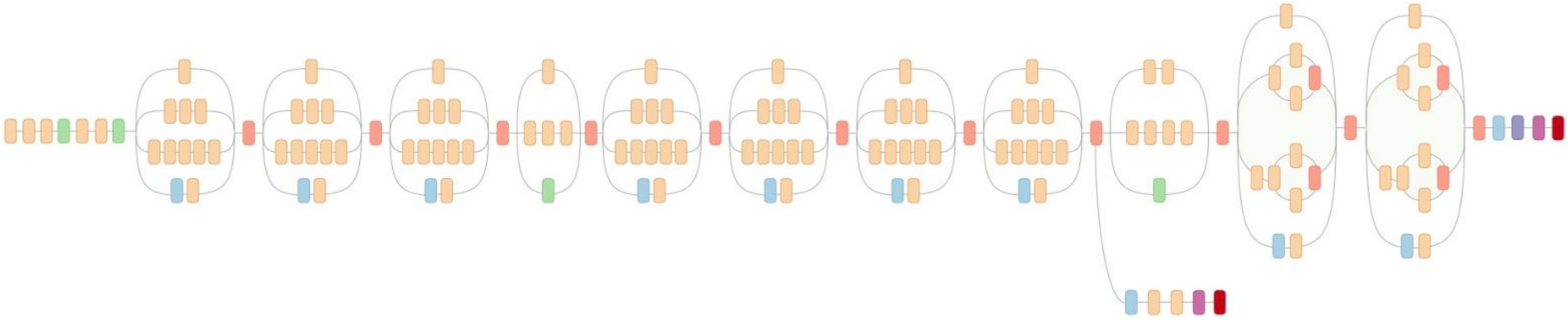


Convolutional Neural Networks

Presenter: Dr. Denis Krompaß
Siemens Corporate Technology – Machine Intelligence Group
Co-Founder - creaidAI

Date: 07.11.2018

Lecture Overview

Introduction and Motivation

The Convolutional Neural Network Layer

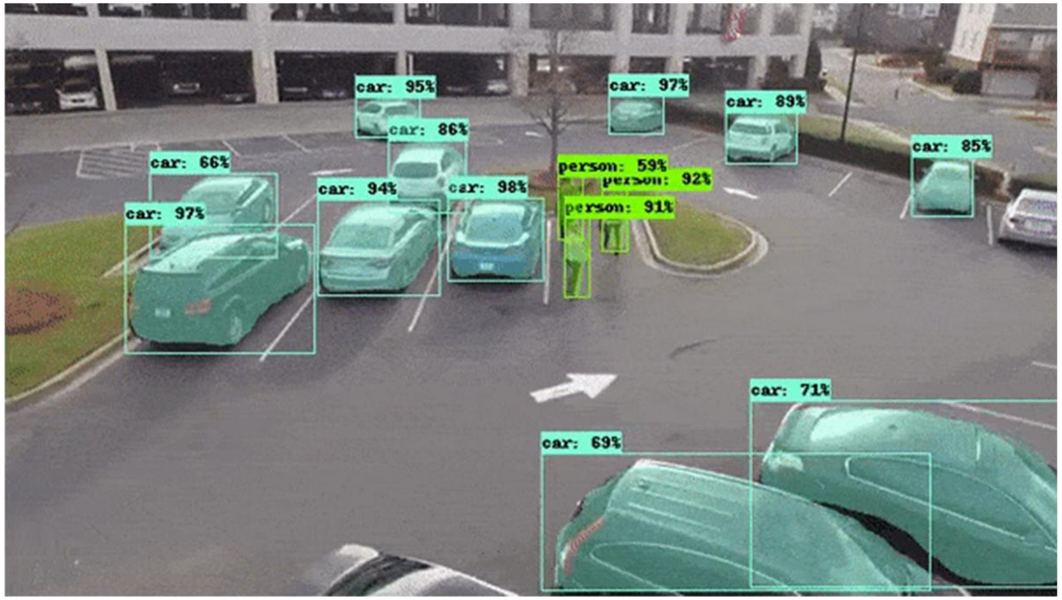
Convolutional Neural Networks

Training Very Deep Convolutional Neural Networks

Convolutional Neural Networks

Applications

Object Detection / Image Segmentation



Source:
https://towardsdatascience.com/using-tensorflow-object-detection-to-do-pixel-wise-classification-702bf2605182

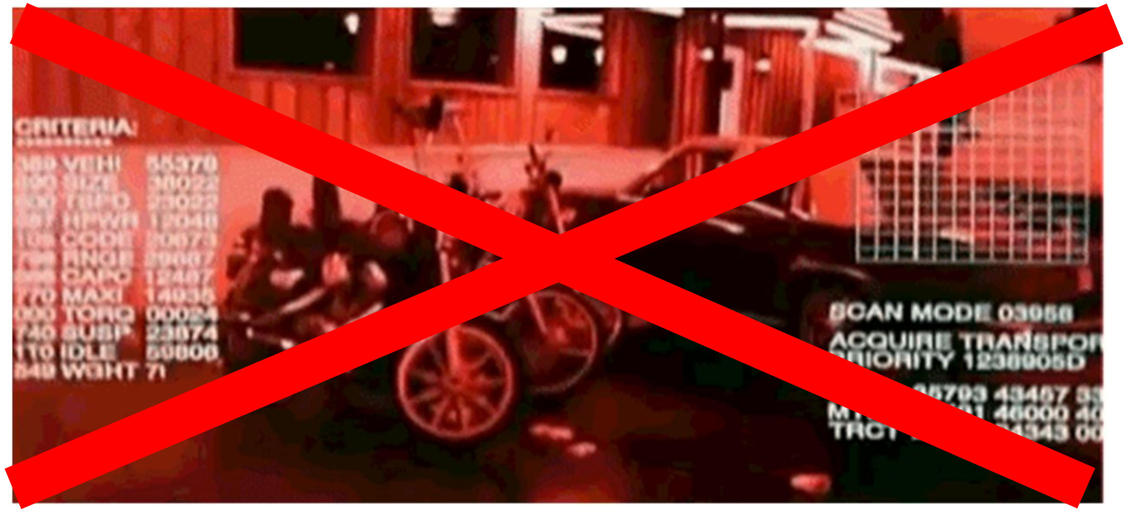
Nice Video:
https://www.youtube.com/watch?v=OOT3UIXZztE

Perception in Control Tasks

Source: https://techcrunch.com/2016/09/21/scientists-teach-machines-to-hunt-and-kill-humans-in-doom-deathmatch-mode/?guccounter=1

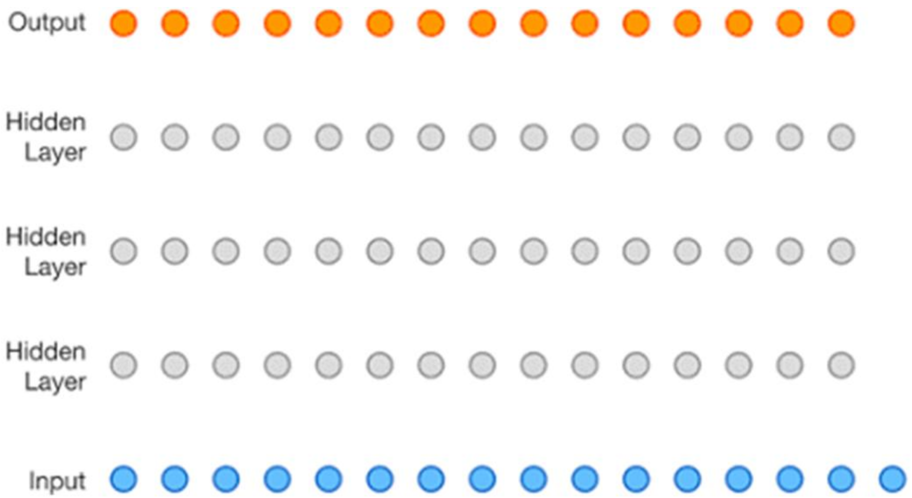
Winning Team:
Alexey Dosovitskiy, Vladlen Koltun. Learning to Act by Predicting the Future. arXiv:1611.01779v2, 2016

Perception in Control Tasks



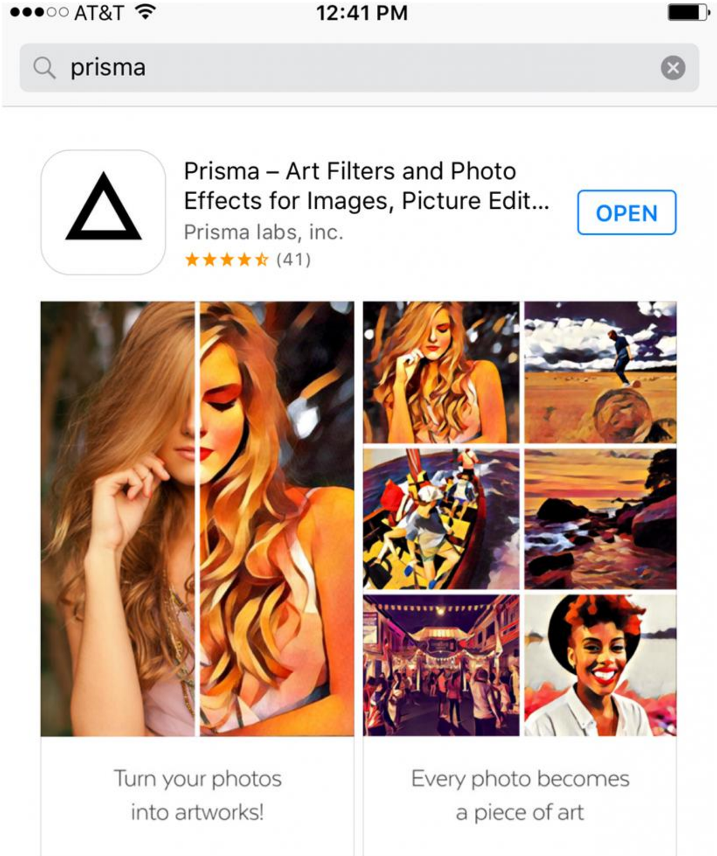
No worries, we are far far away from that …

It‘s not just images…



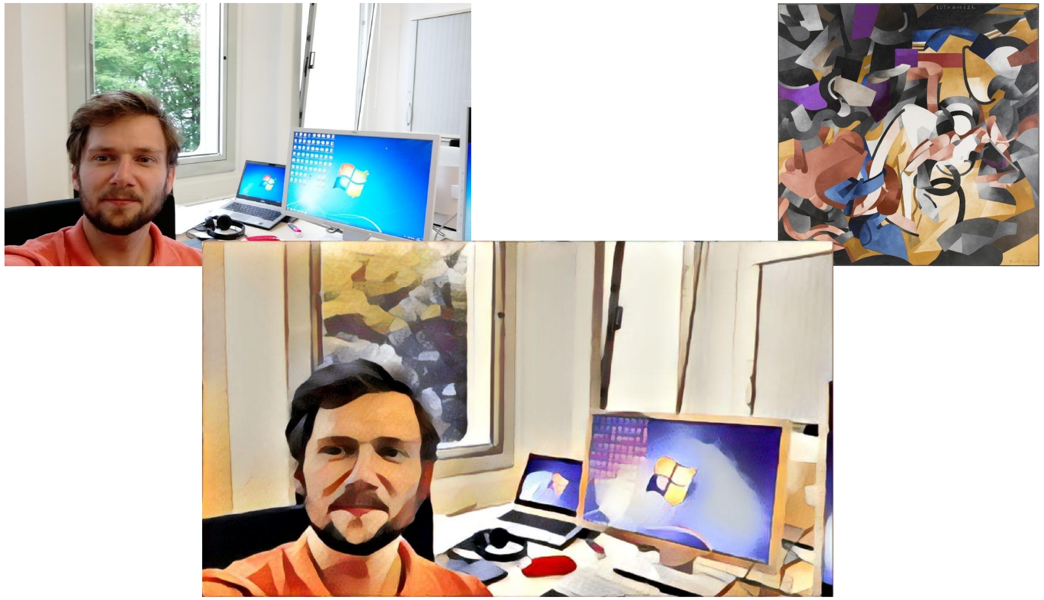
https://deepmind.com/blog/wavenet-generative-model-raw-audio/

Neural Artistic Style Transformations



Original work:
Leon A. Gatys, Alexander S. Ecker, Matthias Bethge.
A Neural Algorithm of Artistic Style. arXiv:1508.06576v2, 2015

- 7.5 Million downloads one week after release.



Also works with videos these days: https://www.youtube.com/watch?v=BcflKNzO31A

Data Generation

Source and work: Tero Karras, Timo Aila, Samuli Laine, Jaakko Lehtinen. Progressive Growing of GANs for Improved Quality, Stability, and Variation. arXiv:1710.10196v3, 2018

Source (gif):
https://www.theverge.com/2017/10/30/16569402/ai-generate-fake-faces-celebs-nvidia-gan

Full Video:
https://www.youtube.com/watch?v=XOxxPcy5Gr4

Convolutional Neural Networks

History

Convolutional Neural Networks - Invention

Yann LeCun

facebook

Generalization and Network Design Strategies **1989**

Yann le Cun *
Department of Computer Science, University of Toronto
Toronto, Ontario, M5S 1A4. CANADA.

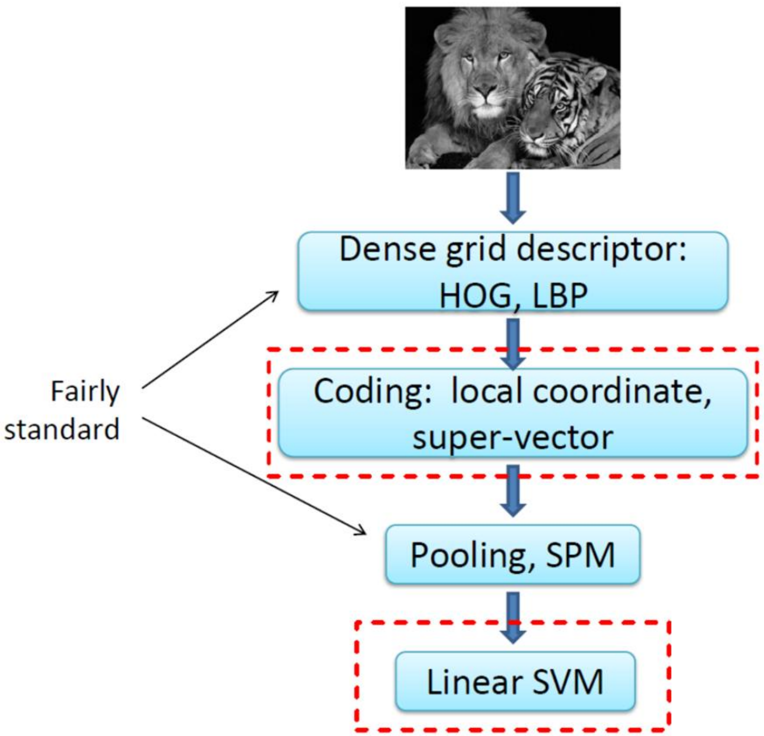
Abstract

An interesting property of connectionist systems is their ability to learn from examples. Although most recent work in the field concentrates on reducing learning times, the most important feature of a learning machine is its generalization performance. It is usually accepted that good generalization performance on real-world problems cannot be achieved unless some *a priori* knowledge about the task is built into the system. Back-propagation networks provide a way of specifying such knowledge by imposing constraints both on the architecture of the network and on its weights. In general, such constraints can be considered as particular transformations of the parameter space

Building a constrained network for image recognition appears to be a feasible task. We describe a small handwritten digit recognition problem and show that, even though the problem is linearly separable, single layer networks exhibit poor generalization performance. Multilayer constrained networks perform very well on this task when organized in a hierarchical structure with shift invariant feature detectors.

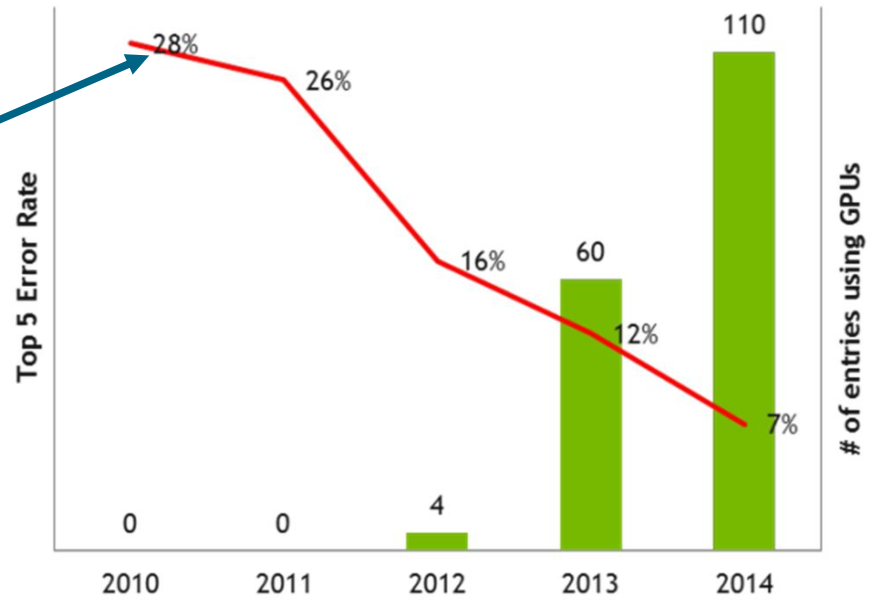
These results confirm the idea that minimizing the number of free parameters in the network enhances generalization.

Convolutional Neural Networks - Breakthrough



http://image-net.org/challenges/LSVRC/2010/ILSVRC2010_NEC-UIUC.pdf

IMAGENET



https://devblogs.nvidia.com/nvidia-ibm-cloud-support-imagenet-large-scale-visual-recognition-challenge/

Convolutional Neural Networks - Breakthrough

IMAGENET

High-dimensional image signatures: Fisher Vectors (FV)

Perronnin, Sánchez and Mensink, "Improving the Fisher kernel for large-scale image classification", ECCV'10.

+

Compression: Product Quantization (PQ)

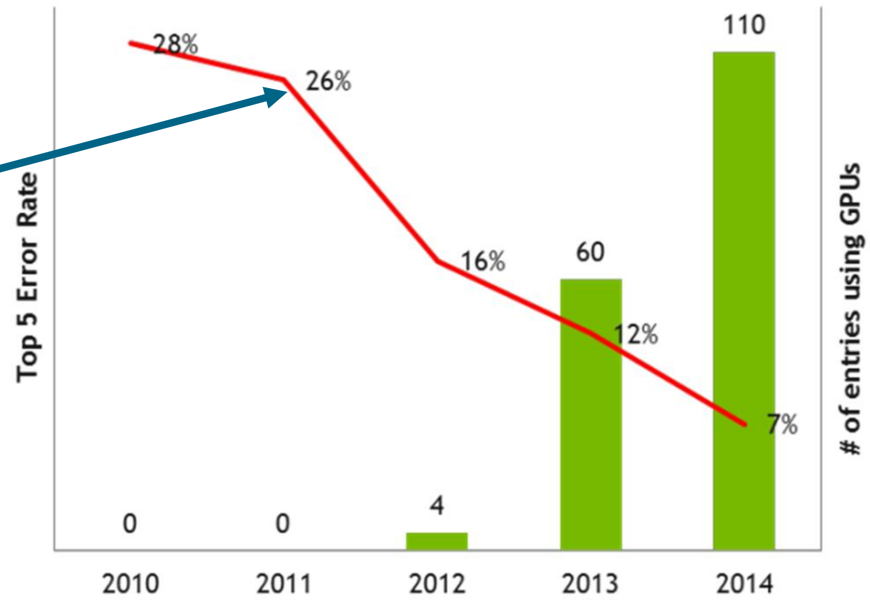
Sánchez and Perronnin, "High-dimensional signature compression for large-scale image classification", CVPR'11.

+

Simple machine learning: one-vs-all linear SVMs

Linear classifiers learned in primal using Stochastic Gradient Descent (SGD)

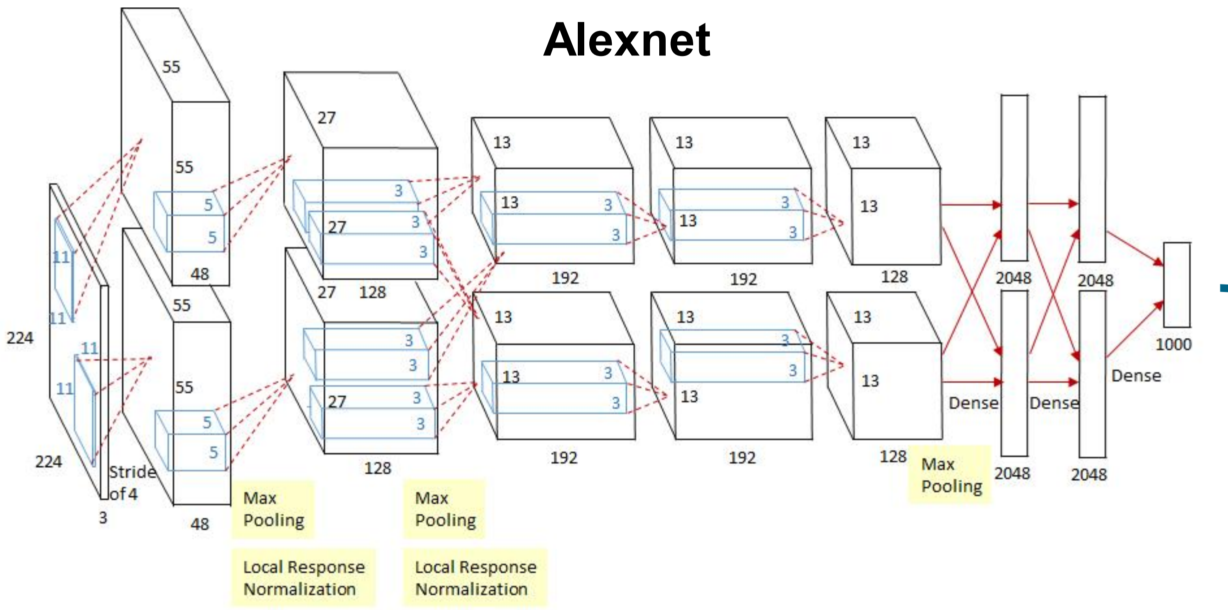
F. Perronnin, J. Sánchez, "Compressed Fisher vectors for LSVRC",
PASCAL VOC / ImageNet workshop, ICCV, 2011



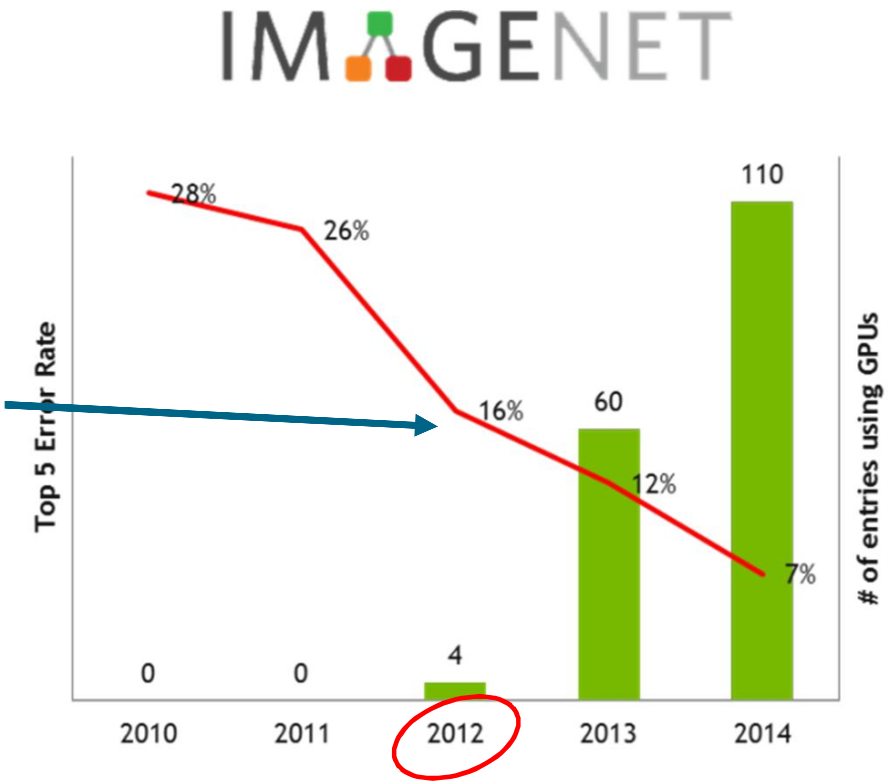
https://devblogs.nvidia.com/nvidia-ibm-cloud-support-imagenet-large-scale-visual-recognition-challenge/

Convolutional Neural Networks - Breakthrough

Alexnet



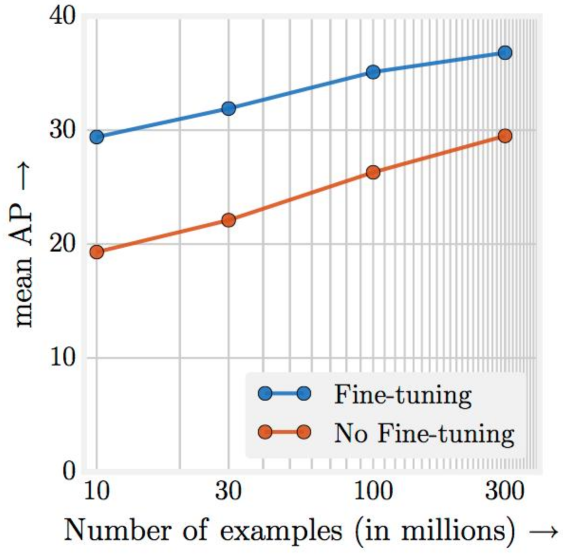
https://medium.com/coinmonks/paper-review-of-alexnet-caffenet-winner-in-ilsvrc-2012-image-classification-b93598314160



https://devblogs.nvidia.com/nvidia-ibm-cloud-support-imagenet-large-scale-visual-recognition-challenge/

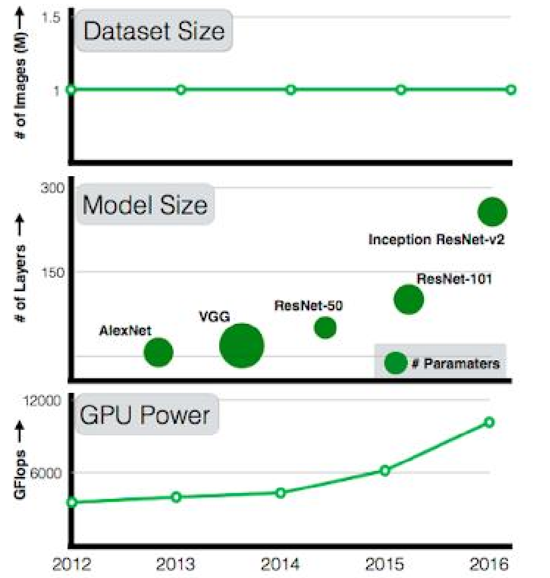
Convolutional Neural Networks - Breakthrough

Huge amounts of labeled data



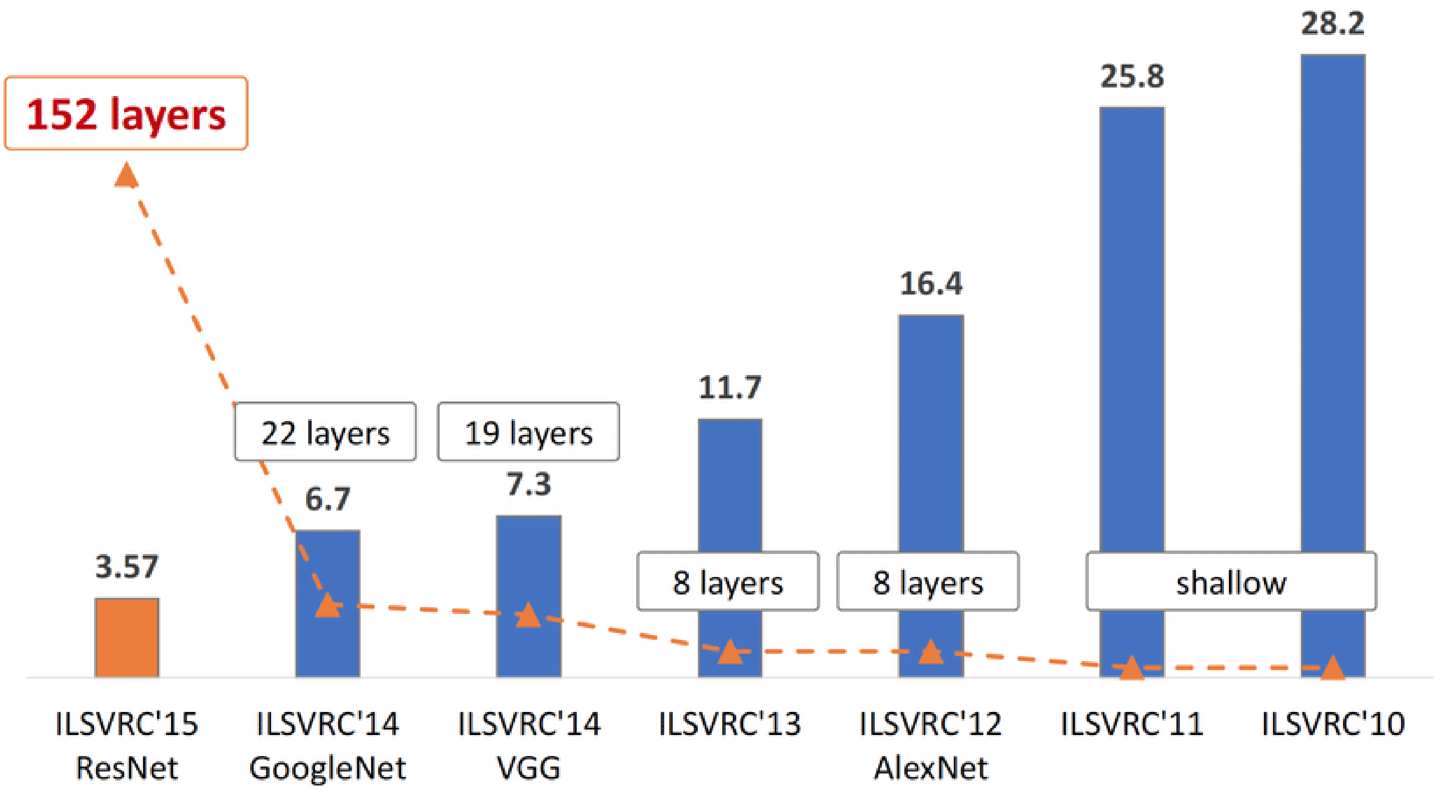
https://ai.googleblog.com/2017/07/revisiting-unreasonable-effectiveness.html

GPUs



https://ai.googleblog.com/2017/07/revisiting-unreasonable-effectiveness.html

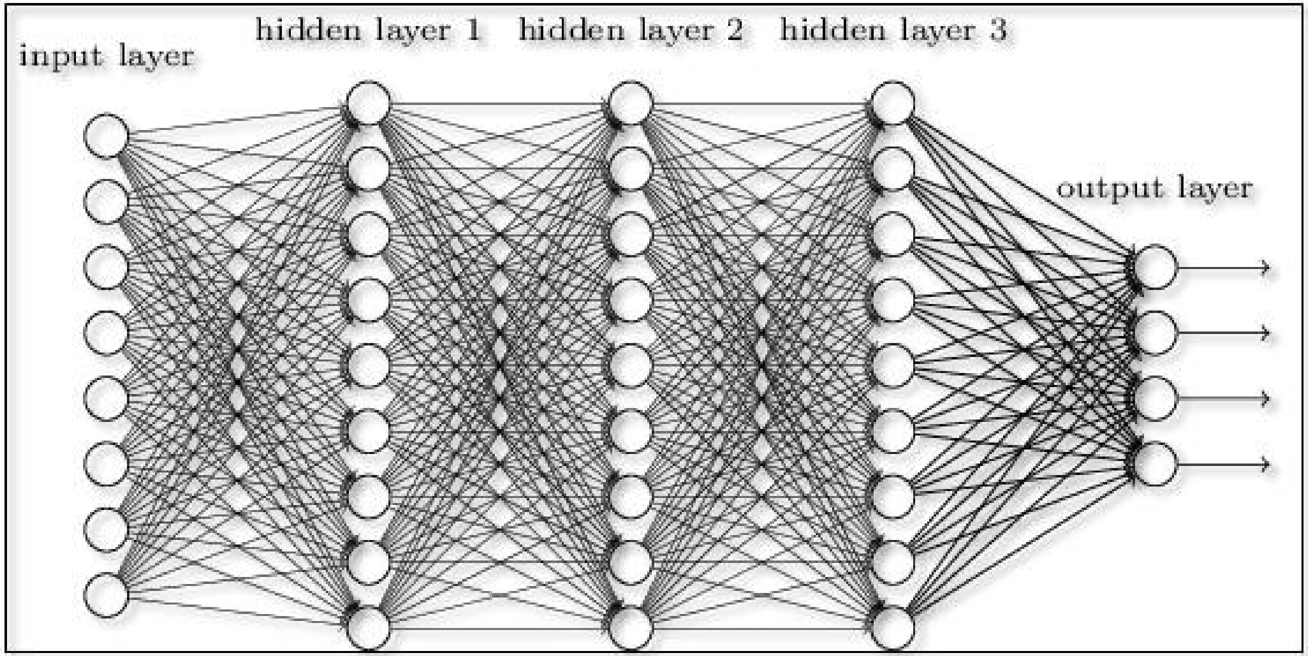
Convolutional Neural Networks - Breakthrough



https://www.researchgate.net/figure/The-evolution-of-the-winning-entries-on-the-ImageNet-Large-Scale-Visual-Recognition_fig1_321896881

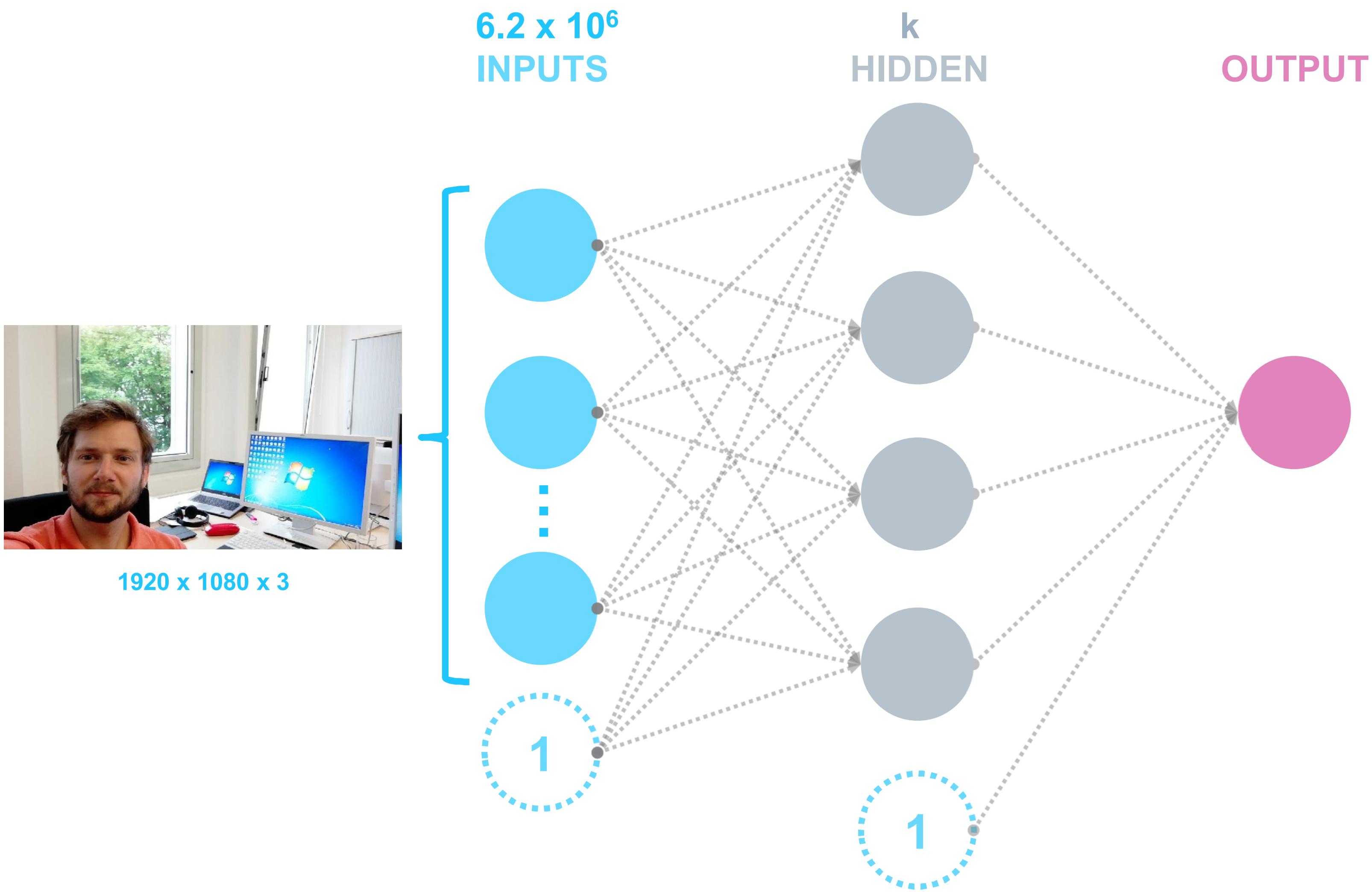
Convolutional Neural Networks

Why we Need Them



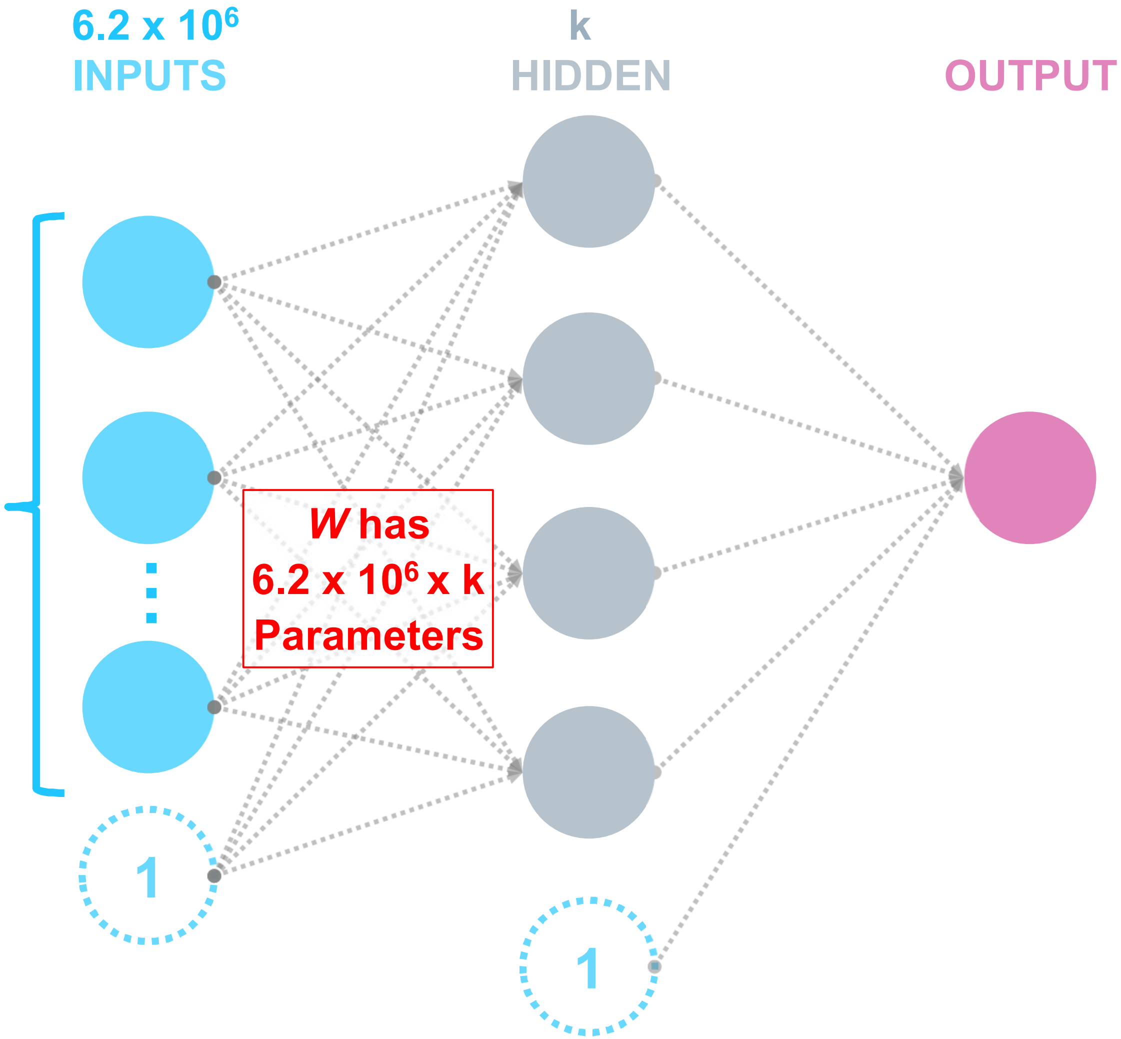
http://houseofbots.com/news-detail/1442-1-what-is-deep-learning-and-neural-network

Dense Layers on High Dimensional Inputs

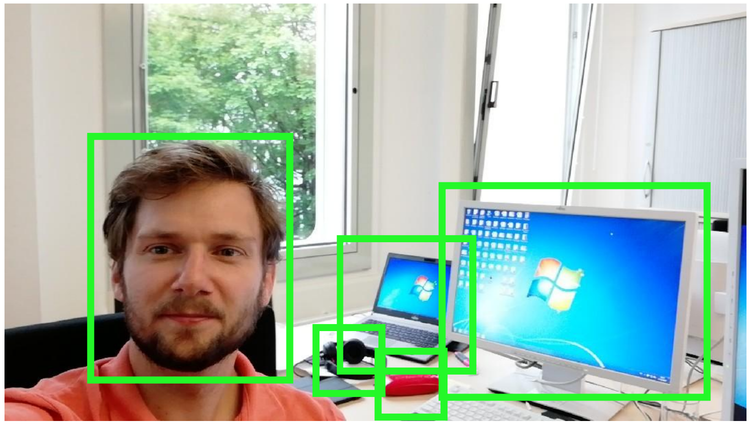


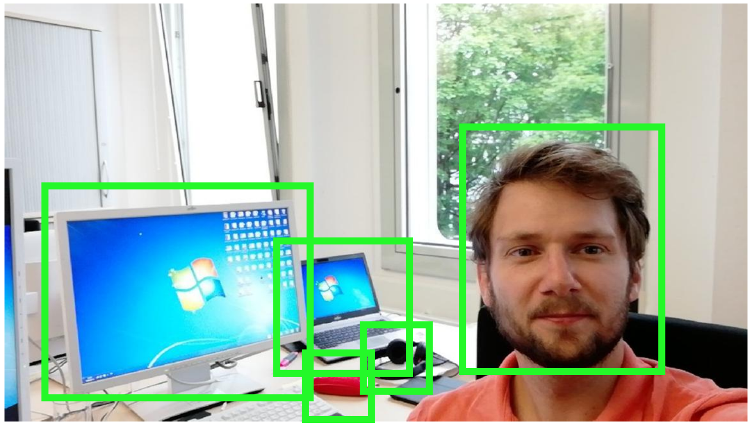
Dense Layers are Expensive

1920 x 1080 x 3



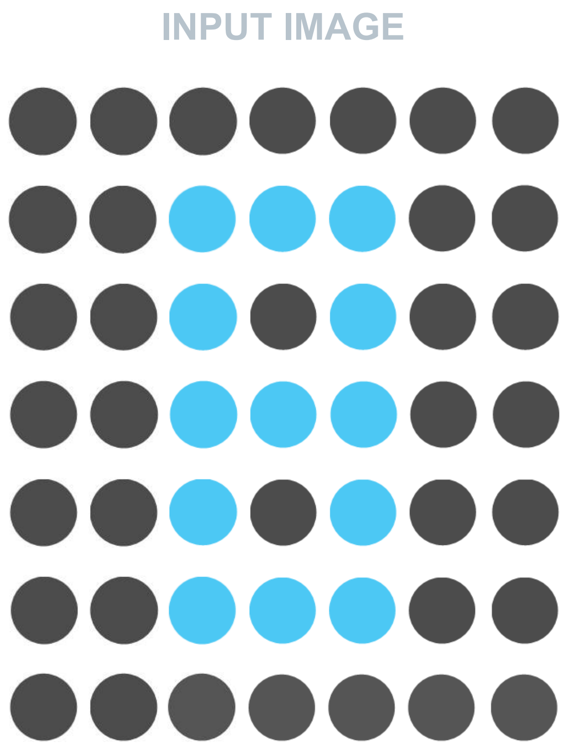
Translation Invariance



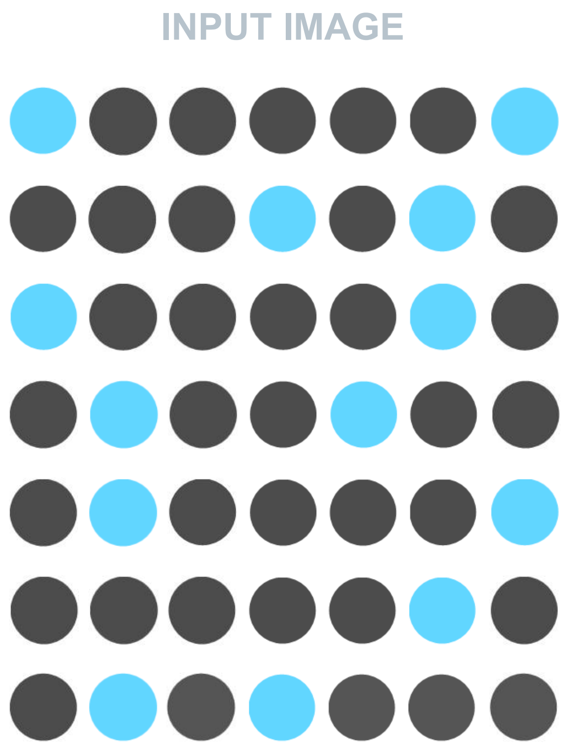


It is natural to have some degree of invariance to where objects occur in a scene.

Perception of a Dense Layer



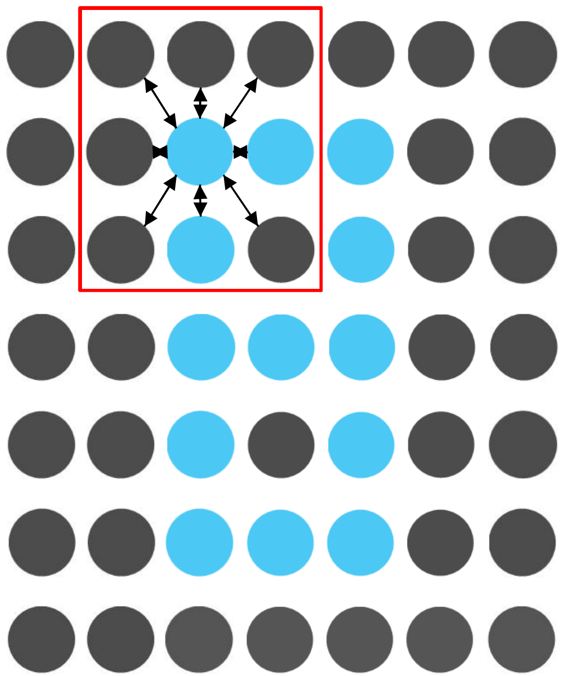
That's an 8!

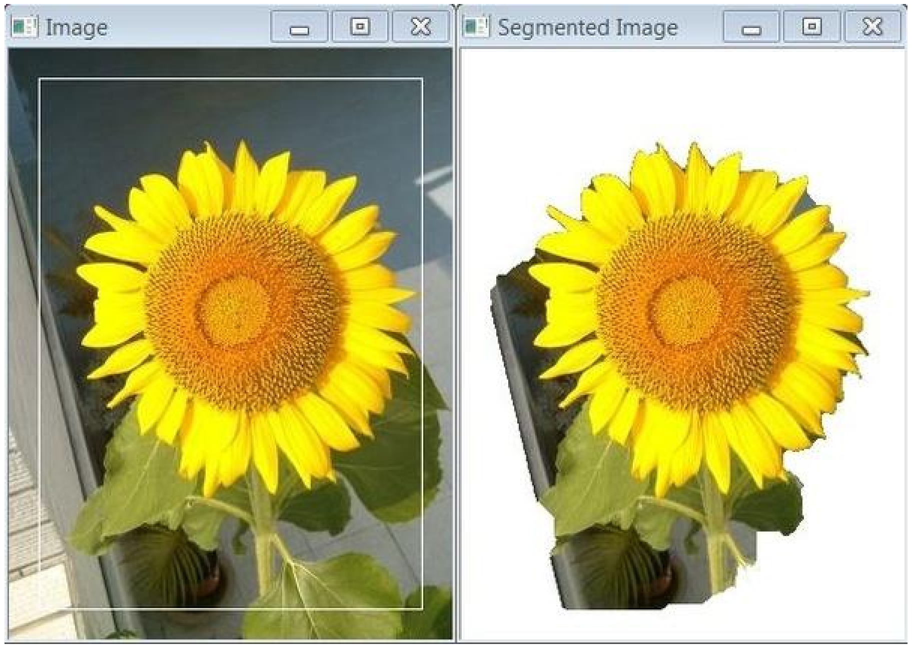


That's an 8!

Can we do better?

INPUT IMAGE





The Convolutional Neural Network Layer

When using this layer as the first layer in a model, provide an `input_shape` argument (tuple of integers or `None`, e.g. `(10, 128)` for sequences of 10 vectors of 128-dimensional vectors, or `(None, 128)` for variable-length sequences of 128-dimensional vectors.

Arguments:

- `filters`: Integer, the dimensionality of the output space (i.e. the number of output filters in the convolution).
- `kernel_size`: An integer or tuple/list of a single integer, specifying the length of the 1D convolution window.
- `strides`: An integer or tuple/list of a single integer, specifying the stride length of the convolution. Specifying any stride value != 1 is incompatible with specifying any `dilation_rate` value != 1.
- `padding`: One of `"valid"`, `"causal"` or `"same"` (case-insensitive). `"causal"` results in causal (dilated) convolutions, e.g. output[t] does not depend on input[t+1:]. Useful when modeling temporal data where the model should not violate the temporal order. See WaveNet: A Generative Model for Raw Audio, section 2.1.
- ~~`data_format`: A string, one of `channels_last` (default) or `channels_first`~~.
- `dilation_rate`: an integer or tuple/list of a single integer, specifying the dilation rate to use for dilated convolution. Currently, specifying any `dilation_rate` value != 1 is incompatible with specifying any `strides` value != 1.
- `activation`: Activation function to use. If you don't specify anything, no activation is applied (ie. "linear" activation: `a(x) = x`).
- `use_bias`: Boolean, whether the layer uses a bias vector.
- `kernel_initializer`: Initializer for the `kernel` weights matrix.
- `bias_initializer`: Initializer for the bias vector.

The 1D Convolution Operator

Discrete Form

FEATURE MAP

KERNEL (or FILTER)

KERNEL SIZE

$$s_i = (\boldsymbol{k} * \boldsymbol{x})_i = \sum_{m=0}^{M-1} k_m x_{i-m+(M-1)}$$

(TIME) INDEX

INPUT

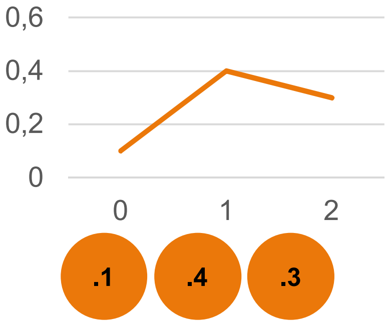
The 1D Convolution Operator

FEATURE MAP

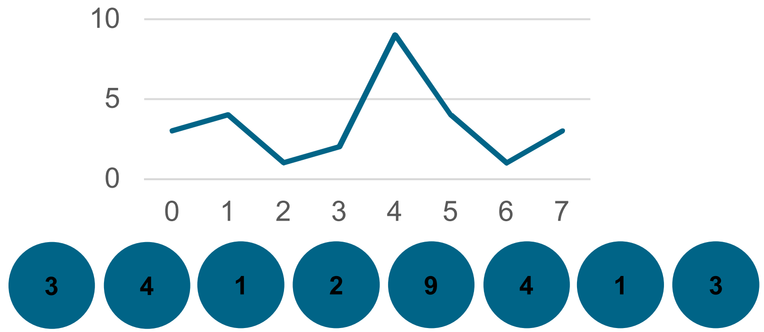
KERNEL SIZE

$$s_i = (k * x)_i = \sum_{m=0}^{M-1} k_m x_{i-m+(M-1)}$$

KERNEL



INPUT



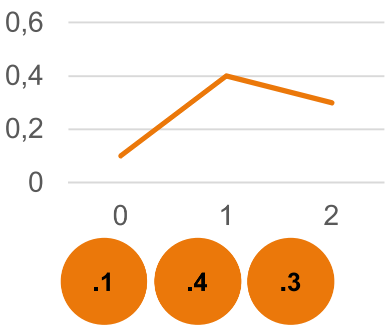
The 1D Convolution Operator

FEATURE MAP

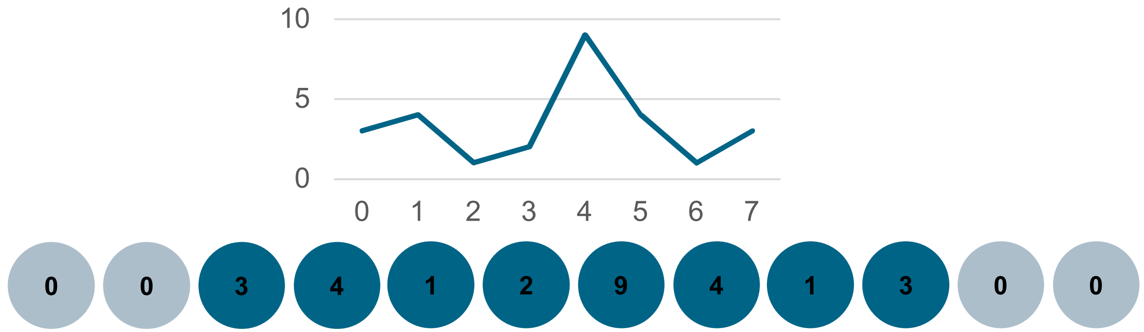
KERNEL SIZE

$$s_i = (k * x)_i = \sum_{m=0}^{M-1} k_m x_{i-m+(M-1)}$$

KERNEL

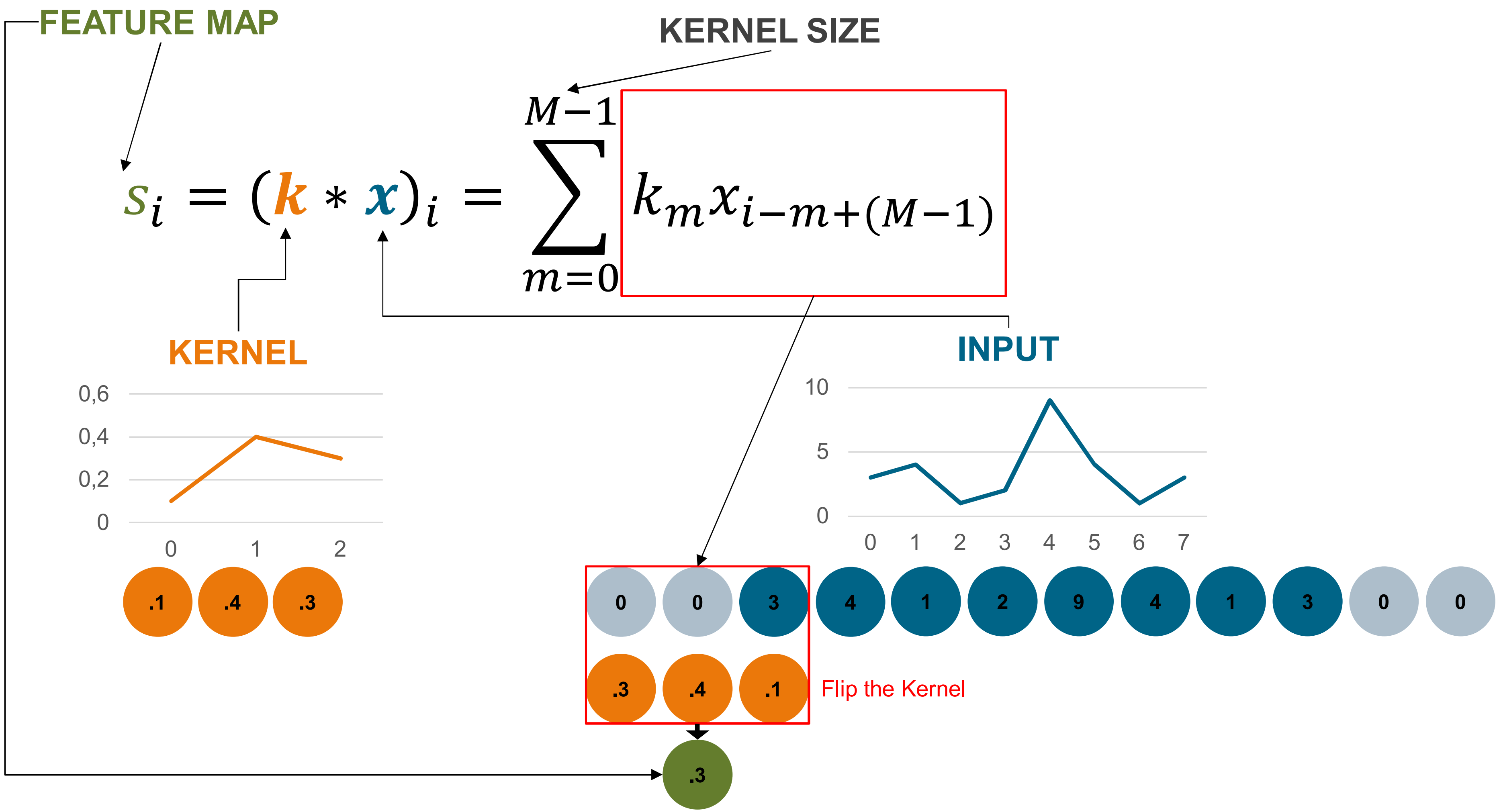


INPUT

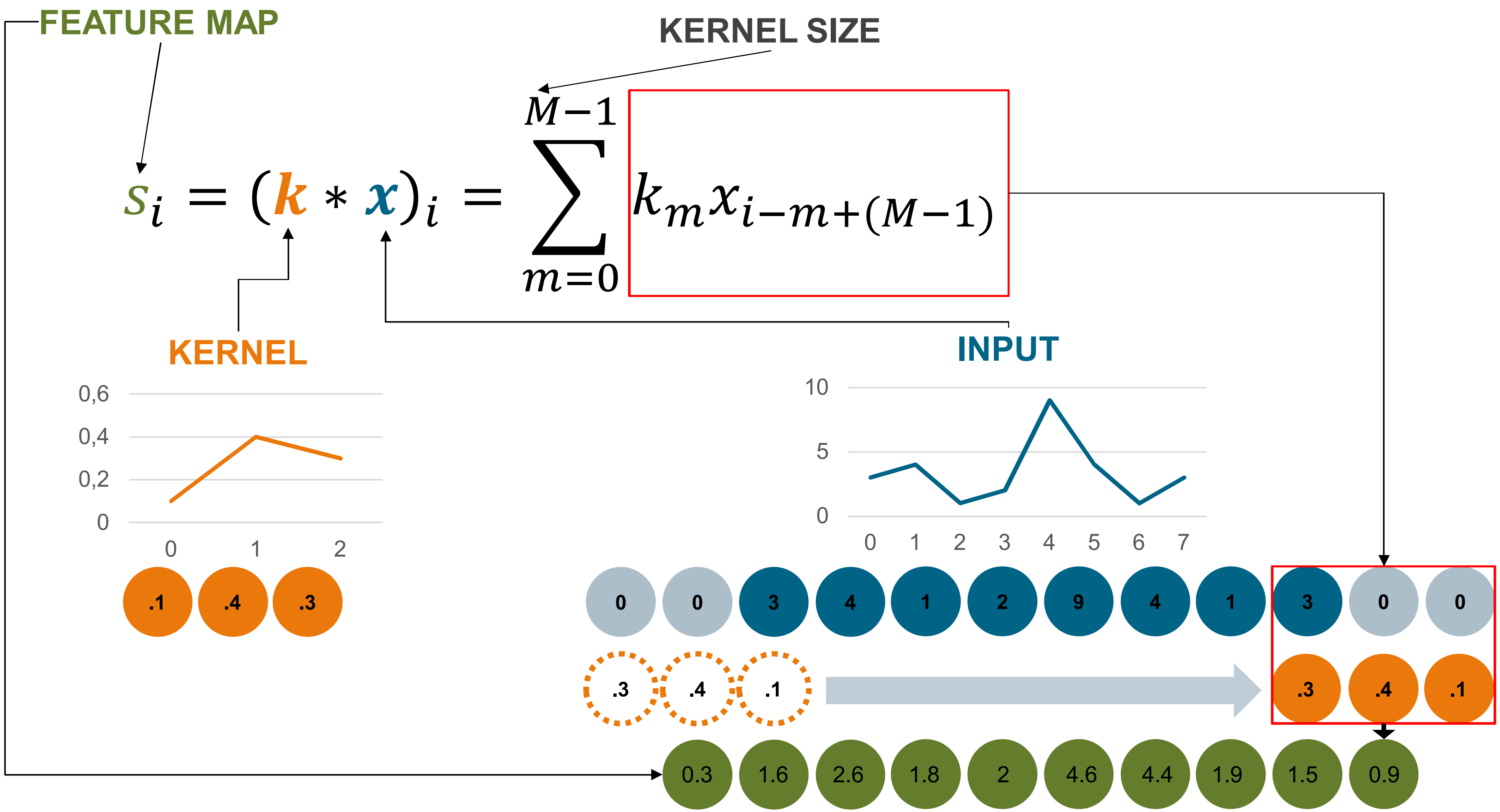


The 1D Convolution Operator

$$s_i = (k * x)_i = \sum_{m=0}^{M-1} k_m x_{i-m+(M-1)}$$



The 1D Convolution Operator

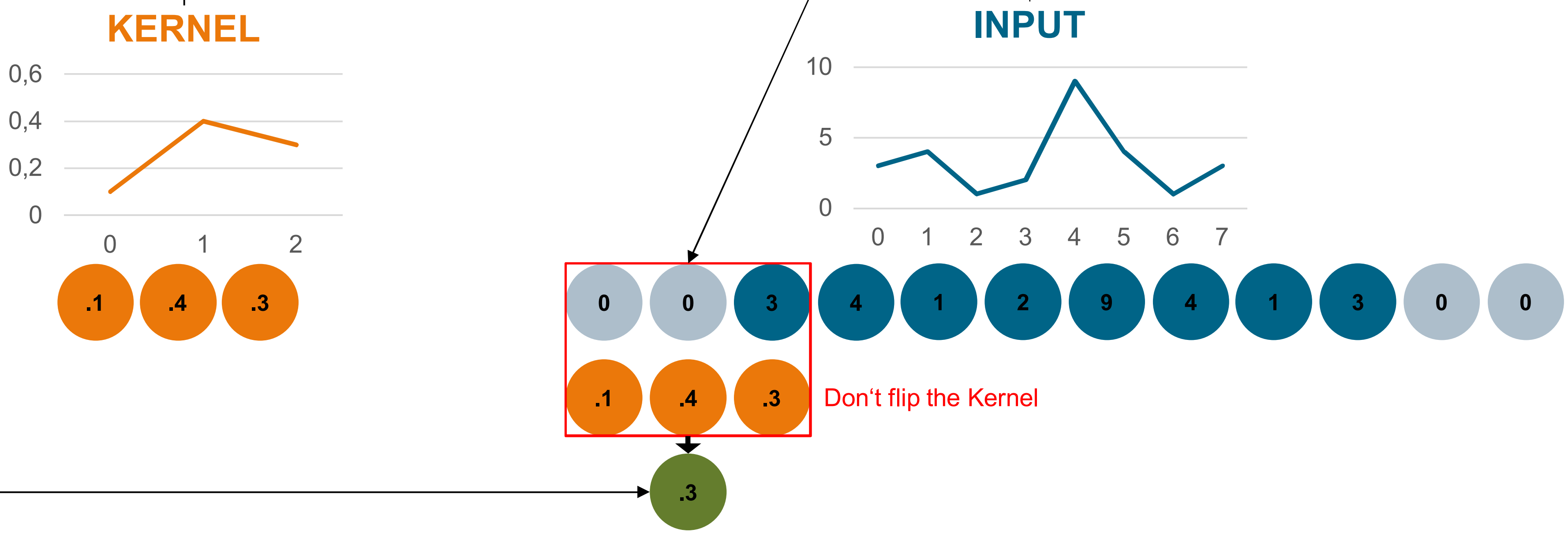


The 1D Cross-Correlation Operator

FEATURE MAP

KERNEL SIZE

$$s_i = (k * x)_i = \sum_{m=0}^{M-1} \boxed{k_m x_{i+m-(M-1)}}$$



The Convolution Operator in Deep Learning

Most Machine Learning libraries implement cross-correlation but call it convolution.

For the model, the difference does not matter!

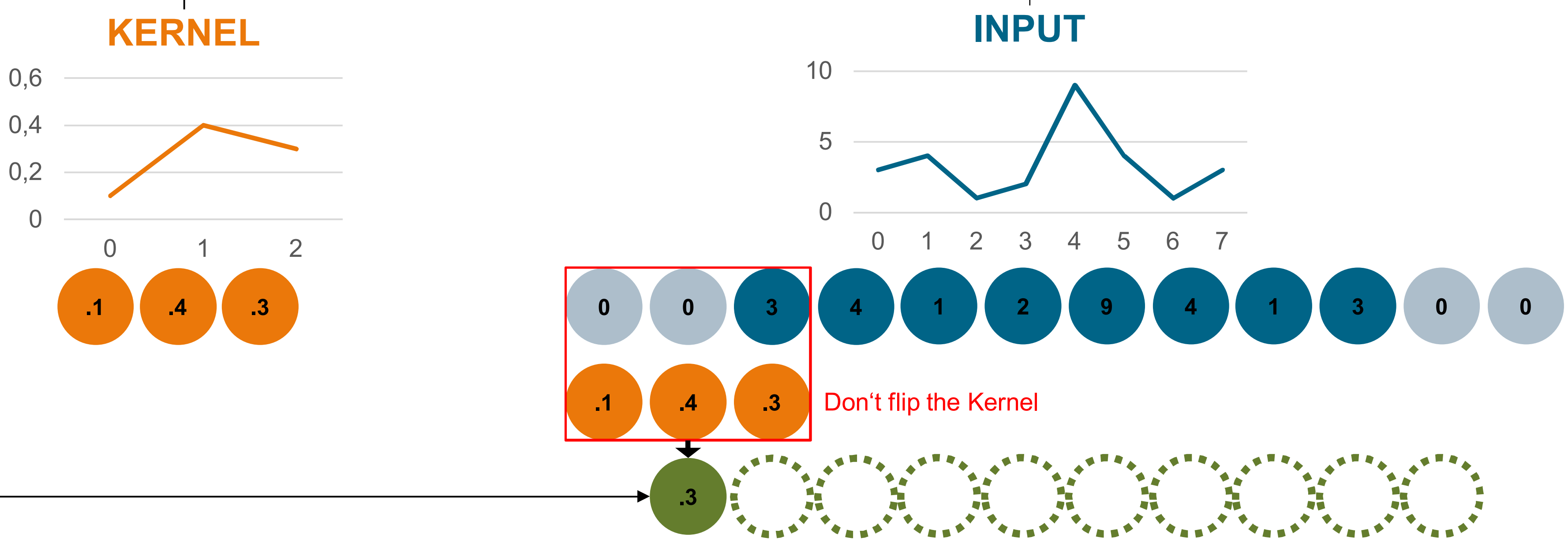
We will also use the term convolution in the following but we are actually doing cross-correlation.

Padding Modes

FEATURE MAP

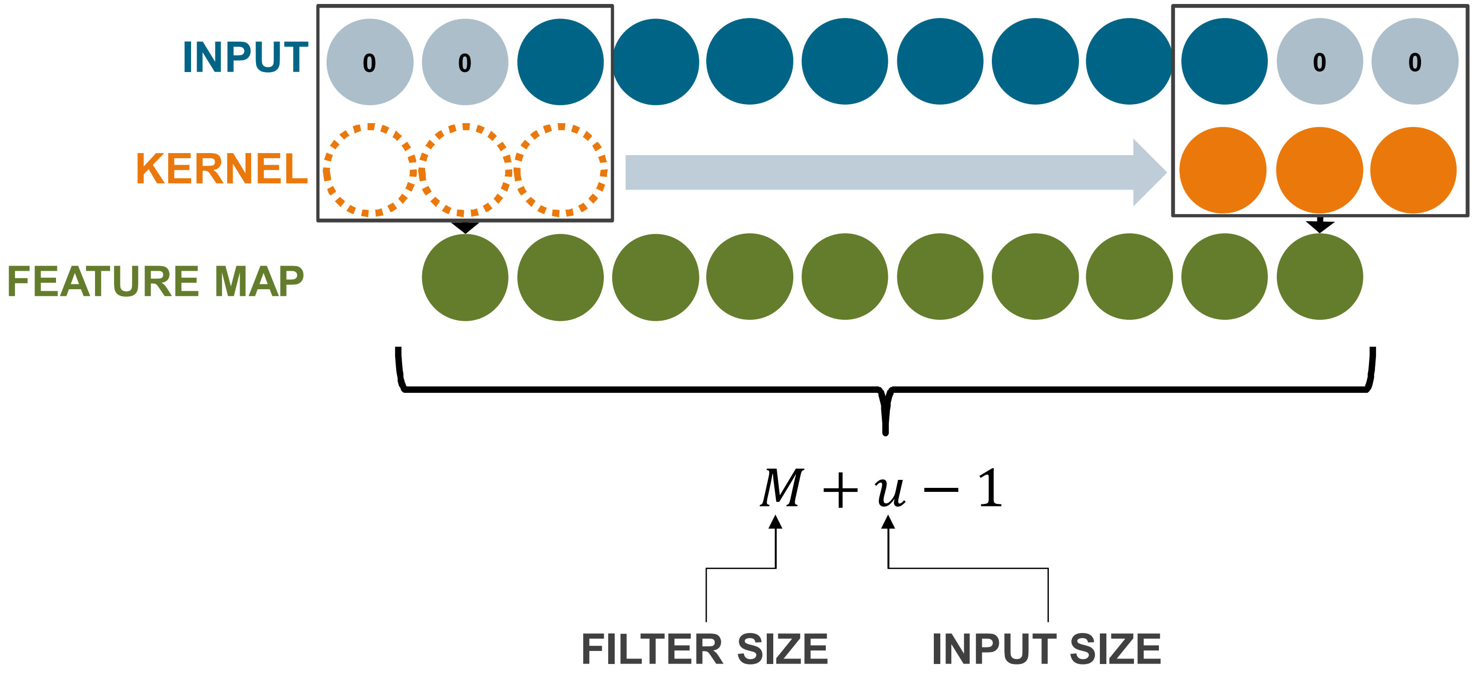
KERNEL SIZE

$$s_i = (k * x)_i = \sum_{m}^{M-1} k_m x_{i+m-(M-1)}$$

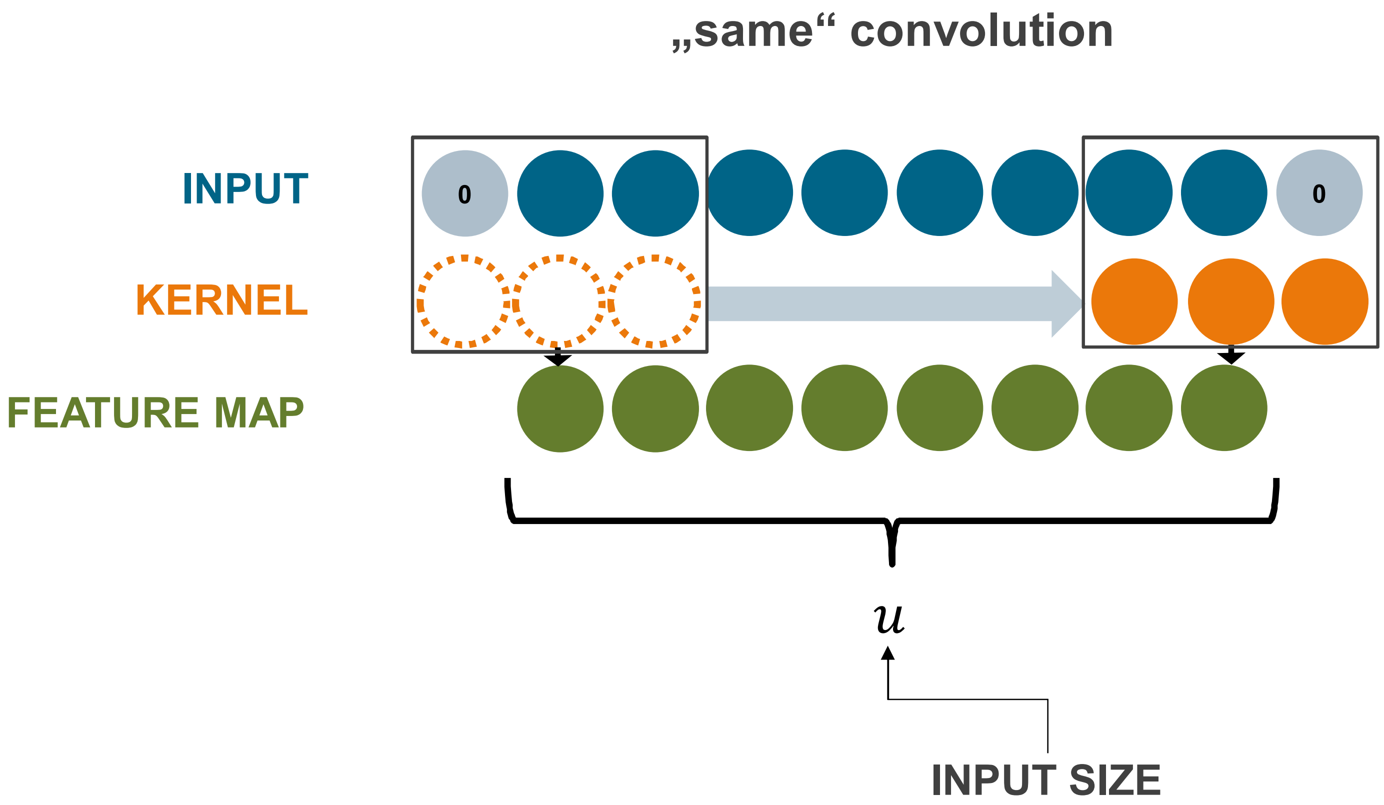


Padding Modes

„full“ convolution

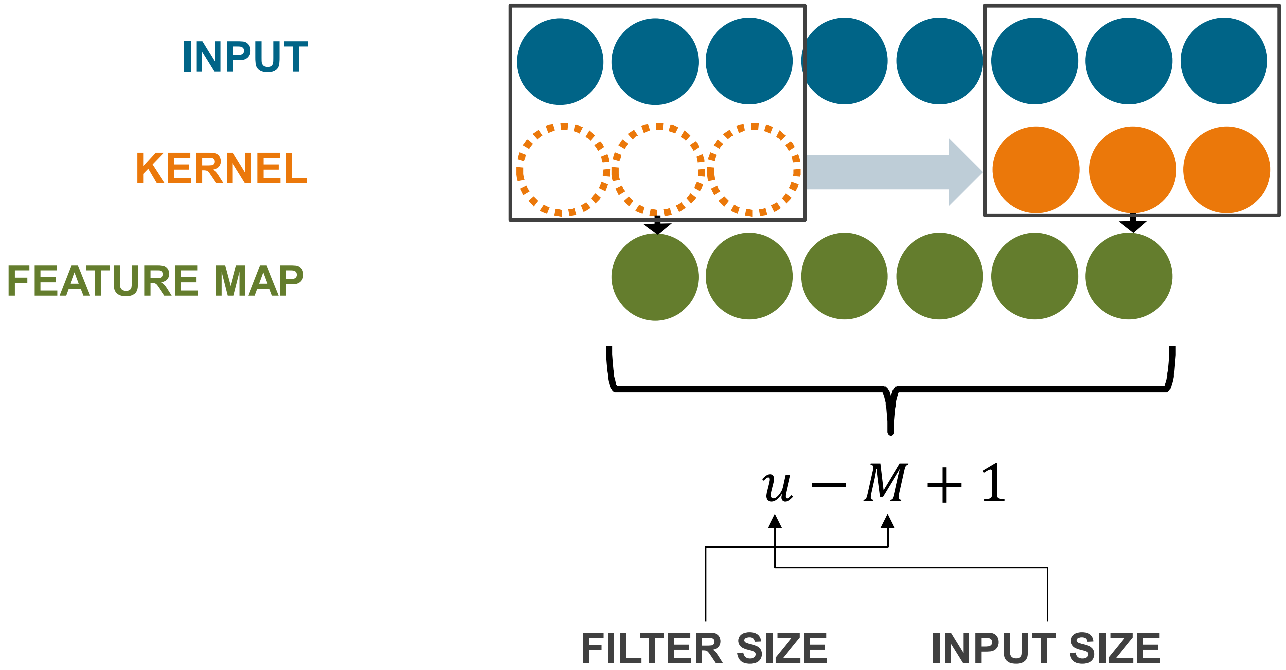


Padding Modes

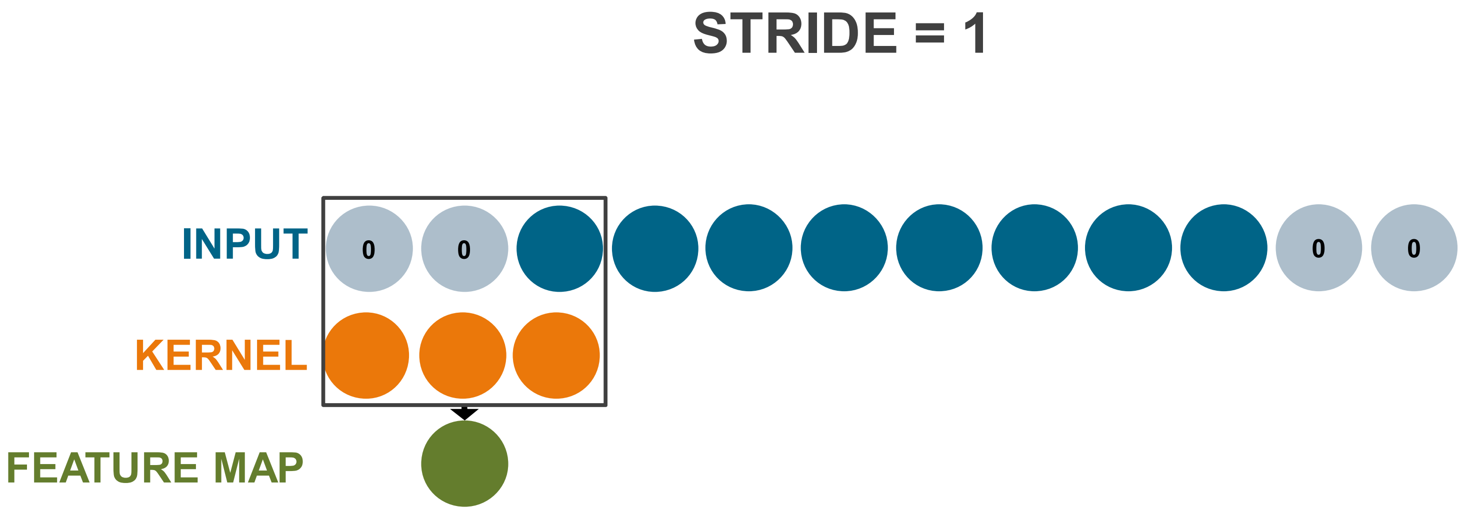


Padding Modes

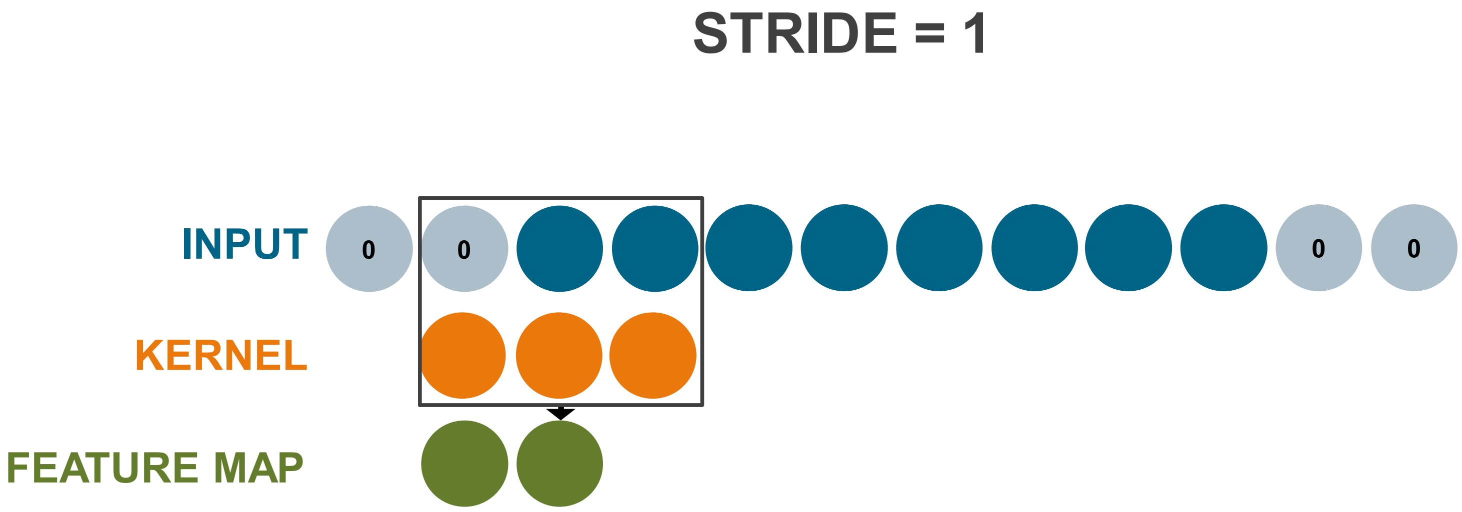
„valid“ convolution



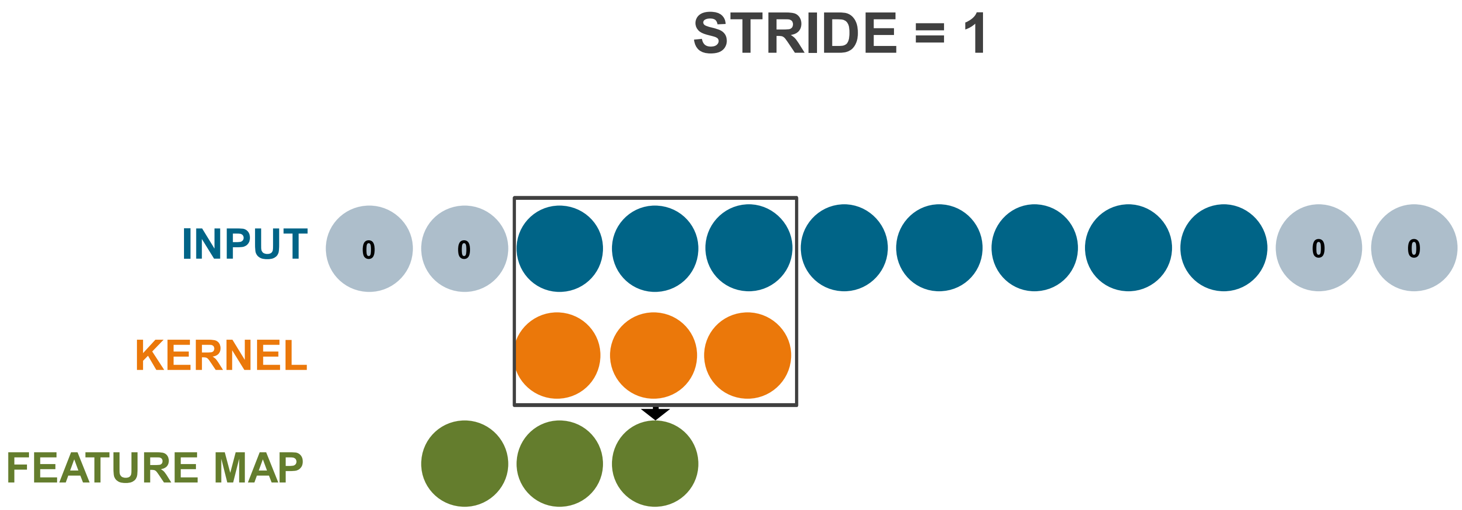
Strided Convolution



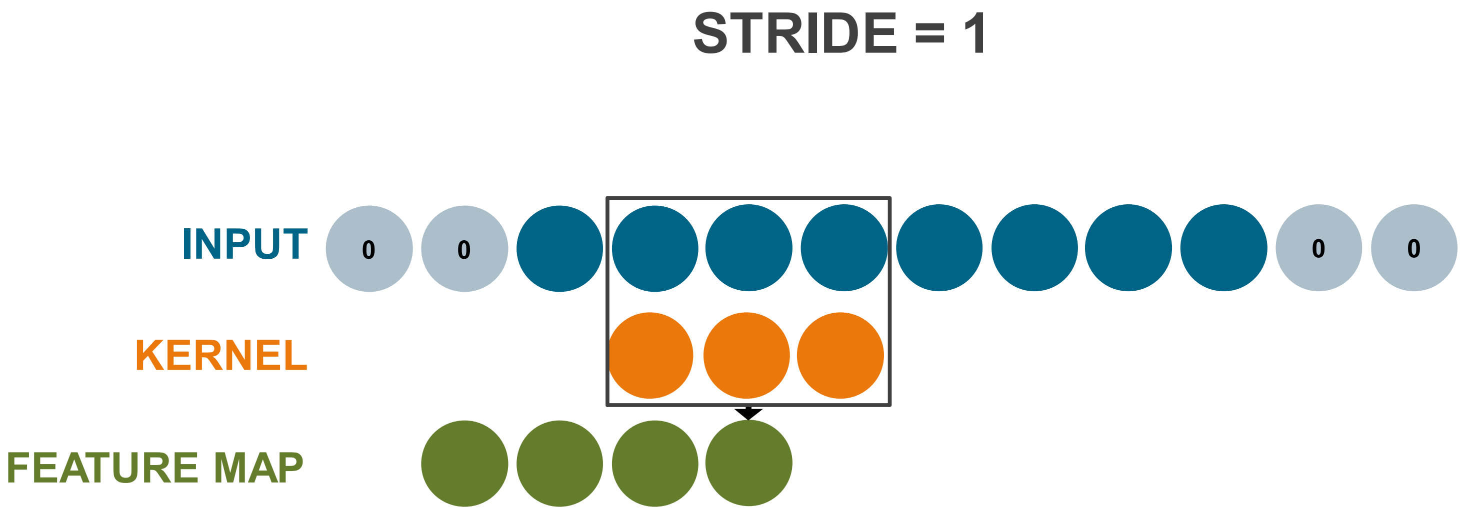
Strided Convolution



Strided Convolution

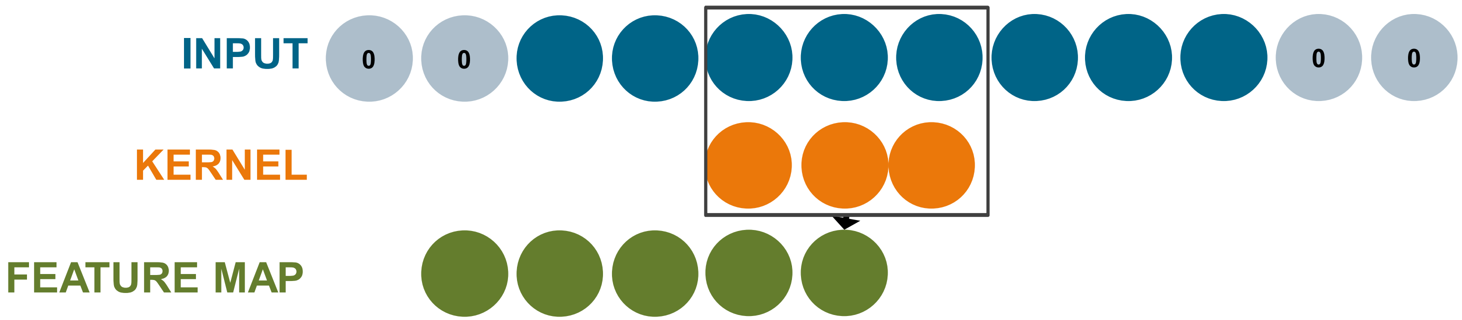


Strided Convolution



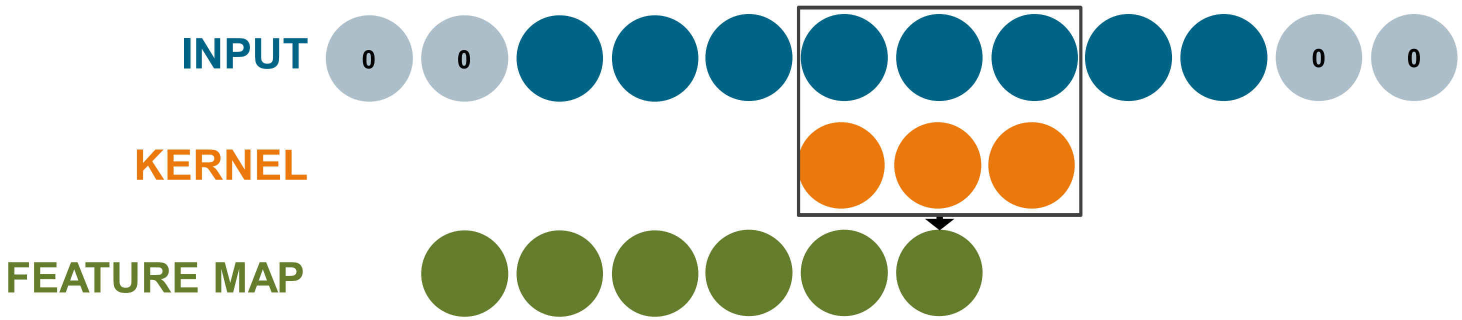
Strided Convolution

STRIDE = 1

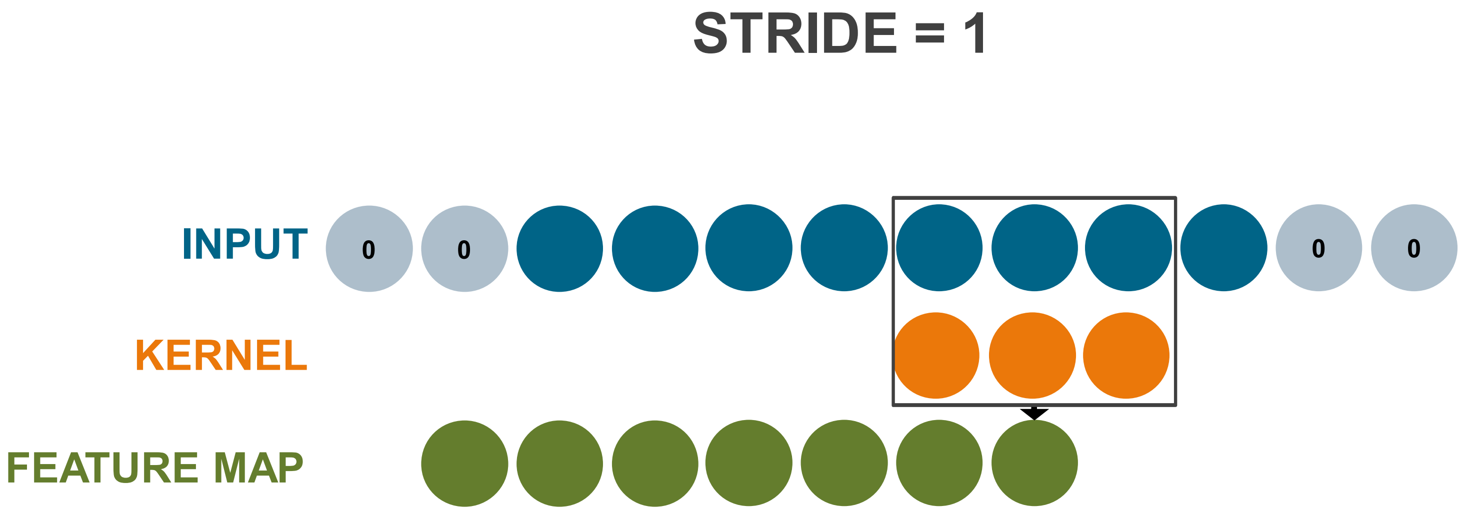


Strided Convolution

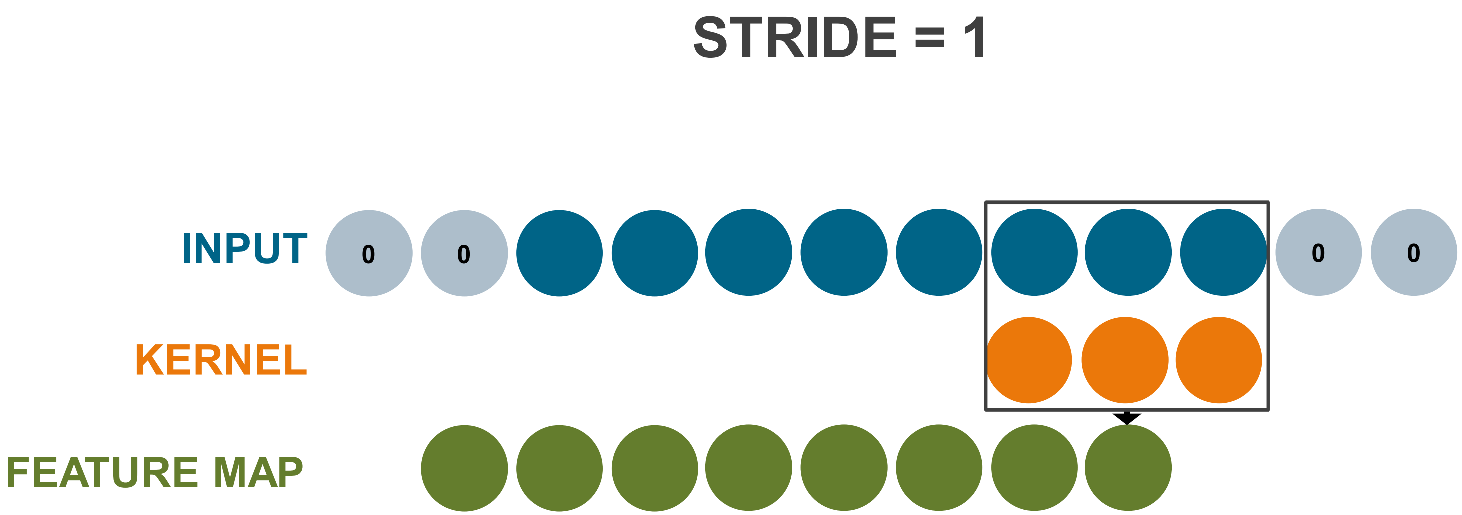
STRIDE = 1



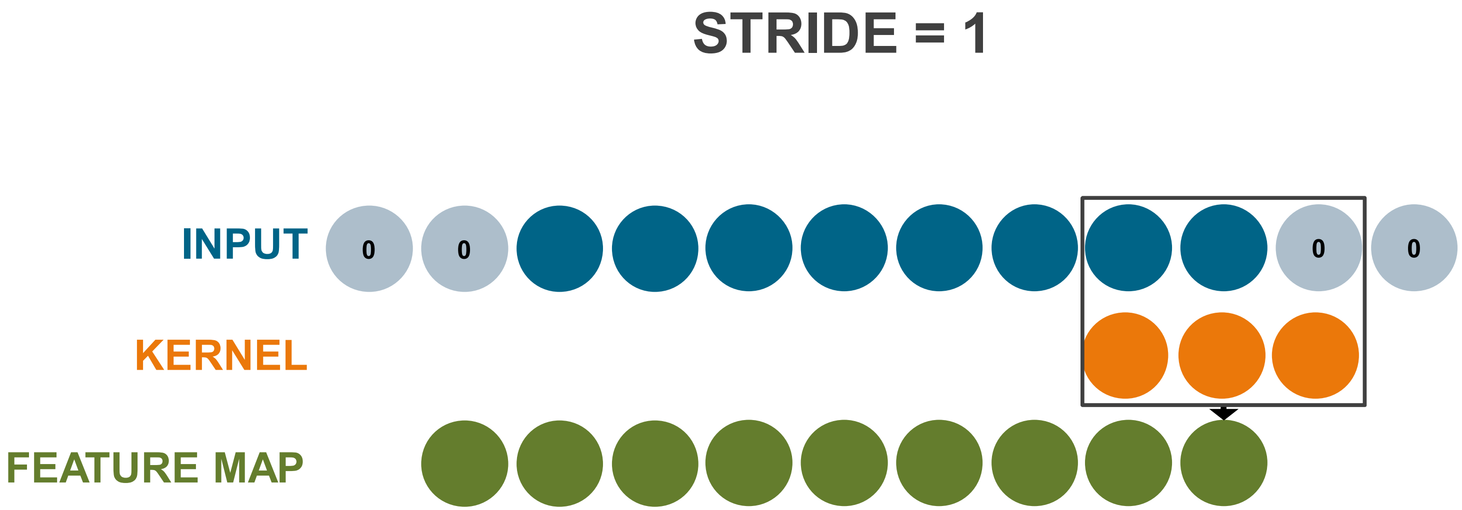
Strided Convolution



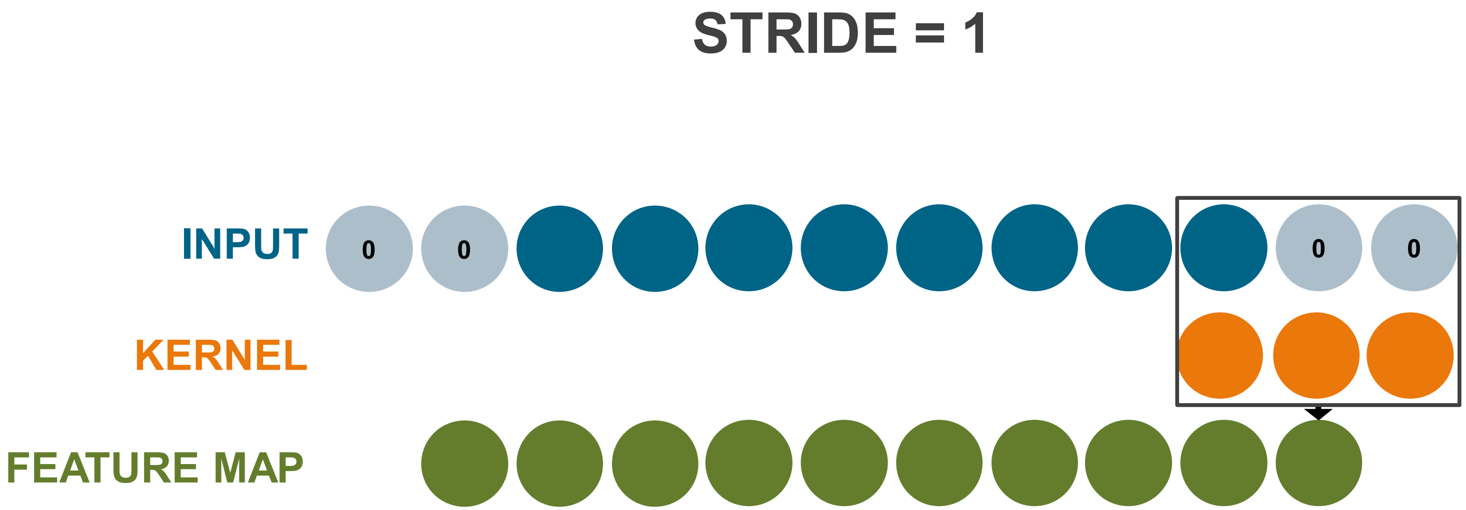
Strided Convolution



Strided Convolution

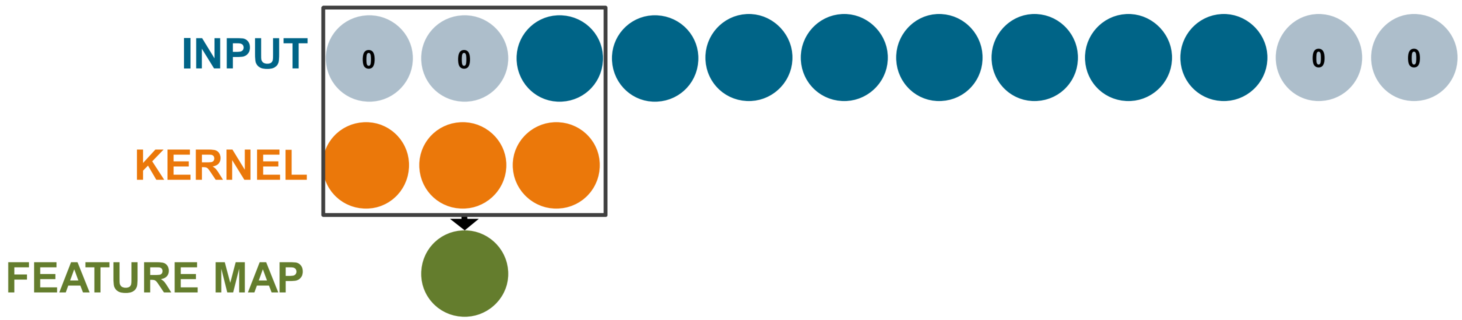


Strided Convolution

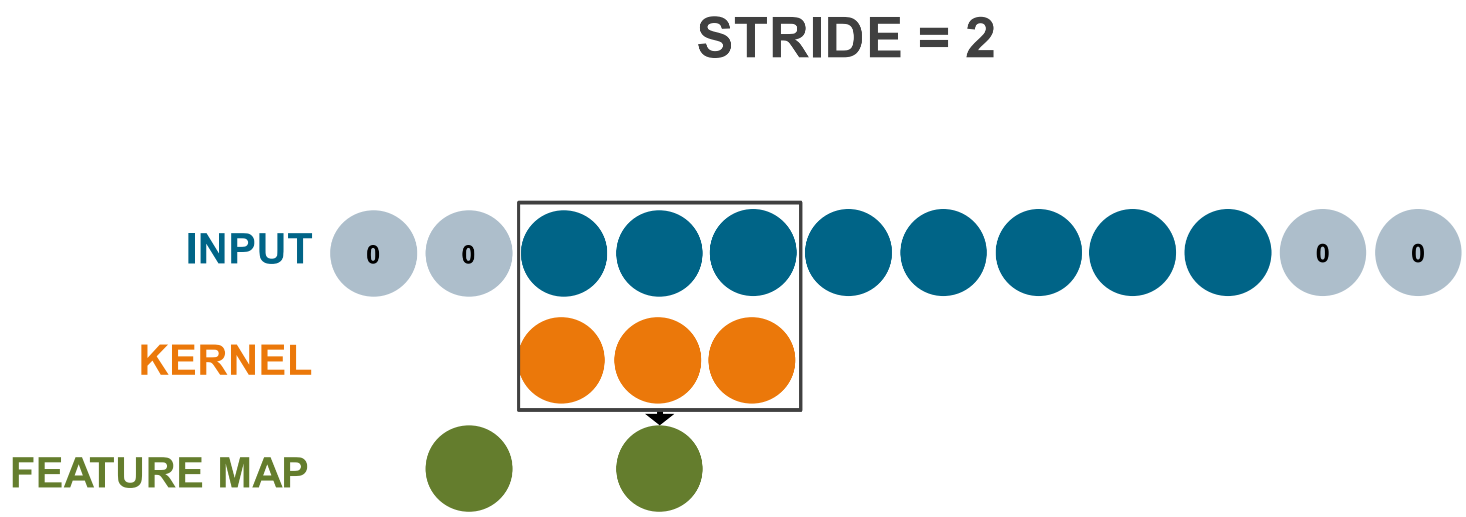


Strided Convolution

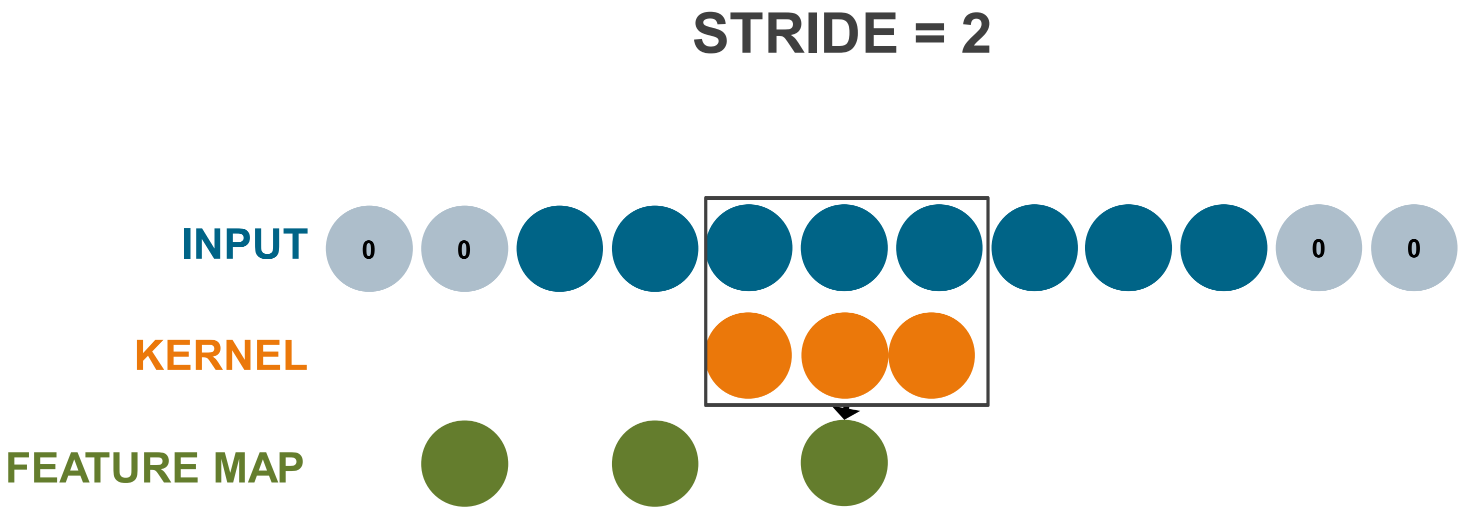
STRIDE = 2



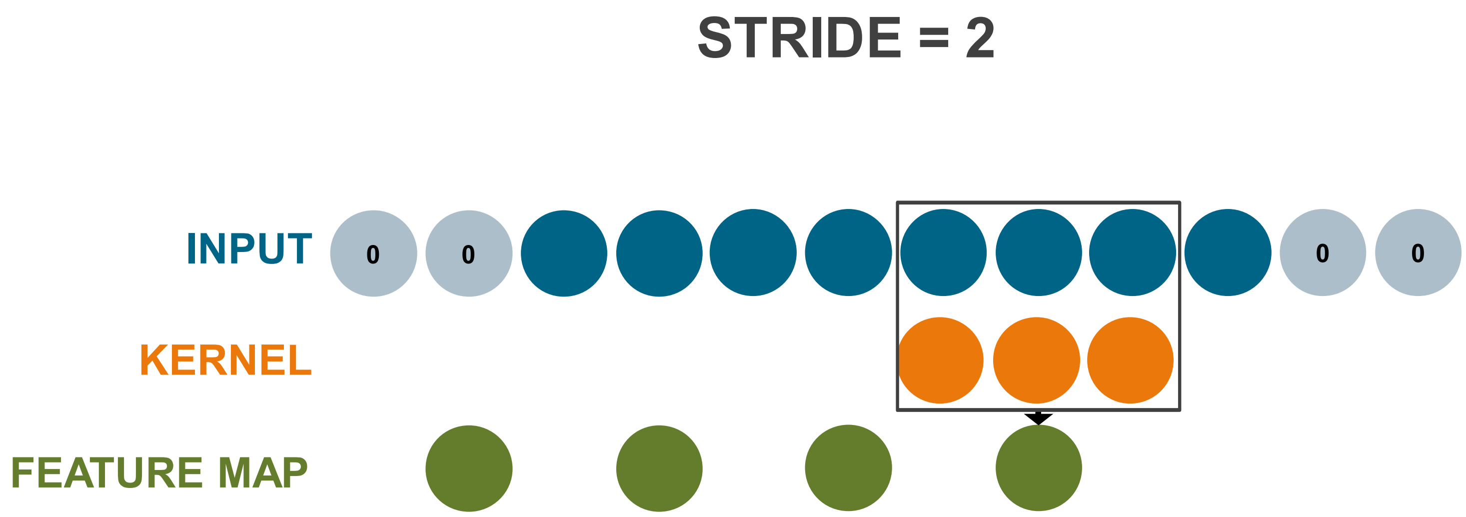
Strided Convolution



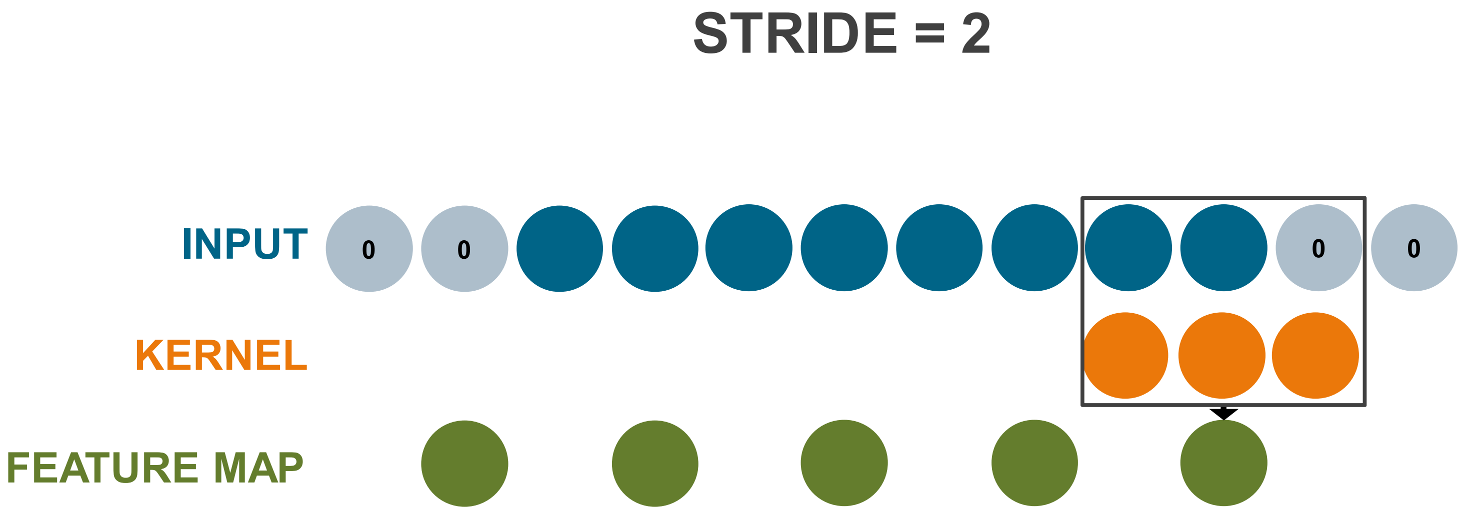
Strided Convolution



Strided Convolution



Strided Convolution



concatenate
Conv1D
Conv2D
Conv2DTranspose
Conv3D
Conv3DTranspose
ConvLSTM2D
Cropping1D
Cropping2D
Cropping3D
CuDNNGRU
CuDNNLSTM
Dense
DepthwiseConv2D
Dot
dot
Dropout
ELU
Embedding
Flatten
GaussianDropout
GaussianNoise
GlobalAveragePooling1D
GlobalAveragePooling2D

When using this layer as the first layer in a model, provide an `input_shape` argument (tuple of integers or `None`, e.g. `(10, 128)` for sequences of 10 vectors of 128-dimensional vectors, or `(None, 128)` for variable-length sequences of 128-dimensional vectors.

Arguments:

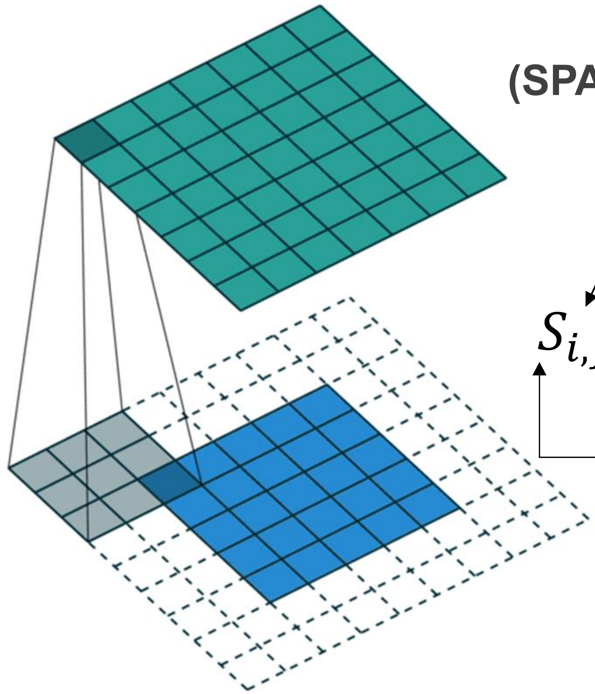
- `filters` : Integer, the dimensionality of the output space (i.e. the number of output filters in the convolution).
- `kernel_size` : An integer or tuple/list of a single integer, specifying the length of the 1D convolution window.
- `strides` : An integer or tuple/list of a single integer, specifying the stride length of the convolution. Specifying any stride value != 1 is incompatible with specifying any `dilation_rate` value != 1.
- `padding` : One of `"valid"`, `"causal"` or `"same"` (case-insensitive). `"causal"` results in causal (dilated) convolutions, e.g. output[t] does not depend on input[t+1:]. Useful when modeling temporal data where the model should not violate the temporal order. See WaveNet: A Generative Model for Raw Audio, section 2.1.
- ~~`data_format` : A string, one of `channels_last` (default) or `channels_first`~~.
- `dilation_rate` : an integer or tuple/list of a single integer, specifying the dilation rate to use for dilated convolution. Currently, specifying any `dilation_rate` value != 1 is incompatible with specifying any `strides` value != 1.
- `activation` : Activation function to use. If you don't specify anything, no activation is applied (ie. "linear" activation: `a(x) = x`).
- `use_bias` : Boolean, whether the layer uses a bias vector.
- `kernel_initializer` : Initializer for the `kernel` weights matrix.
- `bias_initializer` : Initializer for the bias vector.

2D Convolution

FEATURE MAP

KERNEL

INPUT



„full“ convolution

(SPATIAL) INDEX

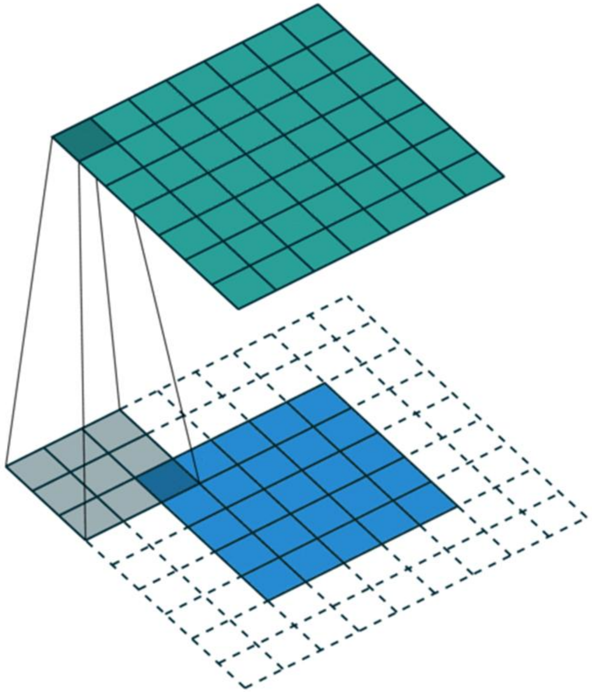
KERNEL

$$S_{i,j} = (K * I)_{i,j} = \sum_{n=0}^{N-1} \sum_{m=0}^{M-1} K_{m,n} I_{i+m-(M-1),j+n-(N-1)}$$

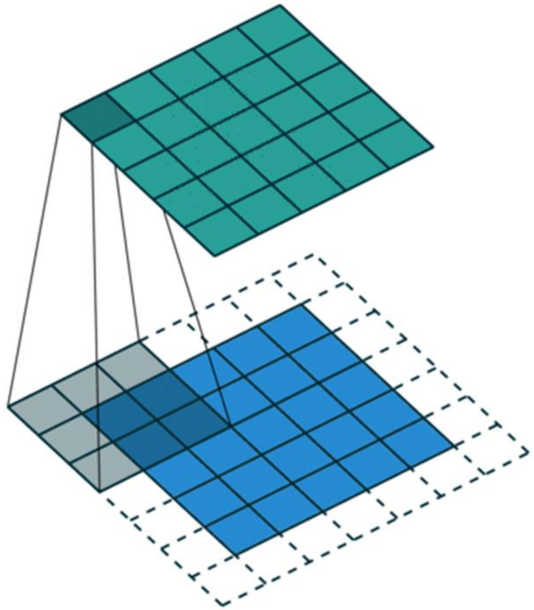
FEATURE MAP INPUT

Animations taken from: http://deeplearning.net/software/theano/tutorial/conv_arithmetic.html

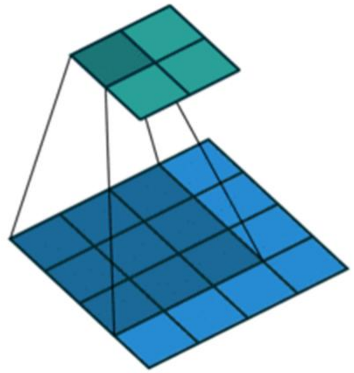
2D Convolution



„full“ convolution



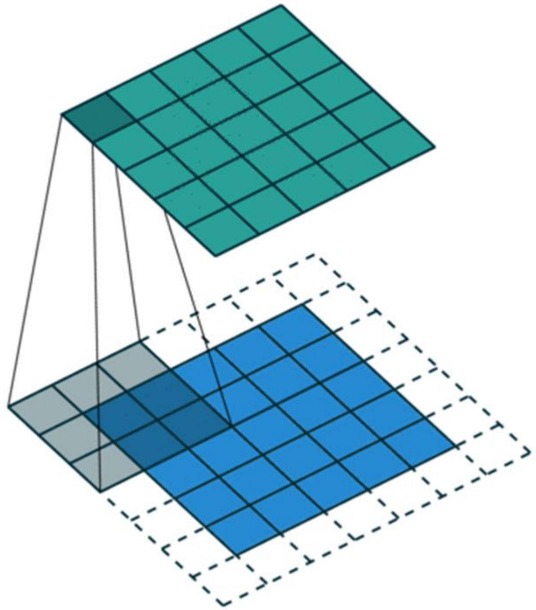
„same“ convolution



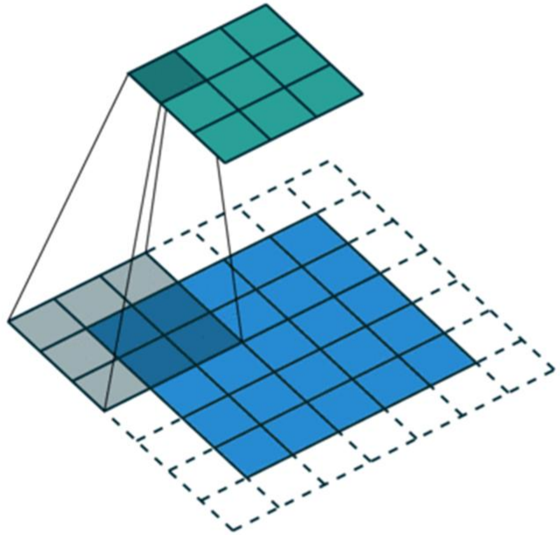
„valid“ convolution

Animations taken from: http://deeplearning.net/software/theano/tutorial/conv_arithmetic.html

2D Convolution



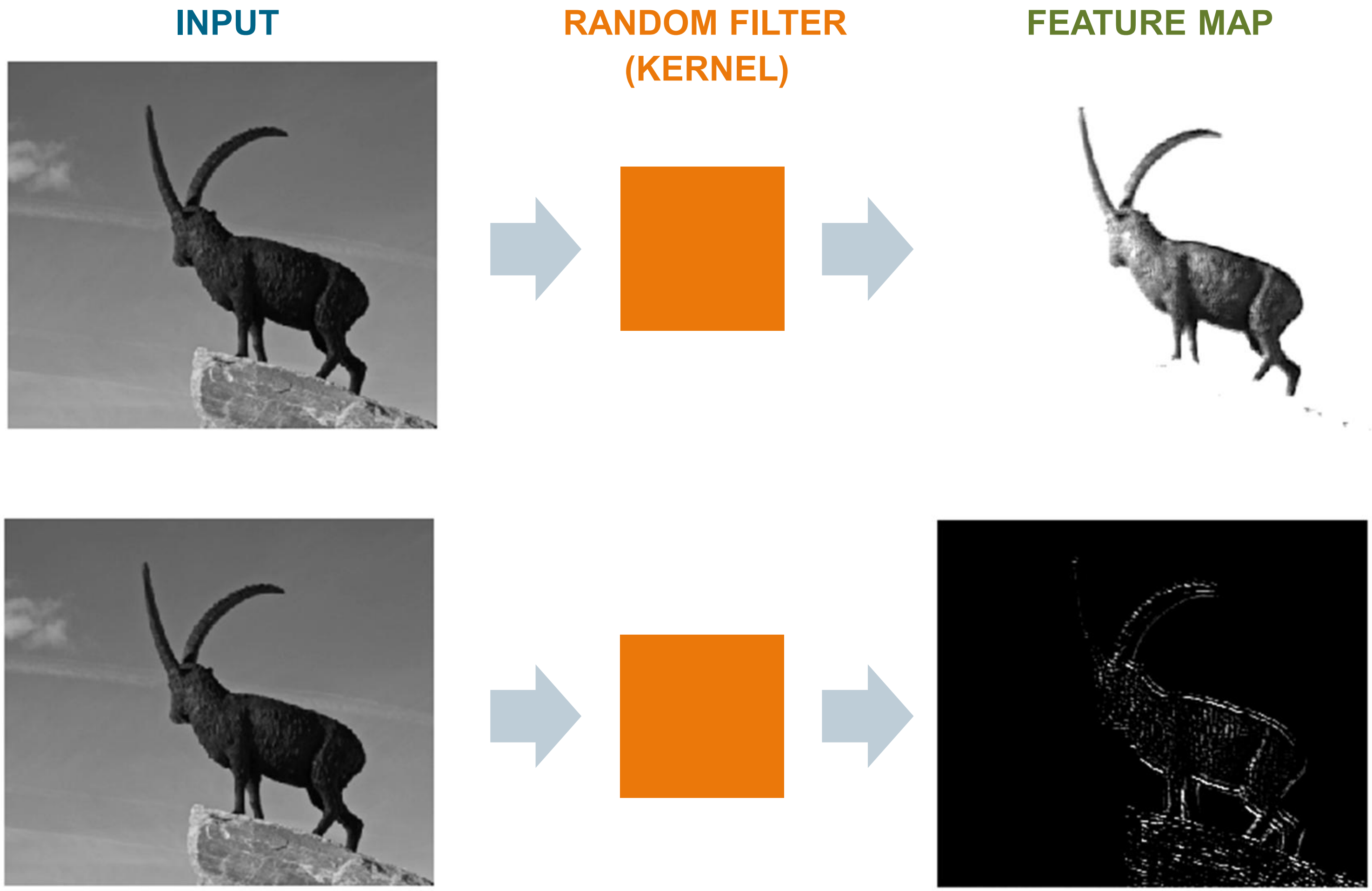
STRIDE = [1, 1]



STRIDE = [2, 2]

Animations taken from: http://deeplearning.net/software/theano/tutorial/conv_arithmetic.html

Convolutional Layer

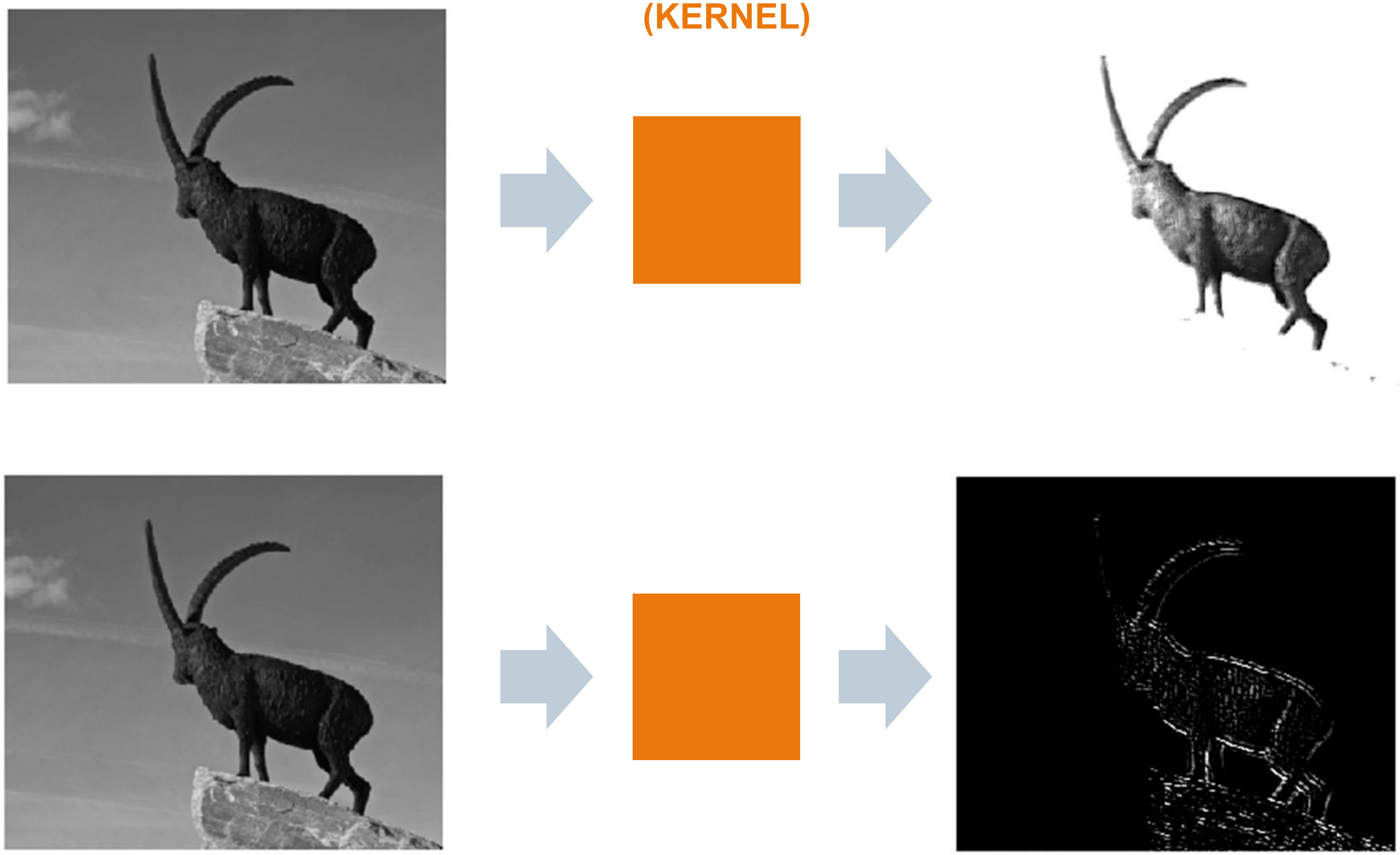


Convolutional Neural Network Layer

INPUT

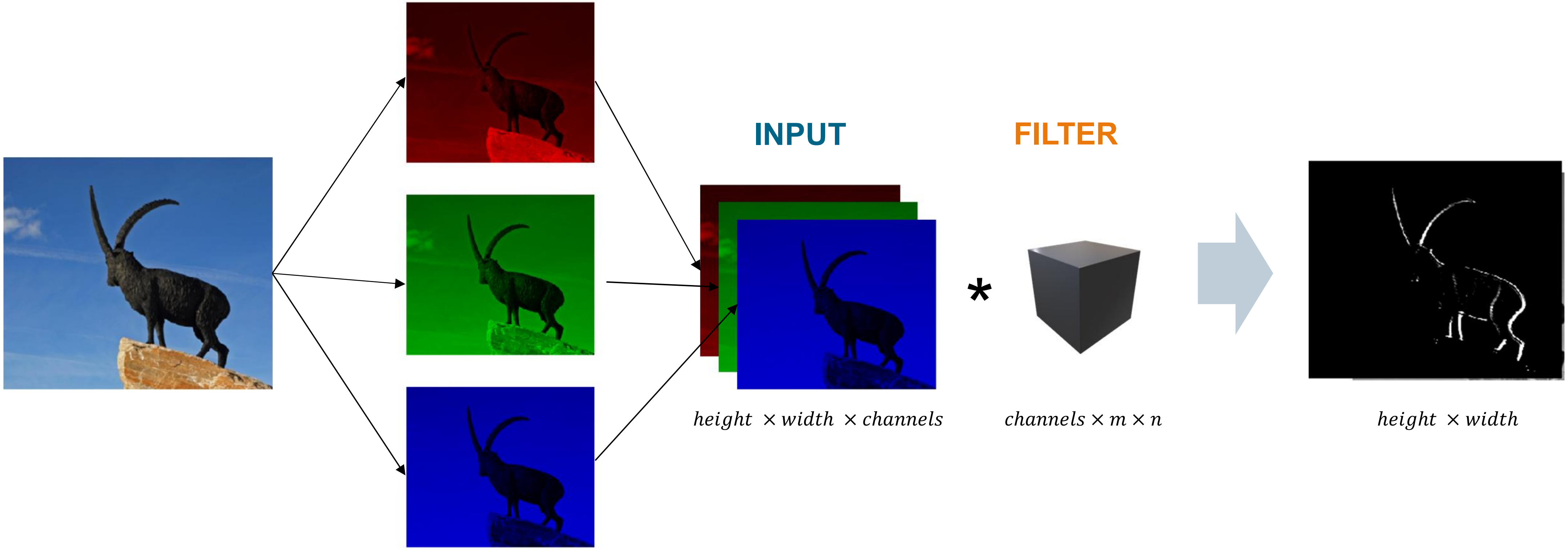
LEARNED FILTER (KERNEL)

FEATURE MAP

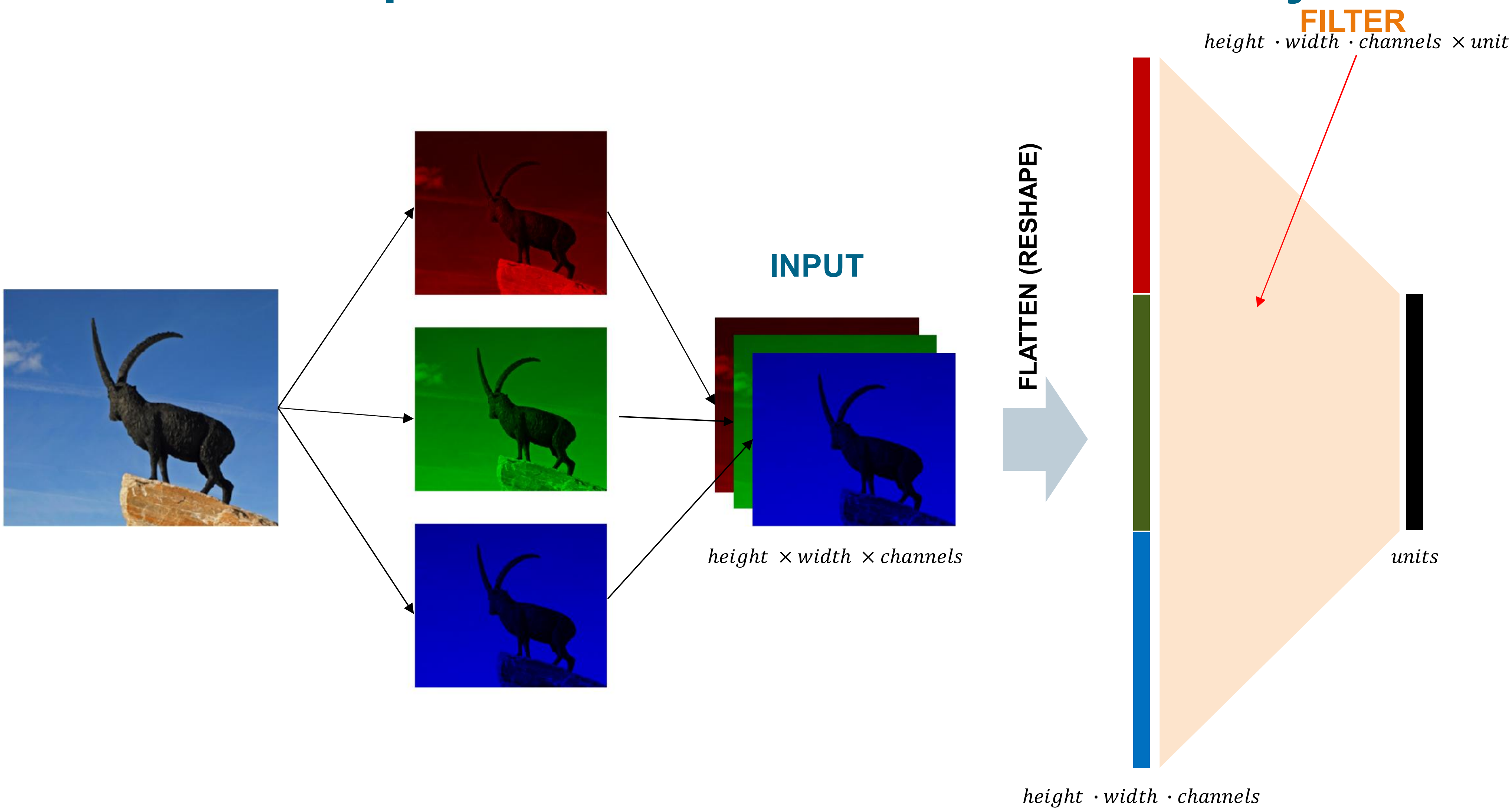


In a Convolutional Neural Network Layer we learn the Kernels.

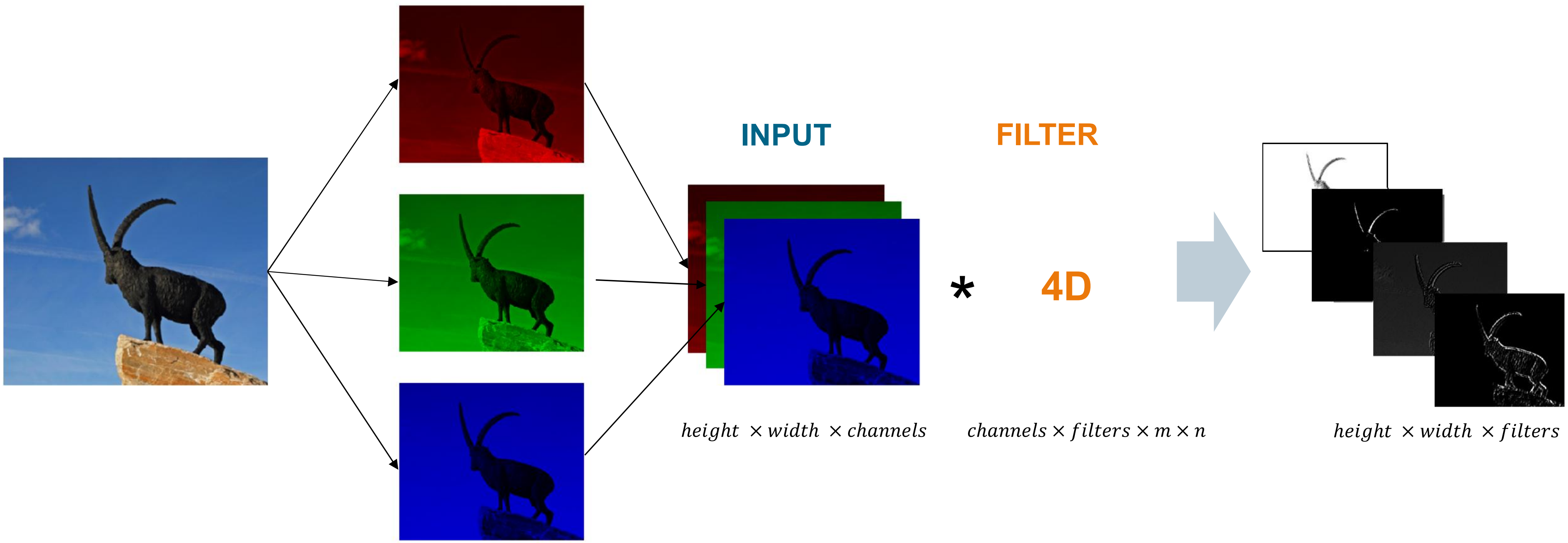
2D Convolutional Neural Network Layer



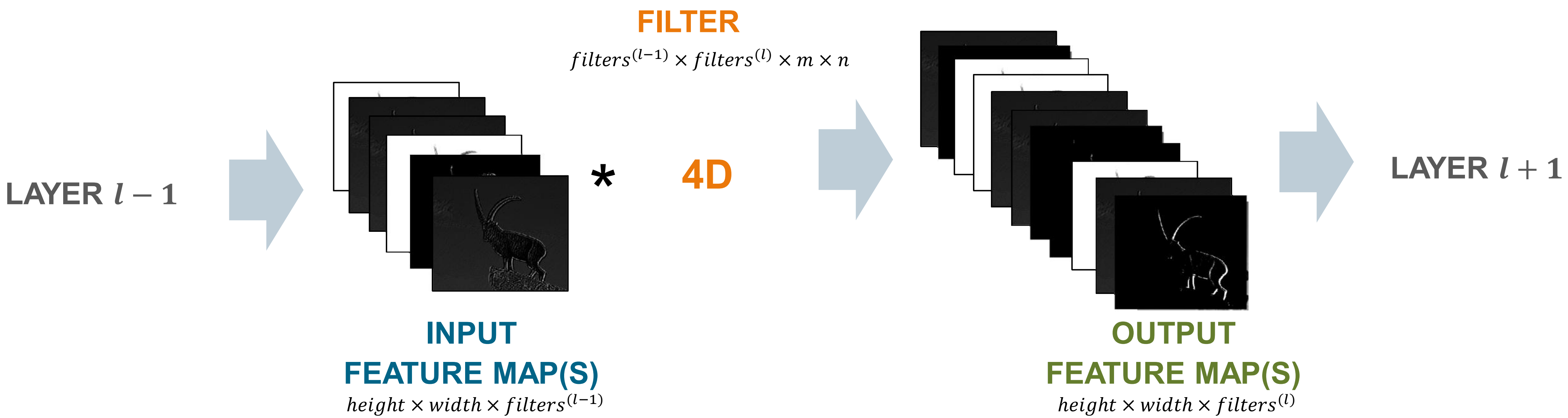
Comparison: Dense Neural Network Layer



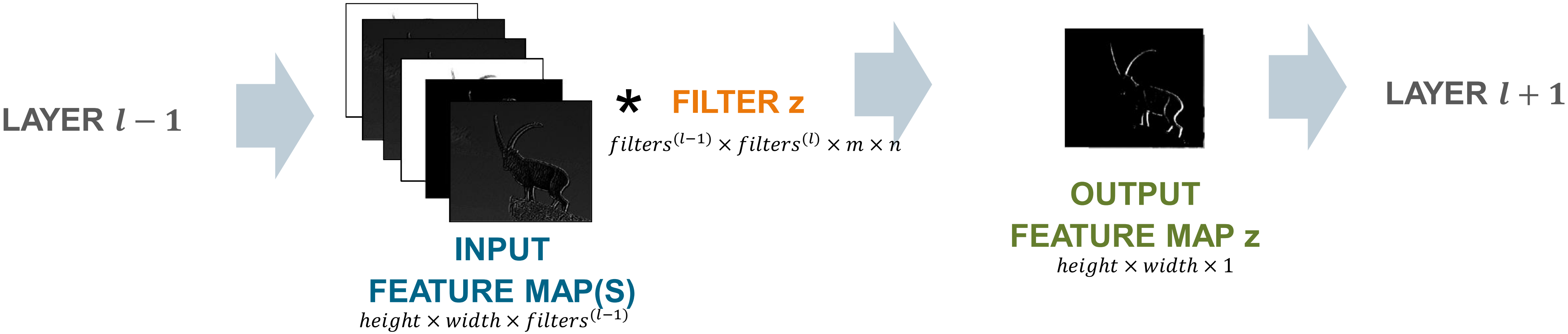
2D Convolutional Neural Network Layer



2D Convolutional Neural Network Layer

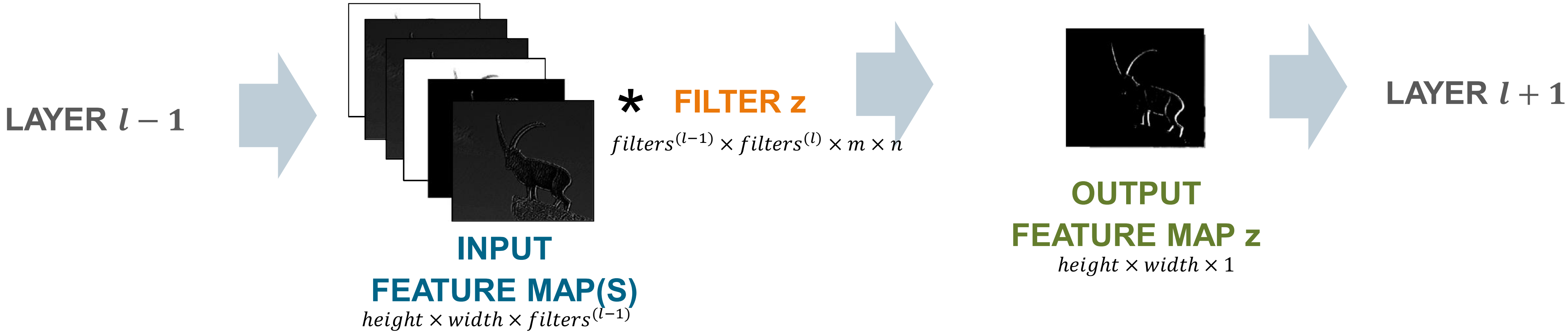


2D Convolutional Neural Network Layer



$$h_{z,i,j}^{(l+1)} = \varphi\left(W_z^{(l)} * H^{(l)} + b_z^{(l)}\right)_{i,j} = \varphi\left(\sum_c \sum_m \sum_n w_{c,z,m,n}^{(l)} h_{i+m-\underbrace{M\backslash 2,j+n-N\backslash 2}_{\text{SAME PADDING}},c}^{(l)} + b_z^{(l)}\right)$$

2D Convolutional Neural Network Layer

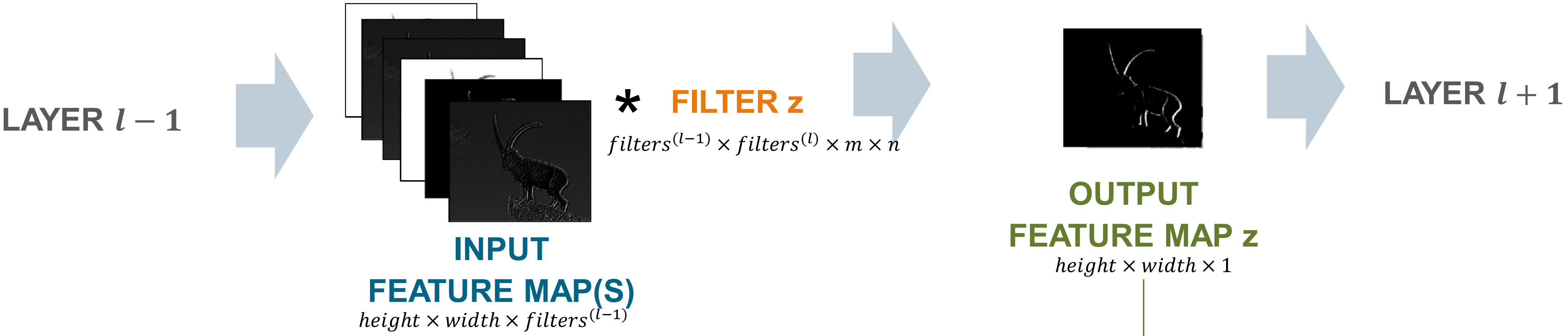


$$h_{z,i,j}^{(l+1)} = \varphi\left(W_z^{(l)} * H^{(l)} + b_z^{(l)}\right)_{i,j} = \varphi\left(\sum_c \sum_m \sum_n w_{c,z,m,n}^{(l)} h_{i+m-M\backslash 2,\, j+n-N\backslash 2,\, c}^{(l)} + b_z^{(l)}\right)$$

ACTIVATION FUNCTION

BIAS

2D Convolutional Neural Network Layer

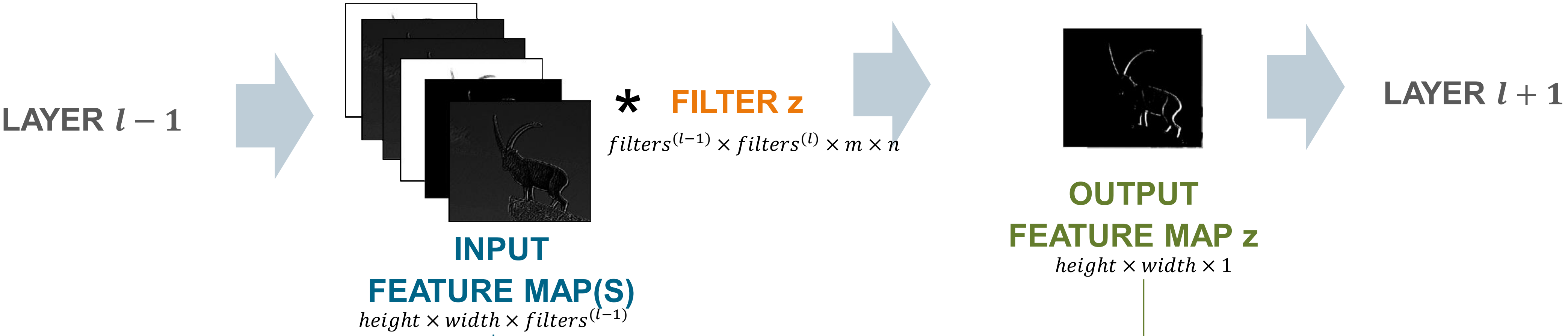


$$h_{z,i,j}^{(l+1)} = \varphi\left(W_z^{(l)} * H^{(l)} + b_z^{(l)}\right)_{i,j} = \varphi\left(\sum_c \sum_m \sum_n w_{c,z,m,n}^{(l)} h_{i+m-M\backslash 2,\, j+n-N\backslash 2,\, c}^{(l)} + b_z^{(l)}\right)$$

ACTIVATION FUNCTION

BIAS

2D Convolutional Neural Network Layer

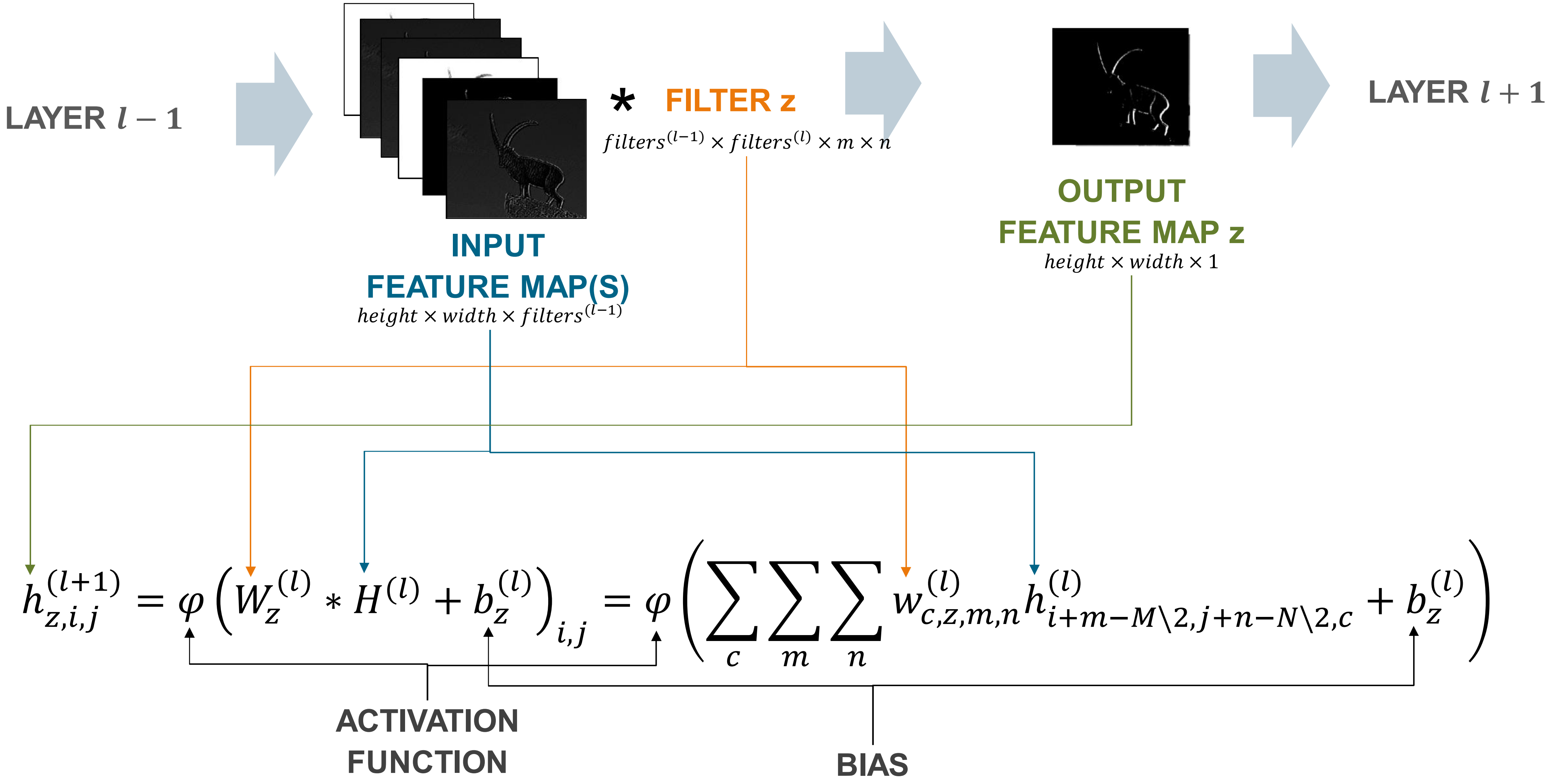


$$h_{z,i,j}^{(l+1)} = \varphi\left(W_z^{(l)} * H^{(l)} + b_z^{(l)}\right)_{i,j} = \varphi\left(\sum_c \sum_m \sum_n w_{c,z,m,n}^{(l)} h_{i+m-M\backslash 2,\, j+n-N\backslash 2,\, c}^{(l)} + b_z^{(l)}\right)$$

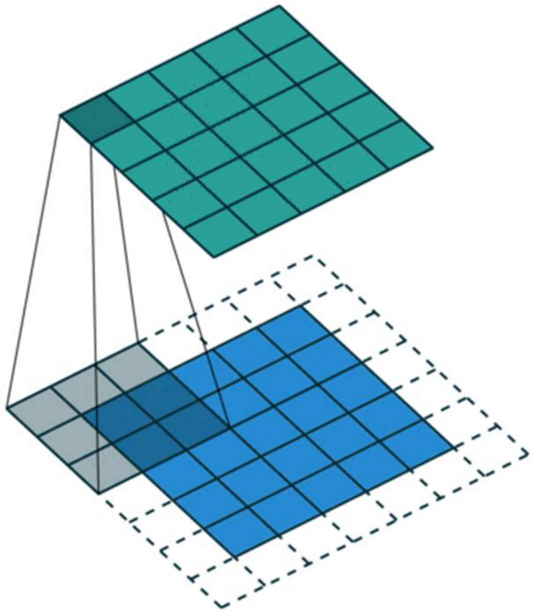
ACTIVATION FUNCTION

BIAS

2D Convolutional Neural Network Layer



Efficiency



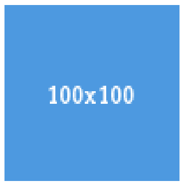
Convolutional layer:

- Exploits neighborhood relations of the inputs (e.g. spatial).
- Applies small fully connected layers to small patches of the input.
  - ➢Very efficient!
  - ➢Weight sharing
  - ➢Number of free parameters

    $$\#\text{input channels} \times \text{filter height} \times \text{filter width} \times \#\text{filters}$$

- The receptive field can be increased by stacking multiple layers
- Should only be used if there is a notion of neighborhood in the input:
  - Text, images, sensor time-series, videos, …

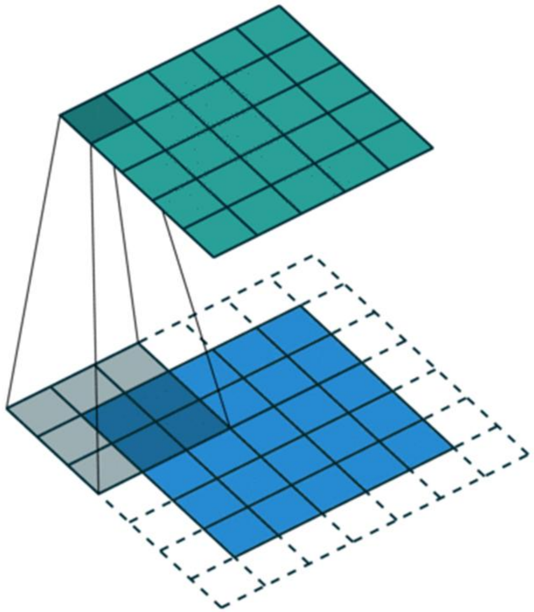
Example:



RGB image of shape
100 x 100 x 3

2,700 free parameters for a convolutional layer with 100 hidden units (filters) with a filter size of 3 x 3!

Implementation



```python
import numpy as np
import tensorflow as tf


# Define placeholder for input 24 x 24 rgb images.
input_images = tf.keras.Input(shape=(24, 24, 3))

# Apply a convolutional layer on the input images.
h = tf.keras.layers.Conv2D(
    filters=8, kernel_size=[3, 3], strides=[1, 1], activation=tf.nn.relu,
    padding='same')(input_images)

# Generate 10 random images which we will feed to the layer.
random_images = np.random.uniform(
    0, 1, size=(10, 24, 24, 3)).astype(np.float32)
with tf.Session() as session:
    # We need to initialize the layer parameters first.
    session.run(tf.global_variables_initializer())
    # Feed the network with the random images.
    output_feature_maps = session.run(
        h, feed_dict={input_images: random_images})
    print output_feature_maps.shape  # gives: (10, 24, 24, 8)
```

TensorFlow ™ Install Develop Community API r1.11 Ecosystem

PYTHON JAVASCRIPT C++ JAVA MORE...

When using this layer as the first layer in a model, provide an `input_shape` argument (tuple of integers or `None`, e.g. `(10, 128)` for sequences of 10 vectors of 128-dimensional vectors, or `(None, 128)` for variable-length sequences of 128-dimensional vectors.

Arguments:

- `filters`: Integer, the dimensionality of the output space (i.e. the number of output filters in the convolution).
- `kernel_size`: An integer or tuple/list of a single integer, specifying the length of the 1D convolution window.
- `strides`: An integer or tuple/list of a single integer, specifying the stride length of the convolution. Specifying any stride value != 1 is incompatible with specifying any `dilation_rate` value != 1.
- `padding`: One of `"valid"`, `"causal"` or `"same"` (case-insensitive). `"causal"` results in causal (dilated) convolutions, e.g. output[t] does not depend on input[t+1:]. Useful when modeling temporal data where the model should not violate the temporal order. See WaveNet: A Generative Model for Raw Audio, section 2.1.
- ~~`data_format`: A string, one of `channels_last` (default) or `channels_first`~~.
- `dilation_rate`: an integer or tuple/list of a single integer, specifying the dilation rate to use for dilated convolution. Currently, specifying any `dilation_rate` value != 1 is incompatible with specifying any `strides` value != 1.
- `activation`: Activation function to use. If you don't specify anything, no activation is applied (ie. "linear" activation: `a(x) = x`).
- `use_bias`: Boolean, whether the layer uses a bias vector.
- `kernel_initializer`: Initializer for the `kernel` weights matrix.
- `bias_initializer`: Initializer for the bias vector.

Convolutional Neural Networks

Layout of a Classic Convolutional Neural Network (CNN)

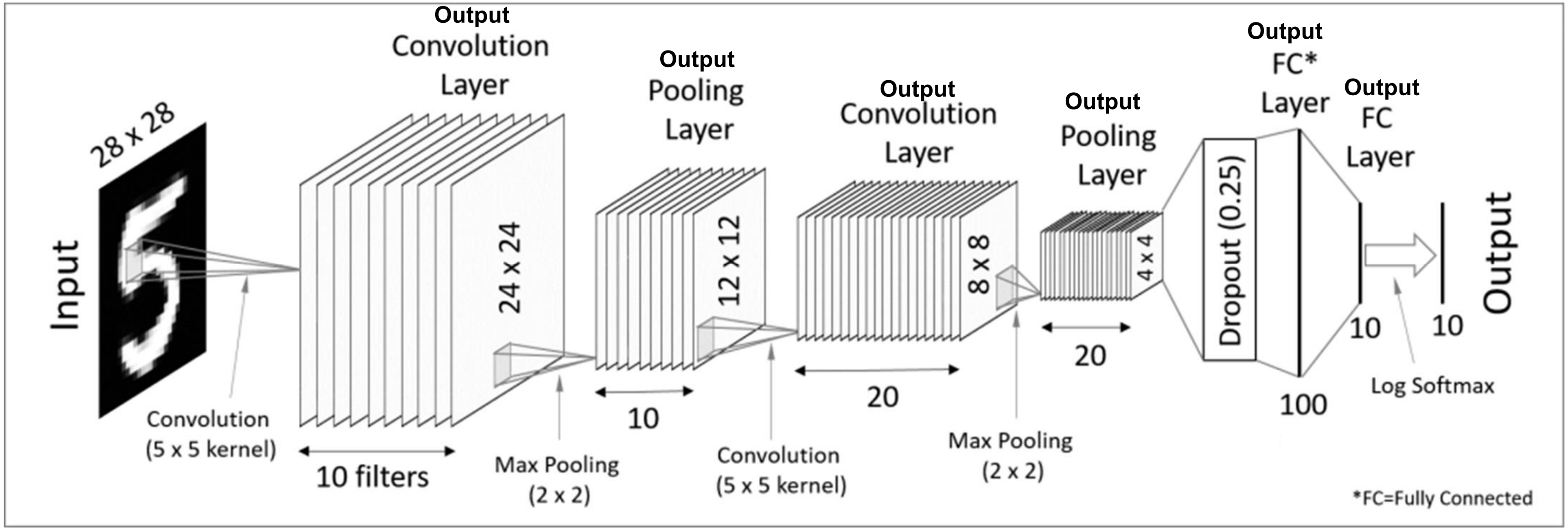


Image taken from: https://codetolight.wordpress.com/2017/11/29/getting-started-with-pytorch-for-deep-learning-part-3-neural-network-basics/

Image generated with: https://blueprints.creaidai.com/

Layout of a Classic Convolutional Neural Network (CNN)

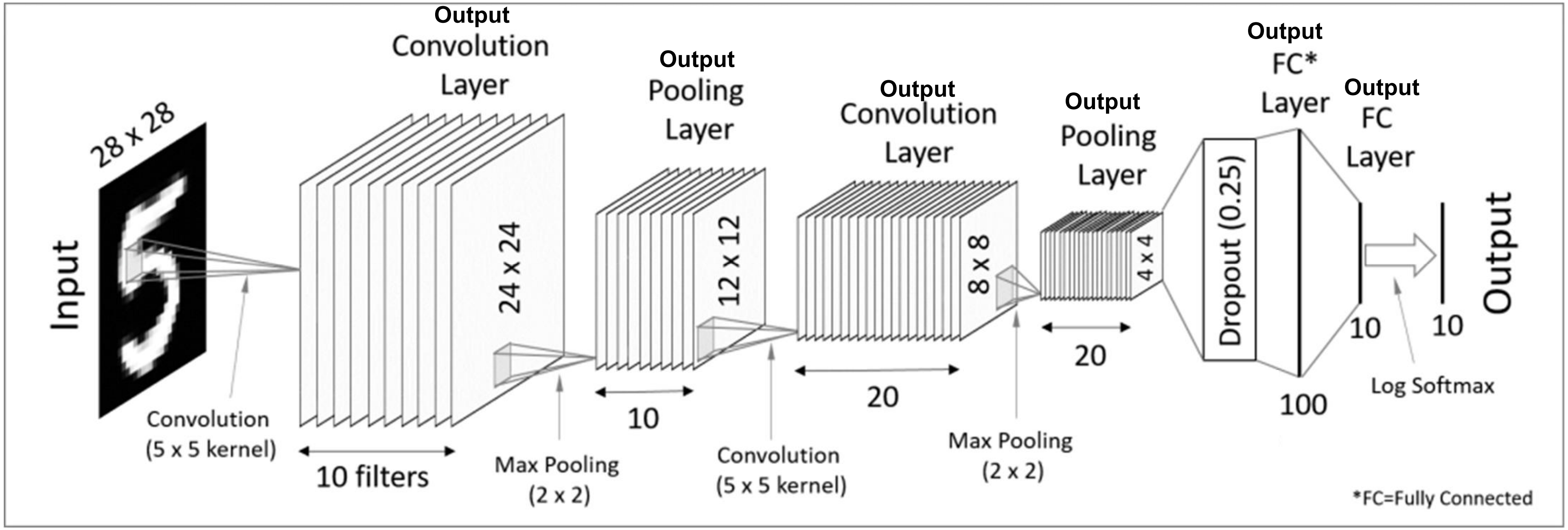


Image taken from: https://codetolight.wordpress.com/2017/11/29/getting-started-with-pytorch-for-deep-learning-part-3-neural-network-basics/

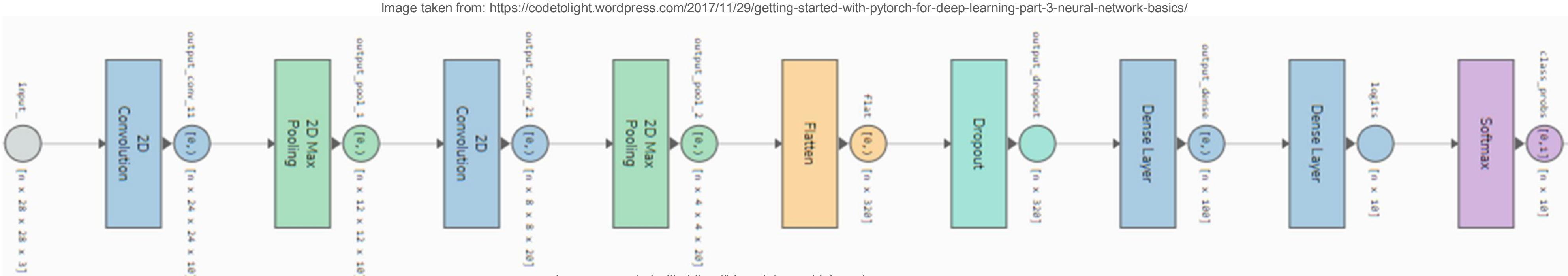


Image generated with: https://blueprints.creaidai.com/

Layout of a Classic Convolutional Neural Network (CNN)

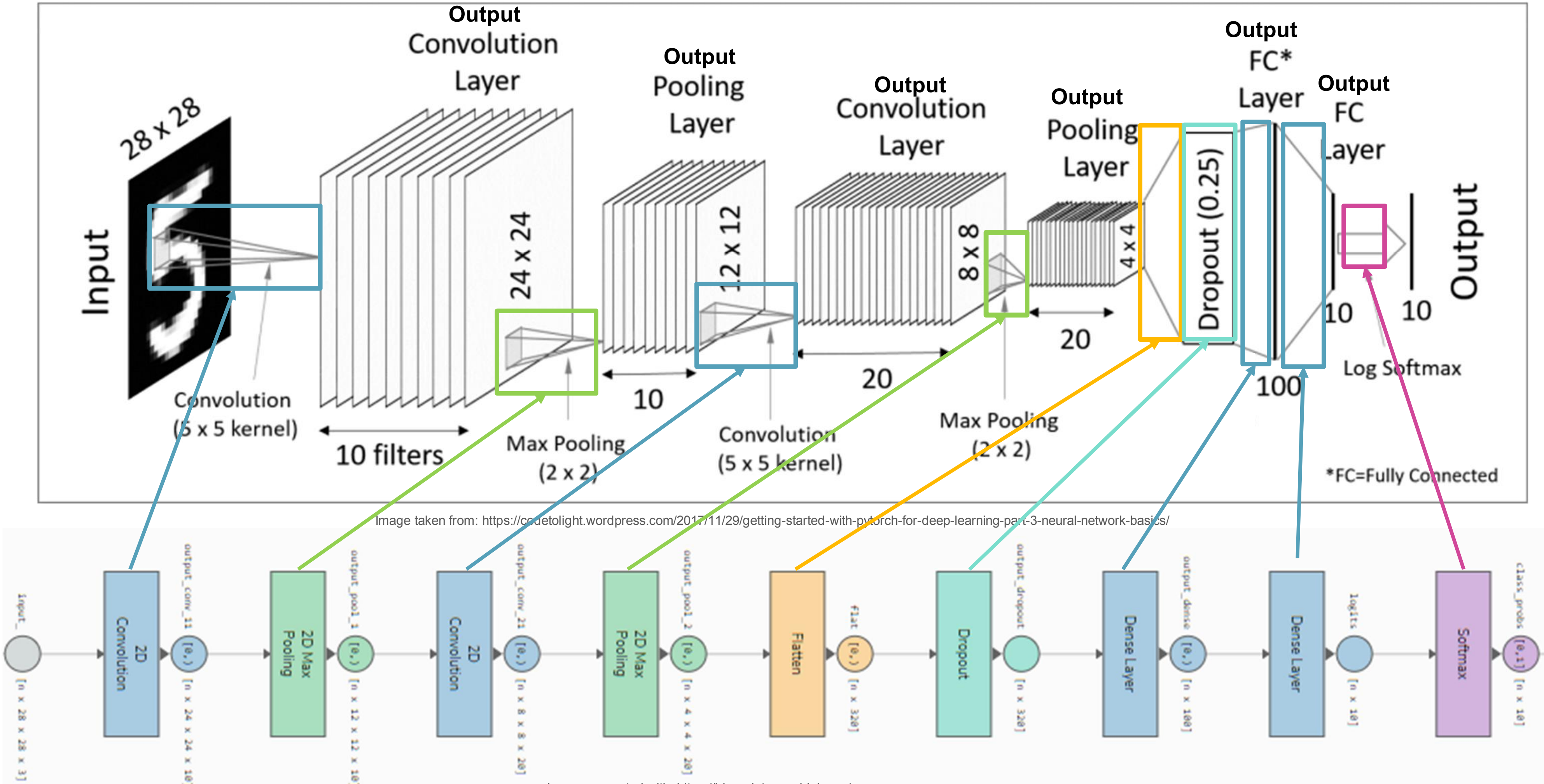


Image taken from: https://codetolight.wordpress.com/2017/11/29/getting-started-with-pytorch-for-deep-learning-part-3-neural-network-basics/

Image generated with: https://blueprints.creaidai.com/

Layout of a Classic Convolutional Neural Network (CNN)

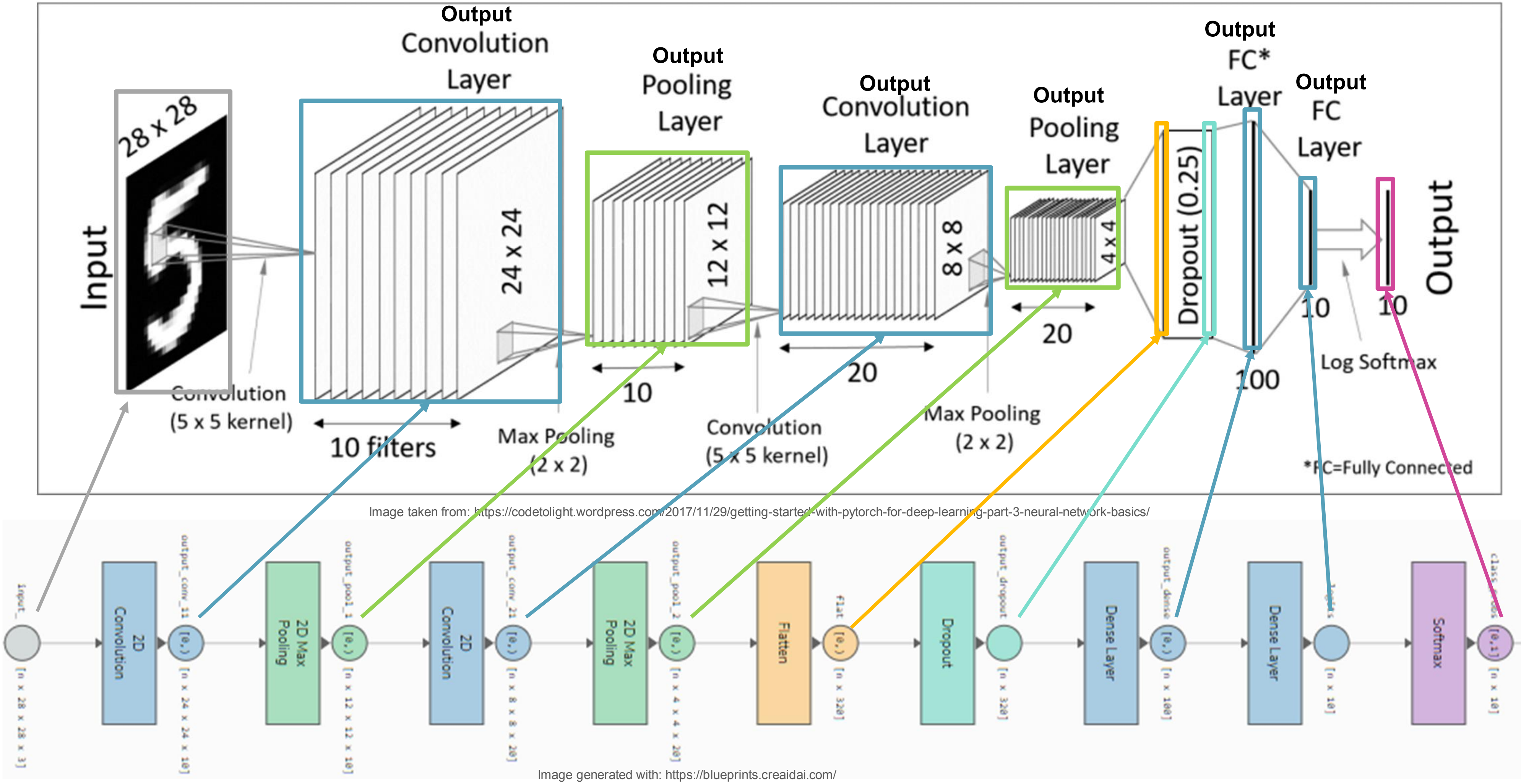


Image taken from: https://codetolight.wordpress.com/2017/11/29/getting-started-with-pytorch-for-deep-learning-part-3-neural-network-basics/

Image generated with: https://blueprints.creaidai.com/

Pooling

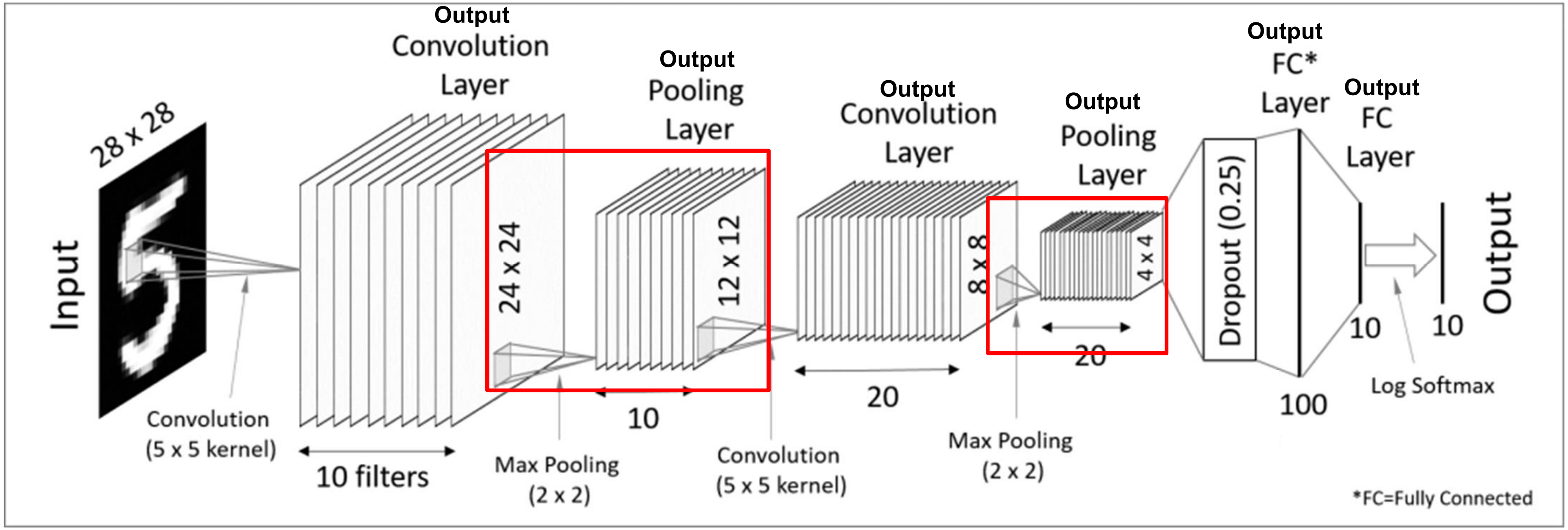


Image taken from: https://codetolight.wordpress.com/2017/11/29/getting-started-with-pytorch-for-deep-learning-part-3-neural-network-basics/

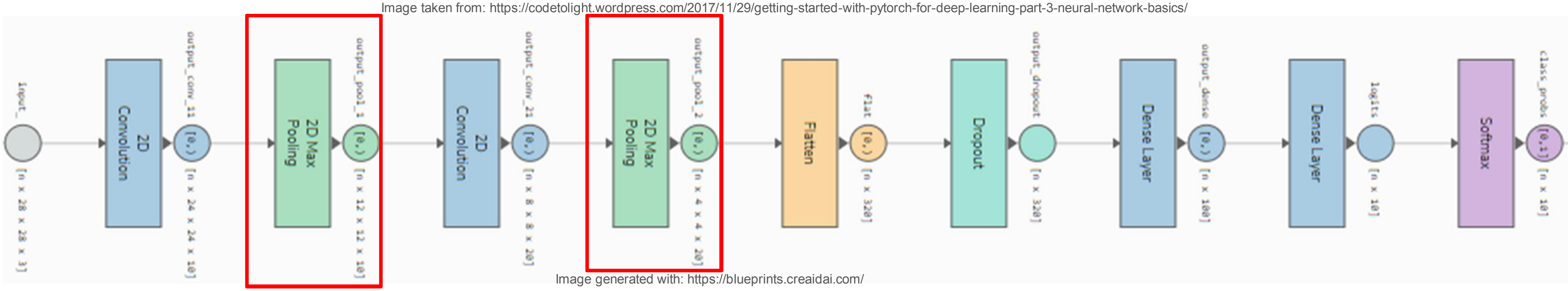
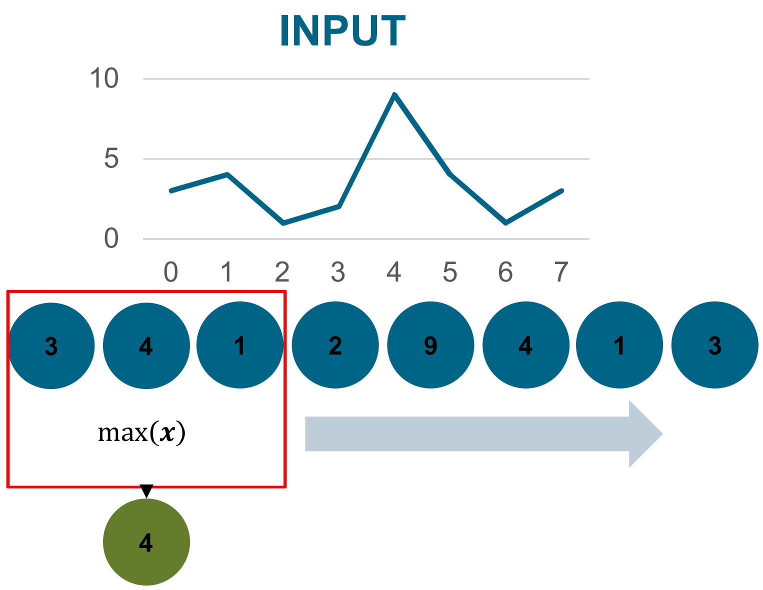
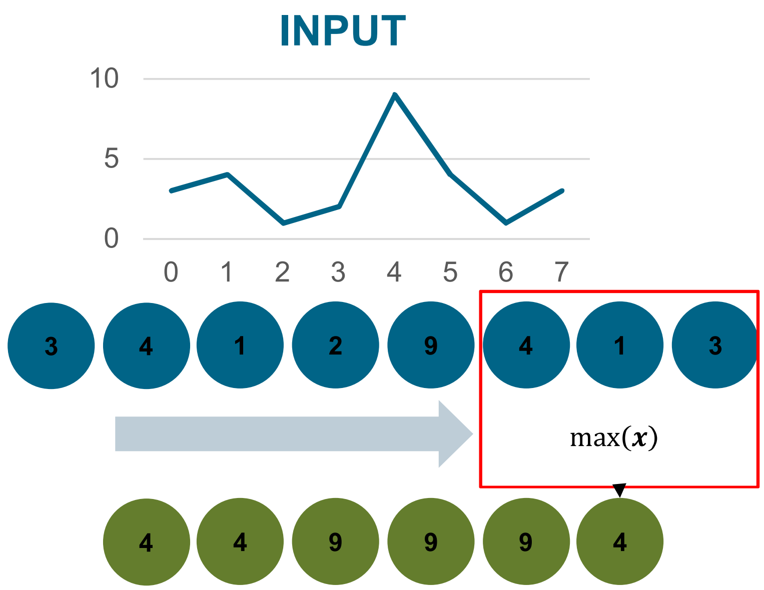


Image generated with: https://blueprints.creaidai.com/

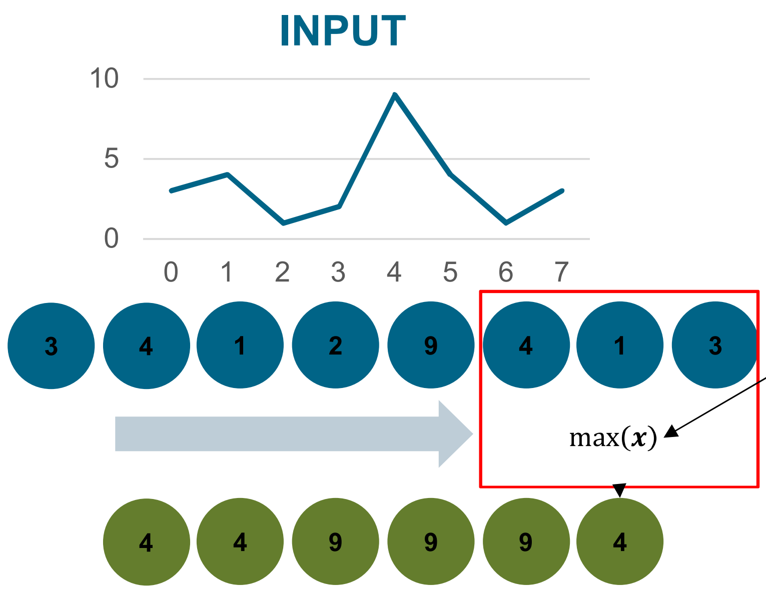
(Max-)Pooling



(Max-)Pooling



(Max-)Pooling



$max(\boldsymbol{x})$ is not the only choice here.

Pooling

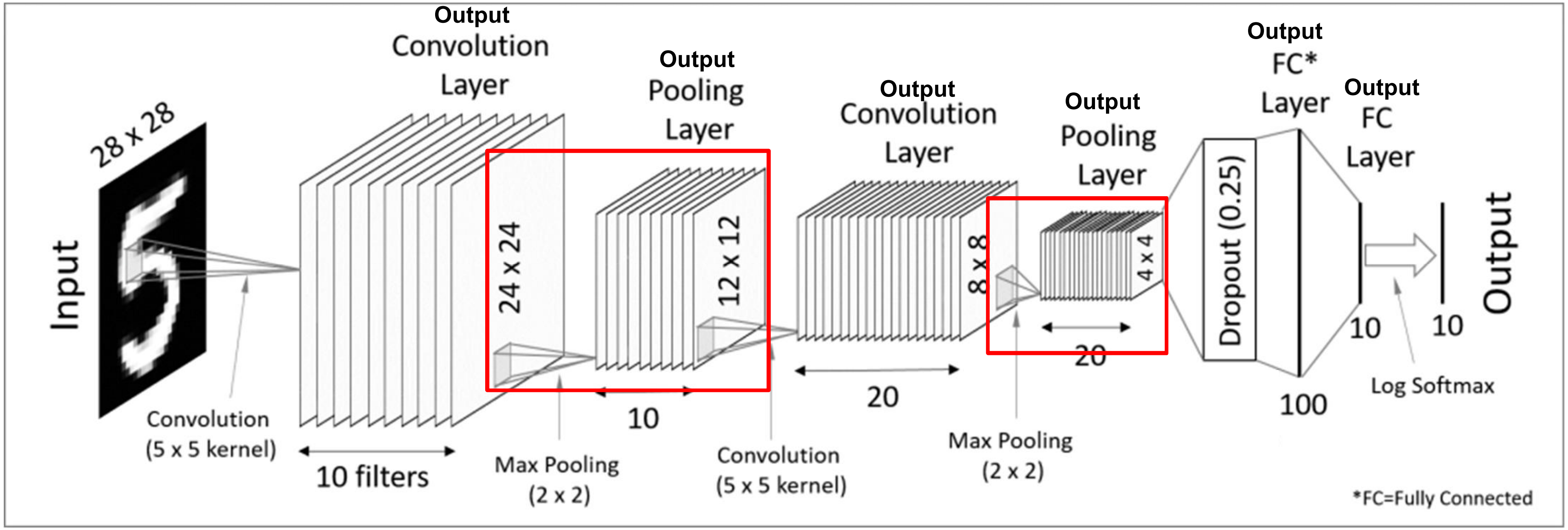


Image taken from: https://codetolight.wordpress.com/2017/11/29/getting-started-with-pytorch-for-deep-learning-part-3-neural-network-basics/

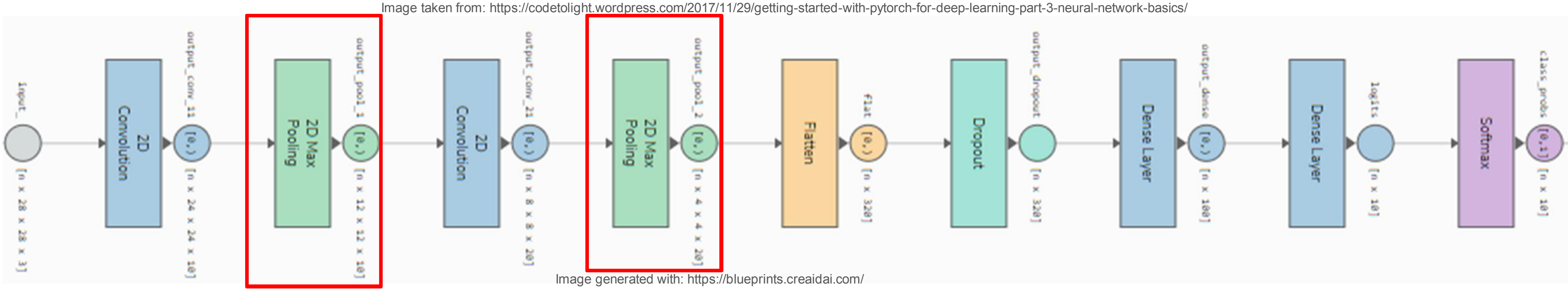


Image generated with: https://blueprints.creaidai.com/

Dropout

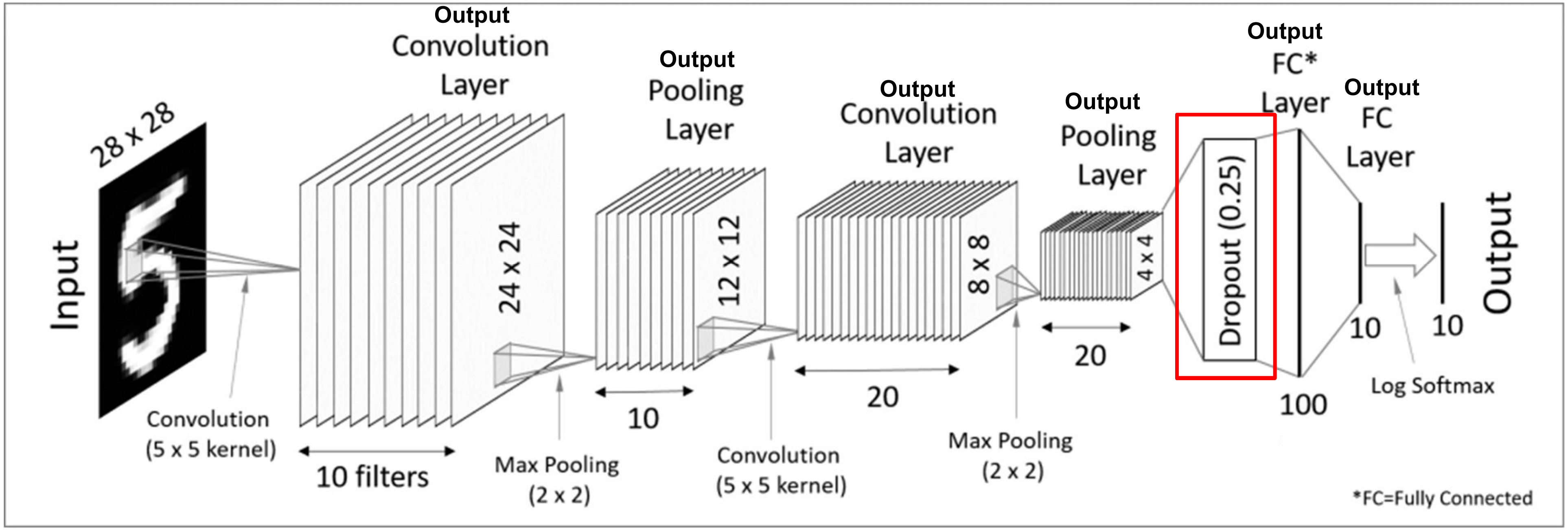


Image taken from: https://codetolight.wordpress.com/2017/11/29/getting-started-with-pytorch-for-deep-learning-part-3-neural-network-basics/

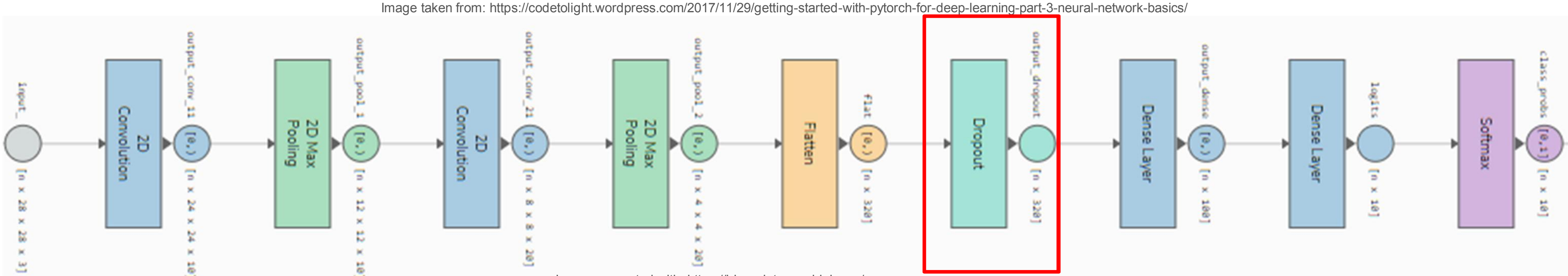


Image generated with: https://blueprints.creaidai.com/

Dropout

Problem

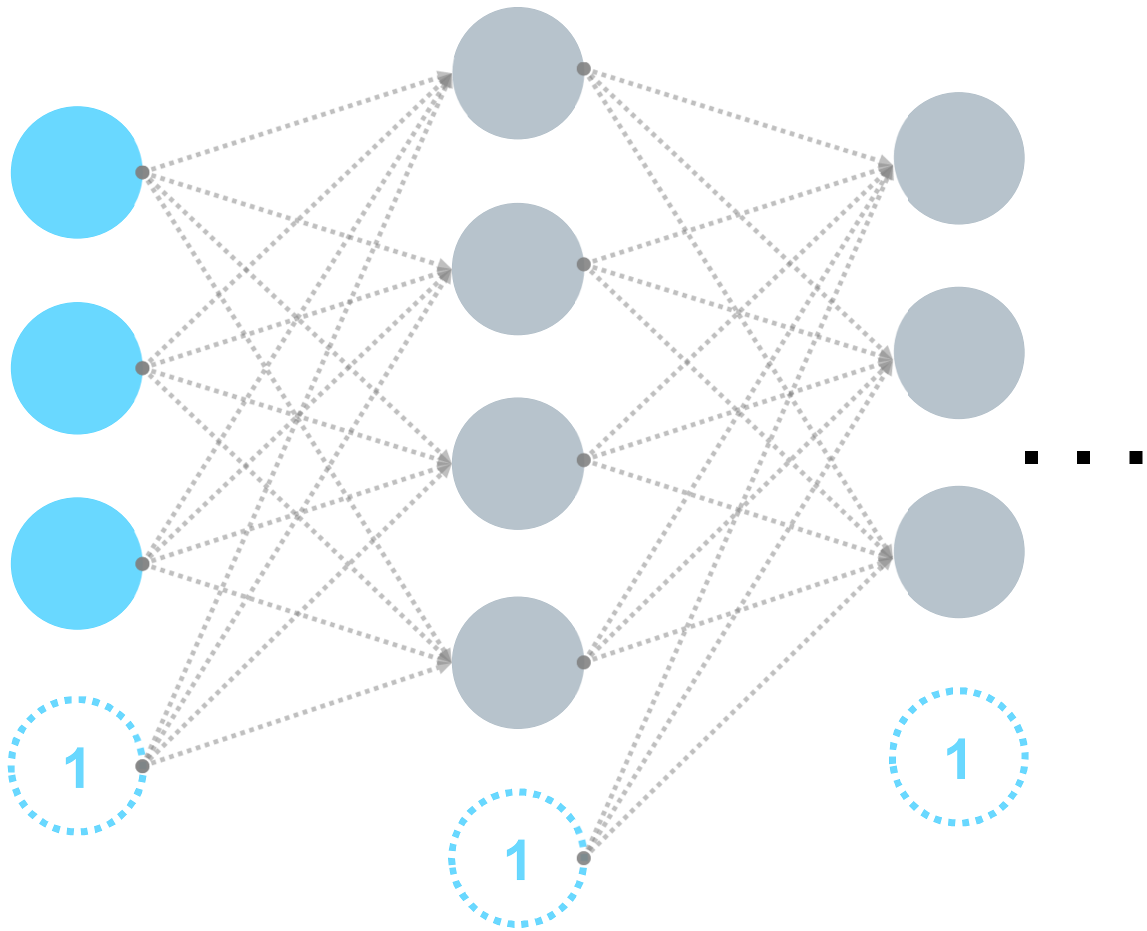
- Deep learning models are often highly over parameterized which allows the model to overfit on or even memorize the training data.

Approach

- Randomly set output neurons to zero
  - ➢Transforms the network into an ensemble with an exponential set of weaker learners whose parameters are shared.

Usage

- Primarily used in dense layers because of the large number of parameters
- Rarely used in convolutional layers
- Rarely used in recurrent neural networks (if at all between the hidden state and output)

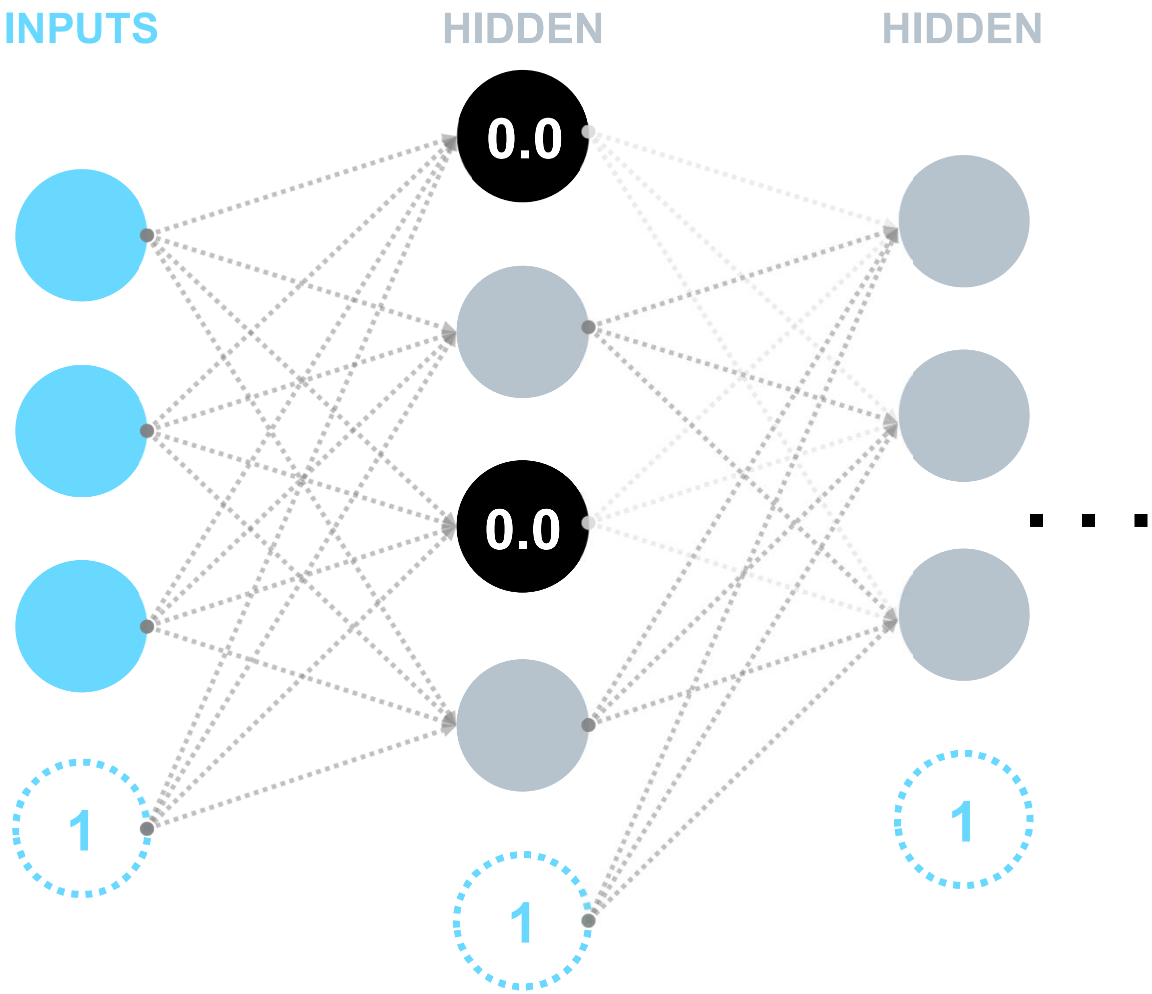


Dropout - Training

Problem
- Deep learning models are often highly over parameterized which allows the model to overfit on or even memorize the training data.

Approach
- Randomly set output neurons to zero
  - ➢Transforms the network into an ensemble with an exponential set of weaker learners whose parameters are shared.

Usage
- Primarily used in dense layers because of the large number of parameters
- Rarely used in convolutional layers
- Rarely used in recurrent neural networks (if at all between the hidden state and output)

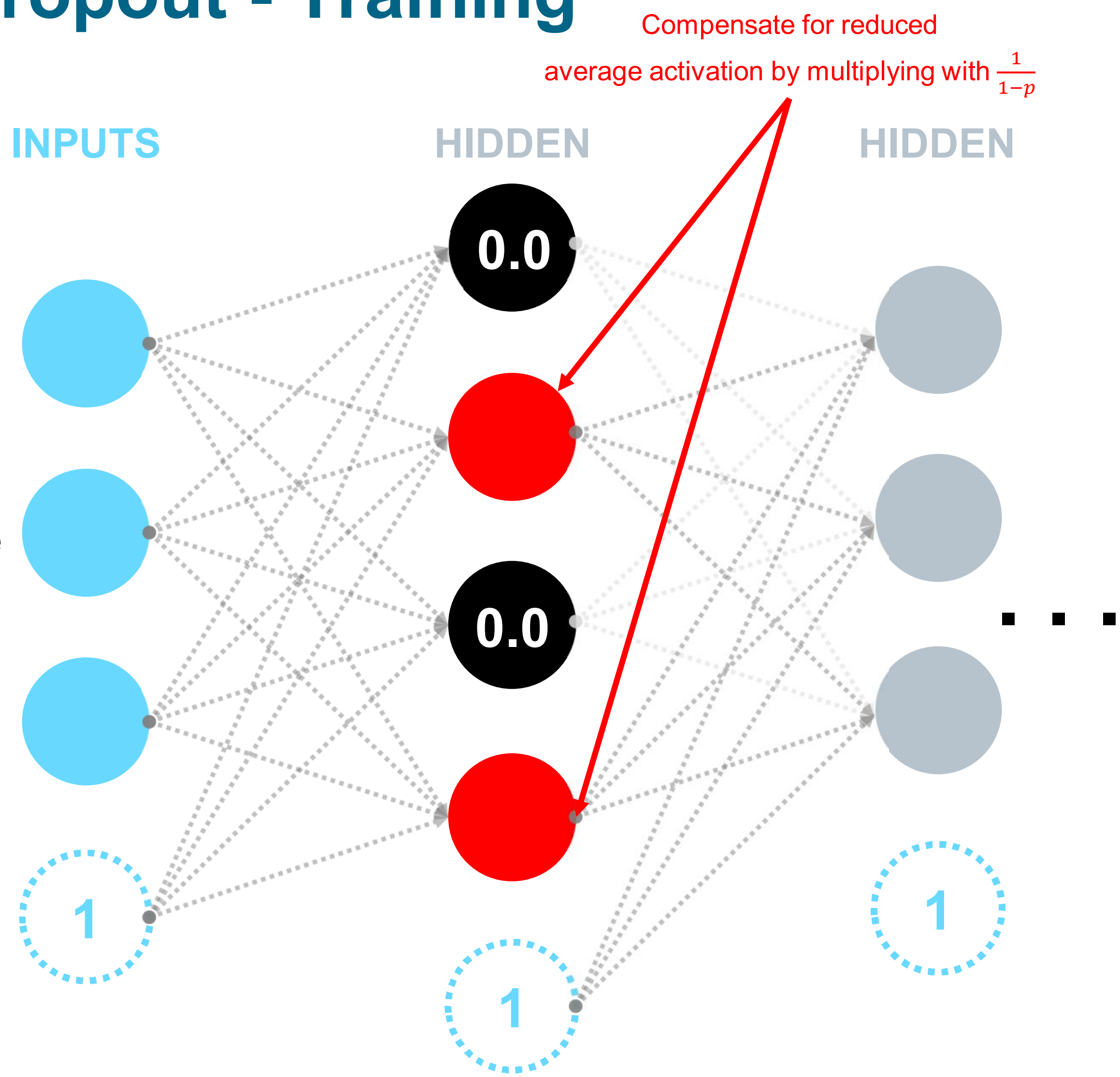


Inverted Dropout - Training

Problem
- Deep learning models are often highly over parameterized which allows the model to overfit on or even memorize the training data.

Approach
- Randomly set output neurons to zero
  - ➢Transforms the network into an ensemble with an exponential set of weaker learners whose parameters are shared.

Usage
- Primarily used in dense layers because of the large number of parameters
- Rarely used in convolutional layers
- Rarely used in recurrent neural networks (if at all between the hidden state and output)



Dropout - Inference

Problem

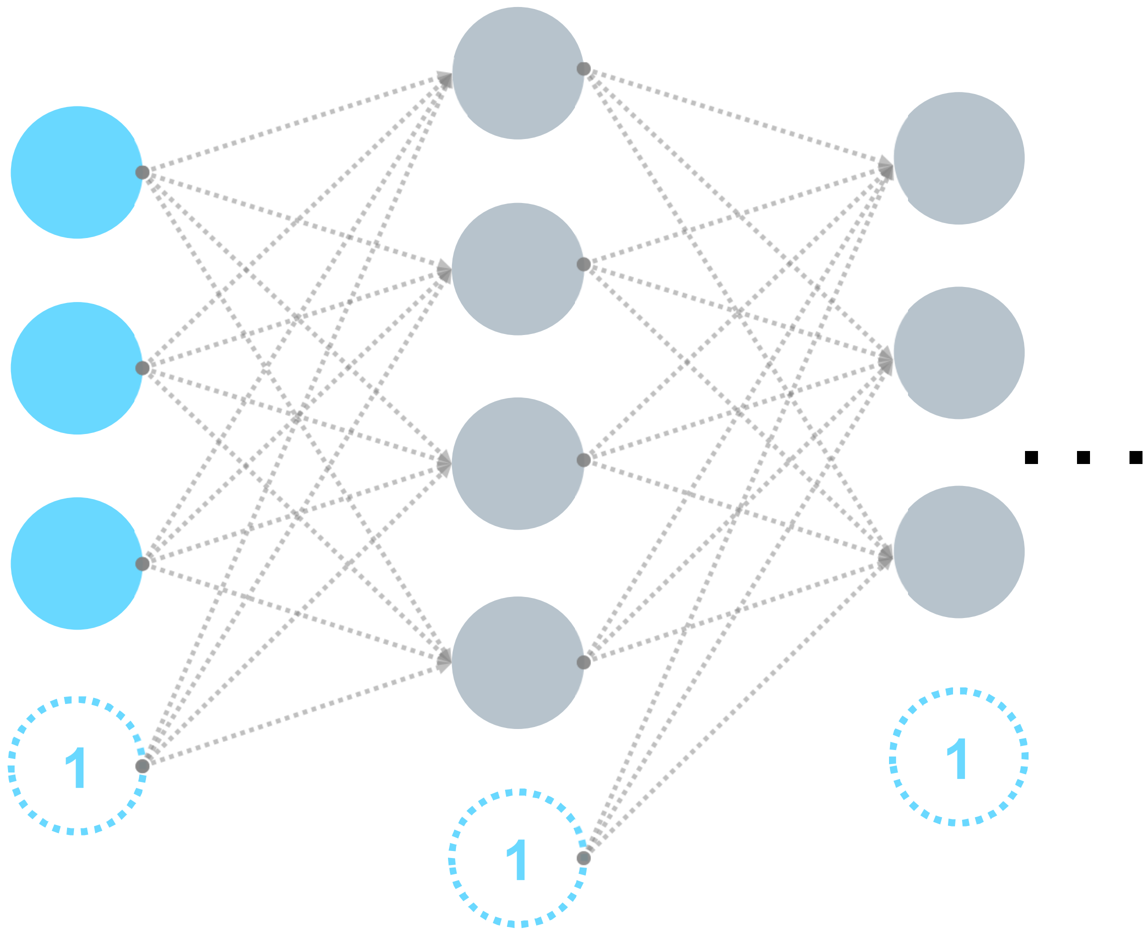
- Deep learning models are often highly over parameterized which allows the model to overfit on or even memorize the training data.

Approach

- Randomly set output neurons to zero
  - ➢ Transforms the network into an ensemble with an exponential set of weaker learners whose parameters are shared.

Usage

- Primarily used in dense layers because of the large number of parameters
- Rarely used in convolutional layers
- Rarely used in recurrent neural networks (if at all between the hidden state and output)



Layout of a Classic Convolutional Neural Network (CNN)

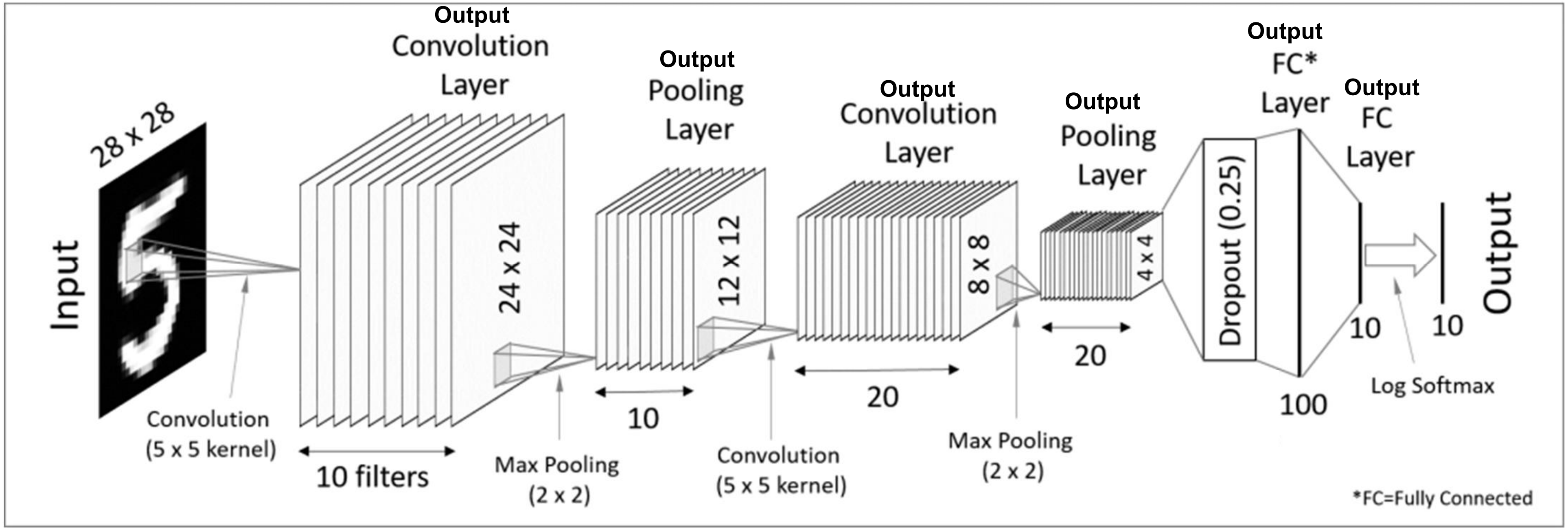


Image taken from: https://codetolight.wordpress.com/2017/11/29/getting-started-with-pytorch-for-deep-learning-part-3-neural-network-basics/

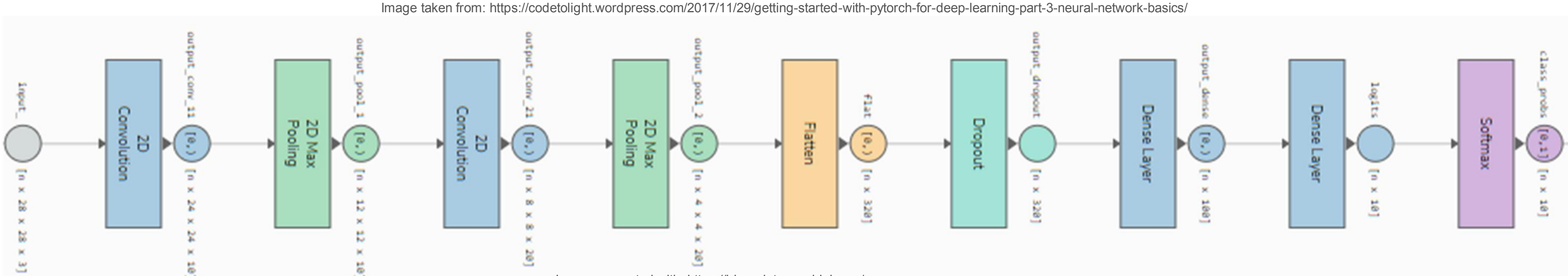


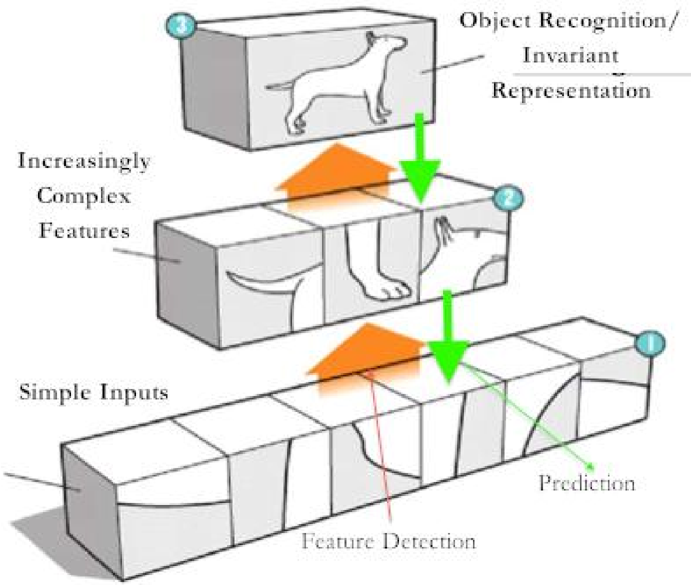
Image generated with: https://blueprints.creaidai.com/

When using this layer as the first layer in a model, provide an `input_shape` argument (tuple of integers or `None`, e.g. `(10, 128)` for sequences of 10 vectors of 128-dimensional vectors, or `(None, 128)` for variable-length sequences of 128-dimensional vectors.

Arguments:

- `filters`: Integer, the dimensionality of the output space (i.e. the number of output filters in the convolution).
- `kernel_size`: An integer or tuple/list of a single integer, specifying the length of the 1D convolution window.
- `strides`: An integer or tuple/list of a single integer, specifying the stride length of the convolution. Specifying any stride value != 1 is incompatible with specifying any `dilation_rate` value != 1.
- `padding`: One of `"valid"`, `"causal"` or `"same"` (case-insensitive). `"causal"` results in causal (dilated) convolutions, e.g. output[t] does not depend on input[t+1:]. Useful when modeling temporal data where the model should not violate the temporal order. See WaveNet: A Generative Model for Raw Audio, section 2.1.
- ~~`data_format`: A string, one of `channels_last` (default) or `channels_first`~~.
- `dilation_rate`: an integer or tuple/list of a single integer, specifying the dilation rate to use for dilated convolution. Currently, specifying any `dilation_rate` value != 1 is incompatible with specifying any `strides` value != 1.
- `activation`: Activation function to use. If you don't specify anything, no activation is applied (ie. "linear" activation: `a(x) = x`).
- `use_bias`: Boolean, whether the layer uses a bias vector.
- `kernel_initializer`: Initializer for the `kernel` weights matrix.
- `bias_initializer`: Initializer for the bias vector.

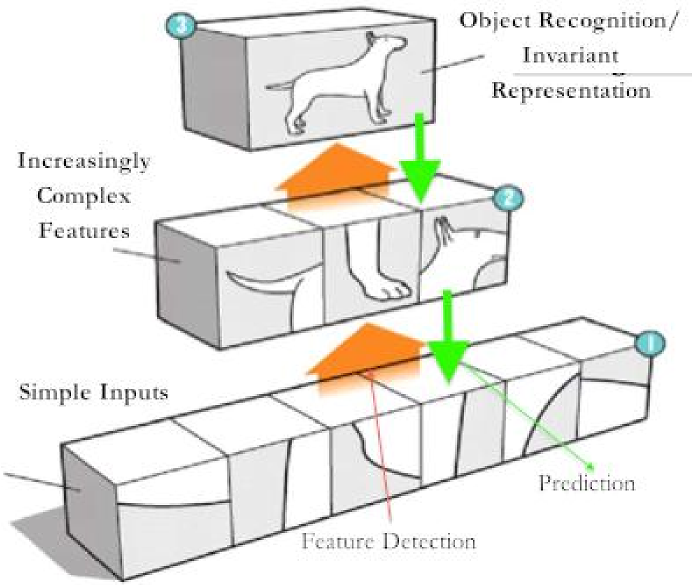
Hierarchical Feature Extraction



SOURCE: http://www.eidolonspeak.com/Artificial_Intelligence/SOA_P3_Fig4.png

This illustration only shows the idea!
In reality the learned features are abstract and hard to interpret most of the time.

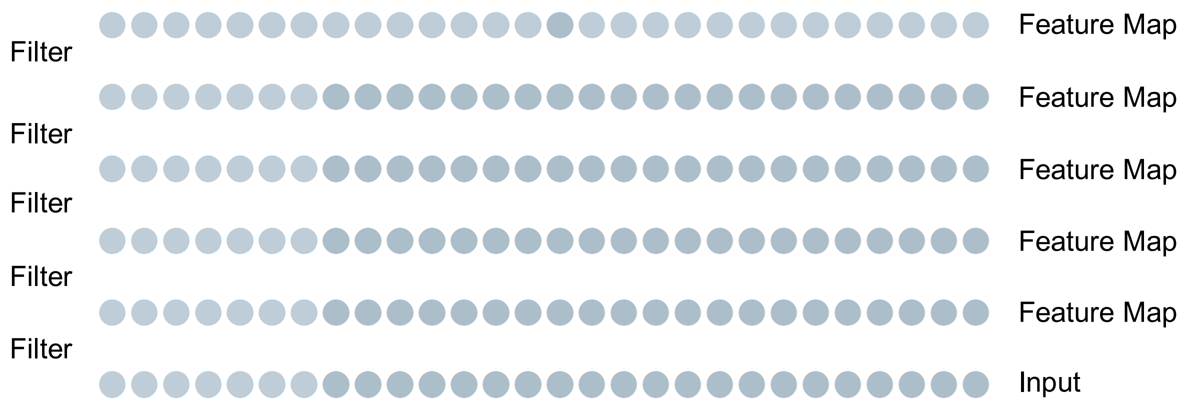
Hierarchical Feature Extraction



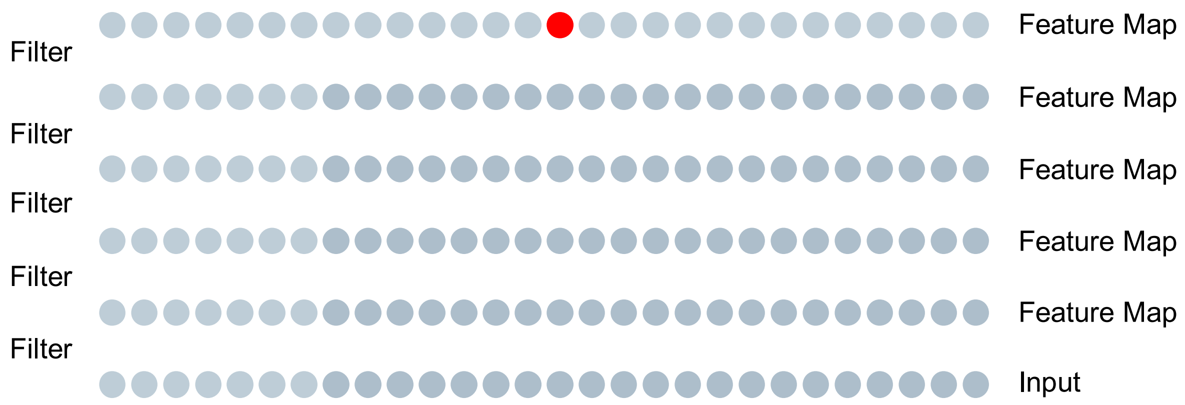
SOURCE: http://www.eidolonspeak.com/Artificial_Intelligence/SOA_P3_Fig4.png

This region is larger than a 3 x 3 or 5 x 5 filter!

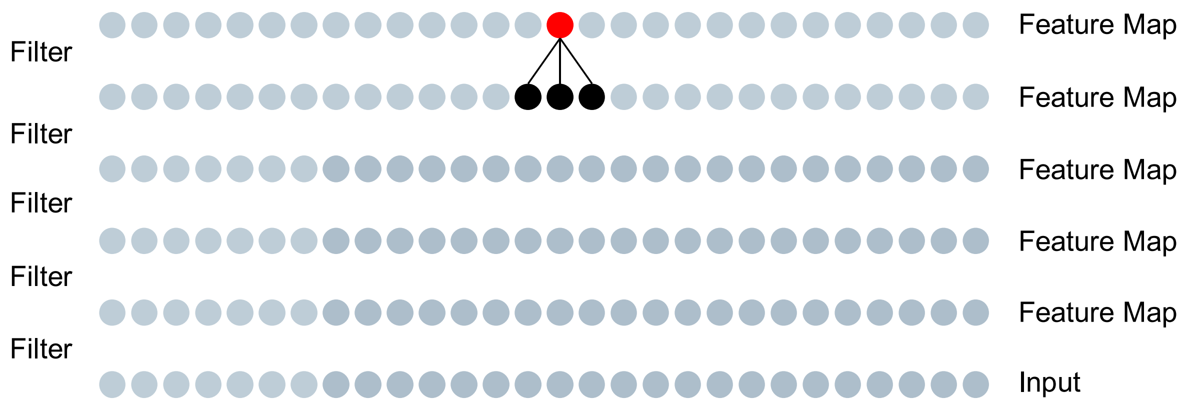
Receptive Field Expansion



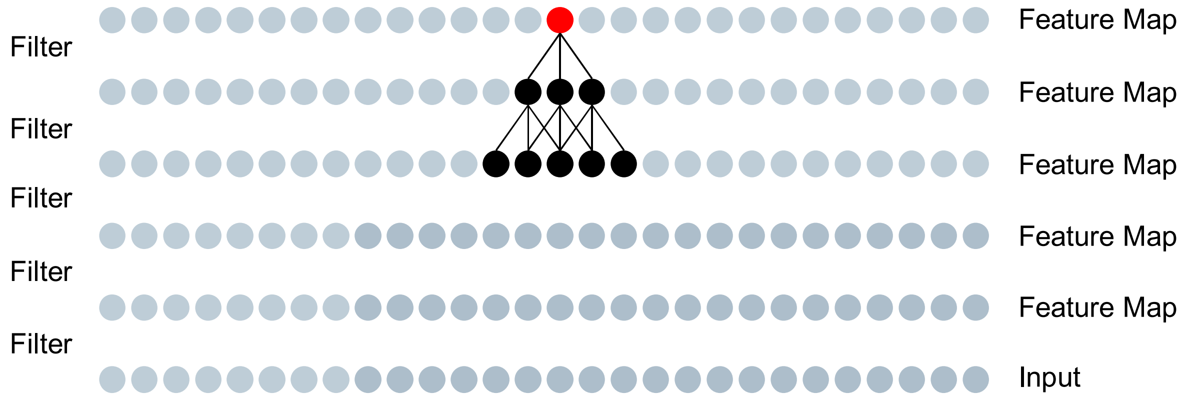
Receptive Field Expansion



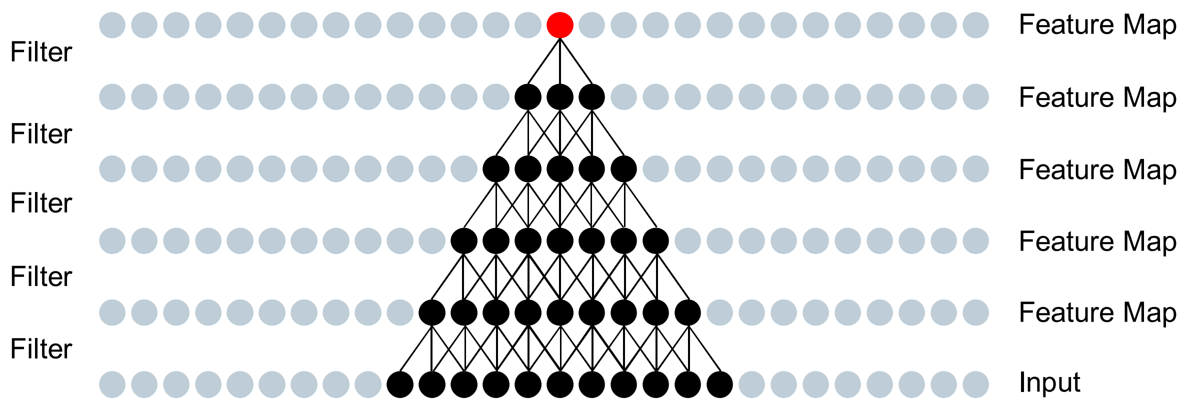
Receptive Field Expansion



Receptive Field Expansion

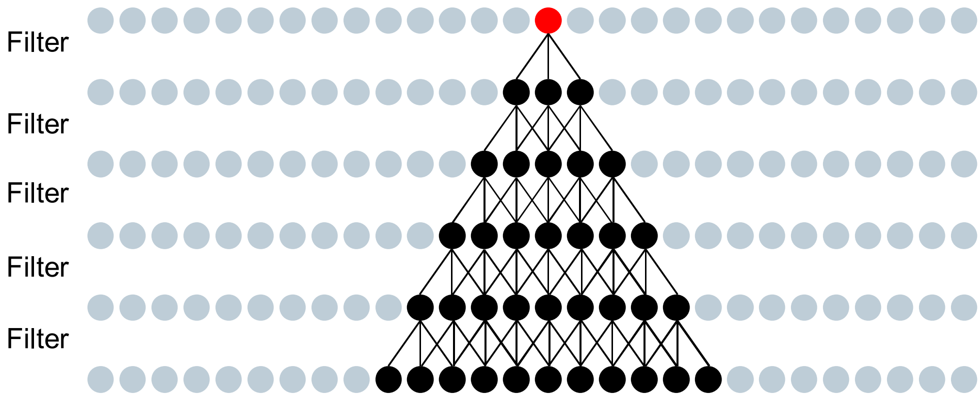


Receptive Field Expansion

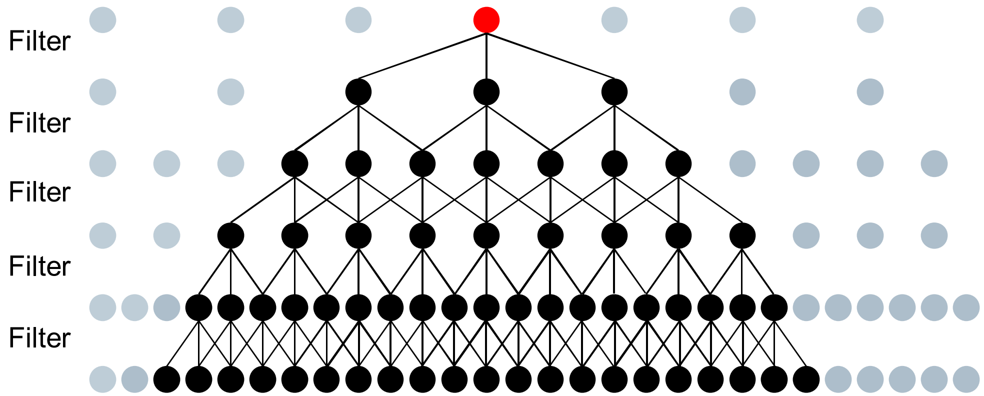


The outputs of the last convolution layer can „see“ information of 11/28 inputs at maximum

Receptive Field Expansion - Strides

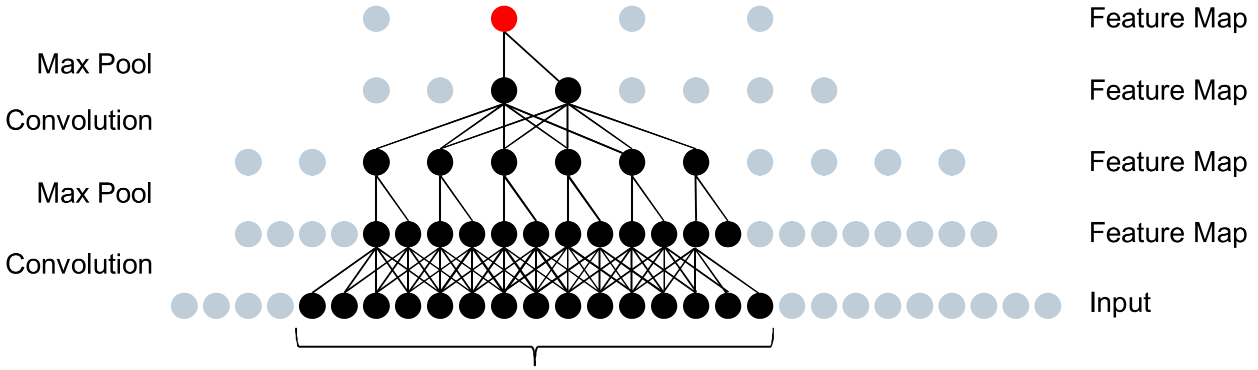


The outputs of the last convolution layer can „see“ information of 11/28 inputs at maximum



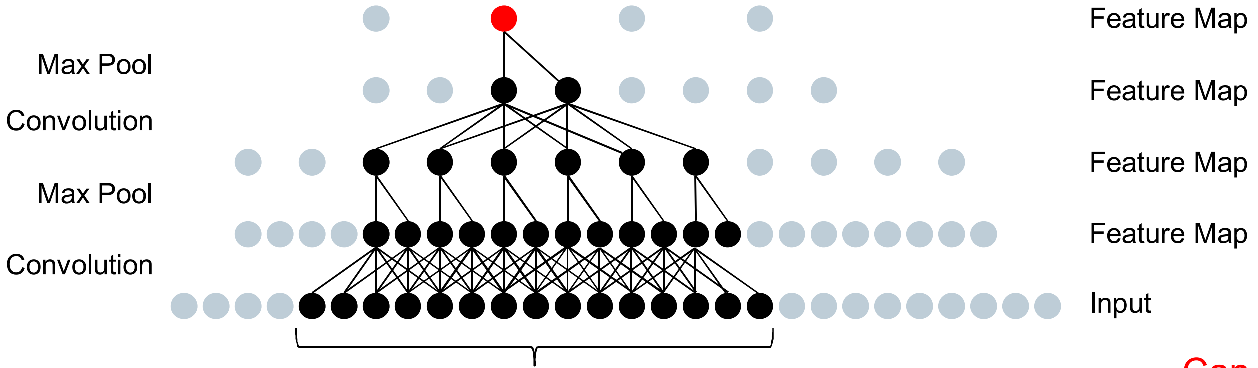
The outputs of the last convolution layer can „see“ information of 21/28 inputs at maximum

Receptive Field Expansion



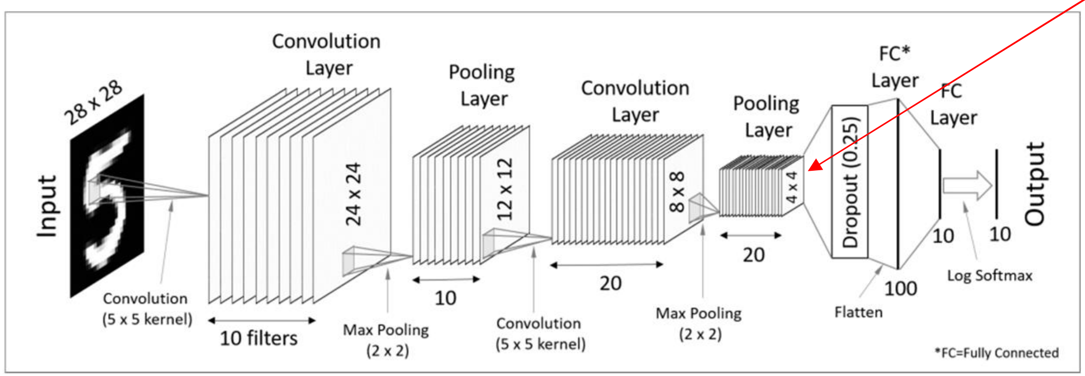
The outputs of the second pooling layer can „see“ information of 15/28 inputs

Receptive Field Expansion

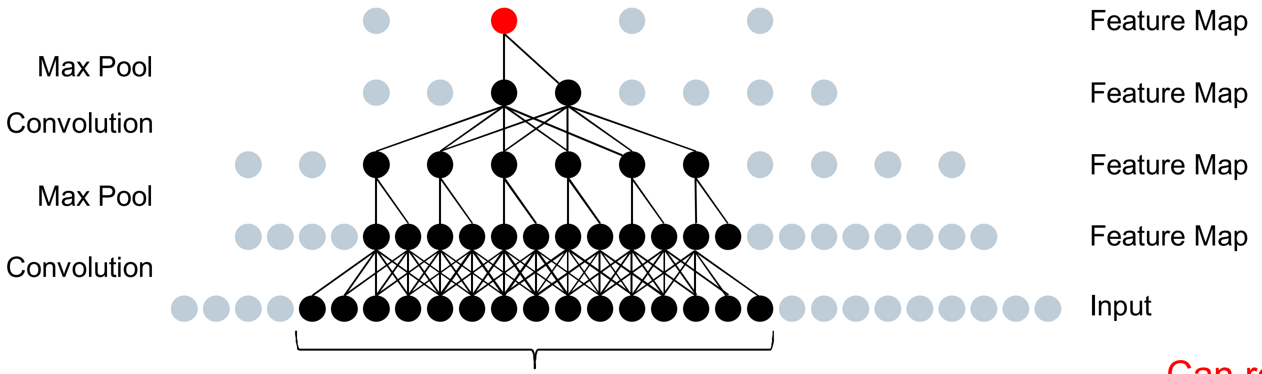


The outputs of the second pooling layer can „see“ information of 15/28 inputs

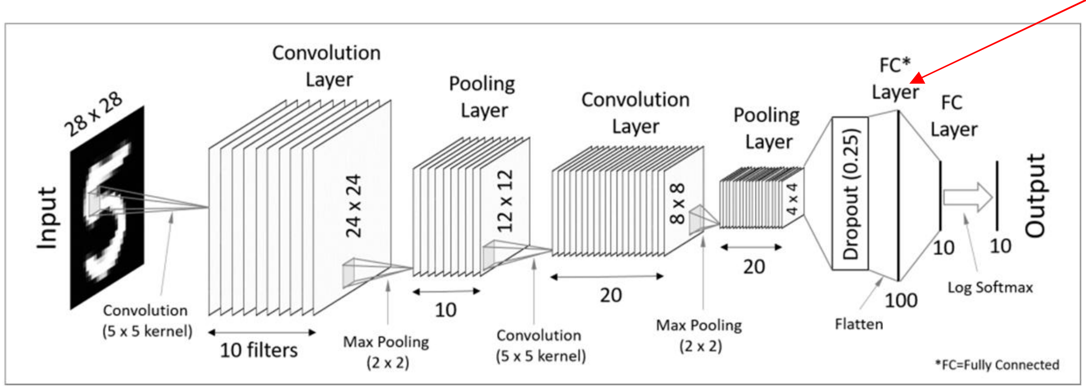
Can extract features that span a 15 x 15 window on the input image.



Receptive Field Expansion



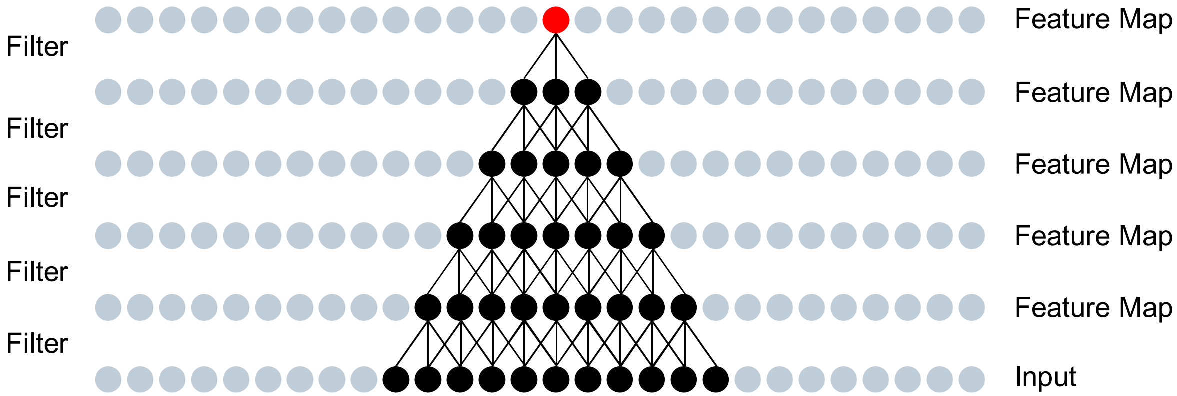
The outputs of the second pooling layer can „see“ information of 15/28 inputs



Can recombine features that span a 15 x 15 window on the input image at maximum.

Receptive Field Expansion

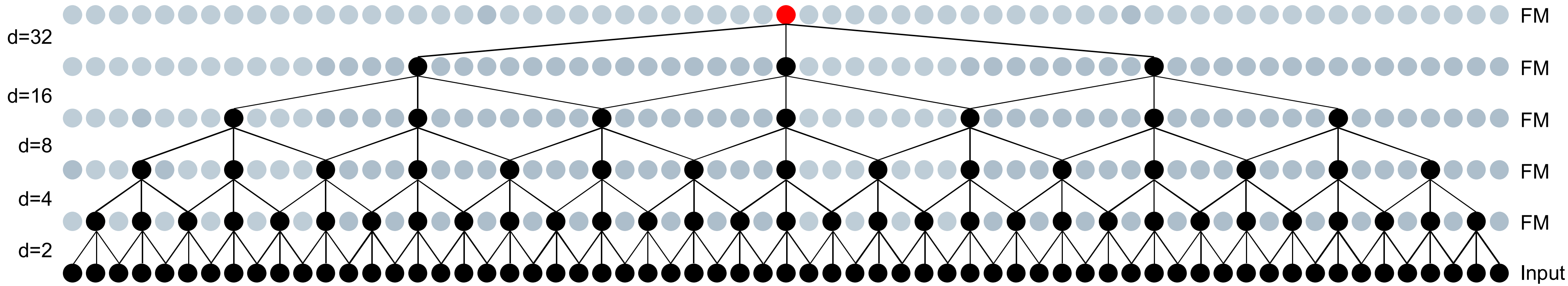
1920 x 1080 x 3



Will need 250 layers to extract features that span a 500 x 500 window if a 3 x 3 filter is used.
Will need 8 layers to extract features that span a 500 x 500 window if a 3x3 filter is used with dilation/or strides of 2.

Receptive Field Expansion

DILATED CONVOLUTION



The outputs of the last convolution layer can „see“
information of 63 inputs at maximum

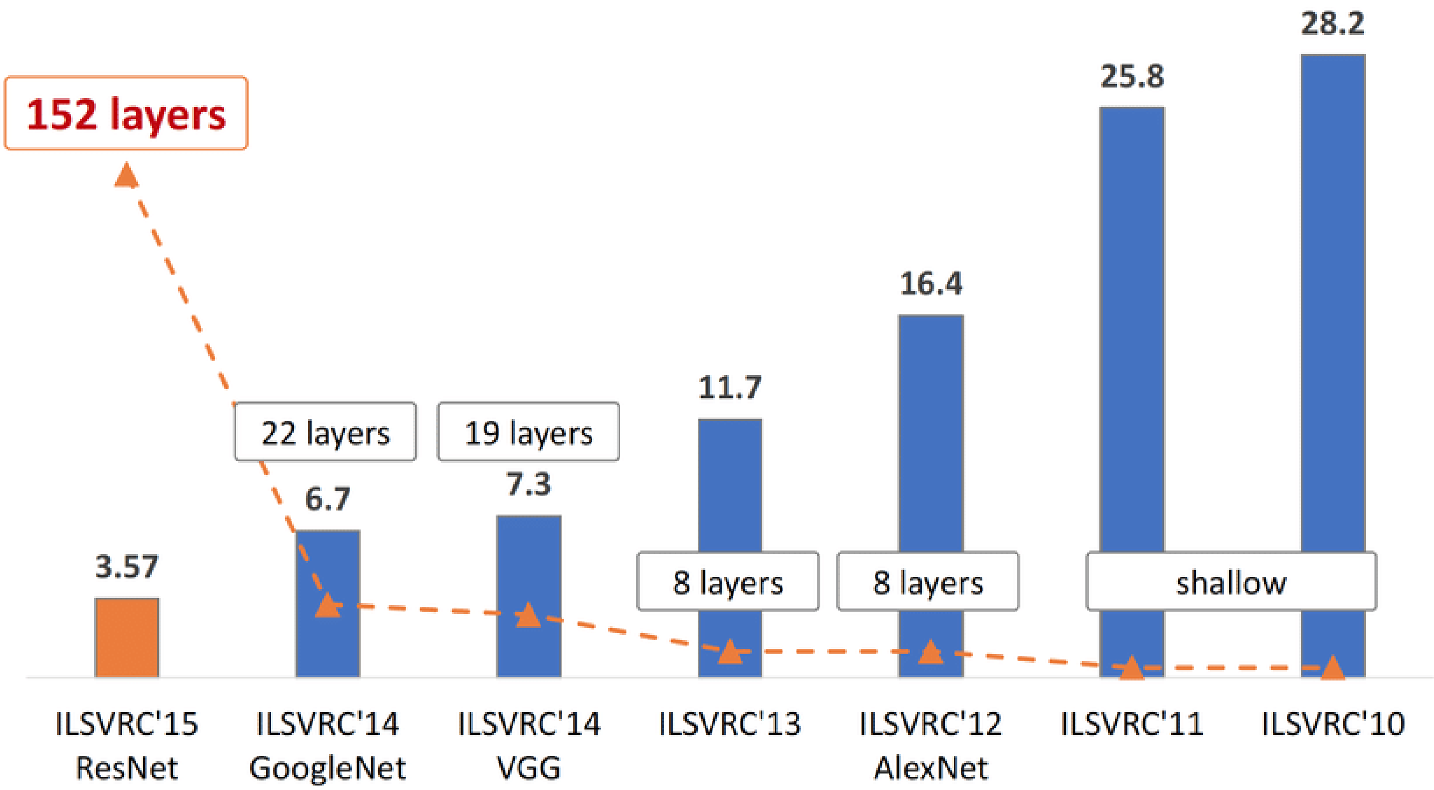
FM = Feature Map

Receptive field expands by $2^{l+1} - 1$

Training Very Deep Convolutional Neural Networks

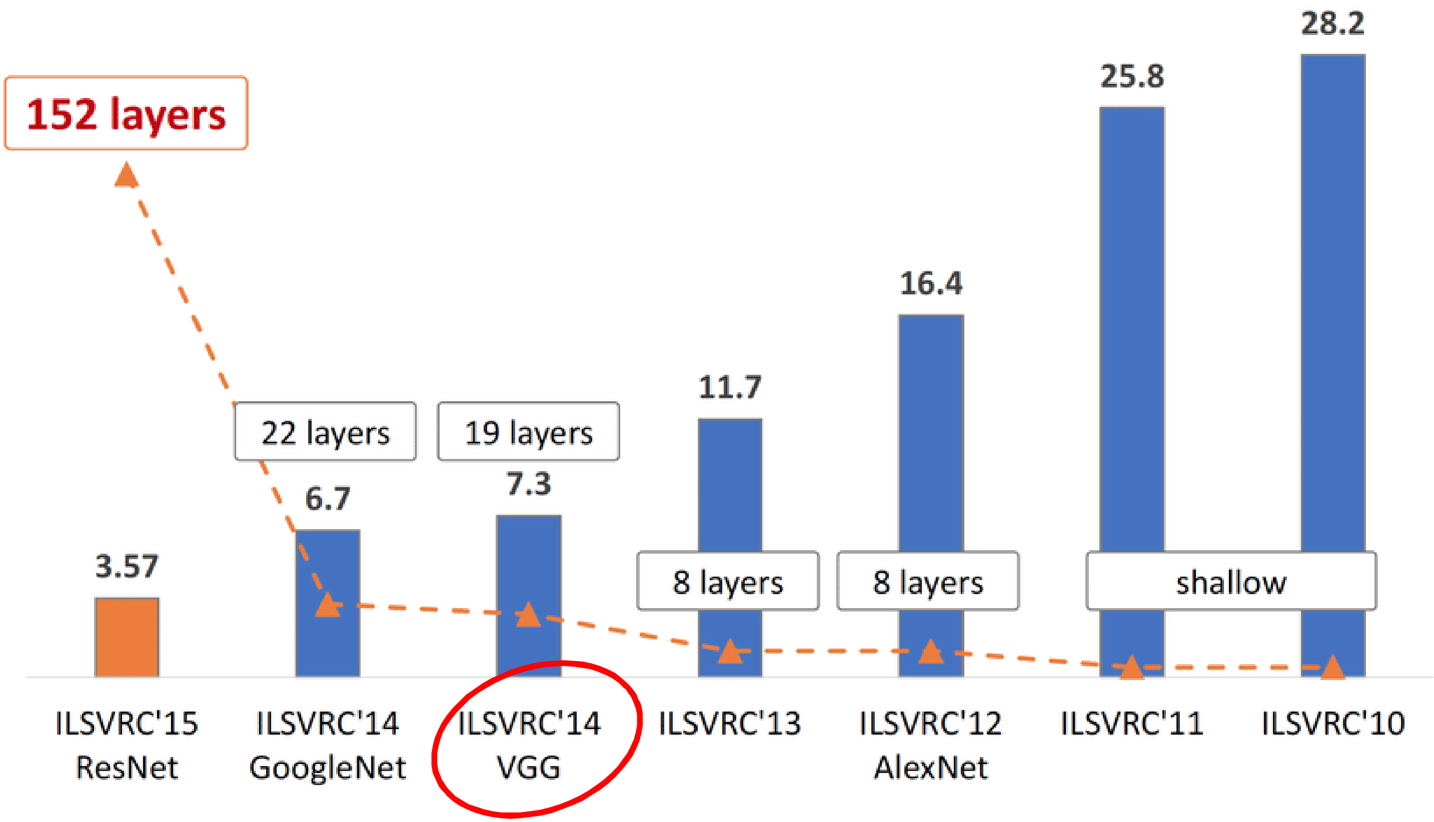
(Not covered in lecture)

Very Deep Convolutional Neural Networks



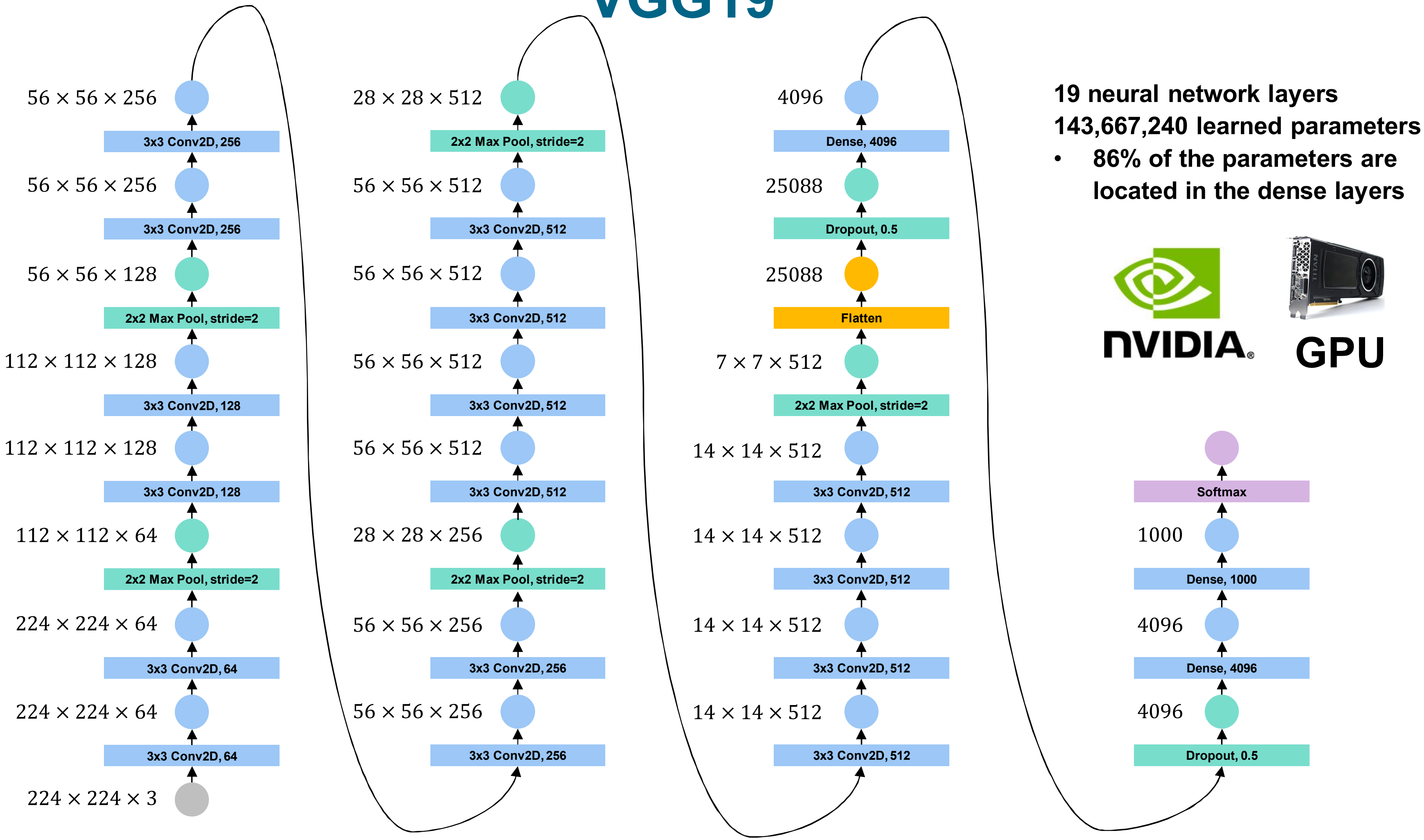
https://www.researchgate.net/figure/The-evolution-of-the-winning-entries-on-the-ImageNet-Large-Scale-Visual-Recognition_fig1_321896881

Very Deep Convolutional Neural Networks



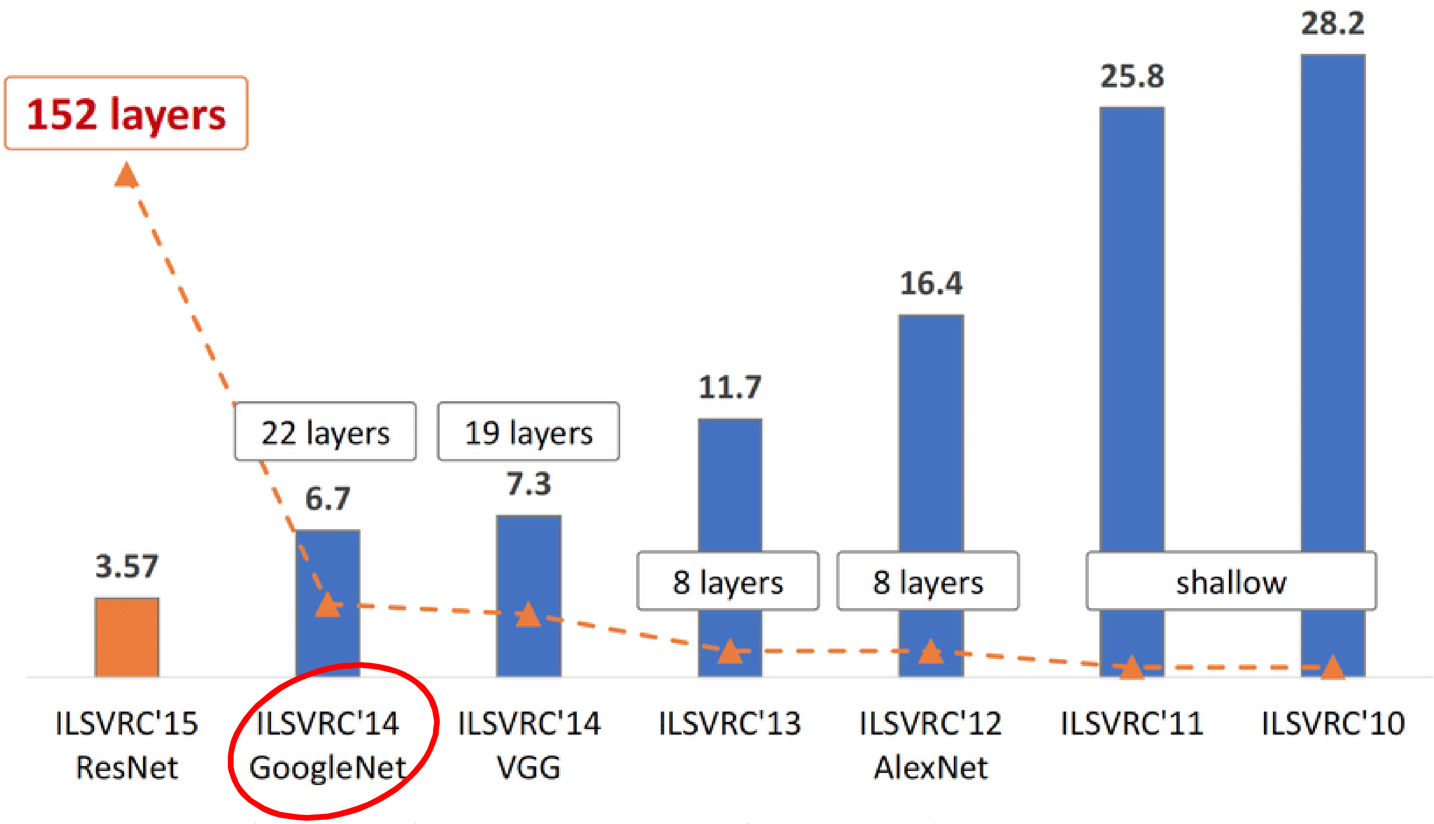
https://www.researchgate.net/figure/The-evolution-of-the-winning-entries-on-the-ImageNet-Large-Scale-Visual-Recognition_fig1_321896881

VGG19



224 × 224 × 3
3x3 Conv2D, 64
224 × 224 × 64
3x3 Conv2D, 64
224 × 224 × 64
2x2 Max Pool, stride=2
112 × 112 × 64
3x3 Conv2D, 128
112 × 112 × 128
3x3 Conv2D, 128
112 × 112 × 128
2x2 Max Pool, stride=2
56 × 56 × 128
3x3 Conv2D, 256
56 × 56 × 256
3x3 Conv2D, 256
56 × 56 × 256
3x3 Conv2D, 256
56 × 56 × 256
3x3 Conv2D, 256
56 × 56 × 256
2x2 Max Pool, stride=2
28 × 28 × 256
3x3 Conv2D, 512
56 × 56 × 512
3x3 Conv2D, 512
56 × 56 × 512
3x3 Conv2D, 512
56 × 56 × 512
3x3 Conv2D, 512
56 × 56 × 512
2x2 Max Pool, stride=2
28 × 28 × 512
3x3 Conv2D, 512
14 × 14 × 512
3x3 Conv2D, 512
14 × 14 × 512
3x3 Conv2D, 512
14 × 14 × 512
3x3 Conv2D, 512
14 × 14 × 512
2x2 Max Pool, stride=2
7 × 7 × 512
Flatten
25088
Dropout, 0.5
25088
Dense, 4096
4096
Dropout, 0.5
4096
Dense, 4096
4096
Dense, 1000
1000
Softmax
19 neural network layers
143,667,240 learned parameters
• 86% of the parameters are located in the dense layers
NVIDIA
GPU

Very Deep Convolutional Neural Networks



https://www.researchgate.net/figure/The-evolution-of-the-winning-entries-on-the-ImageNet-Large-Scale-Visual-Recognition_fig1_321896881

GoogleNet (Inception)

64 neural network layers (22 layer deep)
16,063,912 learned parameters

- **45% of the parameters are located in the dense layers**

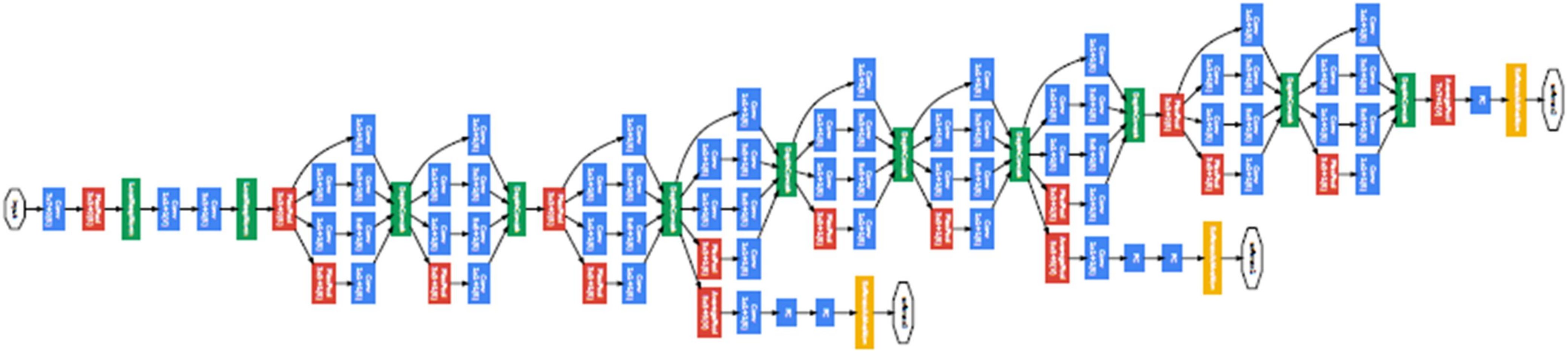
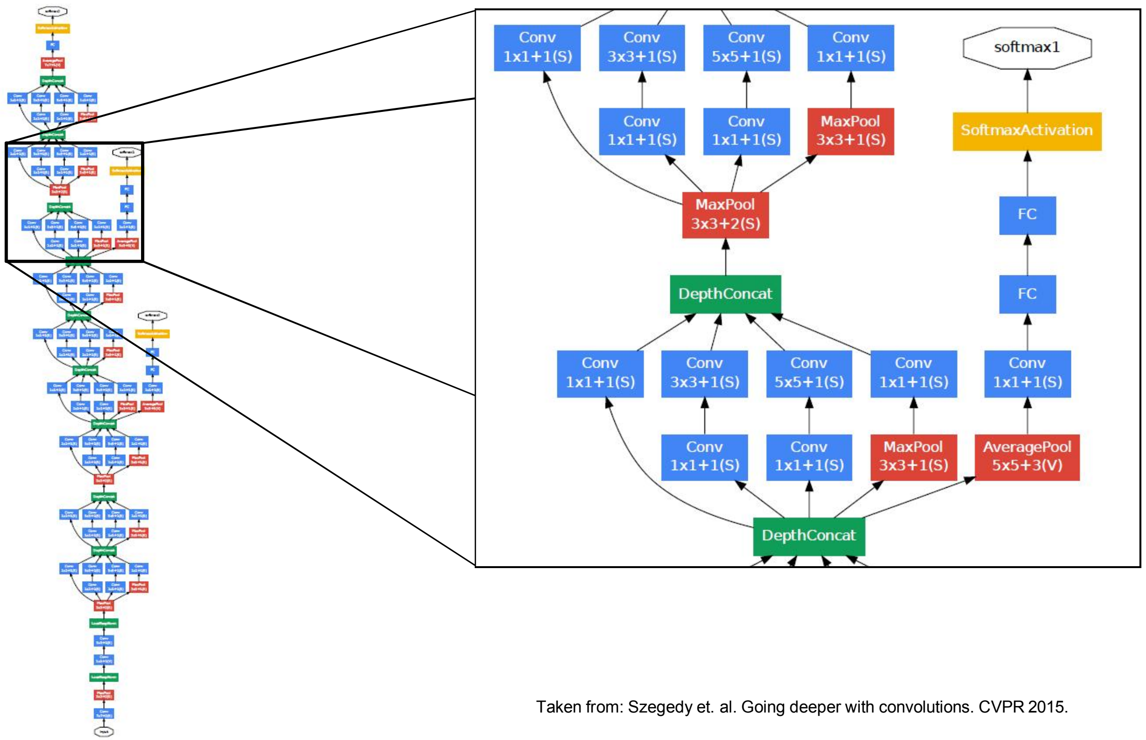


Figure 3: GoogLeNet network with all the bells and whistles

Taken from: Szegedy et. al. Going deeper with convolutions. CVPR 2015.

GoogleNet (Inception)

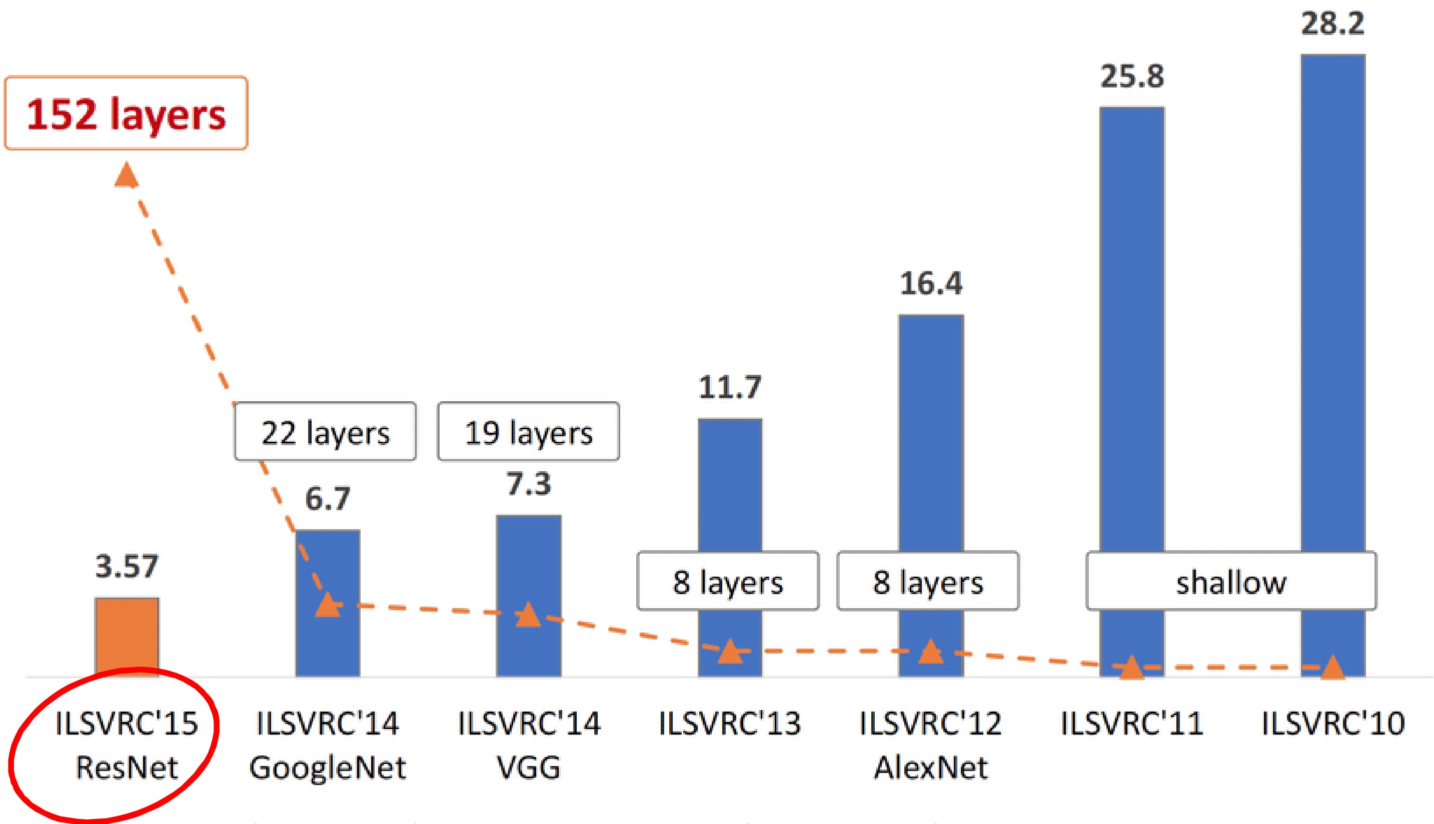
Auxiliary Heads



Taken from: Szegedy et. al. Going deeper with convolutions. CVPR 2015.

Figure 3: GoogLeNet network with all the bells and whistles

Very Deep Convolutional Neural Networks

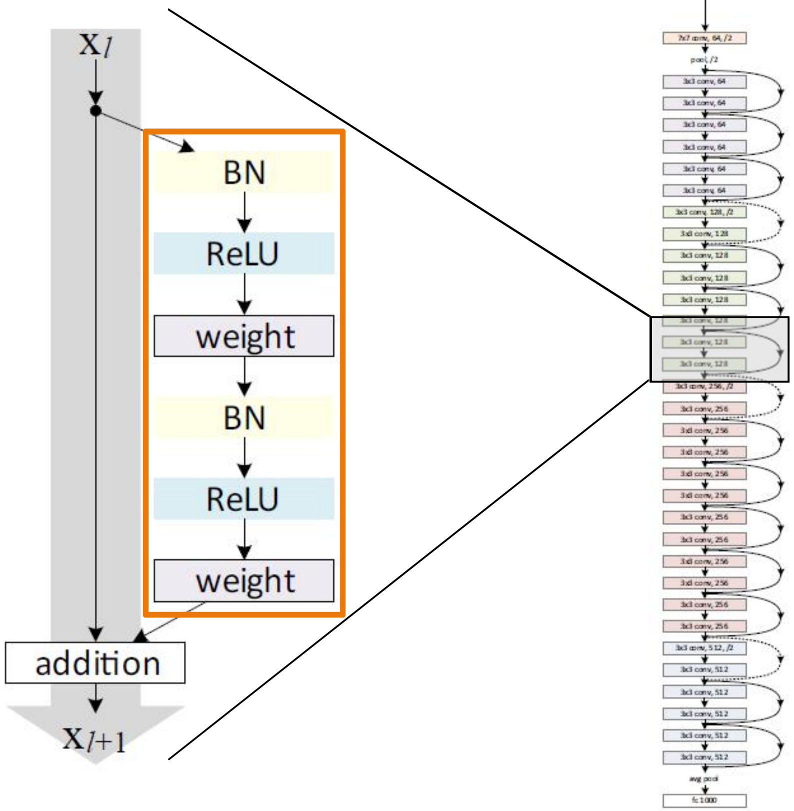


https://www.researchgate.net/figure/The-evolution-of-the-winning-entries-on-the-ImageNet-Large-Scale-Visual-Recognition_fig1_321896881

ResNet

Residual Unit Structure

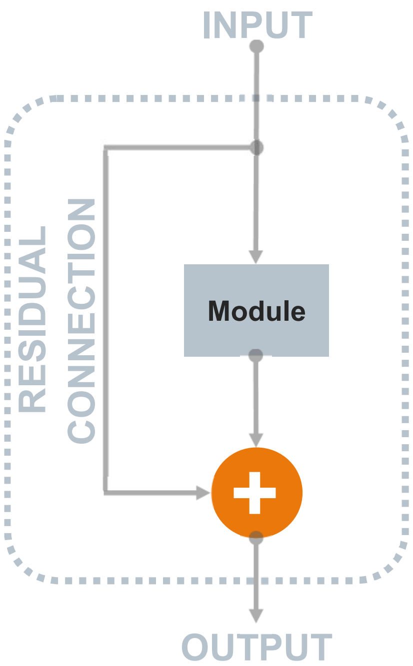
$$\mathbf{x}_{l+1} = \mathbf{x}_l + \mathcal{F}(\mathbf{x}_l, \mathcal{W}_l)$$



32 to up to 1000 neural network layers

Taken from: He et. al. Deep Residual Learning for Image Recognition. CVPR 2016.

Residual Units



$$\mathbf{x}_{l+1} = \mathbf{x}_l + \mathcal{F}(\mathbf{x}_l, \mathcal{W}_l)$$

Module = any differentiable function (e.g. neural network layers) that maps the inputs to some outputs. If the outputs do not have the same shape as the inputs some additional adjustments (e.g. padding) are required.

Reason why deep residual learning works:

Recursive formulation of ResNet:

$$\mathbf{x}_L = \mathbf{x}_l + \sum_{i=l}^{L-1} \mathcal{F}(\mathbf{x}_i, \mathcal{W}_i)$$

Leads to very nice back propagation/gradient properties:

$$\frac{\partial \mathcal{E}}{\partial \mathbf{x}_l} = \frac{\partial \mathcal{E}}{\partial \mathbf{x}_L} \frac{\partial \mathbf{x}_L}{\partial \mathbf{x}_l} = \boxed{\frac{\partial \mathcal{E}}{\partial \mathbf{x}_L}} \left(1 + \frac{\partial}{\partial \mathbf{x}_l} \sum_{i=l}^{L-1} \mathcal{F}(\mathbf{x}_i, \mathcal{W}_i)\right)$$

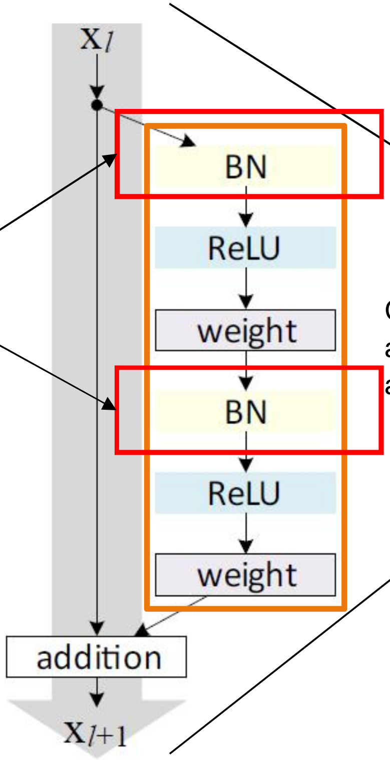
Propagates information directly without concerning any weight layers! (x_l is any shallower layer in the net and x_L is the output any deeper layer L in the net). This becomes clearer if you set l = 0 and L to be the last layer.

ResNet

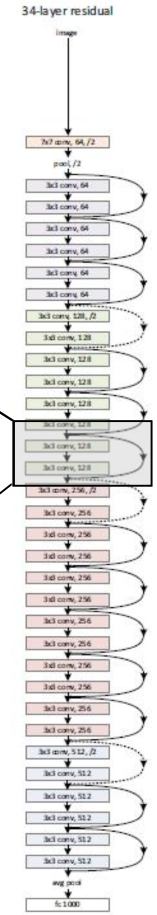
Residual Unit Structure

$$\mathbf{x}_{l+1} = \mathbf{x}_l + \mathcal{F}(\mathbf{x}_l, \mathcal{W}_l)$$

BATCH NORMALIZATION



Can be replaced by any network architecture



32 to up to 1000 neural network layers

Taken from: He et. al. Deep Residual Learning for Image Recognition. CVPR 2016.

Batch Normalization

Problem

- Deep neural networks suffer from internal covariate shift which makes training harder.

Approach

- Normalize the inputs of each layer (zero mean, unit variance)
  - ➢Regularizes because the training network is no longer producing deterministic values in each layer for a given training example

Usage

- Can be used with all layers (FC, RNN, Conv)
- With Convolutional layers, the mini-batch statistics are computed from all patches in the mini-batch.

Normalize the input X of layer k by the mini-batch moments:

$$\hat{X}^{(k)} = \frac{X^{(k)} - \mu_X^{(k)}}{\sigma_X^{(k)}}$$

The next step gives the model the freedom of learning to undo the normalization if needed:

$$\tilde{X}^{(k)} = \gamma^{(k)} \hat{X}^{(k)} + \beta^{(k)}$$

The above two steps in one formula.

$$\tilde{X}^{(k)} = \gamma^{(k)} \cdot \frac{X^{(k)}}{\sigma_X^{(k)}} + \beta^{(k)} - \gamma^{(k)} \cdot \frac{\mu_X^{(k)}}{\sigma_X^{(k)}}$$

Note: At inference time, an unbiased estimate of the mean and standard deviation computed from all seen mini-batches during training is used.

It‘s Not Just Gradient Flow Problems!

Training very deep (Convolutional) neural networks can also lead to the following issues:

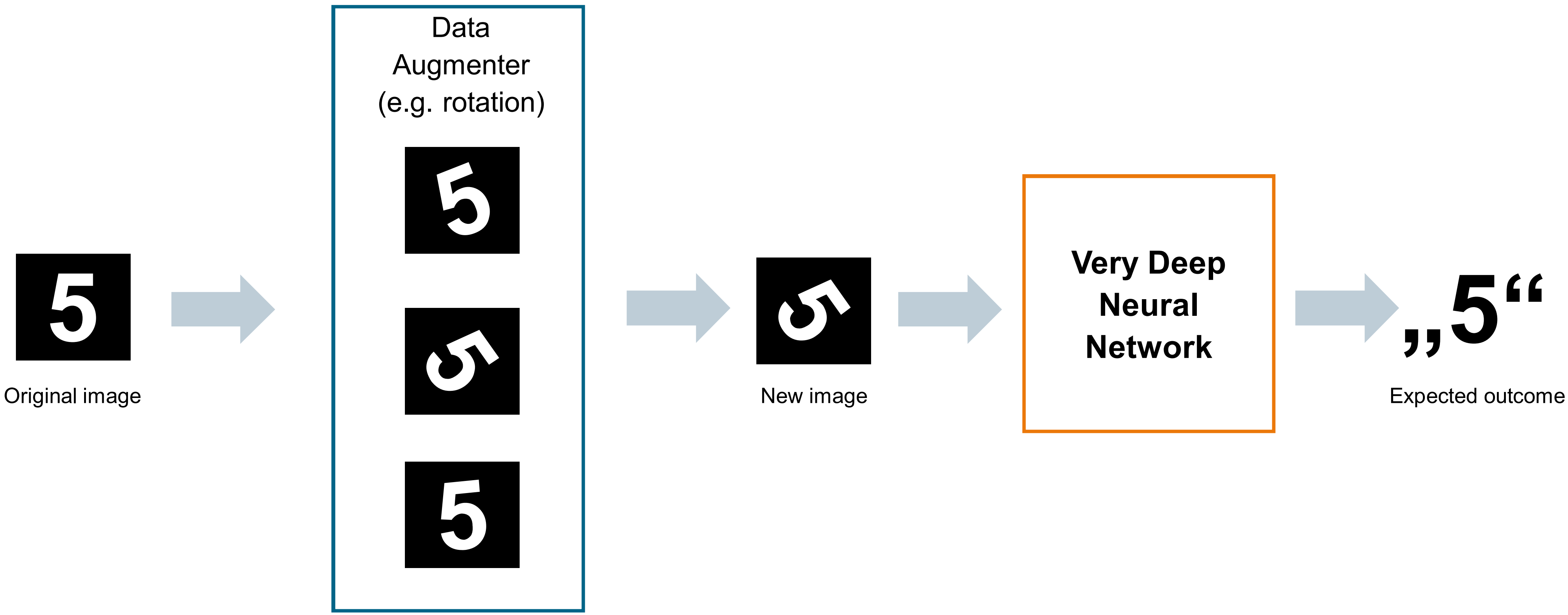
Training data is big, but not big enough.

Training data is very limited.

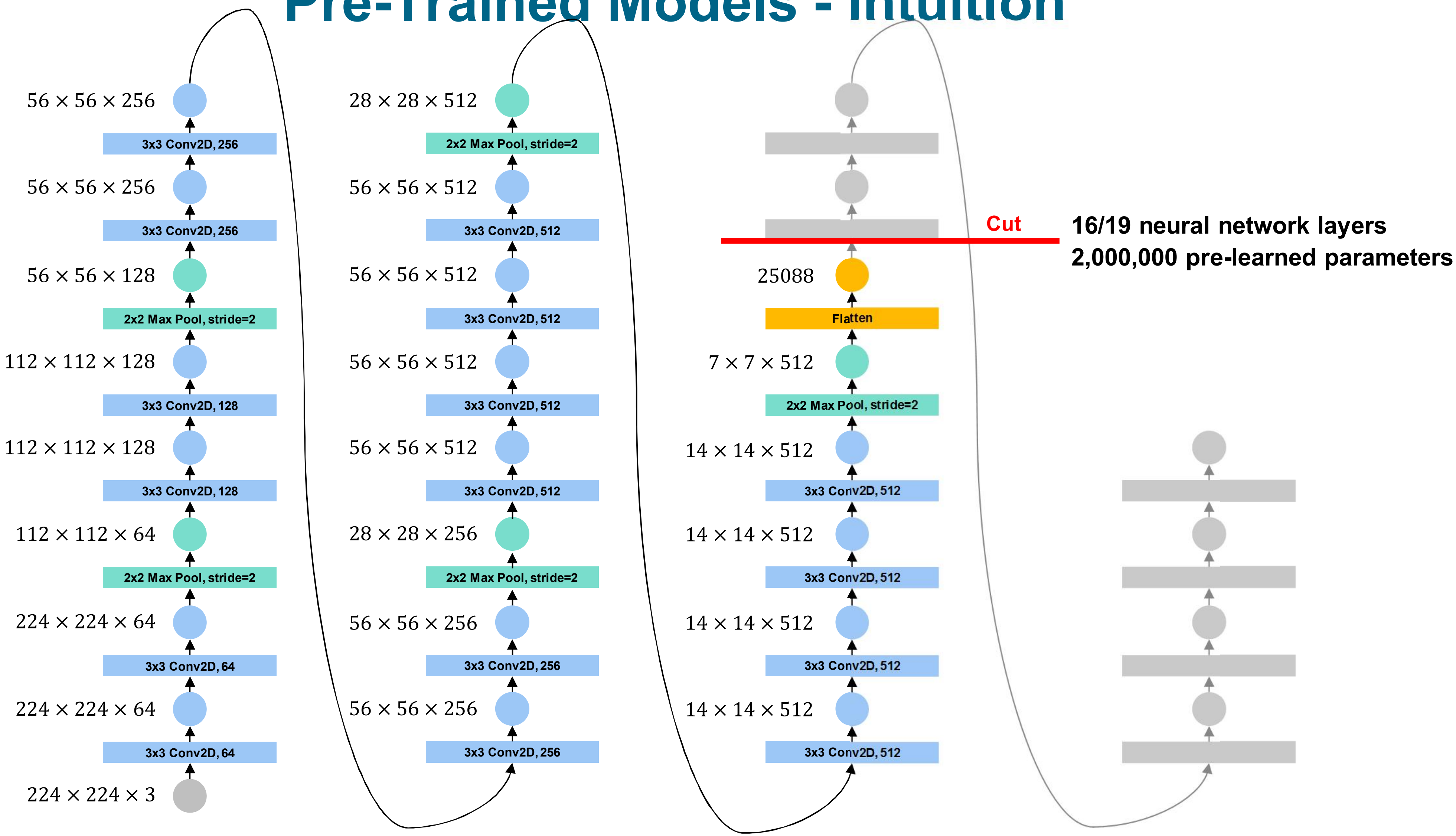
Training needs lots of data and the forward/backward computations are too expensive (take too long).

Model does not fit on a single machine. (Not covered today)

Data Augmentation



Pre-Trained Models - Intuition



Pre-Trained Models

Modules trained on ImageNet (ILSVRC-2012-CLS)

Inception and Inception-ResNet

- Inception V1: classification, feature_vector.
- Inception V2: classification, feature_vector.
- Inception V3: classification, feature_vector.
- Inception-ResNet V2: classification, feature_vector.

MobileNet

MobileNets come in various sizes controlled by a multiplier for the depth (number of features), and trained for various sizes of input images. See the module documentation for details.

- MobileNet V1

	224x224	192x192	160x160	128x128
100%	classification feature_vector	classification feature_vector	classification feature_vector	classification feature_vector
75%	classification feature_vector	classification feature_vector	classification feature_vector	classification feature_vector
50%	classification feature_vector	classification feature_vector	classification feature_vector	classification feature_vector
25%	classification feature_vector	classification feature_vector	classification feature_vector	classification feature_vector

- MobileNet V1 instrumented for quantization with TF-Lite ("/quantops")

	224x224	192x192	160x160	128x128
100%	classification feature_vector	classification feature_vector	classification feature_vector	classification feature_vector

https://www.tensorflow.org/hub/modules/image

```
import tensorflow as tf
import tensorflow_hub as hub


# Define the input placeholder for the image data.
image_data = tf.placeholder(tf.float32, [None, 224, 224, 3])

# Load the blackbox feature extractor for image data.
image_feature_extractor = hub.Module(
    'https://tfhub.dev/google/imagenet/inception_v3/feature_vector',
    trainable=False)
extracted_features = image_feature_extractor(image_data)

# Define the rest of the model.
...

# Train the model on our (small) dataset to solve a complicated task.
with tf.Session() as sess:
    sess.run(tf.global_variables_initializer())
    sess.run(tf.tables_initializer())

    sess.run(update_op, feed_dict=image_data: images}))
```

(A)synchronous Distributed Training