Basics of Deep Neural Networks

Understanding Visual Appearance Through Deep Learning
March 4th, 2020

Preethi Jyothi (CSE)

What is deep learning?

"Deep learning allows computational models that are composed of multiple processing layers to learn representations of data with multiple levels of abstraction."

"Representation learning is a set of methods that allows a machine to be fed with raw data and to automatically discover the representations needed for detection or classification. Deep-learning methods are representation-learning methods with multiple levels of representation, obtained by composing simple but non-linear modules that each transform the representation at one level (starting with the raw input) into a representation at a higher, slightly more abstract level."

LeCun, Y., Bengio, Y., & Hinton, G. (2015). Deep learning. Nature, 521(7553), 436

History of (Deep) Neural Networks

- McCulloch-Pitts Neuron Model (1943)
- Perceptrons (1957)
- Backpropagation (1960)
- Backpropagation for neural networks (1986)
- Convolutional neural networks (1989)

⋮

- Deep learning for speech recognition (2009)
- AlexNet (2012)
- Generative Adversarial Networks (GANs) (2014)
- AlphaGo (2016)

Why the resurgence?

- McCulloch-Pitts Neuron Model (1943)
- Perceptrons (1957)
- Backpropagation (1960)
- Backpropagation for neural networks (1986)
- Convolutional neural networks (1989)

⋮

- Deep learning for speech recognition (2009)
- AlexNet (2012)
- Generative Adversarial Networks (GANs) (2014)
- AlphaGo (2016)

Vast amounts of data

+

Specialized hardware,
Graphics Processing Units (GPUs)

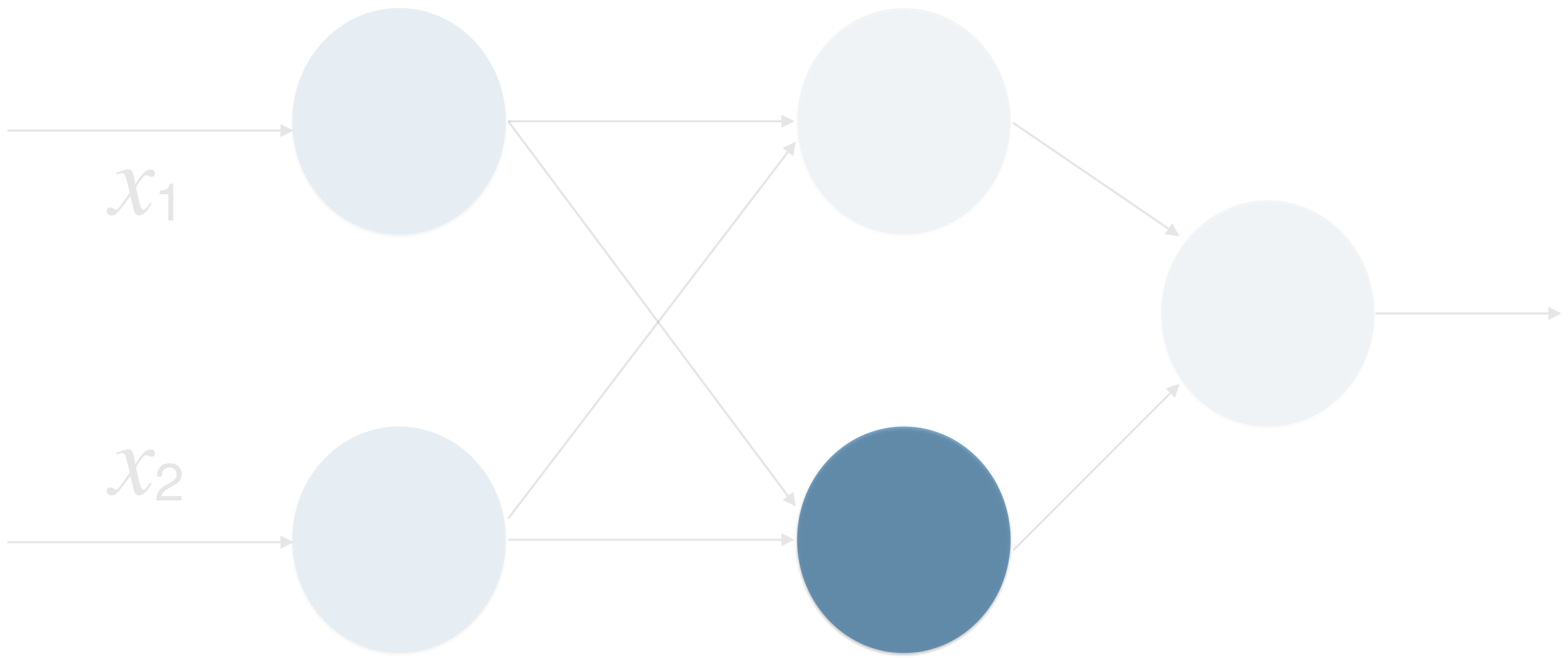
+

Improved optimization techniques
and new model variants/libraries/toolkits

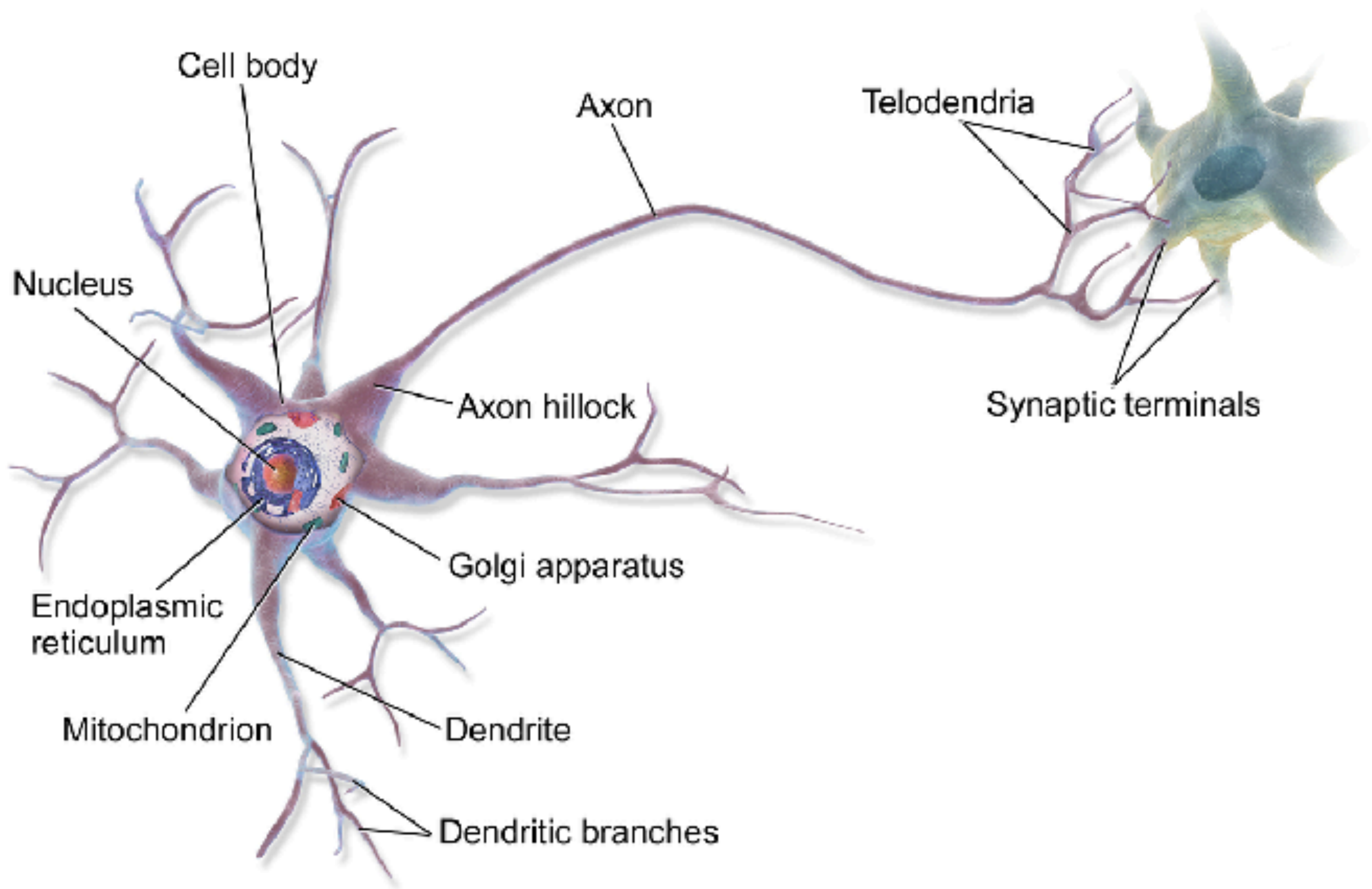
Nuts and Bolts of Deep Neural Networks

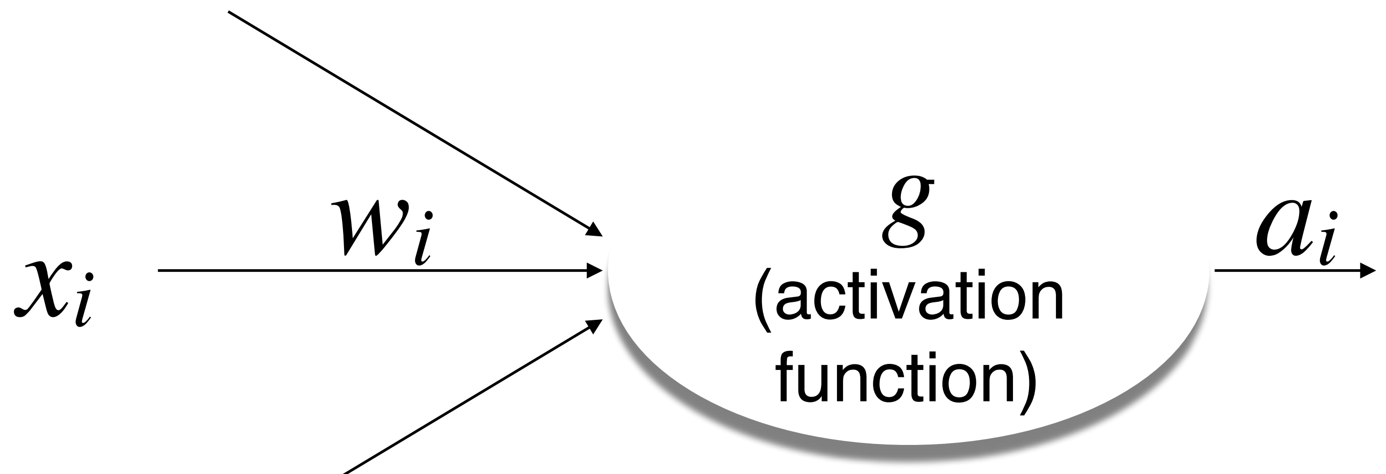
Feed-forward Neural Network

Single Neuron



Single neuron



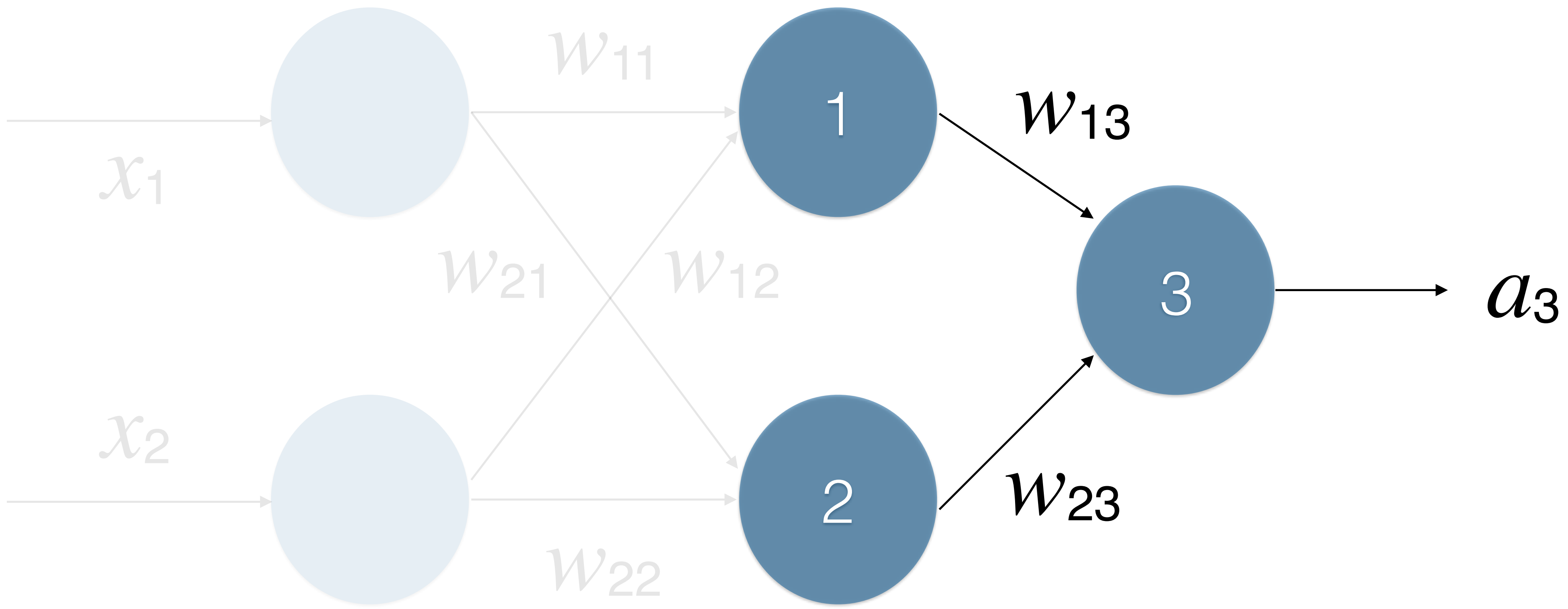


$$a_i = g(\sum_i w_i x_i)$$

Image from: https://upload.wikimedia.org/wikipedia/commons/1/10/Blausen_0657_MultipolarNeuron.png

Feed-forward Neural Network

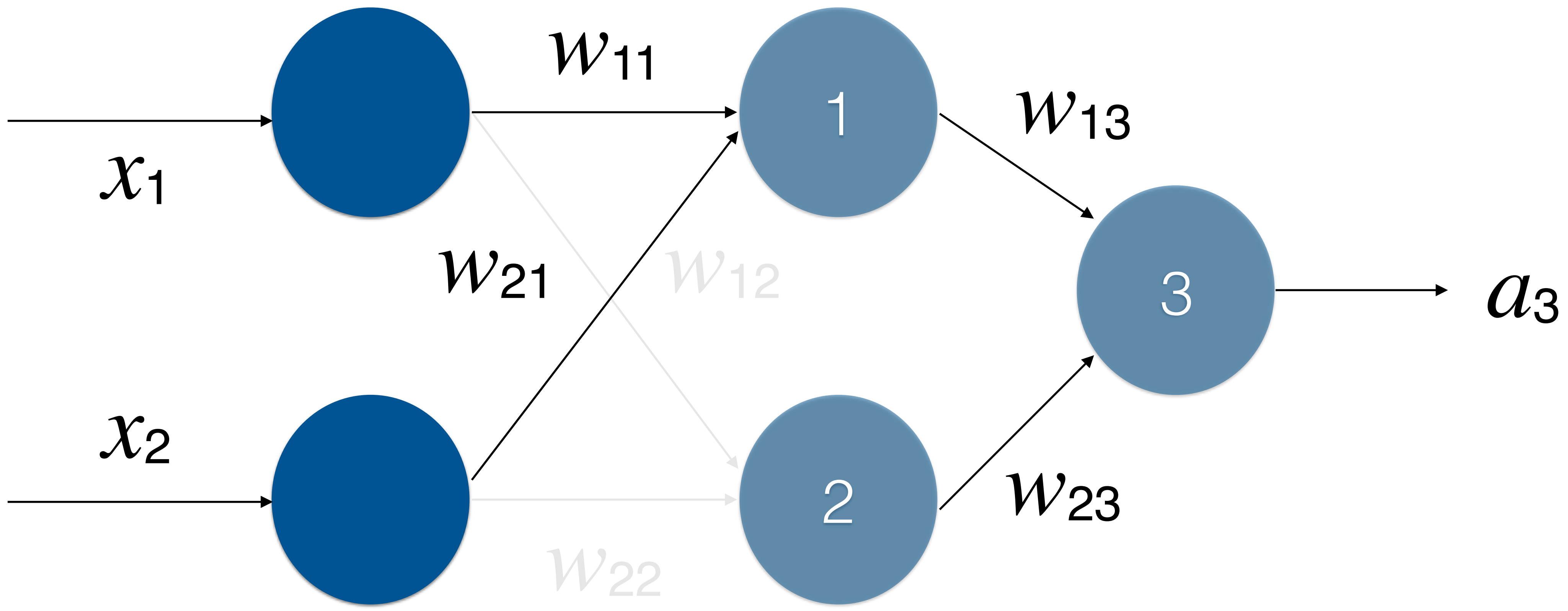
Parameterized Model



$$a_3 = g(w_{13} \cdot a_1 + w_{23} \cdot a_2 + b_3)$$

Feed-forward Neural Network

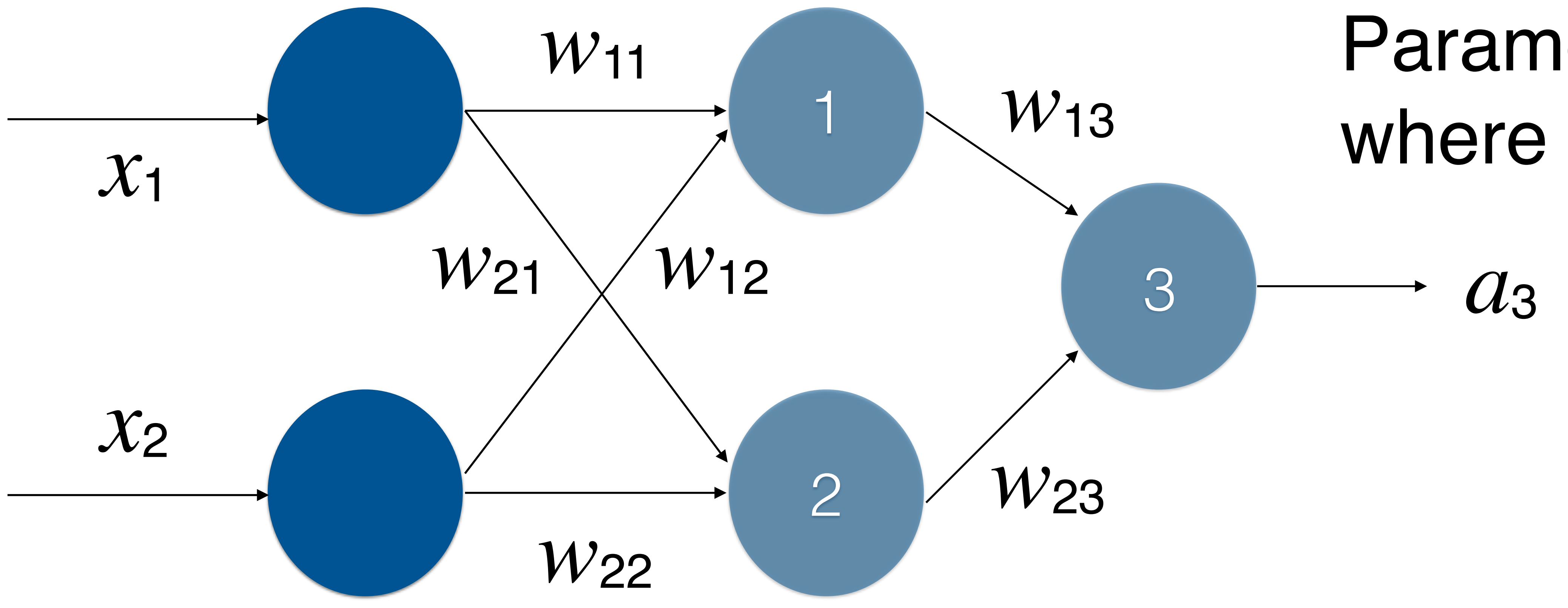
Parameterized Model



$$
\begin{aligned}
a_3 &= g(w_{13} \cdot a_1 + w_{23} \cdot a_2 + b_3) \\
&= g(w_{13} \cdot \big(g(w_{11} \cdot x_1 + w_{21} \cdot x_2 + b_1)\big) \\
&\qquad\qquad + \qquad \cdots
\end{aligned}
$$

Feed-forward Neural Network

Parameterized Model



Parameters of the network: $\mathrm{NN}(\mathbf{x};\theta)$ where θ refers to all w_{ij} and b_j

$$
\begin{aligned}
a_3 &= g(w_{13}\cdot a_1 + w_{23}\cdot a_2 + b_3) \\
&= g(w_{13}\cdot \big(g(w_{11}\cdot x_1 + w_{21}\cdot x_2 + b_1)\big) \\
&\qquad + \; w_{23}\cdot \big(g(w_{12}\cdot x_1 + w_{22}\cdot x_2 + b_2)\big) + b_3)
\end{aligned}
$$

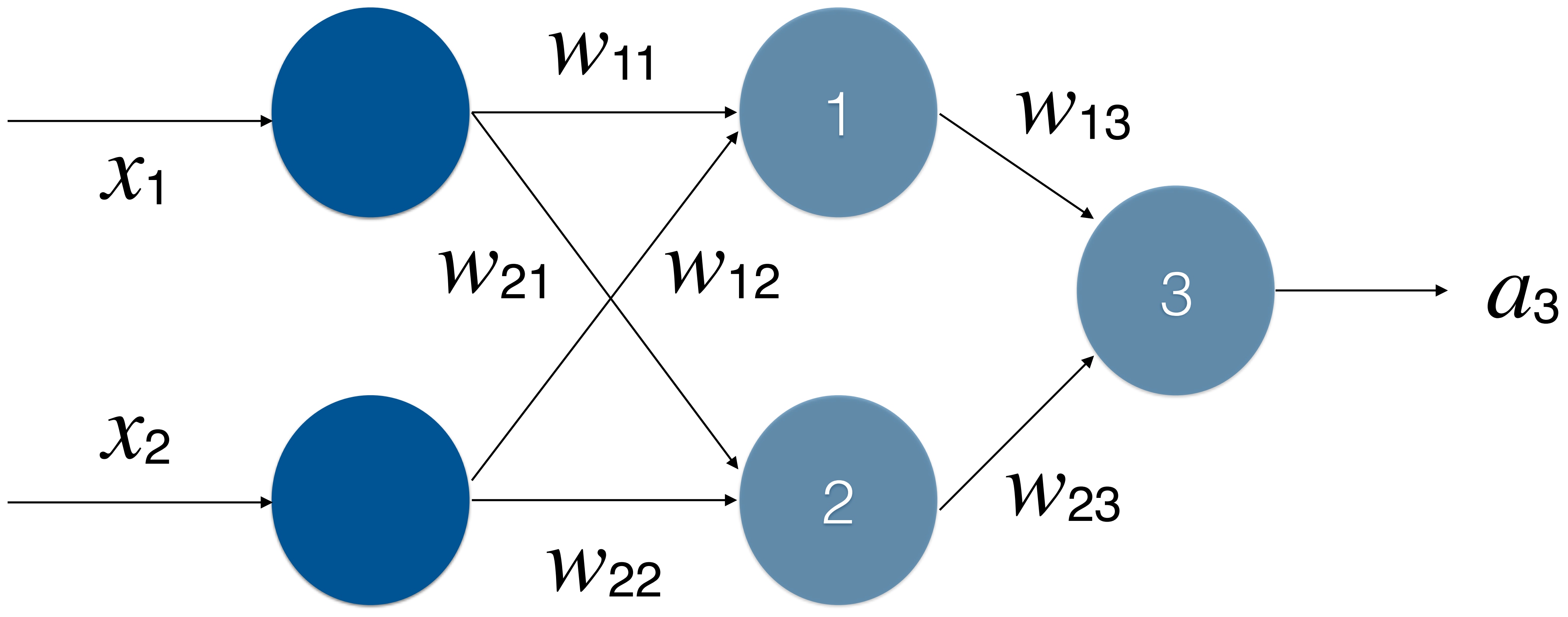
Compact matrix notation: Input $\mathbf{x} = [x_1, x_2]$ is written as a 2-dimensional vector and the layer above it is a 2-dimensional vector $\mathbf{h}$, a fully-connected layer is associated with:

$$\mathbf{h} = \mathbf{x}\mathbf{W} + \mathbf{b}$$

where w_{ij} in $\mathbf{W}$ is the weight of the connection between i^{th} neuron in the input row and j^{th} neuron in the first hidden layer and $\mathbf{b}$ is the bias vector.

Feed-forward Neural Network

Parameterized Model



$$
\begin{aligned}
a_3 &= g(w_{13} \cdot a_1 + w_{23} \cdot a_2 + b_3) \\
&= g(w_{13} \cdot \big(g(w_{11} \cdot x_1 + w_{21} \cdot x_2 + b_1)\big) \\
&\qquad\qquad + \; w_{23} \cdot \big(g(w_{12} \cdot x_1 + w_{22} \cdot x_2 + b_2)\big) + b_3)
\end{aligned}
$$

The simplest neural network is the perceptron:

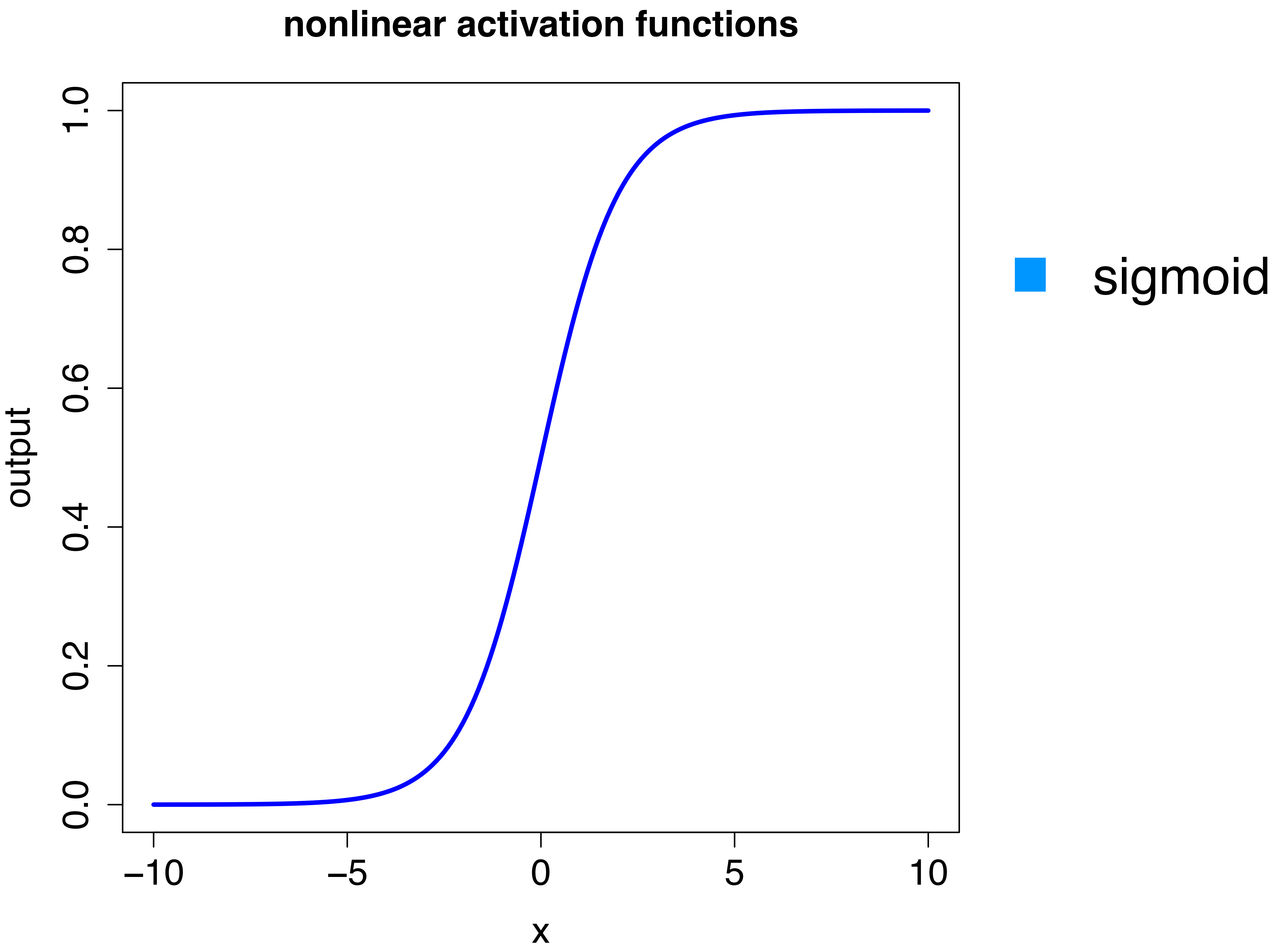
$$\text{Perceptron}(\mathbf{x}) = \mathbf{x}\mathbf{W} + \mathbf{b}$$

A 1-layer feedforward neural network (multi-layer perceptron) has the form:

$$\text{MLP}(\mathbf{x}) = g(\mathbf{x}\mathbf{W}_1 + \mathbf{b}_1)\mathbf{W}_2 + \mathbf{b}_2$$

Common Activation Functions (g)

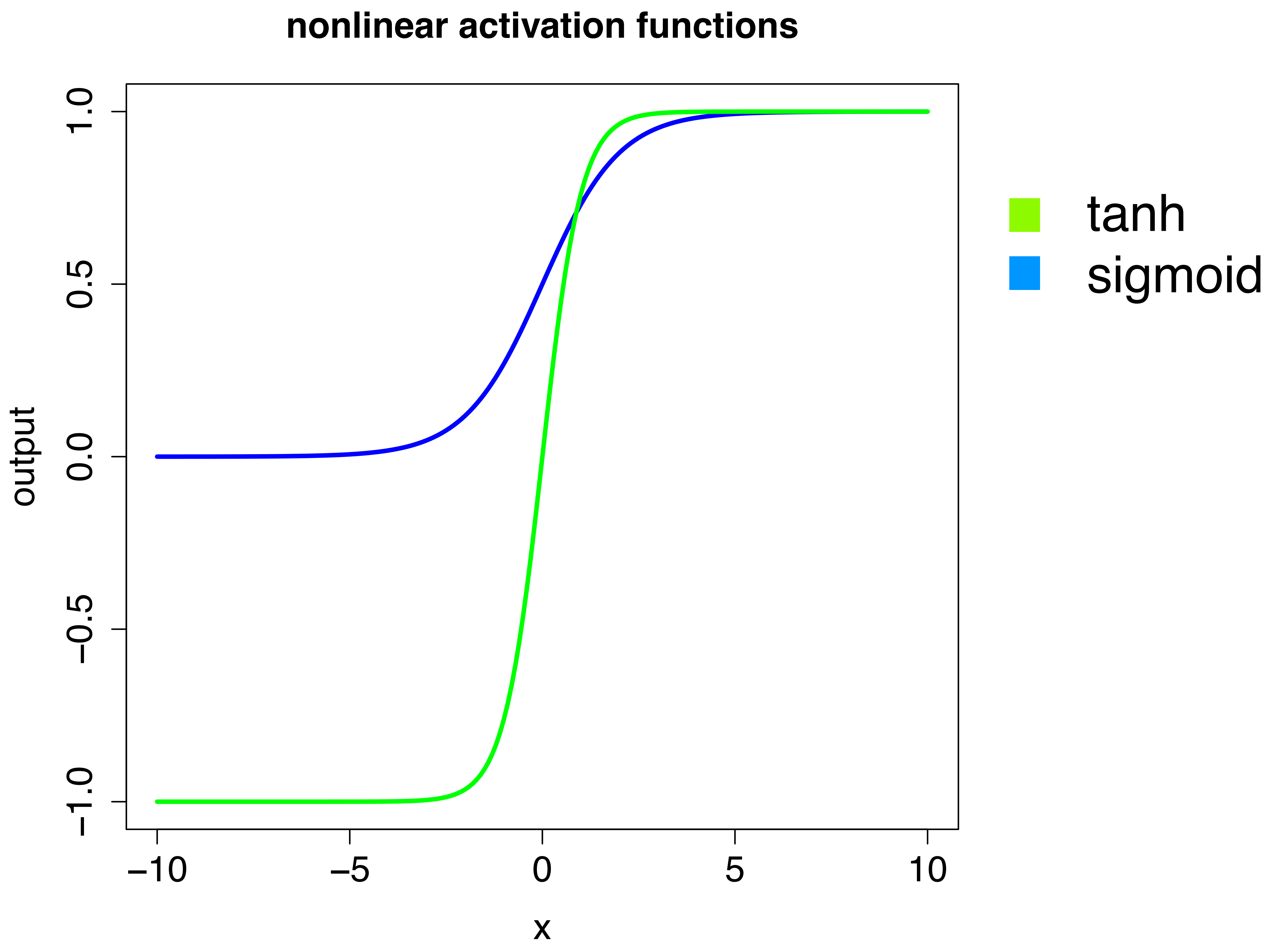
Sigmoid: $\sigma(x) = 1/(1 + e^{-x})$



Common Activation Functions (g)

Sigmoid: $\sigma(x) = 1/(1 + e^{-x})$

Hyperbolic tangent (tanh): $\tanh(x) = (e^{2x} - 1)/(e^{2x} + 1)$

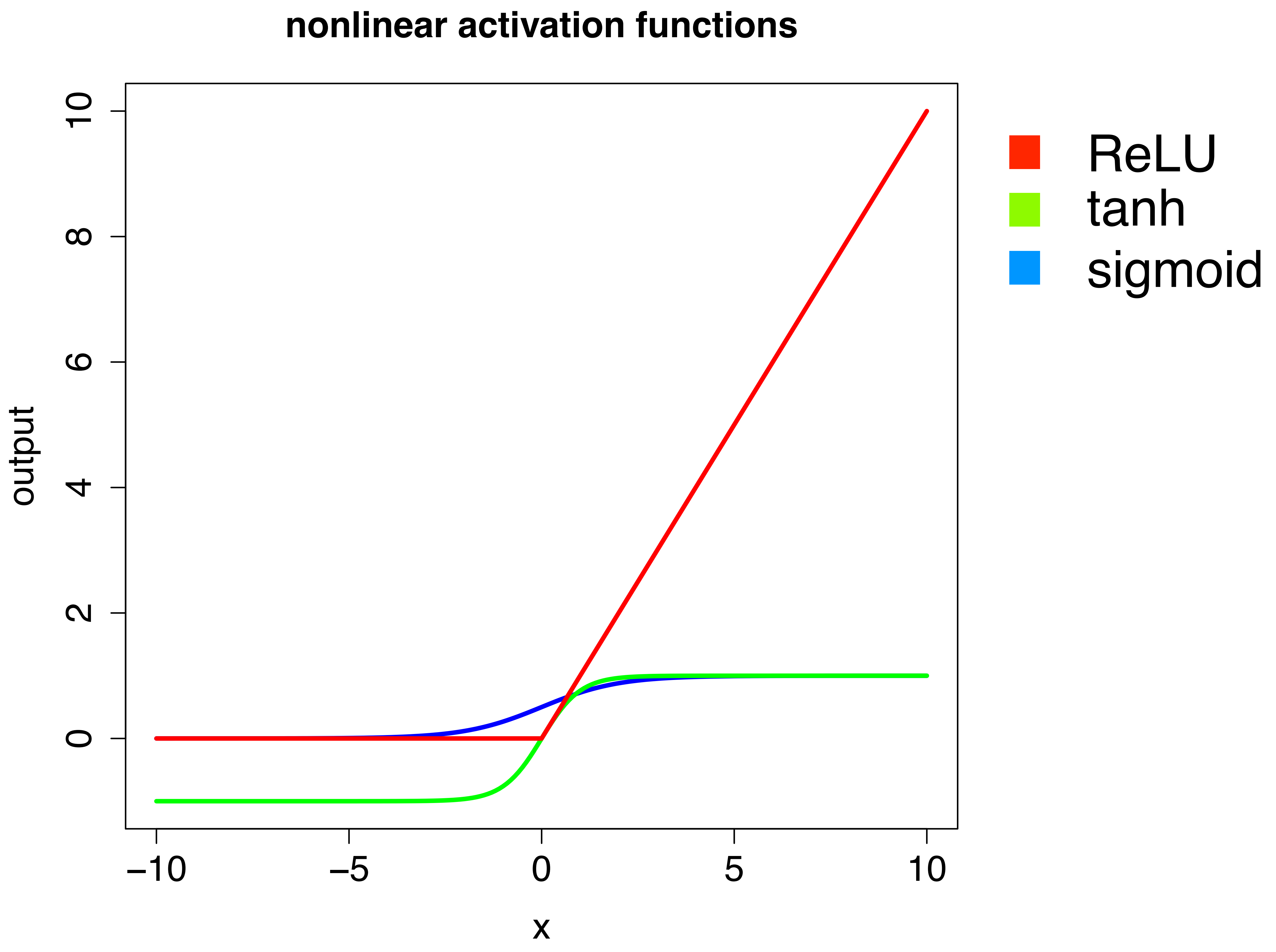


Common Activation Functions (g)

Sigmoid: $\sigma(x) = 1/(1 + e^{-x})$

Hyperbolic tangent (tanh): $\tanh(x) = (e^{2x} - 1)/(e^{2x} + 1)$

Rectified Linear Unit (ReLU): $\mathrm{RELU}(x) = \max(0, x)$



Training Neural Networks

Optimization Problem

- To train a neural network, define a loss function L(y,ỹ):
  a function of the true output y and the predicted output ỹ
- L(y,ỹ) assigns a non-negative numerical score to the neural network's output, ỹ
- The parameters of the network are set to minimise L over the training examples (i.e. a sum of losses over different training samples)
- L is typically minimised using a gradient-based method

Stochastic Gradient Descent (SGD)

SGD Algorithm

Inputs: $\mathrm{NN}(x;\theta)$, Training examples, $x_1 \ldots x_n$; outputs, $y_1 \ldots y_n$ and Loss function L

Randomly initialize θ

do until **stopping criterion**

    Pick a training example $\{x_i, y_i\}$

    Compute the loss $L(\mathrm{NN}(x_i;\theta), y_i)$

    Compute gradient of L, $\nabla_\theta L$ with respect to θ

    $\boxed{\theta \leftarrow \theta - \eta \nabla_\theta L}$ Weight Update Rule

    Learning Rate

done

Return: θ

Mini-batch Gradient Descent (GD)

Mini-batch GD Algorithm

Inputs: $\mathrm{NN}(x; \theta)$, Training examples, $x_1 \ldots x_n$; outputs, $y_1 \ldots y_n$ and Loss function L

Randomly initialize θ
do until **stopping criterion**
    Randomly sample a batch of training examples $\{x_i, y_i\}_{i=1}^{b}$
        (where the batch size, b, is a hyperparameter)
    Compute gradient of L over the batch, $\nabla_\theta L$ with respect to θ
    $\theta \leftarrow \theta - \eta \nabla_\theta L$
done

Return: θ

Loss Function

Overall loss function, $J(\theta)$, measures the total loss over the entire training set:

$$J(\theta) = \frac{1}{N} \sum_{i=1}^{N} L(\mathrm{NN}(\mathbf{x}_i; \theta), y_i)$$

Cross-entropy loss is one of the most popular classification-based loss functions. Assuming $\mathrm{NN}(\mathbf{x}_i; \theta)$ returns a probability, binary cross-entropy can be defined as:

$$J(\theta) = \frac{1}{N} \sum_{i=1}^{N} y_i \log\left(\mathrm{NN}(\mathbf{x}_i; \theta)\right) + (1 - y_i) \log\left(1 - \mathrm{NN}(\mathbf{x}_i; \theta)\right)$$

Training a Neural Network

Define the Loss function to be minimised as a node L

Goal: Learn weights for the neural network which minimise L

Gradient Descent: Find $\partial L/\partial w$ for every weight w, and update it as $w \leftarrow w - \eta\ \partial L/\ \partial w$

How do we efficiently compute $\partial L/\partial w$ for all w?

Will compute $\partial L/\partial u$ for every node u in the network!

$\partial L/\partial w = \partial L/\partial u \cdot \partial u/\partial w$ where u is the node which uses w

Training a Neural Network

New goal: compute $\partial L/\partial u$ for every node u in the network

Simple algorithm: Backpropagation

Key fact: Chain rule of differentiation

If L can be written as a function of variables $v_1,\ldots, v_n$, which in turn depend (partially) on another variable u, then

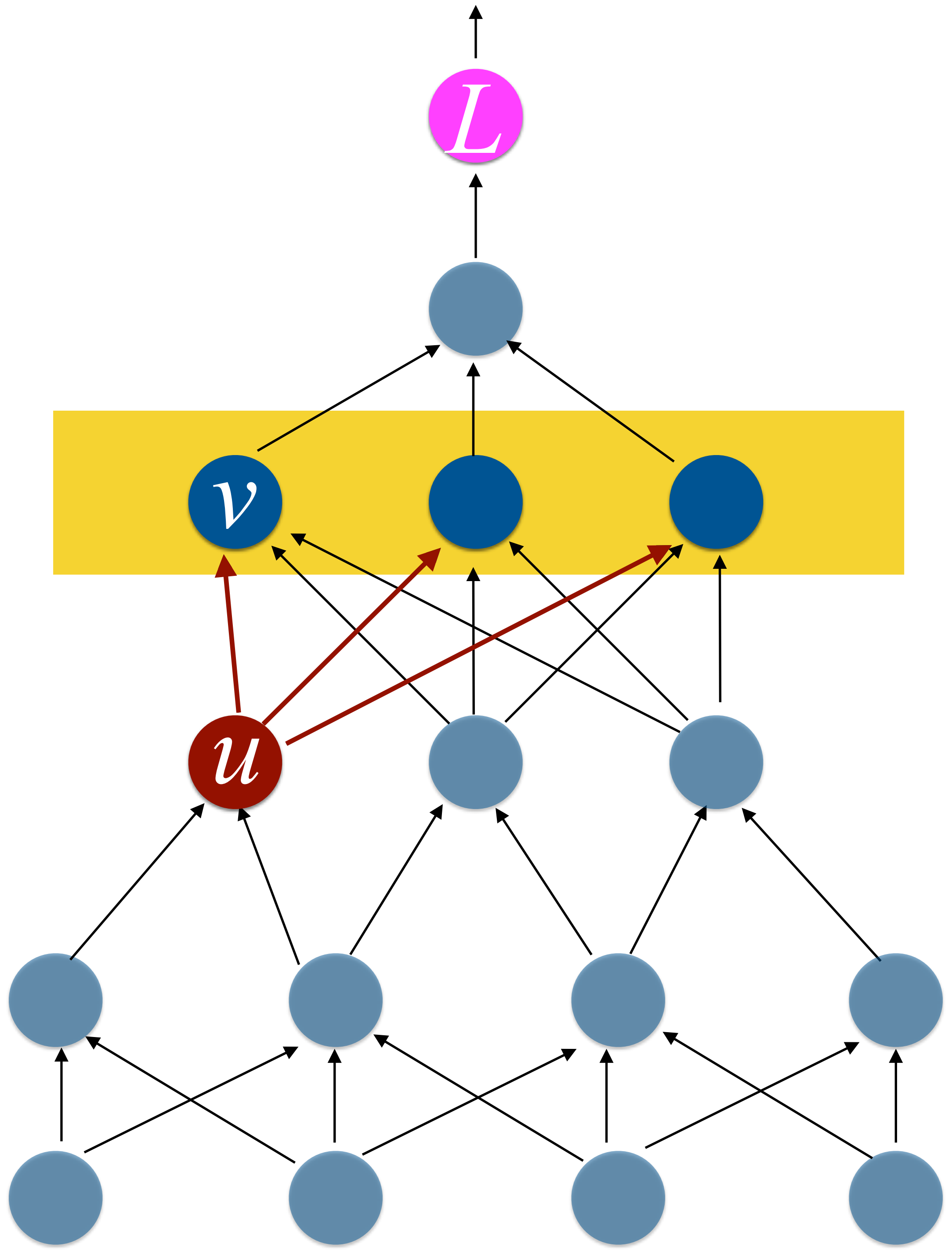
$$\partial L/\partial u = \Sigma_i \, \partial L/\partial v_i \cdot \partial v_i/\partial u$$

Backpropagation

If L can be written as a function of variables $v_1,\ldots, v_n$, which in turn depend (partially) on another variable u, then

$$\partial L/\partial u = \Sigma_i\, \partial L/\partial v_i \cdot \partial v_i/\partial u$$

Consider $v_1,\ldots, v_n$ as the layer above u, $\Gamma(u)$



Then, the chain rule gives

$$\partial L/\partial u = \Sigma_{v \in \Gamma(u)}\, \partial L/\partial v \cdot \partial v/\partial u$$

Backpropagation

$$\partial L/\partial u = \Sigma_{v \in \Gamma(u)} \; \partial L/\partial v \cdot \partial v/\partial u$$

<u>Backpropagation</u>

Base case: $\partial L/\partial L = 1$

For each u (top to bottom):

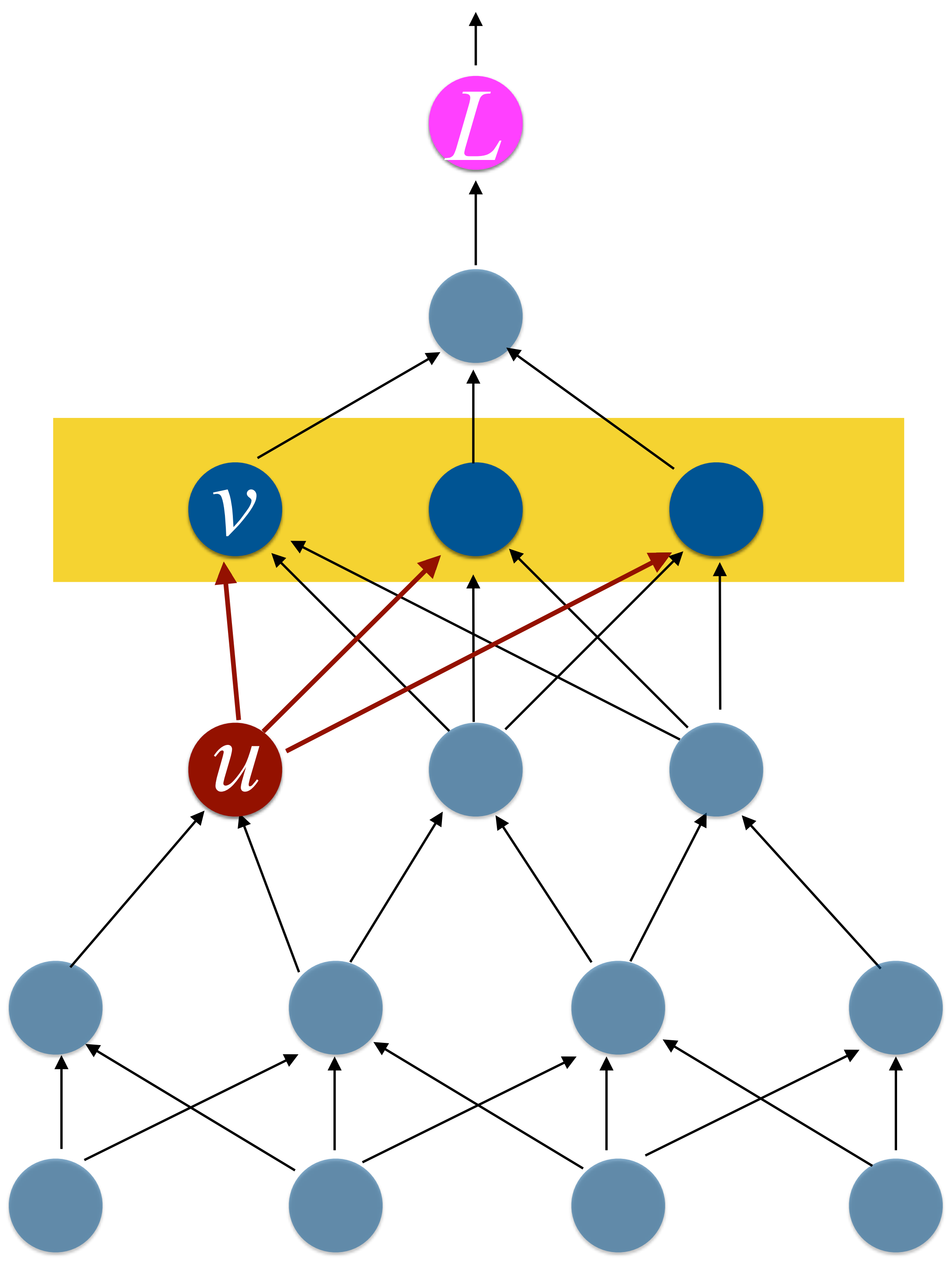
For each $v \in \Gamma(u)$:

Inductively, have computed $\partial L/\partial v$

Directly compute $\partial v/\partial u$

Compute $\partial L/\partial u$

Compute $\partial L/\partial w$ where $\partial L/\partial w = \partial L/\partial u \cdot \partial u/\partial w$



<u>Forward Pass</u>

First, in a forward pass, compute values of all nodes given an input

(The values of each node will be needed during backprop)

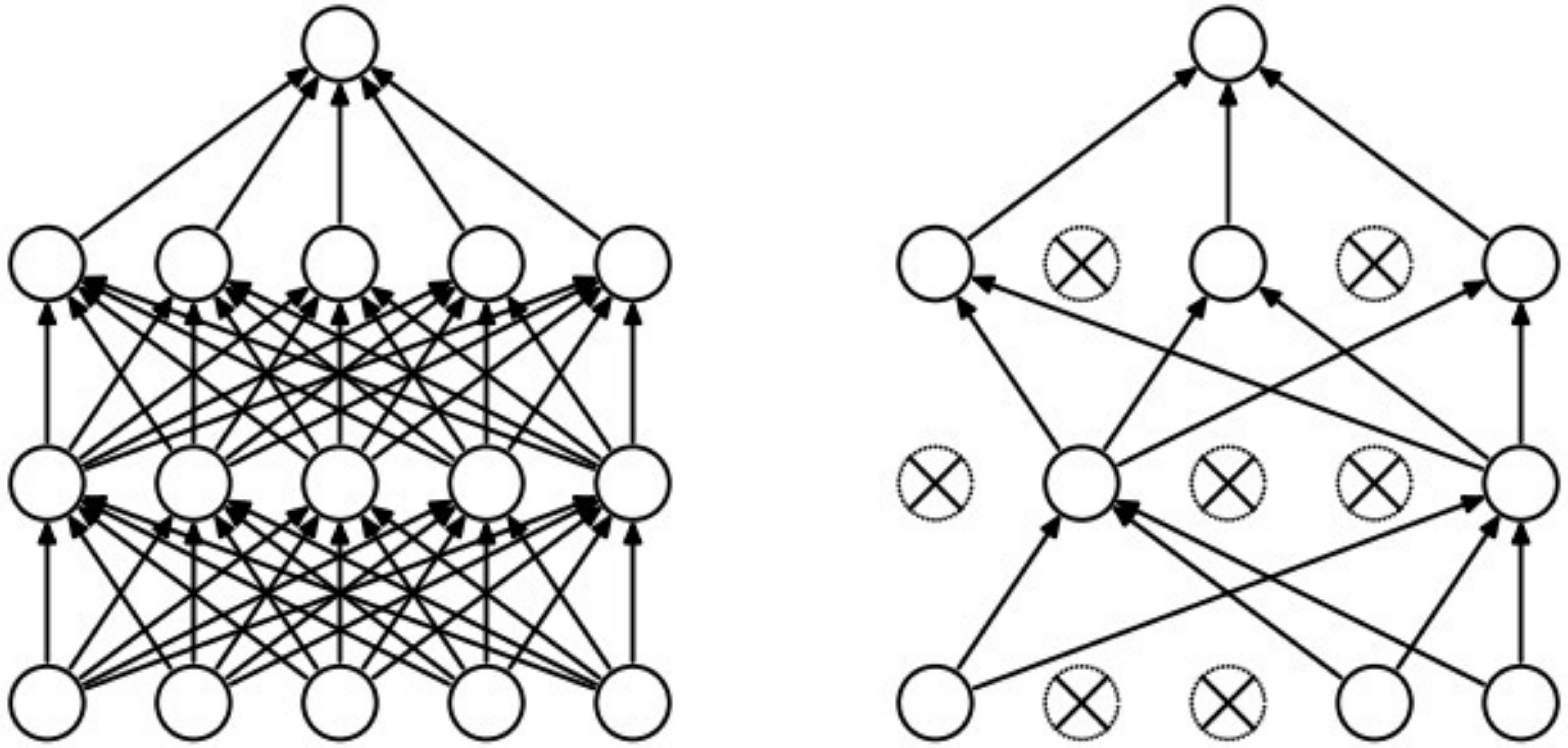
Where values computed in the forward pass may be needed

Tricks of the Trade

Regularization

L2 regularization: Introduce a loss term that penalizes the squared magnitude of all parameters. That is, for every weight w in the network, add the term λw^2 to the objective.

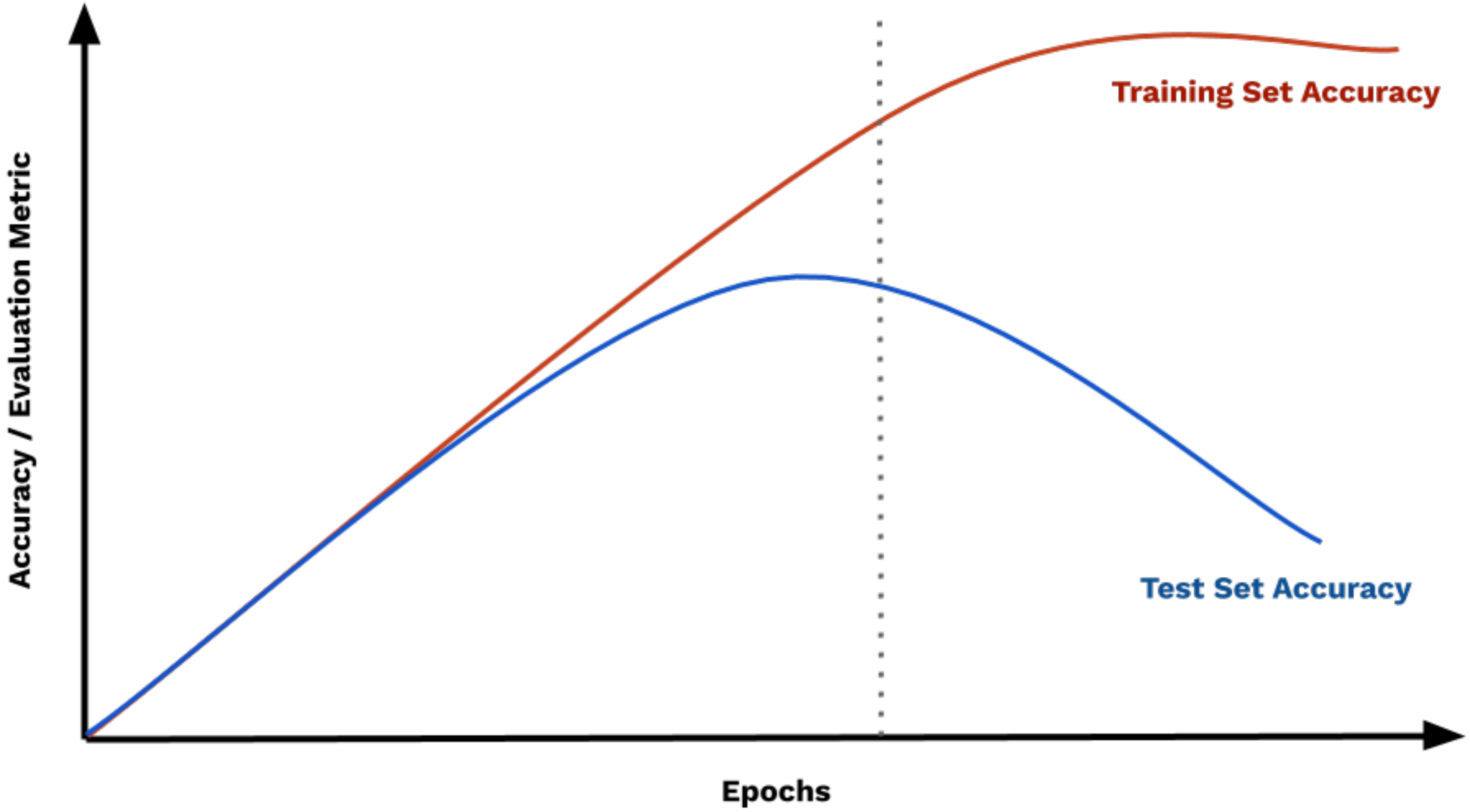
Dropout: During training, keep a neuron active with a (keep) probability of p or set it to 0 otherwise.



(a) Standard Neural Net (b) After applying dropout.

Regularization

Early stopping: Stop training when performance on a validation set has stopped improving



Double descent

Early stopping may not always be an effective strategy considering the “double descent” phenomenon [1]

[1] Belkin et al., https://arxiv.org/abs/1812.11118
Image from: https://openai.com/blog/deep-double-descent/

Learning Rate Schedule

- Observe training losses to understand the effect of different learning rates

- Helpful to decay the learning rate over time. E.g. step decay, exponential decay, etc.

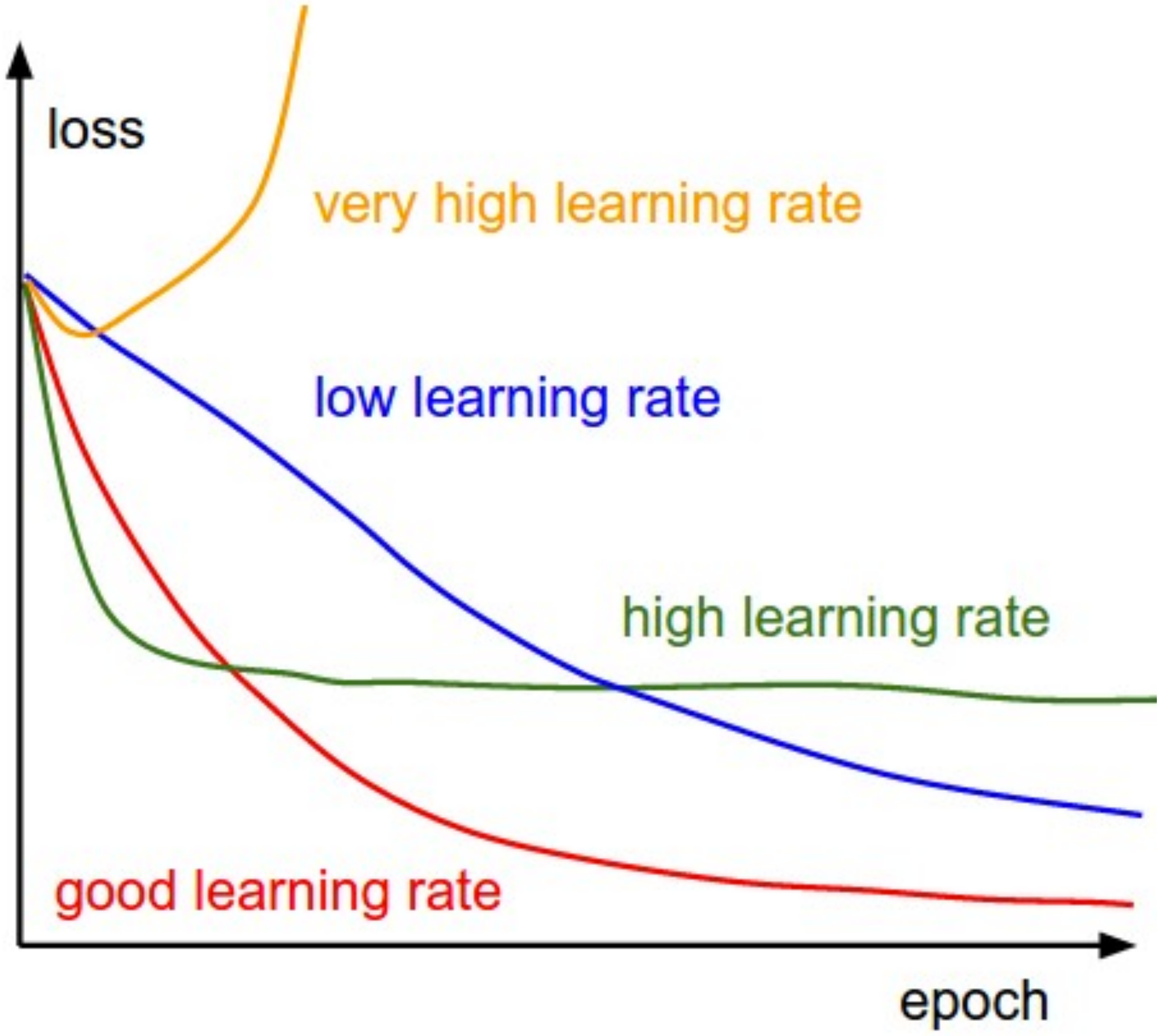


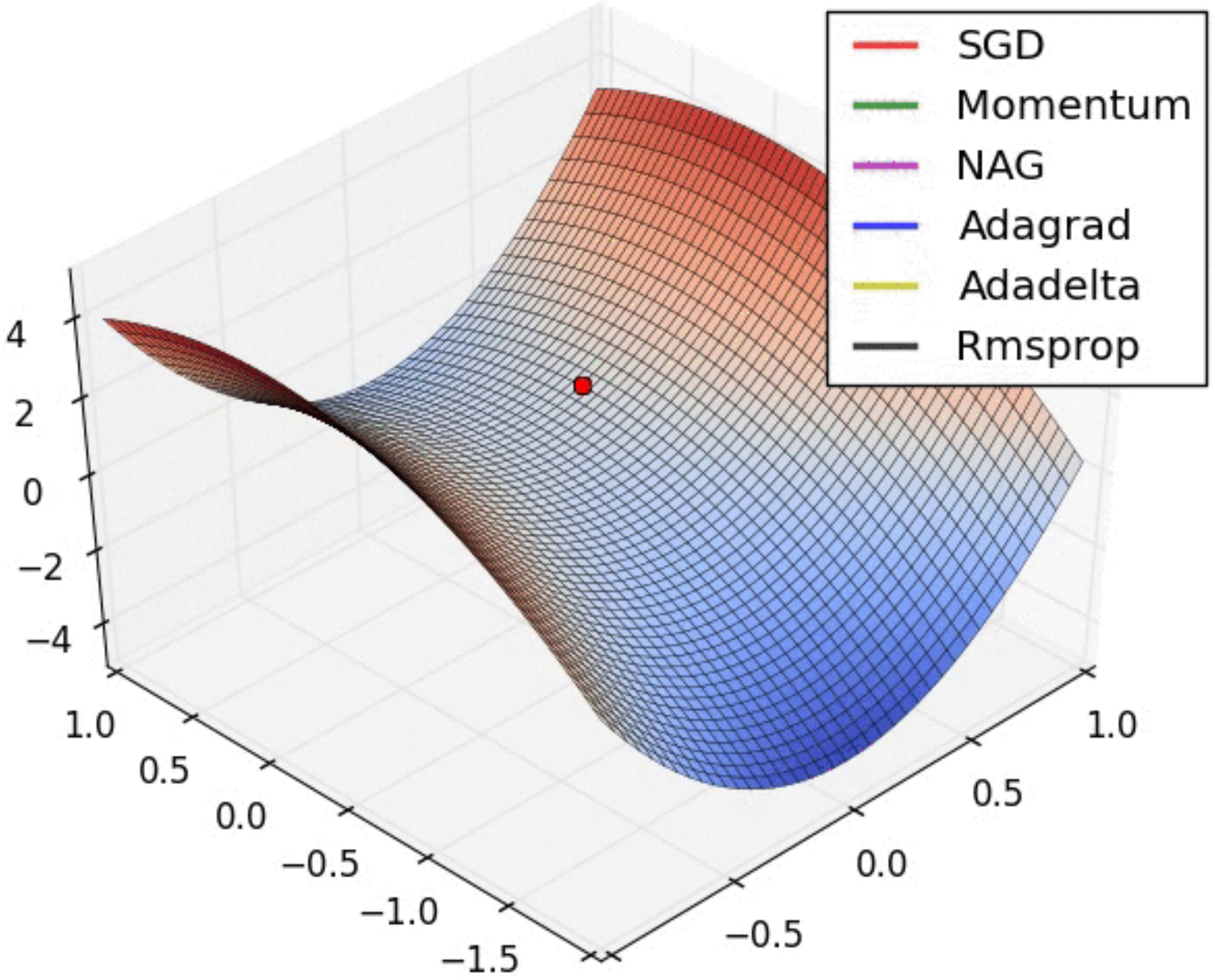
Image from: http://cs231n.github.io/neural-networks-3/

Learning Rate Schedule

- Observe training losses to understand the effect of different learning rates

- Helpful to decay the learning rate over time. E.g. step decay, exponential decay, etc.

- Adaptive learning rate methods like Adagrad, Adam are popular optimizers.



Animation from: http://cs231n.github.io/neural-networks-3/

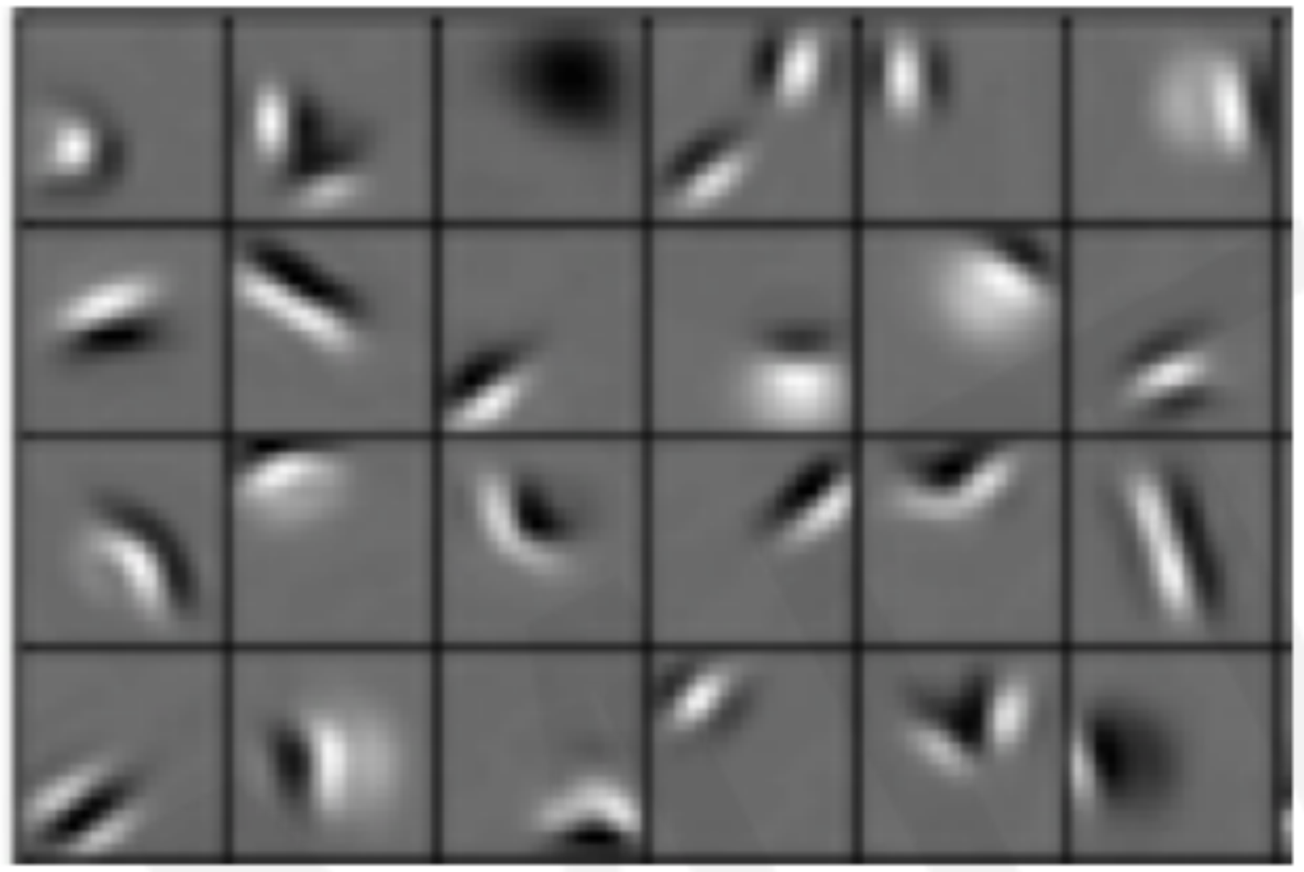
Good reference for optimizers: https://ruder.io/optimizing-gradient-descent/

Convolutional Neural Networks

What is in this image?

Windows, porch, steps, door, etc.

Learning Features



Instead of manually deriving features, can we learn features or representations directly from the data?

Convolutional Neural Networks (CNNs)

- Fully connected (dense) layers have no awareness of spatial information
- Key concept behind convolutional layers is that of ***kernels*** or ***filters***
- Filters slide across an input space to detect spatial patterns (translation invariance) in local regions (locality)

Fully Connected Layers

32x32x3 image -> stretch to 3072 x 1

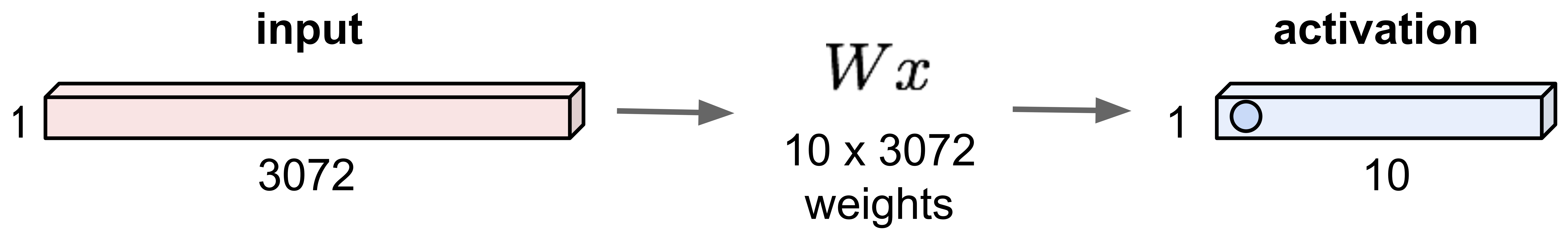
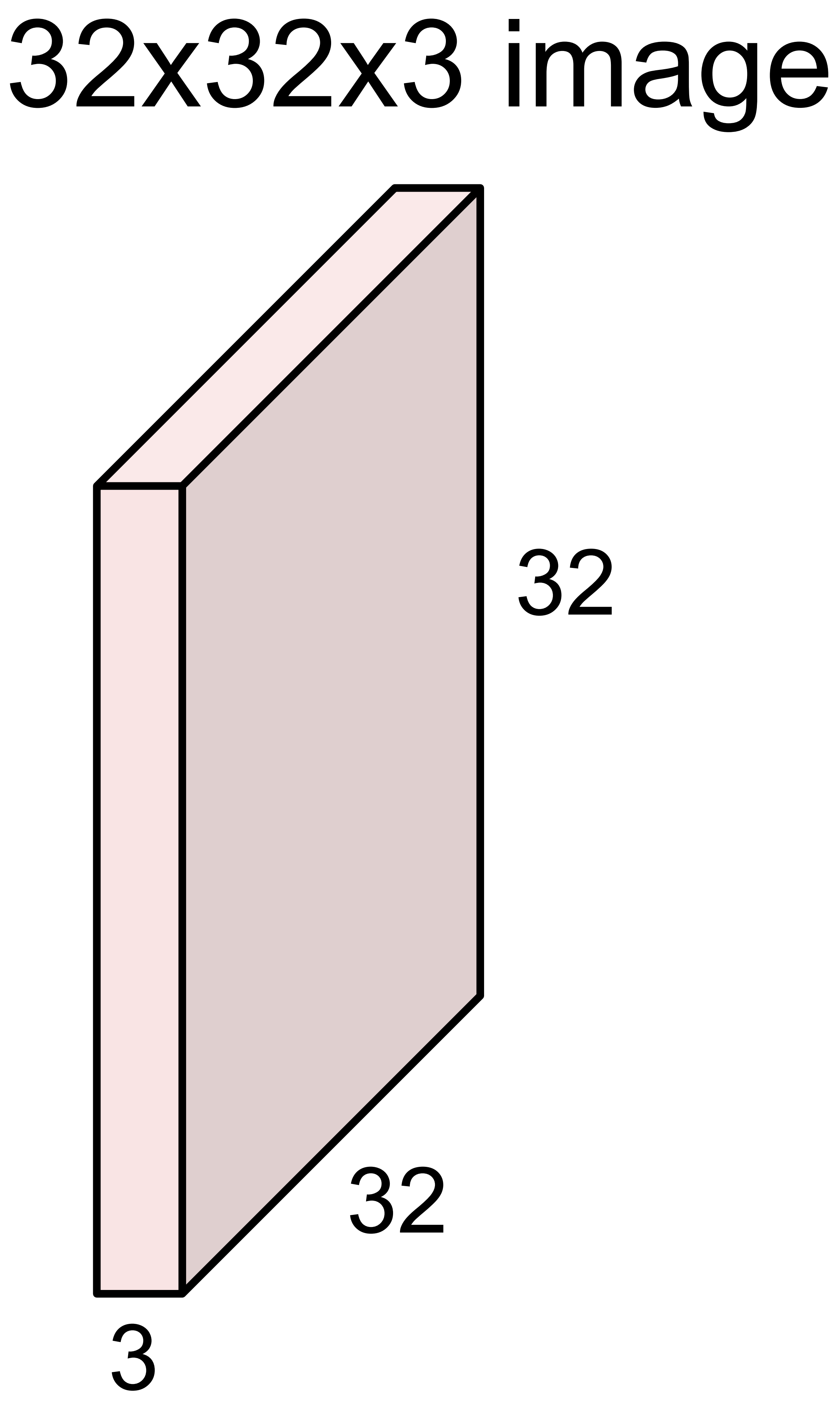


Image from:http://cs231n.stanford.edu/slides/2019/cs231n_2019_lecture05.pdf

Convolution Layer



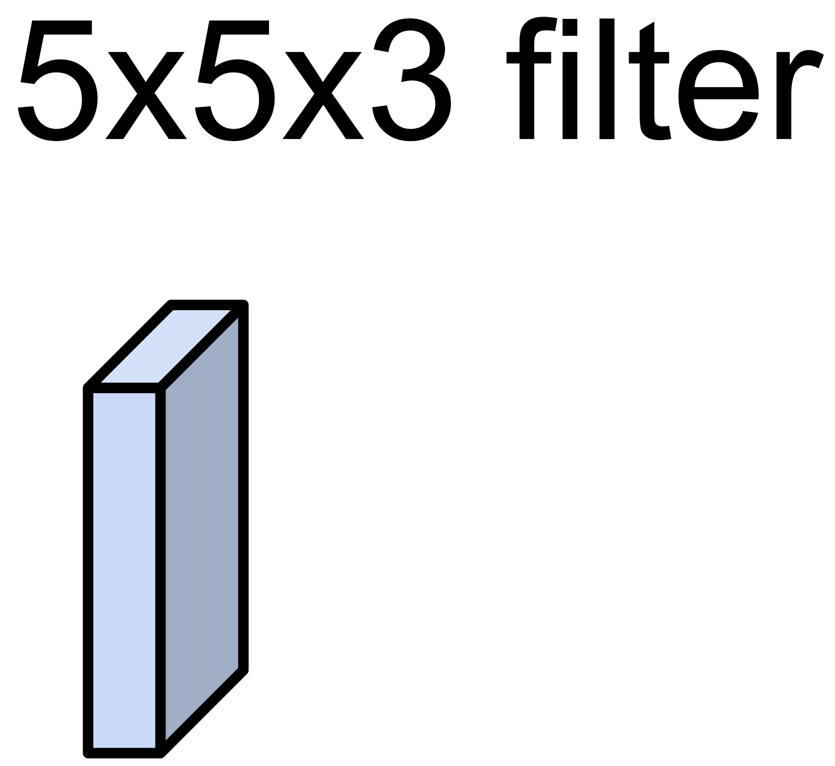
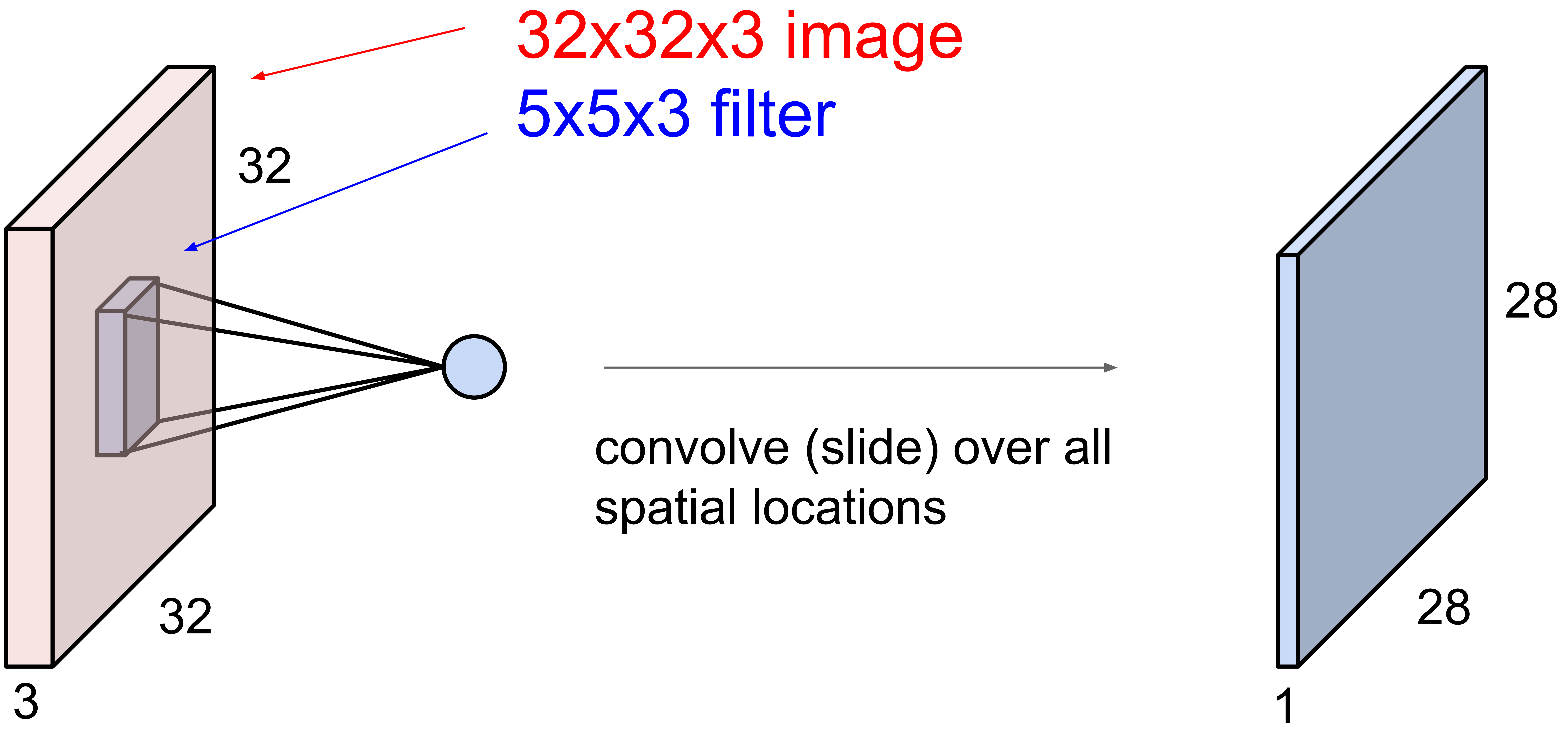


Image from:http://cs231n.stanford.edu/slides/2019/cs231n_2019_lecture05.pdf

Convolution Layer



Convolution Layer

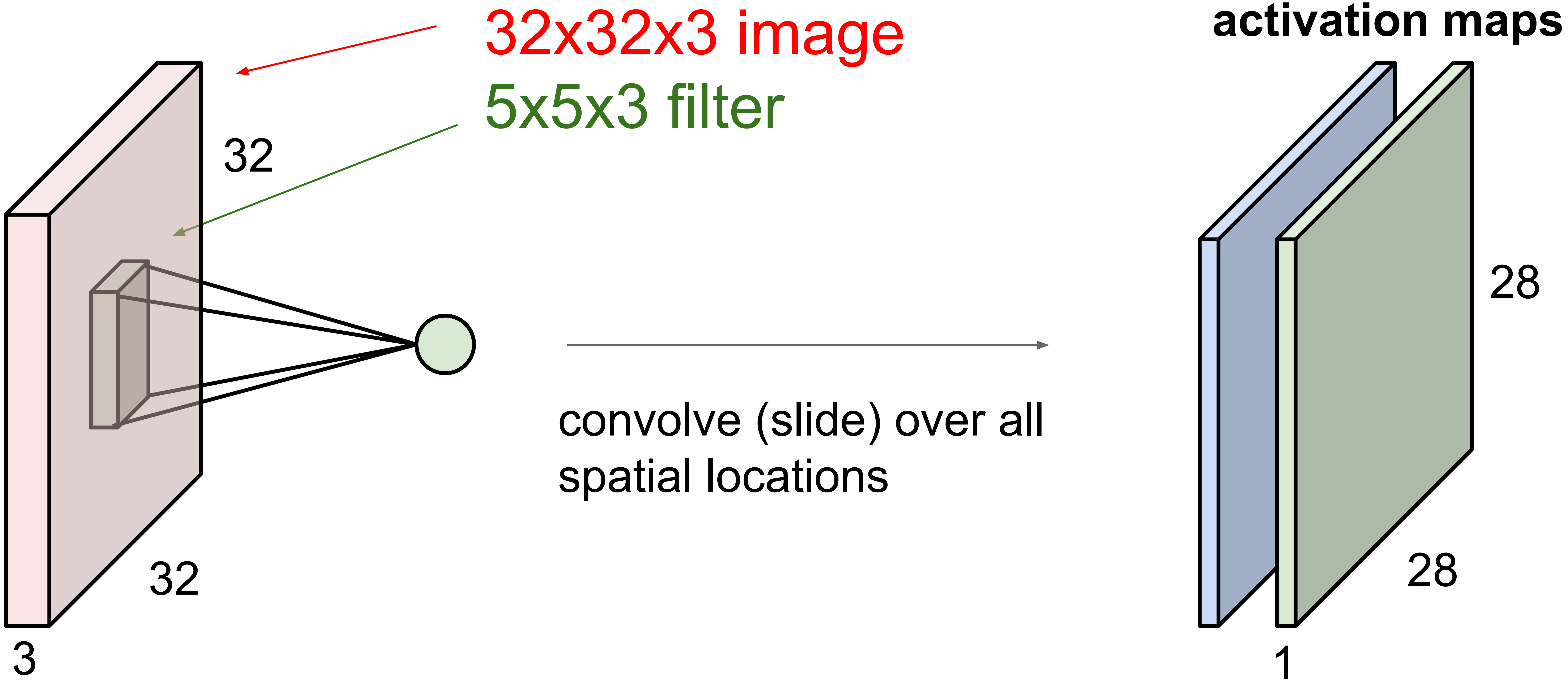


Image from:http://cs231n.stanford.edu/slides/2019/cs231n_2019_lecture05.pdf

Convolution Layer

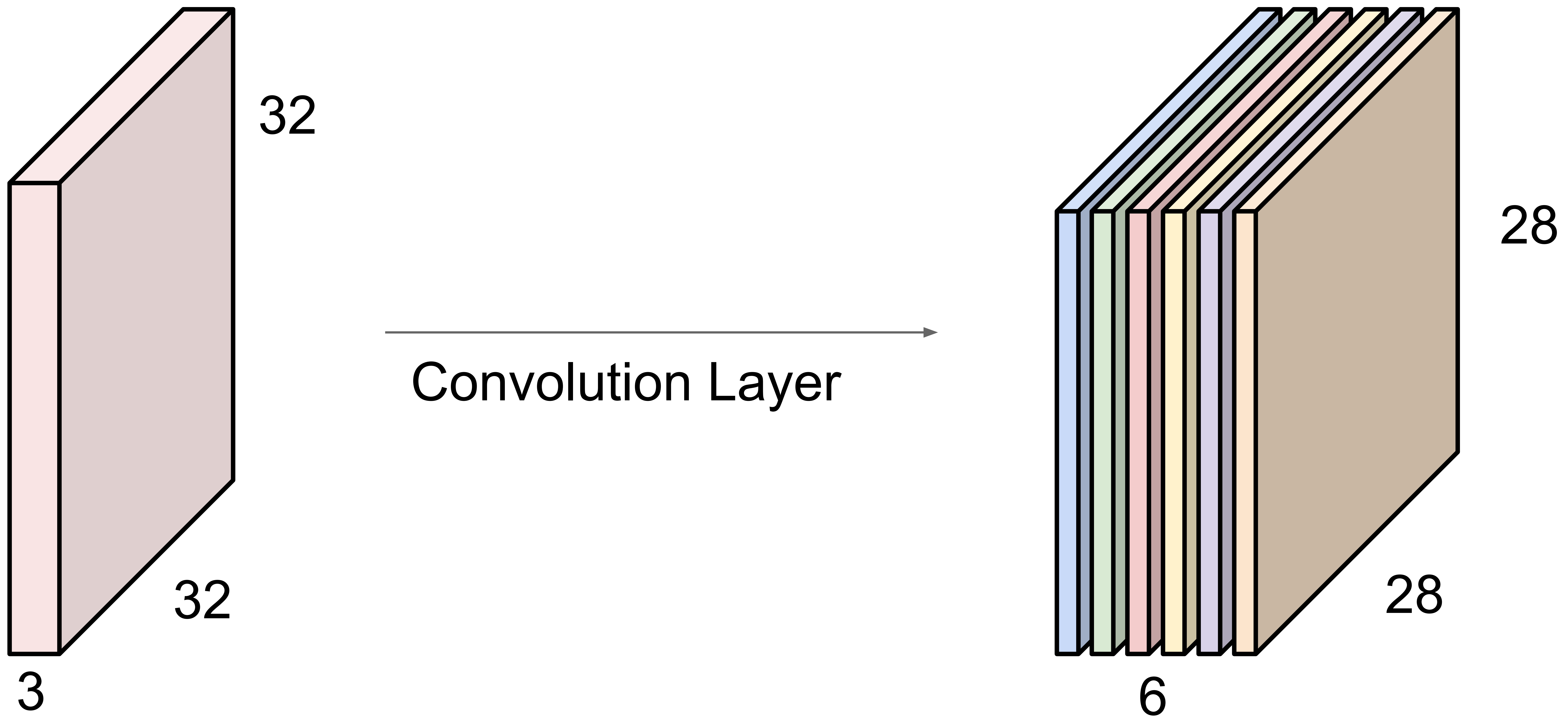


Image from:http://cs231n.stanford.edu/slides/2019/cs231n_2019_lecture05.pdf

Convolutional Neural Network

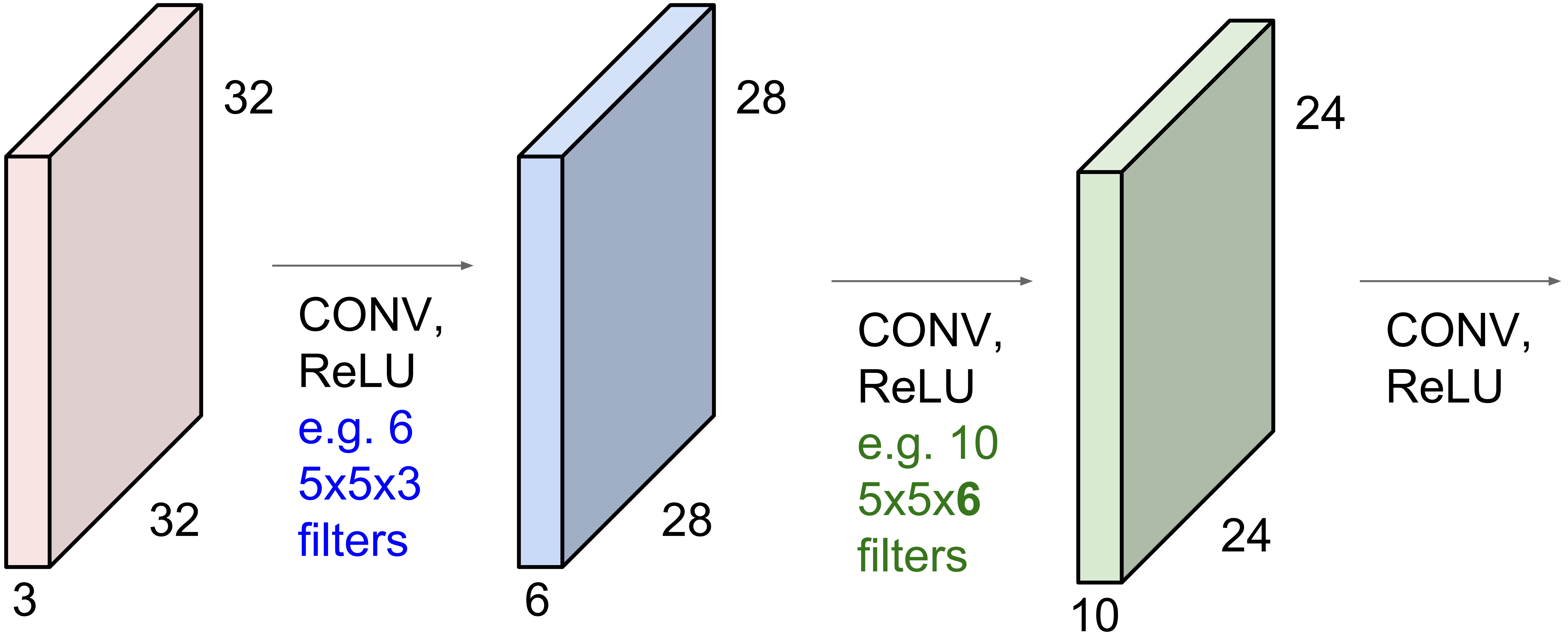


Image from:http://cs231n.stanford.edu/slides/2019/cs231n_2019_lecture05.pdf

What do these layers learn?

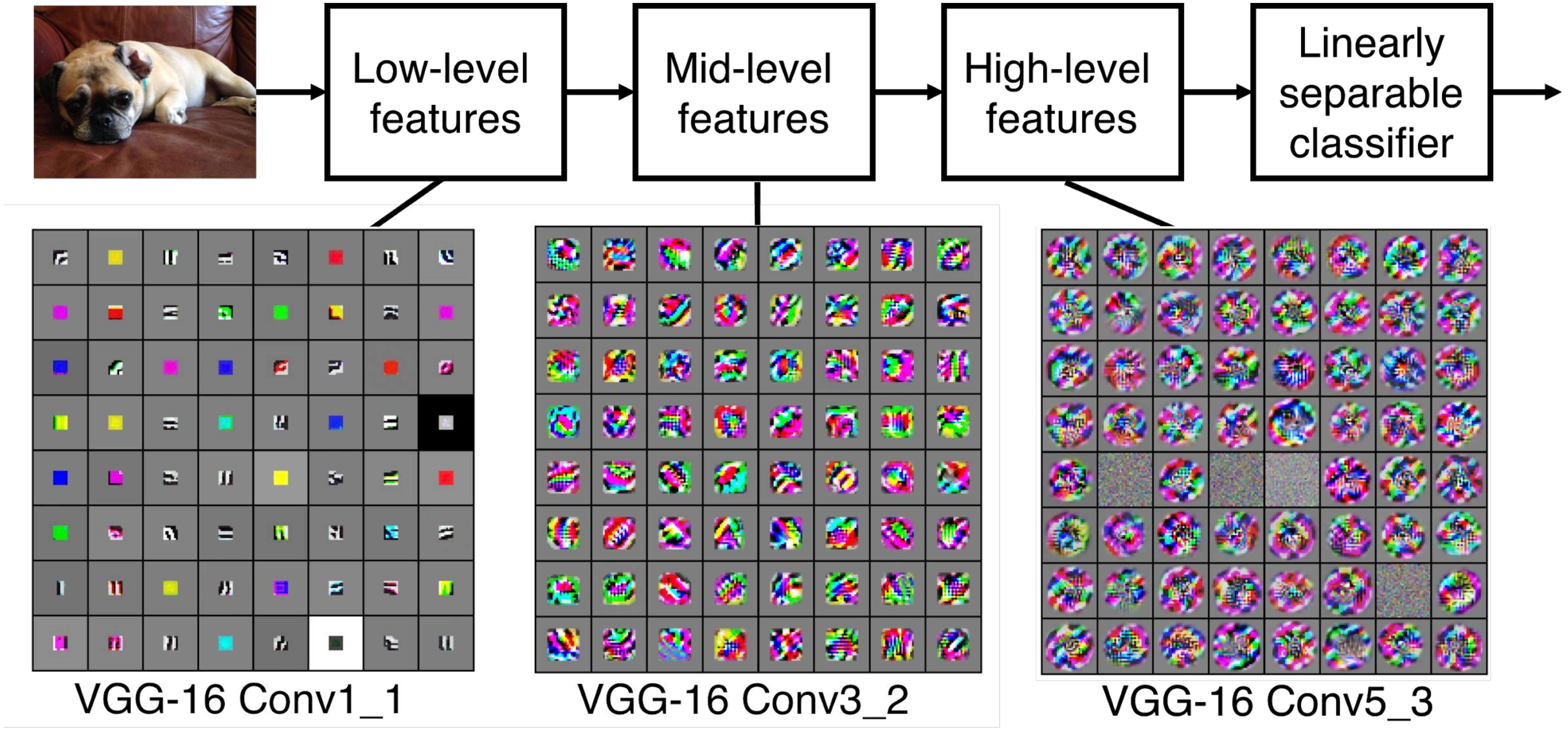
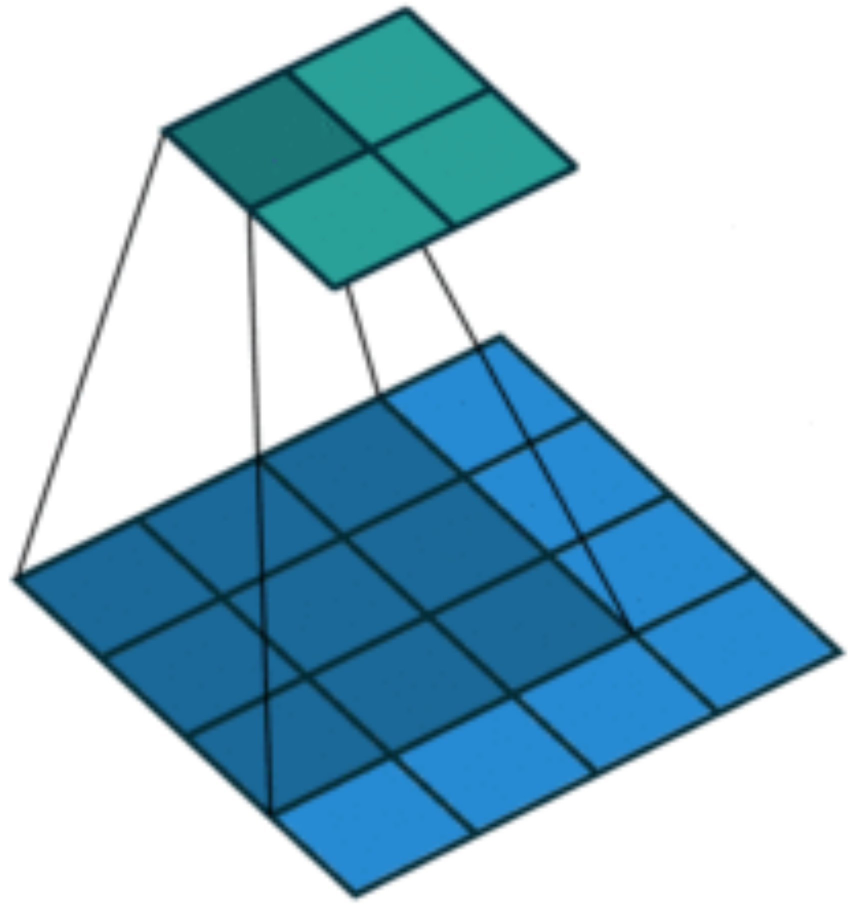
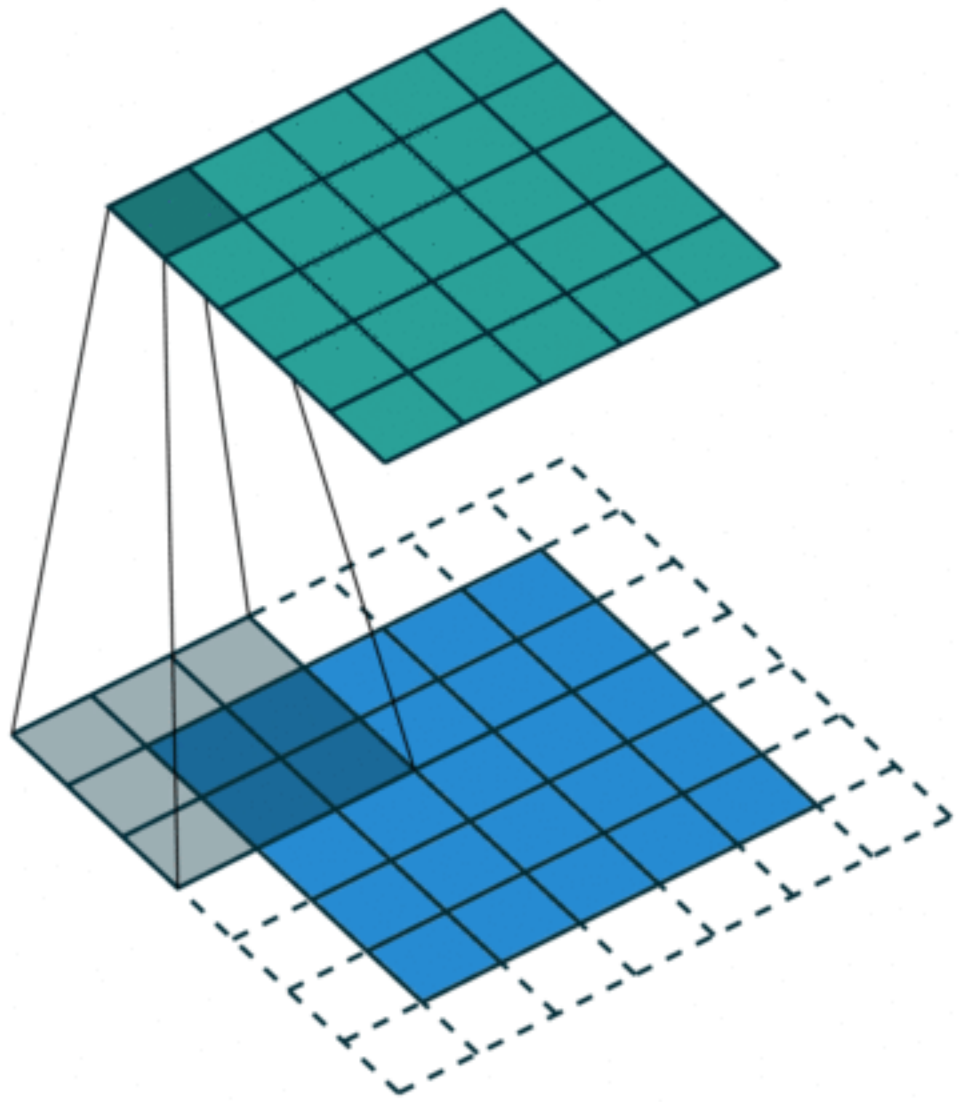


Image from: Simonyan and Zisserman, 2014

Convolutional Neural Networks (CNNs)



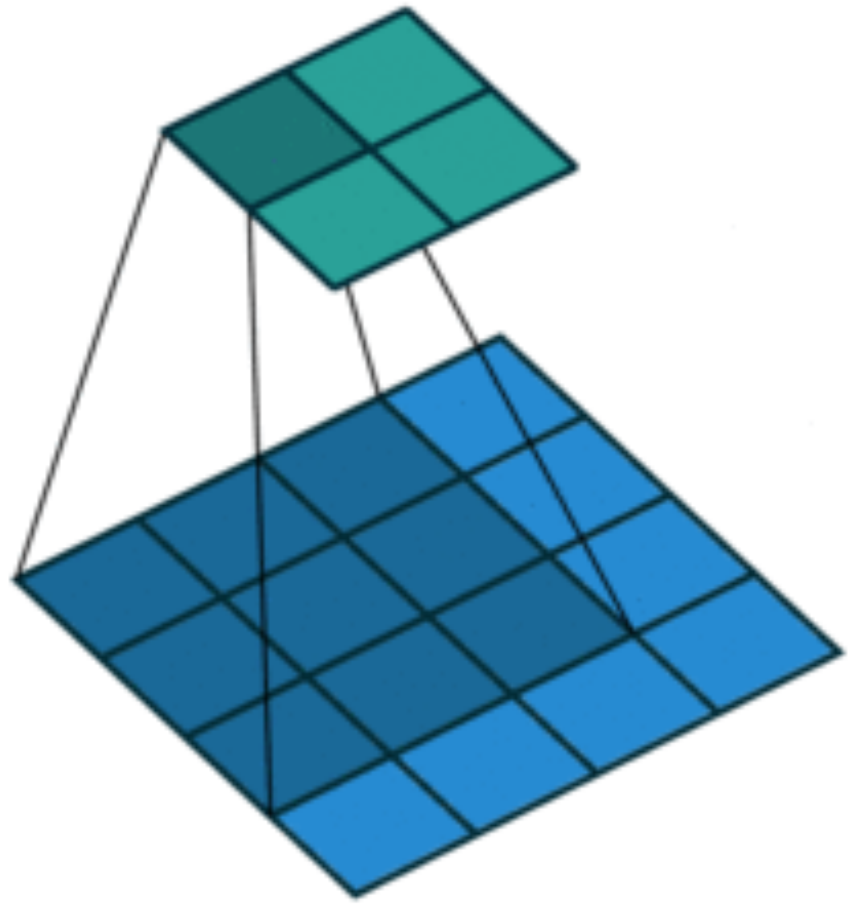
Stride=1, No padding



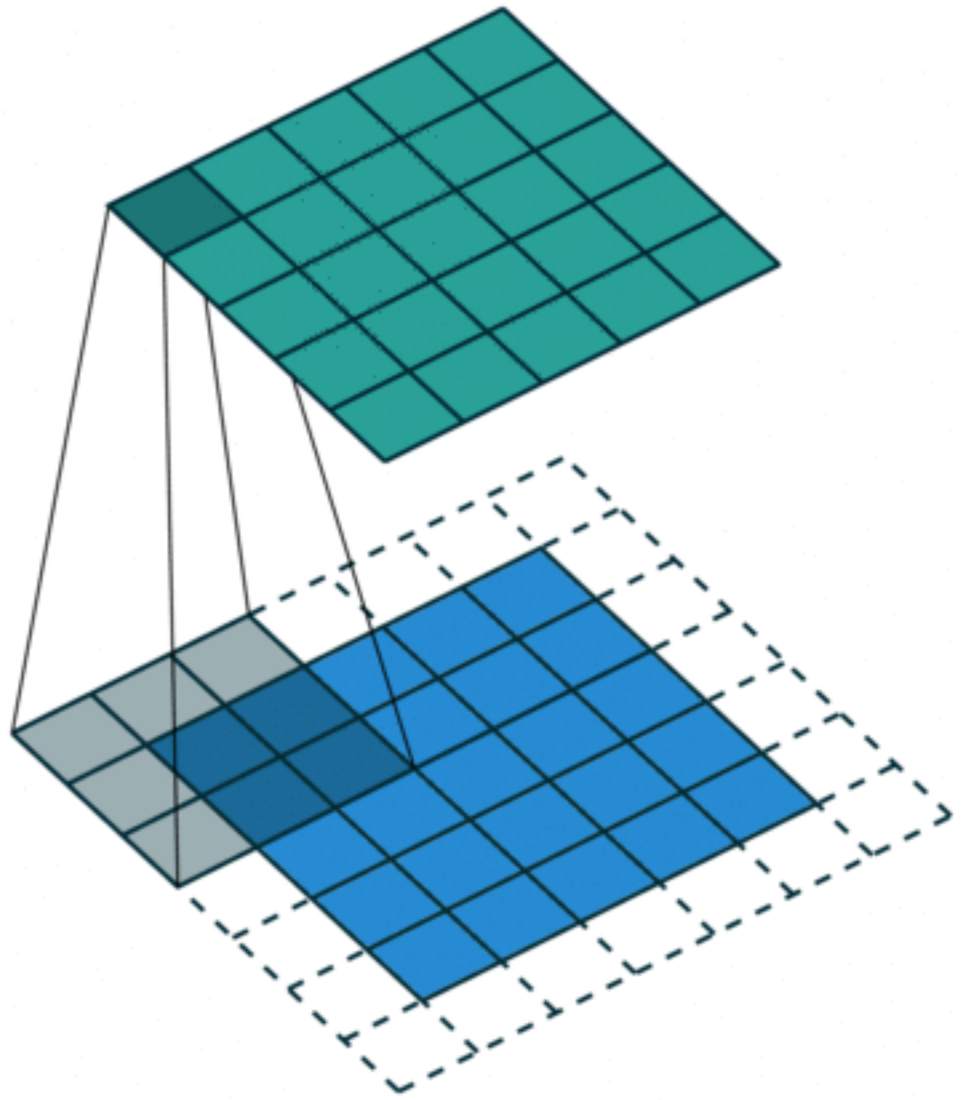
Stride=1, Padding, P=1

All animations are from: https://github.com/vdumoulin/conv_arithmetic

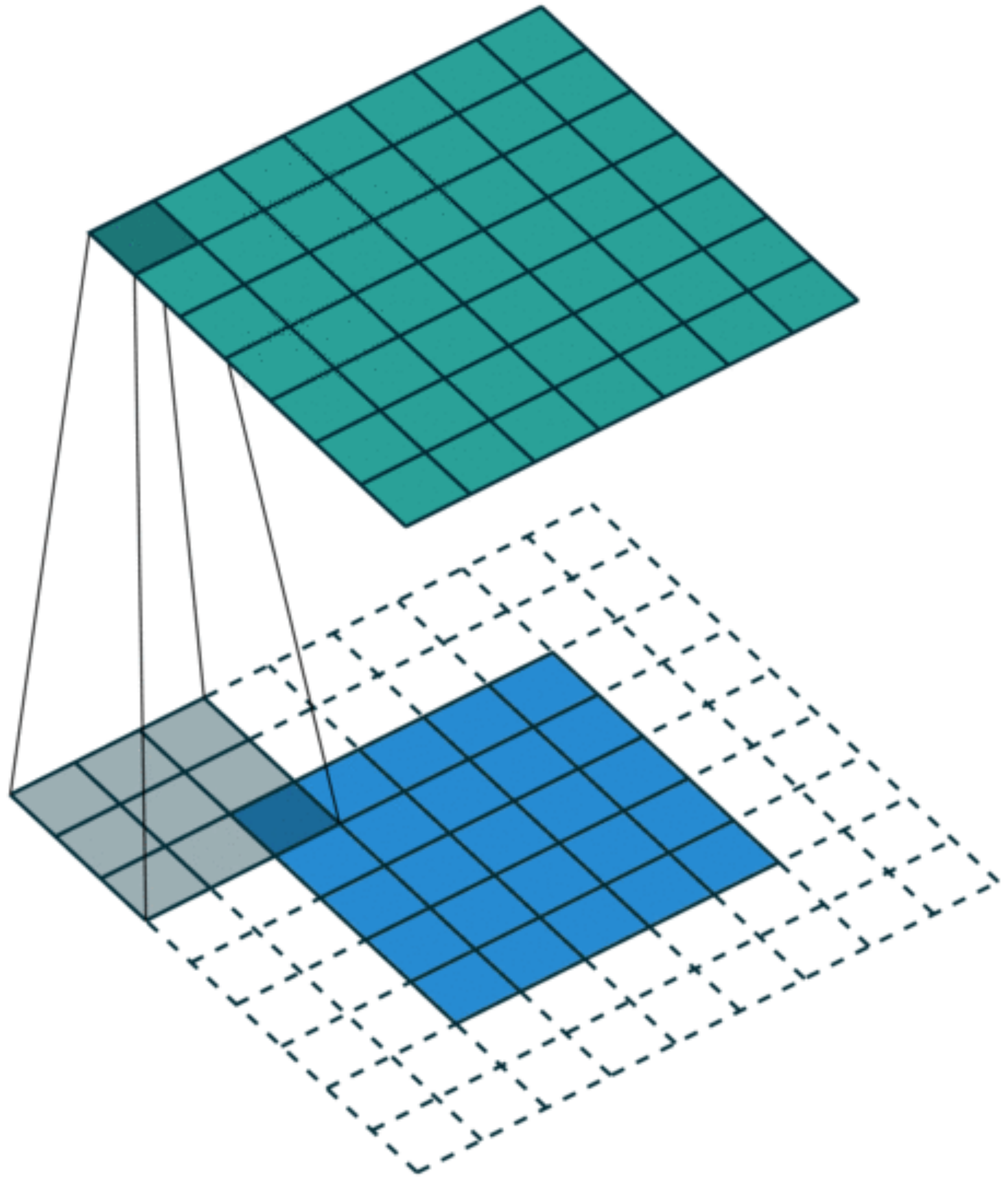
Convolutional Neural Networks (CNNs)



Stride=1, No padding



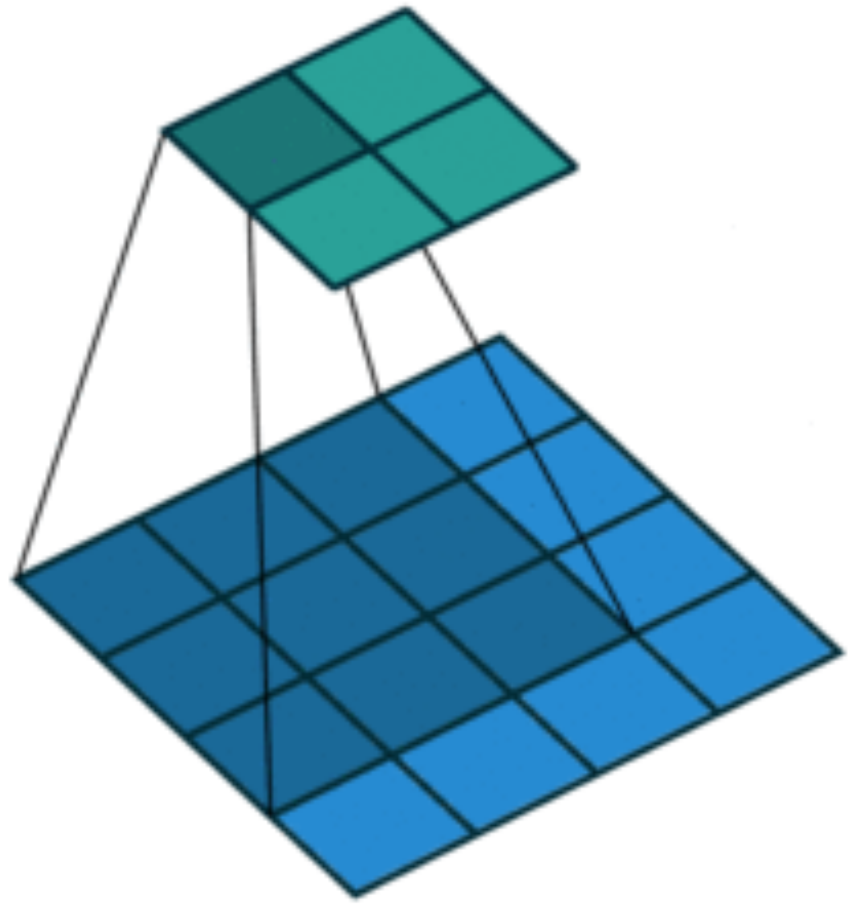
Stride=1, Padding, P=1



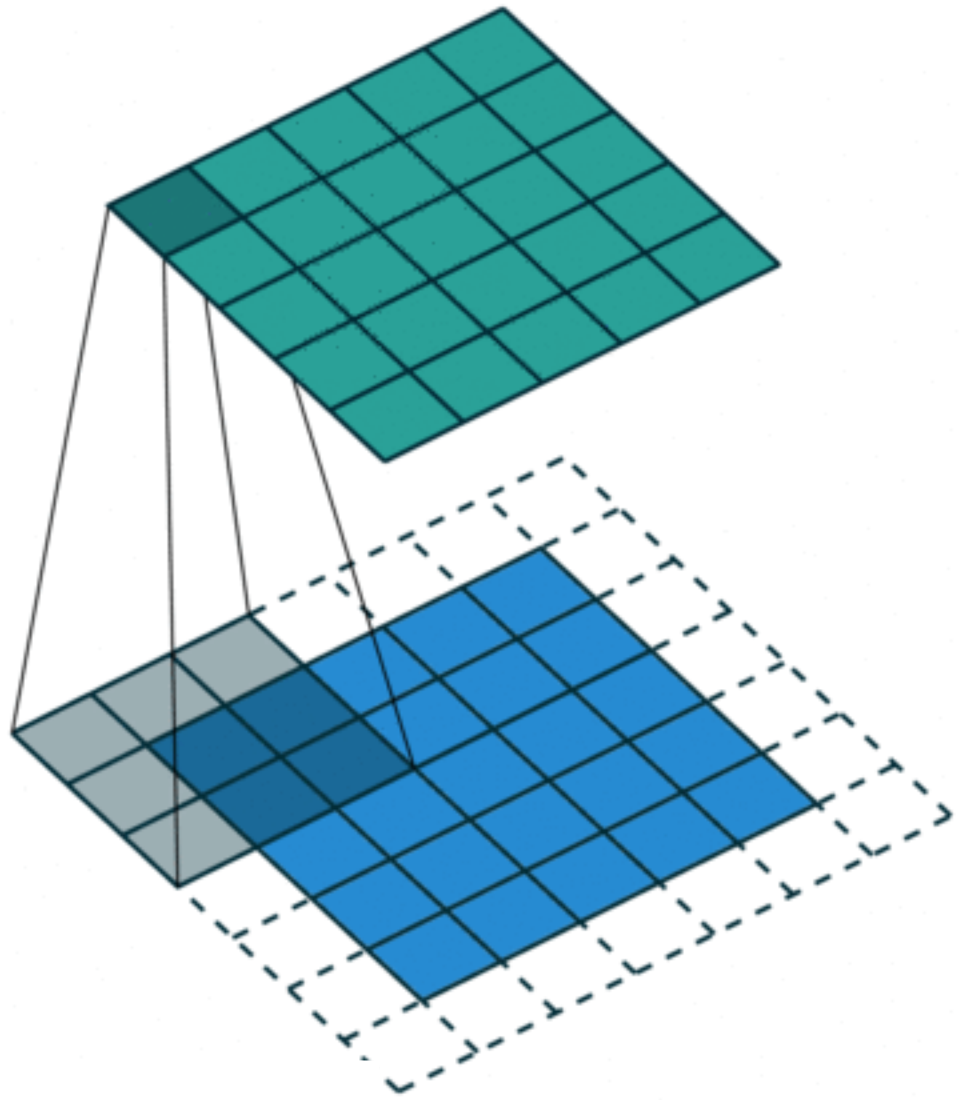
Stride=1, Padding, P=2

All animations are from: https://github.com/vdumoulin/conv_arithmetic

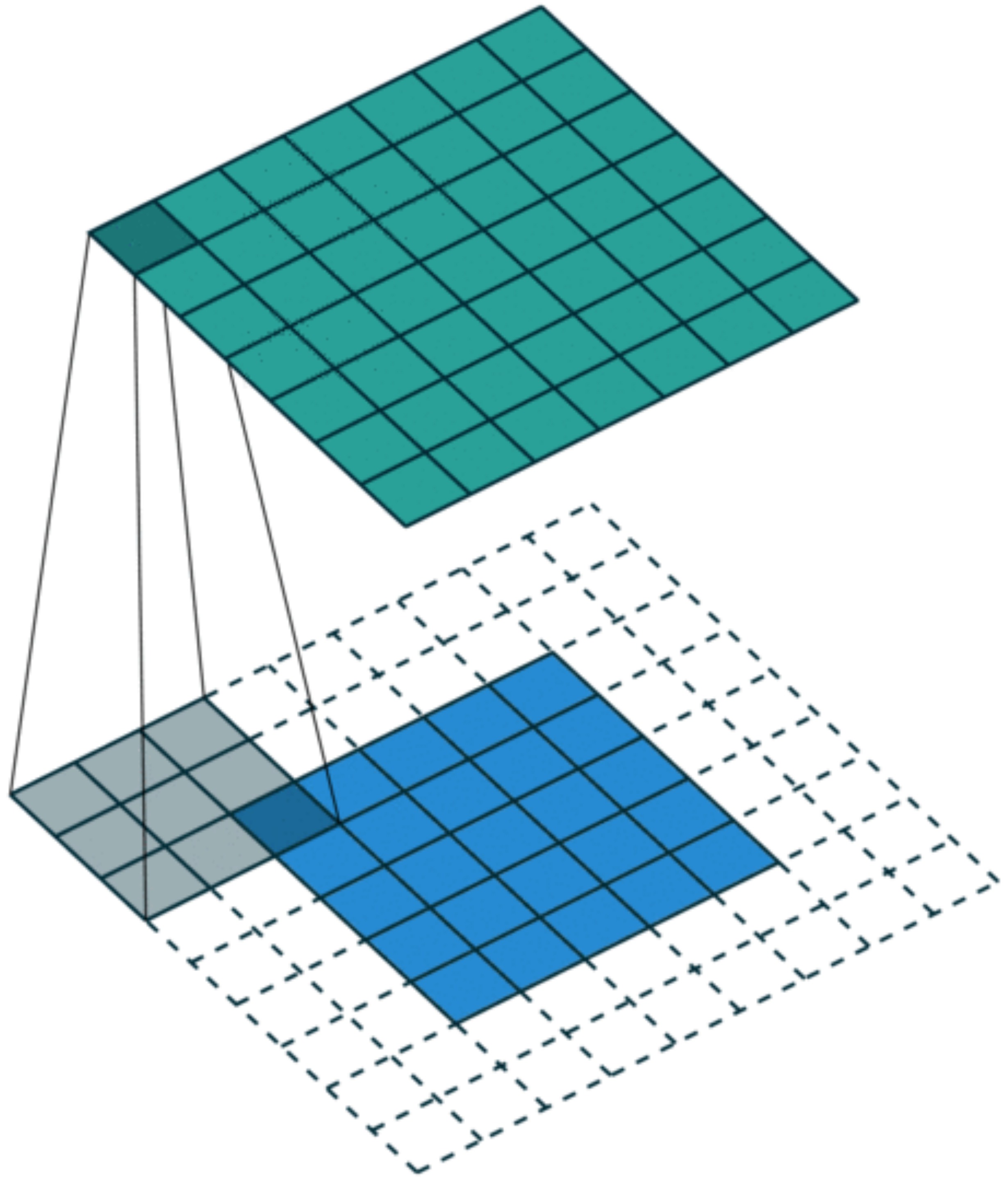
Convolutional Neural Networks (CNNs)



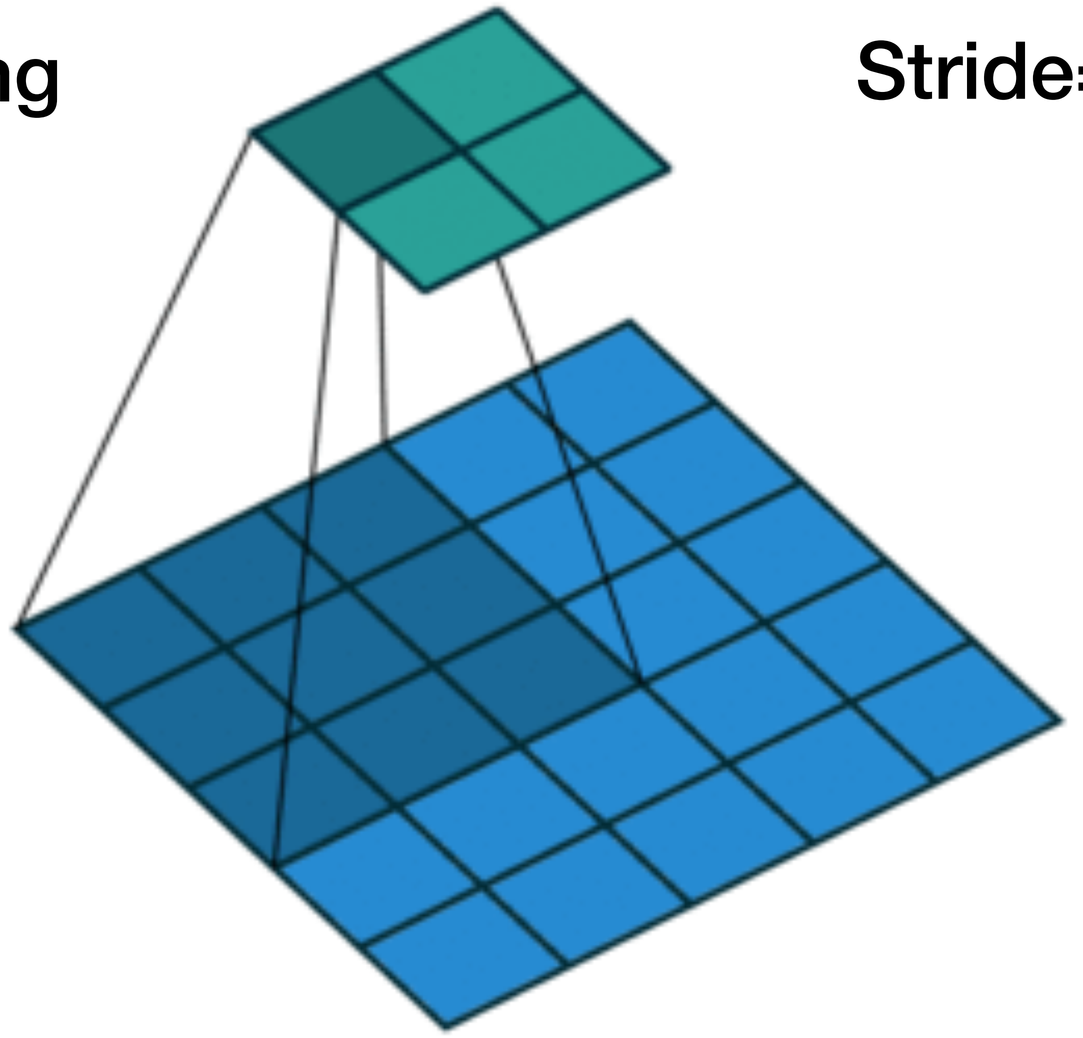
Stride=1, No padding



Stride=1, Padding, P=1



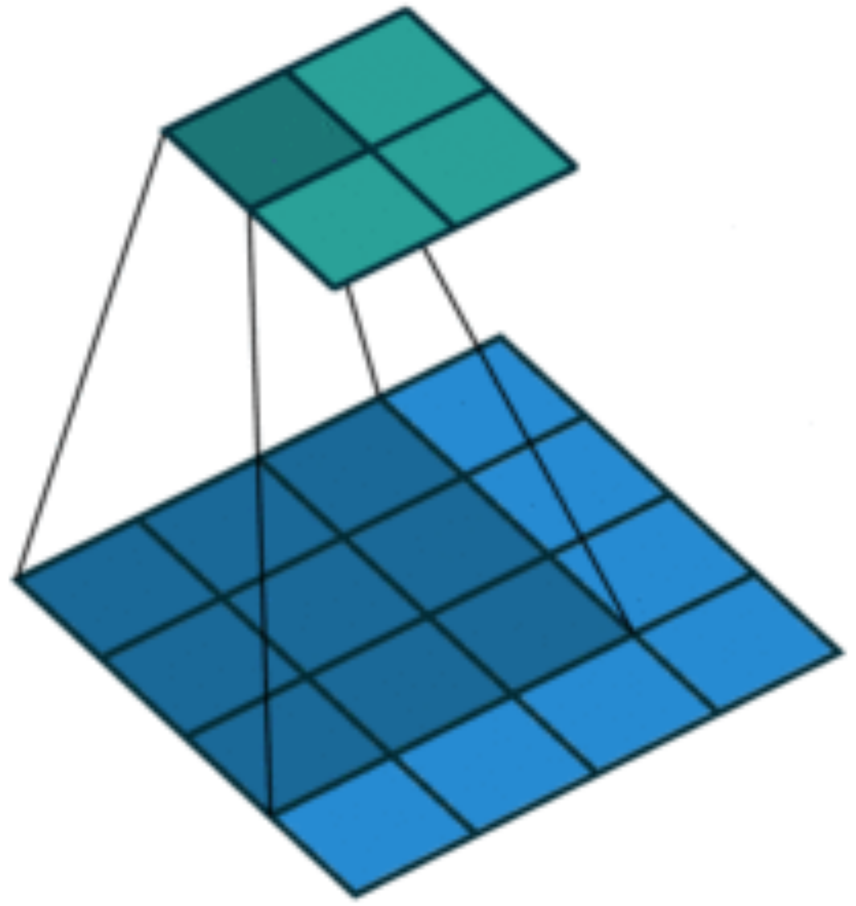
Stride=1, Padding, P=2



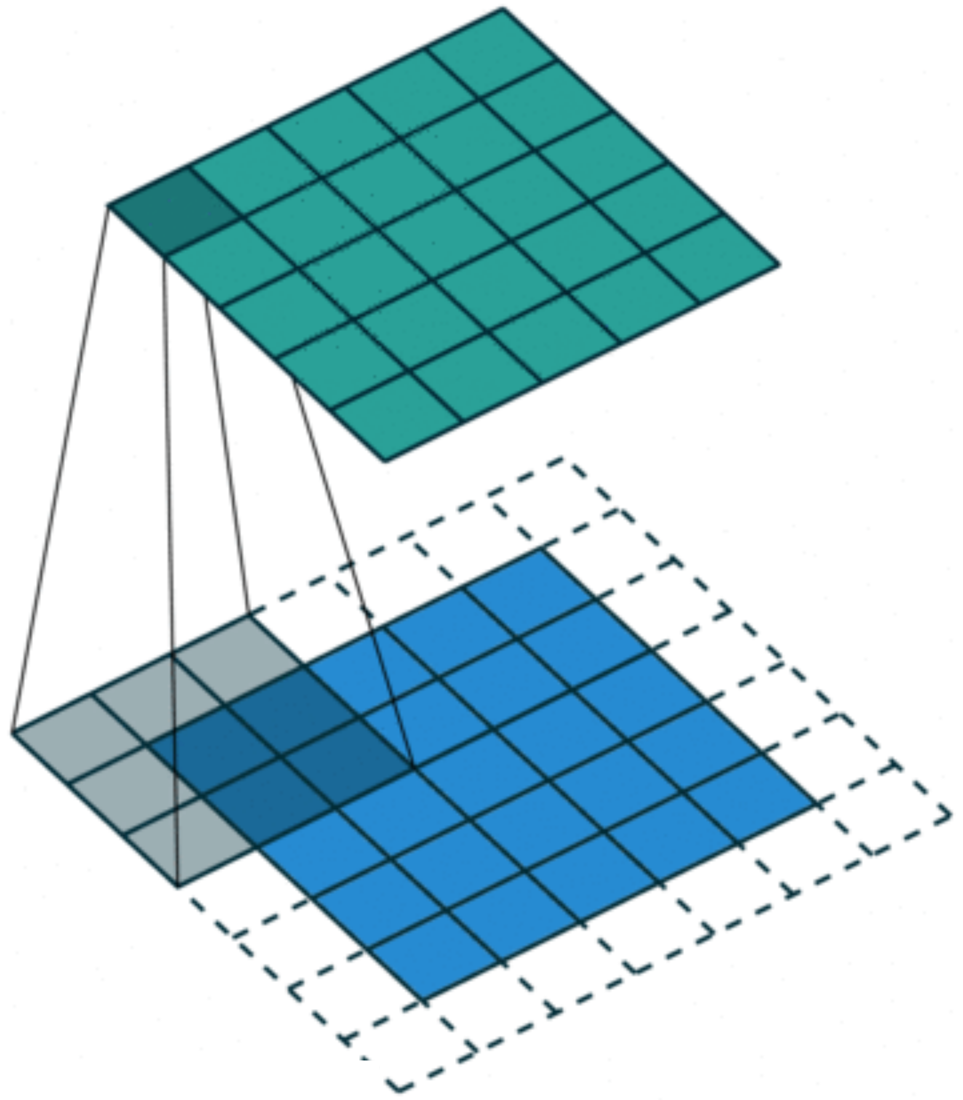
Stride=2, No padding

All animations are from: https://github.com/vdumoulin/conv_arithmetic

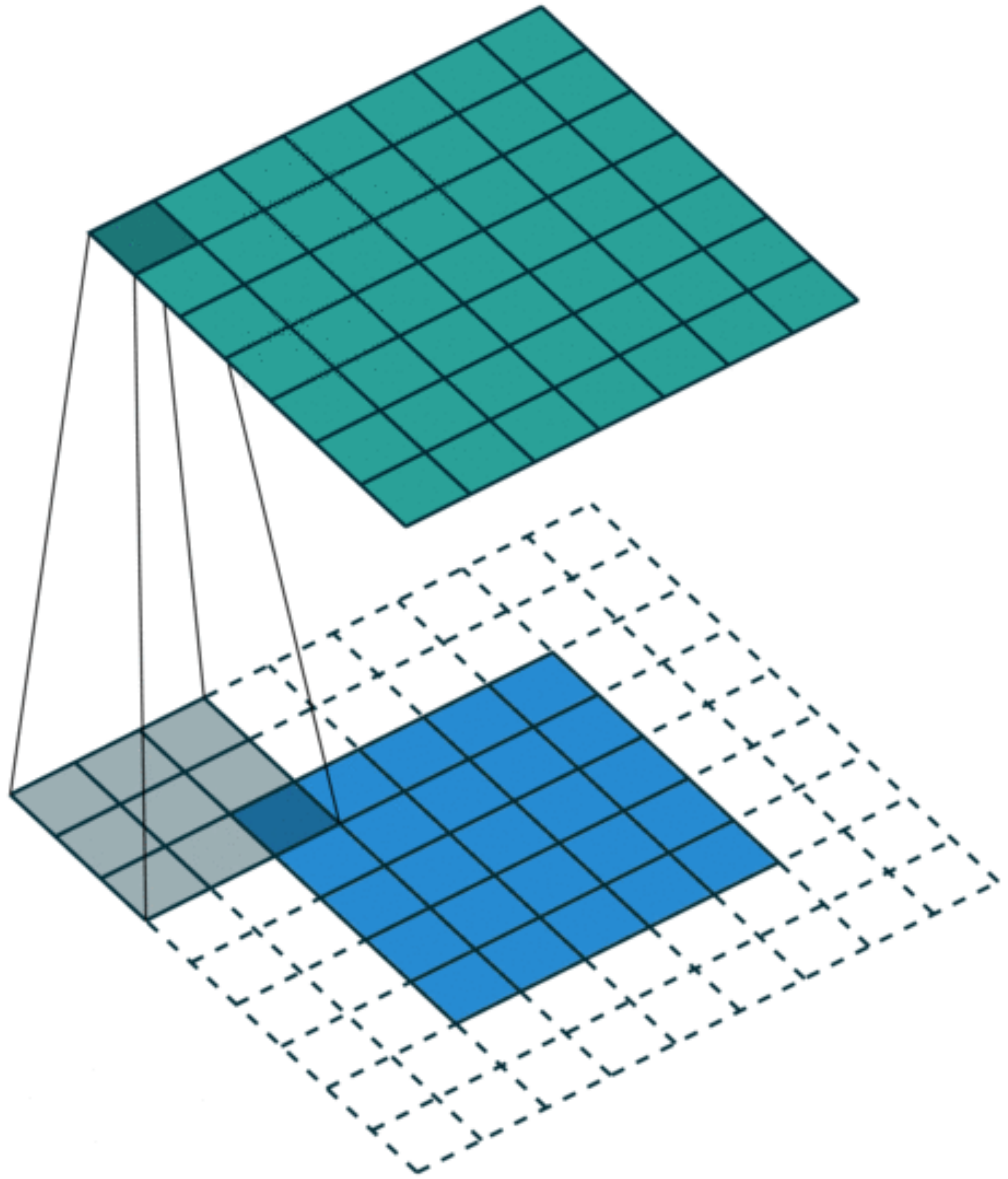
Convolutional Neural Networks (CNNs)



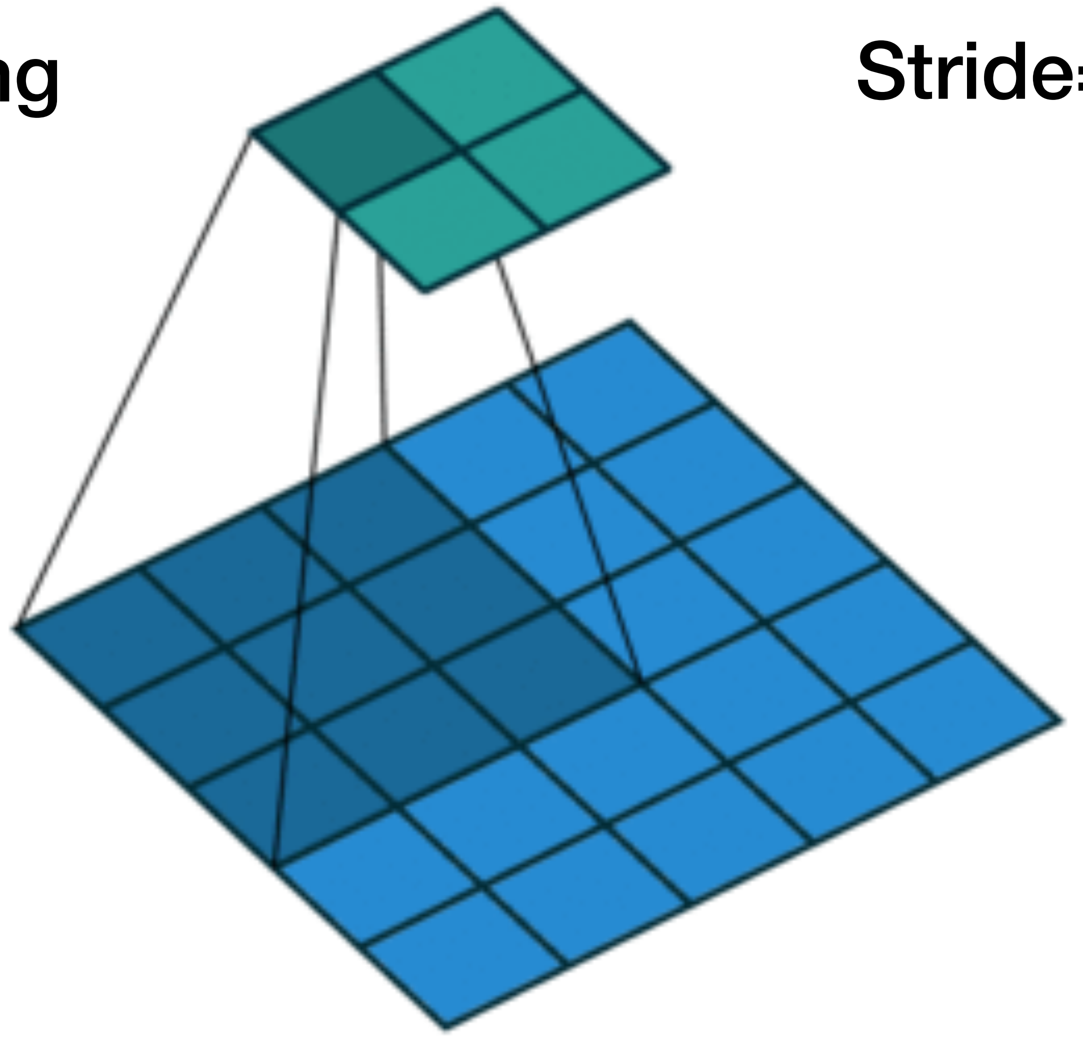
Stride=1, No padding



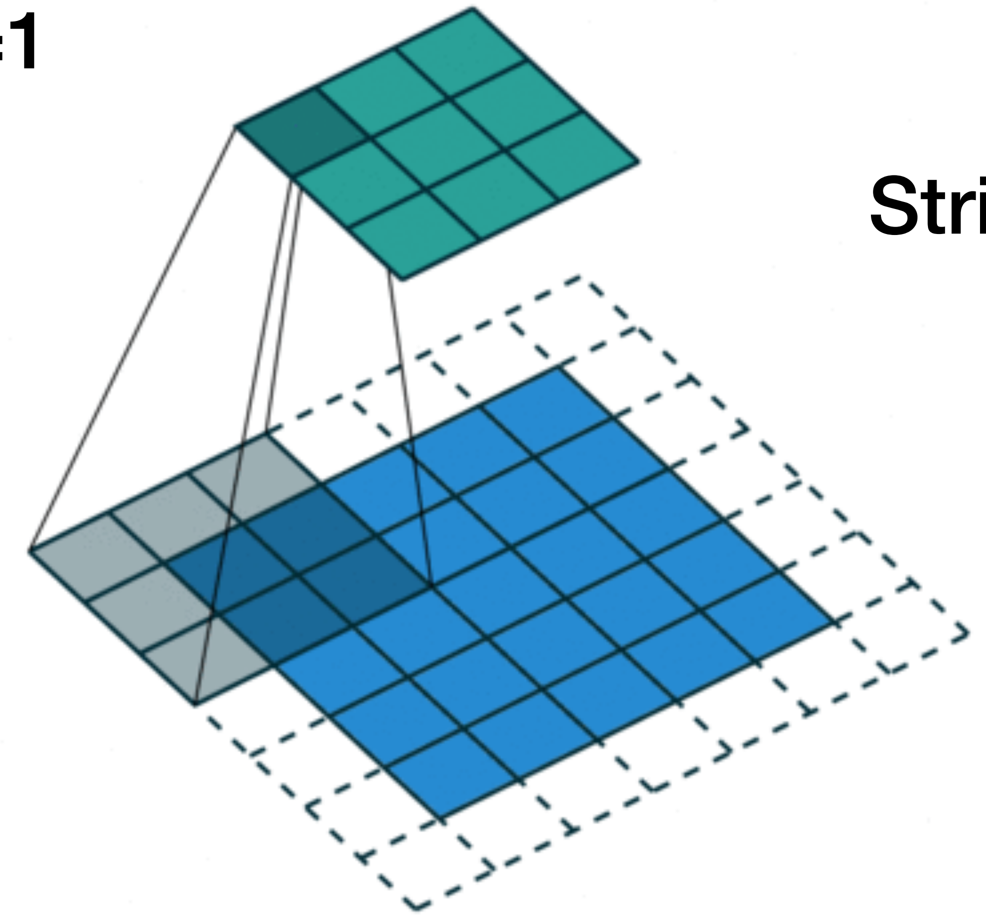
Stride=1, Padding, P=1



Stride=1, Padding, P=2



Stride=2, No padding



Stride=2, Padding, P=1

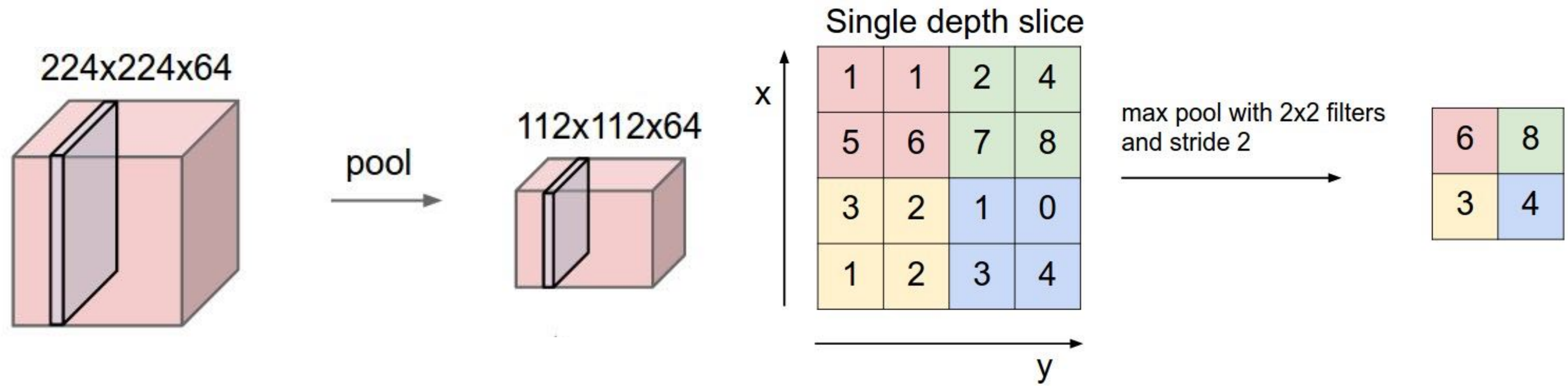
All animations are from: https://github.com/vdumoulin/conv_arithmetic

Convolution Layers: Summary

- Accepts a volume of size $W_1 \times H_1 \times D_1$
- Requires four hyperparameters:
    - Number of filters K,
    - their spatial extent F,
    - the stride S,
    - the amount of zero padding P.
- Produces a volume of size $W_2 \times H_2 \times D_2$ where:
    - $W_2 = (W_1 - F + 2P)/S + 1$
    - $H_2 = (H_1 - F + 2P)/S + 1$ (i.e. width and height are computed equally by symmetry)
    - $D_2 = K$
- With parameter sharing, it introduces $F \cdot F \cdot D_1$ weights per filter, for a total of $(F \cdot F \cdot D_1) \cdot K$ weights and K biases.

Summary from: http://cs231n.github.io/convolutional-networks/

Pooling Layer



- Why pooling?
  Reduce the size of the representation, speed up the computations and make the features a little more robust.

- Max pooling is popularly used in CNNs.

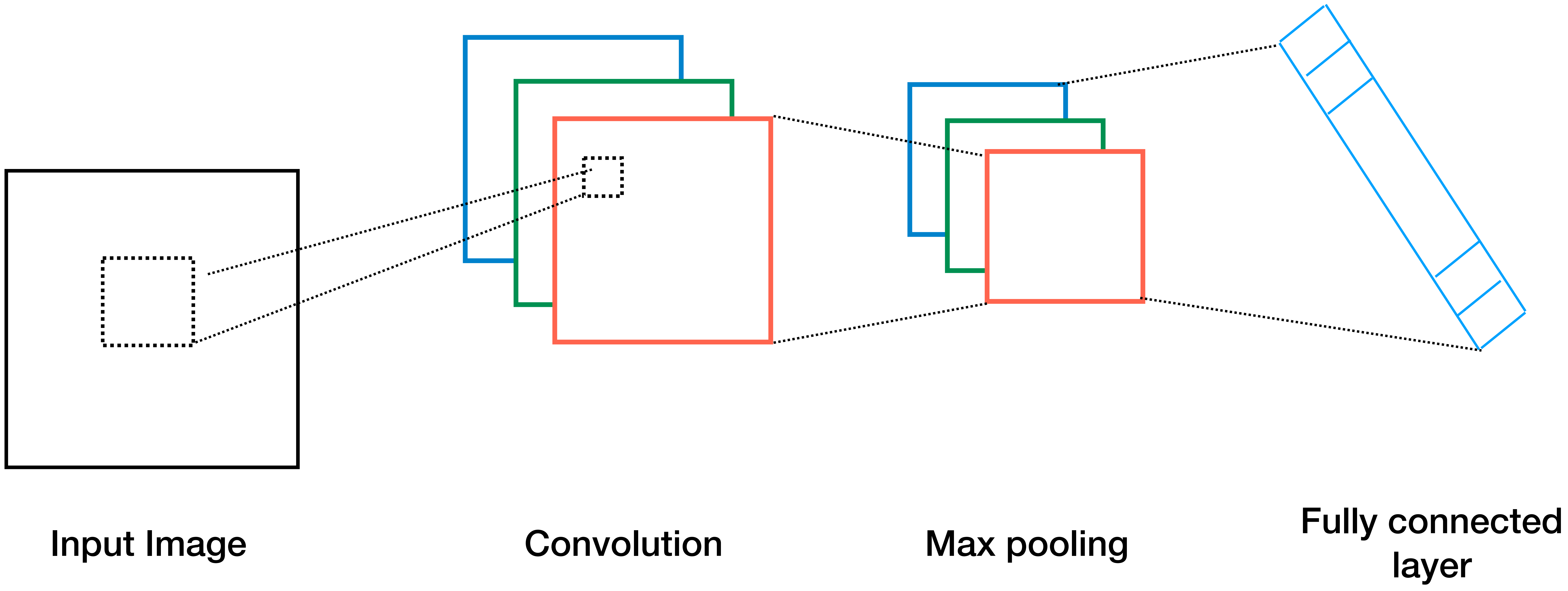
Image from: http://cs231n.github.io/convolutional-networks/

Pooling Layer

- Accepts a volume of size $W_1 \times H_1 \times D_1$
- Requires two hyperparameters:
  - their spatial extent F,
  - the stride S,
- Produces a volume of size $W_2 \times H_2 \times D_2$ where:
  - $W_2 = (W_1 - F)/S + 1$
  - $H_2 = (H_1 - F)/S + 1$
  - $D_2 = D_1$

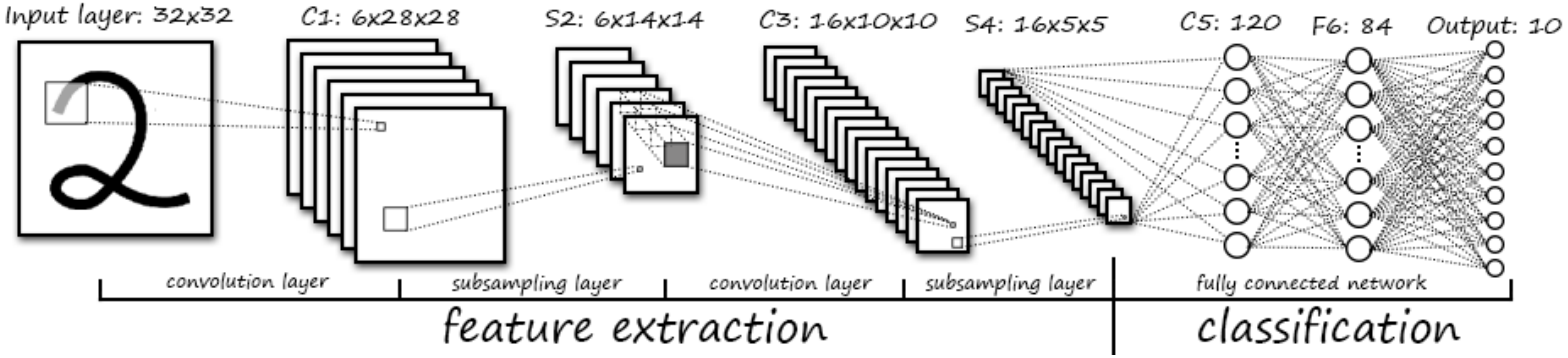
Summary from: http://cs231n.github.io/convolutional-networks/

Convolutional Architectures



- Block that can be repeated: Convolutional layer, followed by non-linearity (e.g. ReLU) + Max pooling
- Fully connected layers before classification

LeNet-5

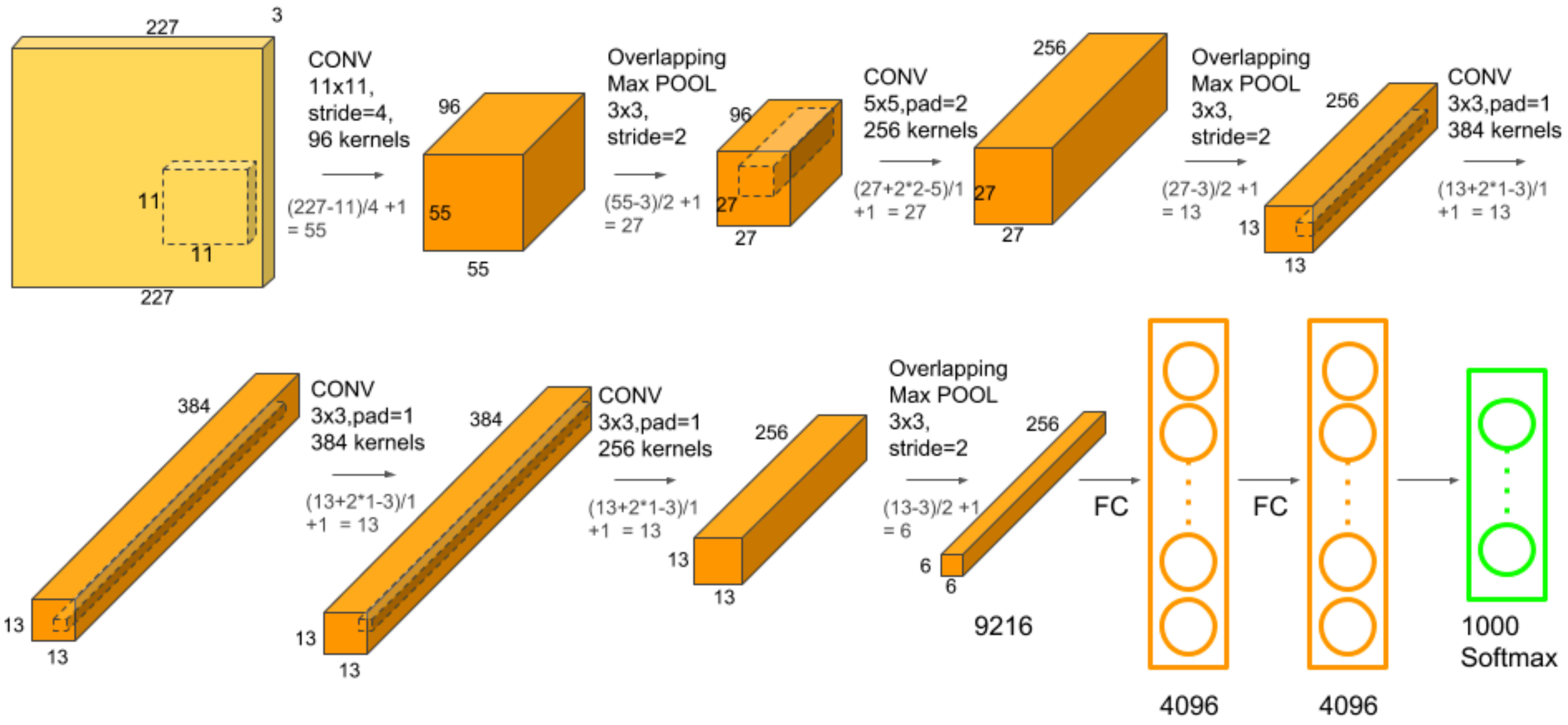


- One of the first successful CNN architectures

- Used to classify images of hand-written digits

LeNet-5: LeCun, Y., Bottou, L., Bengio, Y. and Haffner, P., 1998. "Gradient-based learning applied to document recognition". Proceedings of the IEEE, 86(11), pp.2278-2324.

AlexNet

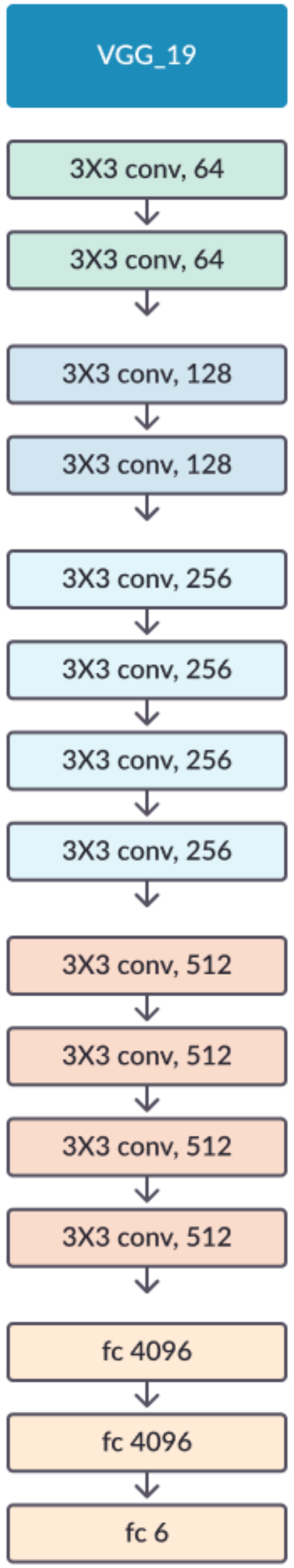


- Winner (by a large margin) of the ImageNet challenge in 2012.
- Much larger than previous architectures.

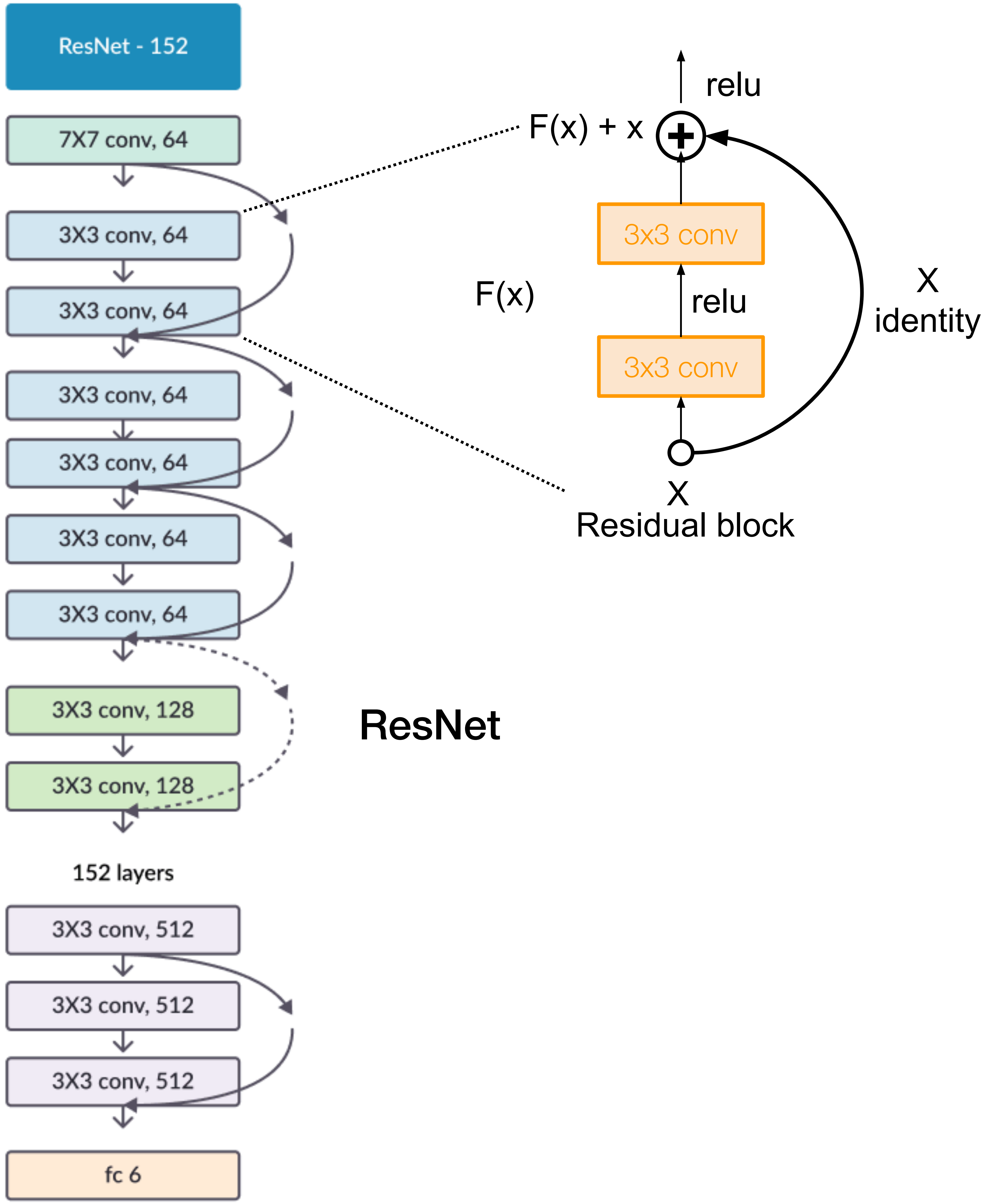
AlexNet: ImageNet Classification with Deep Convolutional Neural Networks by Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton, NIPS 2012

Image from: https://www.learnopencv.com/understanding-alexnet/

Other Architectures



VGG19



ResNet

VGG: Simonyan, Karen, and Andrew Zisserman. "Very deep convolutional networks for large-scale image recognition." arXiv preprint arXiv:1409.1556 (2014).
ResNet: K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. arXiv preprint arXiv:1512.03385,2015.

Vision and deep neural networks

- Some tasks that have seen a lot of progress:
  - Person/face/object recognition
  - Image segmentation
  - Single person pose estimation
- Some tasks that require a lot more work:
  - Comprehensive scene understanding
  - Reasoning geometrically
  - Generalizing to new domains

NLP/speech and deep neural networks

- Some tasks that have seen a lot of progress:
  - Speech recognition on high-resource languages (e.g. English, Chinese)
  - Low-level NLP tasks like part-of-speech tagging, etc.
- Some tasks that require a lot more work:
  - Natural language understanding
  - NLP/speech technologies for low-resource scenarios
  - Reasoning about large documents