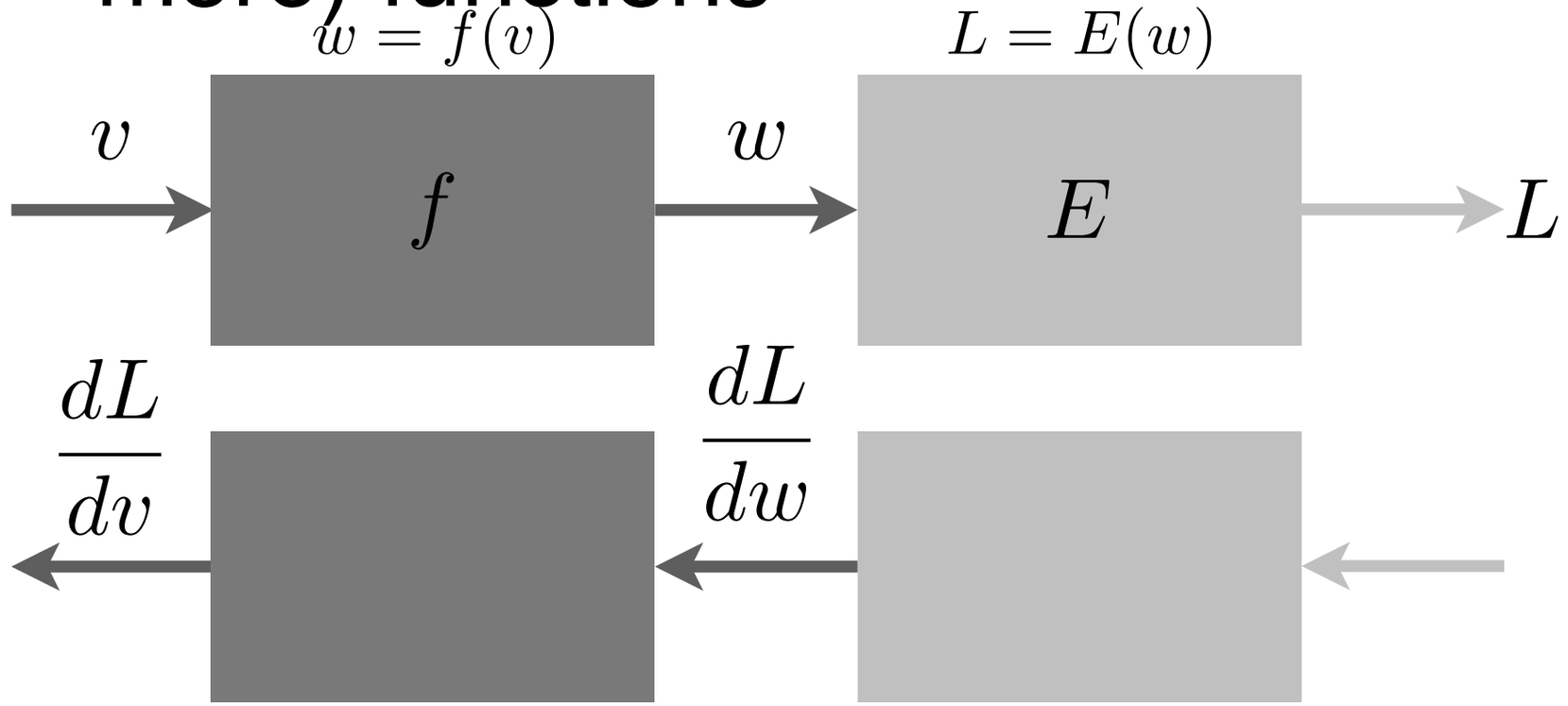


Given $w = f(v)$ and $\frac{dL}{dw}$, find $\frac{dL}{dv}$.

Based on slides by Marc'Aurelio Ranzato, Yann

CHAIN RULE

Computing the derivative of a composition of two (or more) functions



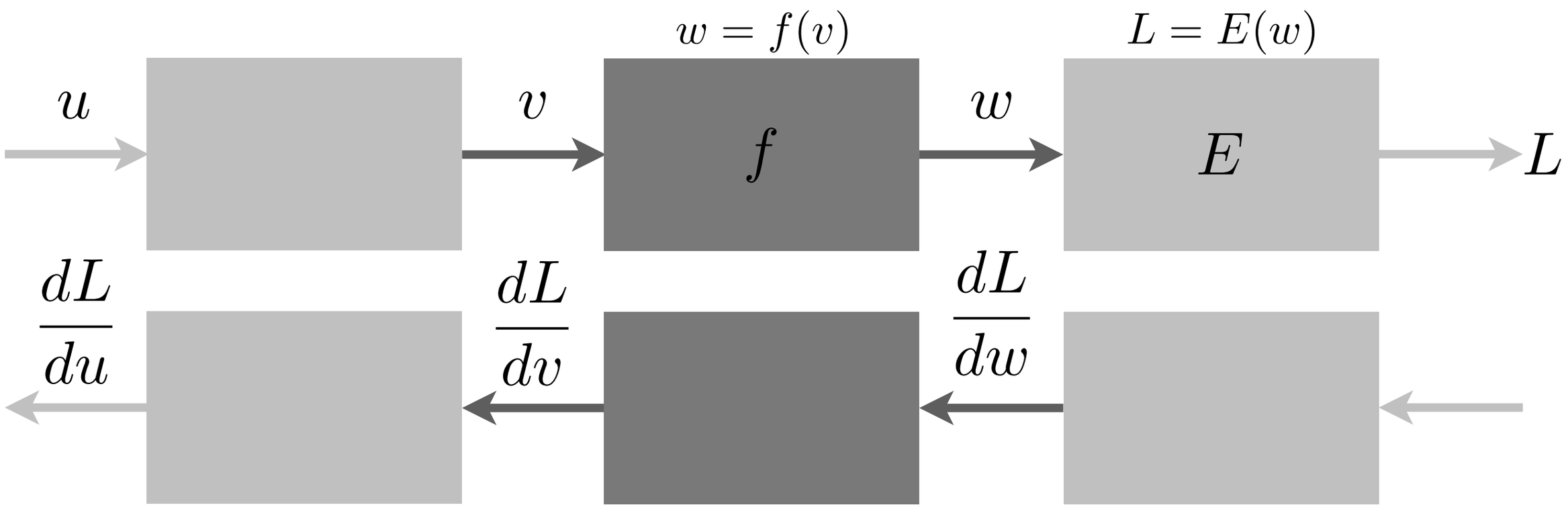
Given $w = f(v)$ and $\frac{dL}{dw}$, find $\frac{dL}{dv}$.

$$\frac{dL}{dv} = \underbrace{\frac{dL}{dw} \cdot \frac{dw}{dv}}_{\text{Chain Rule}} = \frac{dE(w)}{dw} \cdot \frac{df(v)}{dv}$$

Based on slides by Marc'Aurelio Ranzato, Yann

CHAIN RULE

All computations are local



$$\frac{dL}{dv} = \frac{dL}{dw} \cdot \frac{dw}{dv} = \frac{dE(w)}{dw} \cdot \frac{df(v)}{dv}$$

$$\frac{dL}{du} = \frac{dL}{dv} \cdot \frac{dv}{du}$$

Observations

1. This expression does not change with the addition of earlier unit

2. Derivaties computed in later units are used for derivative computation in later units

Based on slides by Marc'Aurelio Ranzato, Yann LeCun

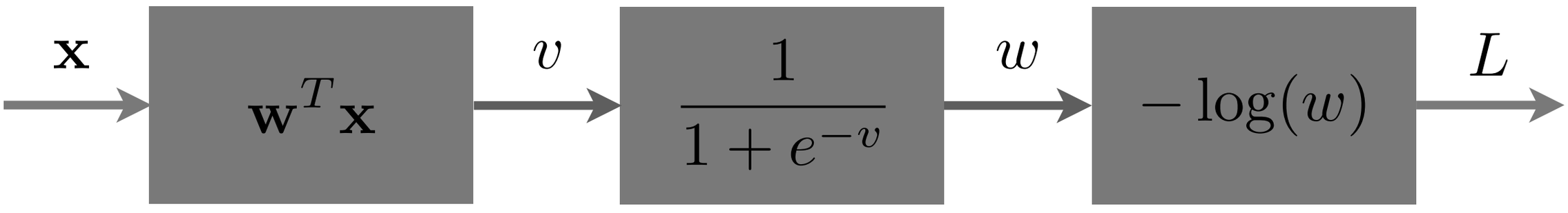
LOGISTIC REG. AS A COMPOSITION

Given a library of simple function

$$\sin(x)$$
$$\log(x)$$
$$x^4$$
$$\tanh(x)$$
$$\vdots$$

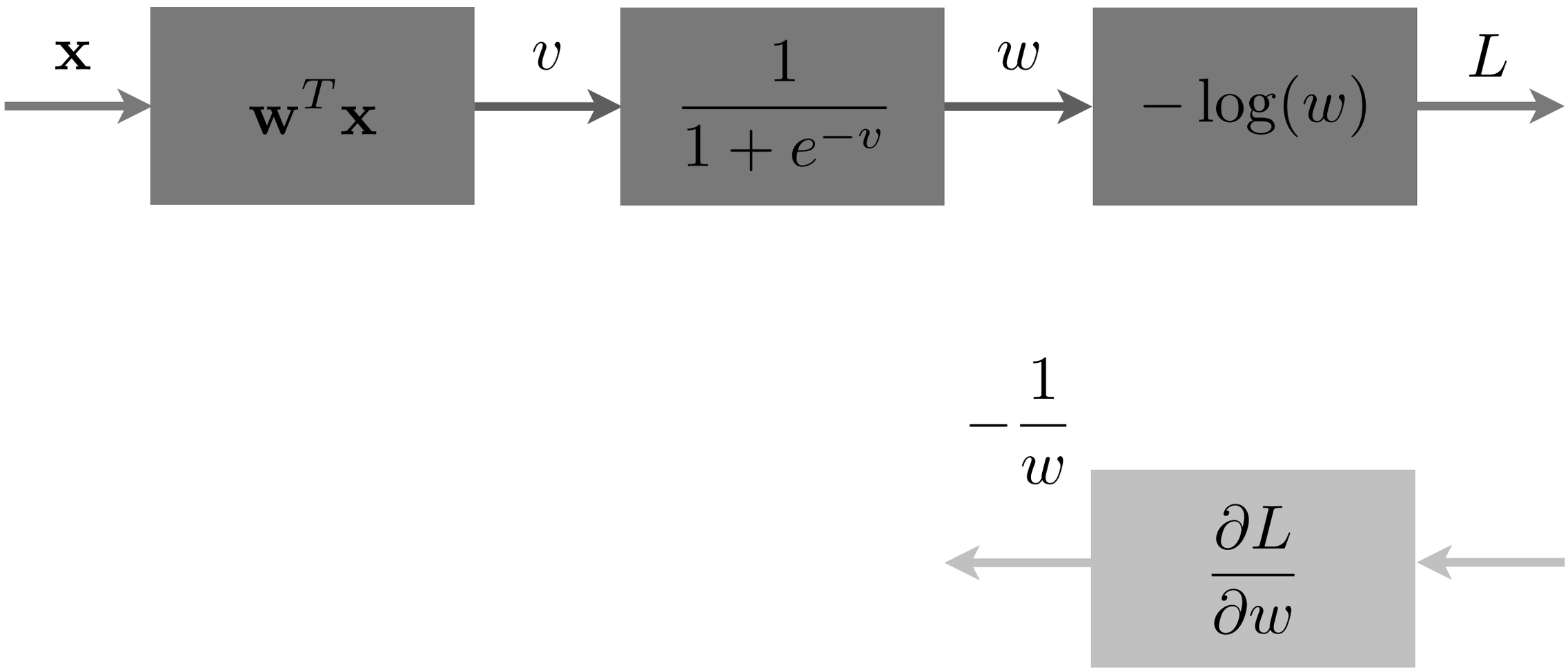
Compose into a "complicated" function

$$-\log\left(\frac{1}{1+e^{-\mathbf{w}^\intercal \mathbf{x}}}\right)$$



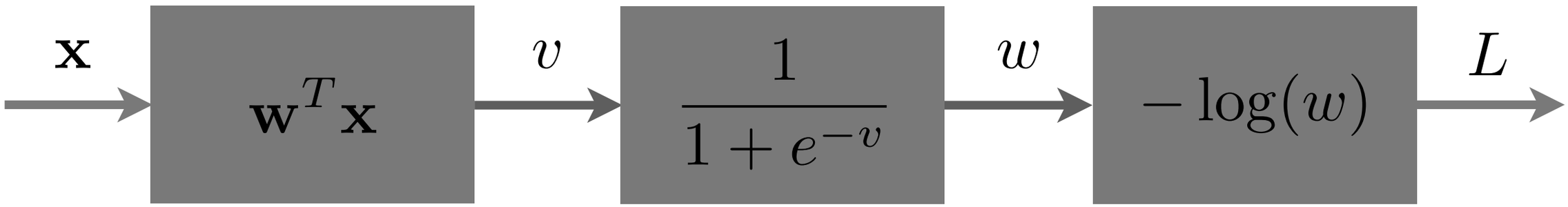
Based on slides by Marc'Aurelio Ranzato, Yann

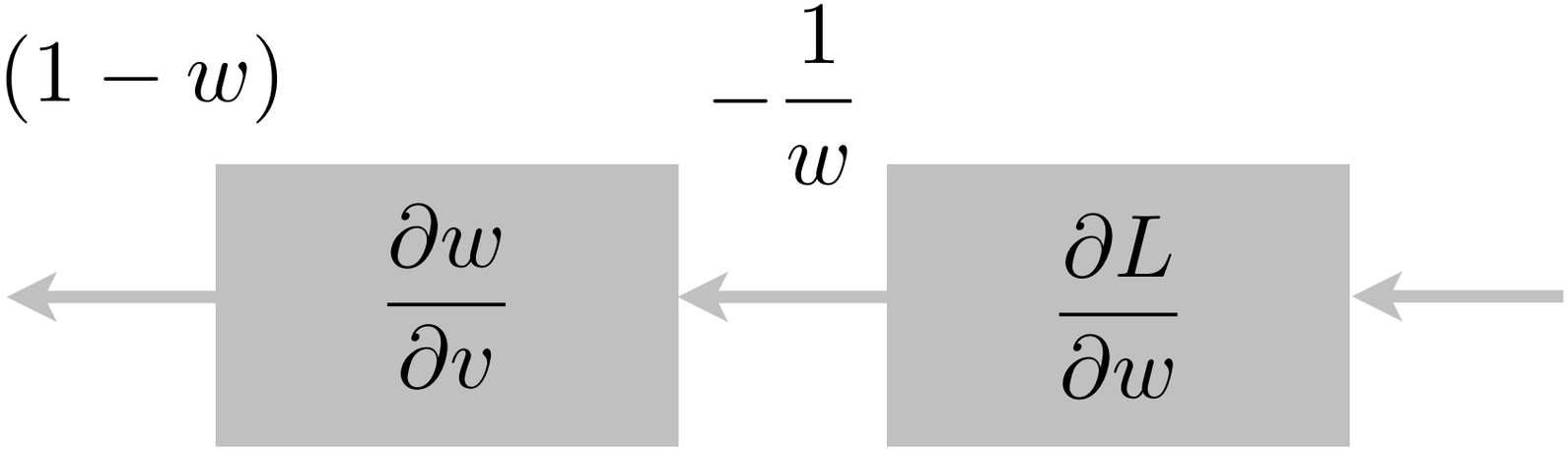
LOGISTIC REG. AS A COMPOSITION



Based on slides by Marc'Aurelio Ranzato, Yann

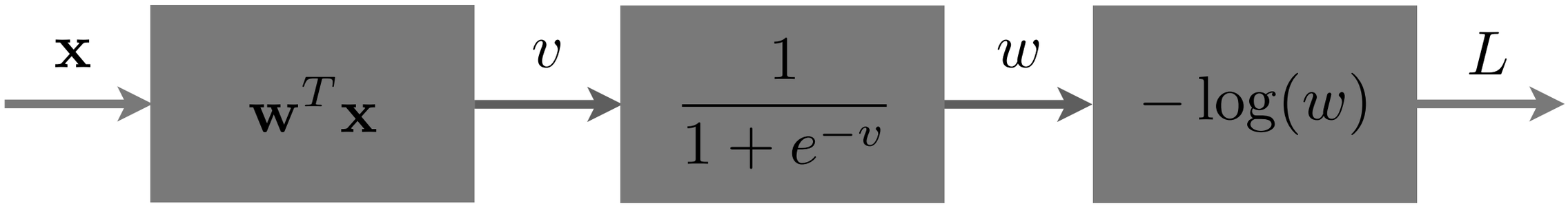
LOGISTIC REG. AS A COMPOSITION

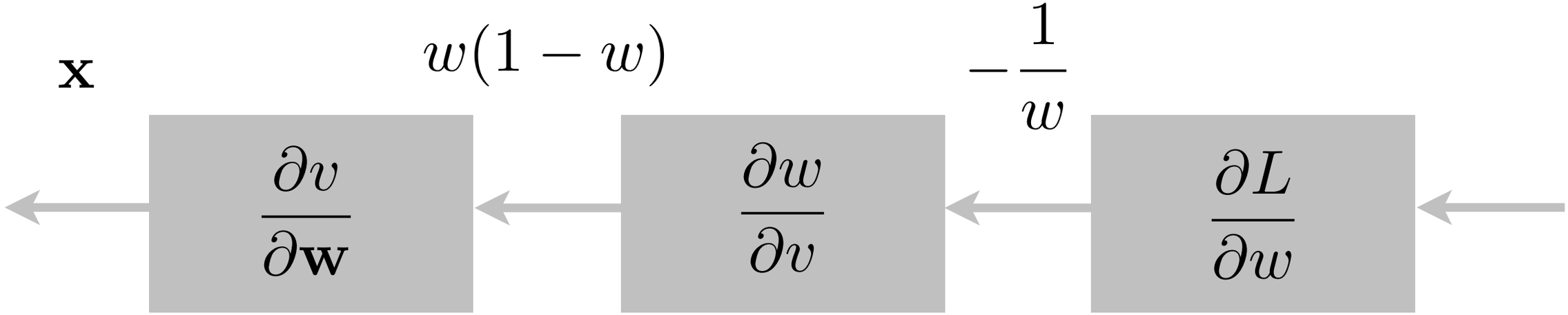




Based on slides by Marc'Aurelio Ranzato, Yann LeCun

LOGISTIC REG. AS A COMPOSITION



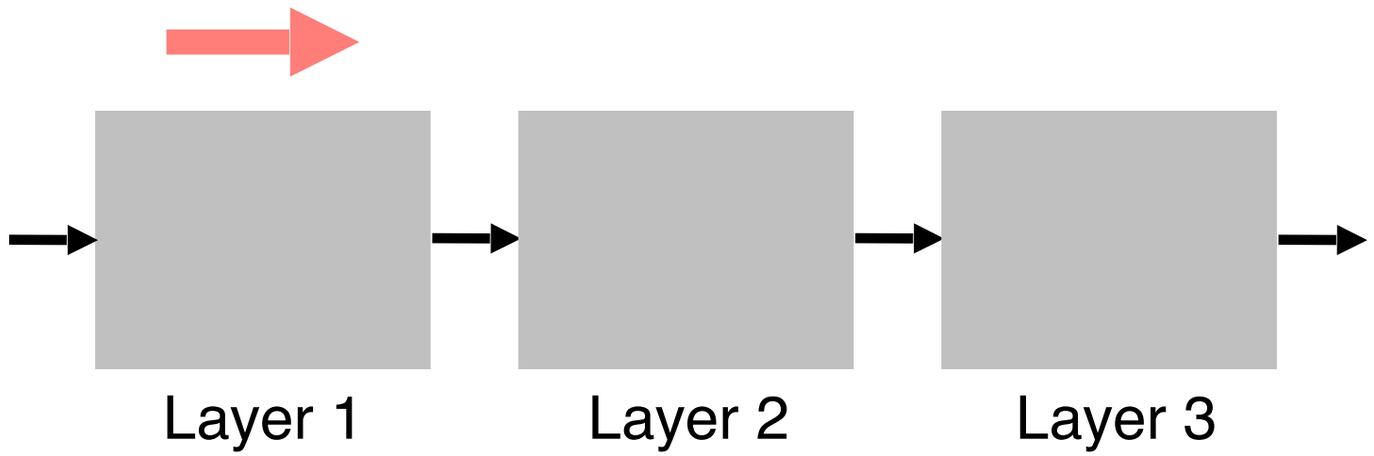


$$\frac{\partial L}{\partial \mathbf{w}} = \frac{\partial L}{\partial w} \cdot \frac{\partial w}{\partial v} \cdot \frac{\partial v}{\partial \mathbf{w}} = (w-1)\mathbf{x}$$

Based on slides by Marc'Aurelio Ranzato, Yann LeCun

NEURAL NETWORK TRAINING

Forward Propagation, Back Propagation, Gradient Descent



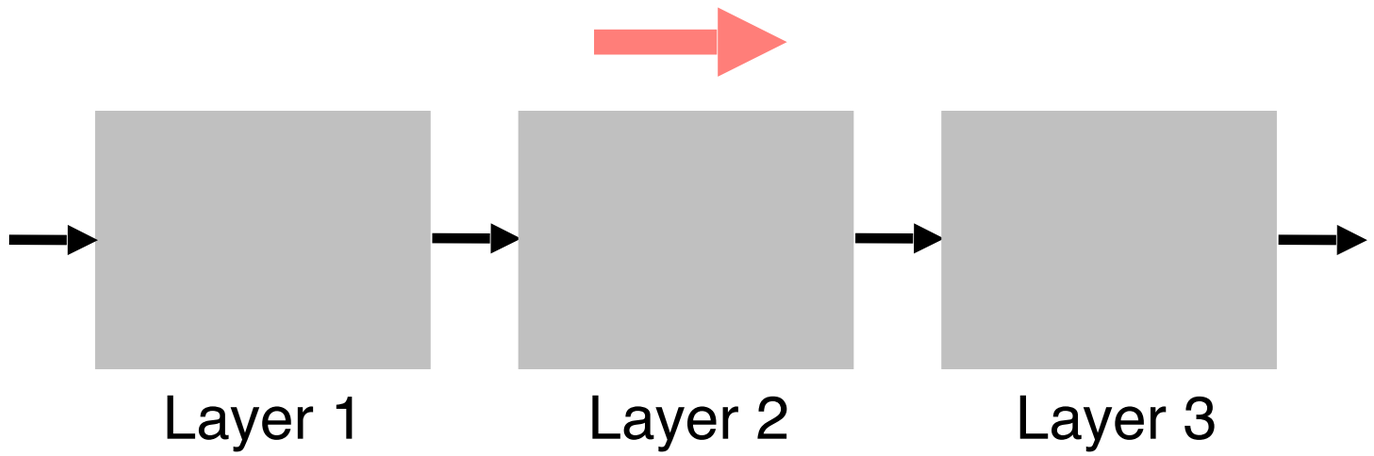
Step 1: Compute loss on mini-batch (Forward Pass)

Step 2: Compute gradients with respect to weights (Backward Pass)

Step 3: Use gradients to update weights (Gradient Descent)

NEURAL NETWORK TRAINING

Forward Propagation, Back Propagation, Gradient Descent



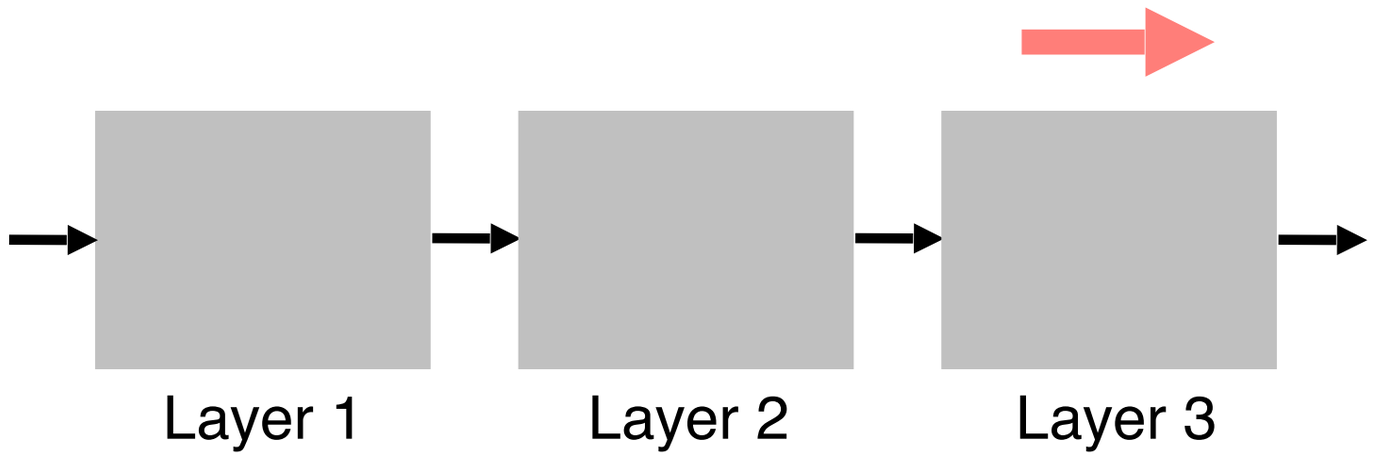
Step 1: Compute loss on mini-batch (Forward Pass)

Step 2: Compute gradients with respect to weights (Backward Pass)

Step 3: Use gradients to update weights (Gradient Descent)

NEURAL NETWORK TRAINING

Forward Propagation, Back Propagation, Gradient Descent



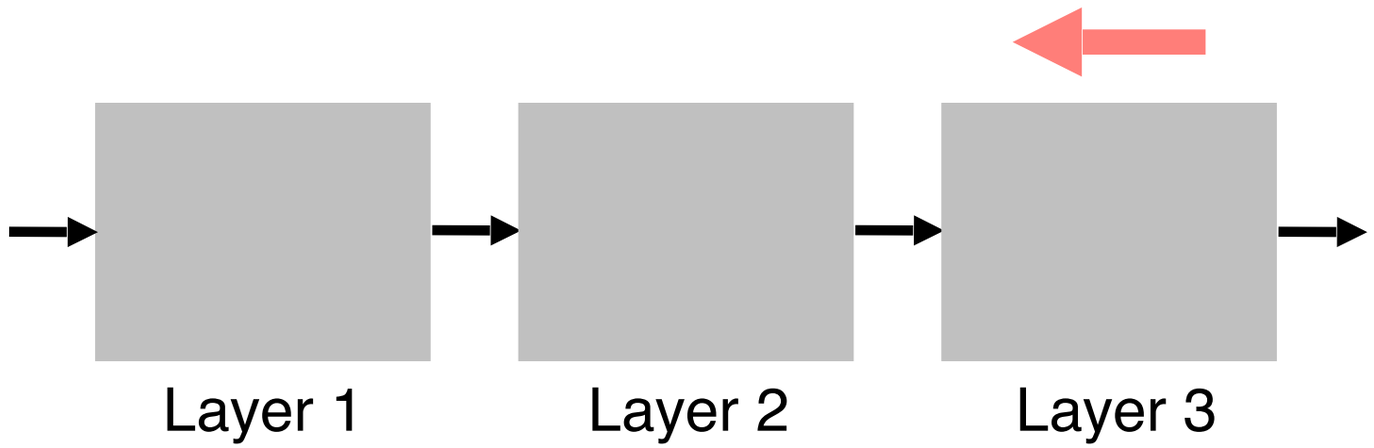
Step 1: Compute loss on mini-batch (Forward Pass)

Step 2: Compute gradients with respect to weights (Backward Pass)

Step 3: Use gradients to update weights (Gradient Descent)

NEURAL NETWORK TRAINING

Forward Propagation, Back Propagation, Gradient Descent



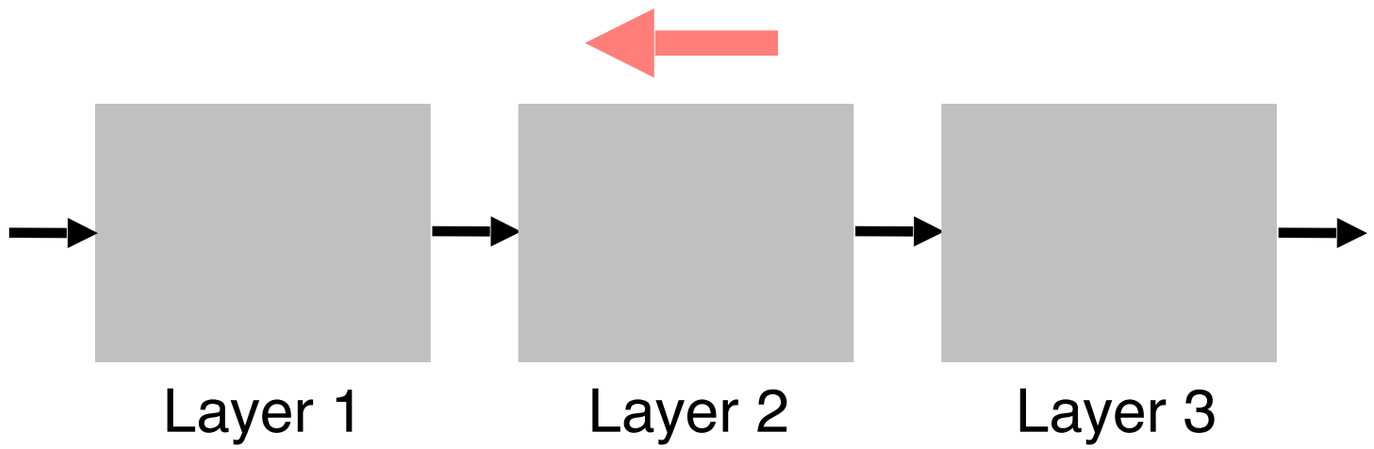
Step 1: Compute loss on mini-batch (Forward Pass)

Step 2: Compute gradients with respect to weights (Backward Pass)

Step 3: Use gradients to update weights (Gradient Descent)

NEURAL NETWORK TRAINING

Forward Propagation, Back Propagation, Gradient Descent



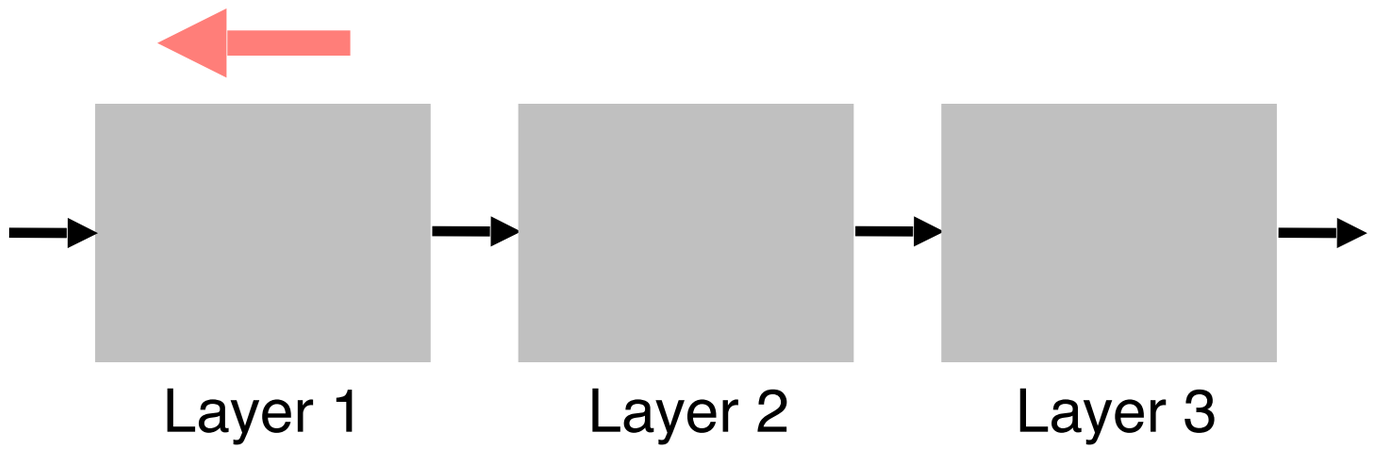
Step 1: Compute loss on mini-batch (Forward Pass)

Step 2: Compute gradients with respect to weights (Backward Pass)

Step 3: Use gradients to update weights (Gradient Descent)

NEURAL NETWORK TRAINING

Forward Propagation, Back Propagation, Gradient Descent



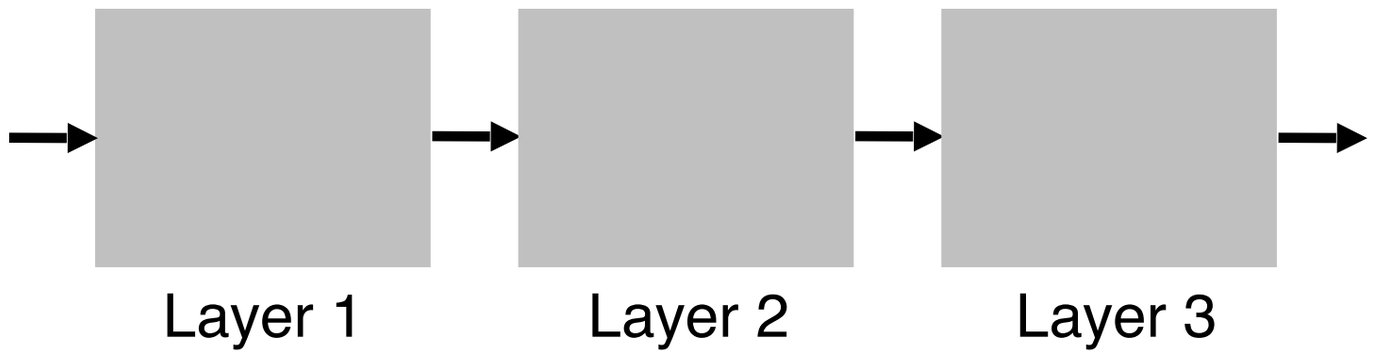
Step 1: Compute loss on mini-batch (Forward Pass)

Step 2: Compute gradients with respect to weights (Backward Pass)

Step 3: Use gradients to update weights (Gradient Descent)

NEURAL NETWORK TRAINING

Forward Propagation, Back Propagation, Gradient Descent



Step 1: Compute loss on mini-batch (Forward Pass)

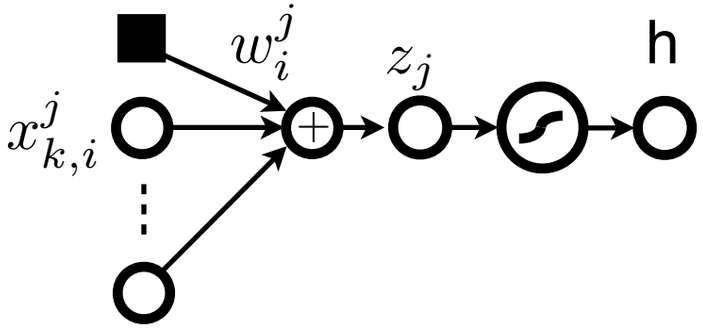
Step 2: Compute gradients with respect to weights (Backward Pass)

Step 3: Use gradients to update weights (Gradient Descent)

$$\mathbf{w}^{(t+1)} = \mathbf{w}^{(t)} - \eta \frac{\partial E(\mathbf{w})}{\partial \mathbf{w}}\bigg|_{\mathbf{w}^{(t)}}$$

MLP TRAINING

Optimizing Weights for a Multi-layer Neural Network



$$\mathbf{w}^{(t+1)} = \mathbf{w}^{(t)} - \eta \frac{\partial E(\mathbf{w})}{\partial \mathbf{w}}\bigg|_{\mathbf{w}^{(t)}}$$

Gradient Descent

$$\frac{\partial E(\mathbf{w})}{\partial \mathbf{w}} = \sum_{k=1}^{K} \frac{\partial E_k(\mathbf{w})}{\partial \mathbf{w}}$$

Sum over all training examples in mini-batch

$$\frac{\partial E_k(\mathbf{w})}{\partial w_i^j} = \frac{\partial}{\partial w_i^j}\Big(h(\mathbf{x}_k;\mathbf{w}) - y_k\Big)^2$$

Find expression for gradient w.r.t. one weight and one training sample

$$= 2\Big(h(\mathbf{x}_k;\mathbf{w}) - y_k\Big)\frac{\partial h(\mathbf{x}_k;\mathbf{w})}{\partial w_i^j}$$

$$= 2\Big(h(\mathbf{x}_k;\mathbf{w}) - y_k\Big)\frac{\partial h(\mathbf{x}_k;\mathbf{w})}{\partial z_j}\frac{\partial z_j}{\partial w_i^j}$$

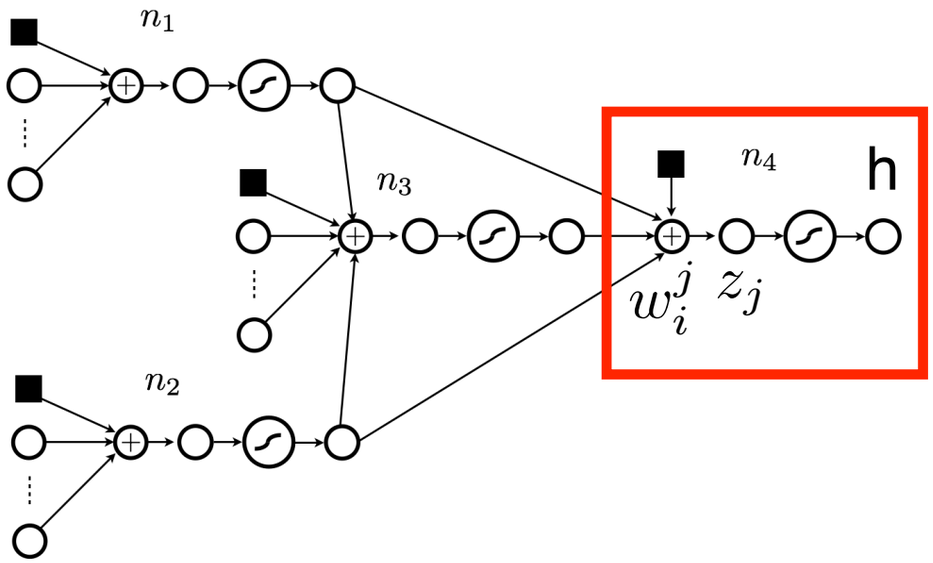
Chain Rule because function of function

$$= 2\Big(h(\mathbf{x}_k;\mathbf{w}) - y_k\Big)\frac{\partial h(\mathbf{x}_k;\mathbf{w})}{\partial z_j}x_{k,i}^j$$

$$z_{kj} = \sum_{i=0}^{d} w_i^j x_{ki}^j$$

MLP TRAINING

Case 1: An Output Node



$$\frac{E_k(\mathbf{w})}{\partial w_i^j} = 2\Big(h(\mathbf{x}_k;\mathbf{w}) - y_i\Big)\frac{\partial h(\mathbf{x}_k;\mathbf{w})}{\partial z_j}x_{k,i}^j$$

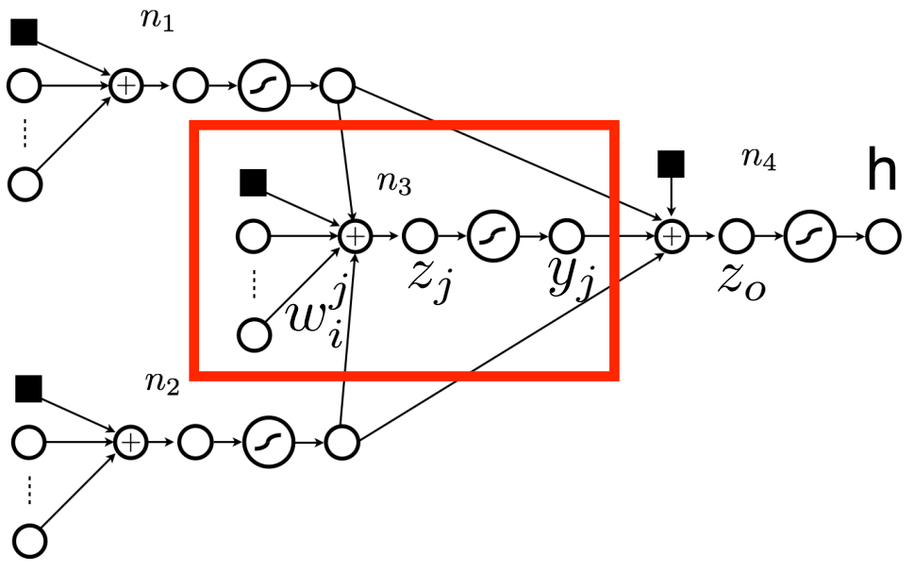
$$\frac{\partial h(\mathbf{x}_k;\mathbf{w})}{\partial z_j} = \frac{\partial \sigma(z_j)}{\partial z_j} = \sigma(z_j)(1-\sigma(z_j))$$

We'll save this term for future use!

$$\frac{\partial E_k(\mathbf{w})}{\partial w_i^j} = 2\Big(h(\mathbf{x}_k;\mathbf{w}) - y_k\Big)\sigma(z_j)(1-\sigma(z_j))x_{k,i}^j$$

MLP TRAINING

Case 2a: A node connected to one child



$$\frac{E_k(\mathbf{w})}{\partial w_i^j} = 2\Big(h(\mathbf{x}_k;\mathbf{w}) - y_i\Big)\frac{\partial h(\mathbf{x}_k;\mathbf{w})}{\partial z_j}x_{k,i}^j$$

A change in z_j can affect the child node

$$\frac{\partial h(\mathbf{x}_k)}{\partial z_j} = \frac{\partial h(\mathbf{x}_k)}{\partial y_j}\frac{\partial y_j}{\partial z_j} = \frac{\partial h(\mathbf{x}_k)}{\partial y_j}\frac{d\sigma(z_j)}{dz_j}$$

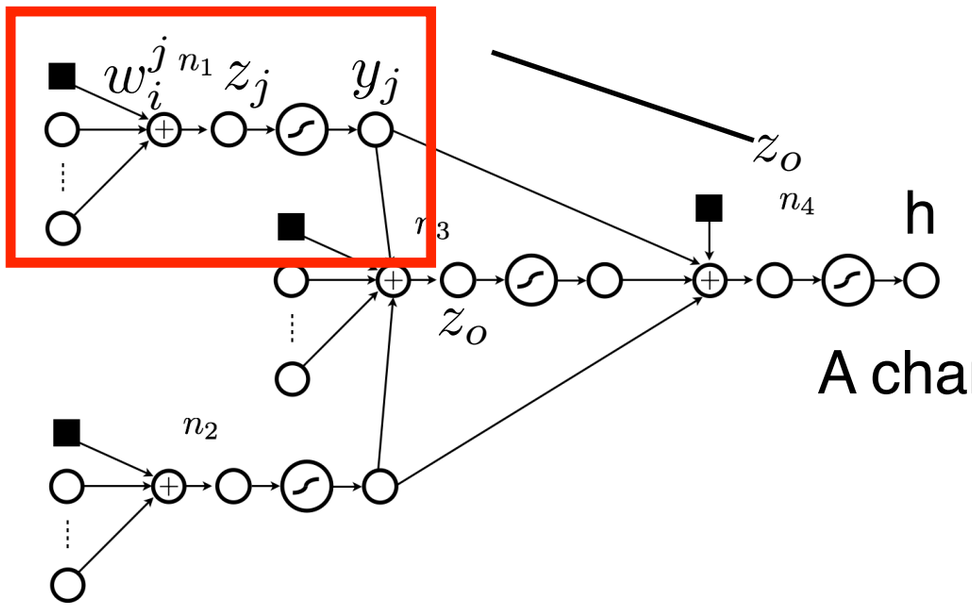
$$= \frac{\partial h(\mathbf{x}_k)}{\partial z_o}\frac{\partial z_o}{\partial y_j}\frac{d\sigma(z_j)}{dz_j}$$

$$= \boxed{\frac{\partial h(\mathbf{x}_k)}{\partial z_o}} w_i^j \sigma(z_j)(1-\sigma(z_j))$$

To complete the term, we can now use the term we calculated at the output node!

MLP TRAINING

Case 2b: A node connected to multiple children



$$\frac{E_k(\mathbf{w})}{\partial w_i^j} = 2\Big(h(\mathbf{x}_k;\mathbf{w}) - y_i\Big)\frac{\partial h(\mathbf{x}_k;\mathbf{w})}{\partial z_j}x_{k,i}^j$$

A change in z_j can affect all nodes connected to the output

$$\frac{\partial h(\mathbf{x}_k)}{\partial z_j} = \frac{\partial h(\mathbf{x}_k)}{\partial y_j}\frac{\partial y_j}{\partial z_j} = \frac{\partial h(\mathbf{x}_k)}{\partial y_j}\frac{d\sigma(z_j)}{dz_j}$$

$$= \sum_{o\in\{\text{Children}(n)\}} \frac{\partial h(\mathbf{x}_k)}{\partial z_o}\frac{\partial z_o}{\partial y_j}\frac{d\sigma(z_j)}{dz_j}$$

$$= \sum_{o\in\{\text{Children}(n)\}} \frac{\partial h(\mathbf{x}_k)}{\partial z_o} w_i^j \sigma(z_j)(1-\sigma(z_j))$$

$$\frac{\partial h(\mathbf{x}_k)}{\partial z_j} = \sum_{o\in\{\text{Children}(n)\}} \frac{\partial h(\mathbf{x}_k)}{\partial z_o} w_i^j \sigma(z_j)(1-\sigma(z_j))$$

OPTIMIZATION DESIGN CONSIDERATION

Design Considerations

How often should we update weights?

- Option 1: Online Update (Stochastic Gradient Descent)
- Option 2: Full Batch Update
- Option 3: Mini-batch Update

How much should we update weights?

- Option 1: Fixed Learning Rate
- Option 2: Adaptive Learning Rate
- Option 3: Learning Schedule

How can we encourage generalization?

- Keep the weights small, weight sharing, convolutions for images, early stopping, model averaging, dropout

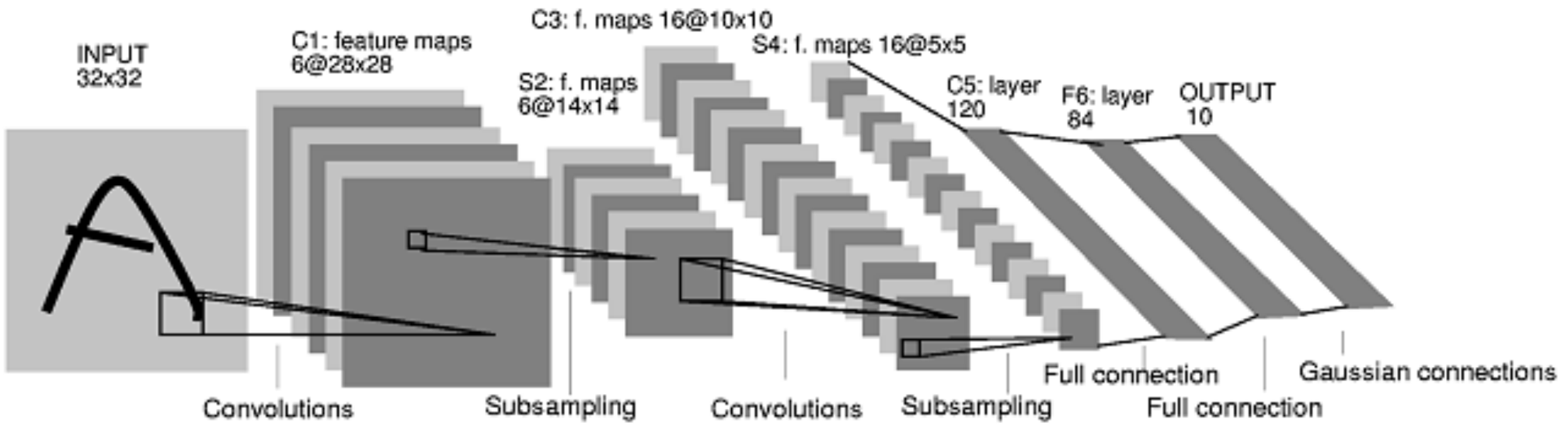
OPTIMIZATION

Design Consideration: How should we initialize weights?

Option 1: Zero Initialization:
Not recommended. No difference in gradients.

Option 2: Small Random Weights:
“Symmetry breaking” initialization.
Sample from a zero mean Gaussian for the weights.
Zeros or small constants for bias term.

Option 3: Pre-trained Weights
e.g., Use VGG-Net

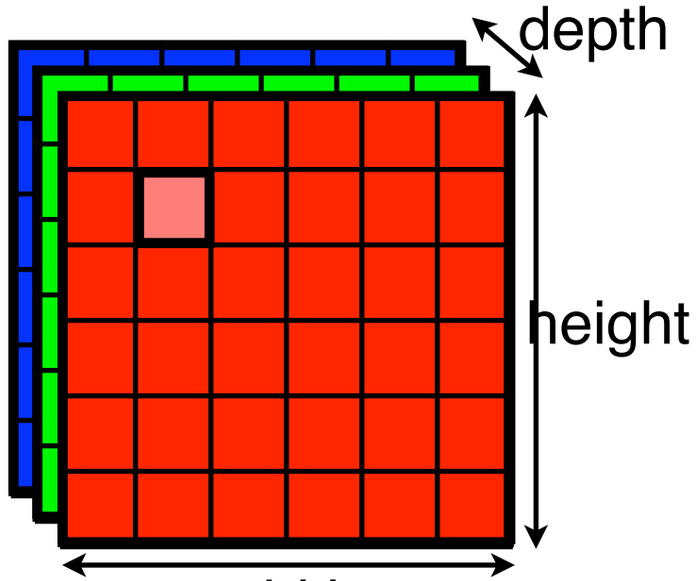


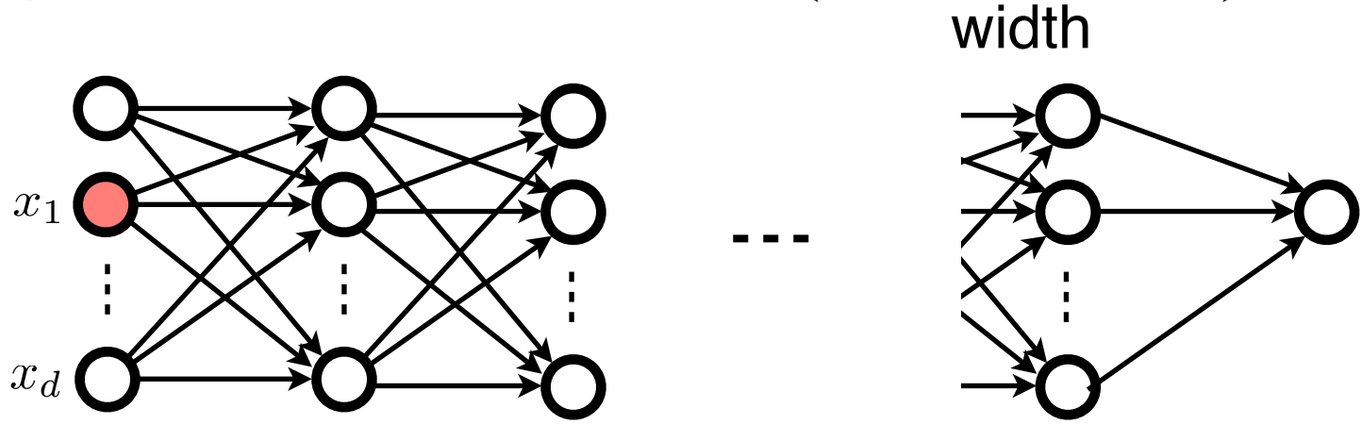
CONVOLUTIONAL NEURAL NETWORKS

WHY CONVNETS?

How many weights for MLPs for images?

Image Size: 1920 x 1080 x 3





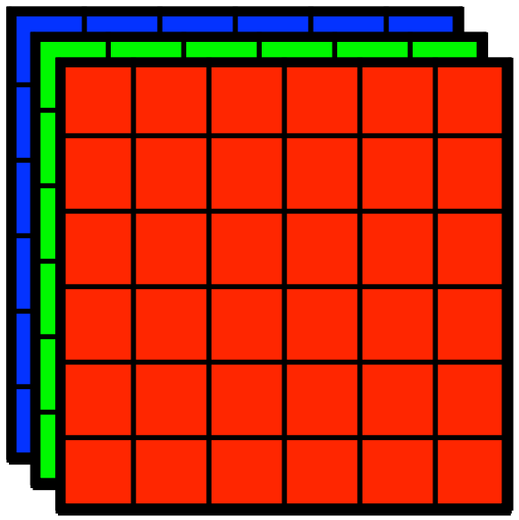
nodes/layer: $d = 1920 \times 1080 \times 3 = 6,220,800$

weights/layer: $d^2 = (1920 \times 1080 \times 3)^2 = 38.69 \times 10^{12}$

total number of weights: $Nd^2 = 10 \times (1920 \times 1080 \times 3)^2 = 38.69 \times 10^{13}$

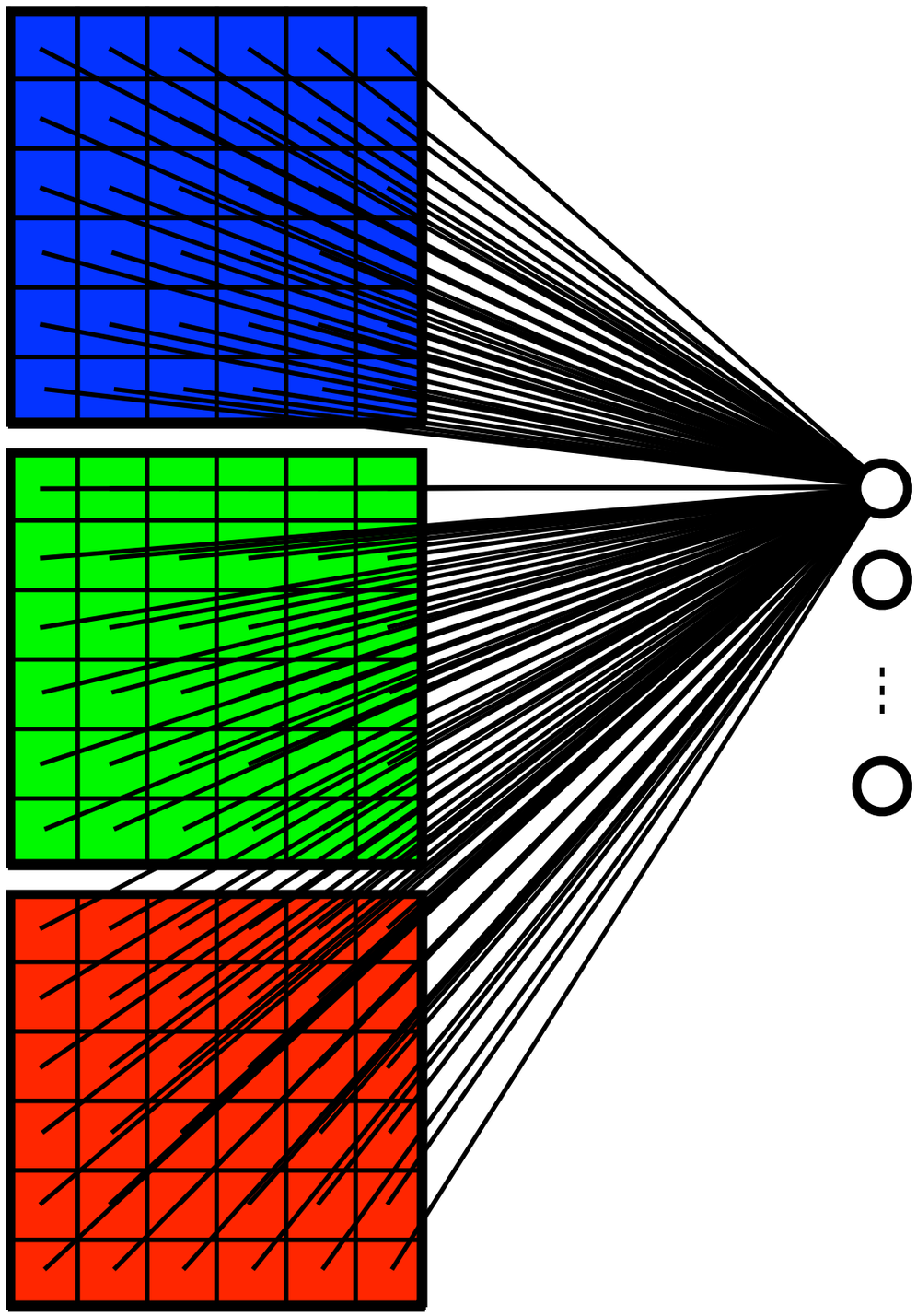
Dimensionality of the Optimization Space!

WHY CONVNETS?



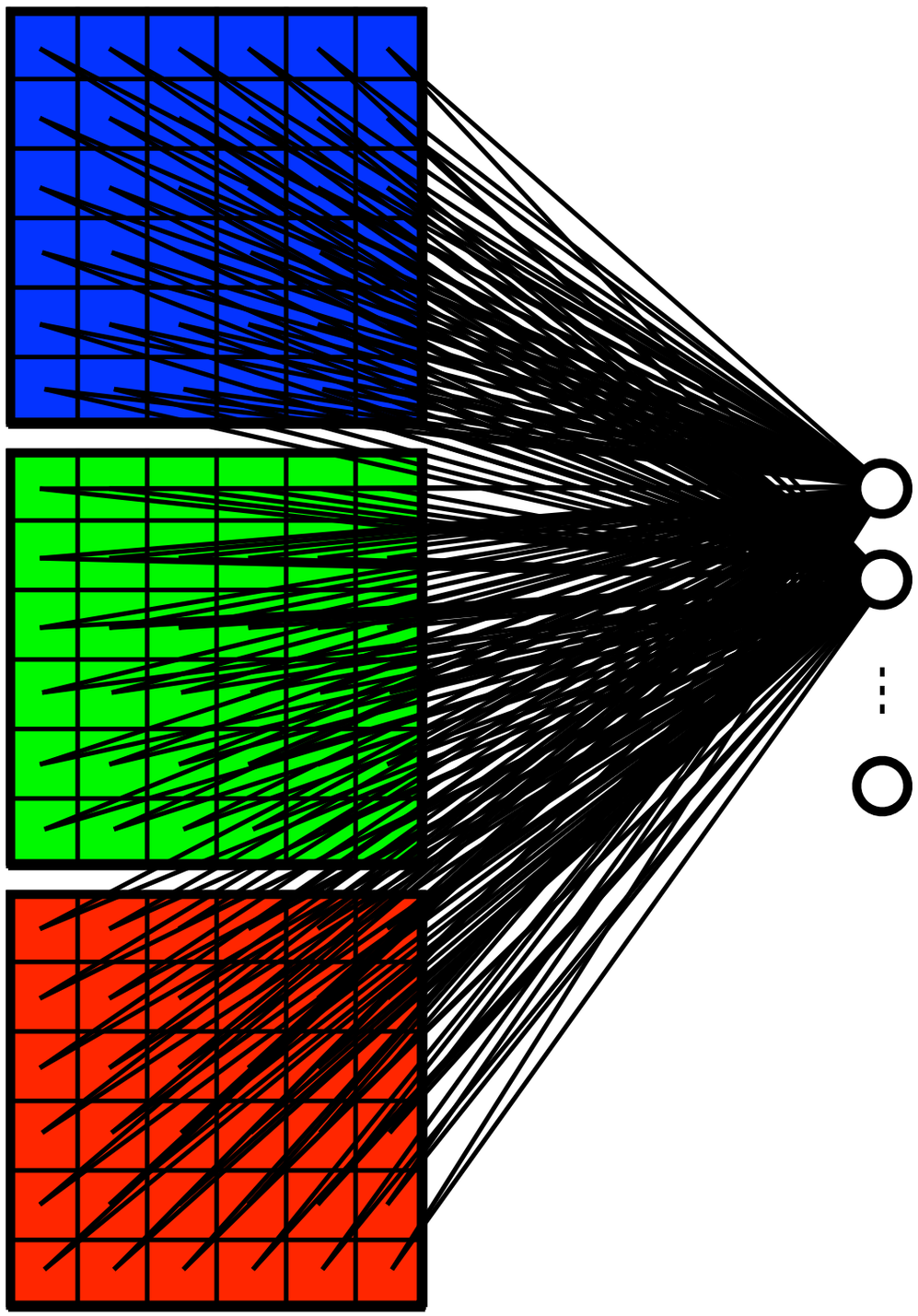
WHY CONVNETS?

Full Connection



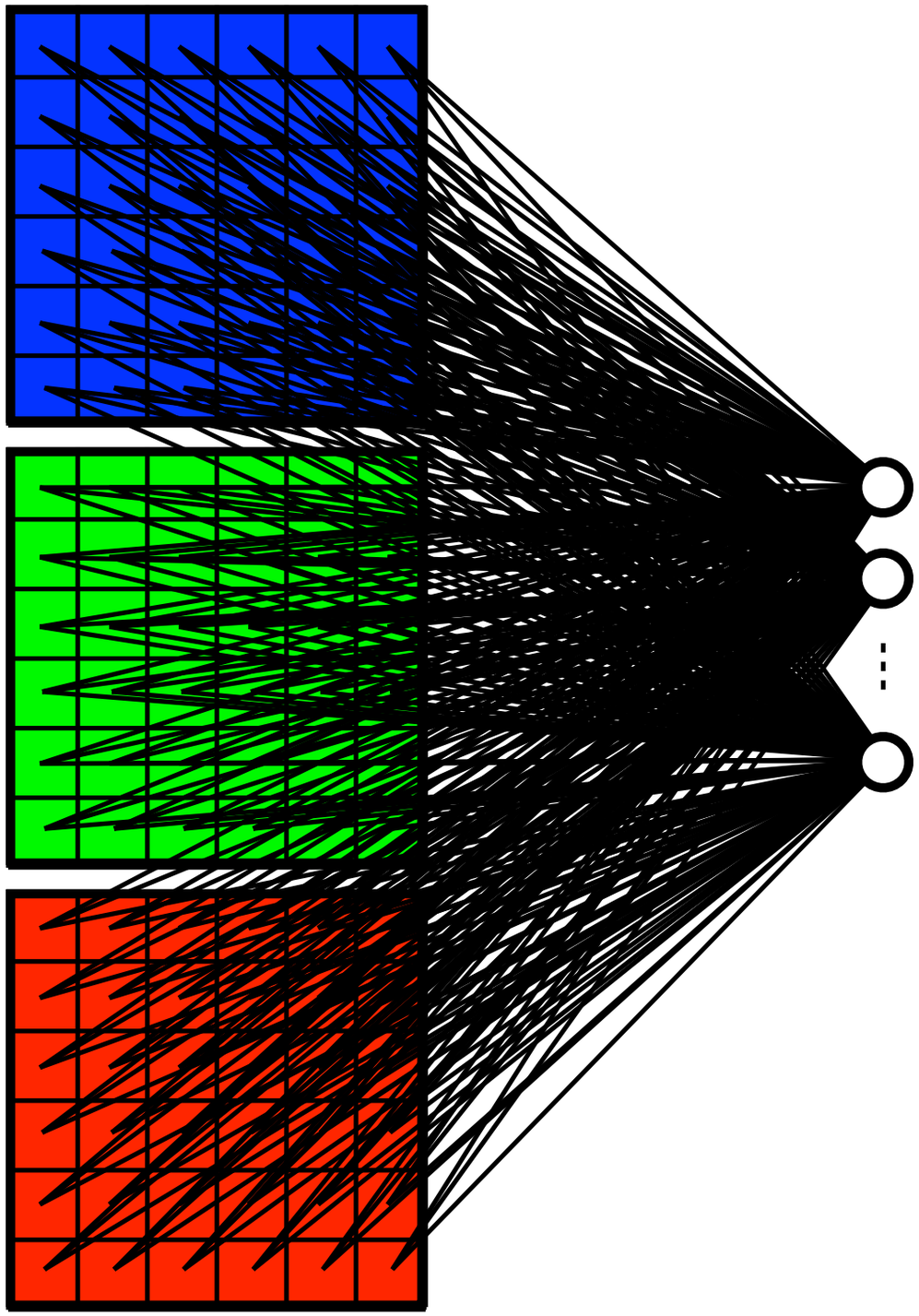
WHY CONVNETS?

Full Connection



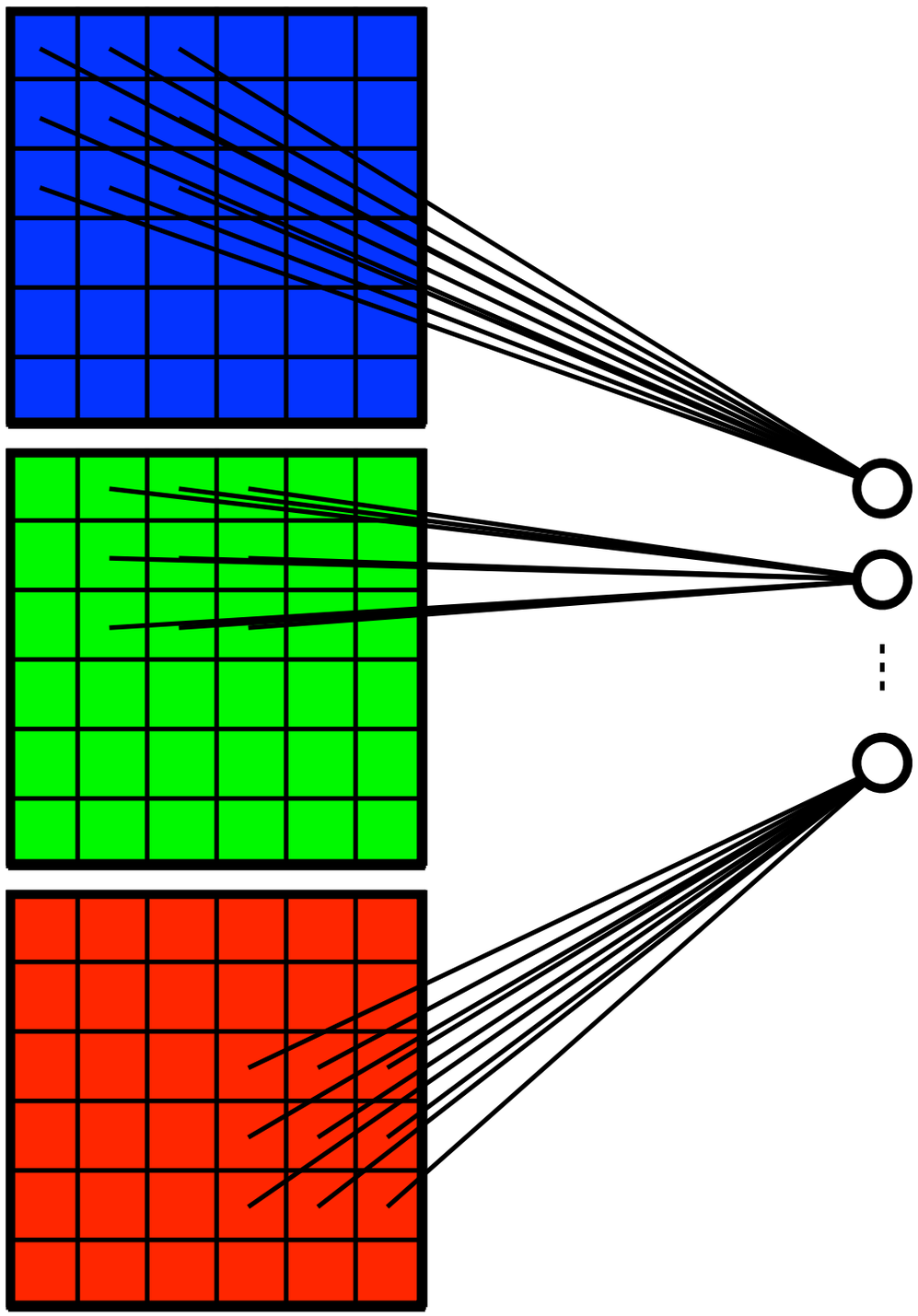
WHY CONVNETS?

Full Connection



CONVOLUTIONAL LAYER

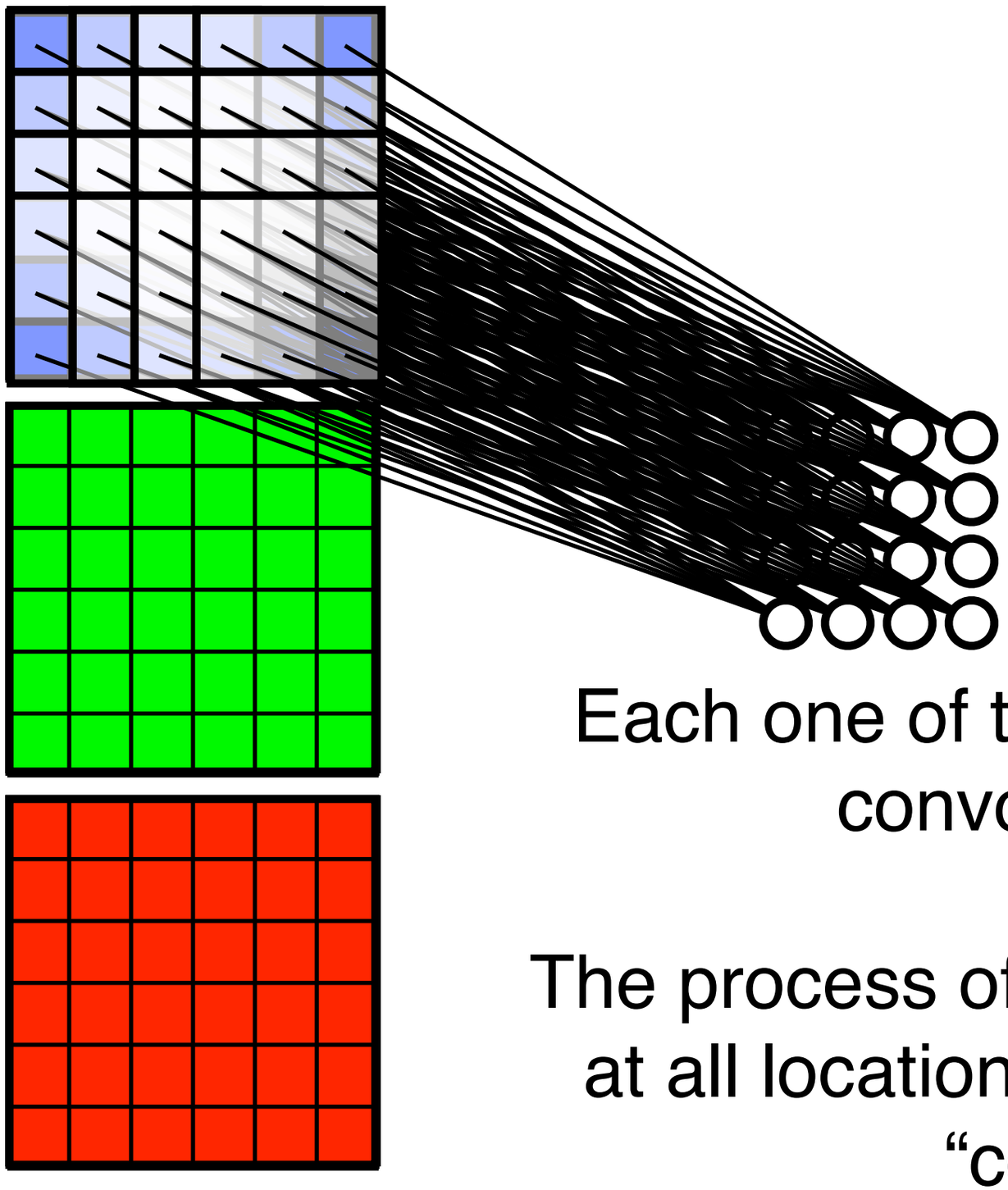
Property I: Local Connectivity



Each Neuron takes information only from a neighborhood of pixels

CONVOLUTIONAL LAYER

Property II: Stationarity



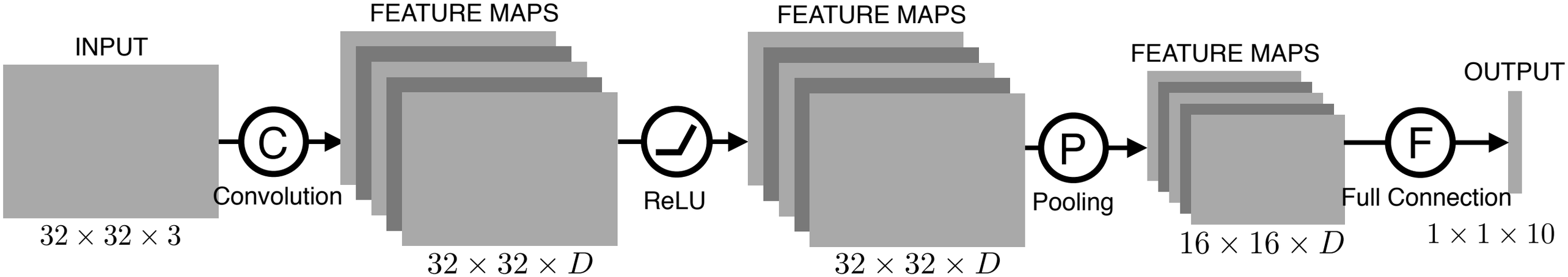
Neurons connecting all neighborhoods have identical weights

Each one of these nodes is called a convolutional “filter”

The process of multiplying the weights at all locations is implemented as a “convolution"

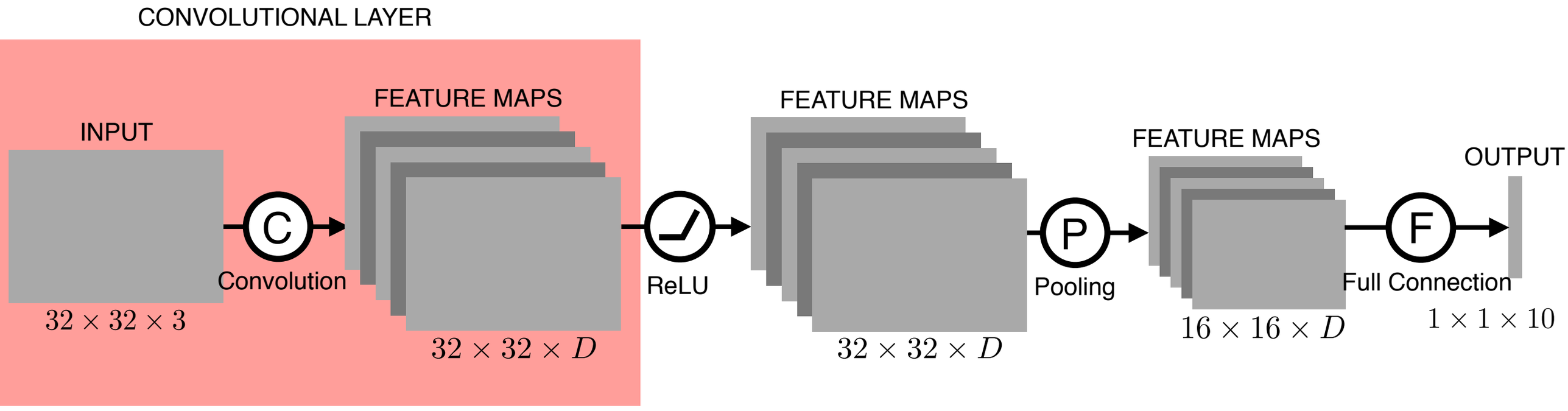
CONVNET ARCHITECTURE

Layers Used to Build Convolutional Networks



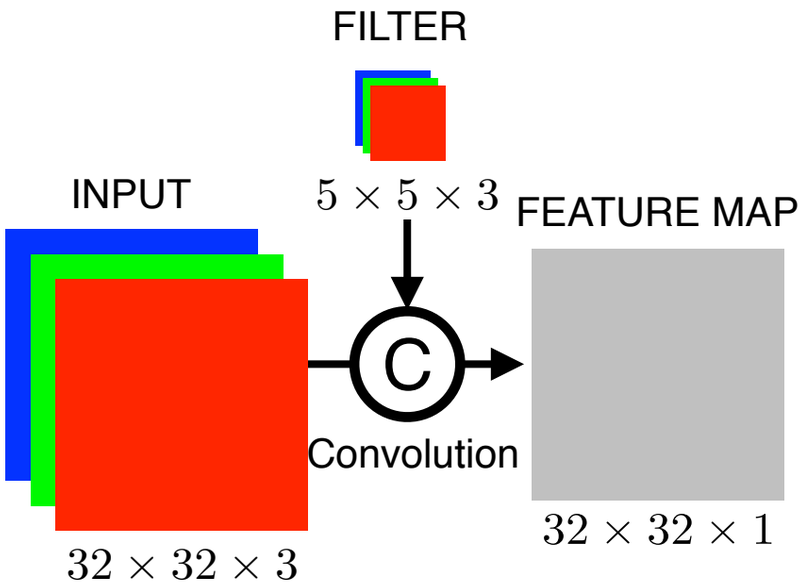
CONVNET ARCHITECTURE

Layers Used to Build Convolutional Networks



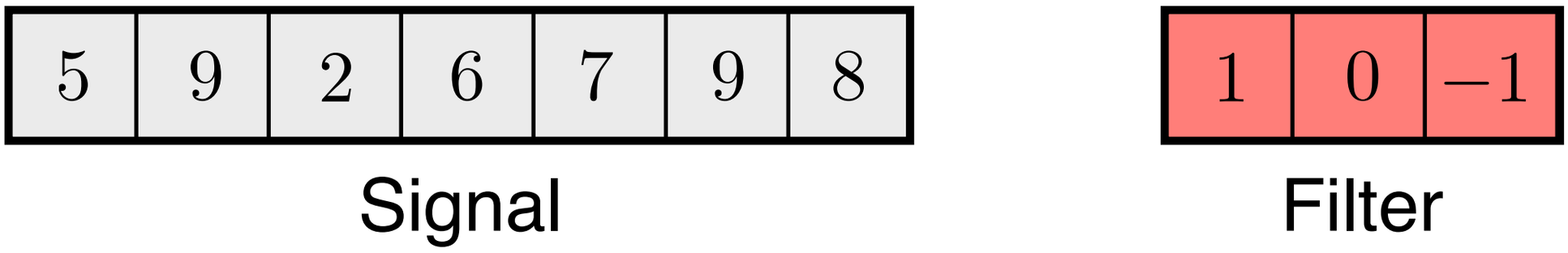
CONVNET ARCHITECTURE

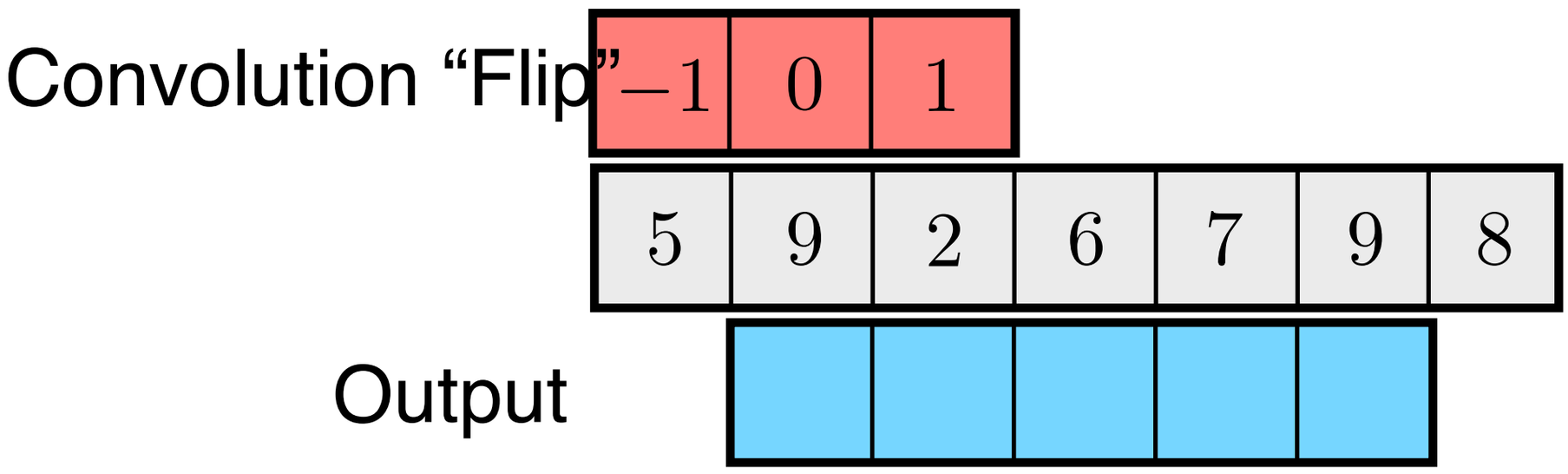
Convolutional Layer



REVIEW: CONVOLUTION

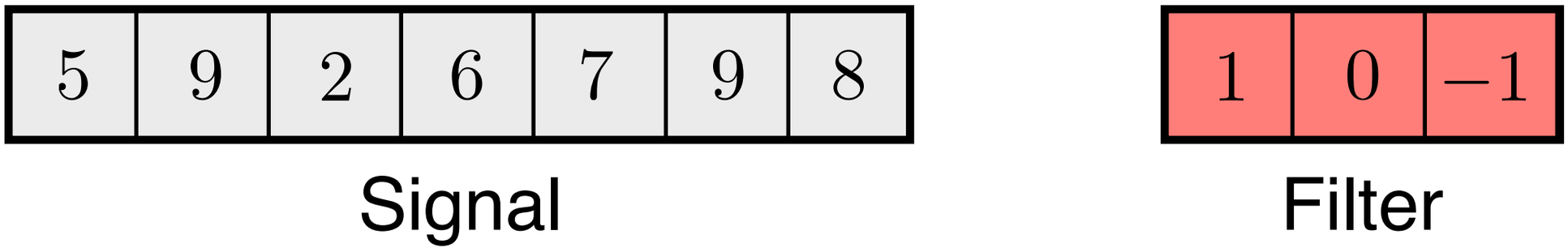
1D Convolution

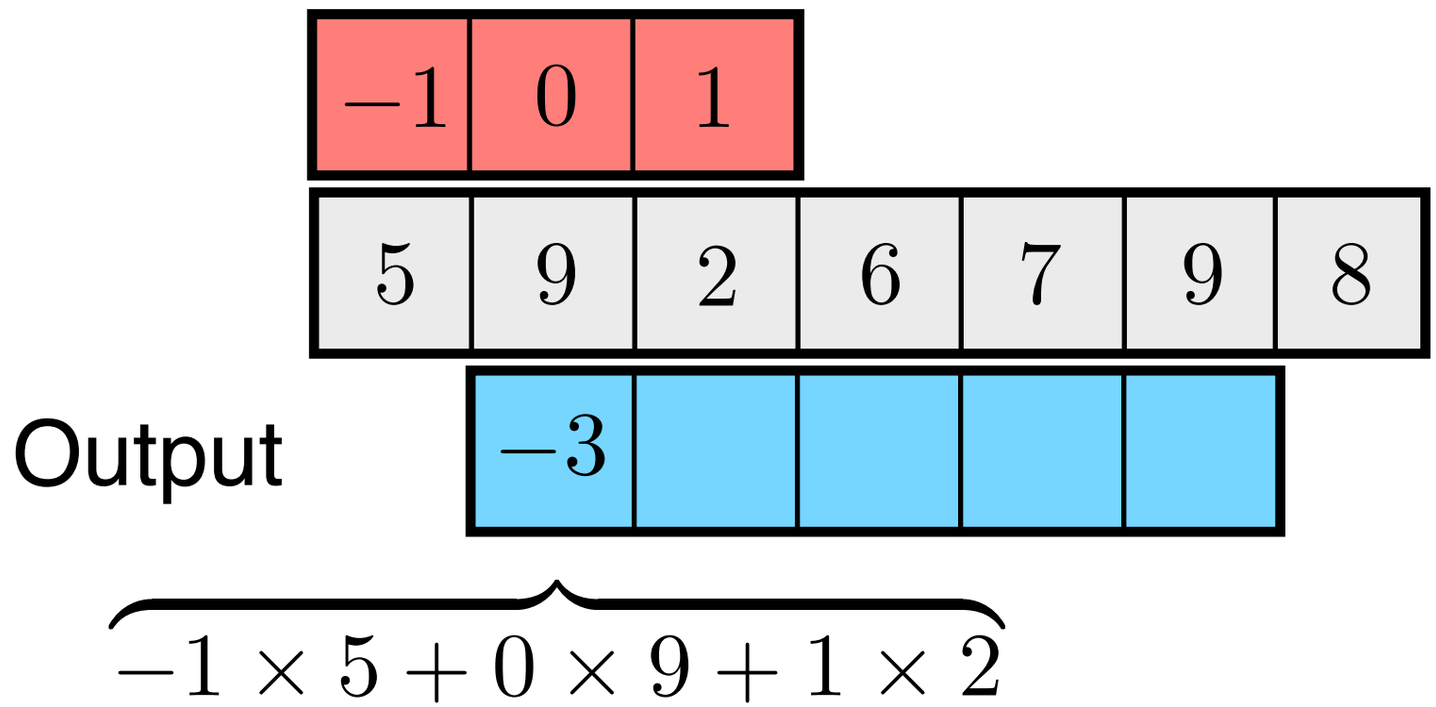




REVIEW: CONVOLUTION

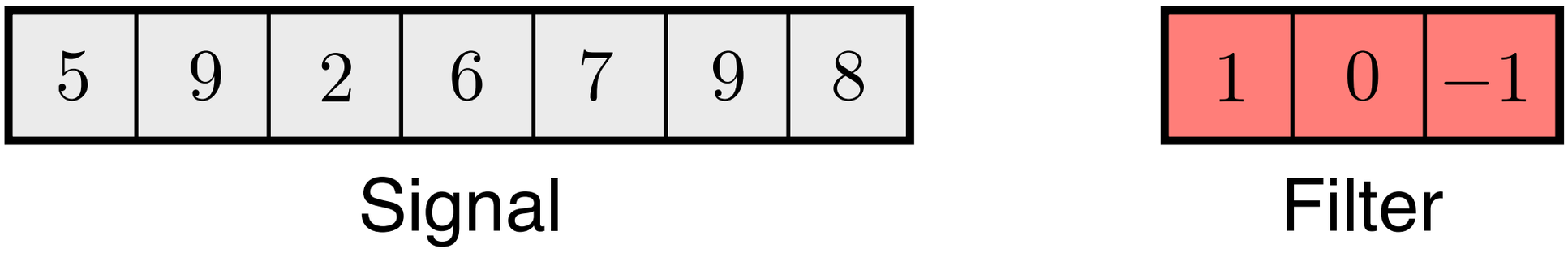
1D Convolution

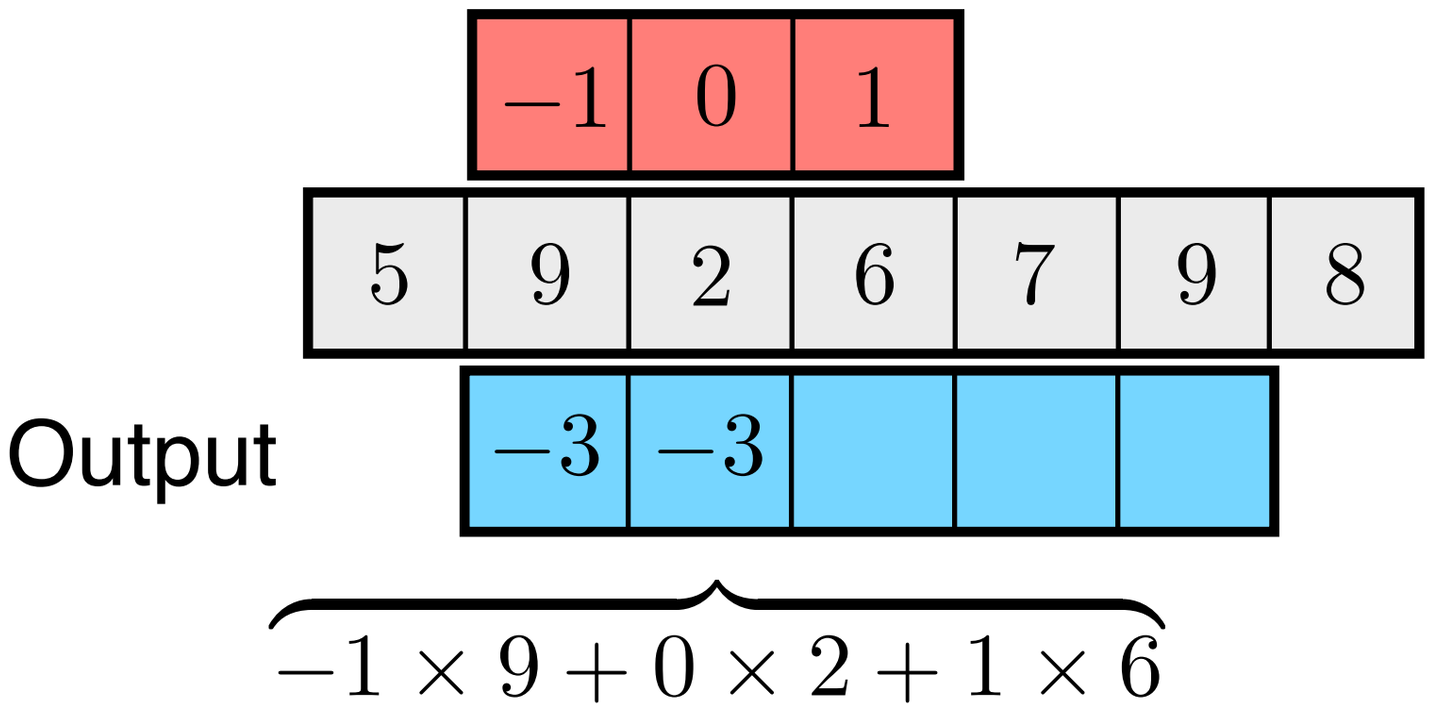




REVIEW: CONVOLUTION

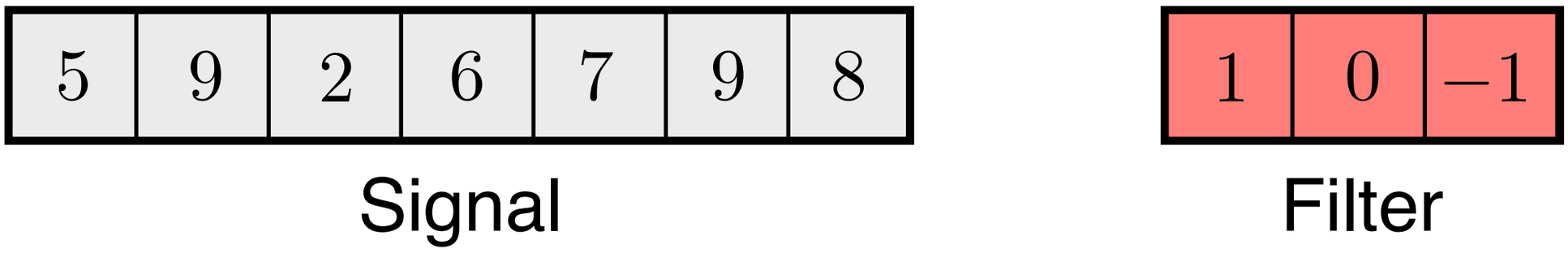
1D Convolution

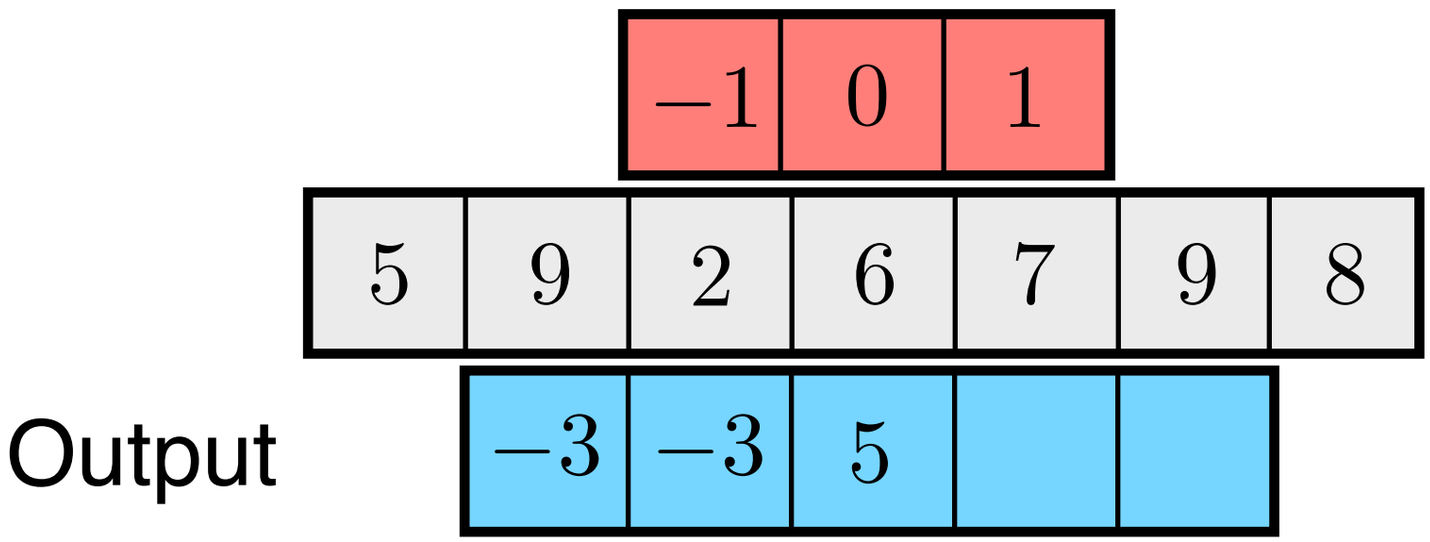




REVIEW: CONVOLUTION

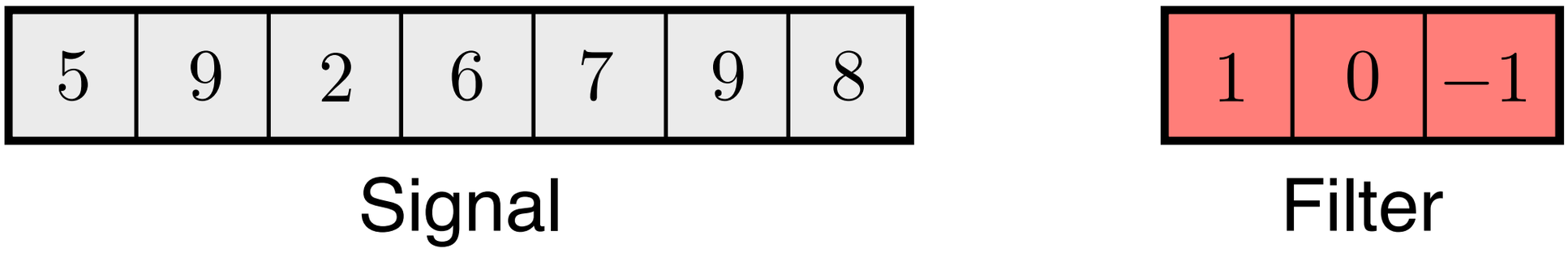
1D Convolution

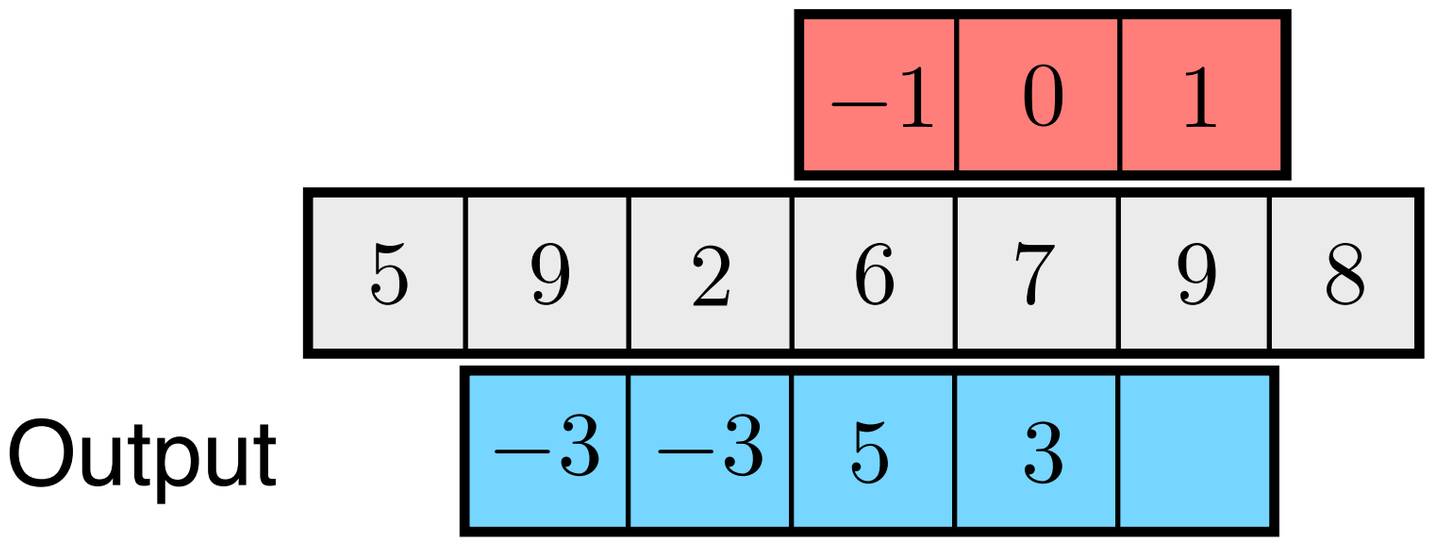




REVIEW: CONVOLUTION

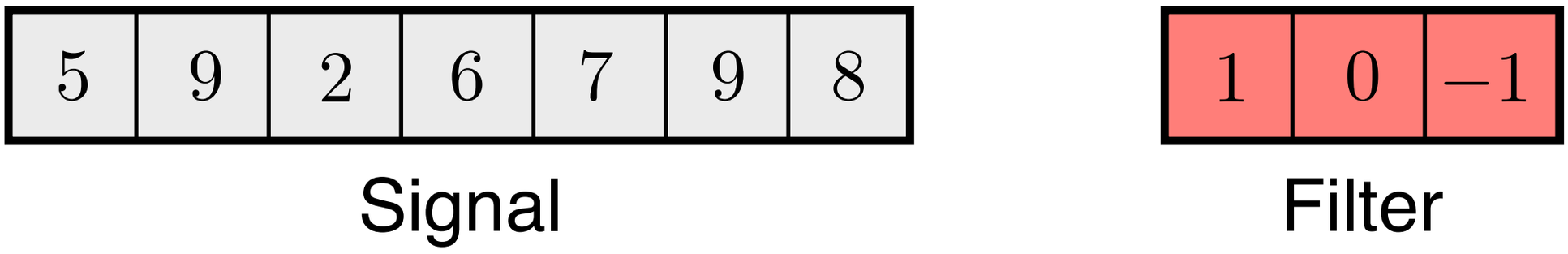
1D Convolution

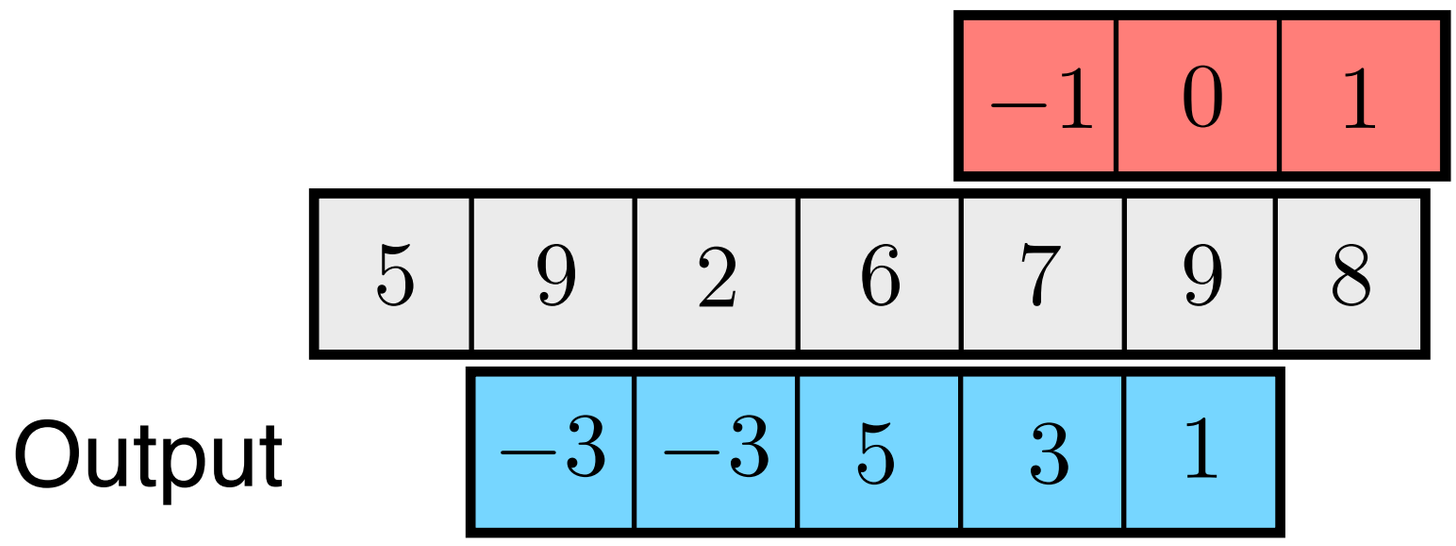




REVIEW: CONVOLUTION

1D Convolution

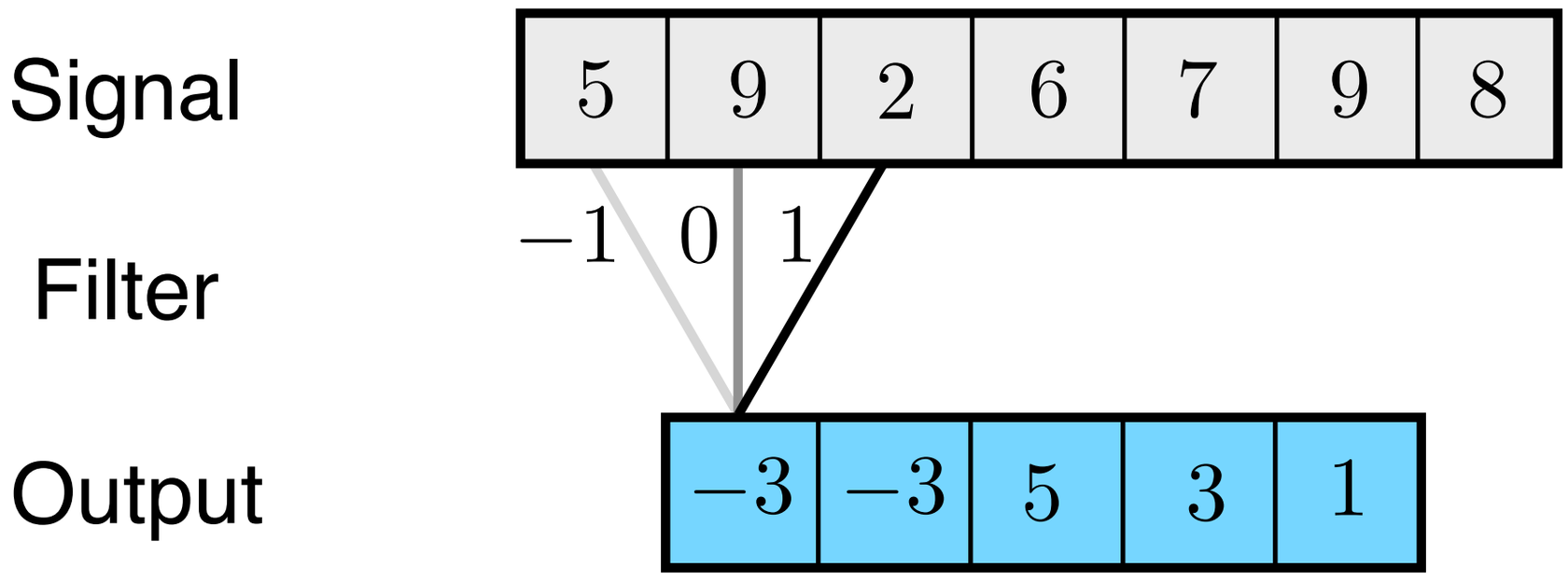




Convolution is a local linear operator

REVIEW: CONVOLUTION

1D Convolution



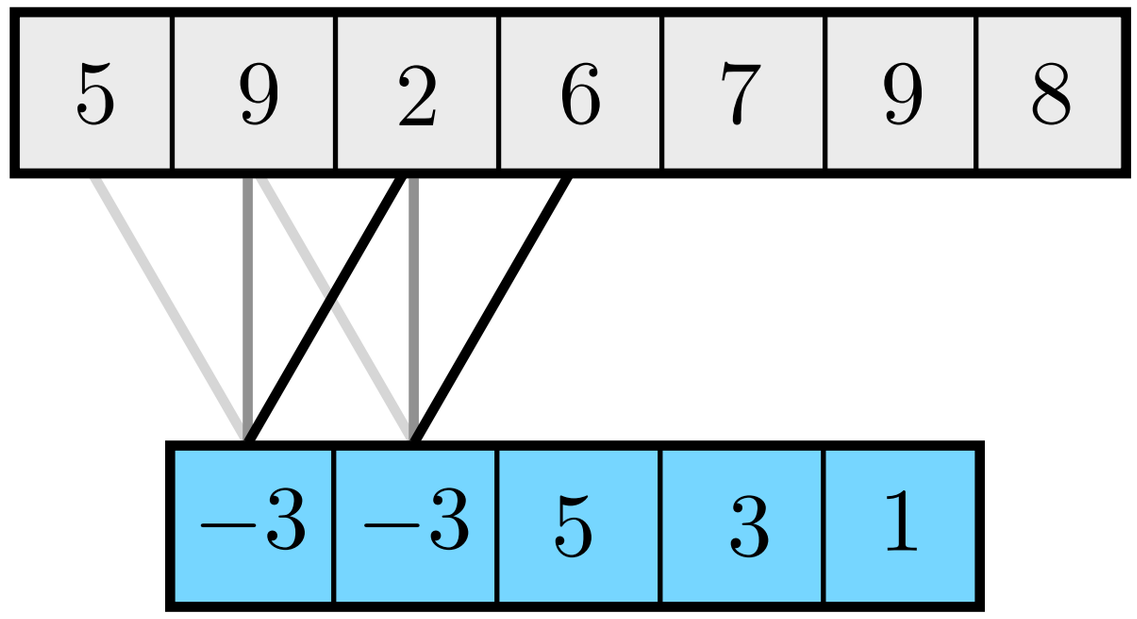
Convolution is a local linear operator

REVIEW: CONVOLUTION

1D Convolution

Signal

Output



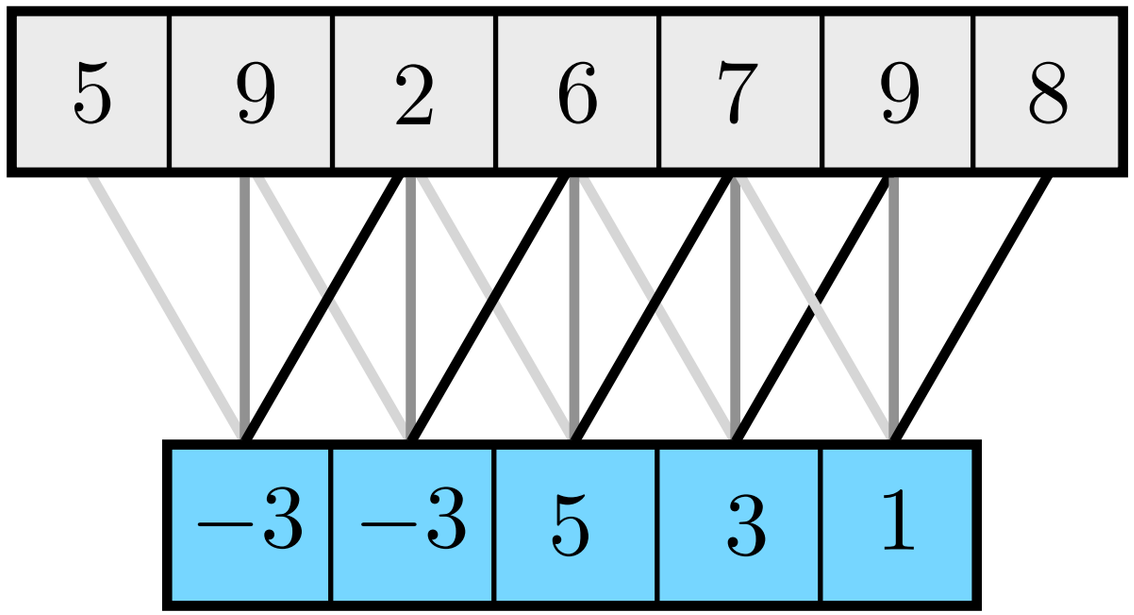
Convolution is a local linear operator

REVIEW: CONVOLUTION

1D Convolution

Signal

Output

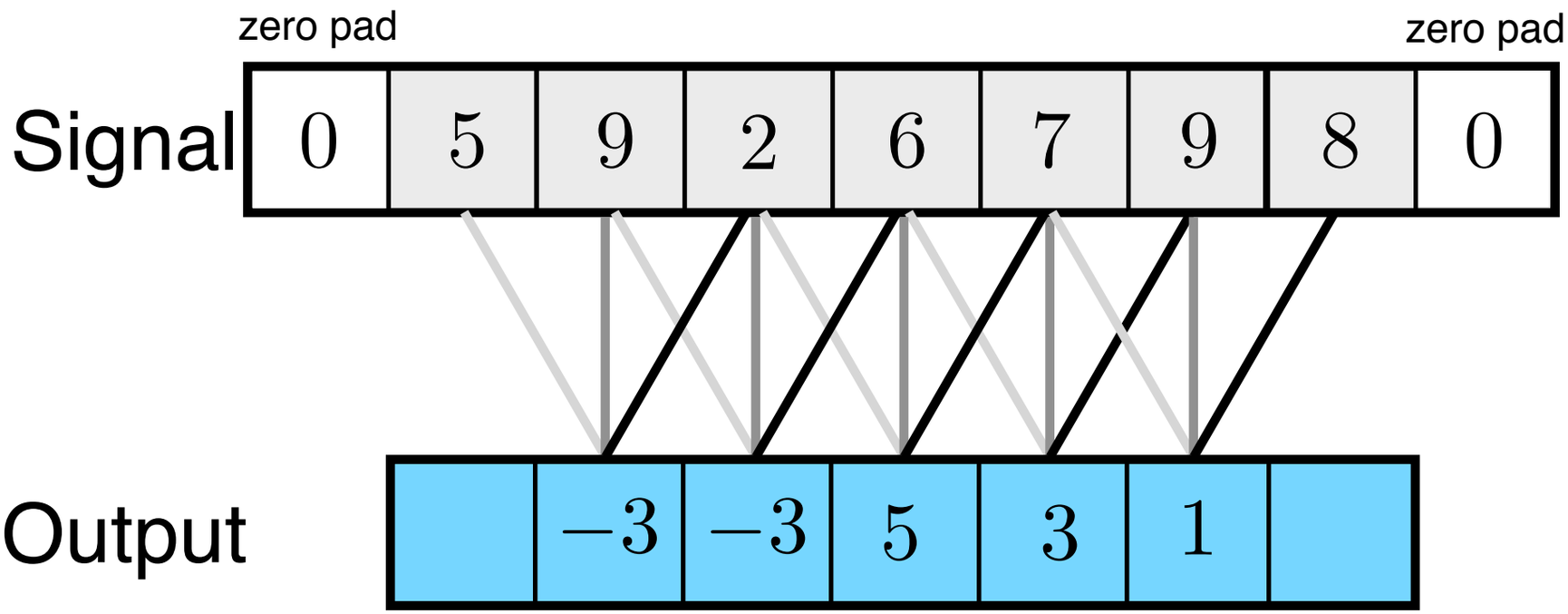


Problem: the edges get “cut off”
How do we deal with this?

Convolution is a local linear operator

REVIEW: CONVOLUTION

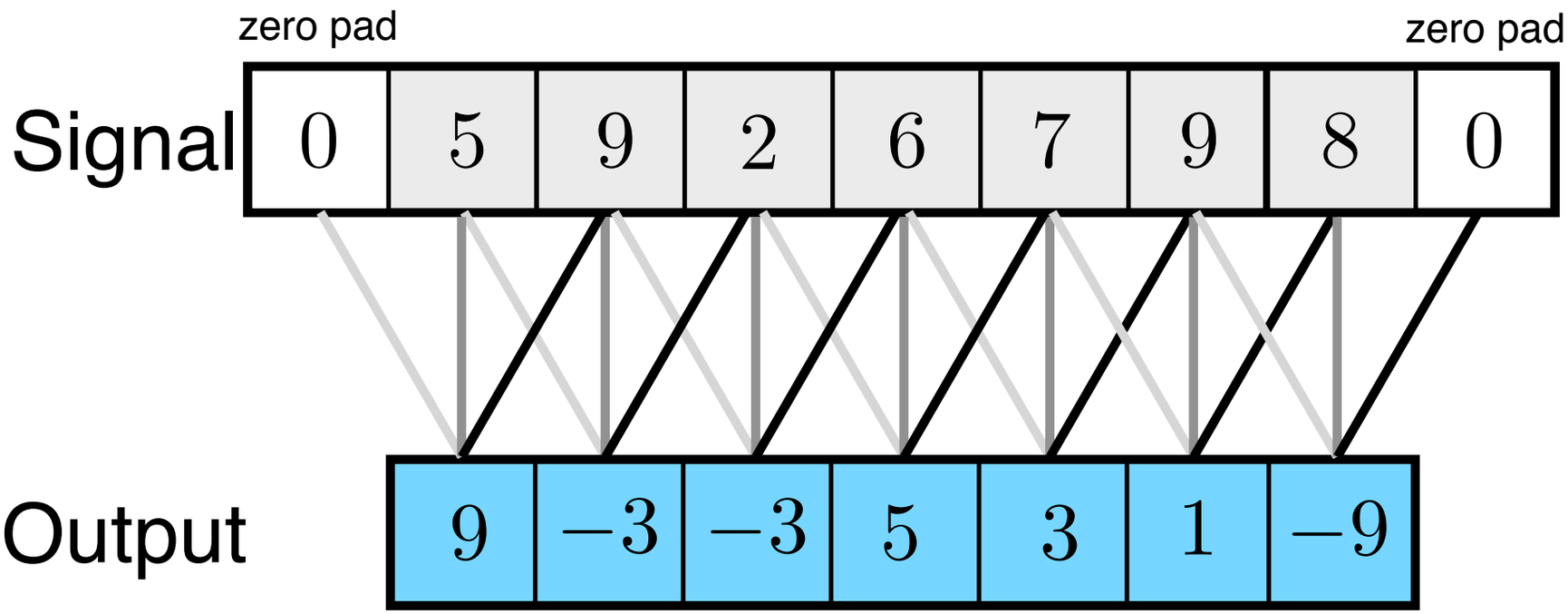
1D Convolution: Zero Padding



Output is the same size as input

REVIEW: CONVOLUTION

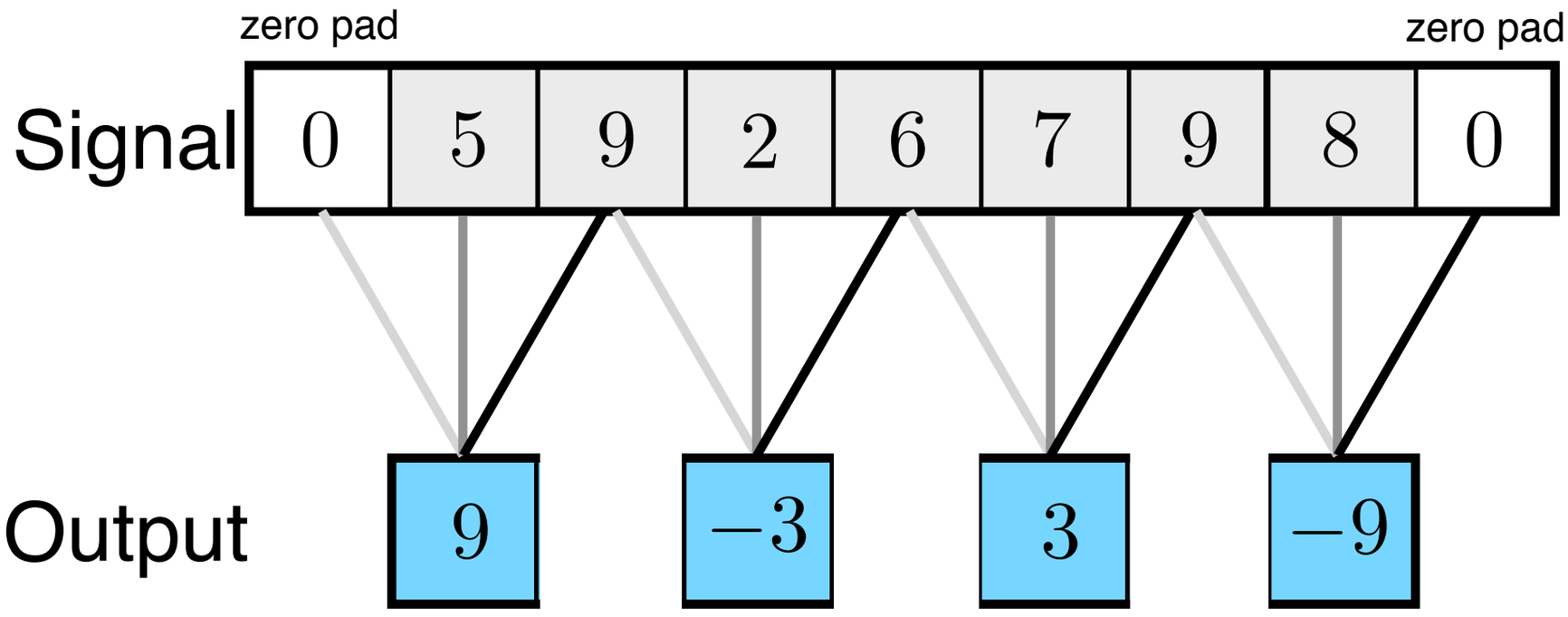
1D Convolution: Zero Padding



Output is the same size as input

REVIEW: CONVOLUTION

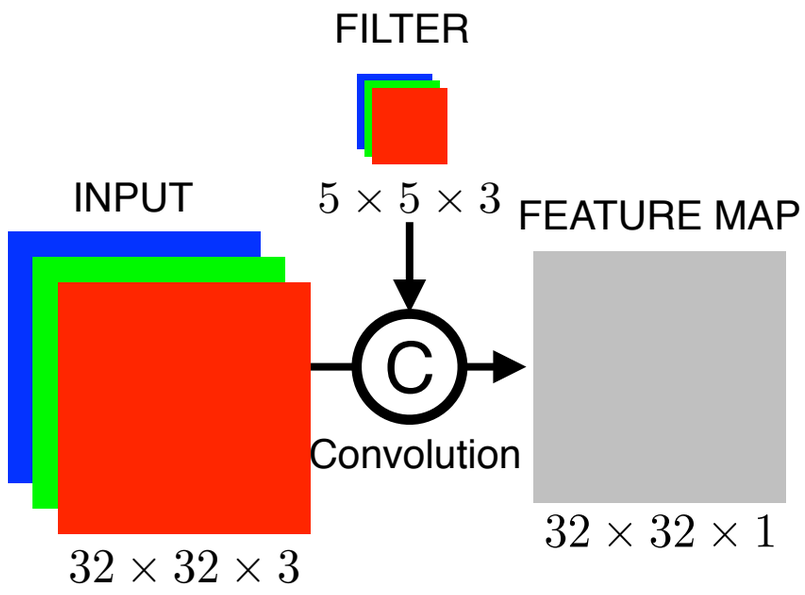
1D Convolution: Stride



Stride: step size across the signal

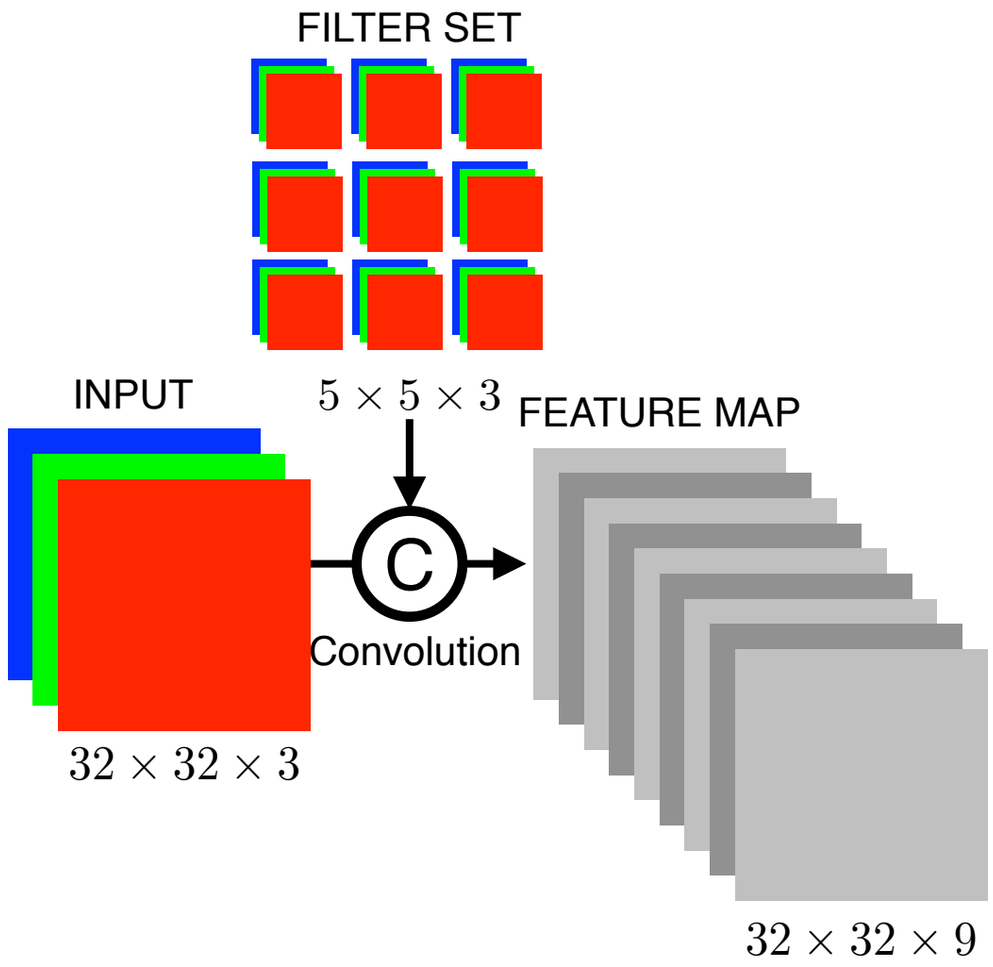
CONVNET ARCHITECTURE

Convolutional Layer



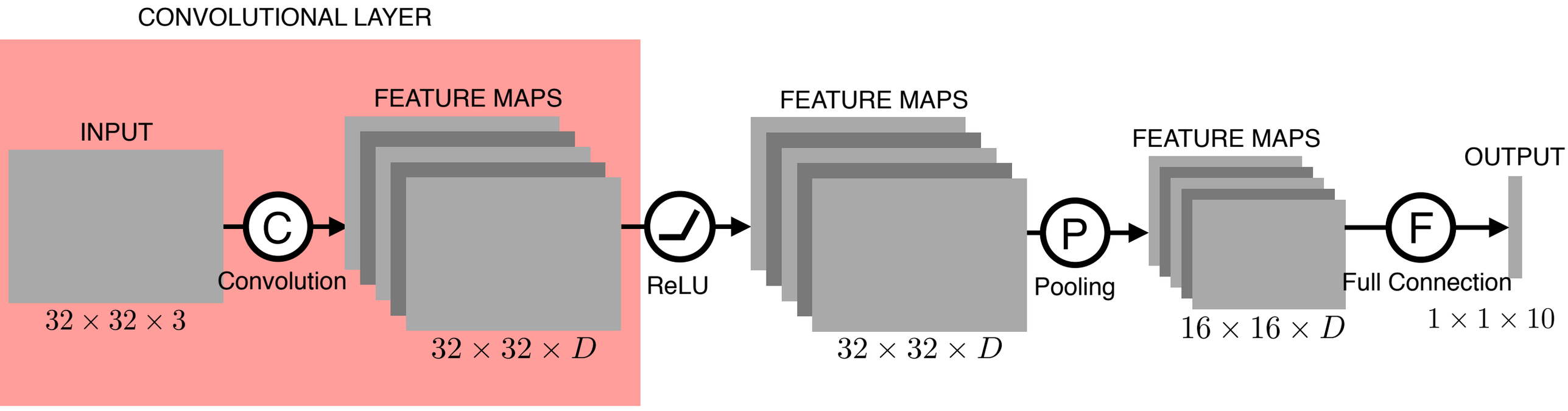
CONVNET ARCHITECTURE

Convolutional Layer



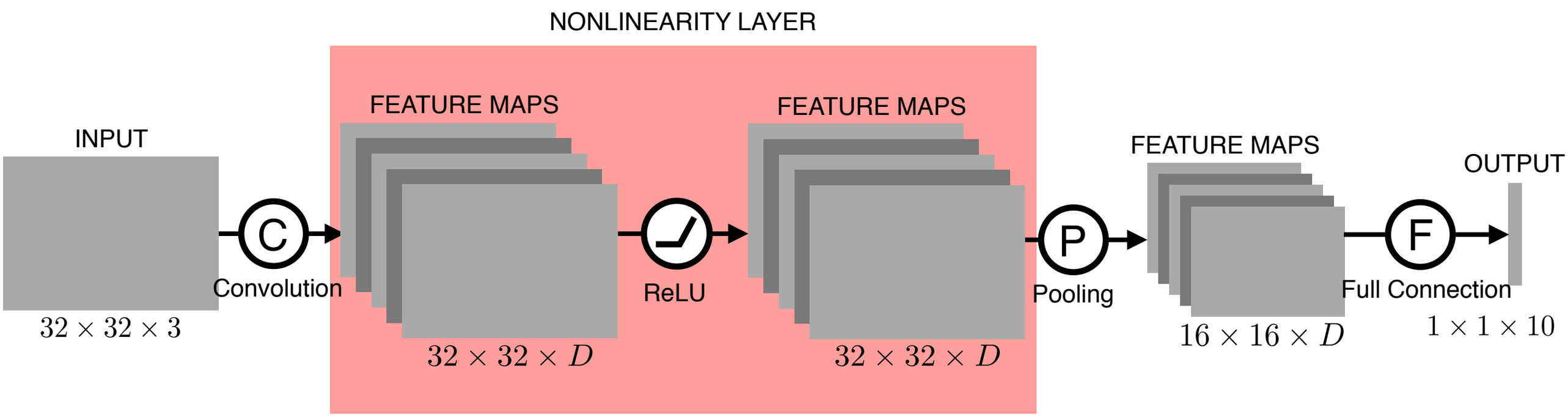
CONVNET ARCHITECTURE

Layers Used to Build Convolutional Networks



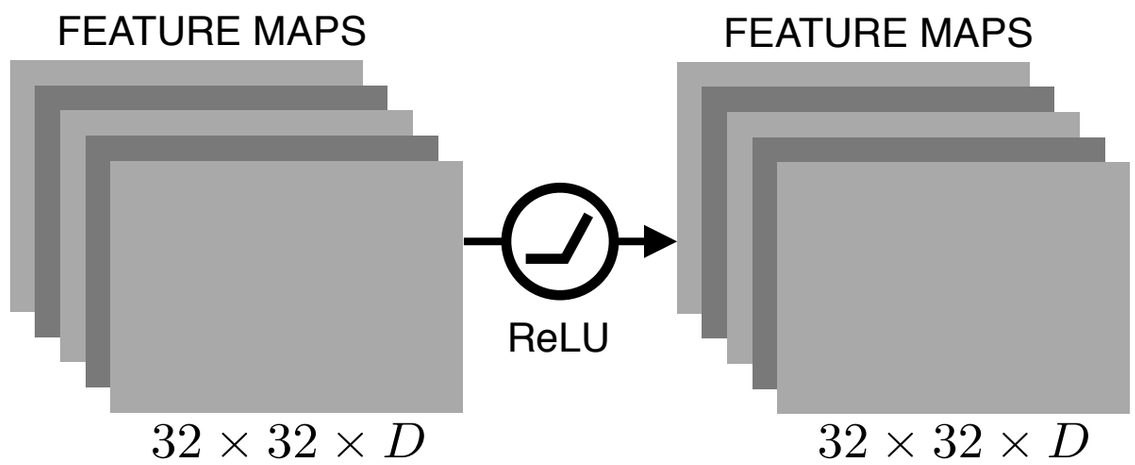
CONVNET ARCHITECTURE

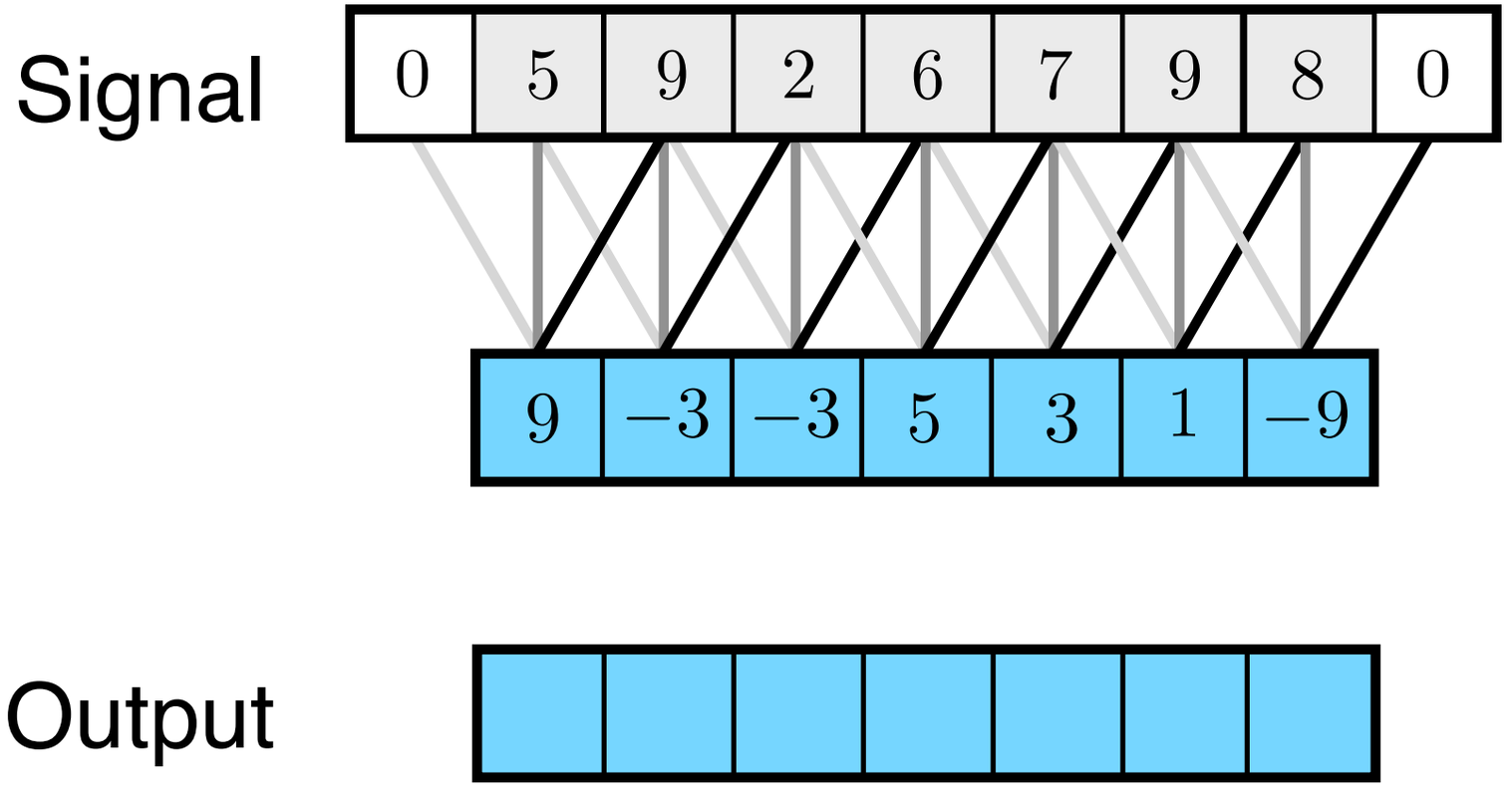
Layers Used to Build Convolutional Networks



CONVNET ARCHITECTURE

Nonlinearity Layer (e.g., ReLU)

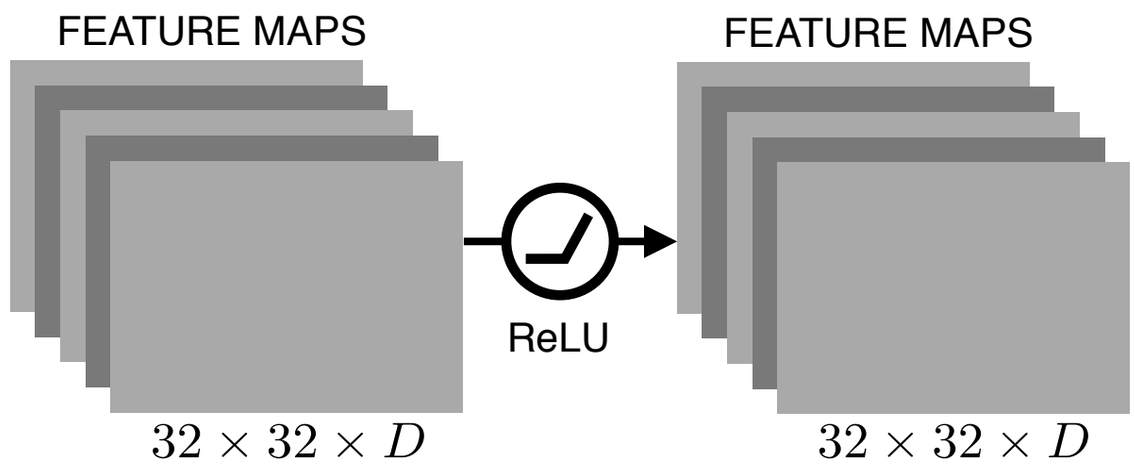


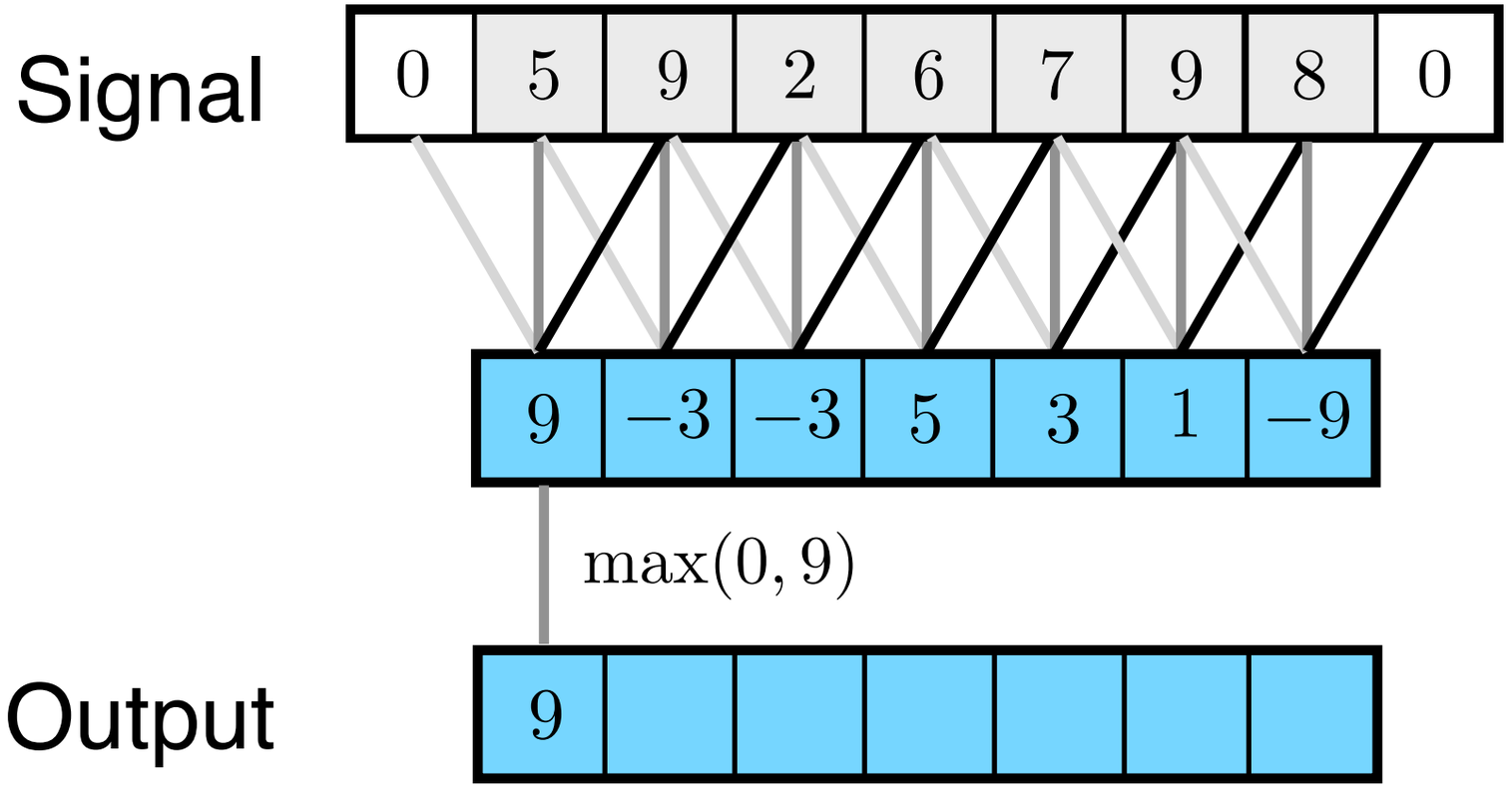


Pixel by pixel computation of $\max(0, x)$

CONVNET ARCHITECTURE

Nonlinearity Layer (e.g., ReLU)

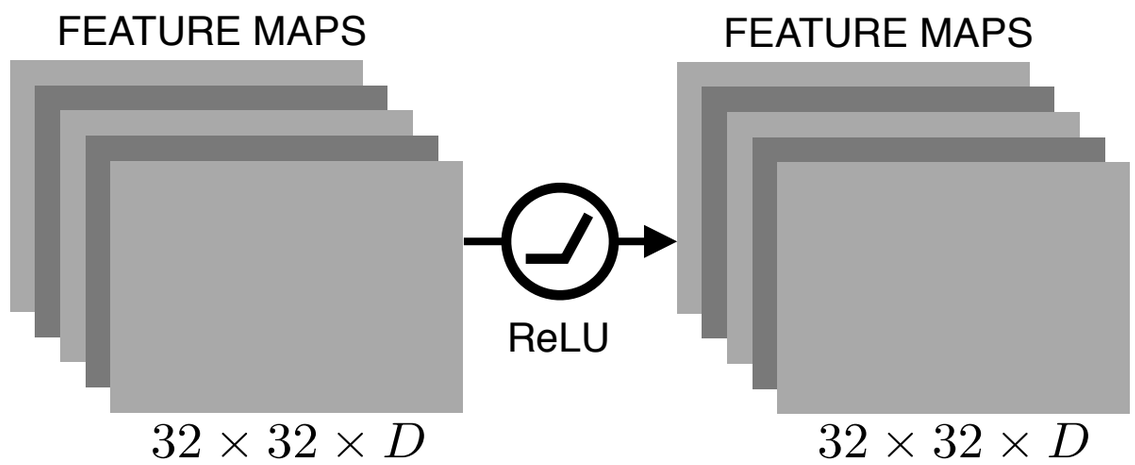


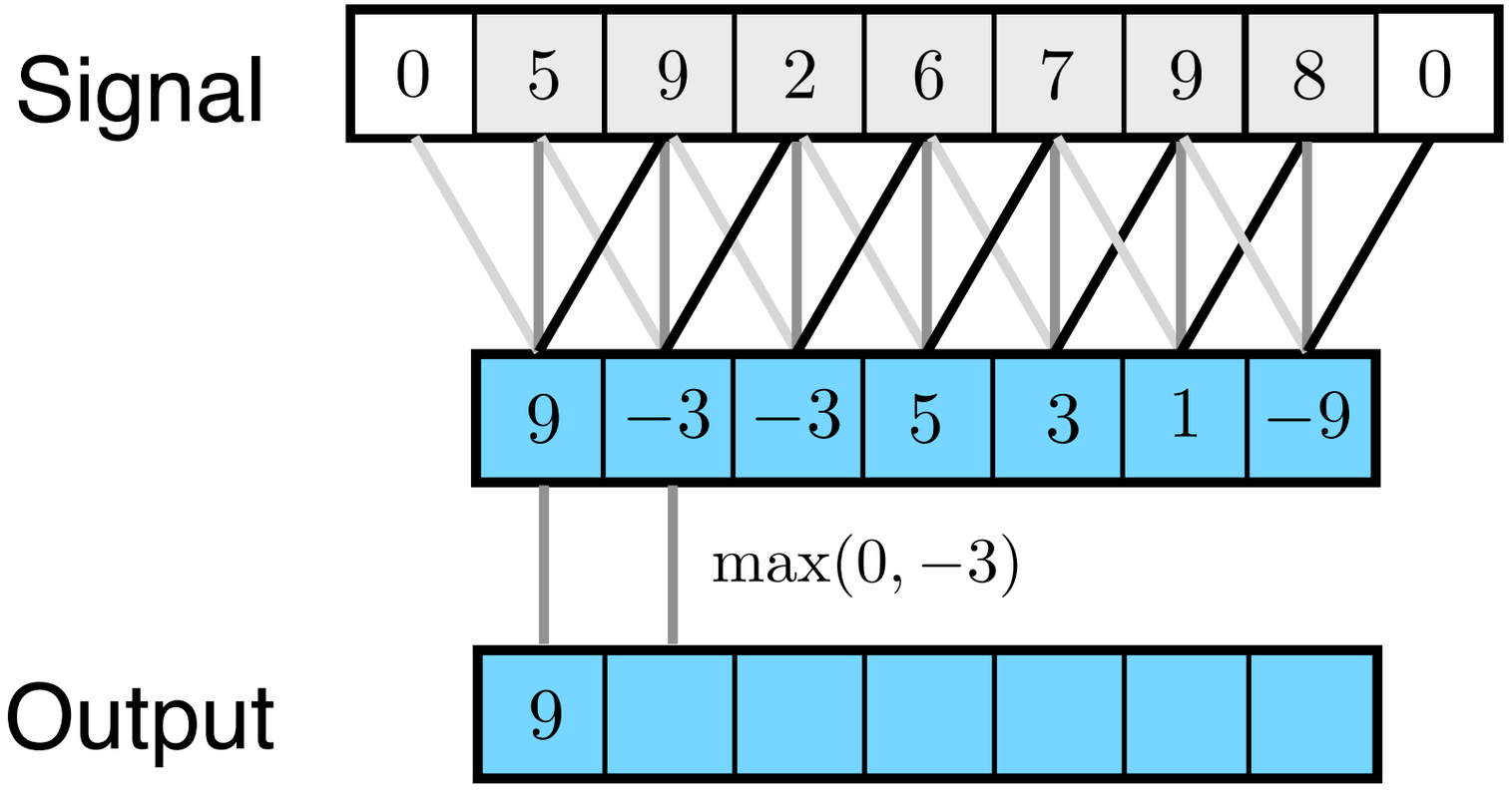


Pixel by pixel computation of $\max(0, x)$

CONVNET ARCHITECTURE

Nonlinearity Layer (e.g., ReLU)

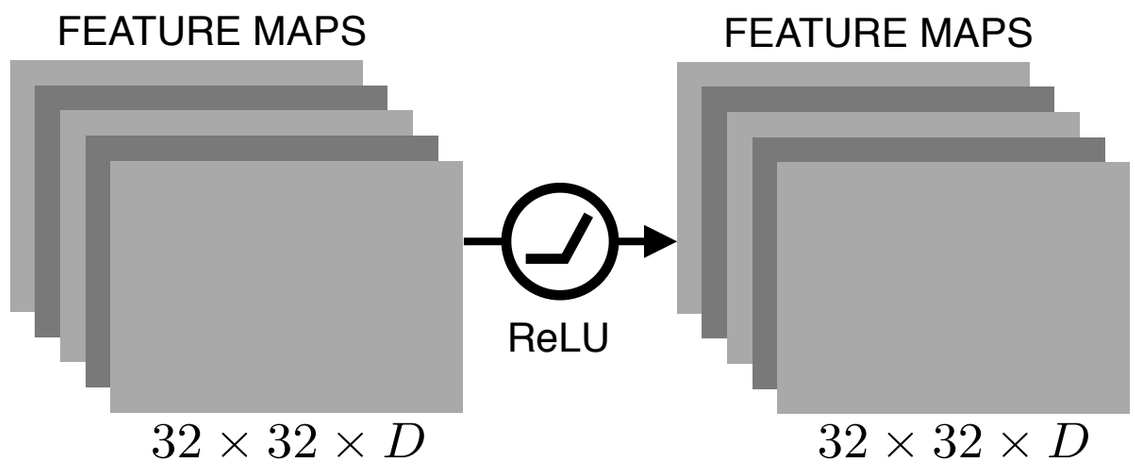


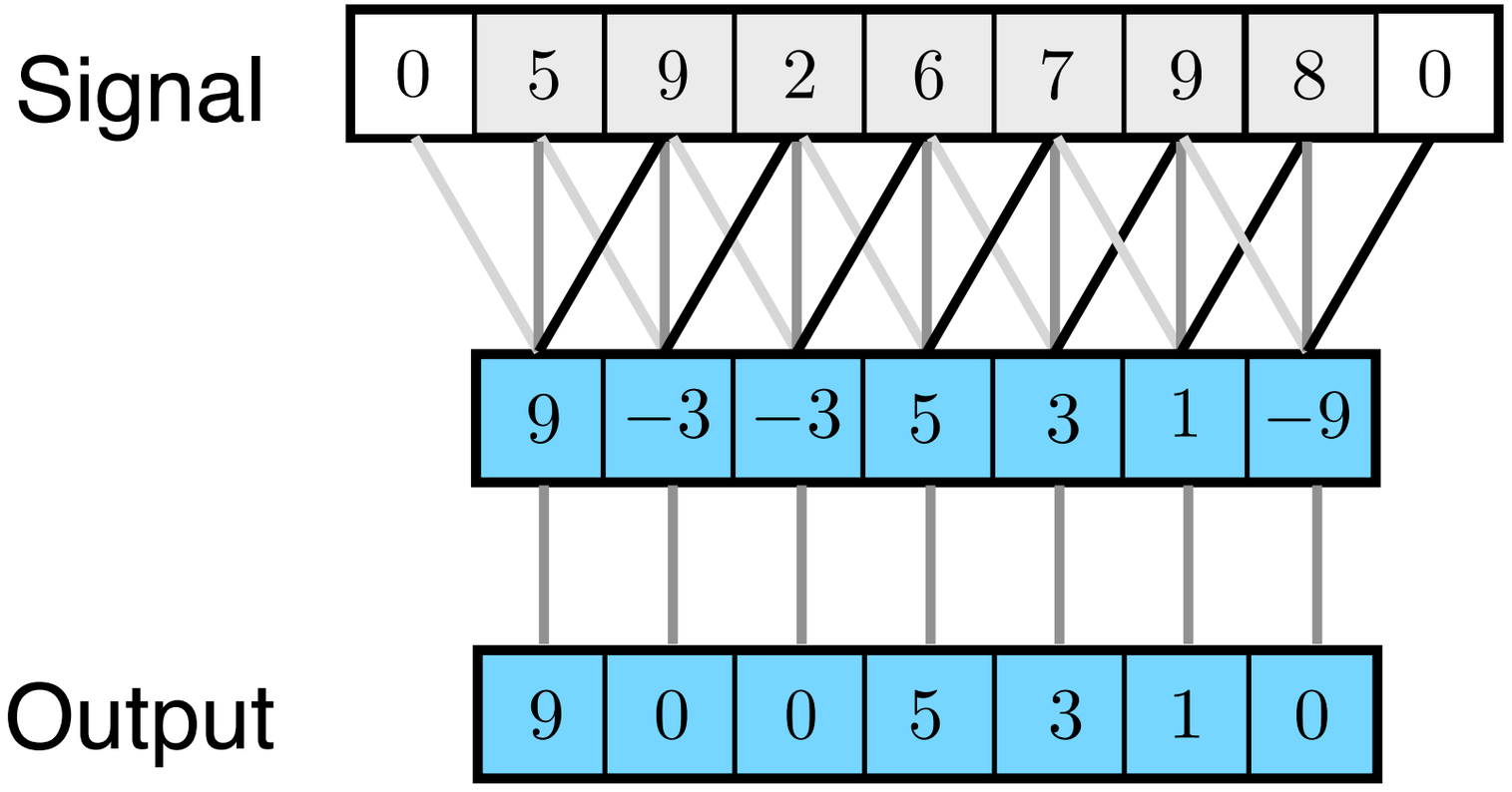


Pixel by pixel computation of $\max(0, x)$

CONVNET ARCHITECTURE

Nonlinearity Layer (e.g., ReLU)

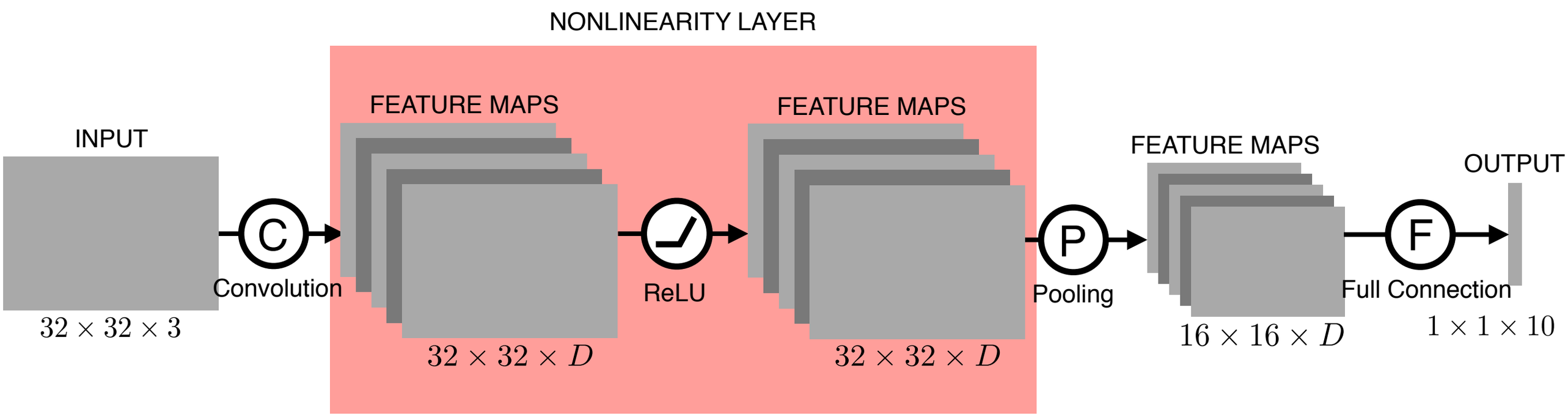




Pixel by pixel computation of $\max(0, x)$

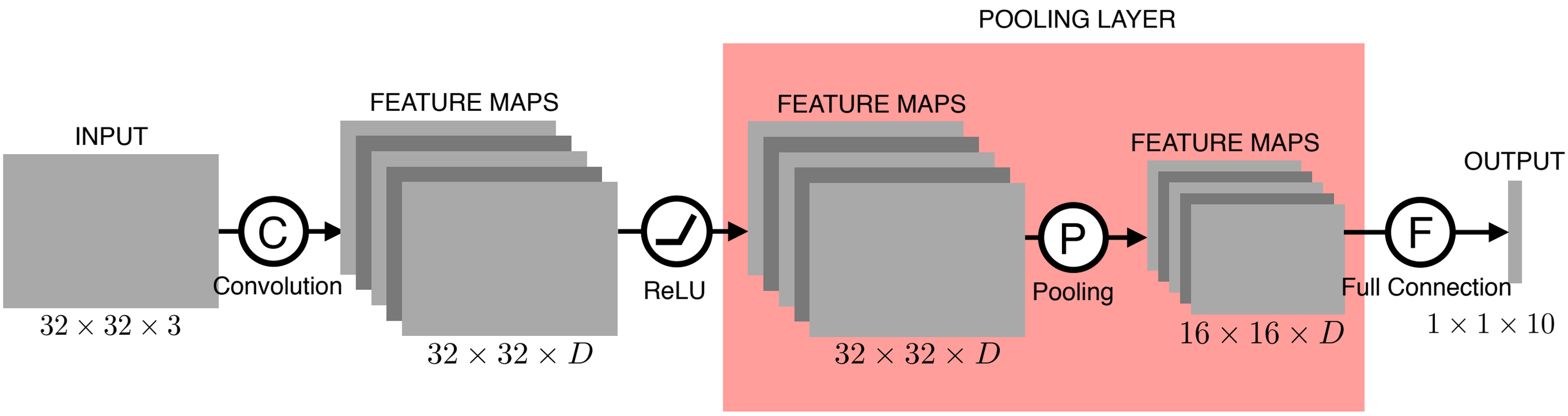
CONVNET ARCHITECTURE

Layers Used to Build Convolutional Networks



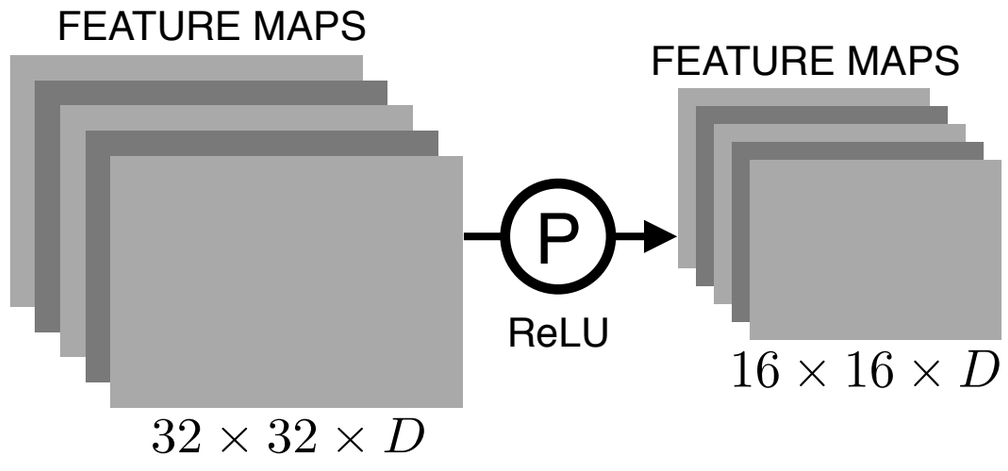
CONVNET ARCHITECTURE

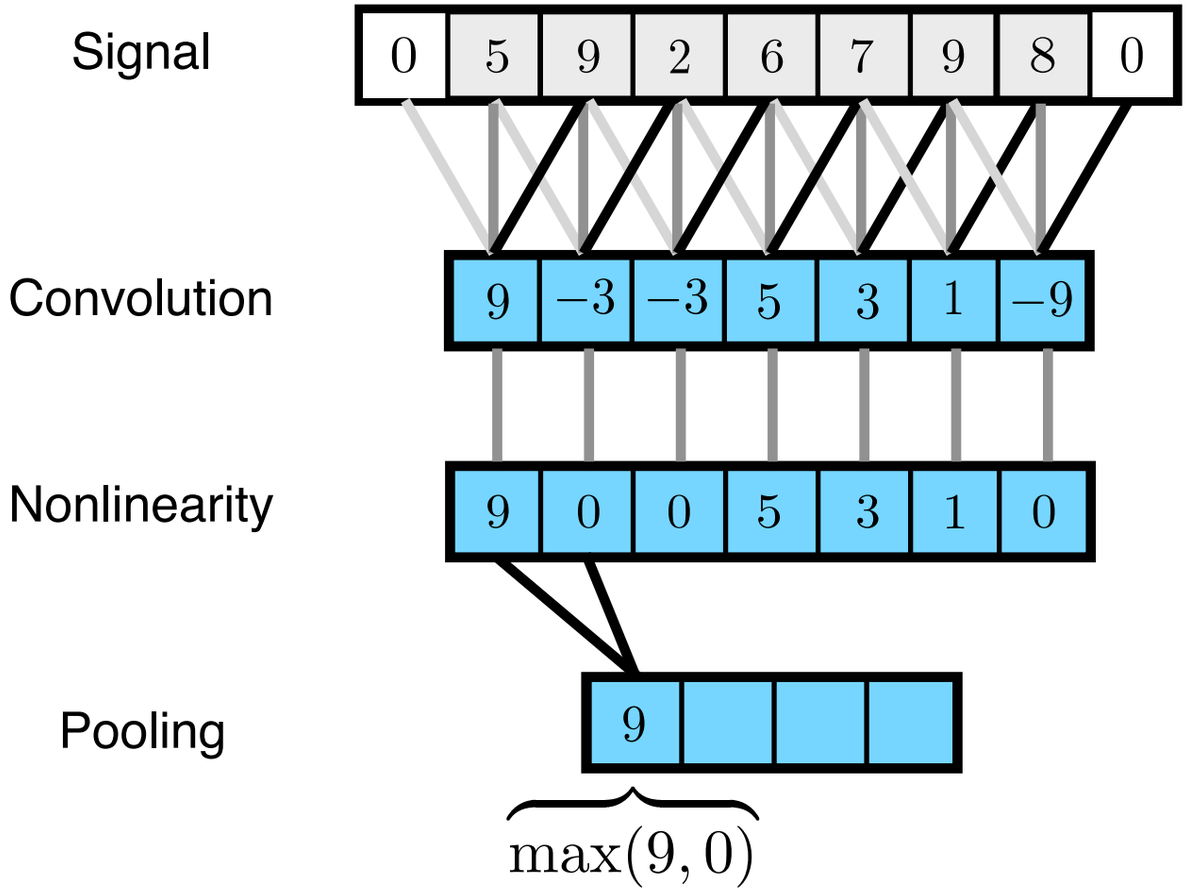
Layers Used to Build Convolutional Networks



CONVNET ARCHITECTURE

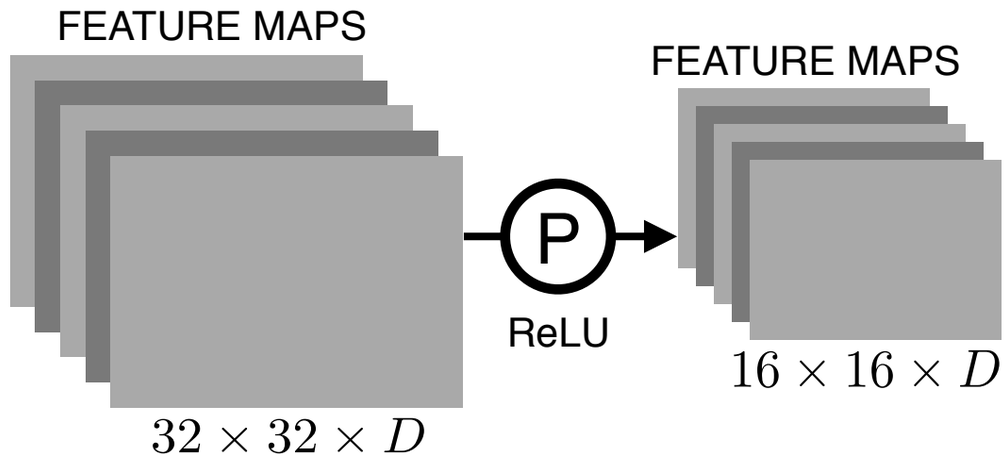
Pooling Layer (e.g., max-pooling)

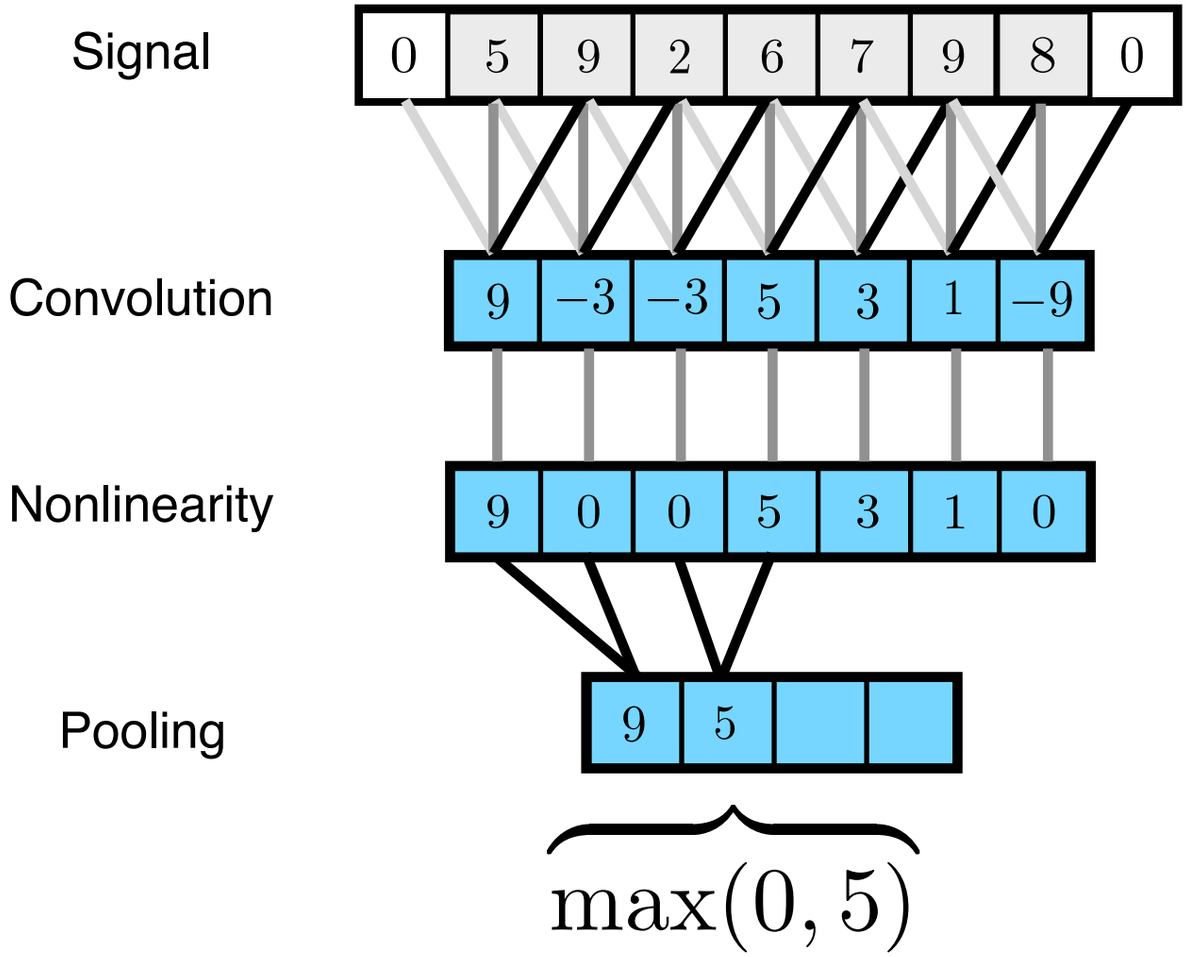




CONVNET ARCHITECTURE

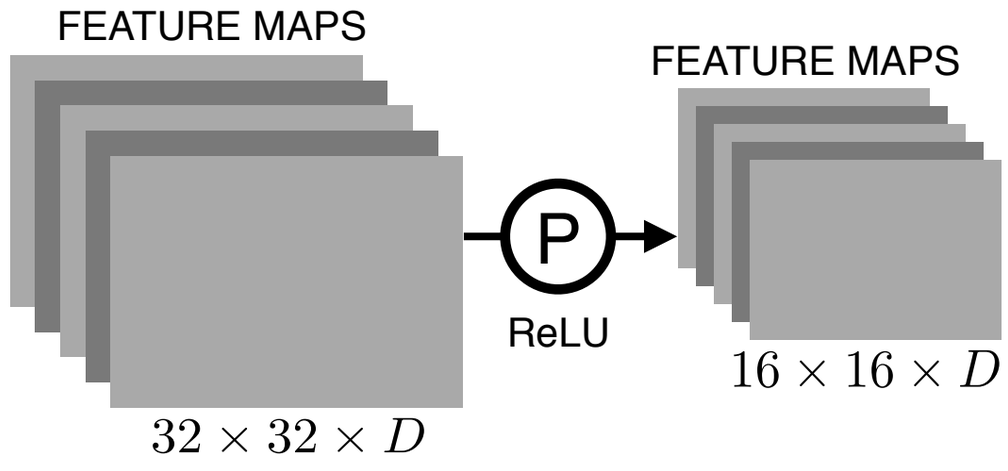
Pooling Layer (e.g., max-pooling)

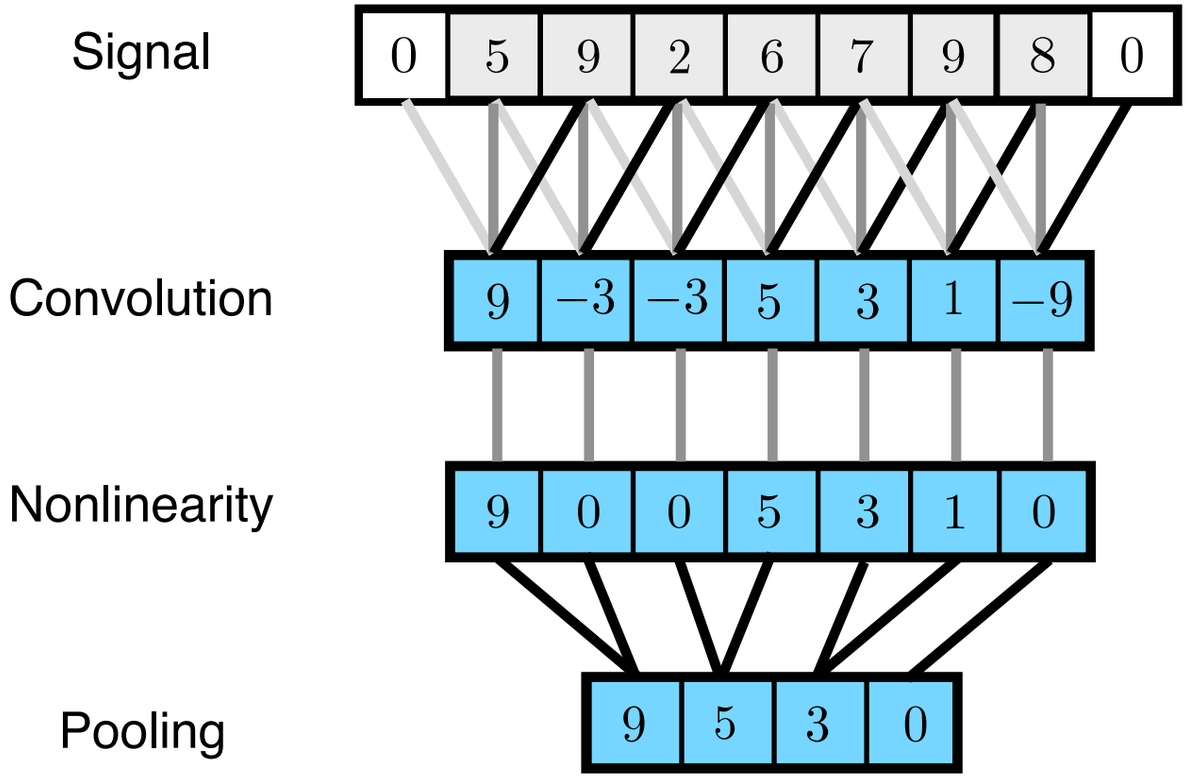




CONVNET ARCHITECTURE

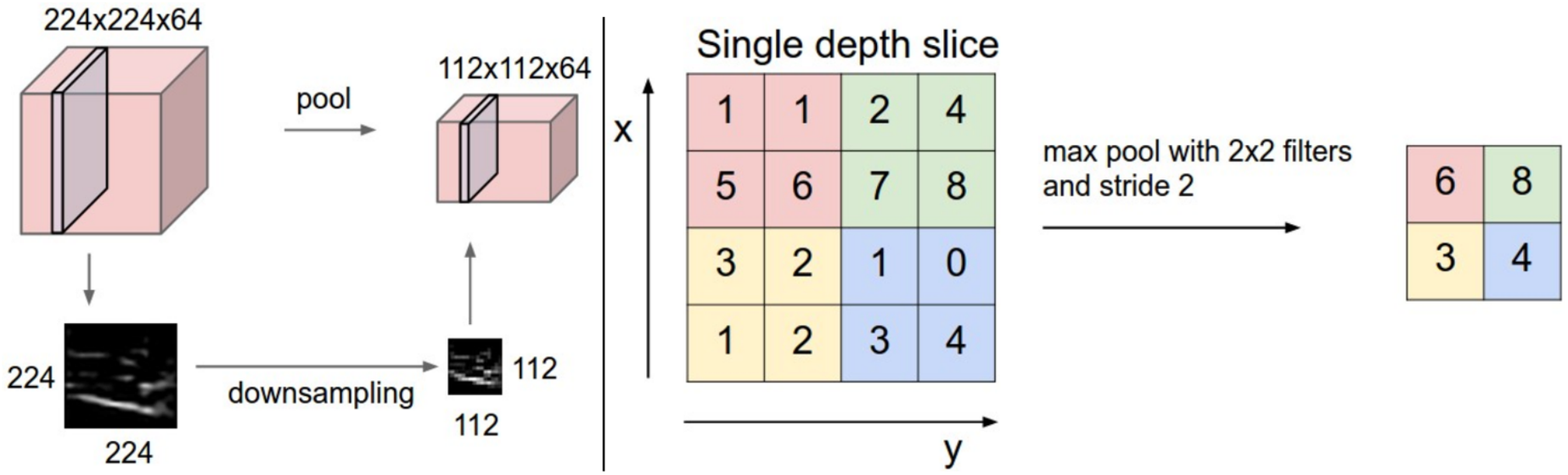
Pooling Layer (e.g., max-pooling)





POOLING LAYER

Pooling filter reduces the spatial size of the representation



Max-pooling: $f_i^n(u,v) = \max_{\bar{u}\in N(u),\bar{v}\in N(v)} f_i^{n-1}(\bar{u},\bar{v})$

Mean-pooling: $f_i^n(u,v) = \text{mean}_{\bar{u}\in N(u),\bar{v}\in N(v)} f_i^{n-1}(\bar{u},\bar{v})$

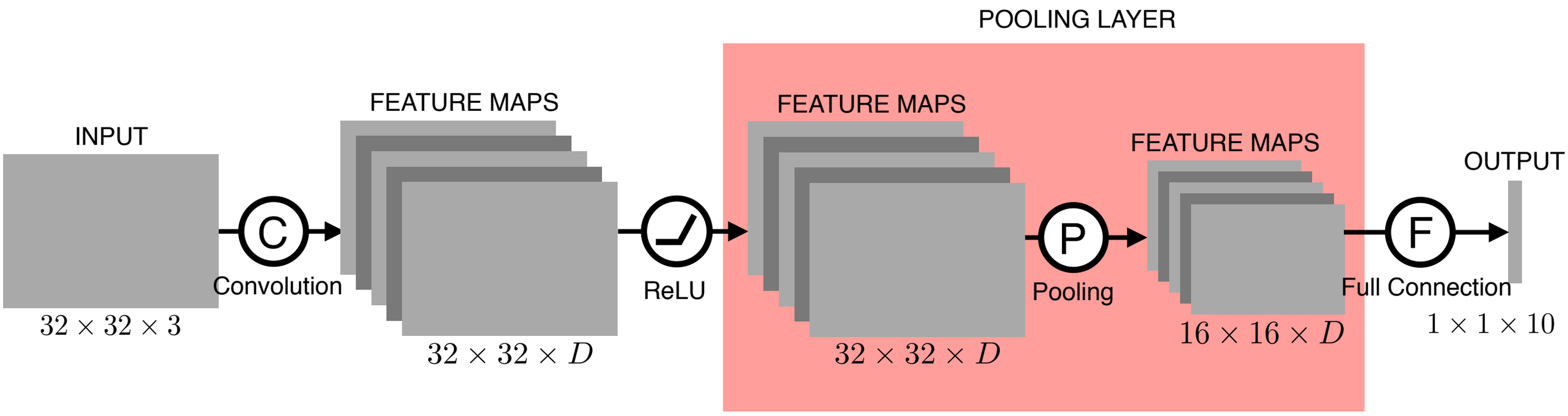
Hyperparameters

Spatial Extent F; Stride S

Content based on text and figure by Andrej Kar

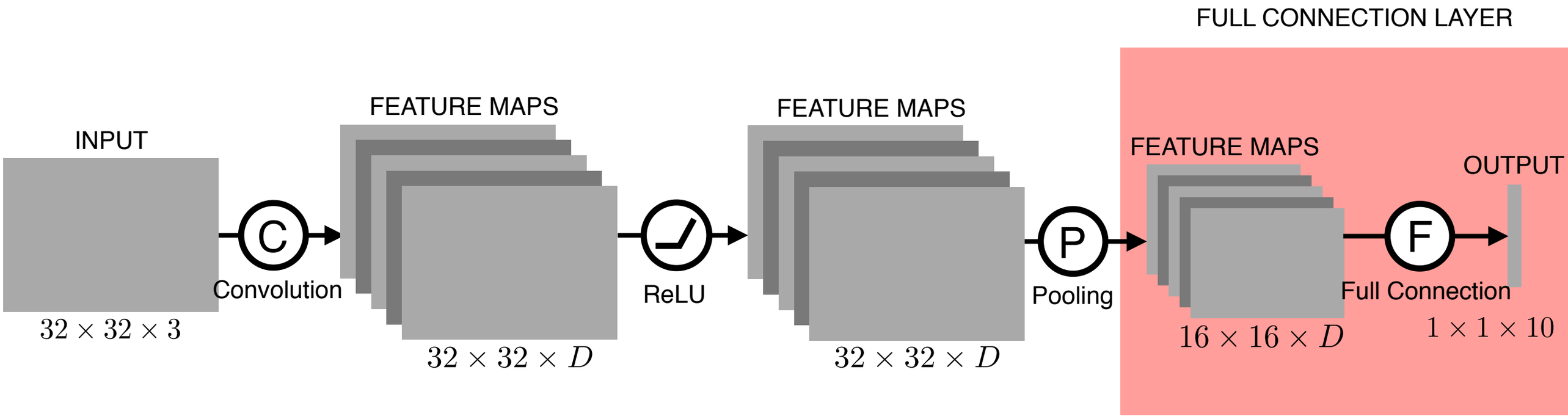
CONVNET ARCHITECTURE

Layers Used to Build Convolutional Networks



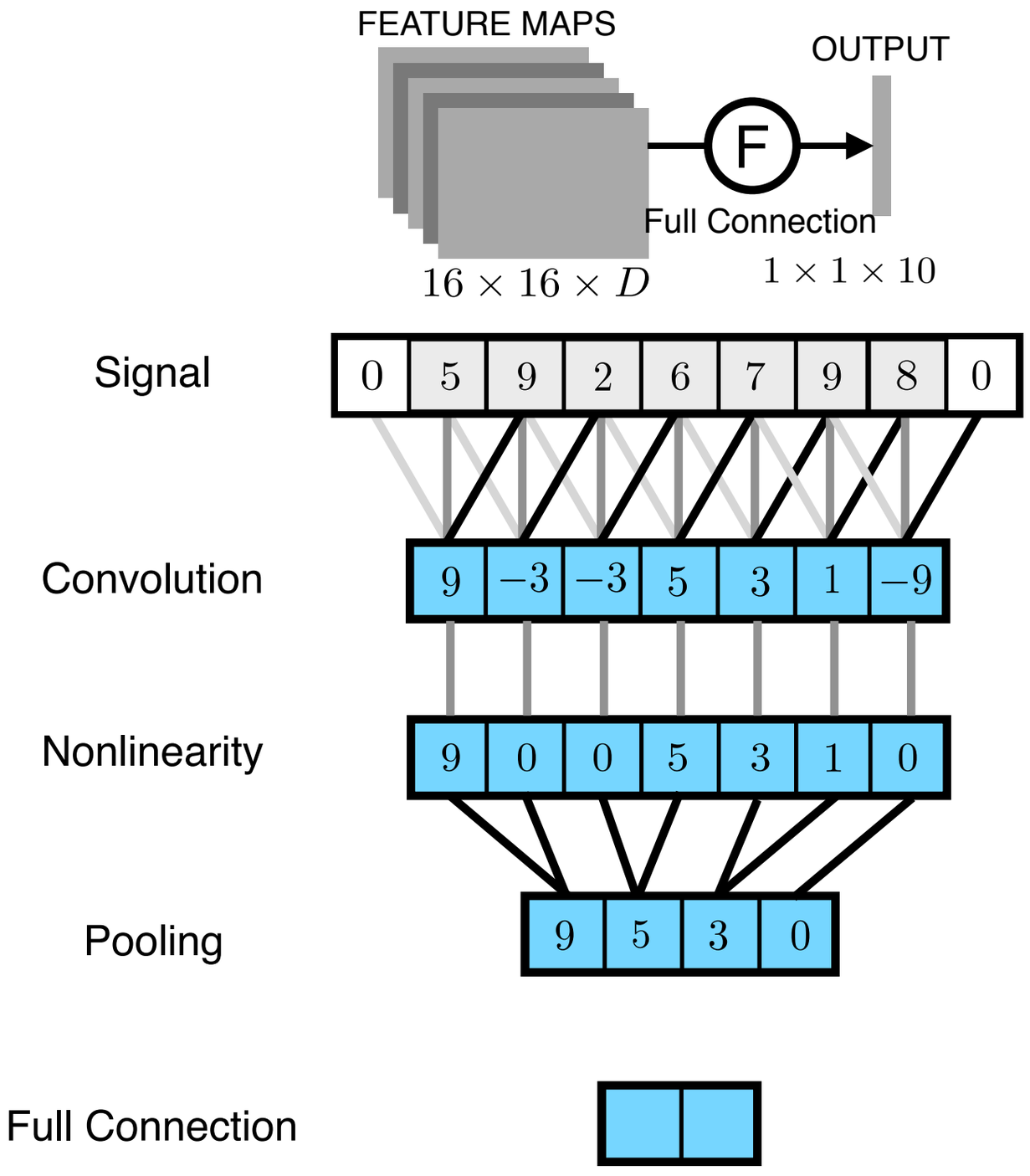
CONVNET ARCHITECTURE

Activations of an Example ConvNet Architecture



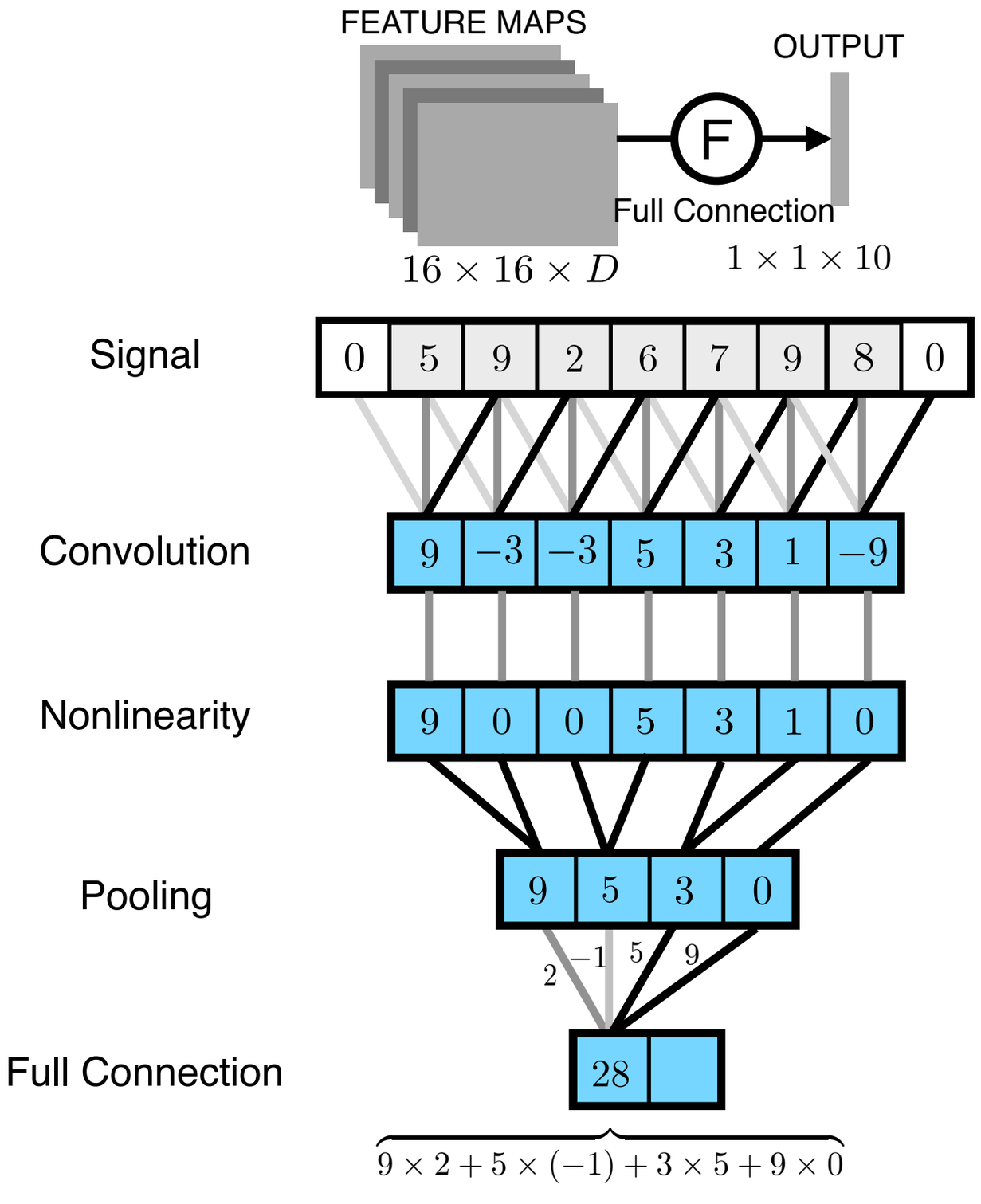
CONVNET ARCHITECTURE

Pooling Layer (e.g., max-pooling)



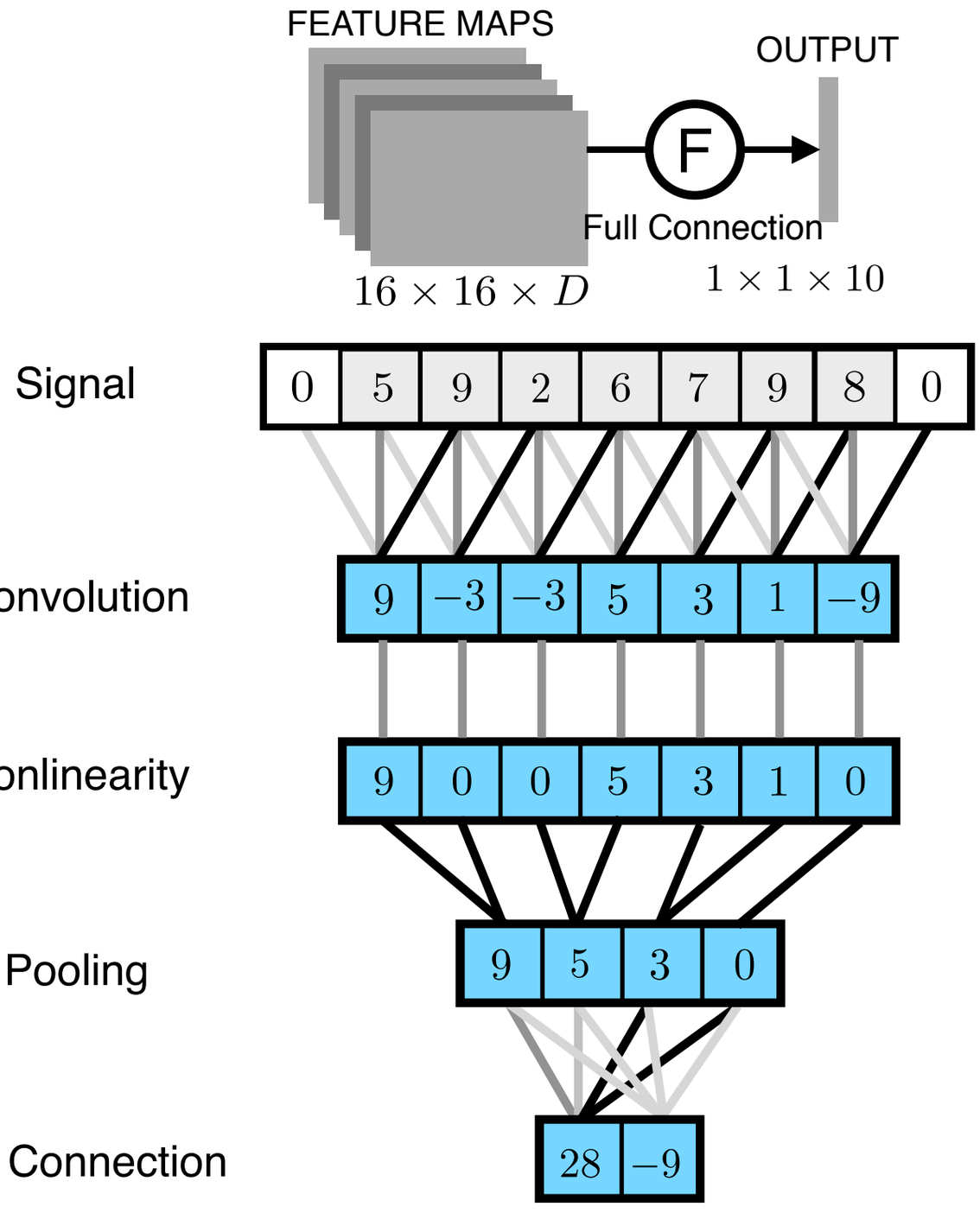
CONVNET ARCHITECTURE

Pooling Layer (e.g., max-pooling)



CONVNET ARCHITECTURE

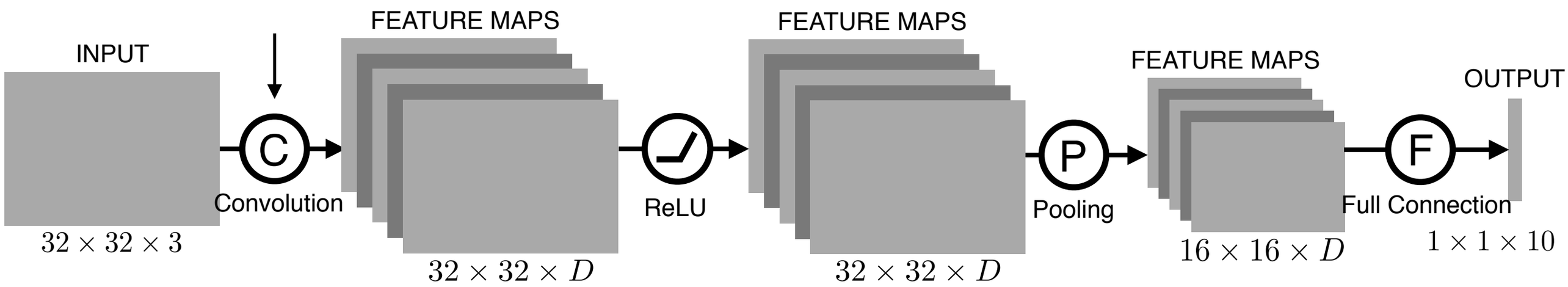
Pooling Layer (e.g., max-pooling)



Question: How many weights were in each layer?

CONV-NET ARCHITECTURE

Activations of an Example Conv-Net Architecture



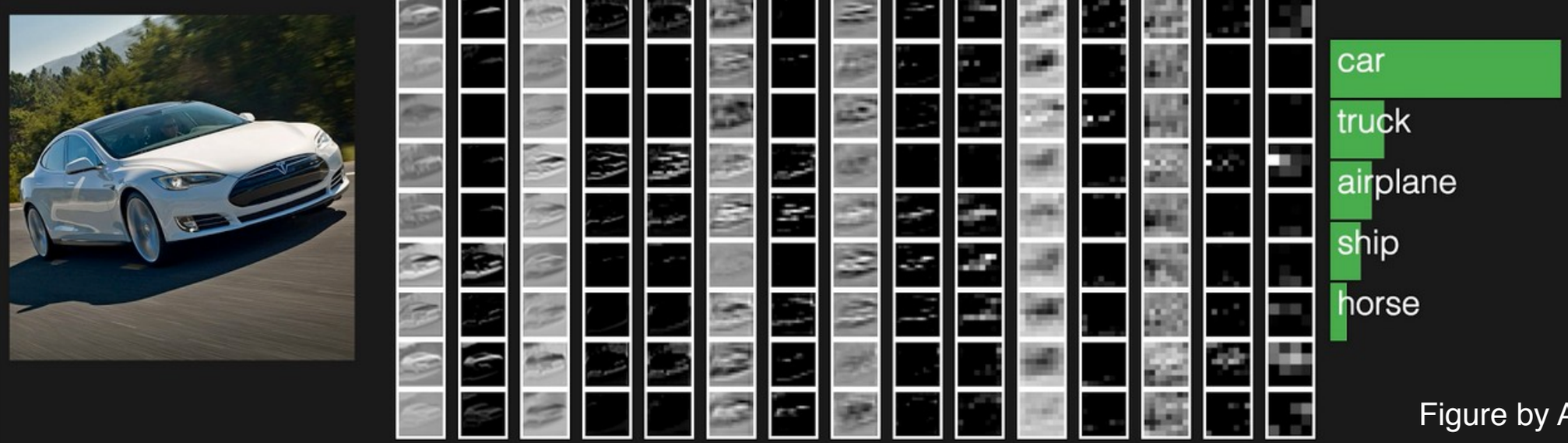
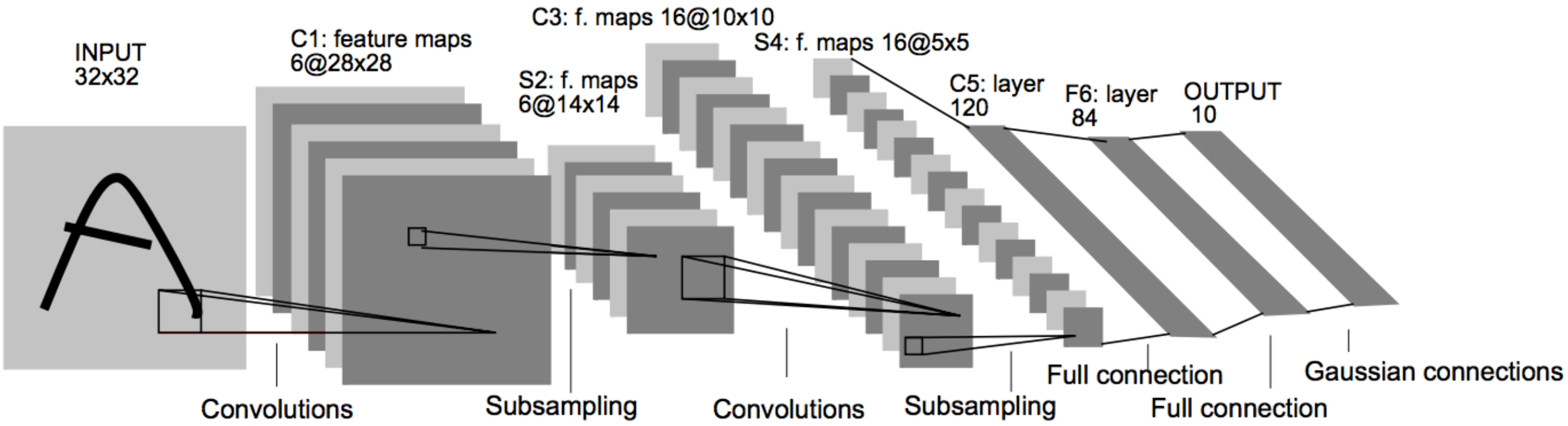


Figure by Andrej Karpathy

CONVNET ARCHITECTURES

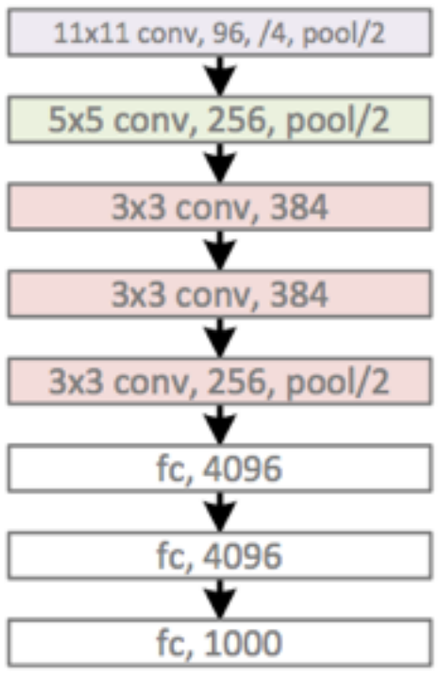
Case Studies



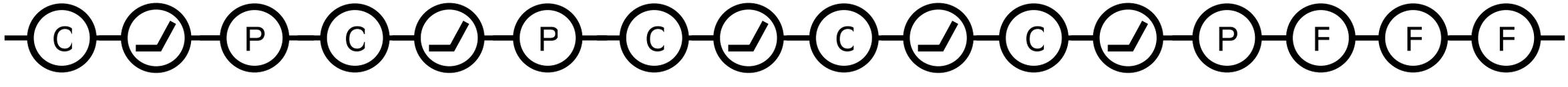
LeNet: Presented by Yann LeCun during the 1990s for reading digits. Had all the elements of modern architectures. Figure shows LeNet-5.

CONVNET ARCHITECTURES

Case Studies

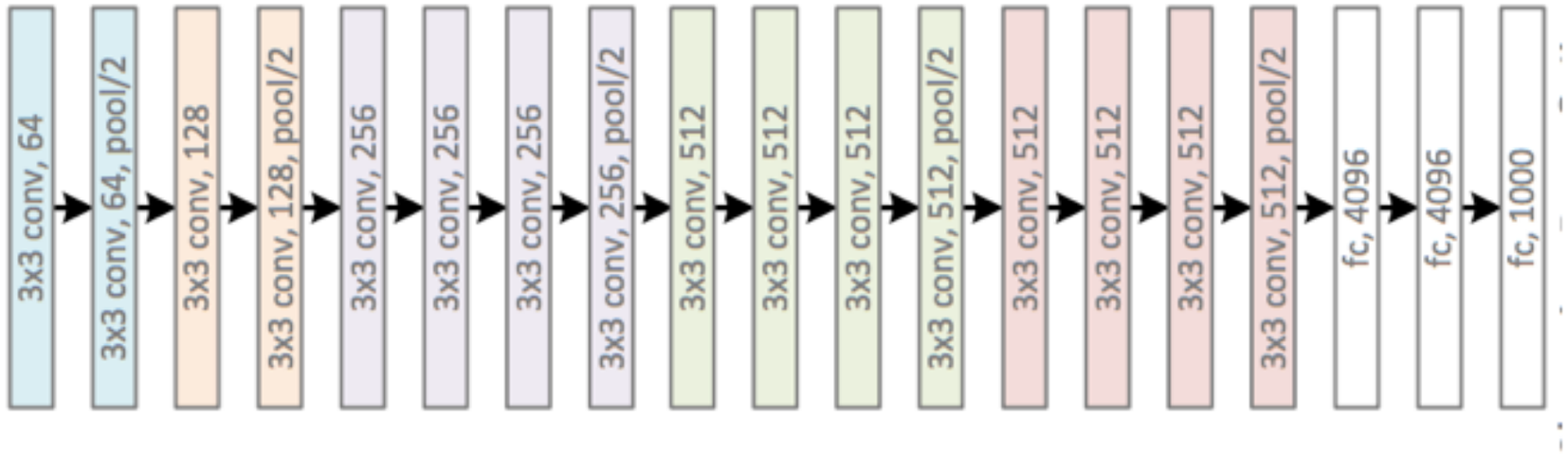


AlexNet: Repopularized convolutional networks by winning the Imagenet Challenge in 2012 [Krizhevsky et al. 2012]. Error of 16% vs 26% for second place!



CONVNET ARCHITECTURES

Case Studies

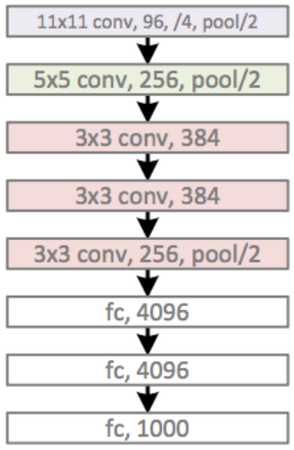


VGGNet

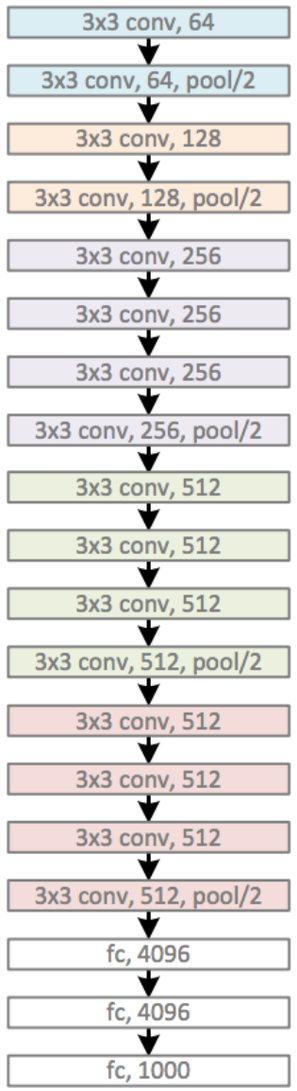
CONVOLUTIONAL NEURAL NETWORKS

A “Revolution of Depth”

AlexNet, 8 layers
(ILSVRC 2012)



VGG, 19 layers
(ILSVRC 2014)

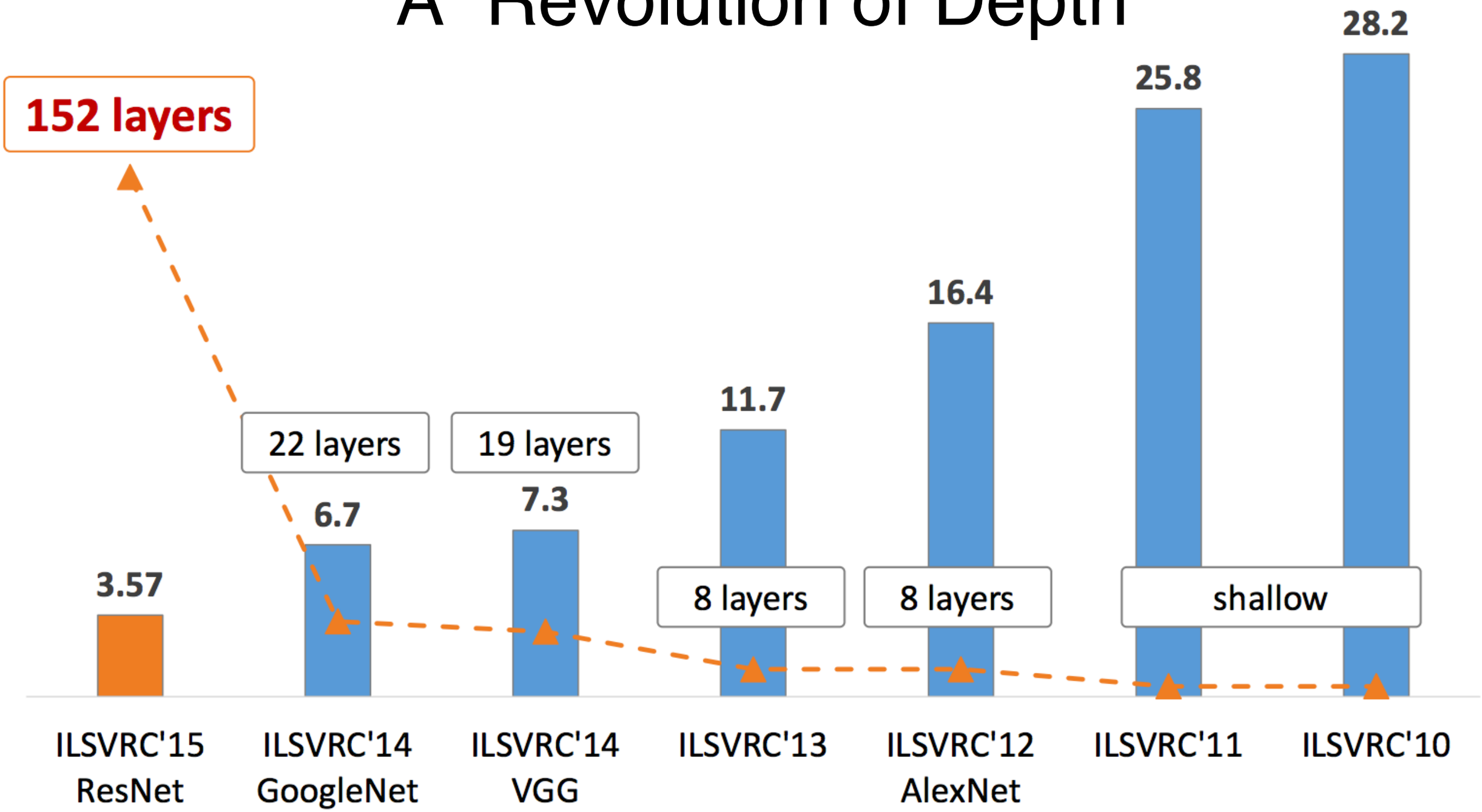


GoogleNet, 22 layers
(ILSVRC 2014)

Slide By Kaiming H

CONVOLUTIONAL NEURAL NETWORKS

A "Revolution of Depth"



Slide By Kaiming H

CONVOLUTIONAL NEURAL NETWORKS

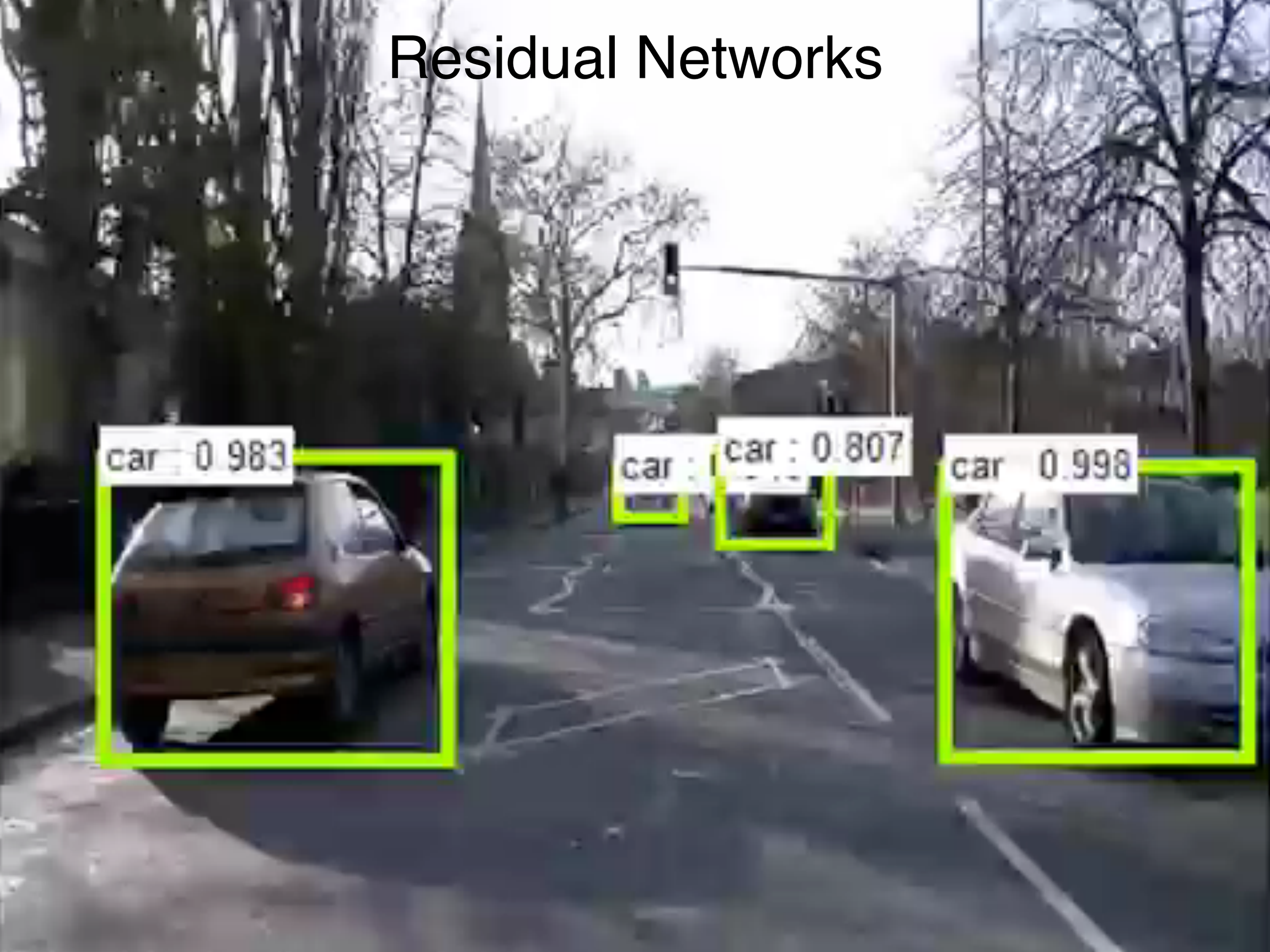
A “Revolution of Depth”

AlexNet, 8 layers
(ILSVRC 2012)

VGG, 19 layers
(ILSVRC 2014)

ResNet, 152 layers
(ILSVRC 2015)

Residual Networks



LECTURE SUMMARY

Deep Learning II

1. Multiple Layer Neural Network: State-of-the-art Classification
2. Backpropagation Algorithm: Feedforward Networks
3. Convolutional Neural Networks
    1. Convolution Layers
    2. Nonlinearity Layers
    3. Max Pooling Layers
    4. Fully Connected Layers
4. Case Studies

REFERENCES

Backpropagation and Convolutional Nets

- LeCun, Yann A., et al. "Efficient backprop." Neural networks: Tricks of the trade. Springer Berlin Heidelberg, 2012. 9-48.
- Bottou, Léon. "Stochastic gradient tricks." Neural Networks, Tricks of the Trade, Reloaded (2012): 430-445.
- Srivastava, Nitish, et al. "Dropout: a simple way to prevent neural networks from overfitting." Journal of Machine Learning Research 15.1 (2014): 1929-1958.
- Ioffe, Sergey, and Christian Szegedy. "Batch normalization: Accelerating deep network training by reducing internal covariate shift." arXiv preprint arXiv:1502.03167 (2015).
- Rumelhart, David E., Geoffrey E. Hinton, and Ronald J. Williams. "Learning representations by back-propagating errors." Cognitive modeling 5.3 (1988): 1.
- Krizhevsky, Alex, Ilya Sutskever, and Geoffrey E. Hinton. "Imagenet classification with deep convolutional neural networks." Advances in neural information processing systems. 2012.
- LeCun, Yann, Yoshua Bengio, and Geoffrey Hinton. "Deep learning." Nature 521.7553 (2015): 436-444.
- Simonyan, Karen, and Andrew Zisserman. "Very deep convolutional networks for large-scale image recognition." arXiv preprint arXiv:1409.1556 (2014).

REFERENCES

Backpropagation and Convolutional Nets

- LeCun, Yann, et al. "Gradient-based learning applied to document recognition." Proceedings of the IEEE 86.11 (1998): 2278-2324.