

Convolutional Neural Networks (CNN) for data classification

Gianluca Filippini
EBV / FAE -ML Specialist



Ray Kurzweil

2017: We will create systems and robots, which are smarter than us

Raymond Kurzweil, Google's Director of Engineering, is a well-known futurist with a high-hitting track record for accurate predictions.

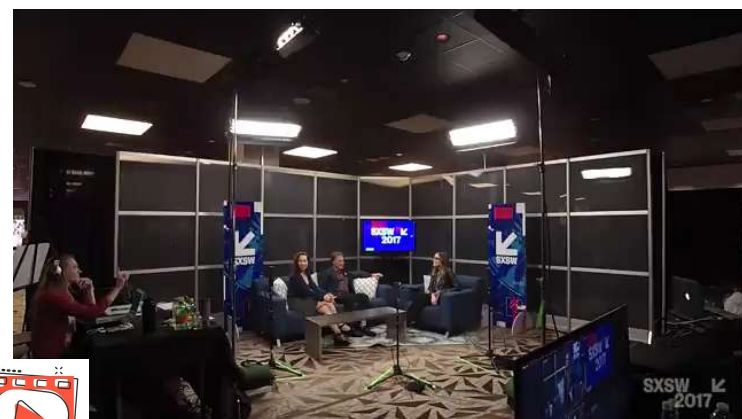
*"**2029** is the consistent date I have predicted for when an AI will pass a valid **Turing test** and therefore **achieve human levels of intelligence**. I have set the date **2045** for the '**Singularity**' which is when we will multiply our effective intelligence a billion fold by merging with the intelligence we have created"*

Using big data, computer programs (artificial intelligence) will be capable of analyzing massive amounts of information, identifying trends and using that knowledge to come up with solutions to the world's biggest problems..

https://en.wikipedia.org/wiki/Ray_Kurzweil

<https://futurism.com/kurzweil-claims-that-the-singularity-will-happen-by-2045>

https://en.wikipedia.org/wiki/Technological_singularity





Alan Turing



Mind 49 :
The Imitation Game

1950: The Imitation Game.

Computing Machinery and Intelligence (*Mind* 49, 433-460)

I propose to consider the question, "Can machines think?"

This should begin with definitions of the meaning of the terms "machine" and "think."

[...]

It is played with three people, a man (A), a woman (B), and an interrogator (C) who may be of either sex. The interrogator stays in a room apart front the other two. The object of the game for the interrogator is to determine which of the other two is the man and which is the woman. He knows them by labels X and Y, and at the end of the game he says either "X is A and Y is B" or "X is B and Y is A."

[...]

We now ask the question, "What will happen when a machine takes the part of A in this game?" Will the interrogator decide wrongly as often when the game is played like this as he does when the game is played between a man and a woman? These questions replace our original, "Can machines think?"

<https://www.csee.umbc.edu/courses/471/papers/turing.pdf>

The Turing machine was invented in **1936**. Turing called it an "a-machine" (automatic machine).

https://en.wikipedia.org/wiki/Turing_machine

Sept. 4, 2019: A Breakthrough for A.I. Technology, passing an 8th-Grade Science Test



Oren Etzioni

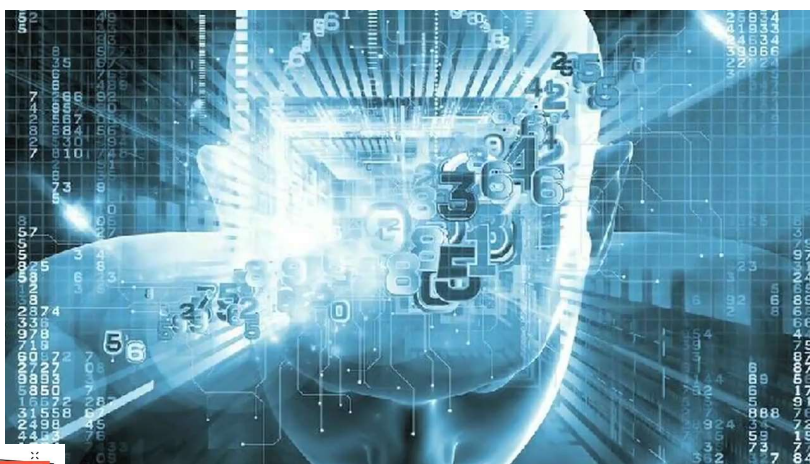
The **Allen Institute for Artificial Intelligence** unveiled a new system that passed the test with room to spare. It correctly answered more than 90 percent of the questions on an eighth-grade science test and more than 80 percent on a 12th-grade exam

**The
New York
Times**

<https://www.nytimes.com/2019/09/04/technology/artificial-intelligence-aristo-passed-test.html>

<https://allenai.org/>

AI2: The Allen Institute for Artificial Intelligence is a research institute founded by late Microsoft co-founder **Paul Allen**. The institute seeks to achieve scientific breakthroughs by constructing AI systems with reasoning, learning, and reading capabilities.

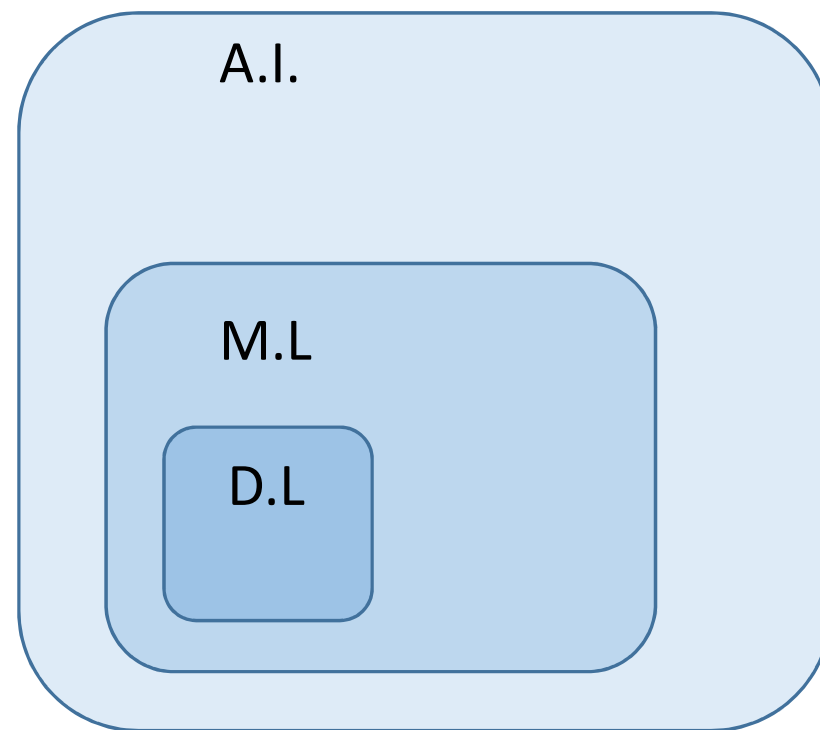


Aristo: The goal is to design an artificially intelligent system that **can successfully read and understand science texts** and ultimately demonstrate its knowledge by **passing an AP biology exam**. The focus of the project is explained by the guiding philosophy that artificial intelligence is about having a mental model for how things operate and refining that mental model based on new knowledge

<https://allenai.org/team/orene/videos.html>

https://en.wikipedia.org/wiki/Oren_Etzioni





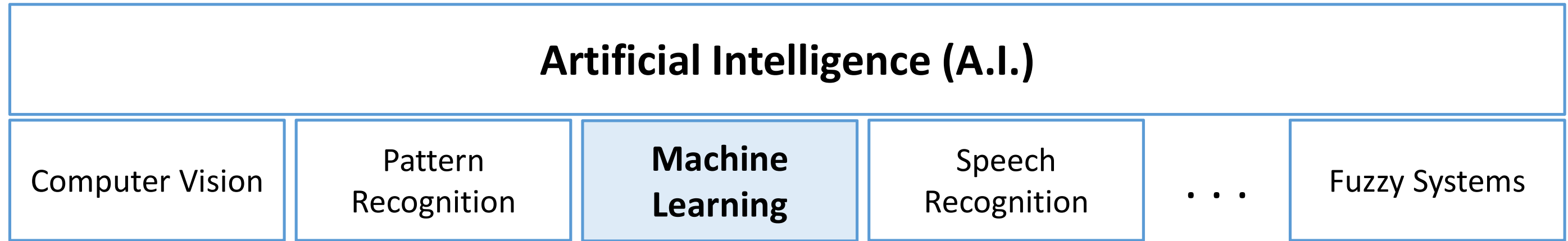
Artificial Intelligence

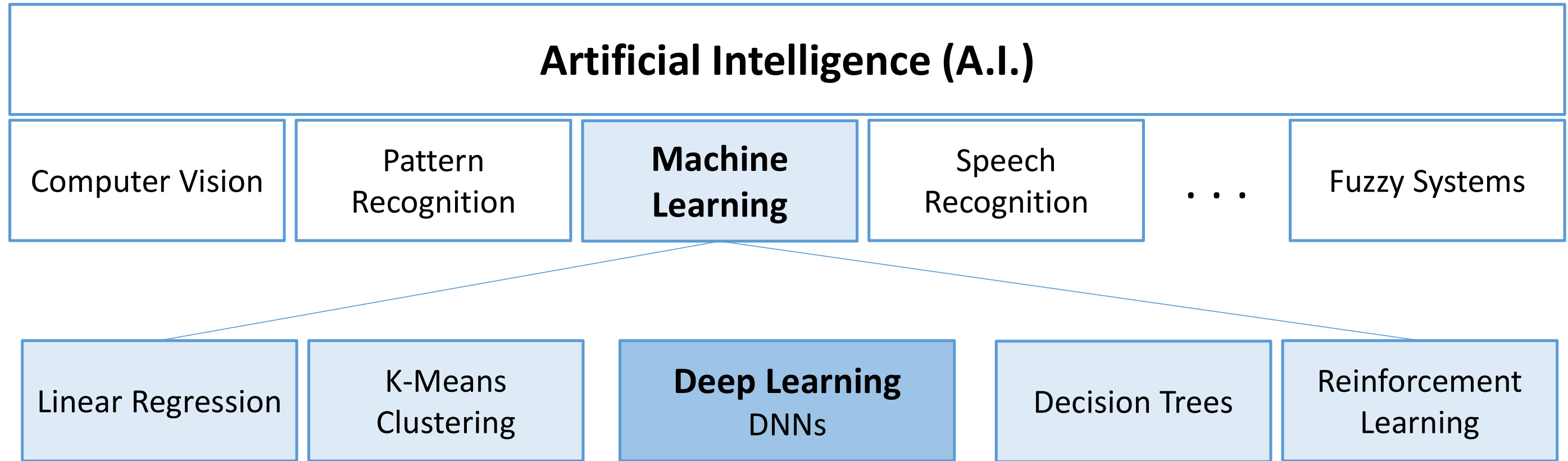
The goal of Artificial Intelligence (A.I) is to provide algorithms and techniques to solve problems that humans perform intuitively and near automatically, but are otherwise very challenging for computers.
(inferring, planning, heuristics etc.)

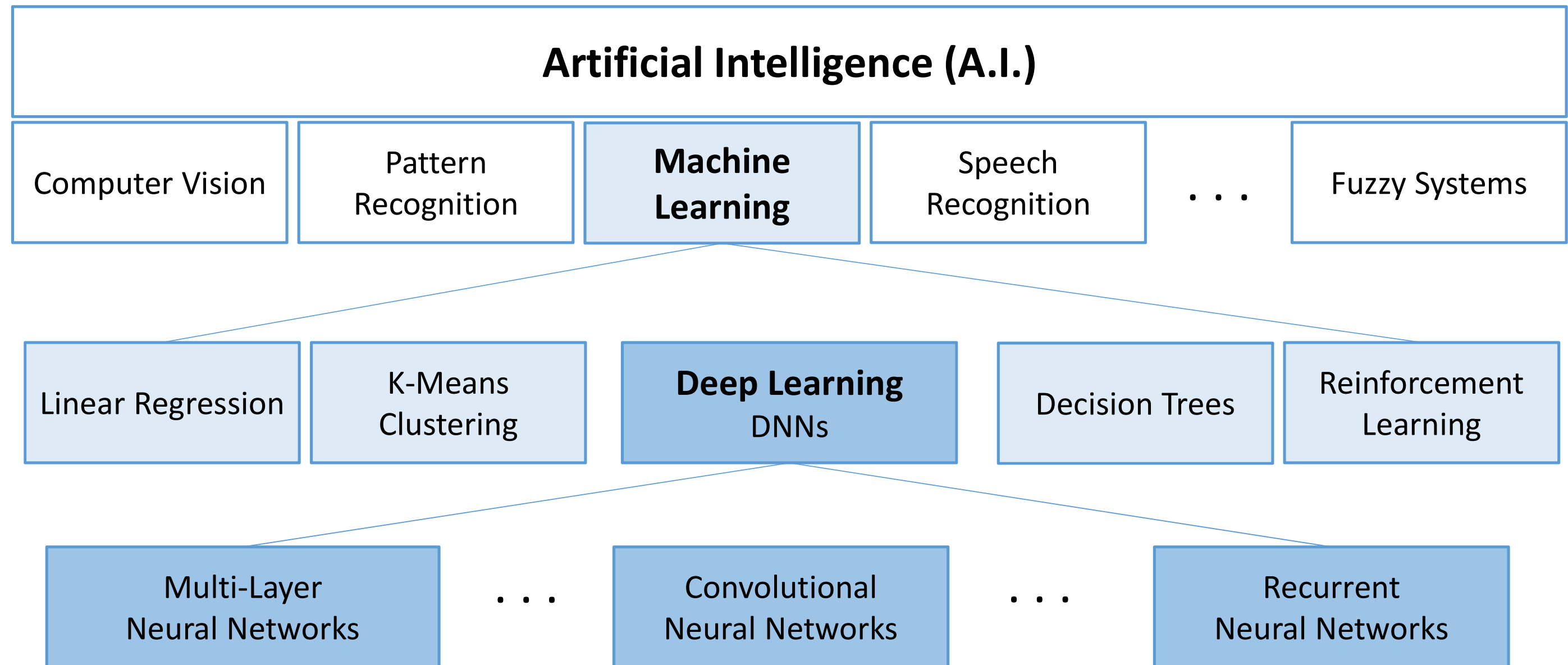
Machine Learning is a subset of AI which focuses on **pattern recognition** and learning from data.

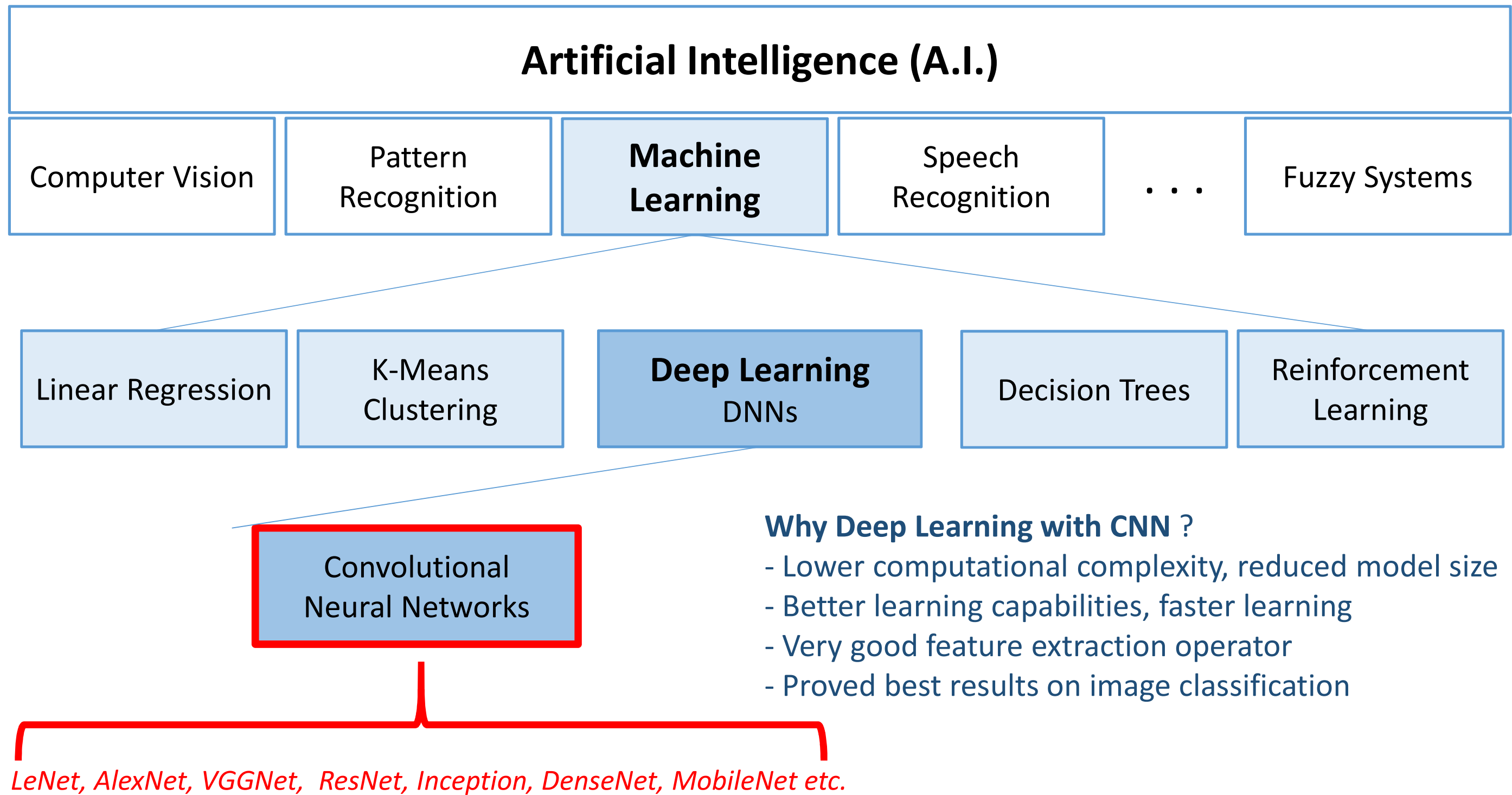
Deep Learning is a subset of ML with peculiar algorithm structures that are very efficient on specific tasks like data classification, object recognition etc. etc.
(computation is intensive even for modern computer systems)

Research is driven by the scientist community, often with the contribution of industry leaders (Google, Facebook etc)







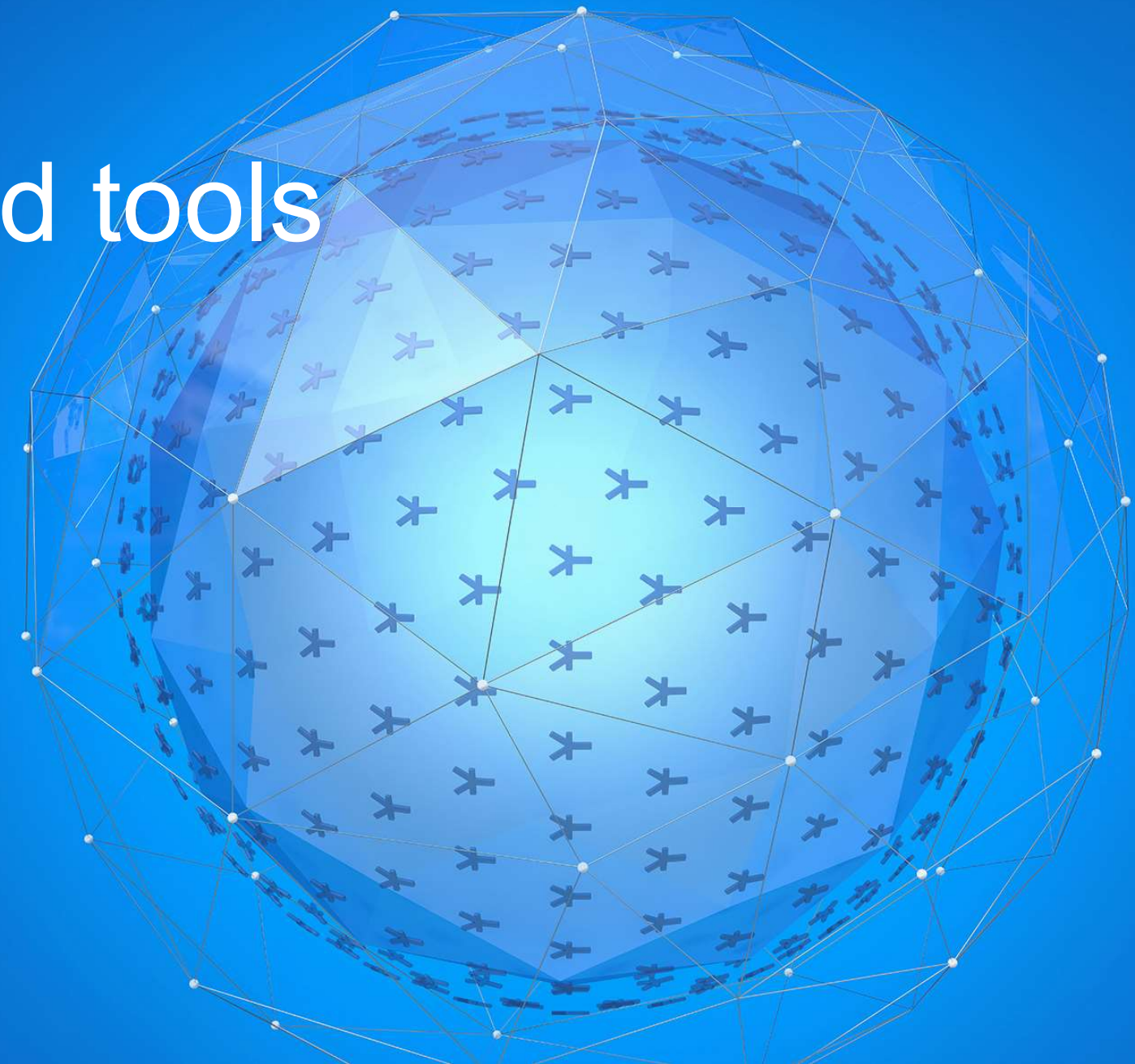


Why Deep Learning with CNN ?

- Lower computational complexity, reduced model size
- Better learning capabilities, faster learning
- Very good feature extraction operator
- Proved best results on image classification

LeNet, AlexNet, VGGNet, ResNet, Inception, DenseNet, MobileNet etc.

Machine Learning, data and tools



Machine learning is based on training data to learn data patterns. How we use the training data depends on the specific ML type.

Supervised Learning: an algorithm is given both the *input* and the *output* result at the same time. The algorithm goal is to map the input to the correct output by automatically learning a pattern on *multiple* input data. (*support vector machines, random forests, neural network ..*) This technique is based on “labeling” the input data. Each data is assigned to a specific class (i.e. the output result)

Semi-Supervised Learning: the input data are not fully labeled. The algorithm has to learn patterns from the fully labeled set of data and also try to improve its own performance by using the remaining data that were not previously classified.

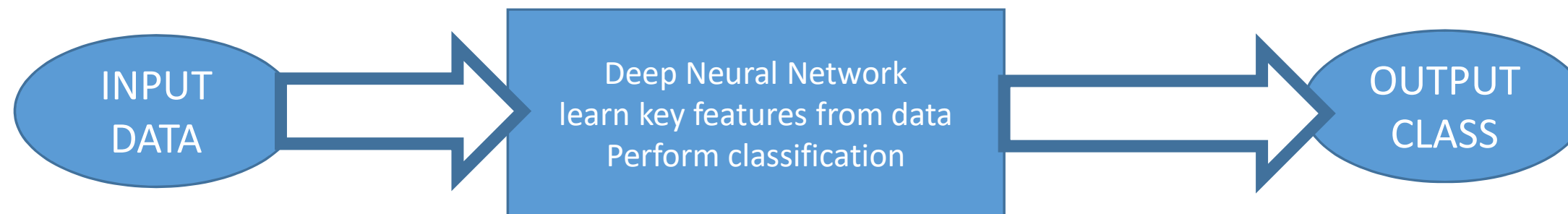
Unsupervised Learning: there are no information on the target output for each input data. The algorithm must search and distinguish features in the incoming data set, *automatically*.

<https://machinelearningmastery.com/types-of-learning-in-machine-learning/>

Classical method: write code for features extraction and classification



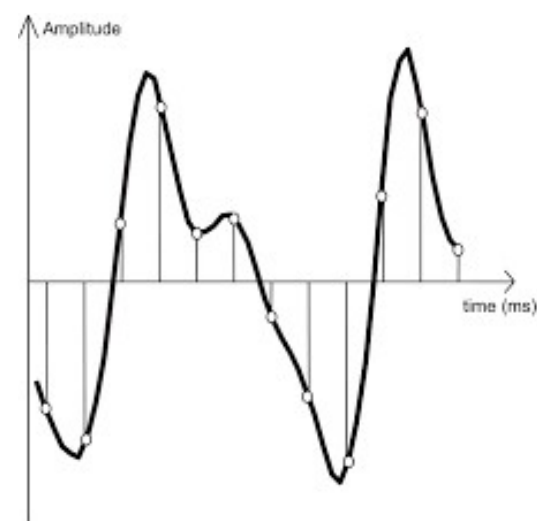
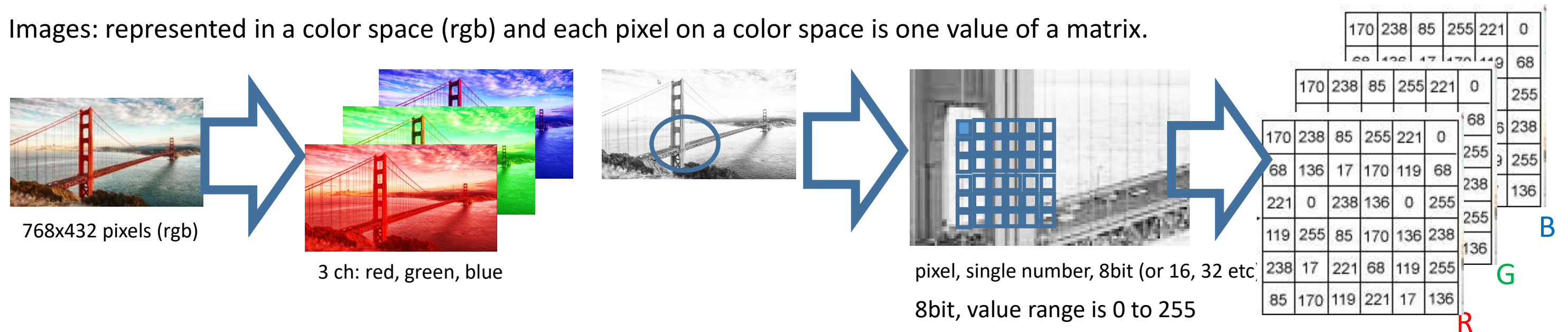
Deep NN method: the network will learn important features *automatically* from the data itself



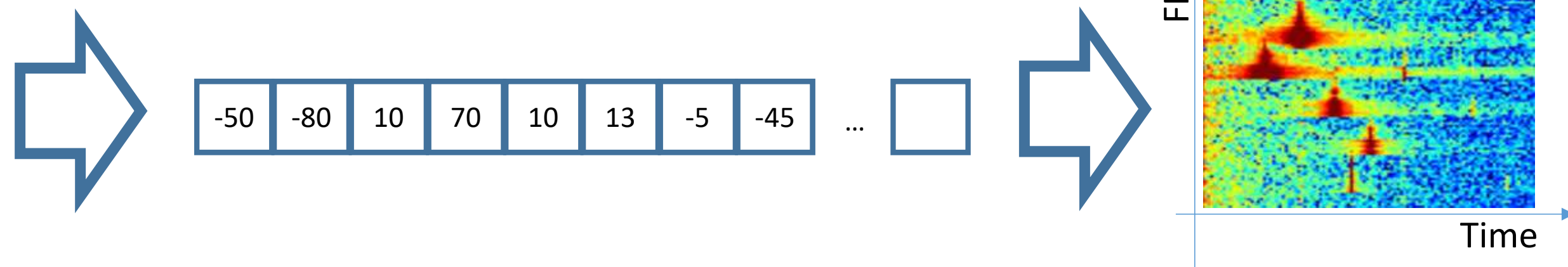
Data, Signals, Numbers ...

NN do apply to many types of signals, for example images (video) and audio recordings. We consider signals to be digital. Digital signals in their lowest representation are numbers. Numbers might have different format (integer, floating point) but are the fundamental building block to understand input datasets for neural networks.

Images: represented in a color space (rgb) and each pixel on a color space is one value of a matrix.



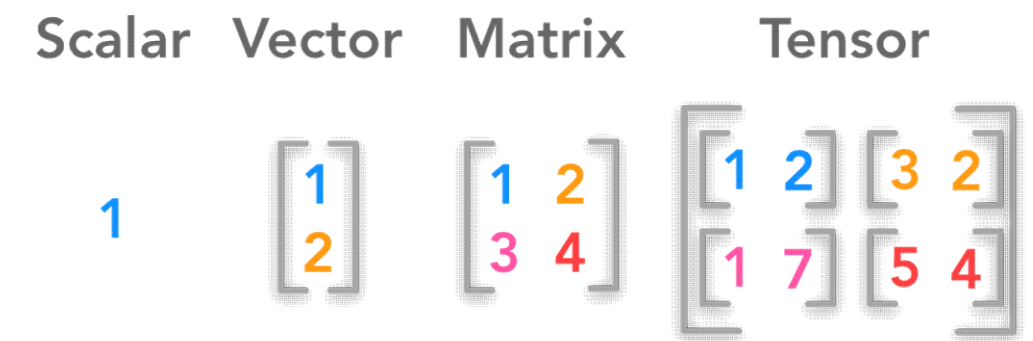
Analog audio signals are converted to digital signals in different digital formats (DSD, PCM etc)
For PCM the audio level is sampled at a specific frequency and quantized with a specific amount of bits.
Each sample is a number, the collection of sample values does form an array.



Tensor

A tensor is a mathematical construct that enables us to represent entities that otherwise we would not be able to describe. For a simple understanding of what is a tensor for ML we can think of:

- A scalar is a single number
- A vector is an array of numbers.
- A matrix is a 2-D array
- A tensor is a N-dimensional array with $n > 2$



Data is often multi-dimensional. Tensors can play an important role in ML by encoding multi-dimensional data. A picture is represented by three fields: width, height and depth (color). It makes sense to encode it as a 3D tensor. However, more than often we are dealing with tens of thousands of pictures. Hence this is where the forth field, **sample size** comes into play. A series of images in a dataset can be stored in a 4D tensor. This representation allows problems involving big data to be solved easily

Dataset: a collection of data points

Data point: a single instance of data. An image, an audio chunk, a text chunk is a data point.

Datasets are used to allow NN to learn features needed to perform data classification. By learning how to detect complex patterns into the input data the network can distinguish object in images, words in sounds recording etc.

<https://towardsdatascience.com/quick-ml-concepts-tensors-eb1330d7760f>

<https://hadrienj.github.io/posts/Deep-Learning-Book-Series-2.1-Scalars-Vectors-Matrices-and-Tensors/>



Data Classification: We perform a test on input data to define if the data point is part of a class or not. The “classification function” effectiveness can be measured based on hit/miss rates (binary classification)

- TP:** True Positive. Input Data is “positive” and is classified as “positive”.
- TN:** True Negative. Input Data is “negative” and is classified as “negative”.
- FP:** False Positive. Input Data is “negative” and is classified as “positive”.
- FN:** False Negative. Input Data is “positive” and is classified as “negative”.

Accuracy = $TP+TN / total_data$

| | | Predicted Data | |
|-----------|---|----------------|----|
| | | P | N |
| Real Data | P | TP | FN |
| | N | FP | TN |

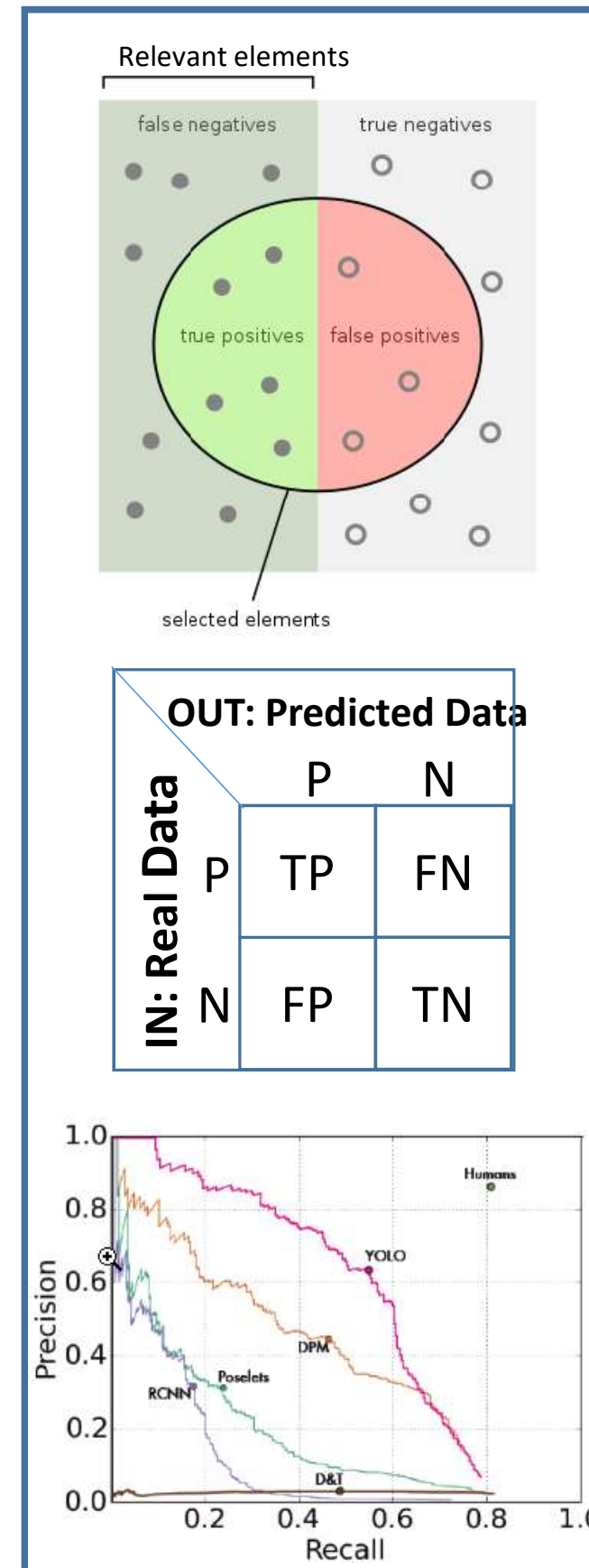
Precision = $TP / (TP+FP)$

| | | Predicted Data | |
|-----------|---|----------------|----|
| | | P | N |
| Real Data | P | TP | FN |
| | N | FP | TN |

(Sensitivity, hit rate)

Recall = $TP / (TP+FN)$

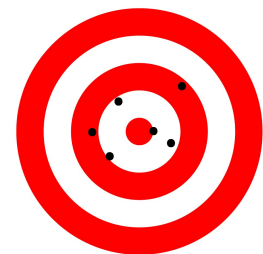
| | | Predicted Data | |
|-----------|---|----------------|----|
| | | P | N |
| Real Data | P | TP | FN |
| | N | FP | TN |



High Accuracy
High Precision



High Accuracy
Low Precision



Low Accuracy
High Precision



Low Accuracy
Low Precision



| | | P | N |
|---|----|----|----|
| | | 40 | 20 |
| P | 10 | 30 | |
| N | | | |

Accuracy = $(40+30)/100 = 70\%$

Precision = $40/(40+10) = 80\%$

Recall = $40/(40+20) = 66\%$

https://en.wikipedia.org/wiki/Sensitivity_and_specificity

https://en.wikipedia.org/wiki/Evaluation_of_binary_classifiers

Data is the new “gold”

Good training of a neural network is mostly based on the *quality* of the dataset and *expertise* of the data scientists.

Dataset quality is related to the coverage of a “core” of data that will allow the final network to generalize better on the remaining datapoints from the field. Modern datasets are databases of *signal samples* and *metadata* associated to the signal itself (class label, bounding boxes, coordinates etc). Dataset tends to be very large for modern networks and an accurate work of “labeling” and manual classification is the real value of a “good” dataset.

The expert data scientist knows how to train a neural network for a specific dataset, tweaking hyper-parameters, starting-stopping training etc.

Public Datasets

Private companies will rarely share their datasets (service fees).

These datasets are usually built over time with many resources to gain a competitive advantage. But *scientific communities* have built many public datasets that are available to learn NN. Some of them are famous for historical reasons:

MNIST : handwritten digits

Modified National Institute of Standards and Technology

https://en.wikipedia.org/wiki/MNIST_database



CIFAR-10 : images of animals and cars

Canadian Institute For Advanced Research

<https://en.wikipedia.org/wiki/CIFAR-10>

contains 60,000 32x32 color images in 10 different classes
airplanes, cars, birds, cats, deer, dogs, frogs, horses, ships,
and trucks

deer



dog

frog

PASCAL-VOC

Pattern Analysis, Statistical modeling,

Computational Learning

<http://host.robots.ox.ac.uk/pascal/VOC/>



ImageNet

Labeled object image database, used in the ImageNet Large Scale Visual Recognition Challenge <http://www.image-net.org/>
<https://devopedia.org/imagenet>

One of the largest databases for object recognition, 14M images over thousands of classes. By Stanford University is “the” benchmark for image classification



CoCo

Common Objects in Context.

Started by Microsoft is now a large contribution from different companies and univ. around wwide.
<https://www.microsoft.com/en-us/research/publication/microsoft-coco-common-objects-in-context/>

COCO is a large-scale object detection, segmentation, and captioning dataset.
<http://cocodataset.org/#home>



Kaggle

Much more than a dataset.

<https://www.kaggle.com/>

Hosted by Google, is an online community of data scientists and machine learners.

Kaggle allows users to find and publish data sets, work with other data scientists and machine learning engineers, and **enter competitions to solve data science challenges.**

The Kaggle logo, consisting of the word 'kaggle' in a lowercase, blue, sans-serif font.

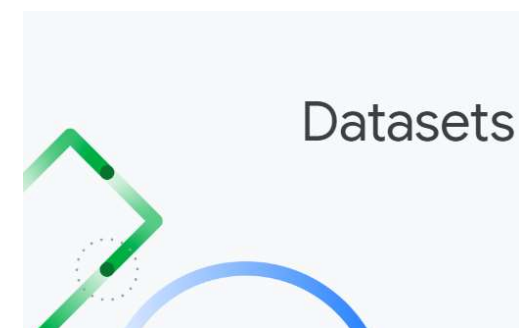
Even More Datasets...

Long list from Wikipedia: image, sound, text, biological, aerial etc. etc.

https://en.wikipedia.org/wiki/List_of_datasets_for_machine-learning_research

Google periodically releases data of interest to researchers in a wide range of areas:

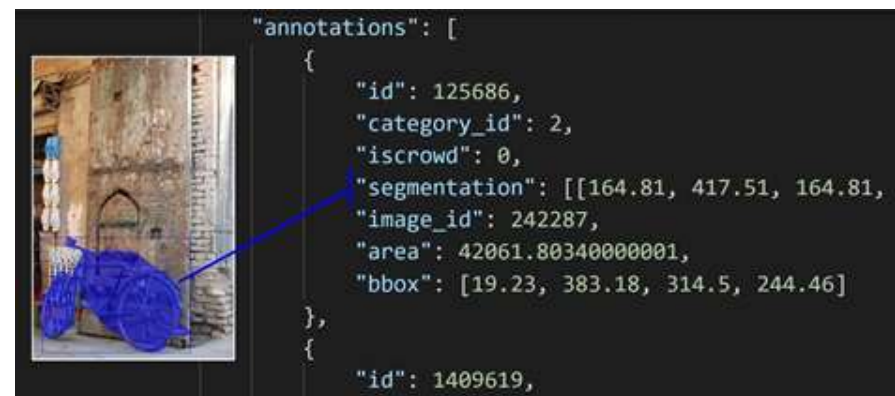
<https://ai.google/tools/datasets/>



Datasets & Tips:

- Public databases to compare performance and save time on your R&D.
- Save time with *pre-trained* models (Model Zoos) <https://modelzoo.co/>
- Your *private* dataset are the most accurate and expensive (time and resources)

Always *study* metadata formats *before* training any networks. Metadata are



the most important value added to a carefully selected dataset.

There are *commercially* available datasets that have been built with extended metadata (automotive, medical etc.)

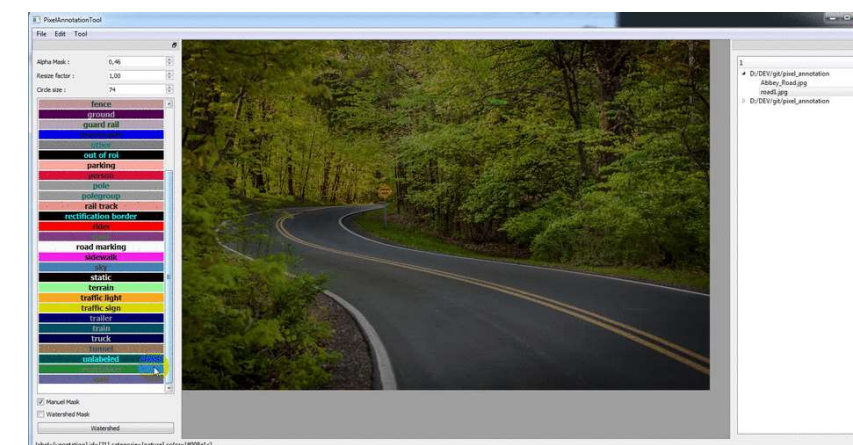
Annotation Tools for Dataset

Image&Text annotation is the process of manually defining regions in an image/text and creating *structured* descriptions of those regions (*yaml, json etc.*). Annotation is also used to classify any type of signals that are represented as a image.

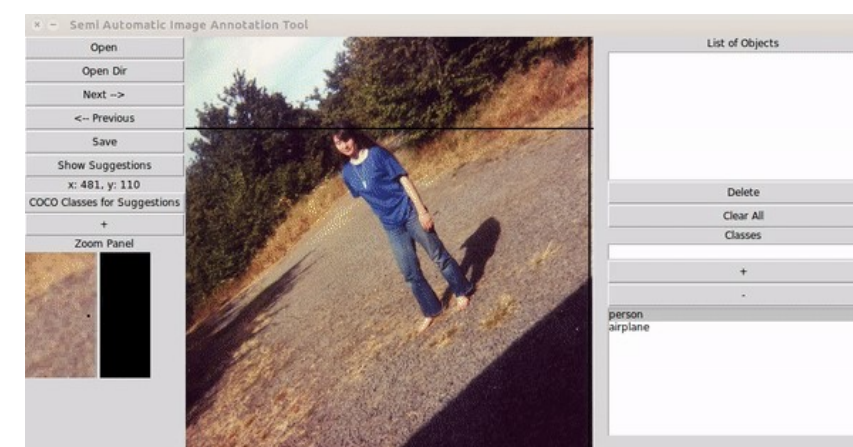
- [LabelImg](#): is an *open source* graphical image annotation tool that you can use to label object bounding boxes in images.
- [PixelAnnotationTool](#): Software that allows you to annotate images in directories. The method is *pseudo manual* because it uses the algorithm [watershed marked](#) of OpenCV.
- [Anno-Mage](#): is an advanced open source image annotation tool that incorporates an existing state-of-the-art object detection model (*RetinaNet*) to show suggestions of 80 common object classes while annotation to reduce the amount of human labeling tasks.
- [ImageTagger](#): ImageTagger is an open source online platform for collaborative image labeling.
- [CVAT](#): Computer Vision Annotation Tool (CVAT) is a free interactive video and image annotation tool for computer vision.
- [Fast Annotation Tool](#): Fast Annotation Tool is an open source online platform for collaborative image annotation for image classification, optical character reading, etc.
- [Labelbox](#): Labelbox is a platform for data labeling, data management, and data science. Its features include image annotation, bounding boxes, text classification, and more.
- [Prodigy](#): Prodigy is an annotation tool for various machine learning models such as image classification, entity recognition and intent detection. You can stream in your own data from live APIs, update your model in real-time, and chain models together to build more complex systems.
- [TrainingData.io](#): TrainingData.io is a medical image annotation tool for data labeling. It supports DICOM image format for radiology AI
- [Supervise.ly](#): provides services and image annotation and data management tool for machine learning models. Also includes a self-hosted infrastructure for training your machine learning models and continuing to improve them with human-in-the-loop.

<https://awesomeopensource.com/projects/annotation-tool>

<https://lionbridge.ai/articles/image-annotation-tools-for-computer-vision/>



PixelAnnotationTool



Anno-Mage

COCO has five annotation types: for object detection, keypoint detection, stuff segmentation, panoptic segmentation, and image captioning. The annotations are stored using JSON. Please note that the COCO API described on the download page can be used to access and manipulate all annotations. All annotations share the same basic data structure below:



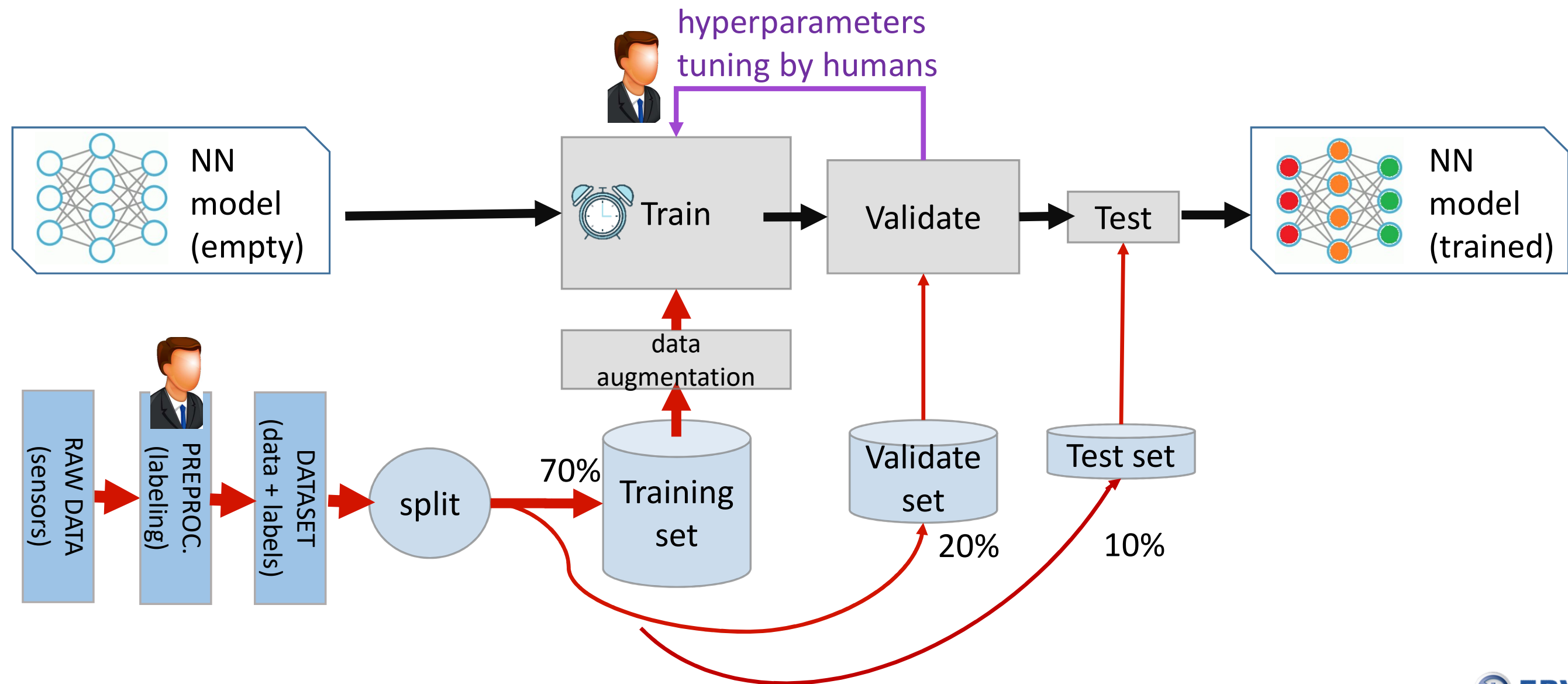
<http://cocodataset.org/#format-data>

Workflow and “data splitting”: the 70-20-10 rule

Once the dataset is available the collection is splitted in three parts (*equally randomized to include all categories*):

- **Training set** : used to train the network
- **Validation set** : used to provide an unbiased evaluation of a model at the end of each epoch
- **Test set** : used for final model evaluation (accuracy)






Dataset partitions should *never* be mixed while training to avoid model evaluation bias on specific datapoints.



DeepLearning Frameworks

A *Framework* is a set of tools (user interface, code library, debugging/monitoring) that allows to build NN models more easily and quickly, leveraging all the work from a wide community of data scientists around the world.

Frameworks do provide a specific API for multiple languages (python, C, C++) and also a collection of pre-built (and sometimes pre-trained) models and optimized components.

| | | | |
|---|---|------|--|
|  Caffe | Berkley university | 2013 | Developed by Berkley Univ AI Research team became the most used framework for data scientist. Allows C, C++, Python, Matlab programming. Provides a large “Model Zoo” with pre-trained networks. Open Source. |
|  TensorFlow Google |  Keras Francois Chollet | 2015 | Developed at Google, joined with Keras (Francois Chollet) in 2019. Written in C++,Python. Provides an high-level API to start writing NNs. Tensorflow allows the usage of CPU and GPU(CUDA). One of the most used in both industry and academic world. |
|  PYTORCH | Facebook | 2016 | Developed at Facebook, based on C, Python. Allows CPU/GPU(CUDA) computation. Provide tensor computation and uses dynamic computation graphs. <i>Instead of predefined graphs with specific functionalities, PyTorch provides a framework for us to build computational graphs as we go, and even change them during runtime.</i> |
|  mxnet | Apache Foundation | 2015 | Apache MXNet is a scalable framework used to train NN. MXNet supports multiple languages like C++, Python, R, Julia, Perl etc. It is very scalable from multiple GPU to distributed servers infrastructures. MXNet has been chosen by Amazon for its Web Services’ Deep Learning frameworks. |

<https://skymind.ai/wiki/comparison-frameworks-dl4j-tensorflow-pytorch>

<https://towardsdatascience.com/top-10-best-deep-learning-frameworks-in-2019-5ccb90ea6de>

DeepLearning Frameworks

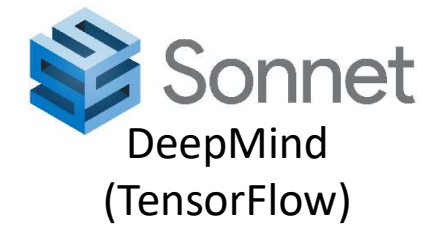


Jeremy Howard 2017

Developed by Jeremy Howard it is a machine learning programming library built on top of PyTorch. It is open source and the goal is to make the programming of NN extremely simple and intuitive. The website <https://www.fast.ai/about/> offers documentation and tutorials.

<https://www.usfca.edu/data-institute/certificates/deep-learning-part-one>
[https://en.wikipedia.org/wiki/Jeremy_Howard_\(entrepreneur\)](https://en.wikipedia.org/wiki/Jeremy_Howard_(entrepreneur))

Even More frameworks...

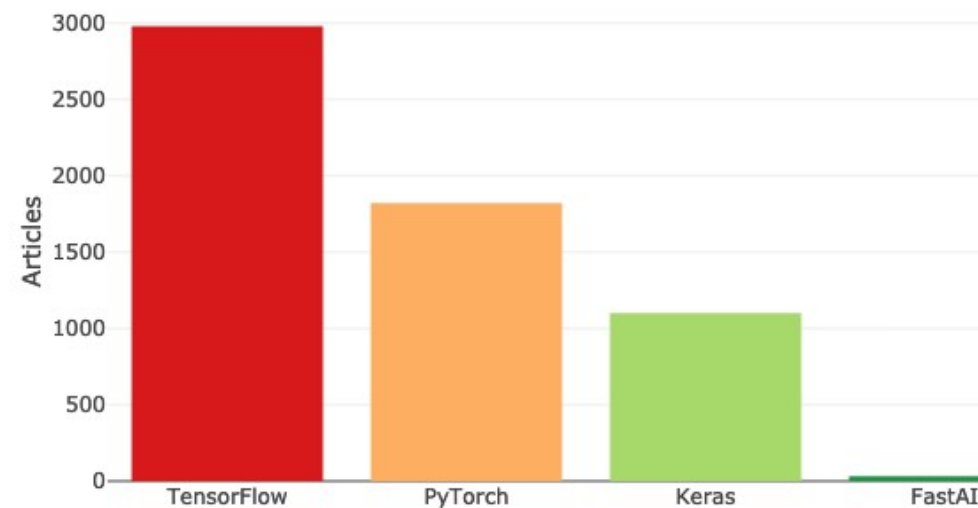


https://en.wikipedia.org/wiki/Comparison_of_deep-learning_software

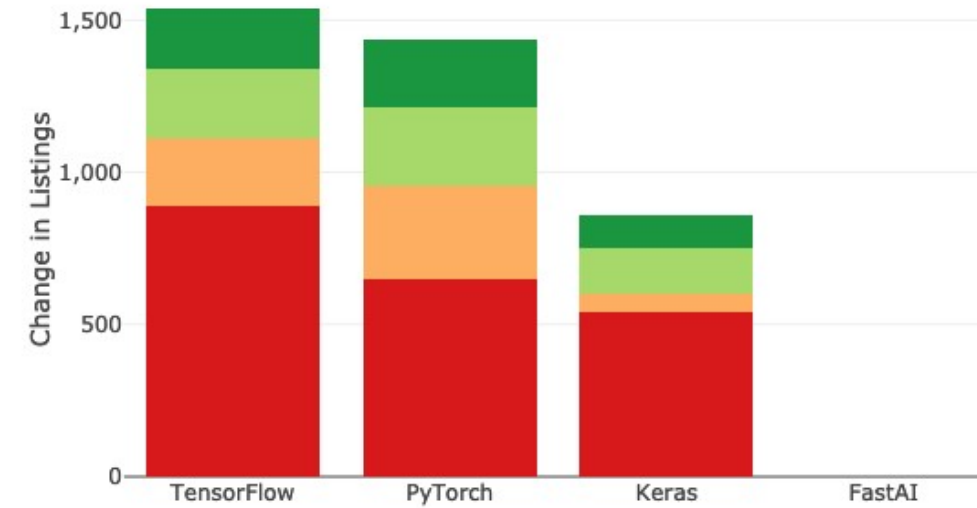
Which one to start from? Research-vs-Industry (April 2019)



Arxiv.org Articles



Online Job Listing



Jeff Hale : <https://towardsdatascience.com/which-deep-learning-framework-is-growing-fastest-3f77f14aa318>

DeepLearnig Frameworks growing trend: 2019



Andrej Karpathy ✓
@karpathy

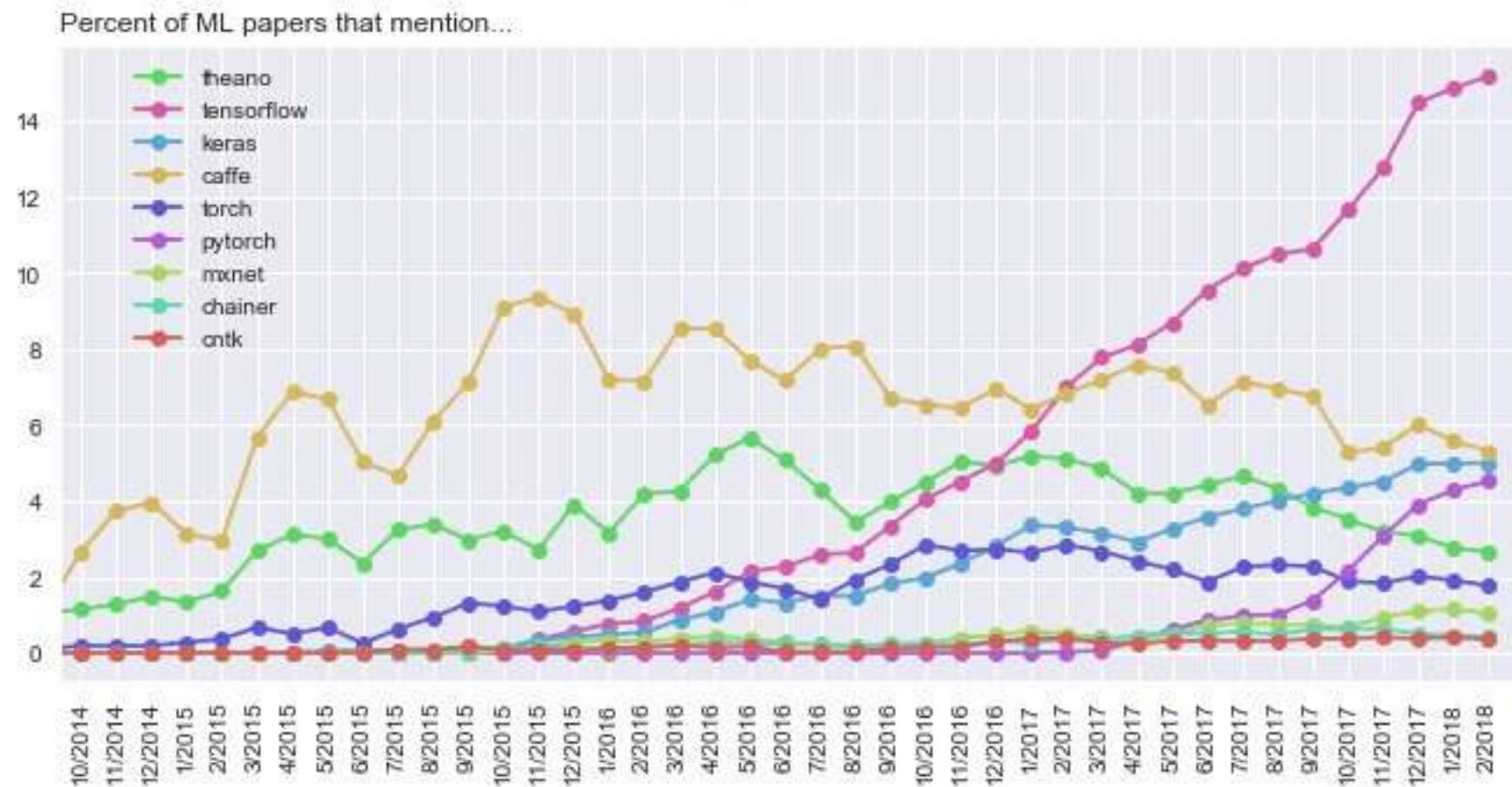
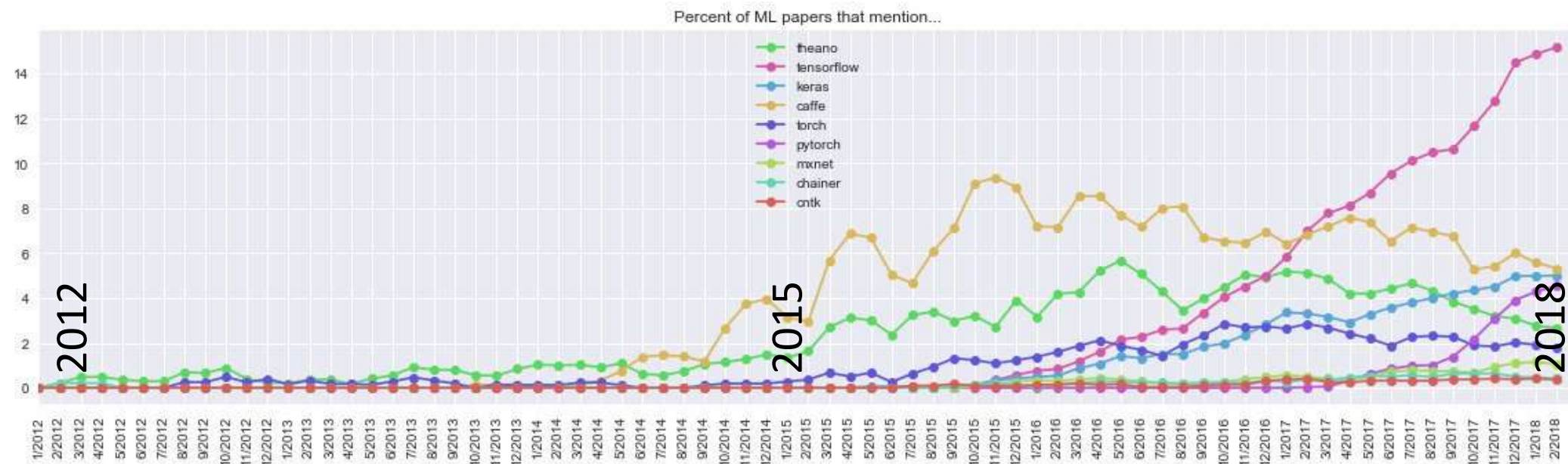
Director of AI at Tesla. Previously a Research Scientist at OpenAI, and CS PhD student at Stanford. I like to train Deep Neural Nets on large datasets.

<https://cs.stanford.edu/~karpathy/>

Director of A.I. and Autopilot Vision at Tesla

On **March, 9 2019** Andrej Karpathy posted an analysis on A.i. Frameworks trends..

*“Unique mentions of deep learning frameworks in arxiv papers (full text) over time, based on 43K ML papers over last 6 years. So far **TF** mentioned in **14.3%** of all papers, **PyTorch 4.7%**, **Keras 4.0%**, **Caffe 3.8%**, **Theano 2.3%**, **Torch 1.5%**, **mxnet/chainer/cntk <1%**.”*



<https://medium.com/@karpathy/a-peek-at-trends-in-machine-learning-ab8a1085a106>

<https://twitter.com/karpathy/status/972295865187512320/photo/1>



DeepLearnig Frameworks growing trend: 2019



Andrej Karpathy ✓
@karpathy

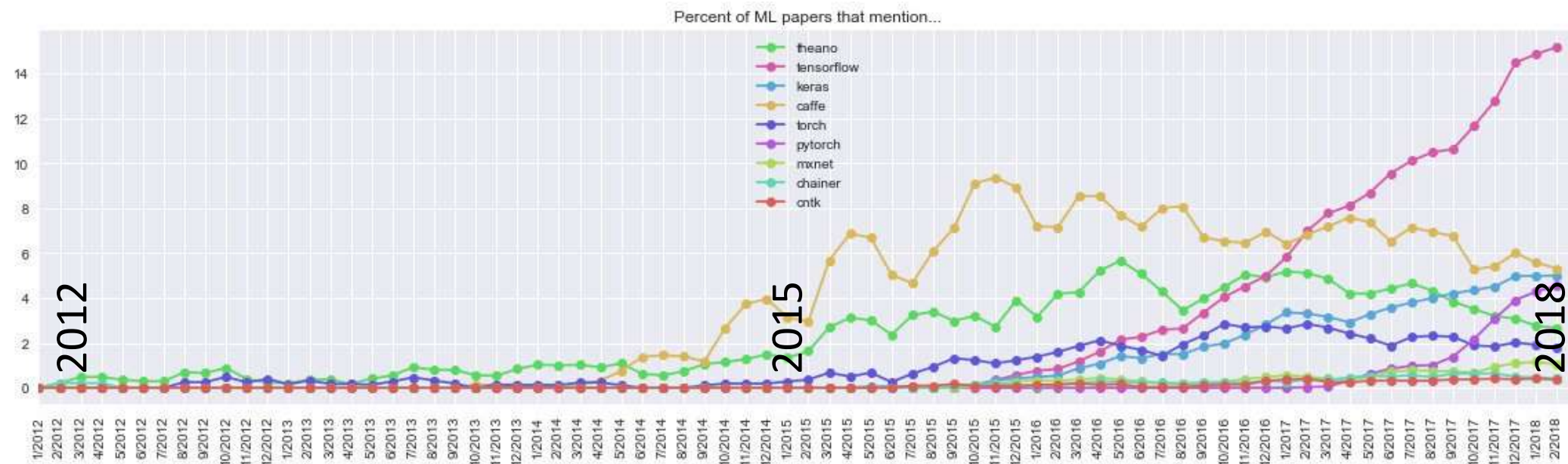
Director of AI at Tesla. Previously a Research Scientist at OpenAI, and CS PhD student at Stanford. I like to train Deep Neural Nets on large datasets.

<https://cs.stanford.edu/~karpathy/>

Director of A.I. and Autopilot Vision at Tesla

On **March, 9 2019** Andrej Karpathy posted an analysis on A.i. Frameworks trends..

*“Unique mentions of deep learning frameworks in arxiv papers (full text) over time, based on 43K ML papers over last 6 years. So far **TF** mentioned in **14.3%** of all papers, **PyTorch 4.7%**, **Keras 4.0%**, **Caffe 3.8%**, **Theano 2.3%**, **Torch 1.5%**, **mxnet/chainer/cntk <1%**.”*



<https://medium.com/@karpathy/a-peek-at-trends-in-machine-learning-ab8a1085a106>

<https://twitter.com/karpathy/status/972295865187512320/photo/1>



DeepLearnig Frameworks growing trend: 2020

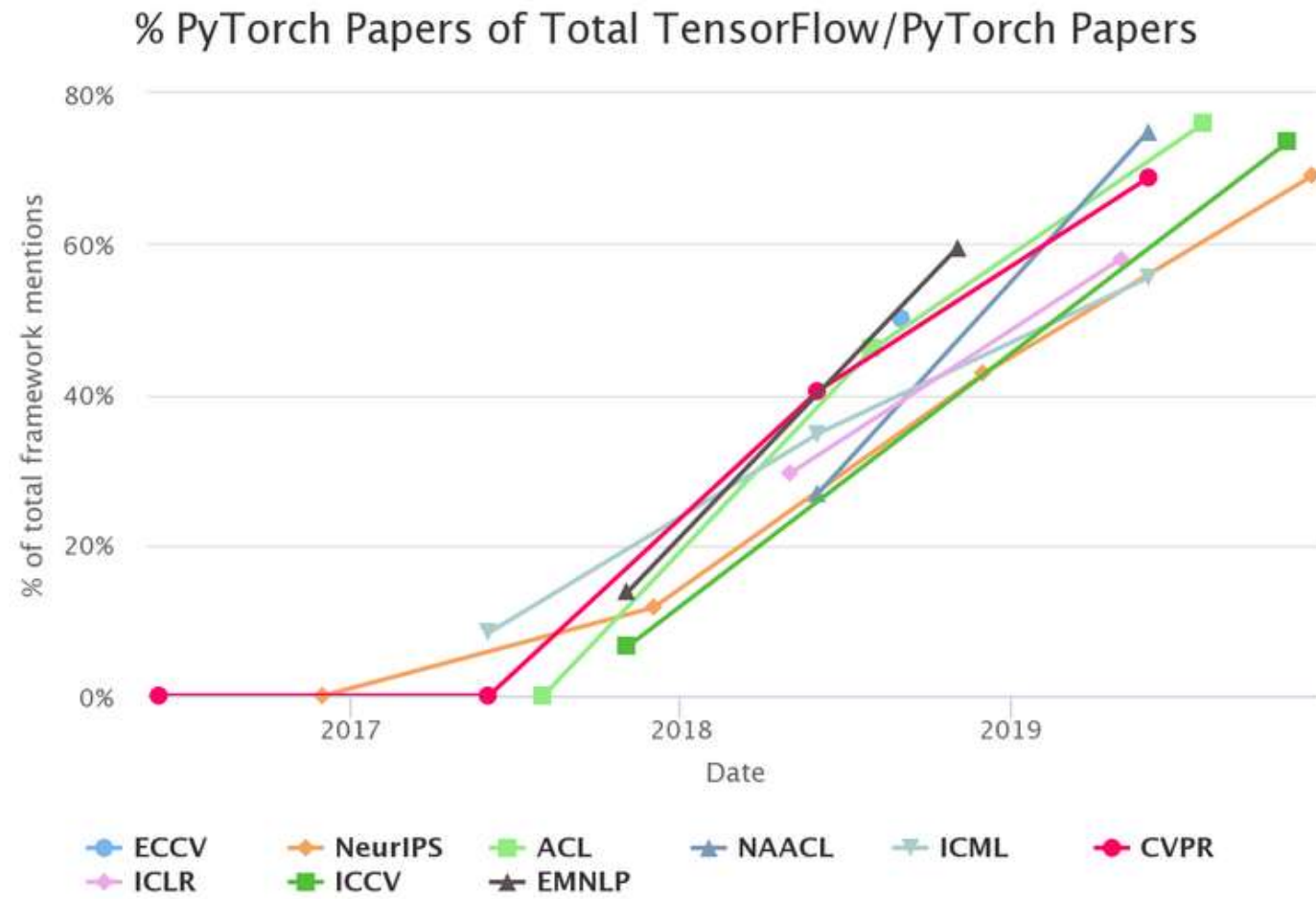


Jeff Hale

<https://towardsdatascience.com/is-pytorch-catching-tensorflow-ca88f9128304>

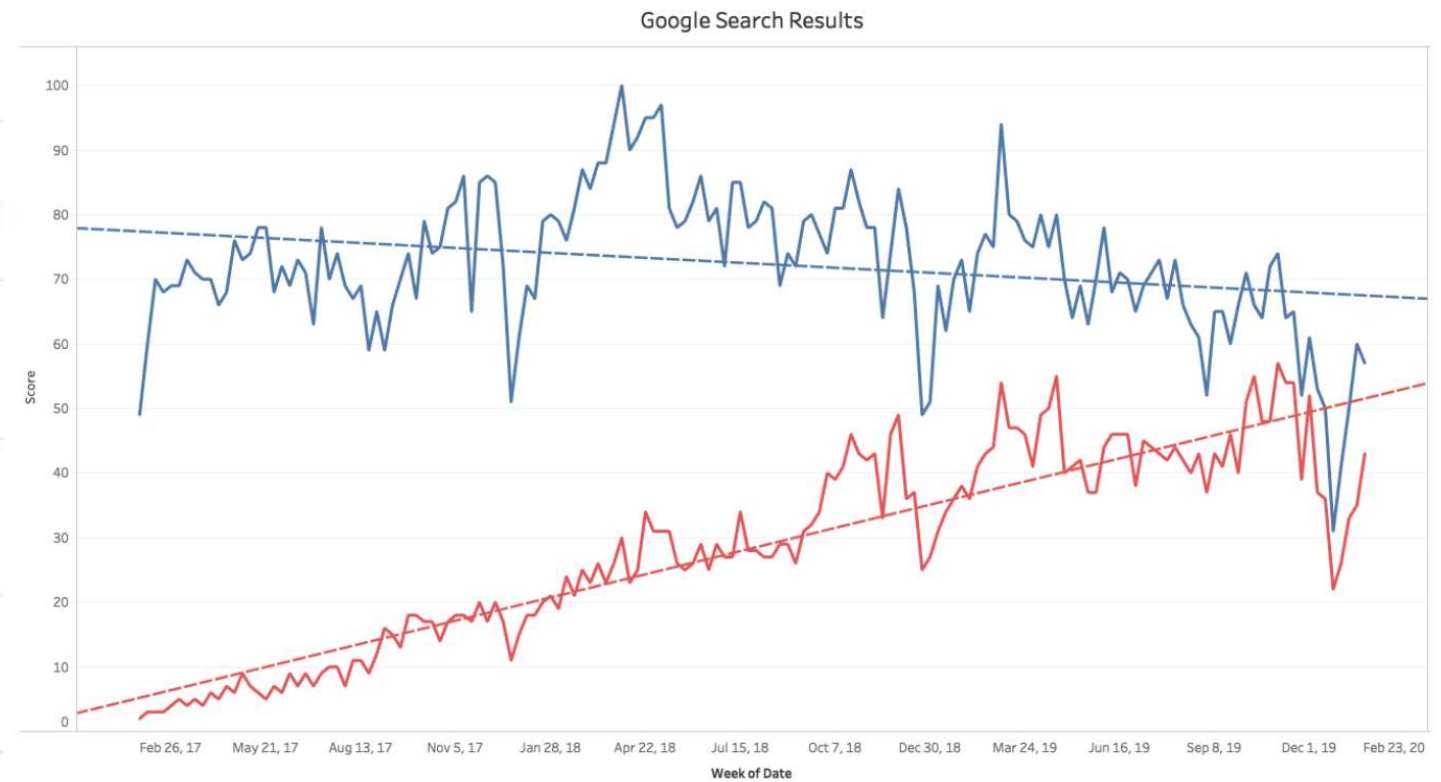
Jan 28, 2020

Research / Academia



Source: <https://chillee.github.io/pytorch-vs-tensorflow/>

Job search



| Framework | Indeed | Monster | Simply Hired | LinkedIn | Mean |
|------------|--------|---------|--------------|----------|-------|
| TensorFlow | 66.3% | 66.8% | 65.5% | 67.7% | 66.6% |
| PyTorch | 33.7% | 33.2% | 34.5% | 32.3% | 33.8% |

Parameterized Learning for Data Classification

The goal is to define a mathematical model which will *learn* from a *large* number of input data but will be defined by a small set of parameters *regardless* of the training size.

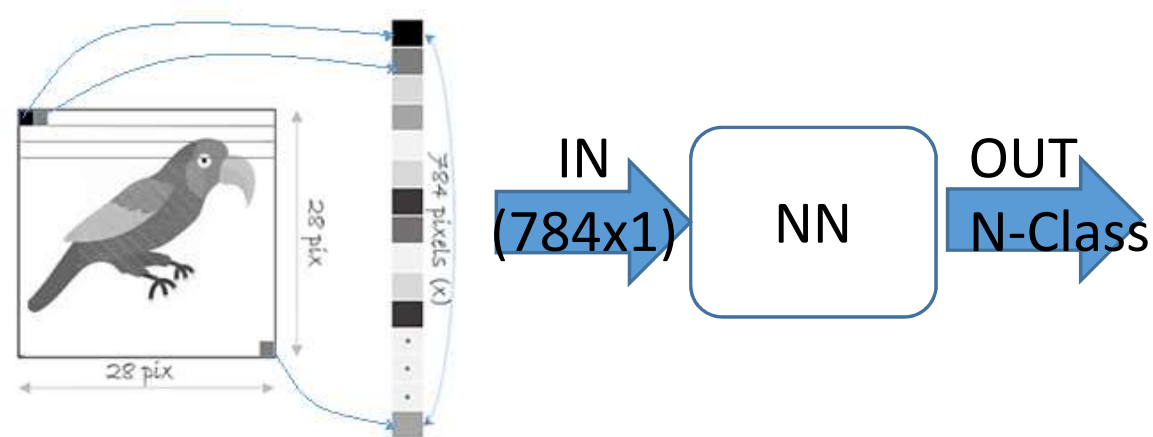
Once the data are *consumed* to compute the parameters the model will *predict* result on a new set of data without the need of the whole data history. No matter how big is the size of training dataset the model will not change size.

The process of defining the parameters of the model requires the following components:

- **Dataset:** where each *datapoint* is composed by the “raw” input data and the associated *class label* (supervised learning)
- **Scoring function:** the mathematical function that allow to map “raw” data in a corresponding class label
- **Loss function:** the mathematical function that measures how accurate is the predicted class (result) in comparison to the original correct class (ground-truth label). The higher is the accuracy of the prediction the lower is the loss function when using the training dataset. The goal of the “training phase” is to minimize the loss function (with specific algorithms and techniques) thereby increasing the classification accuracy
- **Weights and Biases:** collection of parameters which define the final model in its scoring function.

Image as a tensor

An image can be represented as a vector of pixels, i.e. a special case of tensor. In this example a 28x28x1 (1ch, grayscale) pixels are aligned in a 28x28=784 size vector

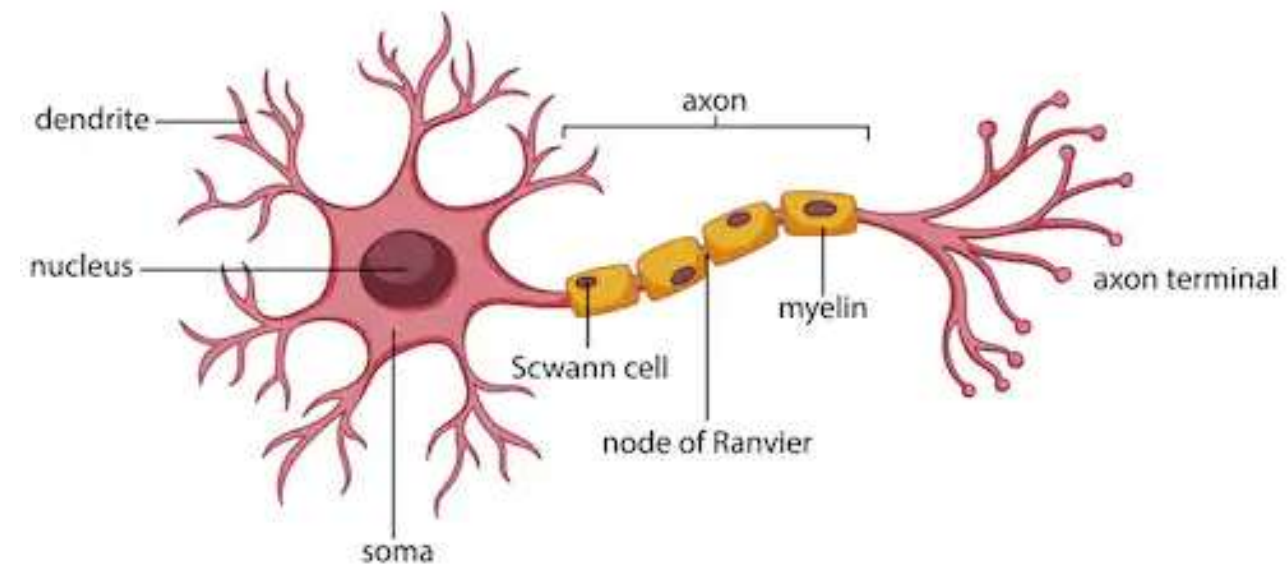
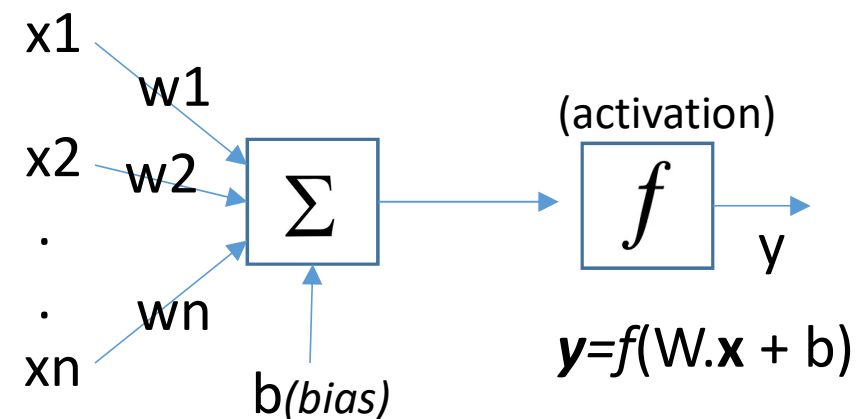


Neurons & Activation functions



Neurons and Activation Functions

Artificial Neural Networks have been *inspired* by biology and what we know about our brain functionality. ANN are modeled on the brain but are *not* a representation of the brain itself.



Rosenblatt: The Perceptron (1958)

[ref_docs\\[NN\]rosenblatt_perceptron_10.1.1.335.3398.pdf](ref_docs\[NN]rosenblatt_perceptron_10.1.1.335.3398.pdf)

Our brain is composed by 10^{12} (ten billions) neurons, each one connected to about 10^3 (ten thousands) other neurons. Each neuron does receive electrochemical inputs from other neurons. Only if the collection of all inputs is sufficient to activate the neuron it will transmit the signal to other neurons on its axon.

The *human* neuron does perform a *binary* operation: it will trigger a signal or not, there is no *signal modulation*.

An *artificial* neuron is modeling the same behavior: computes a sum of *weighted* inputs and has a *non linear* activation function which allows the neuron propagate the information.

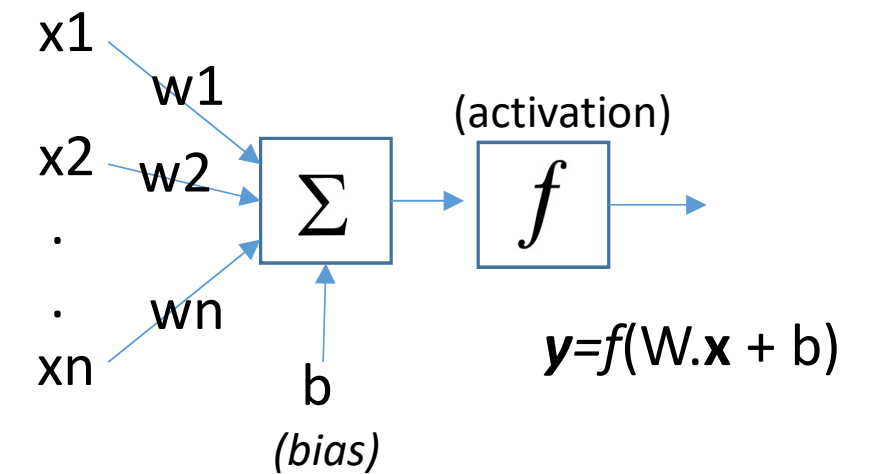
https://en.wikipedia.org/wiki/List_of_animals_by_number_of_neurons

<https://towardsdatascience.com/the-differences-between-artificial-and-biological-neural-networks-a8b46db828b7>

Artificial Neuron

The input vector \mathbf{x} (collection of all the stimulus) is multiplied by a vector of weights \mathbf{w} . The weighted sum is then passed into the *function activation* which will return a *binary* output. (*Perceptron*, Rosenblatt)

$$\text{output} = f(x_1 * w_1 + x_2 * w_2 + \dots + x_n * w_n)$$



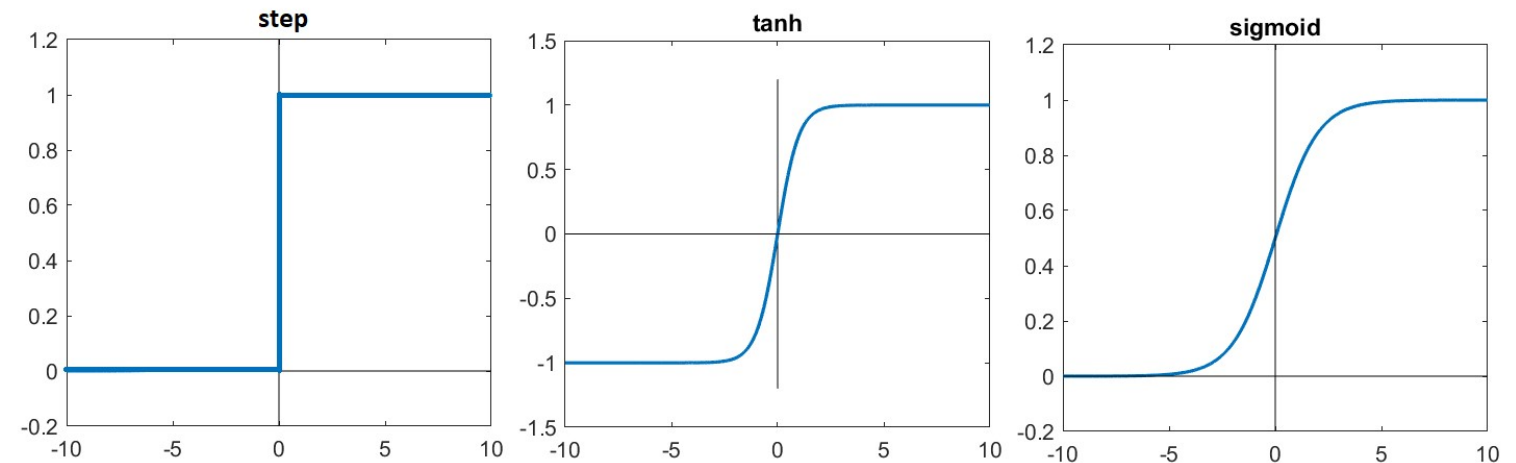
Activation Functions

Multiple activations functions have been proposed.

step: most intuitive but not differentiable

tanh: used until '90s $\tanh(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$

sigmoid: very important $s(z) = \frac{1}{1 + e^{-z}}$



Sigmoid characteristics:

- Continuous and differentiable everywhere
- Symmetric on the output axis
- Asymptotically approaches output values (0,1)

<http://www.cs.cmu.edu/~bhiksha/courses/deeplearning/Fall.2019/www/hwnotes/HW1p1.html>

Other variants have been introduced to simplify computation while maintaining the characteristics of the sigmoid function.

Rectified Linear Unit (ReLU) is part of the *ramp function* family. It has been proved to perform better than sigmoid and tanh, It is the most frequent activation function used in today's CNNs.

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

Leaky ReLU is a variant of ReLU which allows to compute the gradient also when the neuron is *not* firing

$$LeakyReLU(x) = \begin{cases} x & x \geq 0 \\ \alpha x & x < 0 \end{cases}$$

Exponential Linear Unit (ELU) was introduced in 2015 to improve the classification results when using ReLU: <https://arxiv.org/abs/1511.07289>

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

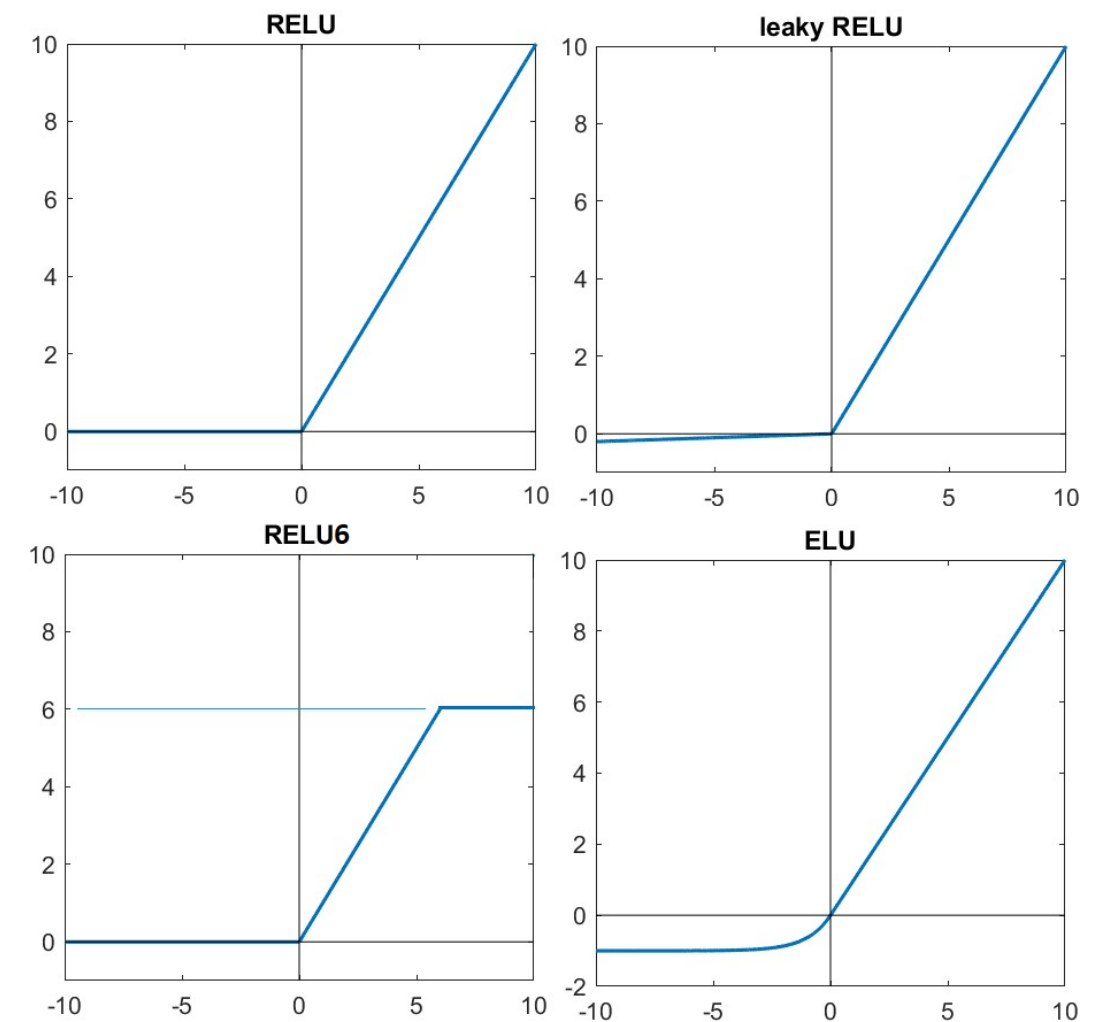
ReLU6 is a “capped” version of ReLU and was empirically tuned to better perform on sparse matrix. It has been used in recent CNN (MobileNet) with good results

https://www.tensorflow.org/api_docs/python/tf/nn/relu6

<https://arxiv.org/pdf/1803.08375.pdf>

<http://www.cs.utoronto.ca/~kriz/conv-cifar10-aug2010.pdf>

[https://en.wikipedia.org/wiki/Rectifier_\(neural_networks\)](https://en.wikipedia.org/wiki/Rectifier_(neural_networks))

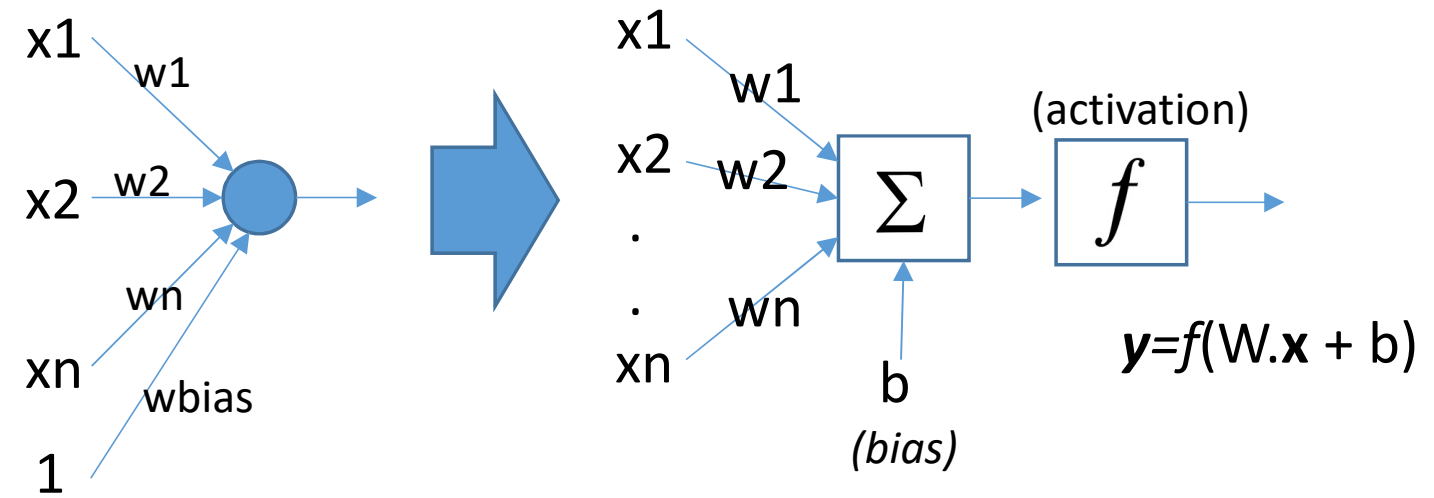


Perceptron and Delta Rule

Rosenblatt defined the perceptron as a system that learns from *vectors of features (x)* and *labeled examples (ground-truth)* mapping the input values into the corresponding output class labels.

The simplest architecture of a perceptron is composed by only *one* layer with *one* neuron.

The training of the perceptron is done by approximations based on all the input data points x_i and by computing a *delta* of the error between the prediction and the *ground-truth* label (expected result). The pseudo-algorithm is:



1) Initialize \mathbf{W} with small random values (uniform distribution)

2) While (convergence==False)

a) Loop over each feature vector \mathbf{x}_j and label \mathbf{d}_j in the whole training dataset D

b) Given feature x_j , compute the output $y_j = f(\mathbf{W}(t) * \mathbf{x}_j)$

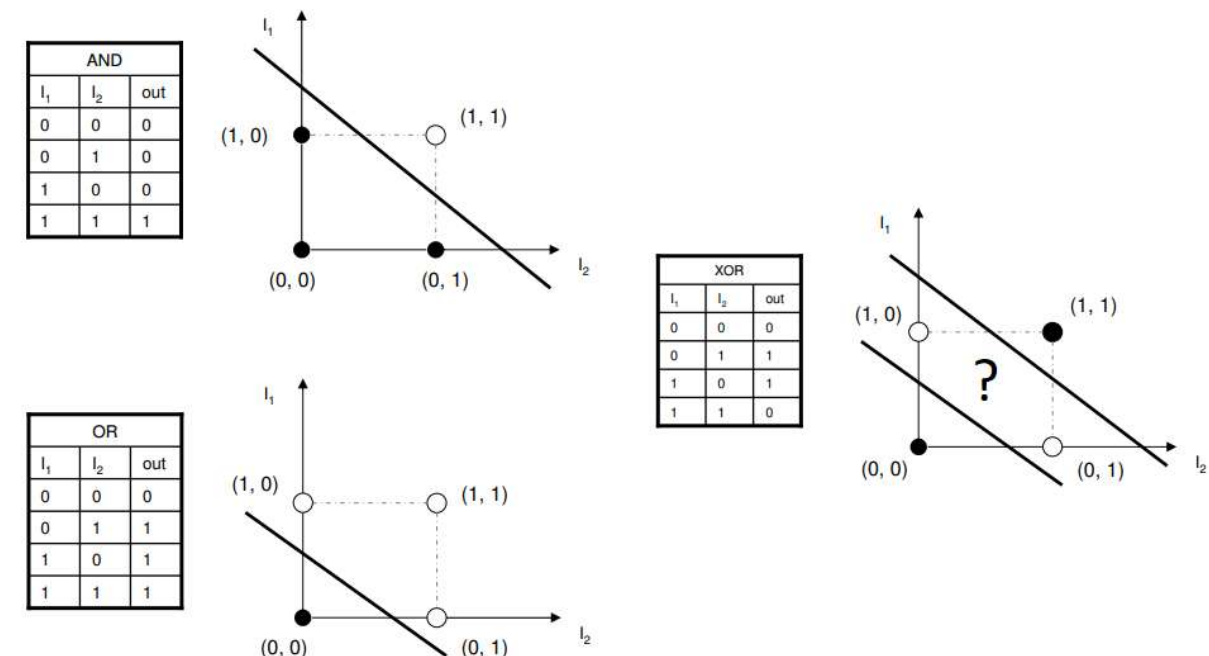
c) Update \mathbf{W} with the *delta rule*:

$$w_i(t+1) = w_i(t) + \alpha * (d_j - y_j) x_{j,i} \quad (i=0..n)$$

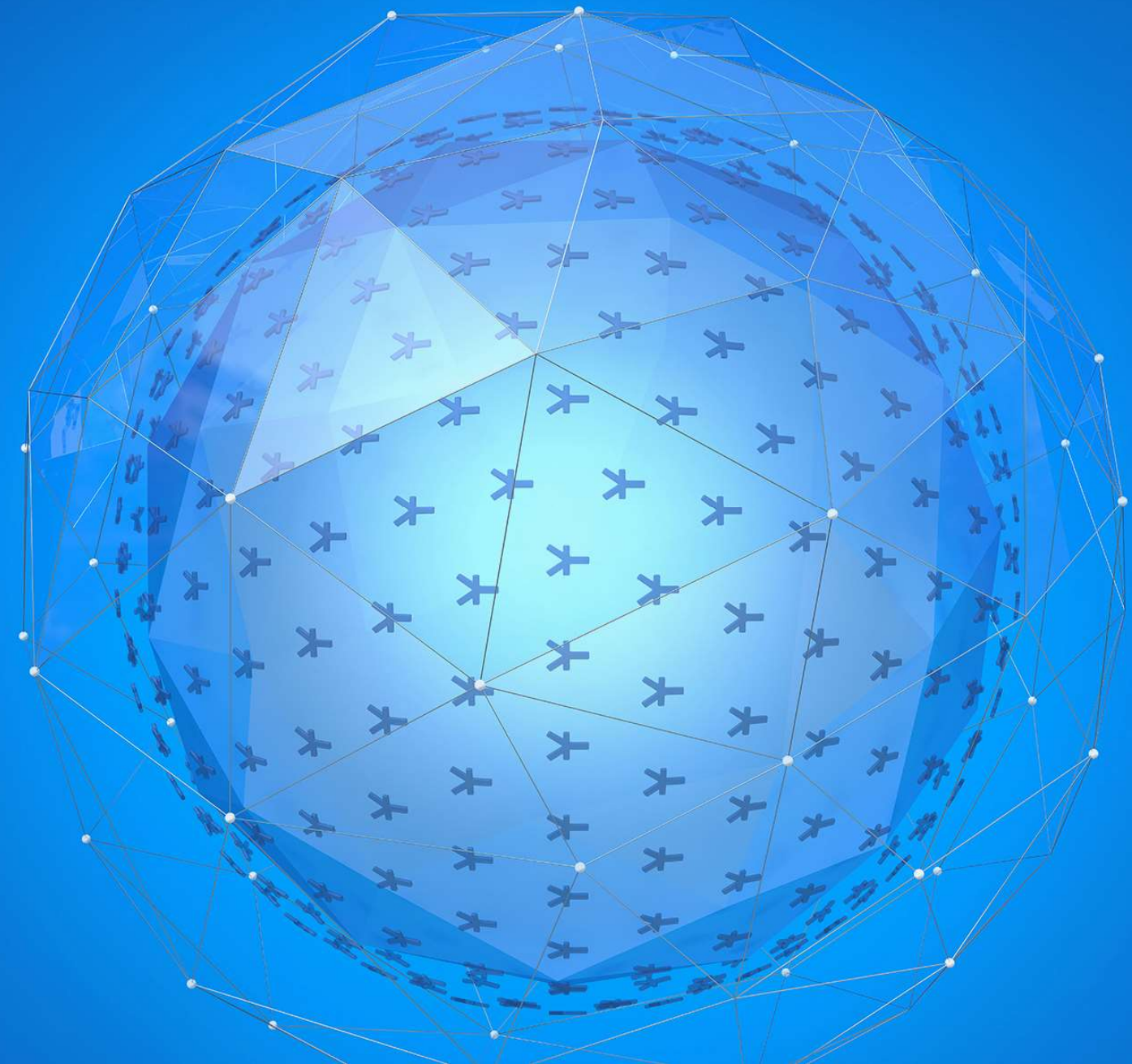
http://www.di.unito.it/~cancelli/retineu11_12/FNN.pdf

<http://www.cs.stir.ac.uk/courses/ITNP4B/lectures/kms/2-Perceptrons.pdf>

Perceptron XOR problem: cannot classify dataset that are nonlinearly separable.



Convolutions & Kernels



Regular Neural Nets don't scale well to full images. In CIFAR-10, images are 32x32x3 (32 wide, 32 high, 3 color channels), a single fully-connected neuron in a first hidden layer of a regular Neural Network would have $32 \times 32 \times 3 = 3072$ weights. *What if we have larger images?*

Convolutional Networks

CNN are a special type of NN which have been widely used in the last decade thanks to its performance in learning features and high accuracy on data classification.

CNN have peculiar architectures which are different from a FC network and uses different operators, in particular the *convolution* operator. CNN also have two characteristics which make them powerful:

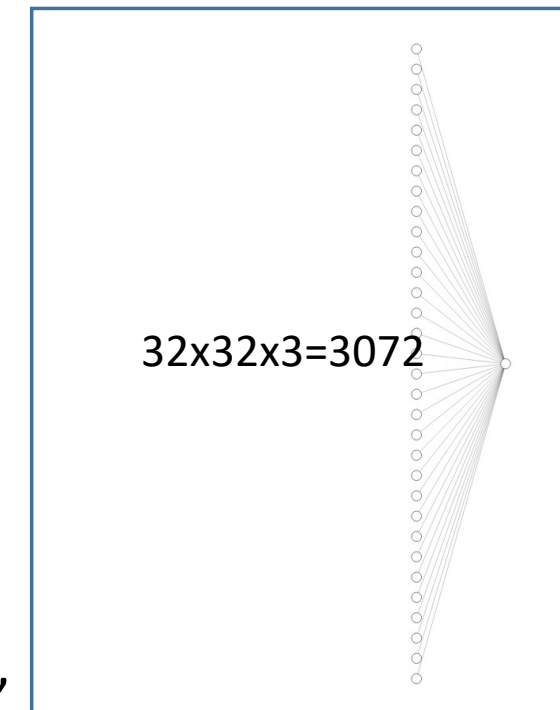
Local invariance: allows to identify a pattern (an object) inside the input signal (an image) regardless of where the pattern is located inside the signal itself.

Compositionality: each layer is building “knowledge” on top of the previous layers and the composition can be altered by specific operators. Features extracted from a high level analysis can be combined with features from a low level analysis (not a sequential architecture).

Convolution is the most important function used to build CNNs.

- Uses one or more filters (kernels) to extract features from the input data (images)
- Images and kernels are both matrix, convolution leverages the dot product.

<http://cs231n.github.io/convolutional-networks/>



Convolution as matrix multiplication (el.wise) and sum

Given an input signal (image, matrix) we have a smaller filter (kernel, matrix). The filter will covers the whole input signal and computes the element-wise matrix multiplication and sum. Each result of the kernel computation generates one value on the output feature map.

input

| | | | | | |
|---|---|---|---|---|---|
| 1 | 1 | 1 | 0 | 0 | 0 |
| 0 | 1 | 1 | 1 | 0 | 1 |
| 0 | 0 | 1 | 1 | 1 | 1 |
| 0 | 0 | 1 | 1 | 0 | 0 |
| 0 | 1 | 1 | 0 | 0 | 0 |
| 0 | 1 | 0 | 1 | 0 | 1 |

kernel

| | | |
|---|---|---|
| 1 | 0 | 1 |
| 0 | 1 | 0 |
| 1 | 0 | 1 |

Feature Map (FM)

| | | | | | |
|--|--|--|--|--|--|
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |

[Rachel Draelos](https://glassboxmedicine.com/2019/07/26/convolution-vs-cross-correlation/) <https://glassboxmedicine.com/2019/07/26/convolution-vs-cross-correlation/>



Convolution as matrix multiplication (el.wise) and sum

Given an input signal (image, matrix) we have a smaller filter (kernel, matrix). The filter will covers the whole input signal and computes the element-wise matrix multiplication and sum. Each result of the kernel computation generates one value on the output feature map.

input

| | | | | | |
|---|---|---|---|---|---|
| 1 | 0 | 1 | 0 | 0 | 0 |
| 0 | 1 | 0 | 1 | 0 | 1 |
| 0 | 0 | 1 | 1 | 1 | 1 |
| 0 | 0 | 1 | 1 | 0 | 0 |
| 0 | 1 | 1 | 0 | 0 | 0 |
| 0 | 1 | 0 | 1 | 0 | 1 |

$$FM(1,1) = 1/9*(1*1+0*1+1*1+0*0+1*1+0*1+1*0+0*0+1*1)=5/9$$

Feature Map (FM)

| | | | | | |
|--|-----|--|--|--|--|
| | | | | | |
| | 5/9 | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |

[Rachel Draelos](https://glassboxmedicine.com/2019/07/26/convolution-vs-cross-correlation/) <https://glassboxmedicine.com/2019/07/26/convolution-vs-cross-correlation/>



Convolution as matrix multiplication (el.wise) and sum

Given an input signal (image, matrix) we have a smaller filter (kernel, matrix). The filter will covers the whole input signal and computes the element-wise matrix multiplication and sum. Each result of the kernel computation generates one value on the output feature map.

input

| | | | | | |
|---|---|---|---|---|---|
| 1 | 1 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 | 0 | 1 |
| 0 | 0 | 0 | 1 | 1 | 1 |
| 0 | 0 | 1 | 1 | 0 | 0 |
| 0 | 1 | 1 | 0 | 0 | 0 |
| 0 | 1 | 0 | 1 | 0 | 1 |

$$FM(1,1) = 1/9*(1*1+0*1+1*1+0*0+1*1+0*1+1*0+0*0+1*1)=5/9$$

$$FM(1,2) = 1/9*(1*1+0*1+1*0+0*1+1*1+0*1+1*0+0*1+1*1)=4/9$$

Feature Map (FM)

| | | | | | |
|--|-----|-----|--|--|--|
| | | | | | |
| | 5/9 | 4/9 | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |

[Rachel Draelos](https://glassboxmedicine.com/2019/07/26/convolution-vs-cross-correlation/) <https://glassboxmedicine.com/2019/07/26/convolution-vs-cross-correlation/>



Convolution as matrix multiplication (el.wise) and sum

Given an input signal (image, matrix) we have a smaller filter (kernel, matrix). The filter will covers the whole input signal and computes the element-wise matrix multiplication and sum. Each result of the kernel computation generates one value on the output feature map.

input

| | | | | | |
|---|---|---|---|---|---|
| 1 | 1 | 1 | 0 | 1 | 0 |
| 0 | 1 | 1 | 1 | 0 | 1 |
| 0 | 0 | 1 | 1 | 1 | 1 |
| 0 | 0 | 1 | 1 | 0 | 0 |
| 0 | 1 | 1 | 0 | 0 | 0 |
| 0 | 1 | 0 | 1 | 0 | 1 |

$$FM(1,1) = 1/9*(1*1+0*1+1*1+0*0+1*1+0*1+1*0+0*0+1*1)=5/9$$

$$FM(1,2) = 1/9*(1*1+0*1+1*0+0*1+1*1+0*1+1*0+0*1+1*1)=4/9$$

$$FM(1,3) = 1/9*(1*1+0*0+1*0+0*1+1*1+0*0+1*1+0*1+1*1)=4/9$$

Feature Map (FM)

| | | | | | |
|--|-----|-----|-----|--|--|
| | | | | | |
| | 5/9 | 4/9 | 4/9 | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |

[Rachel Draelos](https://glassboxmedicine.com/2019/07/26/convolution-vs-cross-correlation/) <https://glassboxmedicine.com/2019/07/26/convolution-vs-cross-correlation/>



Convolution as matrix multiplication (el.wise) and sum

Given an input signal (image, matrix) we have a smaller filter (kernel, matrix). The filter will covers the whole input signal and computes the element-wise matrix multiplication and sum. Each result of the kernel computation generates one value on the output feature map.

input

| | | | | | |
|---|---|---|---|---|---|
| 1 | 1 | 1 | 0 | 0 | 0 |
| 0 | 1 | 1 | 1 | 0 | 1 |
| 0 | 0 | 1 | 1 | 1 | 1 |
| 0 | 0 | 1 | 1 | 0 | 0 |
| 0 | 1 | 1 | 0 | 0 | 0 |
| 0 | 1 | 0 | 1 | 0 | 1 |

$$FM(1,1) = 1/9*(1*1+0*1+1*1+0*0+1*1+0*1+1*0+0*0+1*1)=5/9$$

$$FM(1,2) = 1/9*(1*1+0*1+1*0+0*1+1*1+0*1+1*0+0*1+1*1)=4/9$$

$$FM(1,3) = 1/9*(1*1+0*0+1*0+0*1+1*1+0*0+1*1+0*1+1*1)=4/9$$

$$FM(1,3) = 1/9*(1*0+0*0+1*0+0*1+1*0+0*1+1*1+0*1+1*1)=3/9$$

Feature Map (FM)

| | | | | | |
|--|-----|-----|-----|-----|--|
| | | | | | |
| | 5/9 | 4/9 | 4/9 | 3/9 | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |

[Rachel Draelos](https://glassboxmedicine.com/2019/07/26/convolution-vs-cross-correlation/) <https://glassboxmedicine.com/2019/07/26/convolution-vs-cross-correlation/>



Convolution as matrix multiplication (el.wise) and sum

Given an input signal (image, matrix) we have a smaller filter (kernel, matrix). The filter will covers the whole input signal and computes the element-wise matrix multiplication and sum. Each result of the kernel computation generates one value on the output feature map.

input

| | | | | | |
|---|---|---|---|---|---|
| 1 | 1 | 1 | 0 | 0 | 0 |
| 0 | 1 | 1 | 1 | 0 | 1 |
| 0 | 0 | 1 | 1 | 1 | 1 |
| 0 | 0 | 1 | 1 | 0 | 0 |
| 0 | 1 | 1 | 0 | 0 | 0 |
| 0 | 1 | 0 | 1 | 0 | 1 |

$$FM(1,1) = 1/9*(1*1+0*1+1*1+0*0+1*1+0*1+1*0+0*0+1*1)=5/9$$

$$FM(1,2) = 1/9*(1*1+0*1+1*0+0*1+1*1+0*1+1*0+0*1+1*1)=4/9$$

$$FM(1,3) = 1/9*(1*1+0*0+1*0+0*1+1*1+0*0+1*1+0*1+1*1)=4/9$$

$$FM(1,3) = 1/9*(1*0+0*0+1*0+0*1+1*0+0*1+1*1+0*1+1*1)=3/9$$

$$FM(2,1) = 1/9*(1*0+0*1+1*1+0*0+1*0+0*1+1*0+0*0+1*1)=1/9$$

Feature Map (FM)

| | | | | | |
|--|-----|-----|-----|-----|--|
| | | | | | |
| | 5/9 | 4/9 | 4/9 | 3/9 | |
| | 2/9 | | | | |
| | | | | | |
| | | | | | |
| | | | | | |

[Rachel Draelos](https://glassboxmedicine.com/2019/07/26/convolution-vs-cross-correlation/) <https://glassboxmedicine.com/2019/07/26/convolution-vs-cross-correlation/>



Convolution as matrix multiplication (el.wise) and sum

Given an input signal (image, matrix) we have a smaller filter (kernel, matrix). The filter will covers the whole input signal and computes the element-wise matrix multiplication and sum. Each result of the kernel computation generates one value on the output feature map.

input

| | | | | | |
|---|---|---|---|---|---|
| 1 | 1 | 1 | 0 | 0 | 0 |
| 0 | 1 | 1 | 1 | 0 | 1 |
| 0 | 0 | 1 | 1 | 1 | 1 |
| 0 | 1 | 1 | 1 | 0 | 0 |
| 0 | 1 | 1 | 0 | 0 | 0 |
| 0 | 1 | 0 | 1 | 0 | 1 |

$$FM(1,1) = 1/9*(1*1+0*1+1*1+0*0+1*1+0*1+1*0+0*0+1*1)=5/9$$

$$FM(1,2) = 1/9*(1*1+0*1+1*0+0*1+1*1+0*1+1*0+0*1+1*1)=4/9$$

$$FM(1,3) = 1/9*(1*1+0*0+1*0+0*1+1*1+0*0+1*1+0*1+1*1)=4/9$$

$$FM(1,3) = 1/9*(1*0+0*0+1*0+0*1+1*0+0*1+1*1+0*1+1*1)=3/9$$

$$FM(2,1) = 1/9*(1*0+0*1+1*1+0*0+1*0+0*1+1*0+0*0+1*1)=1/9$$

(...)

Feature Map (FM)

| | | | | | |
|--|-----|-----|-----|-----|--|
| | | | | | |
| | 5/9 | 4/9 | 4/9 | 3/9 | |
| | 2/9 | | | | |
| | | | | | |
| | | | | | |
| | | | | | |

[Rachel Draelos](https://glassboxmedicine.com/2019/07/26/convolution-vs-cross-correlation/) <https://glassboxmedicine.com/2019/07/26/convolution-vs-cross-correlation/>



Convolution as matrix multiplication (el.wise) and sum

Given an input signal (image, matrix) we have a smaller filter (kernel, matrix). The filter will covers the whole input signal and computes the element-wise matrix multiplication and sum. Each result of the kernel computation generates one value on the output feature map.

input

| | | | | | |
|---|---|---|---|---|---|
| 1 | 1 | 1 | 0 | 0 | 0 |
| 0 | 1 | 1 | 0 | 0 | 1 |
| 0 | 0 | 1 | 1 | 0 | 1 |
| 0 | 0 | 1 | 0 | 0 | 0 |
| 0 | 1 | 1 | 0 | 0 | 0 |
| 0 | 1 | 0 | 1 | 0 | 1 |

$$FM(1,1) = 1/9*(1*1+0*1+1*1+0*0+1*1+0*1+1*0+0*0+1*1)=5/9$$

$$FM(1,2) = 1/9*(1*1+0*1+1*0+0*1+1*1+0*1+1*0+0*1+1*1)=4/9$$

$$FM(1,3) = 1/9*(1*1+0*0+1*0+0*1+1*1+0*0+1*1+0*1+1*1)=4/9$$

$$FM(1,3) = 1/9*(1*0+0*0+1*0+0*1+1*0+0*1+1*1+0*1+1*1)=3/9$$

$$FM(2,1) = 1/9*(1*0+0*1+1*1+0*0+1*0+0*1+1*0+0*0+1*1)=1/9$$

(...)

Feature Map (FM)

| | | | | | |
|--|-----|-----|-----|-----|--|
| | | | | | |
| | 5/9 | 4/9 | 4/9 | 3/9 | |
| | 2/9 | | | | |
| | | | | | |
| | | | | | |
| | | | | | |

[Rachel Draelos](https://glassboxmedicine.com/2019/07/26/convolution-vs-cross-correlation/) <https://glassboxmedicine.com/2019/07/26/convolution-vs-cross-correlation/>



Convolution as matrix multiplication (el.wise) and sum

Given an input signal (image, matrix) we have a smaller filter (kernel, matrix). The filter will covers the whole input signal and computes the element-wise matrix multiplication and sum. Each result of the kernel computation generates one value on the output feature map.

input

| | | | | | |
|---|---|---|---|---|---|
| 1 | 1 | 1 | 0 | 0 | 0 |
| 0 | 1 | 1 | 1 | 0 | 1 |
| 0 | 0 | 1 | 1 | 1 | 1 |
| 0 | 0 | 1 | 1 | 0 | 0 |
| 0 | 1 | 1 | 0 | 0 | 0 |
| 0 | 1 | 0 | 1 | 0 | 1 |

$$FM(1,1) = 1/9*(1*1+0*1+1*1+0*0+1*1+0*1+1*0+0*0+1*1)=5/9$$

$$FM(1,2) = 1/9*(1*1+0*1+1*0+0*1+1*1+0*1+1*0+0*1+1*1)=4/9$$

$$FM(1,3) = 1/9*(1*1+0*0+1*0+0*1+1*1+0*0+1*1+0*1+1*1)=4/9$$

$$FM(1,3) = 1/9*(1*0+0*0+1*0+0*1+1*0+0*1+1*1+0*1+1*1)=3/9$$

$$FM(2,1) = 1/9*(1*0+0*1+1*1+0*0+1*0+0*1+1*0+0*0+1*1)=1/9$$

(...)

Feature Map (FM)

| | | | | | |
|--|-----|-----|-----|-----|--|
| | | | | | |
| | 5/9 | 4/9 | 4/9 | 3/9 | |
| | 2/9 | | | | |
| | | | | | |
| | | | | | |
| | | | | | |

[Rachel Draelos](https://glassboxmedicine.com/2019/07/26/convolution-vs-cross-correlation/) <https://glassboxmedicine.com/2019/07/26/convolution-vs-cross-correlation/>



Convolution as matrix multiplication (el.wise) and sum

Given an input signal (image, matrix) we have a smaller filter (kernel, matrix). The filter will covers the whole input signal and computes the element-wise matrix multiplication and sum. Each result of the kernel computation generates one value on the output feature map.

input

| | | | | | |
|---|---|---|---|---|---|
| 1 | 1 | 1 | 0 | 0 | 0 |
| 0 | 1 | 1 | 1 | 0 | 1 |
| 0 | 0 | 1 | 1 | 1 | 1 |
| 0 | 0 | 1 | 1 | 0 | 0 |
| 0 | 1 | 1 | 0 | 0 | 0 |
| 0 | 1 | 0 | 1 | 0 | 1 |

$FM(1,1) = 1/9*(1*1+0*1+1*1+0*0+1*1+0*1+1*0+0*0+1*1)=5/9$
 $FM(1,2) = 1/9*(1*1+0*1+1*0+0*1+1*1+0*1+1*0+0*1+1*1)=4/9$
 $FM(1,3) = 1/9*(1*1+0*0+1*0+0*1+1*1+0*0+1*1+0*1+1*1)=4/9$
 $FM(1,3) = 1/9*(1*0+0*0+1*0+0*1+1*0+0*1+1*1+0*1+1*1)=3/9$
 $FM(2,1) = 1/9*(1*0+0*1+1*1+0*0+1*0+0*1+1*0+0*0+1*1)=1/9$
 (...)

Feature Map (FM)

| | | | | | |
|--|-----|-----|-----|-----|--|
| | | | | | |
| | 5/9 | 4/9 | 4/9 | 3/9 | |
| | 2/9 | | | | |
| | | | | | |
| | | | | | |
| | | | | | |

with signal S and kernel K : $(S * K)(i, j) = \sum_m \sum_n S(i + m, j + n)K(m, n)$

[Rachel Draelos](https://glassboxmedicine.com/2019/07/26/convolution-vs-cross-correlation/) <https://glassboxmedicine.com/2019/07/26/convolution-vs-cross-correlation/>



Convolution as matrix multiplication (el.wise) and sum

Given an input signal (image, matrix) we have a smaller filter (kernel, matrix). The filter will covers the whole input signal and computes the element-wise matrix multiplication and sum. Each result of the kernel computation generates one value on the output feature map.

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 1 | 1 | 1 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 1 | 1 | 0 | 1 | 0 |
| 0 | 0 | 0 | 1 | 1 | 1 | 1 | 0 |
| 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 |
| 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 | 1 | 0 | 1 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

Depending on kernel size we need to apply padding (normally zero-filled) to maintain the same size for input and feature map.

Feature Map (FM)

| | | | | | |
|--|-----|-----|-----|-----|--|
| | | | | | |
| | 5/9 | 4/9 | 4/9 | 3/9 | |
| | 2/9 | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |

[Rachel Draelos](https://glassboxmedicine.com/2019/07/26/convolution-vs-cross-correlation/) <https://glassboxmedicine.com/2019/07/26/convolution-vs-cross-correlation/>

Convolution as matrix multiplication (el.wise) and sum

Given an input signal (image, matrix) we have a smaller filter (kernel, matrix). The filter will covers the whole input signal and computes the element-wise matrix multiplication and sum. Each result of the kernel computation generates one value on the output feature map.

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 1 | 0 | 1 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 1 | 1 | 0 | 1 | 0 |
| 0 | 0 | 0 | 1 | 1 | 1 | 1 | 0 |
| 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 |
| 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 | 1 | 0 | 1 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

Depending on kernel size we need to apply padding (normally zero-filled) to maintain the same size for input and feature map.

Feature Map (FM)

| | | | | | | | |
|--|-----|-----|-----|-----|--|--|--|
| | | | | | | | |
| | 5/9 | 4/9 | 4/9 | 3/9 | | | |
| | 2/9 | | | | | | |
| | | | | | | | |
| | | | | | | | |
| | | | | | | | |
| | | | | | | | |

[Rachel Draelos](https://glassboxmedicine.com/2019/07/26/convolution-vs-cross-correlation/) <https://glassboxmedicine.com/2019/07/26/convolution-vs-cross-correlation/>

Convolution as matrix multiplication (el.wise) and sum

Given an input signal (image, matrix) we have a smaller filter (kernel, matrix). The filter will covers the whole input signal and computes the element-wise matrix multiplication and sum. Each result of the kernel computation generates one value on the output feature map.

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 1 | 1 | 0 | 1 | 0 |
| 0 | 0 | 0 | 1 | 1 | 1 | 1 | 0 |
| 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 |
| 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 | 1 | 0 | 1 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

Depending on kernel size we need to apply padding (normally zero-filled) to maintain the same size for input and feature map.

Feature Map (FM)

| | | | | | |
|--|-----|-----|-----|-----|--|
| | | | | | |
| | 5/9 | 4/9 | 4/9 | 3/9 | |
| | 2/9 | | | | |
| | | | | | |
| | | | | | |
| | | | | | |

[Rachel Draelos](https://glassboxmedicine.com/2019/07/26/convolution-vs-cross-correlation/) <https://glassboxmedicine.com/2019/07/26/convolution-vs-cross-correlation/>

Convolution as matrix multiplication (el.wise) and sum

Given an input signal (image, matrix) we have a smaller filter (kernel, matrix). The filter will covers the whole input signal and computes the element-wise matrix multiplication and sum. Each result of the kernel computation generates one value on the output feature map.

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 1 | 0 | 1 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 | 1 | 0 | 1 | 0 |
| 0 | 0 | 0 | 1 | 1 | 1 | 1 | 0 |
| 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 |
| 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 | 1 | 0 | 1 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

Depending on kernel size we need to apply padding (normally zero-filled) to maintain the same size for input and feature map.

Feature Map (FM)

| | | | | | |
|--|-----|-----|-----|-----|--|
| | | | | | |
| | 5/9 | 4/9 | 4/9 | 3/9 | |
| | 2/9 | | | | |
| | | | | | |
| | | | | | |
| | | | | | |

[Rachel Draelos](https://glassboxmedicine.com/2019/07/26/convolution-vs-cross-correlation/) <https://glassboxmedicine.com/2019/07/26/convolution-vs-cross-correlation/>

Convolution as matrix multiplication (el.wise) and sum

Given an input signal (image, matrix) we have a smaller filter (kernel, matrix). The filter will covers the whole input signal and computes the element-wise matrix multiplication and sum. Each result of the kernel computation generates one value on the output feature map.

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 1 | 1 | 1 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 1 | 1 | 0 | 1 | 0 |
| 0 | 0 | 0 | 1 | 1 | 1 | 1 | 0 |
| 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 |
| 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 | 1 | 0 | 1 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

Depending on kernel size we need to apply padding (normally zero-filled) to maintain the same size for input and feature map.

Feature Map (FM)

| | | | | | |
|--|-----|-----|-----|-----|--|
| | | | | | |
| | 5/9 | 4/9 | 4/9 | 3/9 | |
| | 2/9 | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |

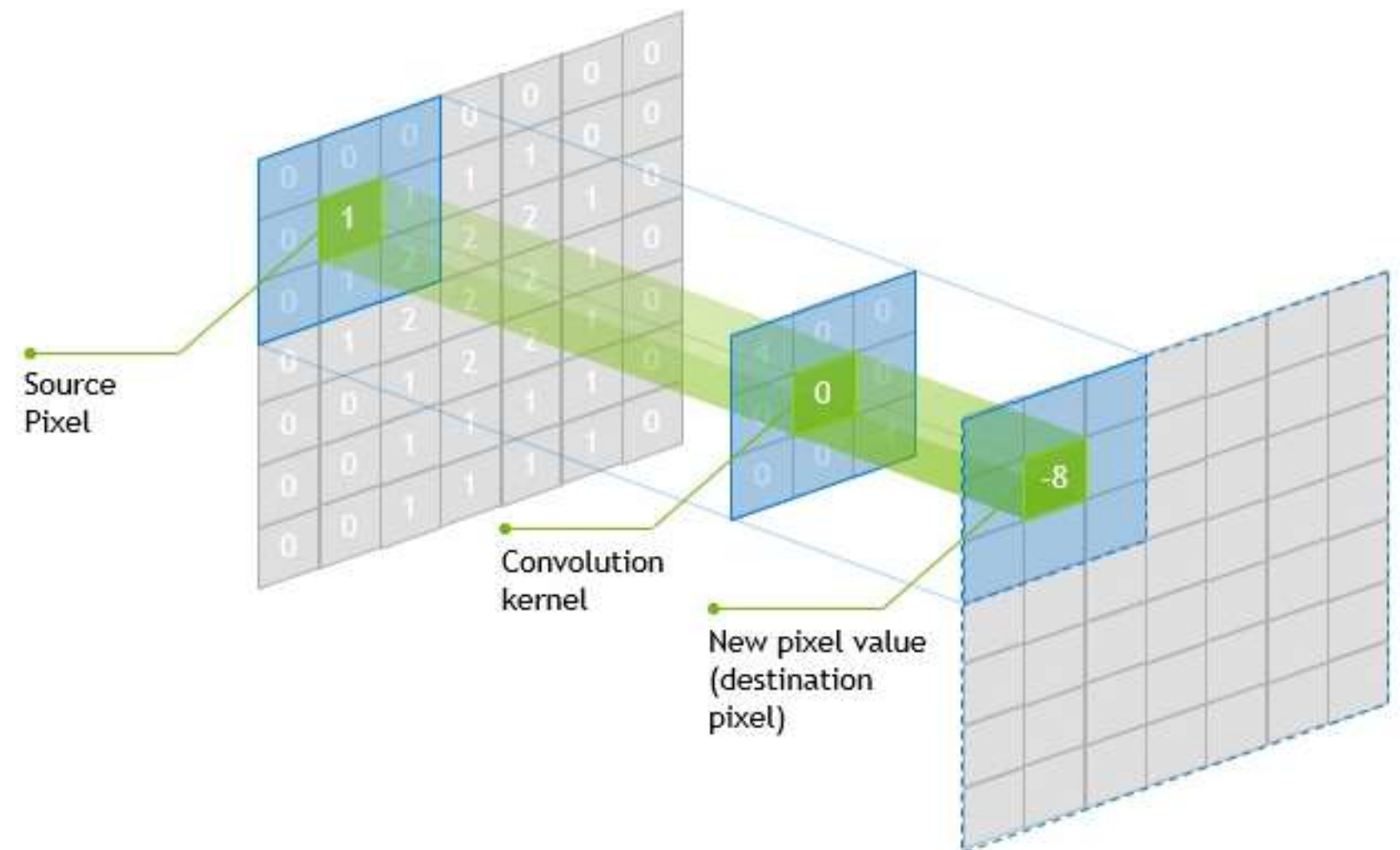
[Rachel Draelos](https://glassboxmedicine.com/2019/07/26/convolution-vs-cross-correlation/) <https://glassboxmedicine.com/2019/07/26/convolution-vs-cross-correlation/>



Convolution as matrix multiplication (el.wise) and sum

Given an input signal (image, matrix) we have a smaller filter (kernel, matrix). The filter will covers the whole input signal and computes the element-wise matrix multiplication and sum. Each result of the kernel computation generates one value on the output feature map.

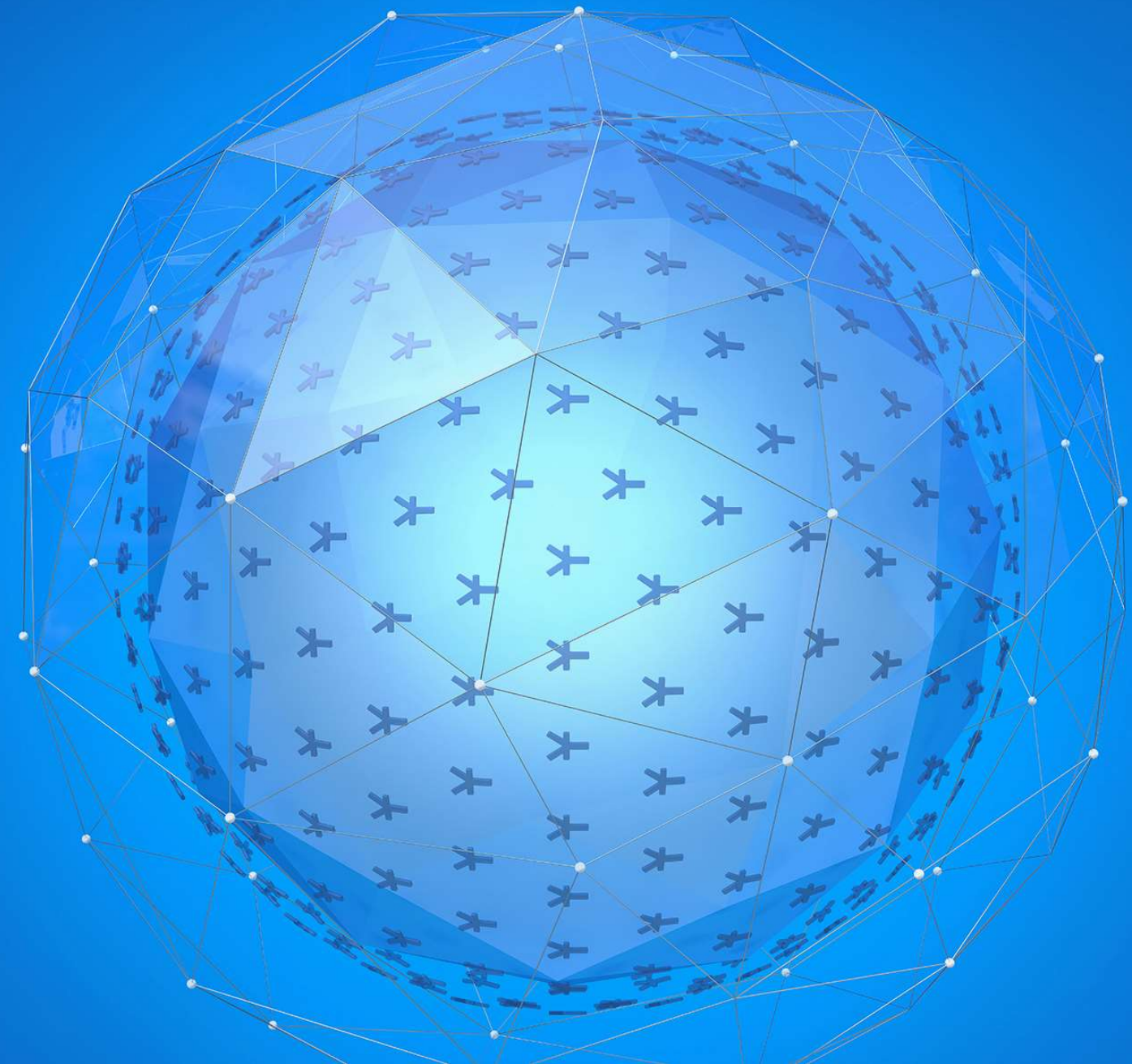
Convolving a smaller filter with a larger signal is related to the concept of **Receptive Field** of a neuron. The amount of data that are used to compute a specific result in the feature map is a subset of the entire input datapoint.



<https://blogs.nvidia.com/blog/2018/09/05/whats-the-difference-between-a-cnn-and-an-rnn/>

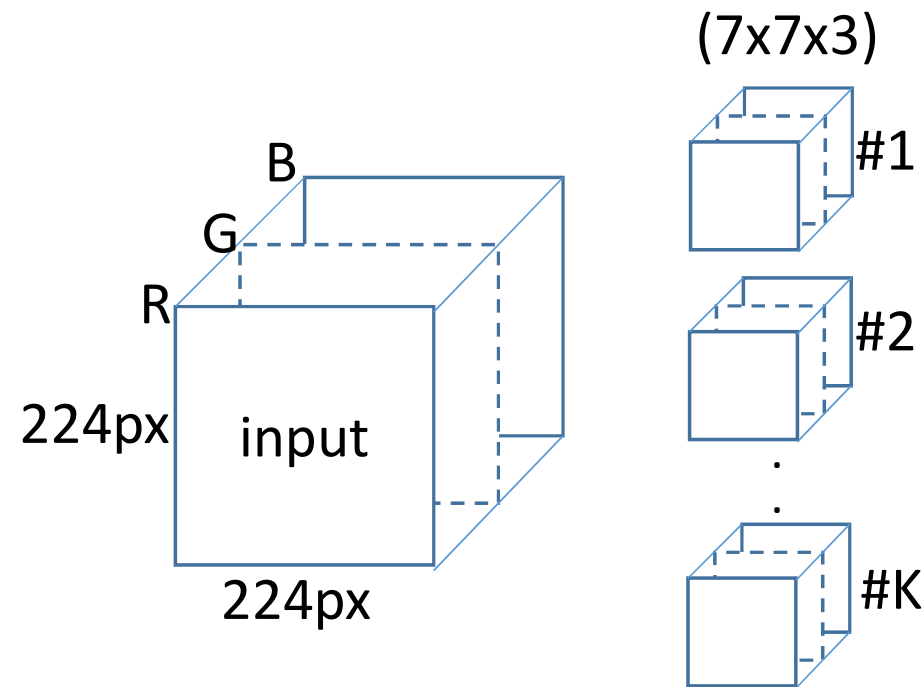
<https://medium.com/mlreview/a-guide-to-receptive-field-arithmetic-for-convolutional-neural-networks-e0f514068807>

CNN building blocks



CNN building blocks: CONV (convolutional) layer

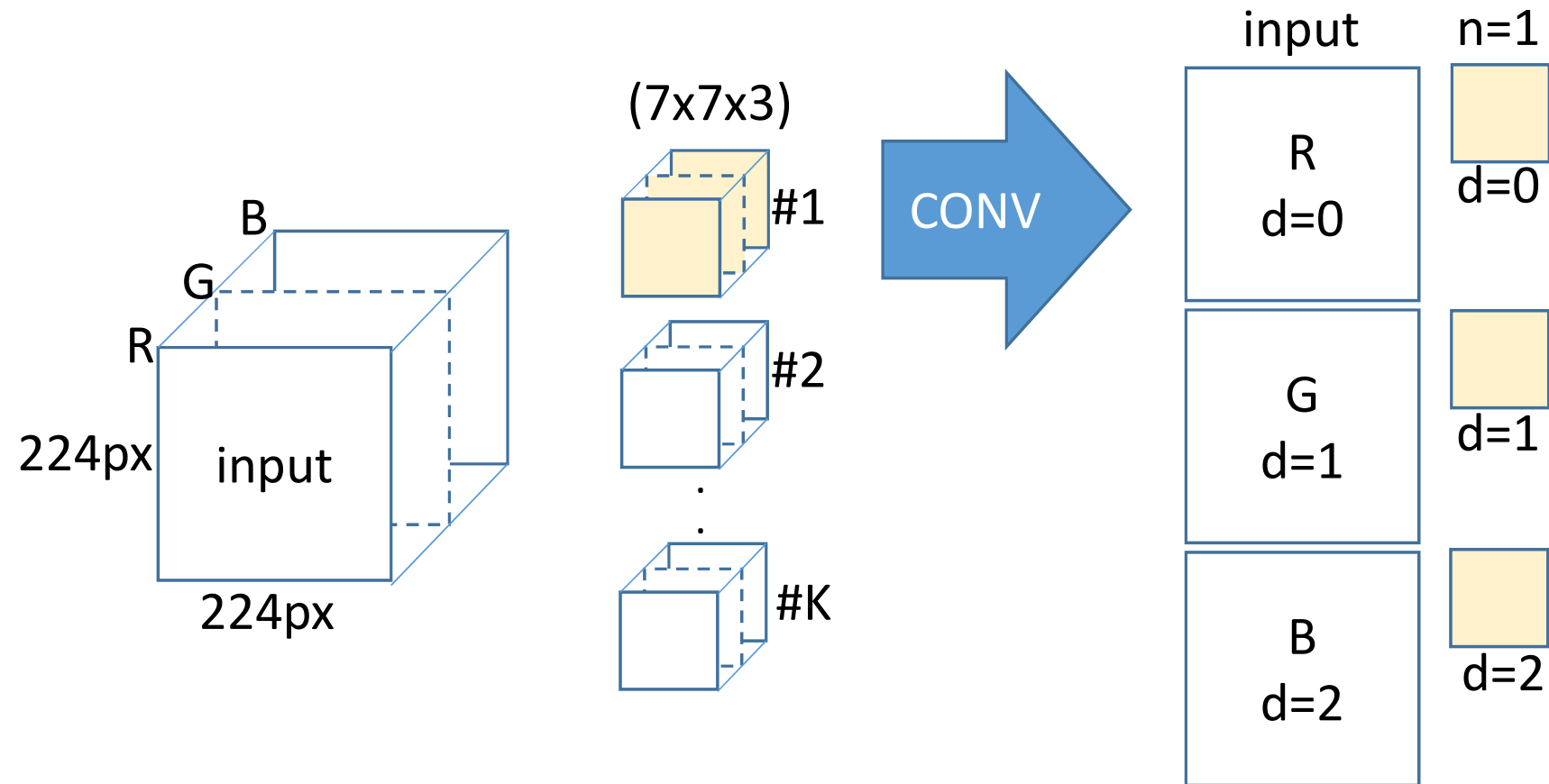
The convolutional layer is the core building block for deep learning. It consists of a set of K kernels (learned during training) with specific width, height, depth, stride. Usually small in size they extend for the whole input volume (channels num)



Kernels have the same depth of the input data. Usually we describe input and kernels in terms of *Volume* size. Each kernel of the CONV layer does generate a 2-dim *activation map*. All the activations maps (K , one for each kernel) are stacked into a volume of $M \times N \times K$ (M, N depends on kernel *stride* & *padding*)

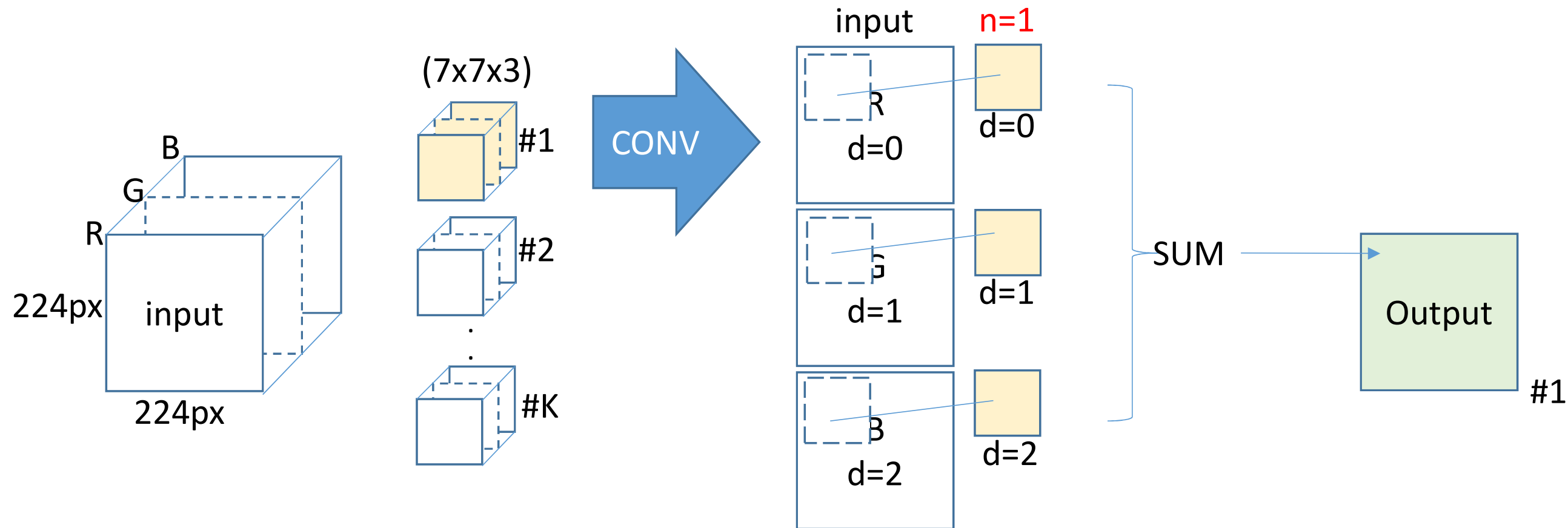
CNN building blocks: CONV (convolutional) layer

The convolutional layer is the core building block for deep learning. It consists of a set of K kernels (learned during training) with specific width, height, depth, stride. Usually small in size they extend for the whole input volume (channels num)



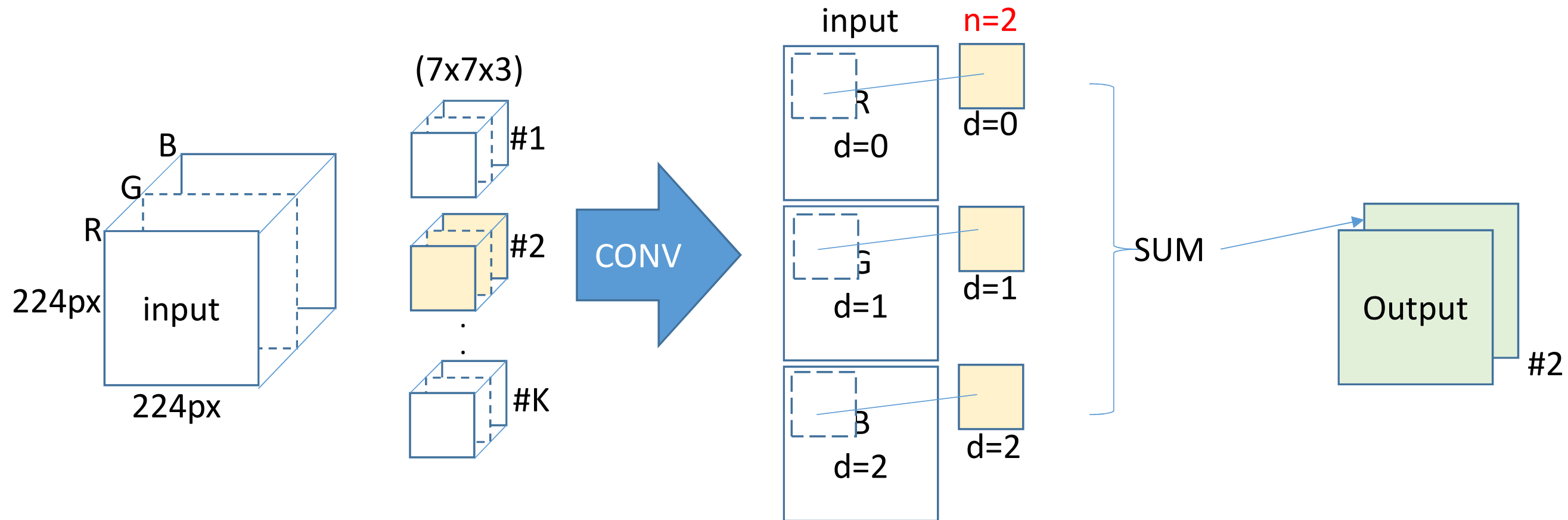
CNN building blocks: CONV (convolutional) layer

The convolutional layer is the core building block for deep learning. It consists of a set of K kernels (learned during training) with specific width, height, depth, stride. Usually small in size they extend for the whole input volume (channels num)



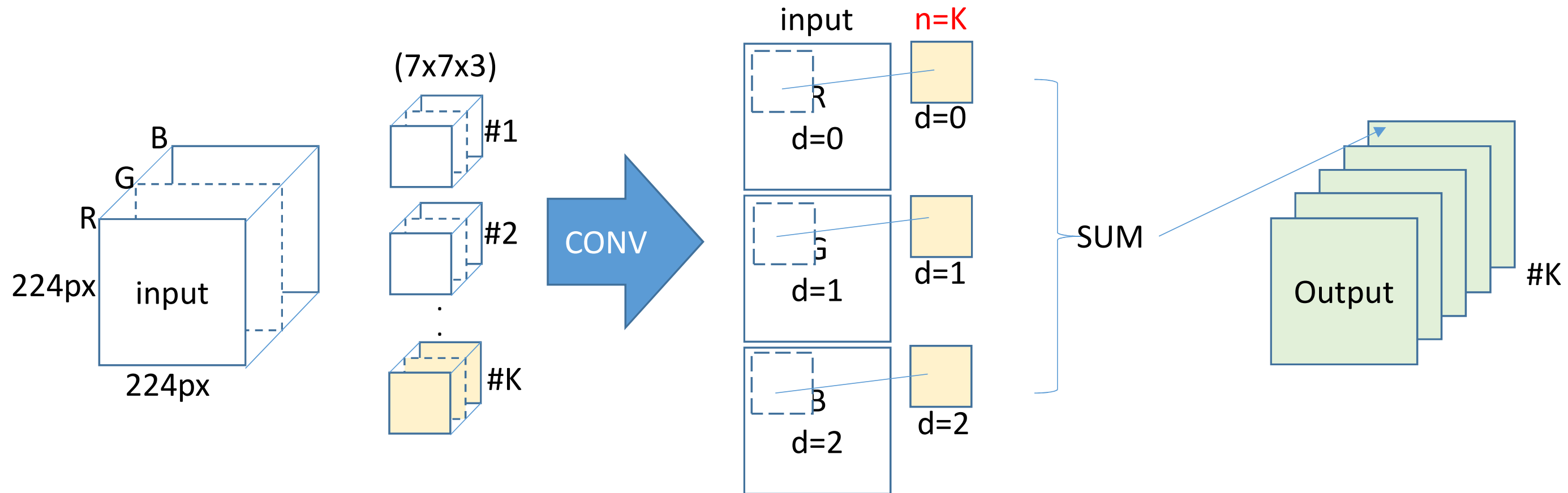
CNN building blocks: CONV (convolutional) layer

The convolutional layer is the core building block for deep learning. It consists of a set of K kernels (learned during training) with specific width, height, depth, stride. Usually small in size they extend for the whole input volume (channels num)



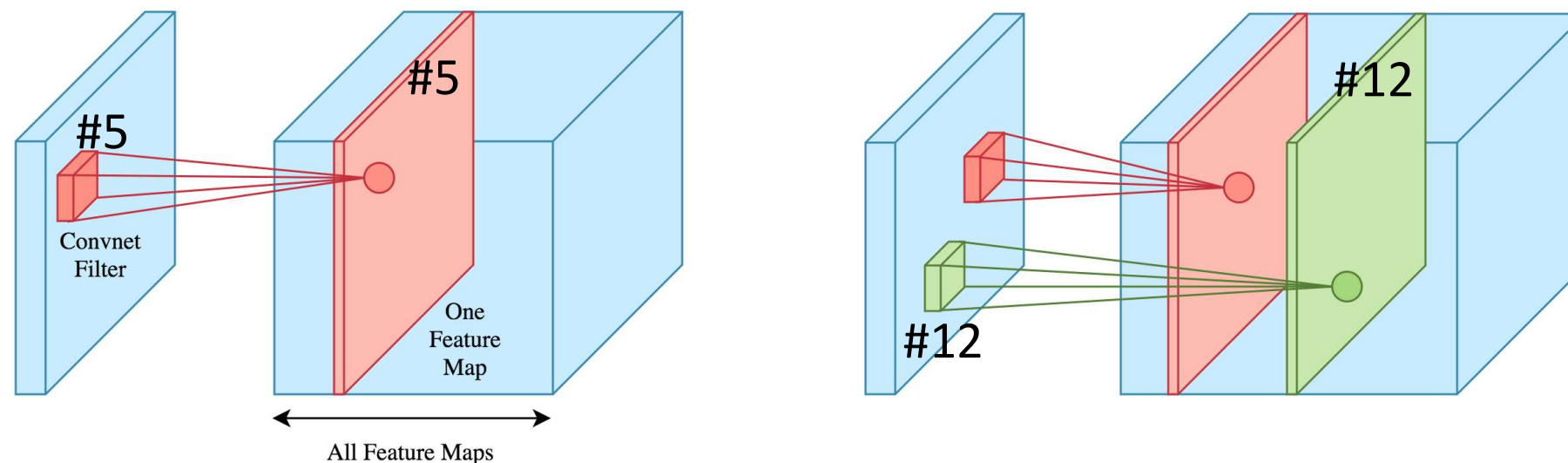
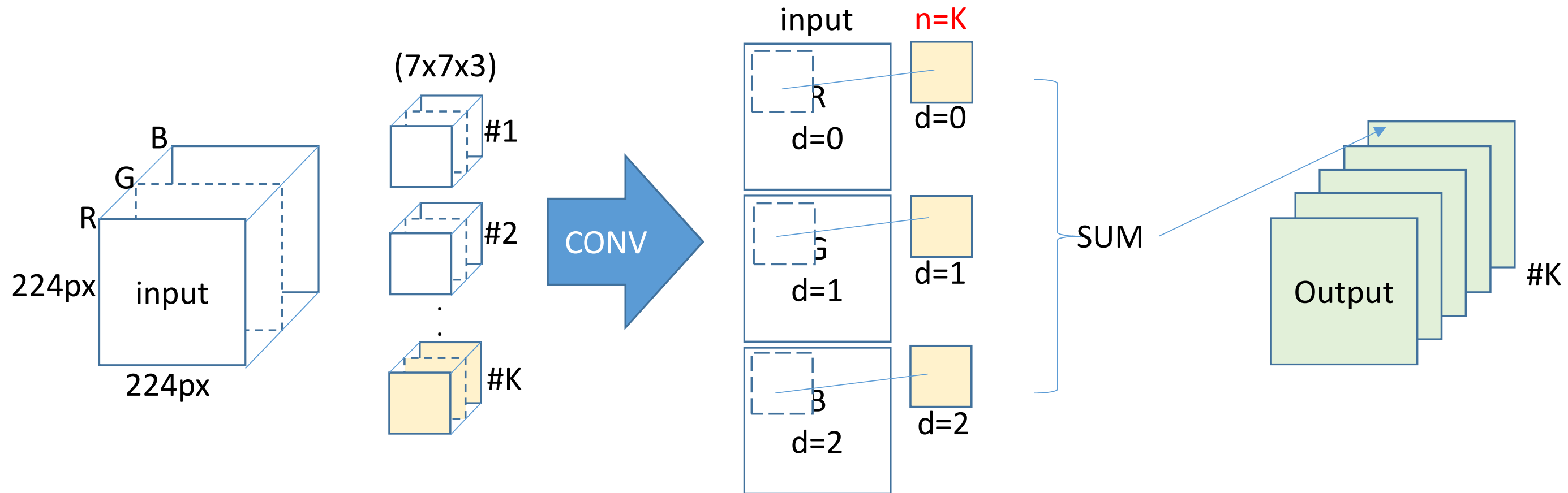
CNN building blocks: CONV (convolutional) layer

The convolutional layer is the core building block for deep learning. It consists of a set of K kernels (learned during training) with specific width, height, depth, stride. Usually small in size they extend for the whole input volume (channels num)



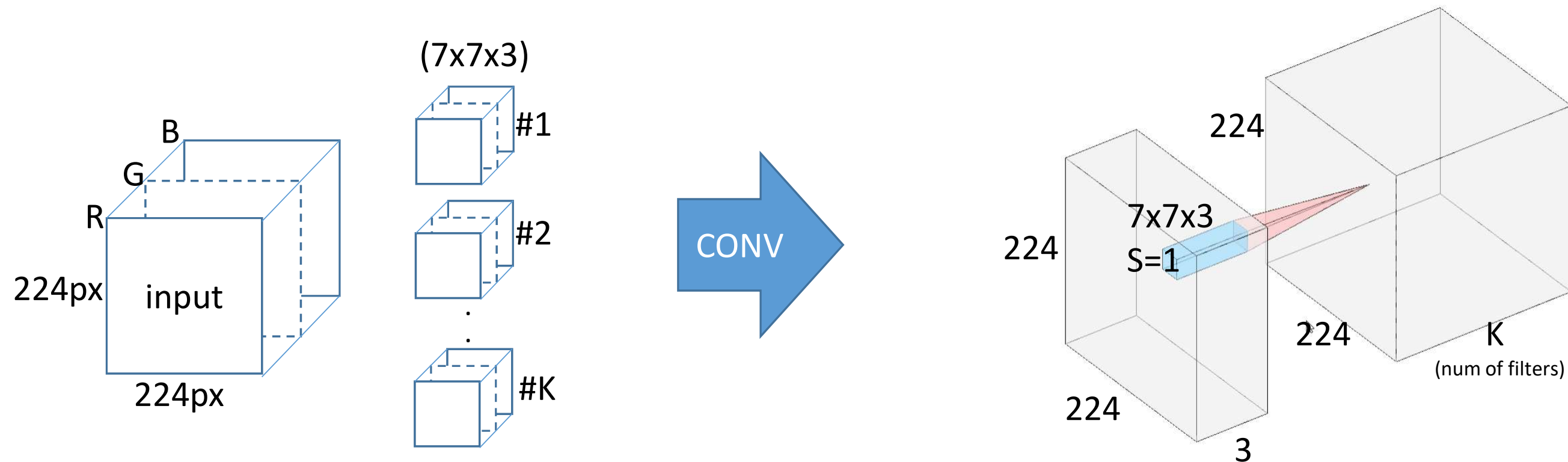
CNN building blocks: CONV (convolutional) layer

The convolutional layer is the core building block for deep learning. It consists of a set of K kernels (learned during training) with specific width, height, depth, stride. Usually small in size they extend for the whole input volume (channels num)



CNN building blocks: CONV (convolutional) layer

The convolutional layer is the core building block for deep learning. It consists of a set of K kernels (learned during training) with specific width, height, depth, stride. Usually small in size they extend for the whole input volume (channels num)



IN: volume of size $W1 \times H1 \times D1$

Hyperparameters:

1. Number of filters K ,
2. their spatial extent F ,
3. the stride S ,
4. the amount of zero padding P

OUT: volume of size $W2 \times H2 \times D2$

$$W2 = (W1 - F + 2P) / S + 1 = (224 - 7 + 2 * 3) / 1 + 1$$

$$H2 = (H1 - F + 2P) / S + 1 = (224 - 7 + 2 * 3) / 1 + 1$$

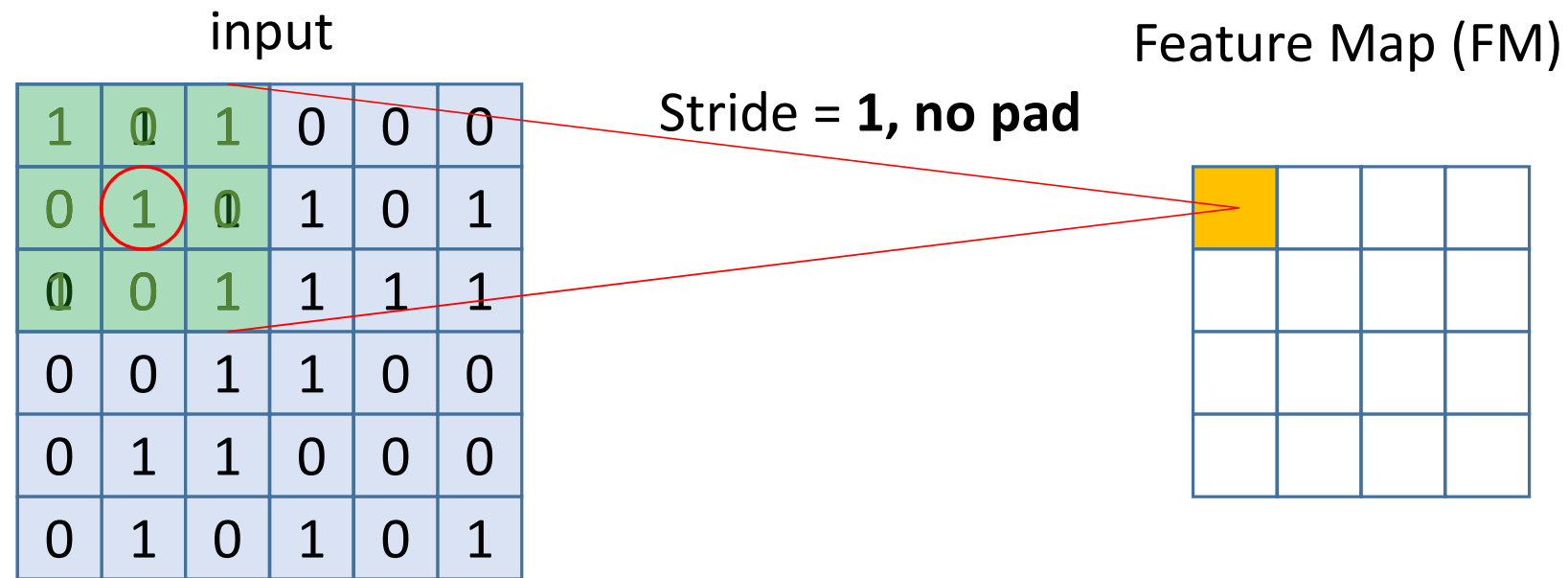
$$D2 = K$$

Stride (S): the amount of “sliding” of the kernel between each computation.

Useful to change the $W \times H$ of the output volume.

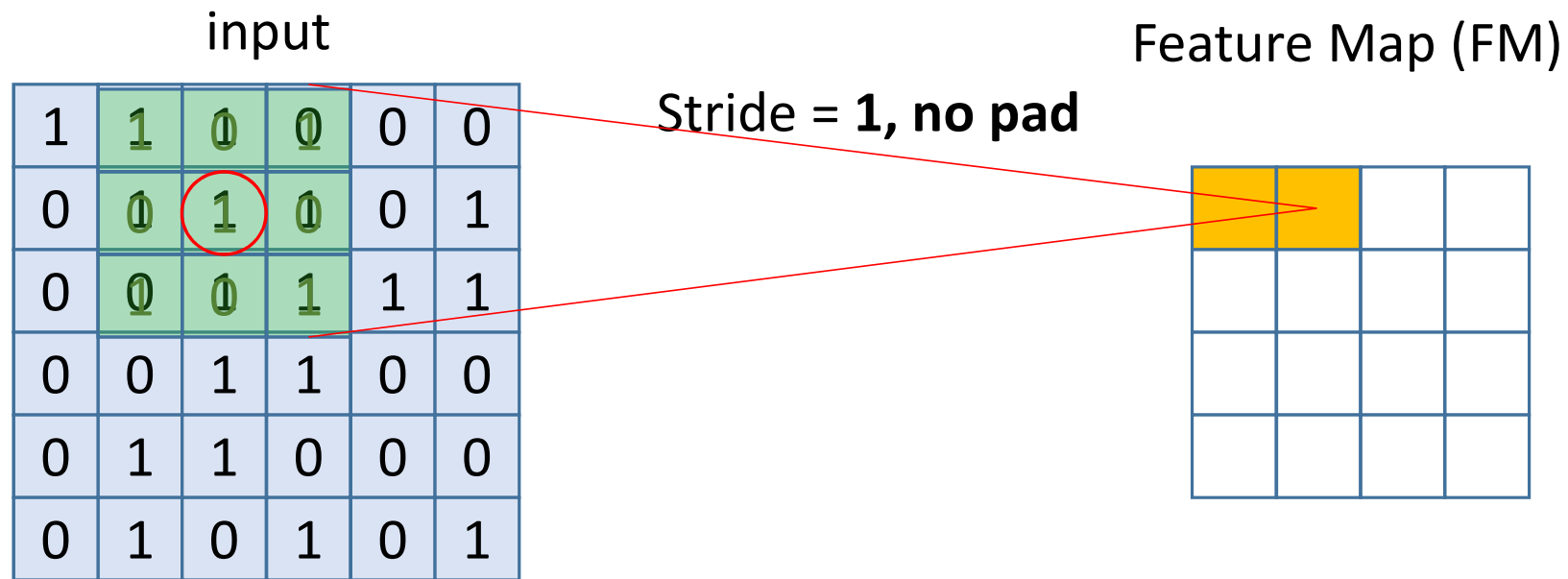
CNN building blocks: CONV (convolutional) layer

The convolutional layer is the core building block for deep learning. It consists of a set of K kernels (learned during training) with specific width, height, depth, stride. Usually small in size they extend for the whole input volume (channels num)



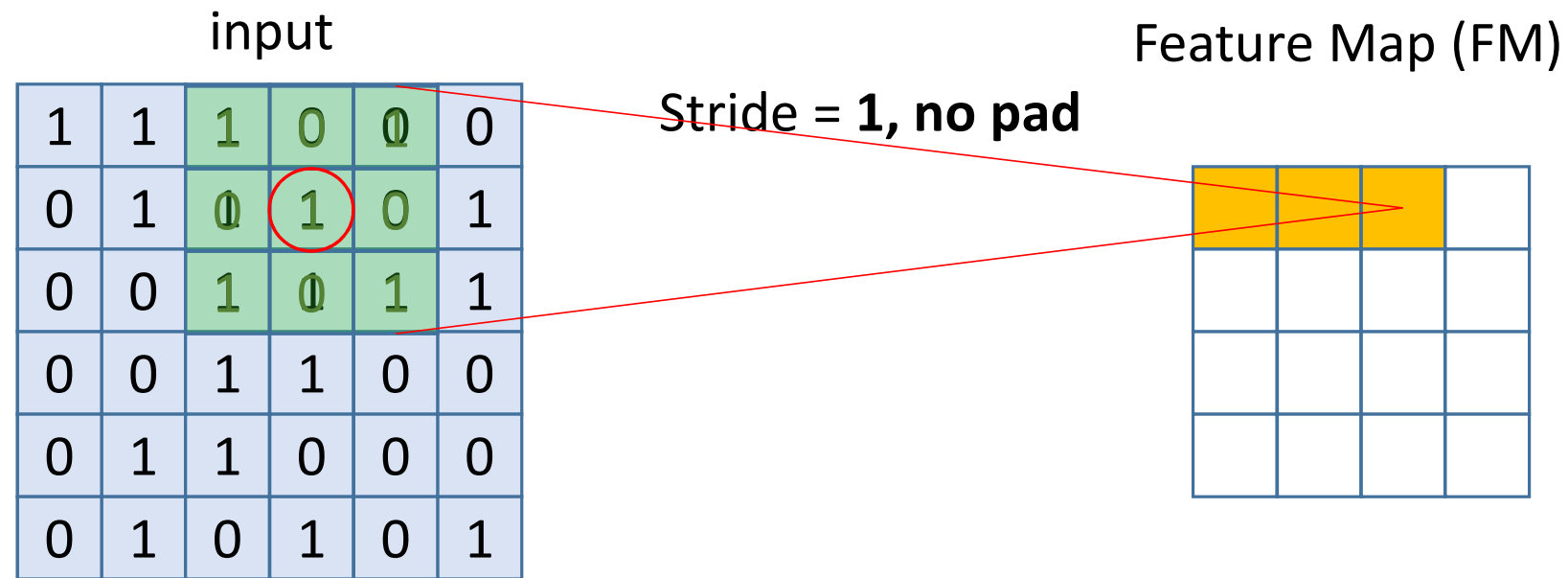
CNN building blocks: CONV (convolutional) layer

The convolutional layer is the core building block for deep learning. It consists of a set of K kernels (learned during training) with specific width, height, depth, stride. Usually small in size they extend for the whole input volume (channels num)



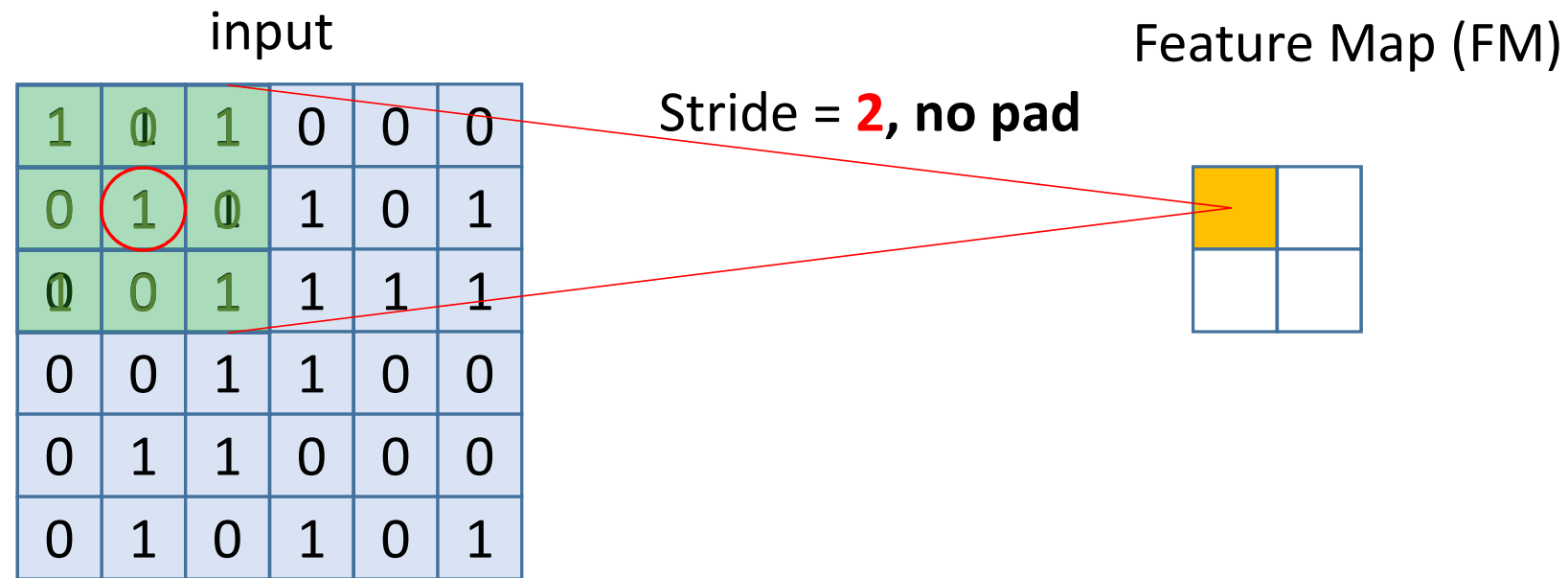
CNN building blocks: CONV (convolutional) layer

The convolutional layer is the core building block for deep learning. It consists of a set of K kernels (learned during training) with specific width, height, depth, stride. Usually small in size they extend for the whole input volume (channels num)



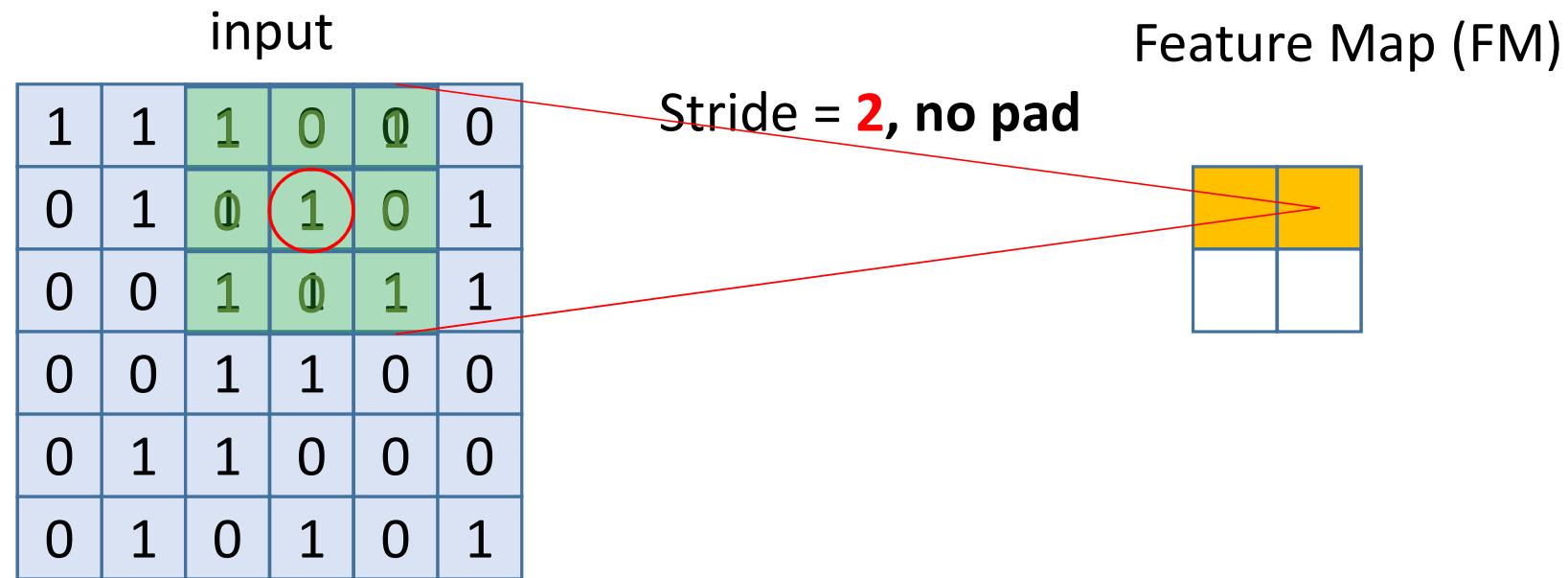
CNN building blocks: CONV (convolutional) layer

The convolutional layer is the core building block for deep learning. It consists of a set of K kernels (learned during training) with specific width, height, depth, stride. Usually small in size they extend for the whole input volume (channels num)



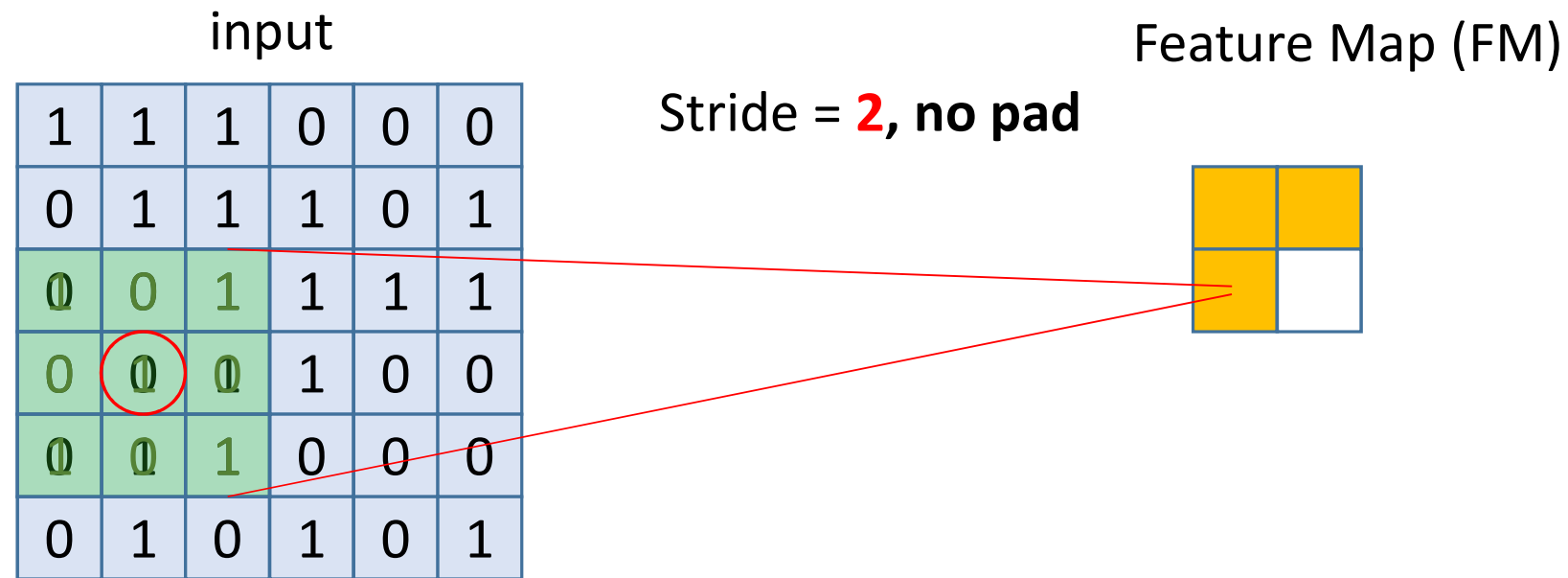
CNN building blocks: CONV (convolutional) layer

The convolutional layer is the core building block for deep learning. It consists of a set of K kernels (learned during training) with specific width, height, depth, stride. Usually small in size they extend for the whole input volume (channels num)



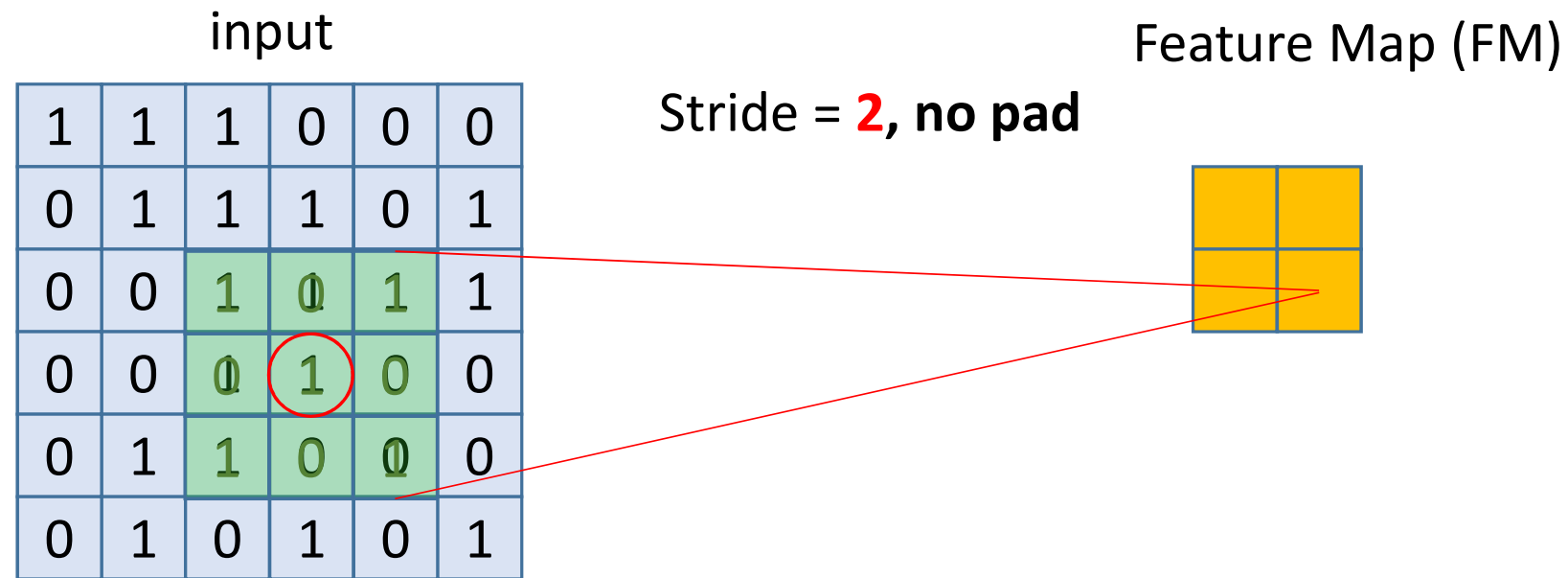
CNN building blocks: CONV (convolutional) layer

The convolutional layer is the core building block for deep learning. It consists of a set of K kernels (learned during training) with specific width, height, depth, stride. Usually small in size they extend for the whole input volume (channels num)



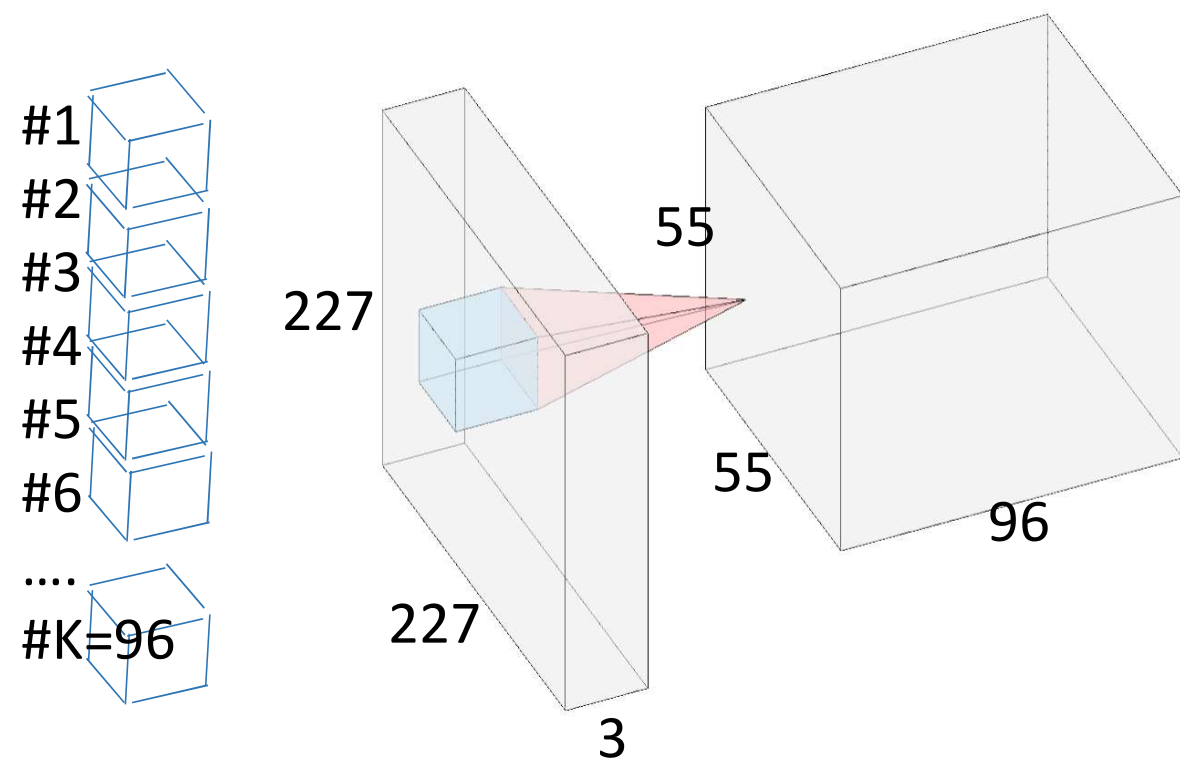
CNN building blocks: CONV (convolutional) layer

The convolutional layer is the core building block for deep learning. It consists of a set of K kernels (learned during training) with specific width, height, depth, stride. Usually small in size they extend for the whole input volume (channels num)

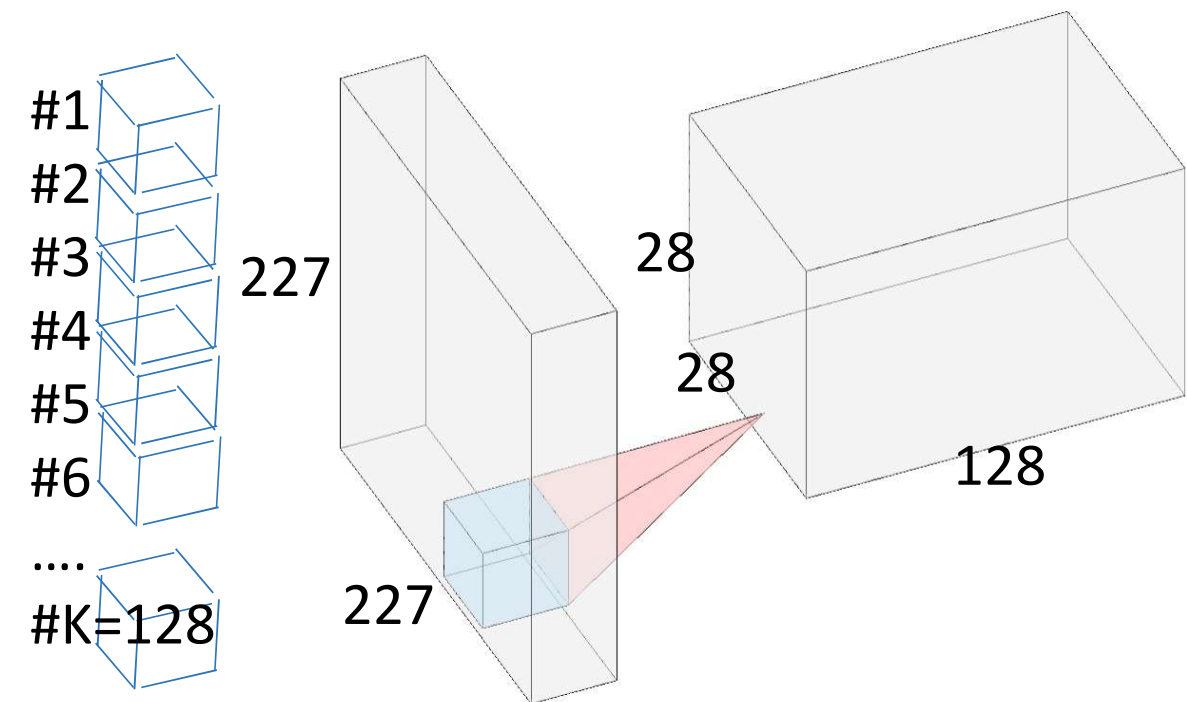


CNN building blocks: CONV (convolutional) layer

The convolutional layer is the core building block for deep learning. It consists of a set of K kernels (learned during training) with specific width, height, depth, stride. Usually small in size they extend for the whole input volume (channels num)



#96 kernels, no padding, 11x11, $s=4$



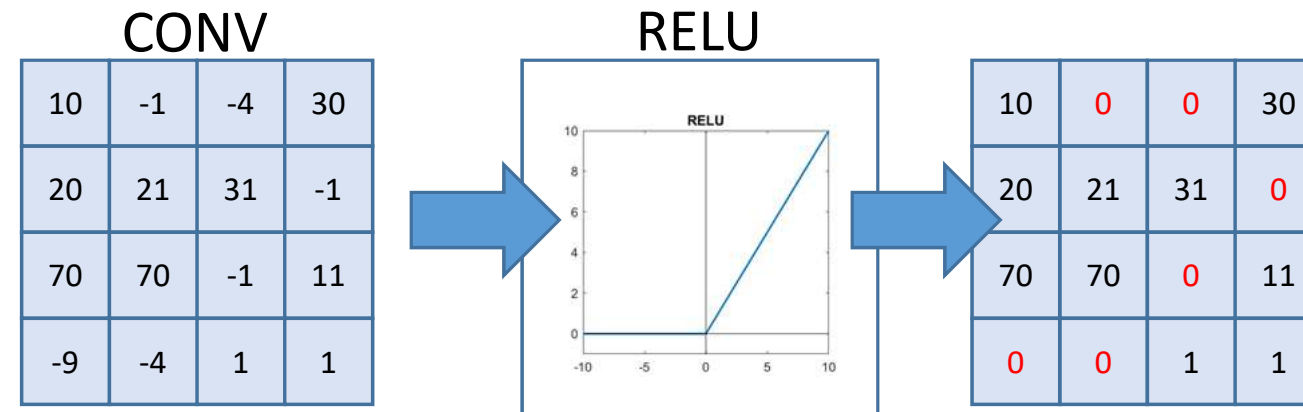
#128 kernels, no padding, 11x11, $s=8$

The output volume has one $W \times H \times D$ feature map for each of the K kernels of the CNN architecture. Each kernel coefficients are learned by the CNN during the training over the given dataset. Each feature map will “learn” one specific feature of the class we want to identify.

Input & Output data are considered as “volumes” of data.

CNN building blocks: ACT (activation) layer

Activation layers are needed to apply a non-linear activation function to all the values of a data volume. Input and output volume sizes do not change.



$$W_{out} = W_{in}$$

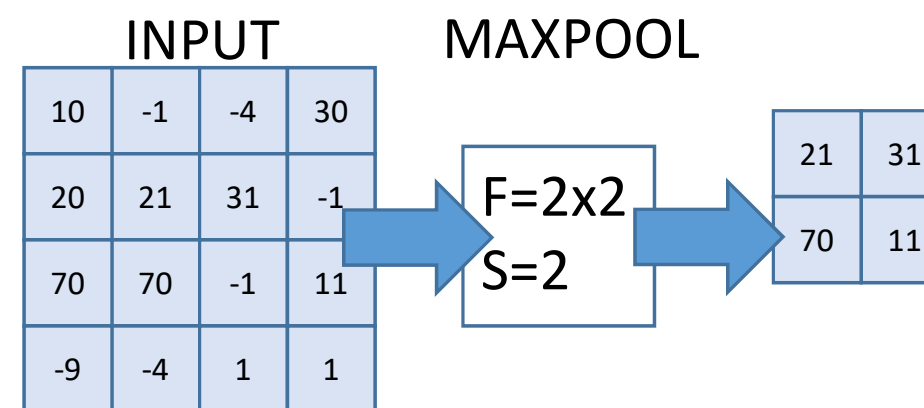
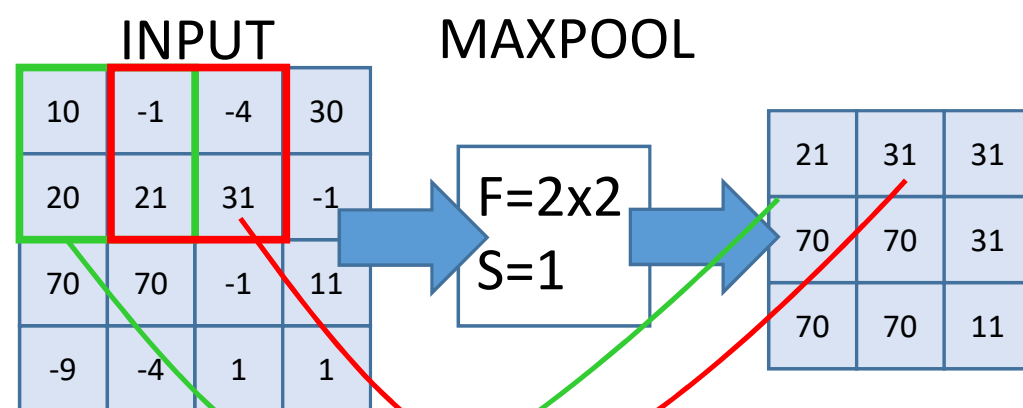
$$H_{out} = H_{in}$$

$$D_{out} = D_{in}$$

CNN building blocks: POOL (pooling) layer

The main purpose of a pooling layer is to reduce (downsample) the size of the input data volume (alternative to a stride >2 for CONV). Pooling does reduce the complexity of a model while controlling *data overfitting*.

Micro-architecture is related to the math operator while macro-architecture is related to the receptive field and stride. The most common is MAX POOL.



$$W_{out} = ((W_{in}-F)/S)+1$$

$$H_{out} = ((H_{in}-F)/S)+1$$

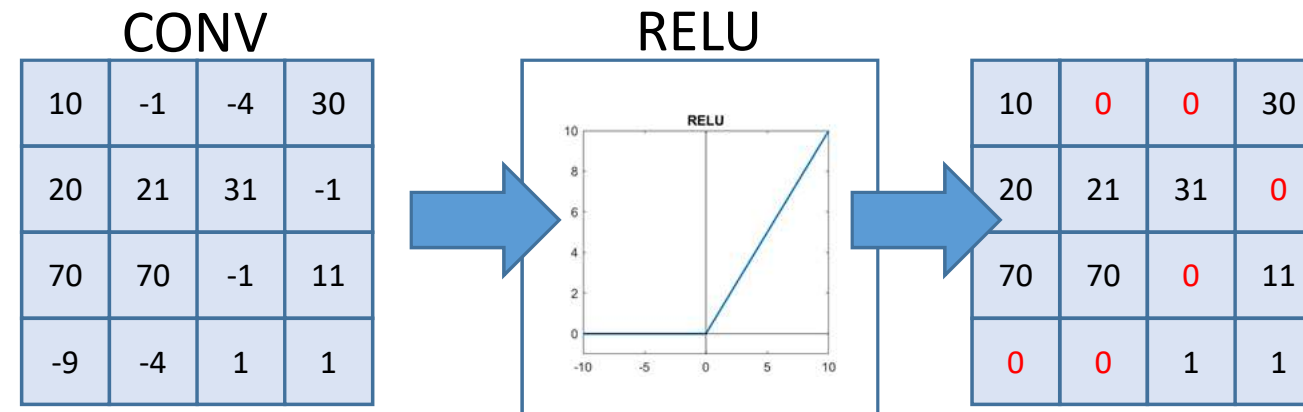
$$D_{out} = D_{in}$$



max(F)

CNN building blocks: ACT (activation) layer

Activation layers are needed to apply a non-linear activation function to all the values of a data volume. Input and output volume sizes do not change.



$$W_{out} = W_{in}$$

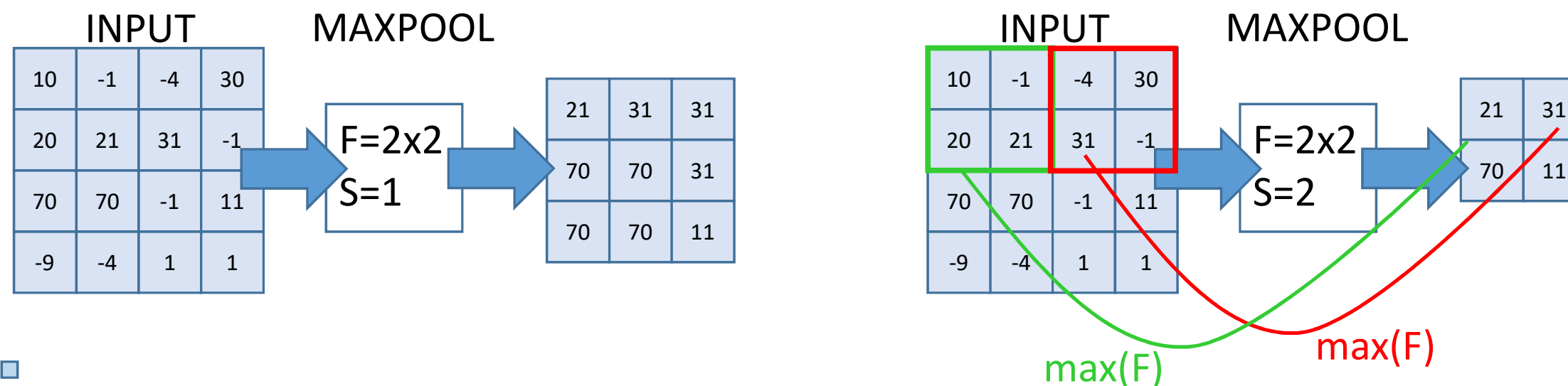
$$H_{out} = H_{in}$$

$$D_{out} = D_{in}$$

CNN building blocks: POOL (pooling) layer

The main purpose of a pooling layer is to reduce (downsample) the size of the input data volume (alternative to a stride >2 for CONV). Pooling does reduce the complexity of a model while controlling *data overfitting*.

Micro-architecture is related to the math operator while macro-architecture is related to the receptive field and stride. The most common is MAX POOL.



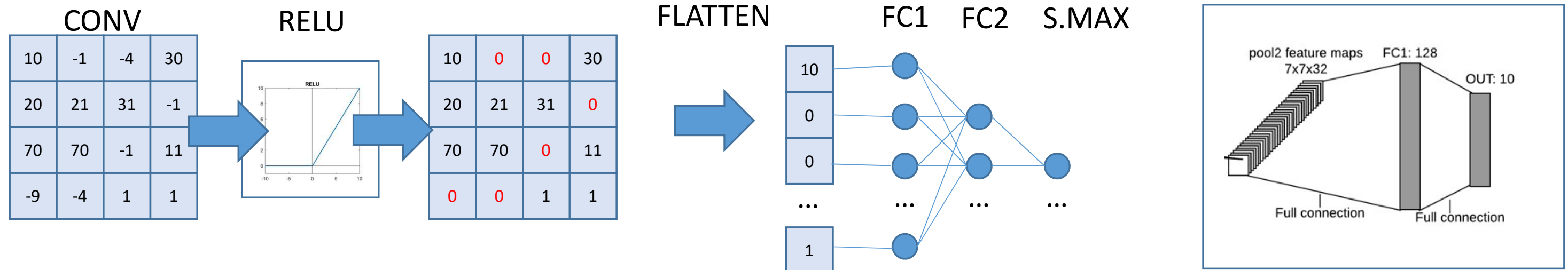
$$W_{out} = ((W_{in}-F)/S)+1$$

$$H_{out} = ((H_{in}-F)/S)+1$$

$$D_{out} = D_{in}$$

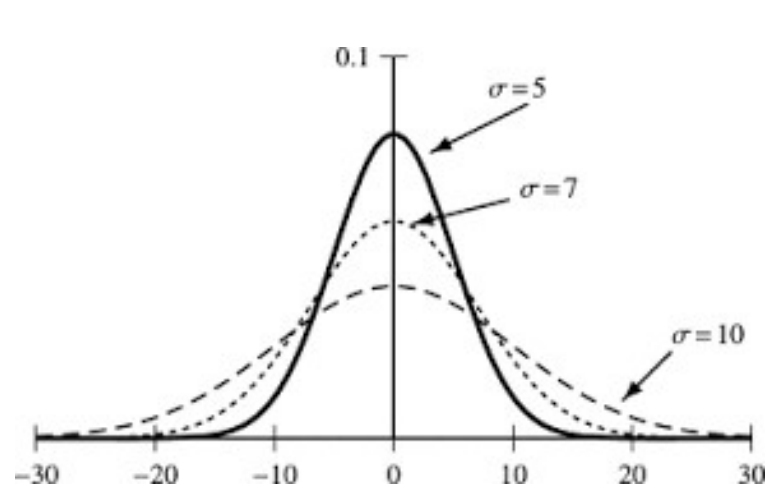
CNN building blocks: FC (fully connector) layer

This is a classical “feedforward” network and these layers are *always* used at the end of a CNN to classify the results from the previous (feature extraction) layers. It is common to place them before a SOFTMAX classifier to understand the result of data classification. This often requires “flattening” the last layer of the feature extraction part of the CNN



CNN building blocks: BN (batch normalization) layer

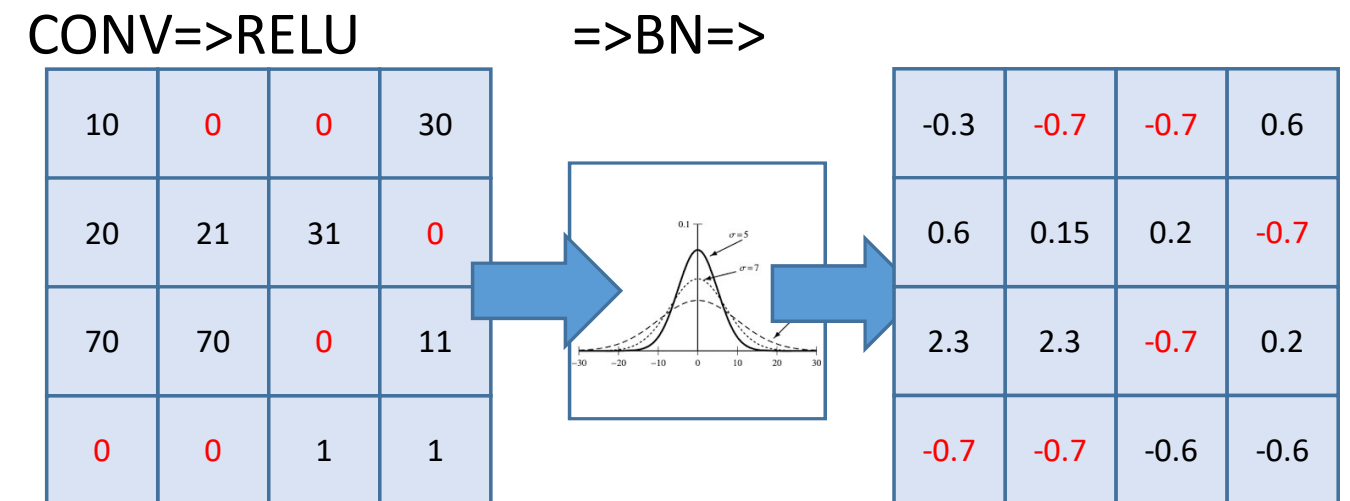
It was introduced in 2015 to have a layer which does a *normalization* of all the signals (activations) from a previous input volume and passing the result to the next layer. Extremely helpful to make the training more *stable*, i.e. robust to the intrinsic variance of input data. Penalty on computation. **It is placed *after* an activation layer.**



$$\hat{x}_i = \frac{x_i - \mu}{\sqrt{\sigma^2 + \epsilon}}$$

$$\mu = \frac{1}{M} \sum_{i=1}^m x_i$$

$$\sigma^2 = \frac{1}{M} \sum_{i=1}^m (x_i - \mu)^2$$



CNN building blocks: DO (dropout) layer

Dropout is a form of *regularization* to prevent overfitting to the expense of testing accuracy. For each batch of training dataset it will disconnect (randomly with probability P) the input of a preceding layer to the next layer in the CNN architecture. By *altering* the network architecture we make sure that there is no “preferred path” between nodes to produce a specific result. It is mostly used between FC layers, for example:

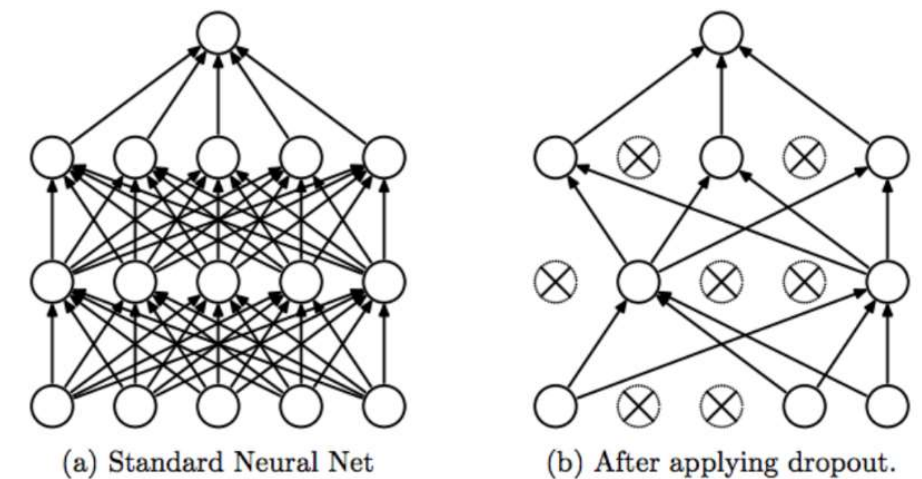
CONV=>RELU=>POOL=>FC=>**DO**=>FC=>**DO**=>FC=>SOFTMAX

CNN and Translation, Rotation, Scaling

When we use a CNN over a specific image (signal) is the result affected by an image geometrical transformation? In general CNNs are invariant only to translation thanks to the convolution properties.

For this reason it is important to *train* network with *data_augmentation*, using multiple (altered) copies of the same input, so that the inner kernels can *learn* to select features even when they are different from the most frequent use case. Using data augmentation like *scaling, cropping, skew, rotation, noise* we will reduce the training accuracy (and increase training time..) but we will obtain a more robust model, capable of *generalize* better for all the use cases.

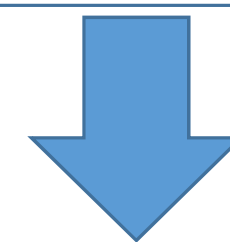
<https://towardsdatascience.com/applied-deep-learning-part-4-convolutional-neural-networks-584bc134c1e2>



Dropout: A Simple Way to Prevent Neural Networks from Overfitting
<http://jmlr.org/papers/volume15/srivastava14a.old/srivastava14a.pdf>

CNNs are:

Rotation invariant: **NO**
Scaling invariant: **NO**
Translation invariant: YES

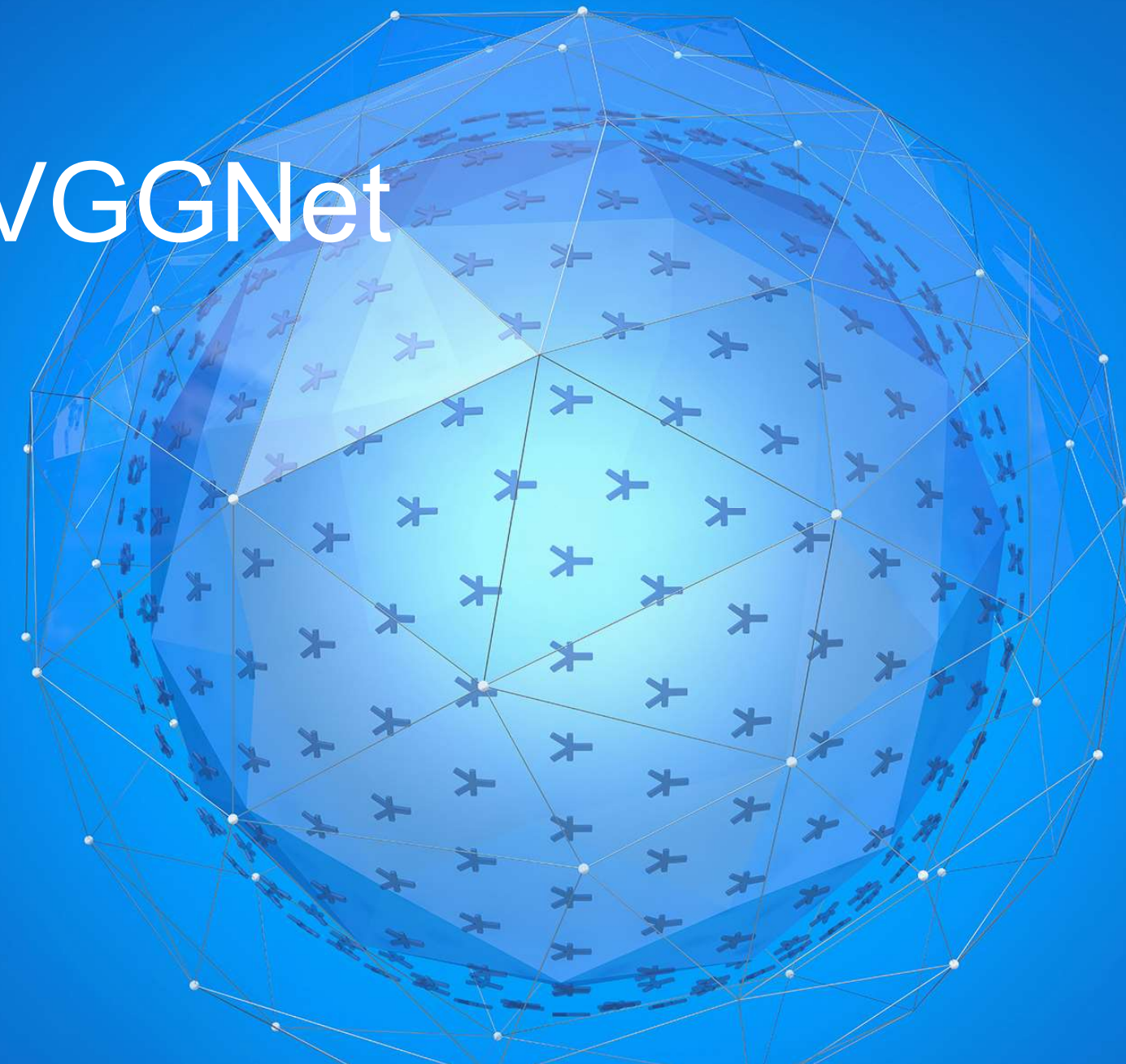


data augmentation
during training

CNNs *can be*:

Rotation invariant: **YES**
Scaling invariant: **YES**
Translation invariant: YES

CNN: LeNet, AlexNet and VGGNet



LeNet (1998)

Fundamental architecture introduced by Yann LeCun in 1998 with the paper “*Gradient-Based Learning Applied to Document Recognition*”. <http://yann.lecun.com/exdb/lenet/index.html>

The original purpose was OCR for handwritten numbers.

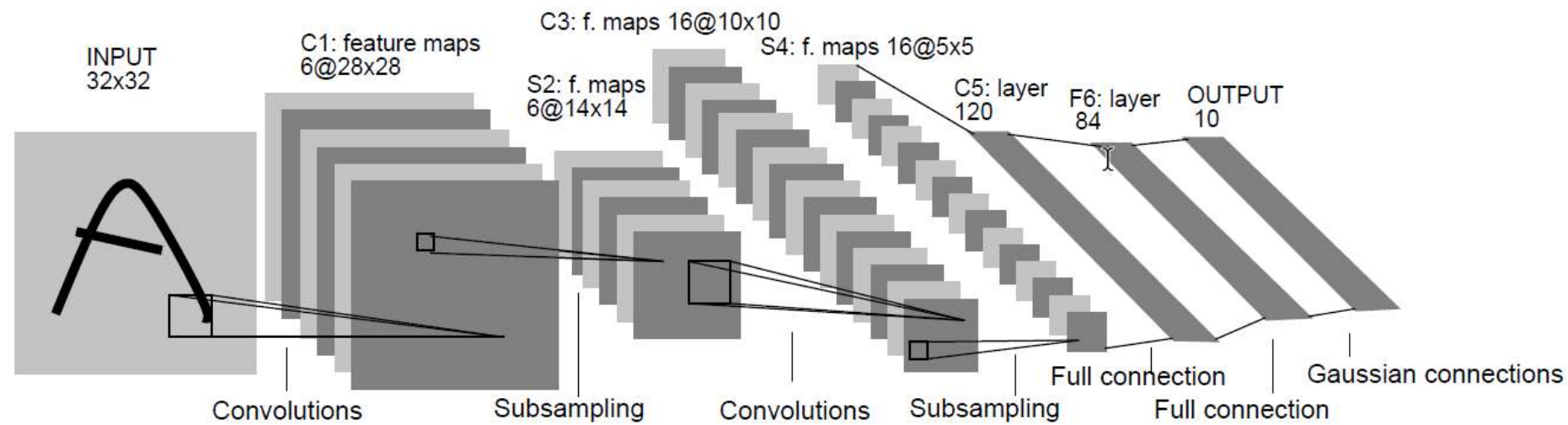


Fig. 2. Architecture of LeNet-5, a Convolutional Neural Network, here for digits recognition. Each plane is a feature map, i.e. a set of units whose weights are constrained to be identical.



https://www.youtube.com/watch?v=FwFduRA_L6Q
1993: Demo of "LeNet 1", the first convolutional network that could recognize handwritten digits with good speed and accuracy.

LeNet-5: today one of the simplest networks to *learn* how to implement CNN, often called the “Hello World” of CNN.

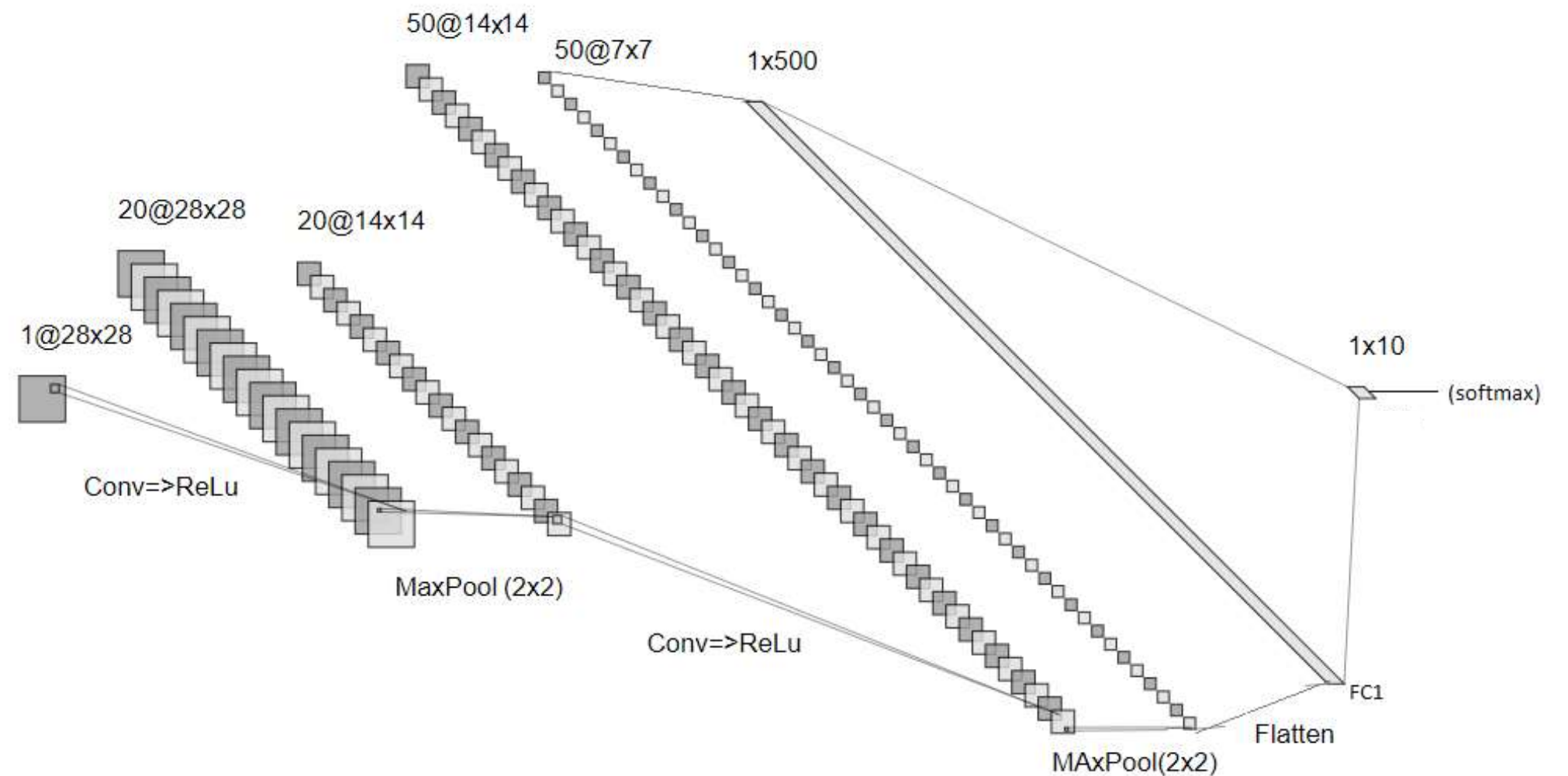
Footprint: The original LeNet-5 has **~60.000 parameters** (this is considered *very small*)

It is a *sequential* CNN, all the data are processed layer-by-layer on single dataflow until the final *classification* section at the end.

LeNet on MNIST

To implement LeNet for MNIST we are going to *modify* the network a bit. Input size is 28x28 pixels and activation is ReLu

| Layer | Output Size px | Kernel |
|-----------|----------------|--------------|
| INPUT | 28x28x1 | |
| CONV | 28x28x20 | 5x5x1, k=20 |
| ACT(ReLu) | 28x28x20 | |
| POOL | 14x14x20 | maxpool 2x2 |
| CONV | 14x14x50 | 5x5x20, k=50 |
| ACT(ReLu) | 14x14x50 | |
| POOL | 7x7x50 | maxpool 2x2 |
| FC | 500 | flattening |
| ACT(ReLu) | 500 | |
| FC | 10 | SoftMax |



IN=>CONV=>ReLu=>POOL=>CONV=>ReLu=>POOL=>FC=>ReLu=>FC=>softmax



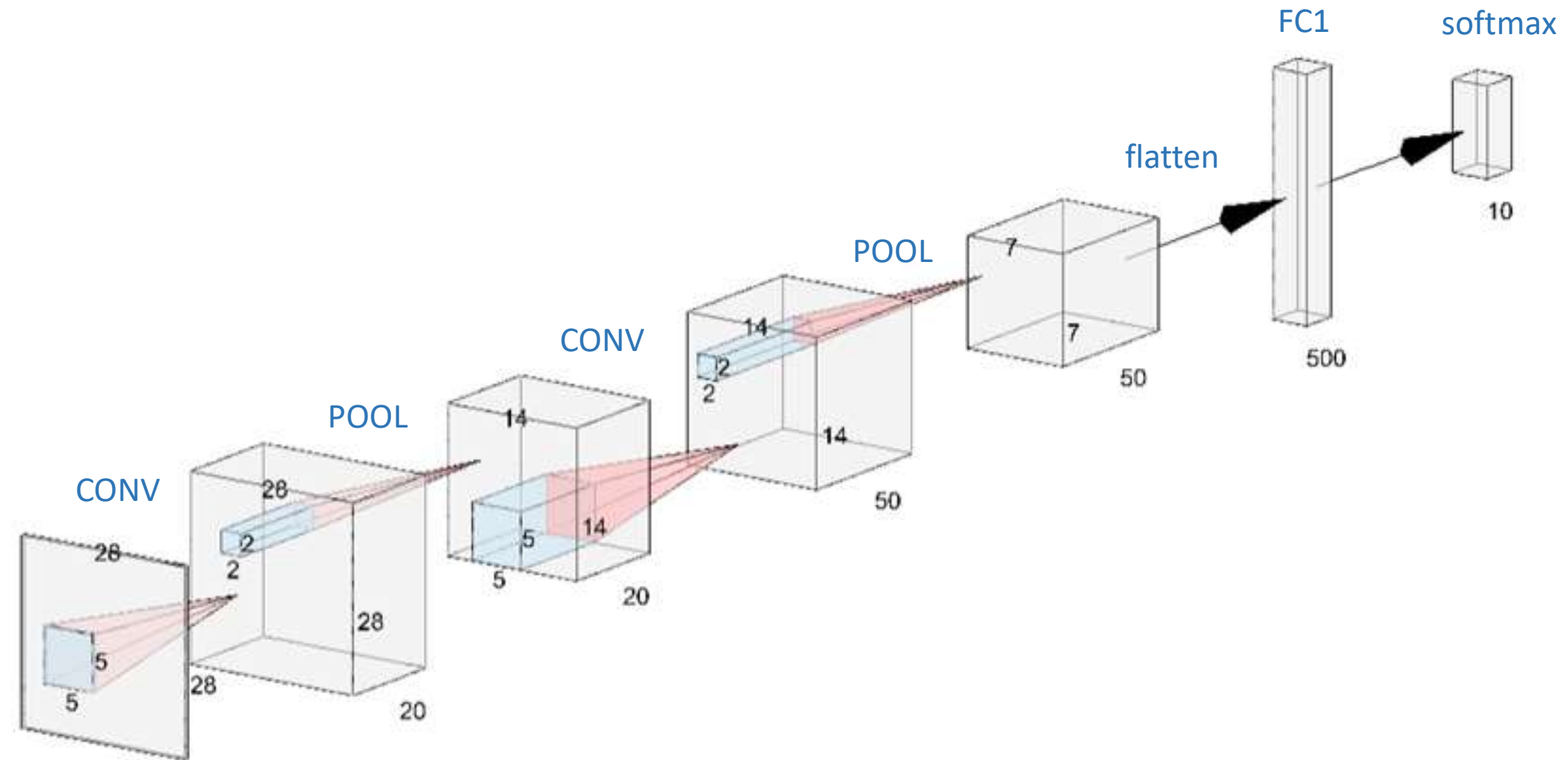
Gradient-Based Learning Applied to Document Recognition

[ref_docs\\[CNN\\]lecun_1998.pdf](#)

LeNet on MNIST

To implement LeNet for MNIST we are going to modify the network a bit. Input size is 28x28 pixels and activation is ReLu

| Layer | Output Size px | Kernel |
|-----------|----------------|--------------|
| INPUT | 28x28x1 | |
| CONV | 28x28x20 | 5x5x1, k=20 |
| ACT(ReLu) | 28x28x20 | |
| POOL | 14x14x20 | maxpool 2x2 |
| CONV | 14x14x50 | 5x5x20, k=50 |
| ACT(ReLu) | 14x14x50 | |
| POOL | 7x7x50 | maxpool 2x2 |
| FC | 500 | flattening |
| ACT(ReLu) | 500 | |
| FC | 10 | SoftMax |



IN=>CONV=>ReLu=>POOL=>CONV=>ReLu=>POOL=>FC=>ReLu=>FC=>softmax

Since the kernel size is always *few pixels wide* as a consequence we have that early layers will learn “tiny details” while deeper layers will learn “macro details”. The more we go deeper the more we add kernels. This is a common strategy for sequential CNNs.

LeNet on MNIST

Keras implementation as a *class* is very compact. Also Keras has already MNIST as an “embedded” database.

| Layer | Output Size px | Kernel |
|-----------|----------------|--------------|
| INPUT | 28x28x1 | |
| CONV | 28x28x20 | 5x5x1, k=20 |
| ACT(ReLu) | 28x28x20 | |
| POOL | 14x14x20 | maxpool 2x2 |
| CONV | 14x14x50 | 5x5x20, k=50 |
| ACT(ReLu) | 14x14x50 | |
| POOL | 7x7x50 | maxpool 2x2 |
| FC | 500 | flattening |
| ACT(ReLu) | 500 | |
| FC | 10 | SoftMax |

```
1 class LeNet:
2     @staticmethod
3     def build(width, height, depth, classes):
4         # initialize the model
5         model = Sequential()
6         inputShape = (height, width, depth)
7
8         # STAGE1: CONV => RELU => POOL layers
9         model.add(Conv2D(20, (5, 5), padding="same", input_shape=inputShape))
10        model.add(Activation("relu"))
11        model.add(MaxPooling2D(pool_size=(2, 2), strides=(2, 2)))
12
13        # STAGE2: CONV => RELU => POOL layers
14        model.add(Conv2D(50, (5, 5), padding="same"))
15        model.add(Activation("relu"))
16        model.add(MaxPooling2D(pool_size=(2, 2), strides=(2, 2)))
17
18        # STAGE3: FC1 => RELU layers
19        model.add(Flatten())
20        model.add(Dense(500))
21        model.add(Activation("relu"))
22
23        # softmax classifier: we have 10 classes
24        model.add(Dense(classes))
25        model.add(Activation("softmax"))
26
27        # return the constructed network architecture
28        return model
```



IT'S
**DEMO
TIME!**

Once the network class is defined we only have three steps to follow: *build&compile*, *fit*(i.e. train), *predict*(i.e. evaluate).

```
#1 model = LeNet.build(width=28, height=28, depth=1, classes=10)
   model.compile(loss="categorical_crossentropy", optimizer=opt, metrics=["accuracy"])
```

```
#2 H = model.fit(trainData, trainLabels,
               validation_data=(testData, testLabels), batch_size=128, epochs=20, verbose=1)
```

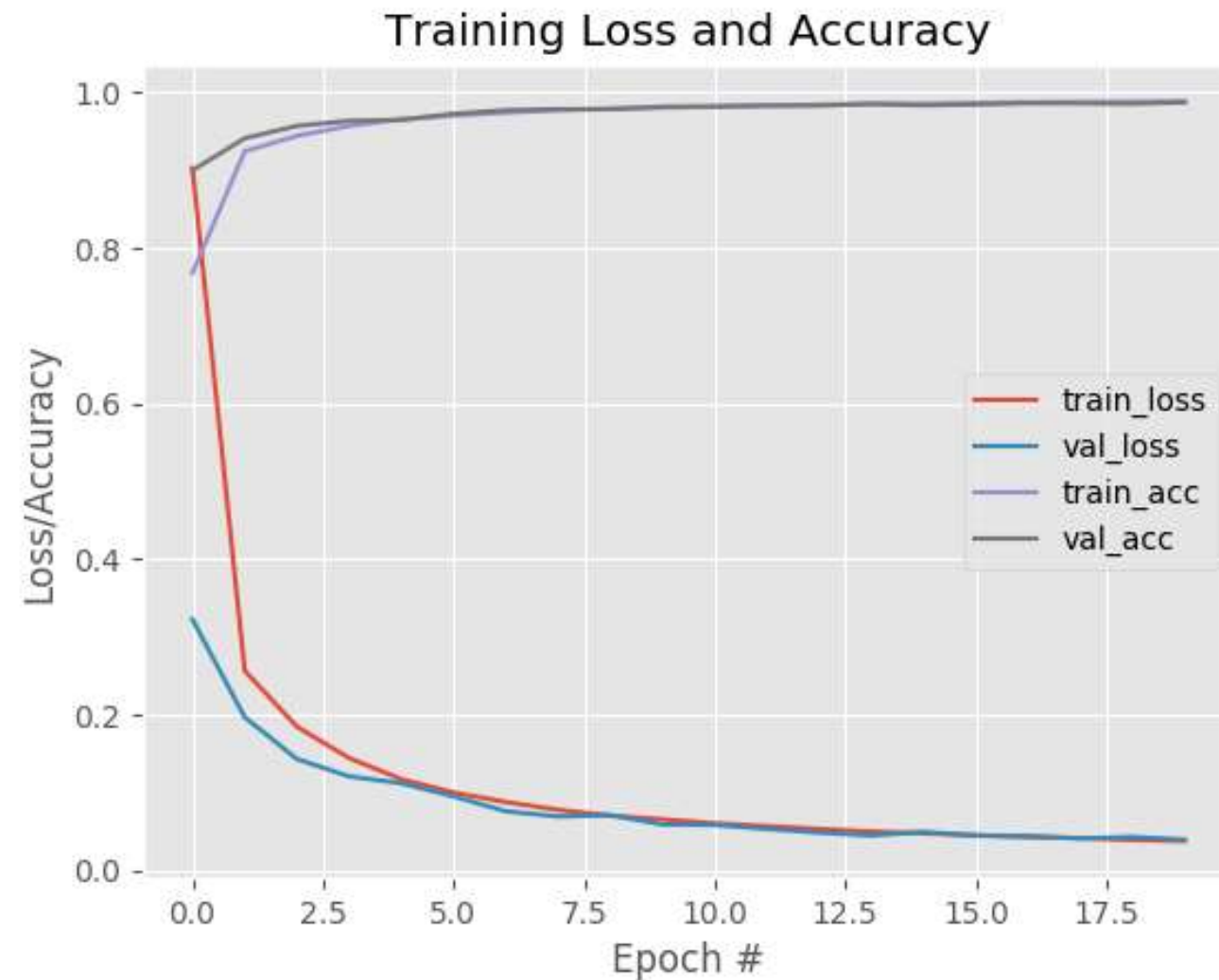
```
#3 predictions = model.predict(testData, batch_size=128)
```

Note: training is done on the training dataset

Note: test dataset is used also for validation
(but is *never* used for training...)

LeNet on MNIST

Keras implementation as a *class* is very compact. Also Keras has already MNIST as an “embedded” database.



| digit | precision | recall | f1-score | support |
|----------|-----------|--------|----------|---------|
| 0 | 0.99 | 0.99 | 0.99 | 980 |
| 1 | 1.00 | 0.99 | 0.99 | 1135 |
| 2 | 0.98 | 0.99 | 0.99 | 1032 |
| 3 | 0.99 | 0.99 | 0.99 | 1010 |
| 4 | 0.99 | 0.99 | 0.99 | 982 |
| 5 | 0.99 | 0.99 | 0.99 | 892 |
| 6 | 0.99 | 0.99 | 0.99 | 958 |
| 7 | 0.99 | 0.98 | 0.99 | 1028 |
| 8 | 0.97 | 0.99 | 0.98 | 974 |
| 9 | 0.99 | 0.97 | 0.98 | 1009 |
| accuracy | | | 0.99 | 10000 |

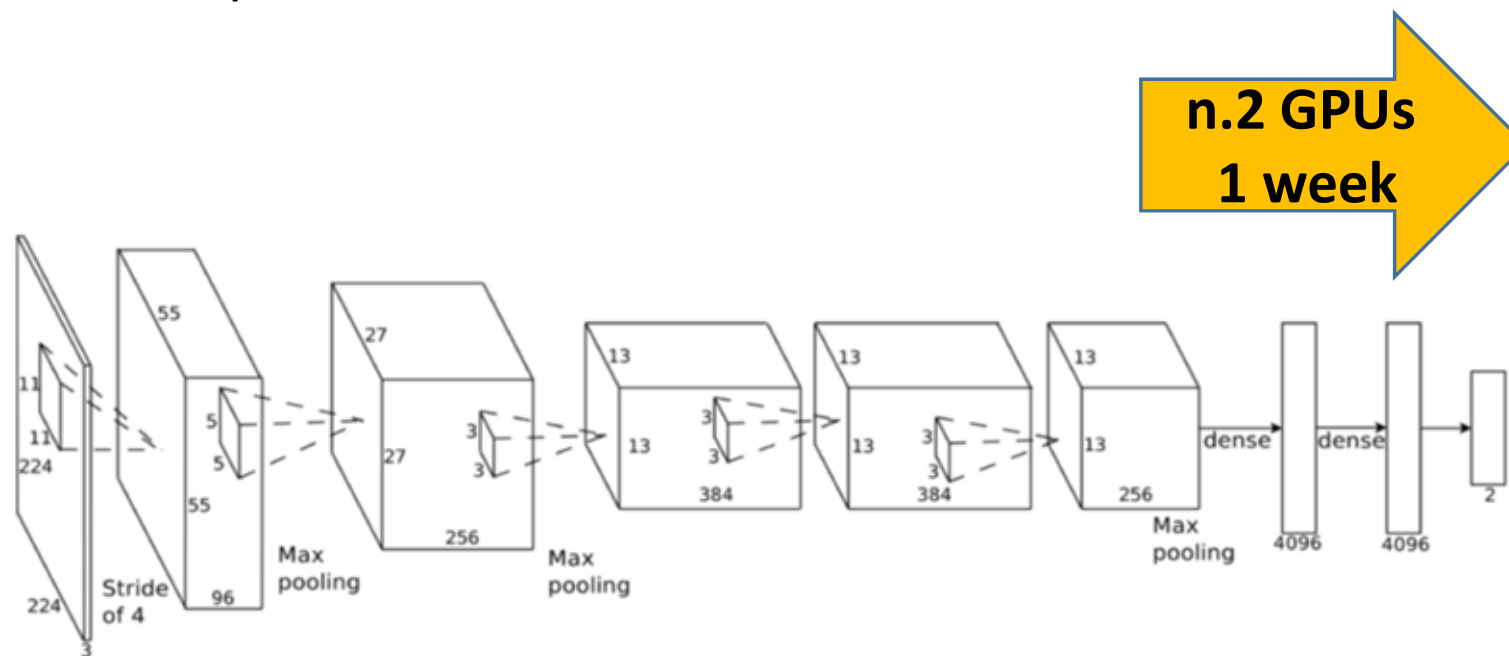
NOTE: MNIST is an “easy” dataset for todays networks (*high risk of overfitting*)

AlexNET (2012)

Developed at the Univ. of Toronto. Was specifically created to compete at the **ImageNet Large Scale Visual Recognition Challenge 2012** (ILSVRC2012). It won the competition thanks to the “SuperVision” team.

The model follows the footsteps of LeNet with a larger footprint.

The success of AlexNet was a turning point for the computer vision community which started to work heavily on new optimized models of CNNs



n.2 GPUs
1 week

SuperVision

Alex Krizhevsky, Ilya Sutskever, Geoffrey Hinton
University of Toronto

Our model is a large, deep convolutional neural network trained on raw RGB pixel values. The neural network, which has **60 million parameters** and **650,000 neurons**, consists of **five convolutional layers**, some of which are followed by max-pooling layers, and three globally-connected layers with a final 1000-way softmax. **It was trained on two NVIDIA GPUs for about a week.** To make training faster, we used non-saturating neurons and a very efficient GPU implementation of convolutional nets. To reduce overfitting in the globally-connected layers we employed hidden-unit "dropout", a recently-developed regularization method that proved to be very effective.

<http://image-net.org/challenges/LSVRC/2012/results.html#t1>



ImageNet Classification with Deep Convolutional Neural Networks

[ref_docs\\[CNN\]AlexNet.pdf](#)

AlexNET (2012)

Section-A

| Layer | Output Size px | Kernel |
|---------|----------------|------------------------|
| INPUT | 227x227x3 | (NOTE: 224 was a typo) |
| CONV | 55x55x96 | 11x11 S=4 K=96 |
| ReLu | 55x55x96 | |
| BN | 55x55x96 | |
| POOL | 27x27x96 | 3x3, S=2 |
| DROPOUT | 27x27x96 | 0.25 |
| CONV | 27x27x256 | 5x5, S=1 K=256 |
| ReLu | 27x27x256 | |
| BN | 27x27x256 | |
| POOL | 13x13x256 | 3x3, S=2 |
| DROPOUT | 13x13x256 | 0.25 |

Section-B

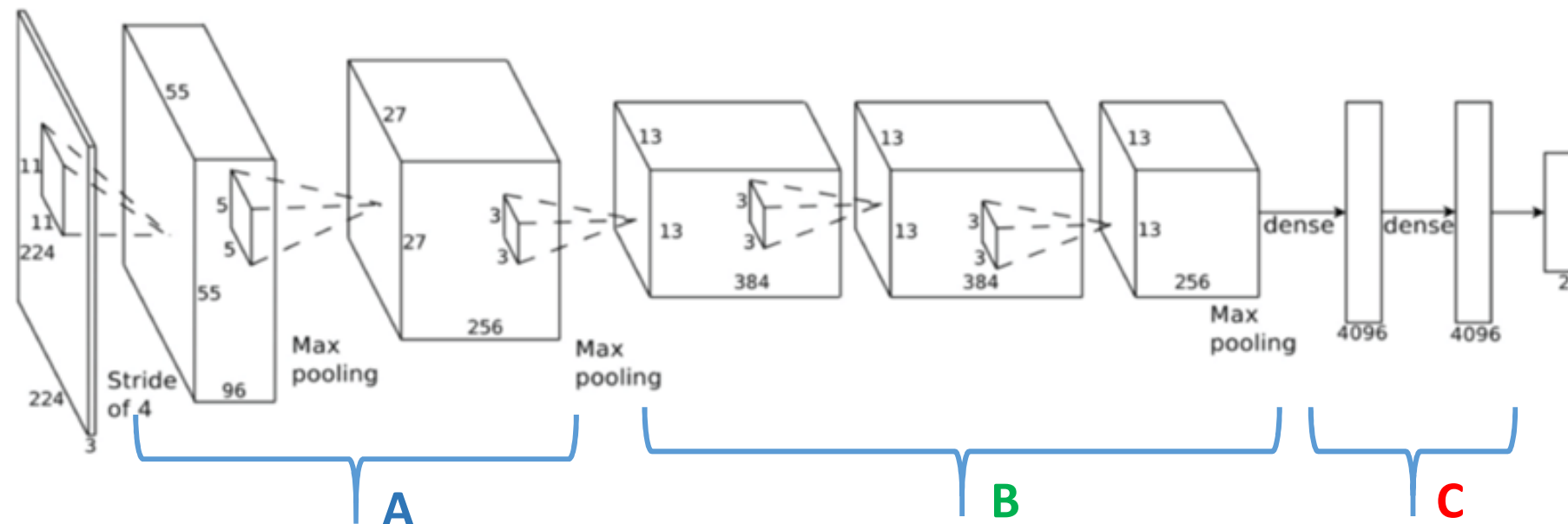
| Layer | Output Size px | Kernel |
|---------|----------------|----------------|
| CONV | 13x13x384 | 3x3, S=1 K=384 |
| ReLu | 13x13x384 | |
| BN | 13x13x384 | |
| CONV | 13x13x384 | 3x3, S=1 K=384 |
| ReLu | 13x13x384 | |
| BN | 13x13x384 | |
| CONV | 13x13x256 | 3x3, S=1 K=256 |
| ReLu | 13x13x256 | |
| BN | 13x13x256 | |
| POOL | 6x6x256 | 3x3, S=2 |
| DROPOUT | 6x6x256 | |

Section-C

| Layer | Output Size px | Kernel |
|---------|----------------|---------|
| FC | 4096 | |
| ReLu | 4096 | |
| BN | 4096 | |
| DROPOUT | 4096 | |
| FC | 4096 | |
| ReLu | 4096 | |
| BN | 4096 | |
| DROPOUT | 4096 | |
| FC | 1000 | SoftMax |

CNN Complexity:

- 60M parameters
- 5 CONV layers
- 3 FC layers



AlexNET (modified)

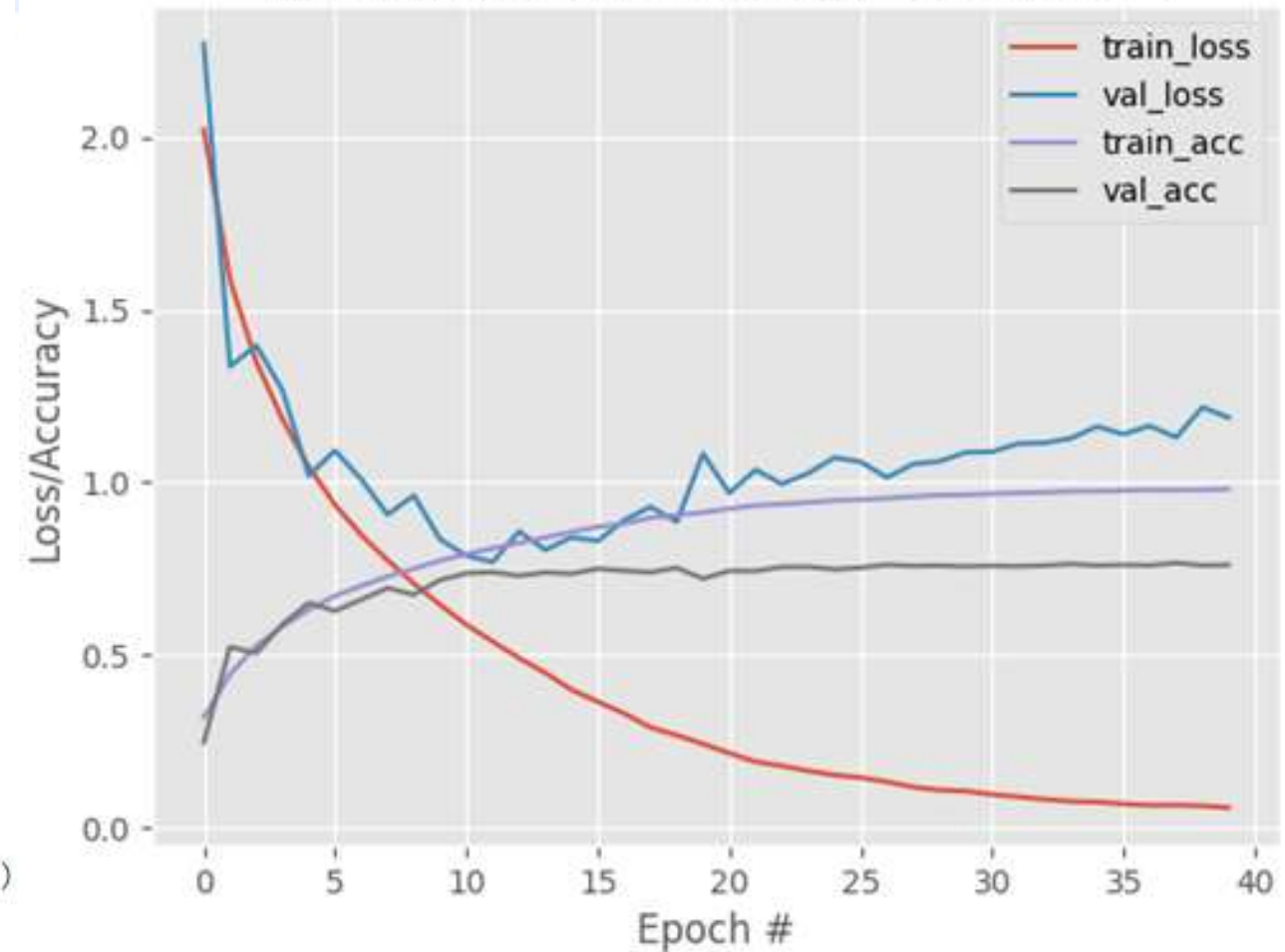
Modified (sub-optimal) network to show training on CIFAR-10 on the local CPU (no GPU)

```
1 class AlexNet:
2     @staticmethod
3     def build(width, height, depth, classes):
4         # initialize
5         model = Sequential()
6         inputShape = (height, width, depth)
7         chanDim = -1
8
9         #
10        # SECTION - A
11        #
12
13        # CONV n.1
14        model.add(Conv2D(filters=96, input_shape=(32,32,3), kernel_size=(7,7),\
15            strides=(2,2), padding='same'))
16        model.add(Activation('relu'))
17        # BN
18        model.add(BatchNormalization())
19        # POOL
20        model.add(MaxPooling2D(pool_size=(2,2), strides=(2,2), padding='same'))
21        # Dropout to prevent overfitting
22        model.add(Dropout(0.25))
23
24        # CONV n.2
25        model.add(Conv2D(filters=256, kernel_size=(5,5), strides=(1,1), padding='same'))
26        model.add(Activation('relu'))
27        # BN
28        model.add(BatchNormalization())
29        # POOL
30        model.add(MaxPooling2D(pool_size=(2,2), strides=(2,2), padding='same'))
31        # Dropout
32        model.add(Dropout(0.25))
33
34        #
35        # SECTION - B
36        #
37
38        # CONV n.3
39        model.add(Conv2D(filters=384, kernel_size=(3,3), strides=(1,1), padding='same'))
40        model.add(Activation('relu'))
41        # BN
```



IT'S
DEMO
TIME!

Training Loss and Accuracy on CIFAR-10



| | precision | recall | f1-score | support |
|------------|-----------|--------|----------|---------|
| airplane | 0.82 | 0.82 | 0.82 | 1000 |
| automobile | 0.89 | 0.84 | 0.87 | 1000 |
| bird | 0.73 | 0.63 | 0.67 | 1000 |
| cat | 0.56 | 0.59 | 0.57 | 1000 |
| deer | 0.70 | 0.74 | 0.72 | 1000 |
| dog | 0.61 | 0.71 | 0.66 | 1000 |
| frog | 0.79 | 0.83 | 0.81 | 1000 |
| horse | 0.84 | 0.77 | 0.80 | 1000 |
| ship | 0.88 | 0.85 | 0.87 | 1000 |
| truck | 0.85 | 0.82 | 0.83 | 1000 |
| accuracy | | | 0.76 | 10000 |

AlexNET (modified)

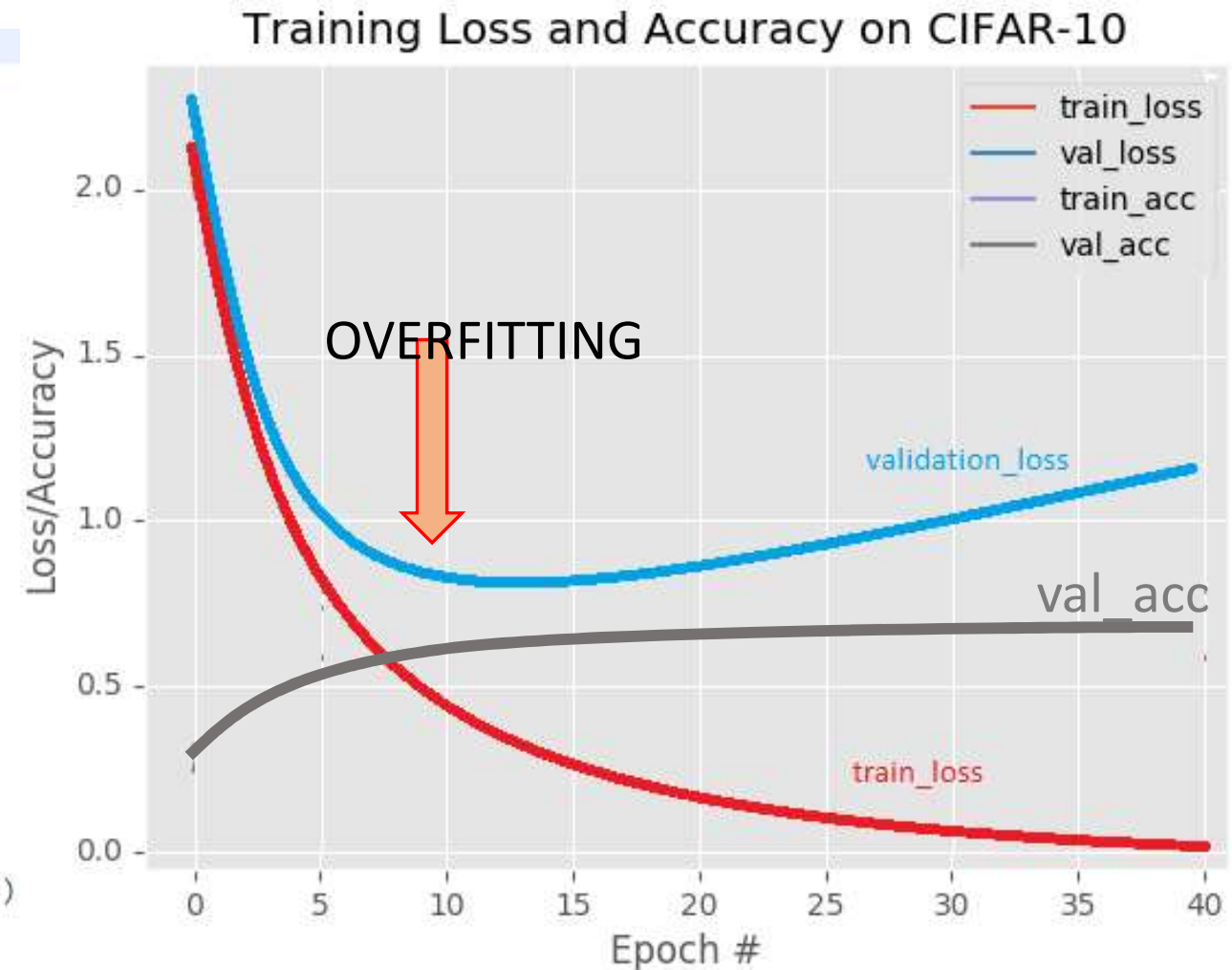
Modified (sub-optimal) network to show training on CIFAR-10 on the local CPU (no GPU)



```

1 class AlexNet:
2     @staticmethod
3     def build(width, height, depth, classes):
4         # initialize
5         model = Sequential()
6         inputShape = (height, width, depth)
7         chanDim = -1
8
9         #
10        # SECTION - A
11        #
12
13        # CONV n.1
14        model.add(Conv2D(filters=96, input_shape=(32,32,3), kernel_size=(7,7),\
15            strides=(2,2), padding='same'))
16        model.add(Activation('relu'))
17        # BN
18        model.add(BatchNormalization())
19        # POOL
20        model.add(MaxPooling2D(pool_size=(2,2), strides=(2,2), padding='same'))
21        # Dropout to prevent overfitting
22        model.add(Dropout(0.25))
23
24        # CONV n.2
25        model.add(Conv2D(filters=256, kernel_size=(5,5), strides=(1,1), padding='same'))
26        model.add(Activation('relu'))
27        # BN
28        model.add(BatchNormalization())
29        # POOL
30        model.add(MaxPooling2D(pool_size=(2,2), strides=(2,2), padding='same'))
31        # Dropout
32        model.add(Dropout(0.25))
33
34        #
35        # SECTION - B
36        #
37
38        # CONV n.3
39        model.add(Conv2D(filters=384, kernel_size=(3,3), strides=(1,1), padding='same'))
40        model.add(Activation('relu'))
41        # BN

```



| | precision | recall | f1-score | support |
|------------|-----------|--------|----------|---------|
| airplane | 0.82 | 0.82 | 0.82 | 1000 |
| automobile | 0.89 | 0.84 | 0.87 | 1000 |
| bird | 0.73 | 0.63 | 0.67 | 1000 |
| cat | 0.56 | 0.59 | 0.57 | 1000 |
| deer | 0.70 | 0.74 | 0.72 | 1000 |
| dog | 0.61 | 0.71 | 0.66 | 1000 |
| frog | 0.79 | 0.83 | 0.81 | 1000 |
| horse | 0.84 | 0.77 | 0.80 | 1000 |
| ship | 0.88 | 0.85 | 0.87 | 1000 |
| truck | 0.85 | 0.82 | 0.83 | 1000 |
| accuracy | | | 0.76 | 10000 |

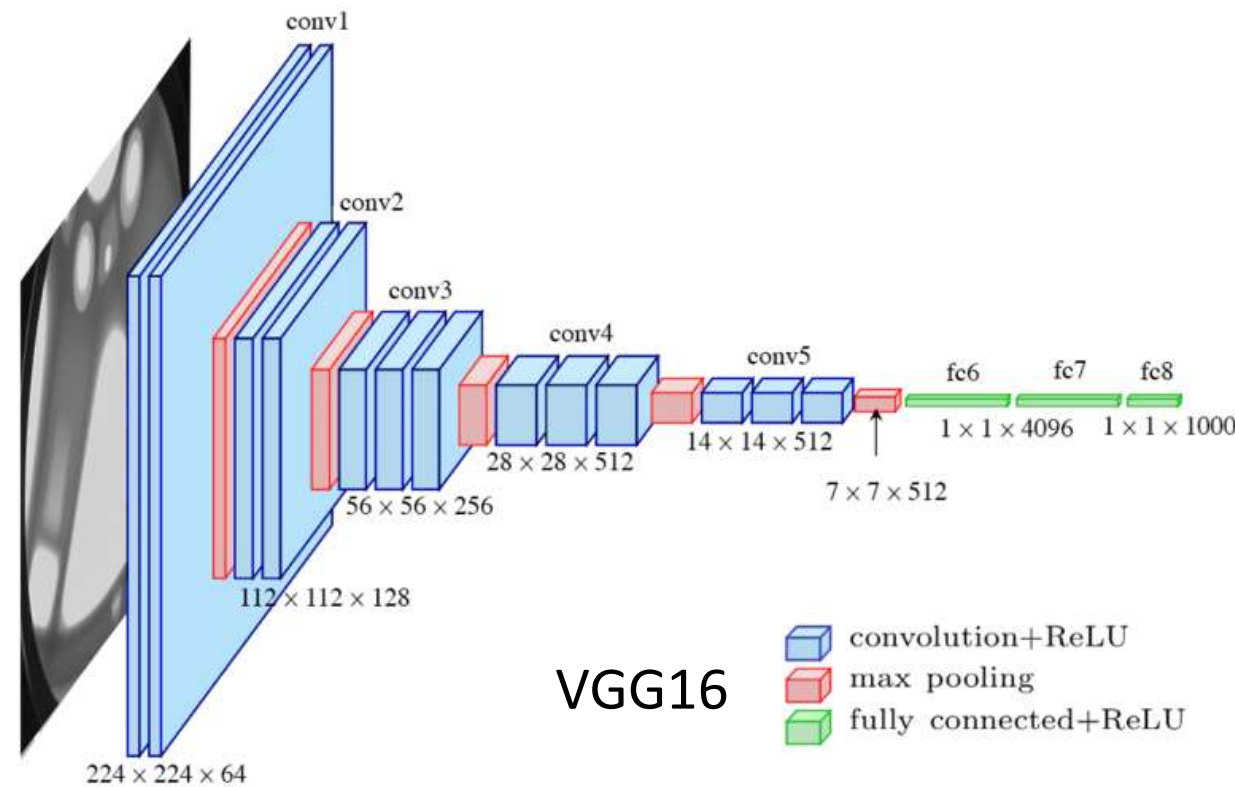


VGGNet (2014)

Visual Geometry Group, Department of Engineering Science, University of Oxford.

Introduced in 2014 was the first architecture to improve the performance for large scale image recognition. While previous networks used a variety of kernel sizes VGG uses small 3x3 kernels across the whole network. In addition it does use *multiple* convolutional layers “*stacked*” to improve the feature extraction.

This network has a large number of parameters (VGG16 ~138M, VGG19 ~144M) and requires a lot of computational power. The VGG11/VGG16/VGG19 refers to the number of layers (CONV+FC)



VGG16

| ConvNet Configuration | | | | | |
|-----------------------------|------------------------|------------------------|-------------------------------------|-------------------------------------|--|
| A | A-LRN | B | C | D | E |
| 11 weight layers | 11 weight layers | 13 weight layers | 16 weight layers | 16 weight layers | 19 weight layers |
| input (224 x 224 RGB image) | | | | | |
| conv3-64 | conv3-64 LRN | conv3-64 conv3-64 | conv3-64 conv3-64 | conv3-64 conv3-64 | conv3-64 conv3-64 |
| maxpool | | | | | |
| conv3-128 | conv3-128 | conv3-128 conv3-128 | conv3-128 conv3-128 | conv3-128 conv3-128 | conv3-128 conv3-128 |
| maxpool | | | | | |
| conv3-256 conv3-256 | conv3-256 conv3-256 | conv3-256 conv3-256 | conv3-256 conv3-256 conv1-256 | conv3-256 conv3-256 conv3-256 | conv3-256 conv3-256 conv3-256 conv3-256 |
| maxpool | | | | | |
| conv3-512 conv3-512 | conv3-512 conv3-512 | conv3-512 conv3-512 | conv3-512 conv3-512 conv1-512 | conv3-512 conv3-512 conv3-512 | conv3-512 conv3-512 conv3-512 conv3-512 |
| maxpool | | | | | |
| conv3-512 conv3-512 | conv3-512 conv3-512 | conv3-512 conv3-512 | conv3-512 conv3-512 conv1-512 | conv3-512 conv3-512 conv3-512 | conv3-512 conv3-512 conv3-512 conv3-512 |
| maxpool | | | | | |
| FC-4096 | | | | | |
| FC-4096 | | | | | |
| FC-1000 | | | | | |
| soft-max | | | | | |



Very Deep Convolutional Networks for Large-Scale Image Recognition

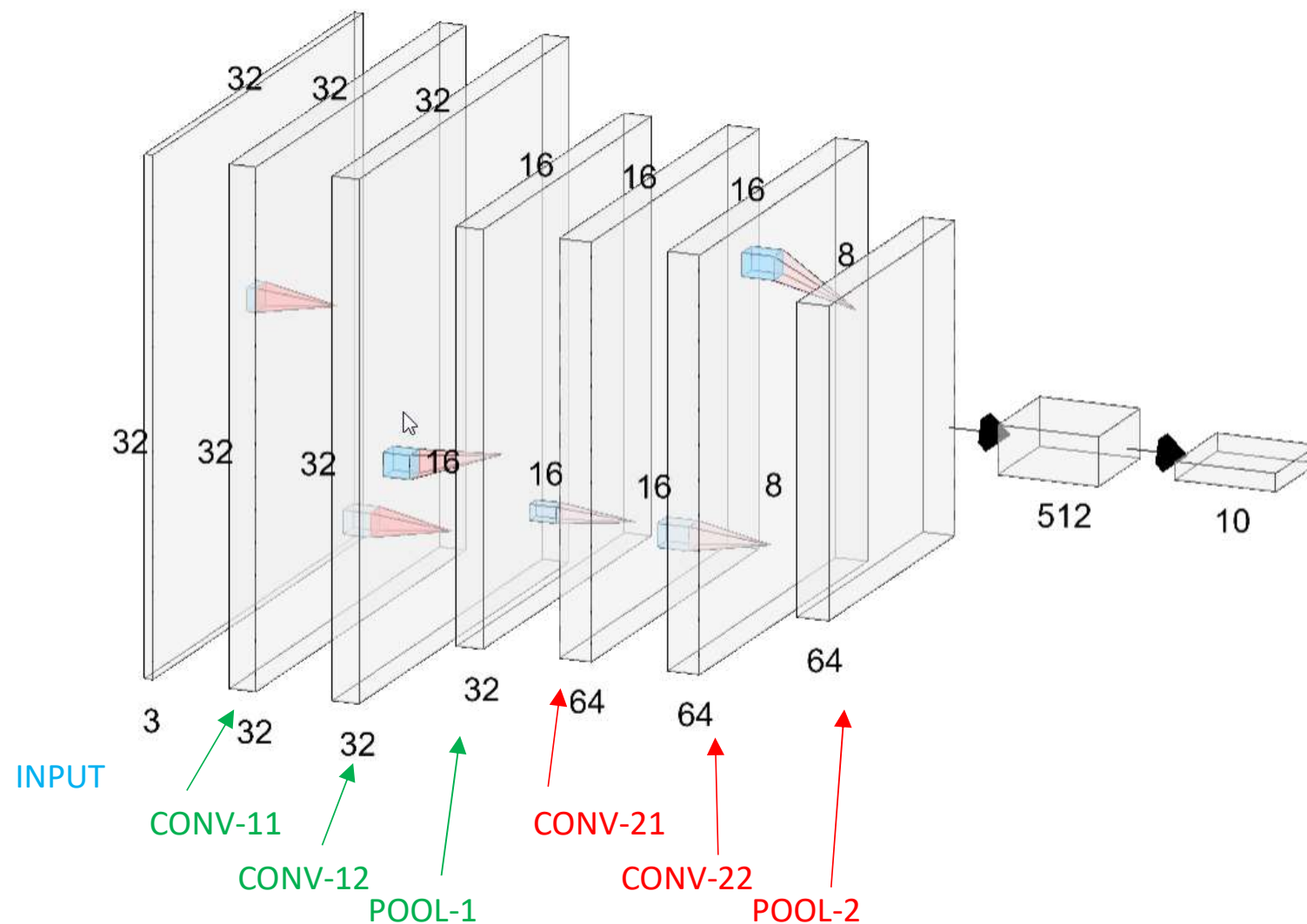
[ref_docs/\[CNN\]VGGNet_1409.1556.pdf](ref_docs/[CNN]VGGNet_1409.1556.pdf)

VGGNet (modified)

The most important feature of VGGNet family are:

- 3x3 convolutions everywhere
- Multiple CONV=>RELU layers stacked *before* a POOL.

We use a simplified version to speed up computation on CIFAR-10 dataset (note: *still very slow on CPU*)



| | Layer | Output Size px | Kernel |
|---------|---------|----------------|-------------|
| | INPUT | 32x32x3 | |
| CONV-11 | CONV | 32x32x32 | 3x3x3, K=32 |
| | ReLu | 32x32x32 | |
| | BN | 32x32x32 | |
| CONV-12 | CONV | 32x32x32 | 3x3x3, K=32 |
| | ReLu | 32x32x32 | |
| | BN | 32x32x32 | |
| POOL-1 | POOL | 16x16x32 | 2x2 |
| | DROPOUT | 16x16x64 | 0.25 |
| CONV-11 | CONV | 16x16x64 | 3x3x3, K=64 |
| | ReLu | 16x16x64 | |
| | BN | 16x16x64 | |
| CONV-12 | CONV | 16x16x64 | 3x3x3, K=64 |
| | ReLu | 16x16x64 | |
| | BN | 16x16x64 | |
| POOL-2 | POOL | 8x8x64 | 2x2 |
| | DROPOUT | 8x8x64 | 0.25 |
| | FC | 512 | |
| | ReLu | 512 | |
| | BN | 512 | |
| | DROPOUT | 512 | 0.5 |
| | FC | 10 | SoftMax |

VGGNet (modified)

```

1 class MiniVGGNet:
2     @staticmethod
3     def build(width, height, depth, classes):
4         # initialize
5         model = Sequential()
6         inputShape = (height, width, depth)
7         chanDim = -1
8
9         # CONV1:
10        # CONV => RELU => CONV => RELU => POOL layer set
11        model.add(Conv2D(32, (3, 3), padding="same",
12            input_shape=inputShape))
13        model.add(Activation("relu"))
14        model.add(BatchNormalization(axis=chanDim))
15        model.add(Conv2D(32, (3, 3), padding="same"))
16        model.add(Activation("relu"))
17        model.add(BatchNormalization(axis=chanDim))
18        model.add(MaxPooling2D(pool_size=(2, 2)))
19        model.add(Dropout(0.25))
20
21        # CONV2:
22        # CONV => RELU => CONV => RELU => POOL layer set
23        model.add(Conv2D(64, (3, 3), padding="same"))
24        model.add(Activation("relu"))
25        model.add(BatchNormalization(axis=chanDim))
26        model.add(Conv2D(64, (3, 3), padding="same"))
27        model.add(Activation("relu"))
28        model.add(BatchNormalization(axis=chanDim))
29        model.add(MaxPooling2D(pool_size=(2, 2)))
30        model.add(Dropout(0.25))
31
32        # FC => RELU
33        model.add(Flatten())
34        model.add(Dense(512))
35        model.add(Activation("relu"))
36        model.add(BatchNormalization())
37        model.add(Dropout(0.5))
38
39        # softmax classifier
40        model.add(Dense(classes))
41        model.add(Activation("softmax"))
42
43        # return the constructed network architecture
44        return model

```

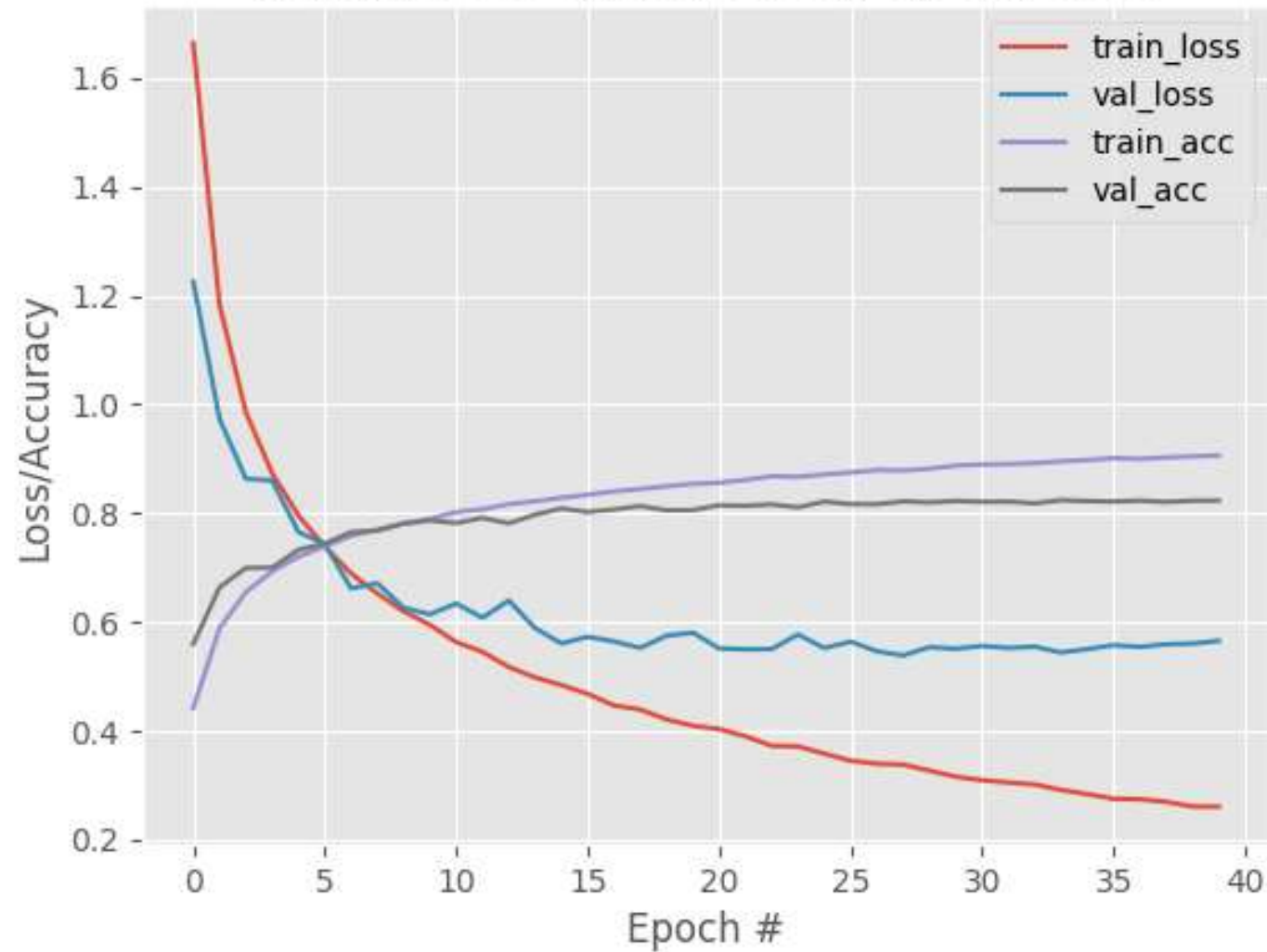


| Layer | Output Size px | Kernel |
|---------|----------------|-------------|
| INPUT | 32x32x3 | |
| CONV | 32x32x32 | 3x3x3, K=32 |
| ReLu | 32x32x32 | |
| BN | 32x32x32 | |
| CONV | 32x32x32 | 3x3x3, K=32 |
| ReLu | 32x32x32 | |
| BN | 32x32x32 | |
| POOL | 16x16x32 | 2x2 |
| DROPOUT | 16x16x64 | 0.25 |
| CONV | 16x16x64 | 3x3x3, K=64 |
| ReLu | 16x16x64 | |
| BN | 16x16x64 | |
| CONV | 16x16x64 | 3x3x3, K=64 |
| ReLu | 16x16x64 | |
| BN | 16x16x64 | |
| POOL | 8x8x64 | 2x2 |
| DROPOUT | 8x8x64 | 0.25 |
| FC | 512 | |
| ReLu | 512 | |
| BN | 512 | |
| DROPOUT | 512 | 0.5 |
| FC | 10 | SoftMax |

VGGNet on CIFAR-10



Training Loss and Accuracy on CIFAR-10



| | precision | recall | f1-score | support |
|------------|-----------|--------|----------|---------|
| airplane | 0.87 | 0.81 | 0.84 | 1000 |
| automobile | 0.93 | 0.90 | 0.92 | 1000 |
| bird | 0.78 | 0.72 | 0.75 | 1000 |
| cat | 0.67 | 0.65 | 0.66 | 1000 |
| deer | 0.76 | 0.86 | 0.80 | 1000 |
| dog | 0.73 | 0.77 | 0.75 | 1000 |
| frog | 0.82 | 0.89 | 0.86 | 1000 |
| horse | 0.91 | 0.83 | 0.87 | 1000 |
| ship | 0.91 | 0.90 | 0.90 | 1000 |
| truck | 0.87 | 0.90 | 0.89 | 1000 |
| accuracy | | | 0.82 | 10000 |

VGGNet on ILSVRC2014

Team VGG was the winner of the ILSVRC (ImageNet Large Scale Visual Recognition Competition) 2014 in the classification & localization (task 2a)

Classification+localization

Task 2a: Classification+localization with provided training data

Classification+localization with provided training data: Ordered by localization error

| Team name | Entry description | Loc error |
|------------|--|-----------|
| VGG | a combination of multiple ConvNets (by averaging) | 0.25 |
| VGG | a combination of multiple ConvNets (fusion weights learnt on the validation set) | 0.25 |
| VGG | a combination of multiple ConvNets, including a net trained on images of different size (fusion done by averaging); detected boxes were not updated | 0.25 |
| VGG | a combination of multiple ConvNets, including a net trained on images of different size (fusion weights learnt on the validation set); detected boxes were not updated | 0.25 |
| GoogLeNet | Model with localization ~26% top5 val error. | 0.26 |
| GoogLeNet | Model with localization ~26% top5 val error, limiting number of classes. | 0.26 |
| VGG | a single ConvNet (13 convolutional and 3 fully-con | 0.26 |

n.4 GPUs
n.3 weeks

VGG

Karen Simonyan, University of Oxford
Andrew Zisserman, *University of Oxford*

We have used three ConvNet architectures with the following weight layer configurations:

1. ten 3x3 convolutional layers, three 1x1 convolutional layers, and three fully-connected layers - 16 weight layers in total;
2. thirteen 3x3 convolutional layers and three fully-connected layers - 16 weight layers in total;
3. sixteen 3x3 convolutional layers and three fully-connected layers - 19 weight layers in total.

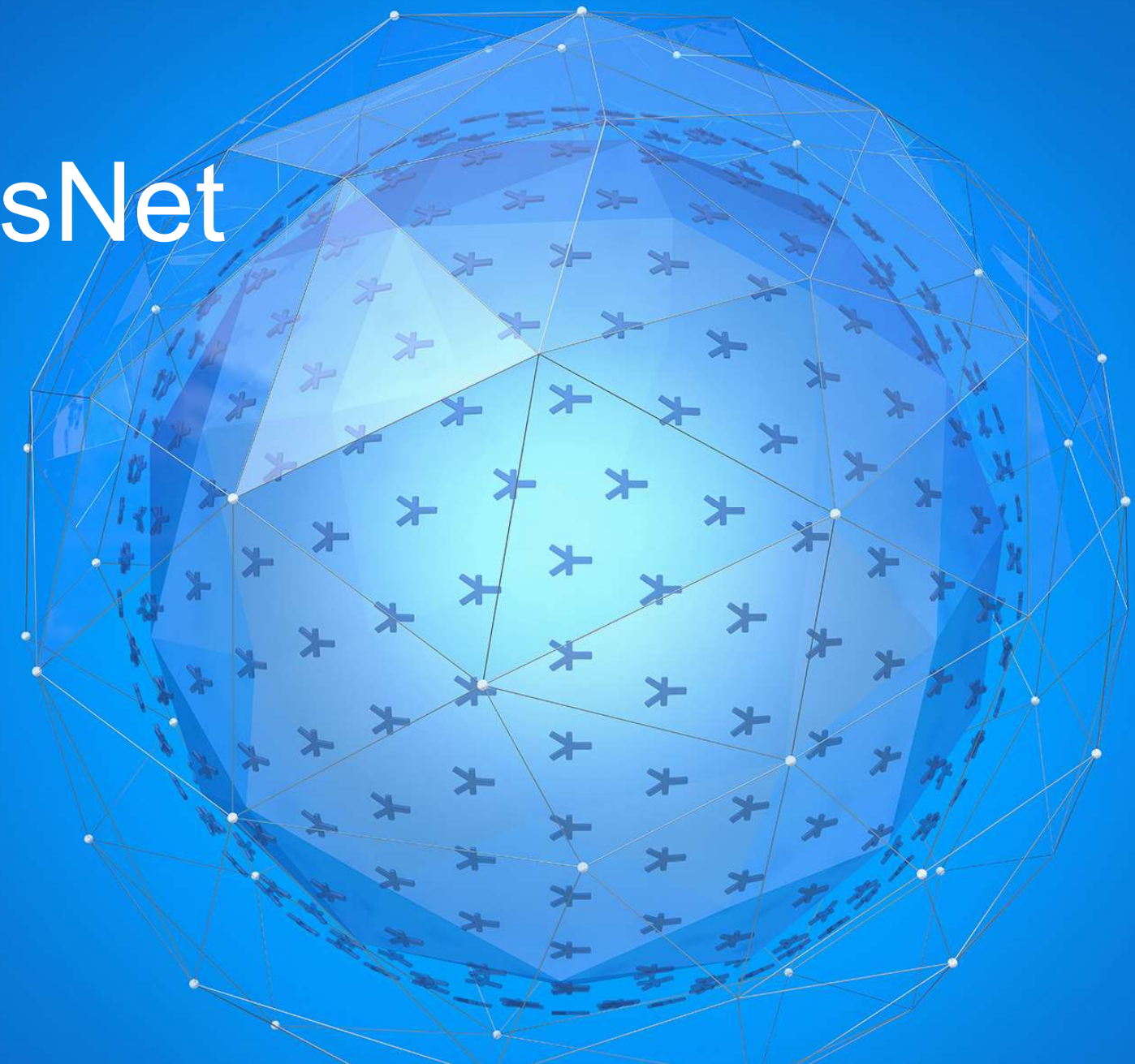
{...}

Our implementation is derived from the Caffe toolbox, but contains a number of significant modifications, including parallel training on multiple GPUs installed in a single system. **Training a single ConvNet on 4 NVIDIA Titan GPUs took from 2 to 3 weeks (depending on the ConvNet configuration).**

<http://www.image-net.org/challenges/LSVRC/2014/results>

<https://medium.com/coinmonks/paper-review-of-vggnet-1st-runner-up-of-ilsvlc-2014-image-classification-d0235543a11>

CNN: InceptionNet and ResNet



GoogLeNet a.k.a InceptionNet (2014)

Google research team started the idea of building networks with *multi-level feature extraction*. For this purpose a special “core module” was created.

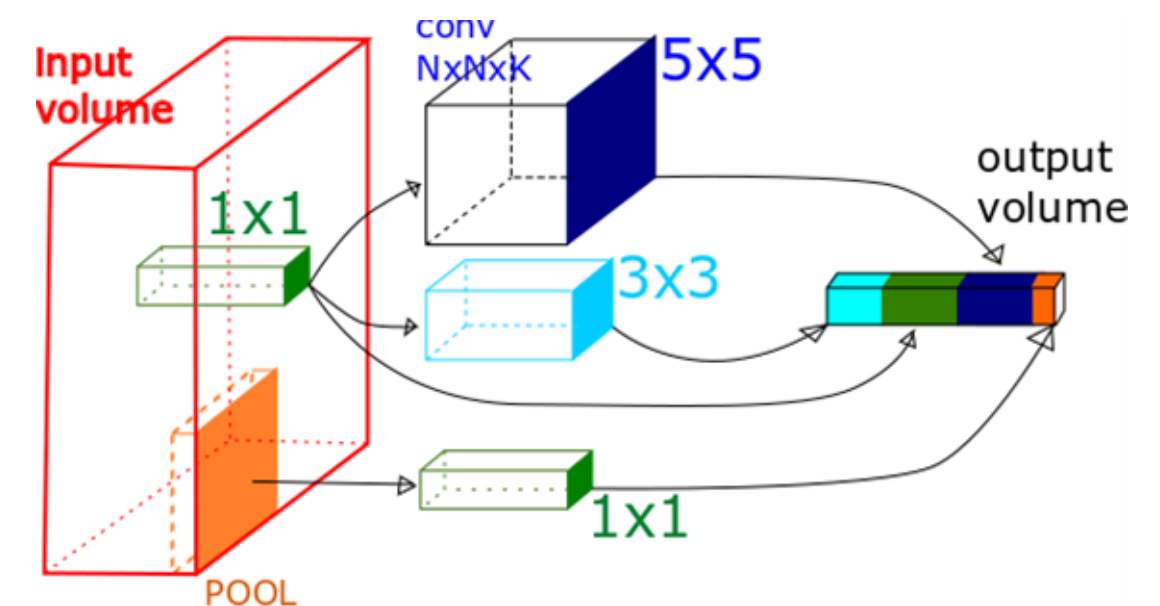
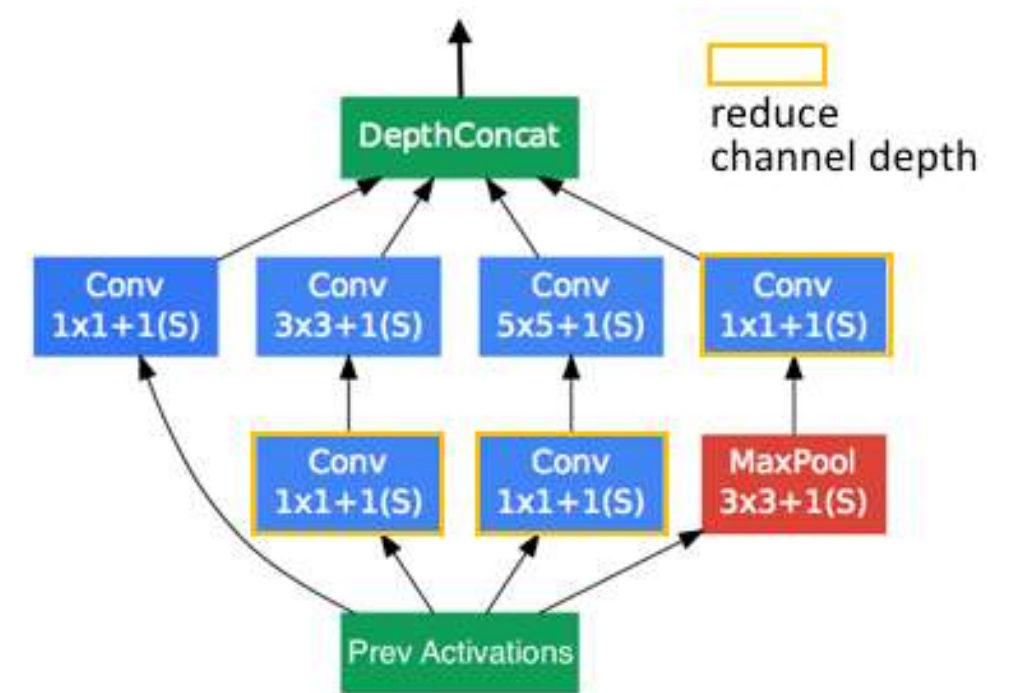
The *Inception module* is an example of a “graph” network which is not “sequential” anymore since multiple path are taken to compute the module output. This is also called *Network-in-Network* architecture.

We have four branches:

- 1st branch: a series of 1x1 convolutions to learn **local features** (think FC)
- 2nd branch: **reduce** the volume with 1x1 conv (*num3x3Reduce conv*) and **expand** with 3x3 conv (*num3x3 conv*)
- 3rd branch: same **reduce&expand**, but with a 5x5 conv (*num5x5Reduce* and *num5x5*)
- 4th branch: **pool projection** branch. A MaxPool reduction with a 1x1 conv. (NOTE: it was added for historical/empirical reasons since MaxPool was very common...)

A new tool: **DepthConcat**

the *volumes* from all the branches are “concatenated” on the depth axis to form the output volume result.

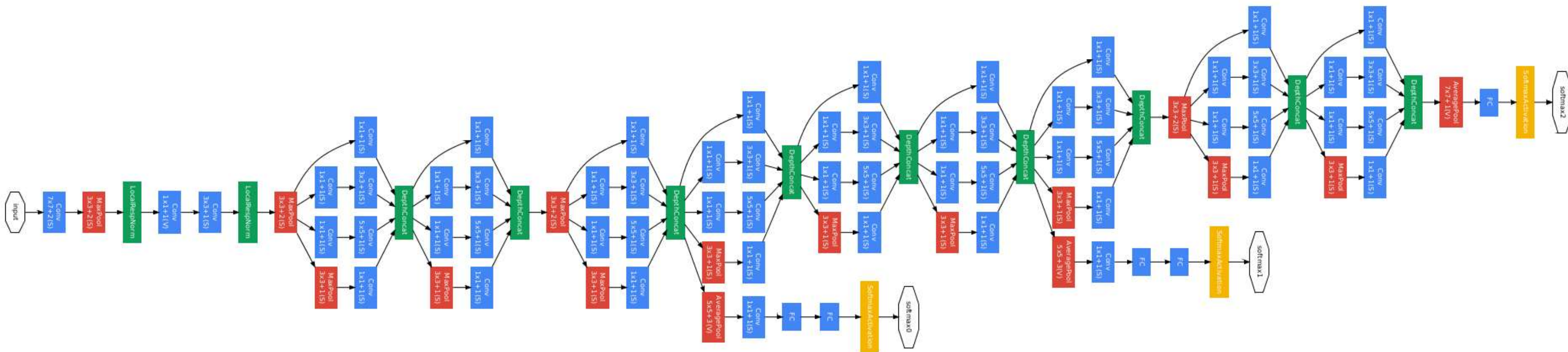


GoogLeNet a.k.a InceptionNet (2014)

The network used a CNN inspired by LeNet but implemented a novel, the inception module.

This module is based on several very small convolutions in order to **drastically reduce** the number of parameters.

Their architecture consisted of a **22 layer** deep CNN but reduced the number of parameters **from 60 million (AlexNet) to 4 million.**



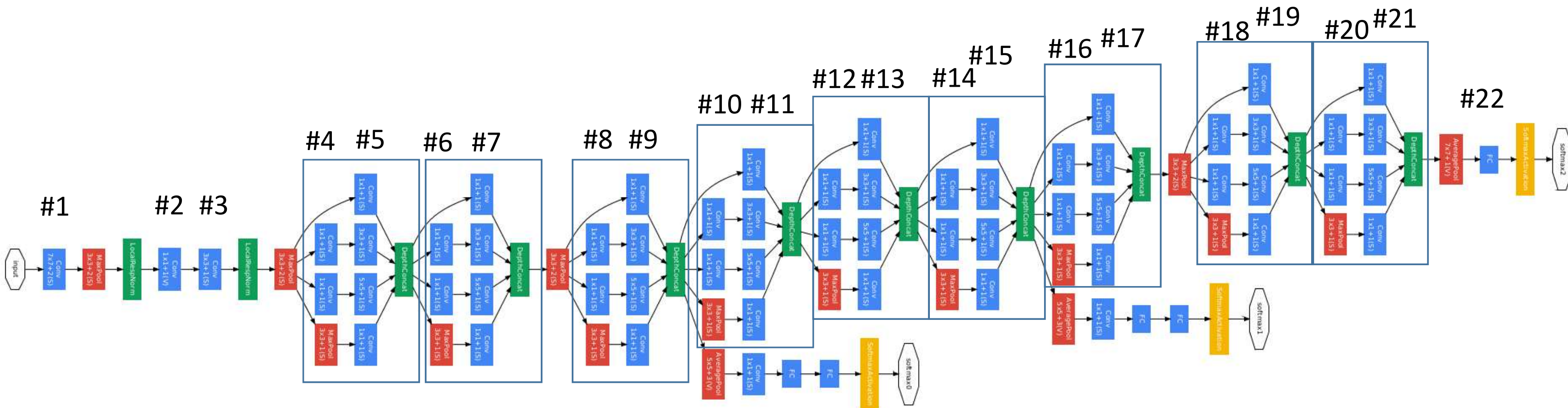
<https://sites.google.com/site/aidysft/objectdetection/recent-list-items>

GoogLeNet a.k.a InceptionNet (2014)

The network used a CNN inspired by LeNet but implemented a novel, the inception module.

This module is based on several very small convolutions in order to **drastically reduce** the number of parameters.

Their architecture consisted of a **22 layer** deep CNN but reduced the number of parameters **from 60 million (AlexNet) to 4 million.**



Note: count layers only if they contains weights...

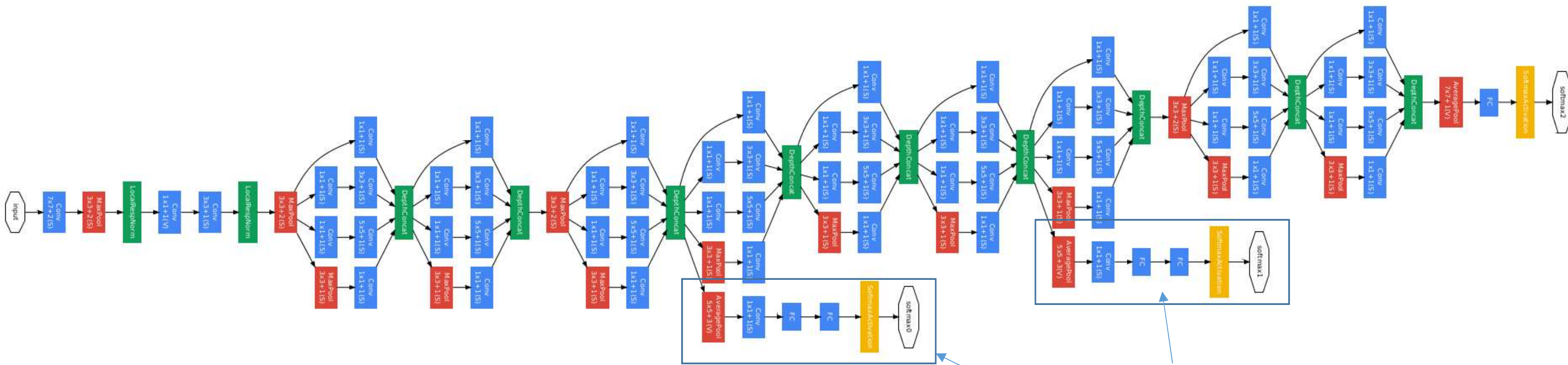
<https://sites.google.com/site/aidysft/objectdetection/recent-list-items>

GoogLeNet a.k.a InceptionNet (2014)

The network used a CNN inspired by LeNet but implemented a novel, the inception module.

This module is based on several very small convolutions in order to **drastically reduce** the number of parameters.

Their architecture consisted of a **22 layer** deep CNN but reduced the number of parameters **from 60 million (AlexNet) to 4 million.**



These are intermediate results...

training time: few *days* by Andrej Karpathy ...

<https://sites.google.com/site/aidysft/objectdetection/recent-list-items>

GoogLeNet (modified)

```

1 class GoogLeNet:
2     @staticmethod
3     def conv_module(x, K, kX, kY, stride, chanDim, padding="same"):
4         # define a CONV => BN => RELU pattern
5         x = Conv2D(K, (kX, kY), strides=stride, padding=padding)(x)
6         x = BatchNormalization(axis=chanDim)(x)
7         x = Activation("relu")(x)
8         return x
9
10    @staticmethod
11    def inception_module(x, numK1x1, numK3x3, chanDim):
12        # two CONV modules, concatenate across depth axis
13        conv_1x1 = GoogLeNet.conv_module(x, numK1x1, 1, 1, (1, 1), chanDim)
14        conv_3x3 = GoogLeNet.conv_module(x, numK3x3, 3, 3, (1, 1), chanDim)
15        x = concatenate([conv_1x1, conv_3x3], axis=chanDim)
16        return x
17
18    @staticmethod
19    def downsample_module(x, K, chanDim):
20        # CONV module and POOL, concatenate across depth axis
21        conv_3x3 = GoogLeNet.conv_module(x, K, 3, 3, (2, 2), chanDim, padding="valid")
22        pool = MaxPooling2D((3, 3), strides=(2, 2))(x)
23        x = concatenate([conv_3x3, pool], axis=chanDim)
24        return x

```

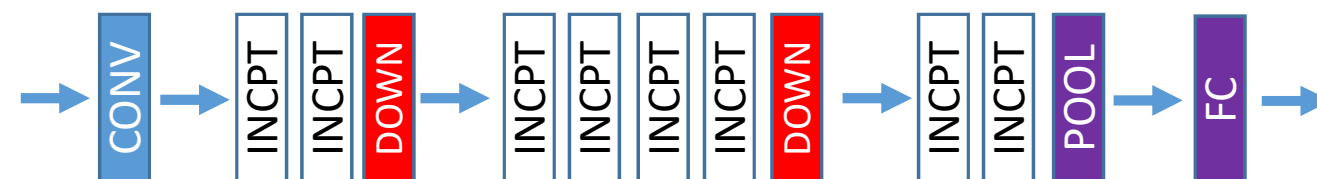
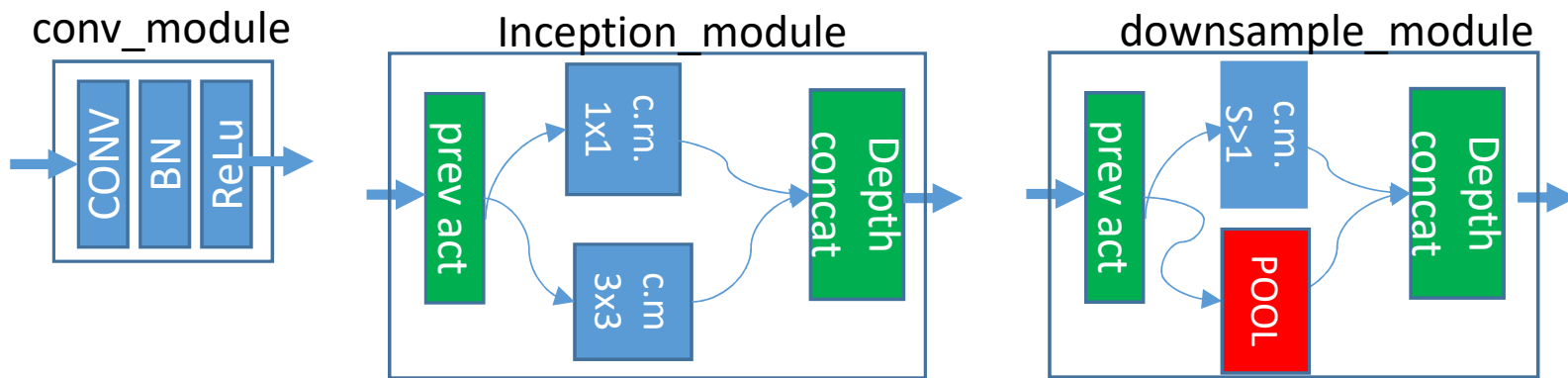


IT'S
DEMO
TIME!

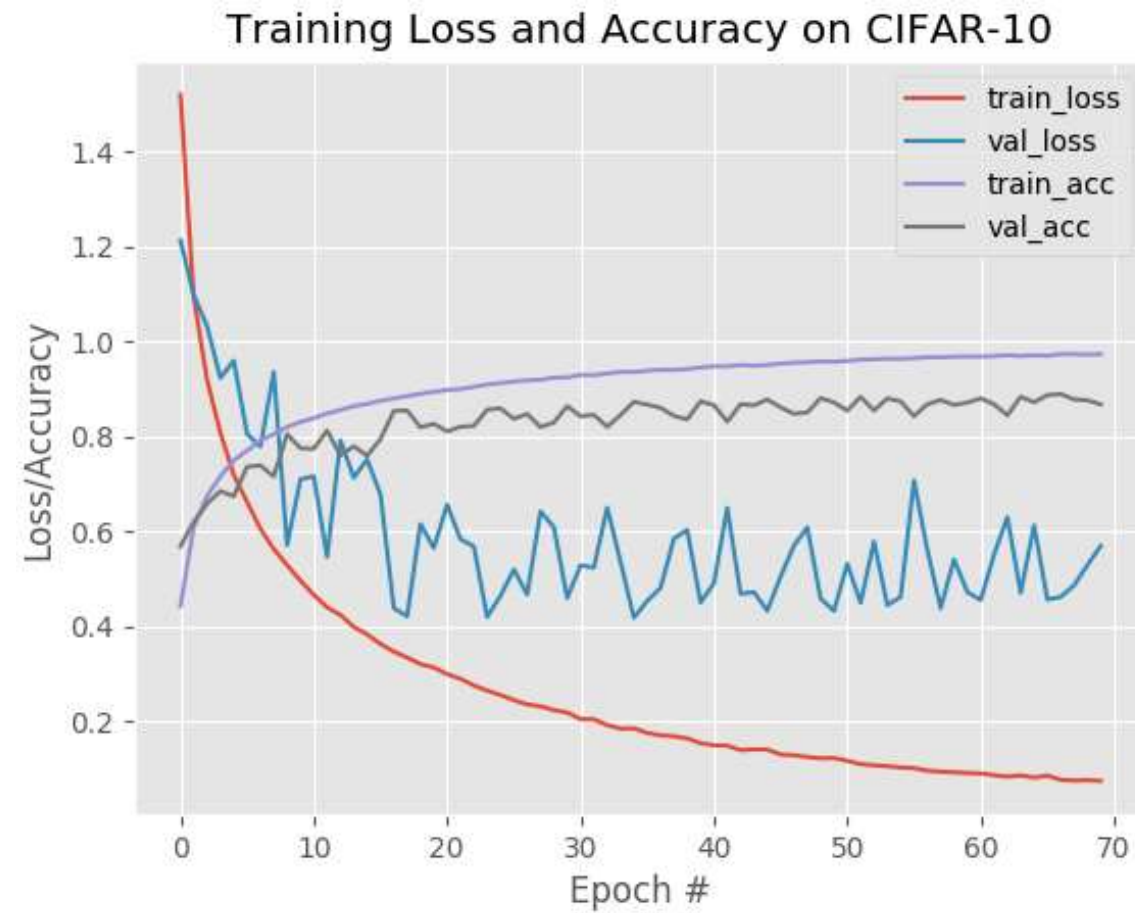
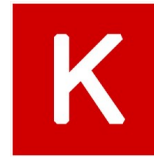
```

1 def build(width, height, depth, classes):
2     # initialize the input shape to be "channels last" and the
3     # channels dimension itself
4     inputShape = (height, width, depth)
5     chanDim = -1
6
7     # if we are using "channels first", update the input shape
8     # and channels dimension
9     if K.image_data_format() == "channels_first":
10        inputShape = (depth, height, width)
11        chanDim = 1
12
13    # define the model input and first CONV module
14    inputs = Input(shape=inputShape)
15    x = GoogLeNet.conv_module(inputs, 96, 3, 3, (1, 1), chanDim)
16
17    # two Inception modules followed by a downsample module
18    x = GoogLeNet.inception_module(x, 32, 32, chanDim)
19    x = GoogLeNet.inception_module(x, 32, 48, chanDim)
20    x = GoogLeNet.downsample_module(x, 80, chanDim)
21
22    # four Inception modules followed by a downsample module
23    x = GoogLeNet.inception_module(x, 112, 48, chanDim)
24    x = GoogLeNet.inception_module(x, 96, 64, chanDim)
25    x = GoogLeNet.inception_module(x, 80, 80, chanDim)
26    x = GoogLeNet.inception_module(x, 48, 96, chanDim)
27    x = GoogLeNet.downsample_module(x, 96, chanDim)
28
29    # two Inception modules followed by global POOL and dropout
30    x = GoogLeNet.inception_module(x, 176, 160, chanDim)
31    x = GoogLeNet.inception_module(x, 176, 160, chanDim)
32    x = AveragePooling2D((7, 7))(x)
33    x = Dropout(0.5)(x)
34
35    # softmax classifier
36    x = Flatten()(x)
37    x = Dense(classes)(x)
38    x = Activation("softmax")(x)
39
40    # create the model
41    model = Model(inputs, x, name="googlenet")
42
43    return model

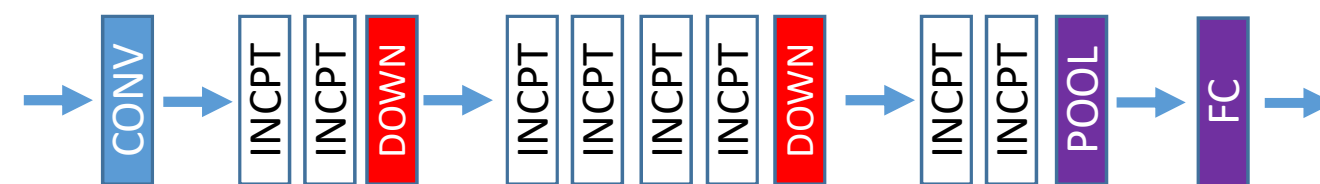
```



GoogLeNet (modified)



| | precision | recall | f1-score | support |
|------------|-----------|--------|----------|---------|
| airplane | 0.92 | 0.86 | 0.89 | 1000 |
| automobile | 0.94 | 0.96 | 0.95 | 1000 |
| bird | 0.75 | 0.88 | 0.81 | 1000 |
| cat | 0.66 | 0.88 | 0.75 | 1000 |
| deer | 0.97 | 0.70 | 0.81 | 1000 |
| dog | 0.85 | 0.80 | 0.83 | 1000 |
| frog | 0.98 | 0.79 | 0.88 | 1000 |
| horse | 0.91 | 0.92 | 0.91 | 1000 |
| ship | 0.90 | 0.96 | 0.93 | 1000 |
| truck | 0.94 | 0.93 | 0.93 | 1000 |
| accuracy | | | 0.87 | 10000 |



GoogLeNet a.k.a InceptionNet (2014)

Team Google was the winner of the ILSVRC (ImageNet Large Scale Visual Recognition Competition) 2014 in the Object Detection (task 1b)

Task 1b: Object detection with additional training data

Object detection with additional training data: Ordered by number of categories won

| Team name | Entry description | Description of outside data used |
|------------------------|---|---|
| GoogLeNet | Ensemble of detection models. Validation is 44.5% mAP | Pretraining on ILSVRC12 classification data. |
| <u>CUHK DeepID-Net</u> | Combine multiple models described in the abstract without contextual modeling | ImageNet classification and localization data |
| Deep Insight | Combination of three detection models | Three CNNs from classification for initialization |
| <u>UvA-Eurovision</u> | Deep learning with outside data | ImageNet 1000 |
| <u>Berkeley Vision</u> | R-CNN baseline | The CNN was pre-trained on the ILSVRC 2013 CLS dataset. |

less parameters

GoogLeNet

Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Drago Anguelov, Dumitru Erhan, Andrew Rabinovich

We explore an improved convolutional neural network architecture which combines the multi-scale idea with intuitions gained from the Hebbian principle. Additional dimension reduction layers based on embedding learning intuition allow us to increase both the depth and the width of the network significantly without incurring significant computational overhead. Combining these ideas allow for increasing the number of parameters in convolutional layers significantly while **cutting the total number of parameters and resulting in improved generalization**. Various incarnations of this architecture are trained for and applied at various scales and the resulting scores are averaged for each image.

<http://www.image-net.org/challenges/LSVRC/2014/results>



Going deeper with convolutions

[ref_docs\\[CNN\]Inception-v1_1409.4842.pdf](#)



Rethinking the Inception Architecture for Computer Vision

[ref_docs\\[CNN\]Inception-v2_1512.00567.pdf](#)

GoogLeNet becomes InceptionNet (2014)

CNN names are often selected with references to their main functionality.

Originally Google's research team decided to make a reference to the seminal (*goog*)LeNet arch, but in the original paper for *Inception Net* they referred to the movie as follow:

*In this paper, we will focus on an efficient deep neural network architecture for computer vision, **codenamed Inception**, which derives its name from the Network in network paper by Lin et al [12] in conjunction with the famous “we need to go deeper” internet meme [1]. In our case, the word “deep” is used in two different meanings: first of all, in the sense that we introduce a new level of organization in the form of the “Inception module” and also in the more direct sense of increased network depth.*

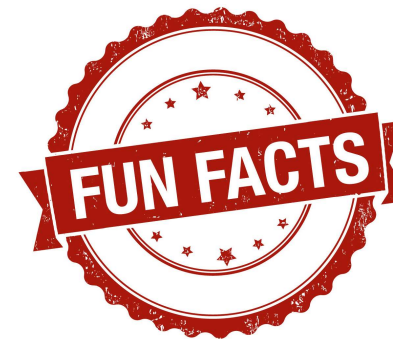
References

[1] Know your meme: We need to go deeper. <http://knowyourmeme.com/memes/we-need-to-go-deeper>. Accessed: 2014-09-15.



Going deeper with convolutions

[ref_docs\\[CNN\]Inception-v1_1409.4842.pdf](#)



Going deeper with convolutions

Christian Szegedy
Google Inc.

Wei Liu
University of North Carolina, Chapel Hill

Yangqing Jia
Google Inc.

Pierre Sermanet
Google Inc.

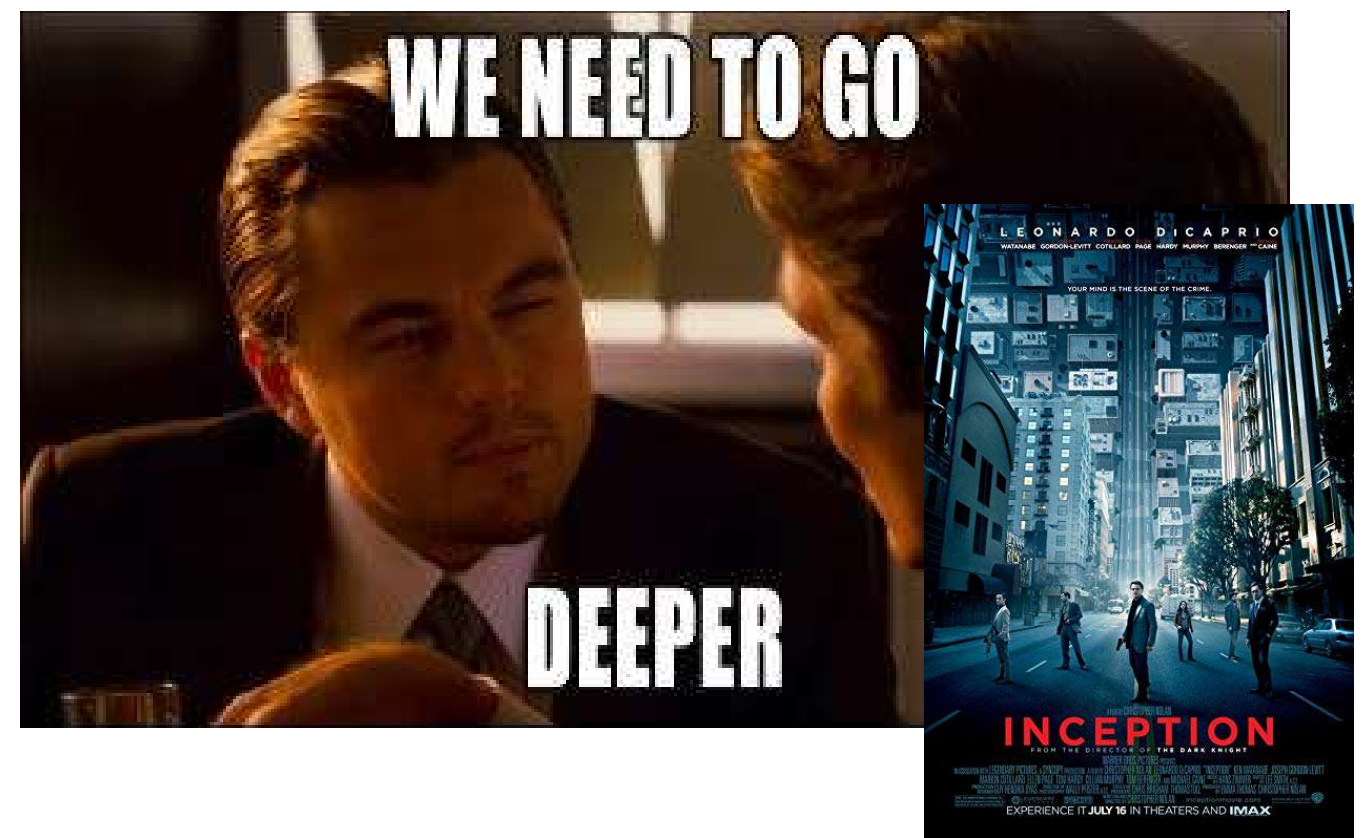
Scott Reed
University of Michigan

Dragomir Anguelov
Google Inc.

Dumitru Erhan
Google Inc.

Vincent Vanhoucke
Google Inc.

Andrew Rabinovich
Google Inc.



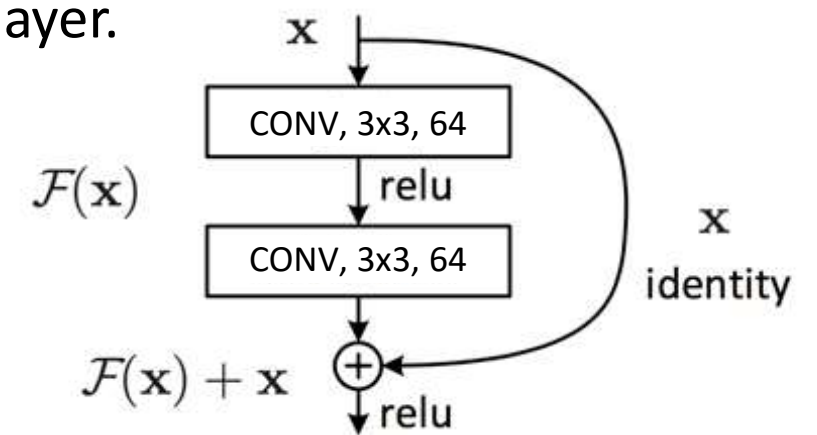
ResNet(2015) a.k.a ResidualNet

Very *deep* networks with a sequential model suffer from one problem: vanishing gradient.

(https://en.wikipedia.org/wiki/Vanishing_gradient_problem). Backpropagation di SGD becomes more and more inaccurate and adding layers does not improve network performance.

ResNet solves this problem with an “*identity mapping*” layer (also called “*linear shortcut*”) which takes part of the previous activation layer and joins with the current layer right *before* the final ACT/ReLU layer.

This core architecture is than stacked over and over to reach very deep networks, ResNet50 is a de-facto reference network for benchmarking performance on CNN.

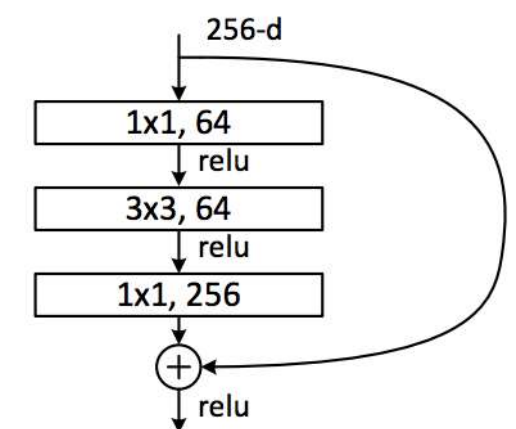


Original ResNet “residual block”

ResNet advantage: smaller model footprint

The are only two POOL layers (at the beginning and at the end) but the network volume is controlled by the usage of *convolutions with stride>1* instead of pooling.

Variant “**Bottleneck**”: improvement to the original block that leverages two CONV layers with a smaller volume (1/4 of the input depth) and a one final CONV layer with the same depth as the input.



Variant: “Bottleneck” residual block



Deep Residual Learning for Image Recognition
[ref_docs\\[CNN\]resnet_1512.03385.pdf](ref_docs/[CNN]resnet_1512.03385.pdf)



Identity Mappings in Deep Residual Networks
[ref_docs\\[CNN\]resnet_modified_1603.05027.pdf](ref_docs/[CNN]resnet_modified_1603.05027.pdf)

ResNet (modified)

ResNet_bottleneck_block

```

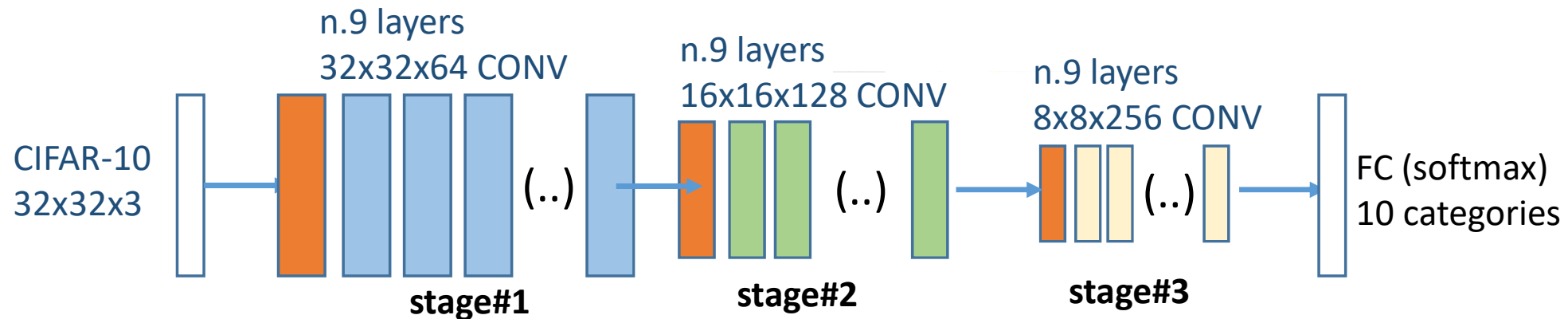
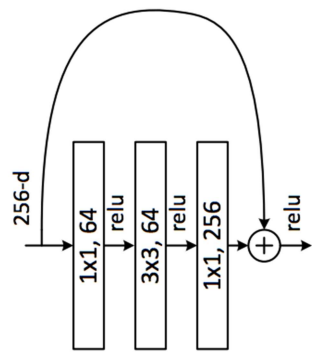
1 class ResNet:
2     @staticmethod
3     def residual_module(data, K, stride, chanDim, red=False,
4         reg=0.0001, bnEps=2e-5, bnMom=0.9):
5         # the shortcut branch of the ResNet module should be
6         # initialize as the input (identity) data
7         shortcut = data
8
9         # ResNet module: first BN=>ReLu=>1x1 CONVs
10        bn1 = BatchNormalization(axis=chanDim, epsilon=bnEps,
11            momentum=bnMom)(data)
12        act1 = Activation("relu")(bn1)
13        conv1 = Conv2D(int(K * 0.25), (1, 1), use_bias=False,
14            kernel_regularizer=l2(reg))(act1)
15
16        # ResNet module: second BN=>ReLu=>3x3 CONVs
17        bn2 = BatchNormalization(axis=chanDim, epsilon=bnEps,
18            momentum=bnMom)(conv1)
19        act2 = Activation("relu")(bn2)
20        conv2 = Conv2D(int(K * 0.25), (3, 3), strides=stride,
21            padding="same", use_bias=False,
22            kernel_regularizer=l2(reg))(act2)
23
24        # ResNet module: third BN=>ReLu=> 1x1 CONVs
25        bn3 = BatchNormalization(axis=chanDim, epsilon=bnEps,
26            momentum=bnMom)(conv2)
27        act3 = Activation("relu")(bn3)
28        conv3 = Conv2D(K, (1, 1), use_bias=False,
29            kernel_regularizer=l2(reg))(act3)
30
31        # if needed to reduce size use a CONV layer (do not pool)
32        if red:
33            shortcut = Conv2D(K, (1, 1), strides=stride,
34                use_bias=False, kernel_regularizer=l2(reg))(act1)
35
36        # SUM: add together the shortcut and the final CONV
37        x = add([conv3, shortcut])
38
39        return x
    
```



ResNet_build_loop

```

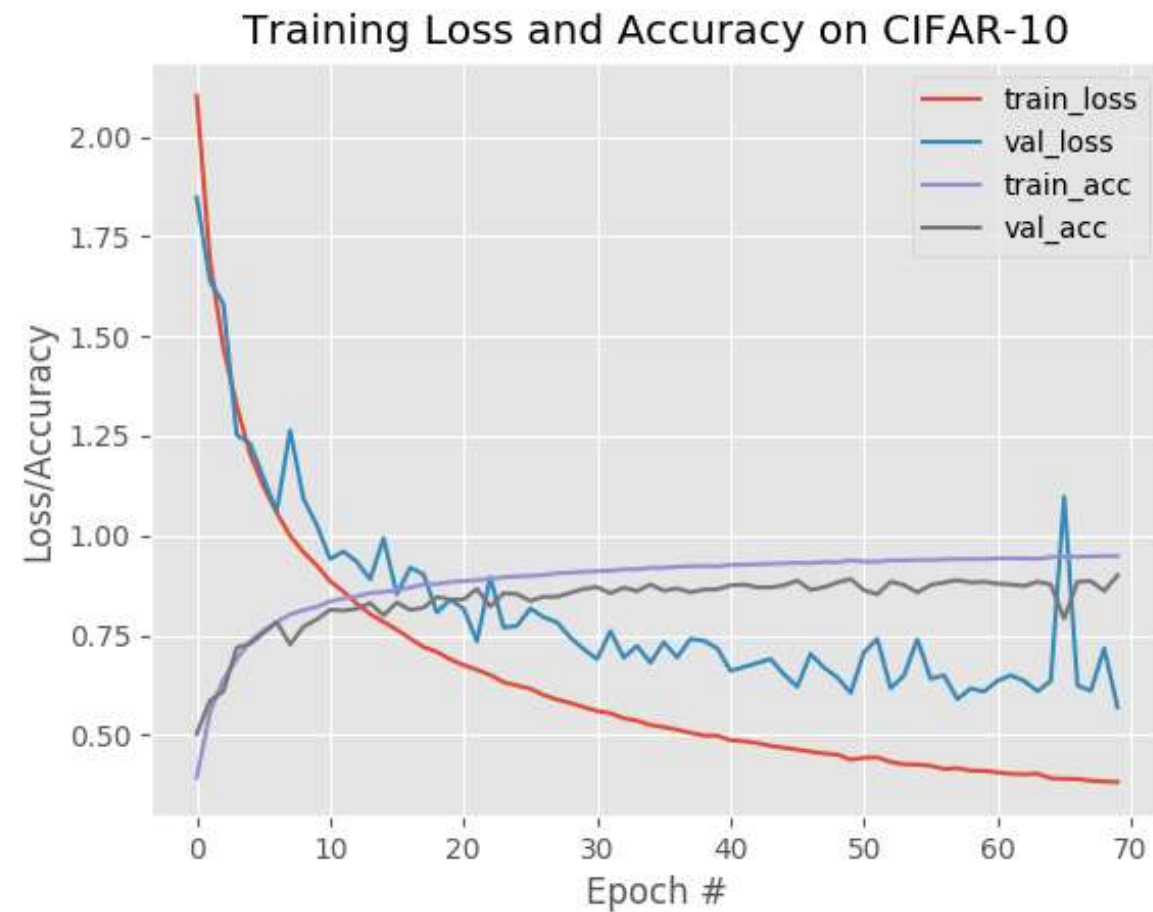
1 @staticmethod
2 def build(width, height, depth, classes, stages, filters,
3     reg=0.0001, bnEps=2e-5, bnMom=0.9):
4     # input shape for "channel first/last"
5     inputShape = (height, width, depth)
6     chanDim = -1
7
8     # set the input and apply BN
9     inputs = Input(shape=inputShape)
10    x = BatchNormalization(axis=chanDim, epsilon=bnEps,
11        momentum=bnMom)(inputs)
12
13    # loop over the number of stages
14    for i in range(0, len(stages)):
15        # stride is (1,1) only for the first (input) stage
16        stride = (1, 1) if i == 0 else (2, 2)
17
18        x = ResNet.residual_module(x, filters[i], stride,
19            chanDim, red=True, bnEps=bnEps, bnMom=bnMom)
20
21        # loop over the number of layers in the stage
22        for j in range(0, stages[i] - 1):
23            # apply a ResNet module
24            x = ResNet.residual_module(x, filters[i],
25                (1, 1), chanDim, red=False, bnEps=bnEps, bnMom=bnMom)
26
27        # apply BN => ACT => POOL
28        x = BatchNormalization(axis=chanDim, epsilon=bnEps,
29            momentum=bnMom)(x)
30        x = Activation("relu")(x)
31        x = AveragePooling2D((8, 8))(x)
32
33        # softmax classifier
34        x = Flatten()(x)
35        x = Dense(classes, kernel_regularizer=l2(reg))(x)
36        x = Activation("softmax")(x)
37
38        # create the model
39        model = Model(inputs, x, name="resnet")
    
```



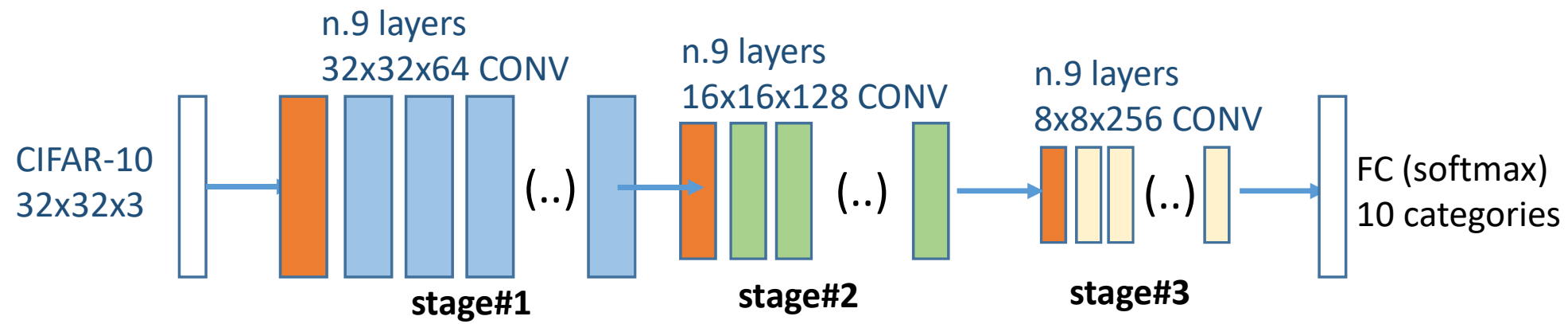
```

# SYNTAX: ResNet.build(width, height, depth, classes, stages, filters, reg, bnEps, bnMom)
model = ResNet.build(32, 32, 3, 10, (9, 9, 9), (64, 128, 256), reg=0.0005)
    
```

ResNet(modified)



| | precision | recall | f1-score | support |
|------------|-----------|--------|----------|---------|
| airplane | 0.92 | 0.89 | 0.91 | 1000 |
| automobile | 0.92 | 0.97 | 0.94 | 1000 |
| bird | 0.89 | 0.88 | 0.88 | 1000 |
| cat | 0.83 | 0.80 | 0.81 | 1000 |
| deer | 0.88 | 0.91 | 0.90 | 1000 |
| dog | 0.92 | 0.79 | 0.85 | 1000 |
| frog | 0.91 | 0.94 | 0.92 | 1000 |
| horse | 0.89 | 0.95 | 0.92 | 1000 |
| ship | 0.91 | 0.96 | 0.93 | 1000 |
| truck | 0.93 | 0.92 | 0.93 | 1000 |
| accuracy | | | 0.90 | 10000 |



ResNet(2015)

Was ResNet Successful?

- Won **1st place** in the **ILSVRC 2015** classification competition with top-5 error rate of 3.57%
- Won the **1st place** in **ILSVRC and COCO 2015** competition in ImageNet Detection, ImageNet localization, Coco detection and Coco segmentation.
- Replacing VGG-16 layers in Faster R-CNN with ResNet-101. They observed a relative improvements of 28%
- Efficiently trained networks with *100 layers* and *1000 layers* also



MSRA (ResNet)

Kaiming He, Xiangyu Zhang, Shaoqing Ren
Jian Sun, *Microsoft Research*

We train neural networks with depth of over 150 layers. We propose a "deep residual learning" framework [a] that eases the optimization and convergence of extremely deep networks. Our "deep residual nets" enjoy accuracy gains when the networks are substantially deeper than those used previously. Such accuracy gains are not witnessed for many common networks when going deeper.

Our localization and detection systems are based on deep residual nets and the "Faster R-CNN" system in our NIPS paper [b]. The extremely deep representations generalize well, and greatly improve the results of the Faster R-CNN system. Furthermore, we show that the region proposal network (RPN) in [b] is a generic framework and performs excellent for localization.

<https://towardsdatascience.com/review-resnet-winner-of-ilsvrc-2015-image-classification-localization-detection-e39402bfa5d8>

<http://image-net.org/challenges/LSVRC/2015/results>

<https://medium.com/@14prakash/understanding-and-implementing-architectures-of-resnet-and-resnext-for-state-of-the-art-image-cf51669e1624>

CNN: classification & object detection



Classical object detection algorithms

Before CNN became popular few algorithms were common for feature detection, for example “*histogram of oriented gradients (HOG)*” used with “*scalar vector machine (SVM)*” for classification.

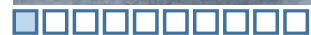
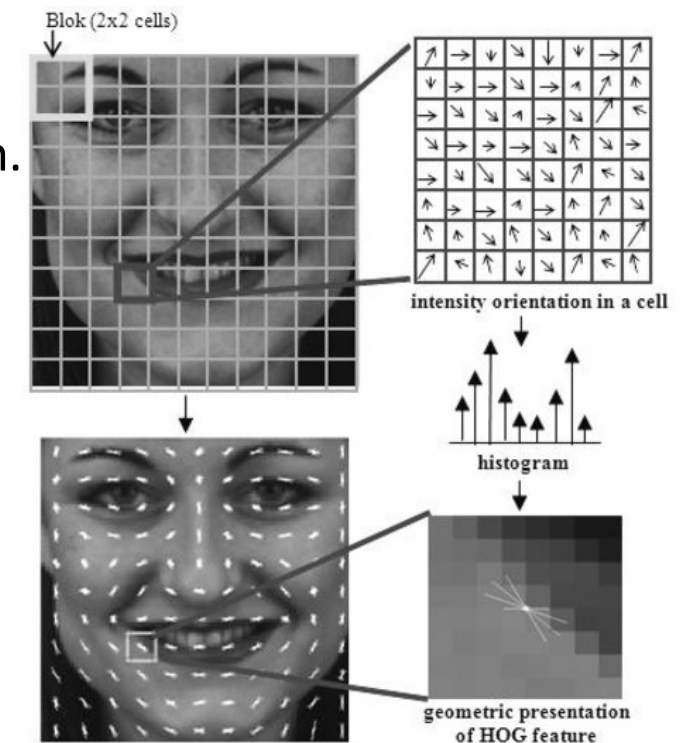
https://en.wikipedia.org/wiki/Histogram_of_oriented_gradients
<https://www.learnopencv.com/histogram-of-oriented-gradients/>

See also “*Haar Cascade Classifier*”

<http://www.willberger.org/cascade-haar-explained/>

These algorithms are all based on “sliding window” and “image pyramid”.

Given an input image we extract multiple “sliding” sub-pictures and we perform image classification based on the features extracted from it. Multi-resolution (image pyramid) is needed to support multiple size detection and it is a penalty for the complexity of the whole algorithm.



Classical object detection algorithms

Before CNN became popular few algorithms were common for feature detection, for example “*histogram of oriented gradients (HOG)*” used with “*scalar vector machine (SVM)*” for classification.

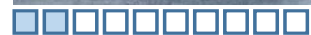
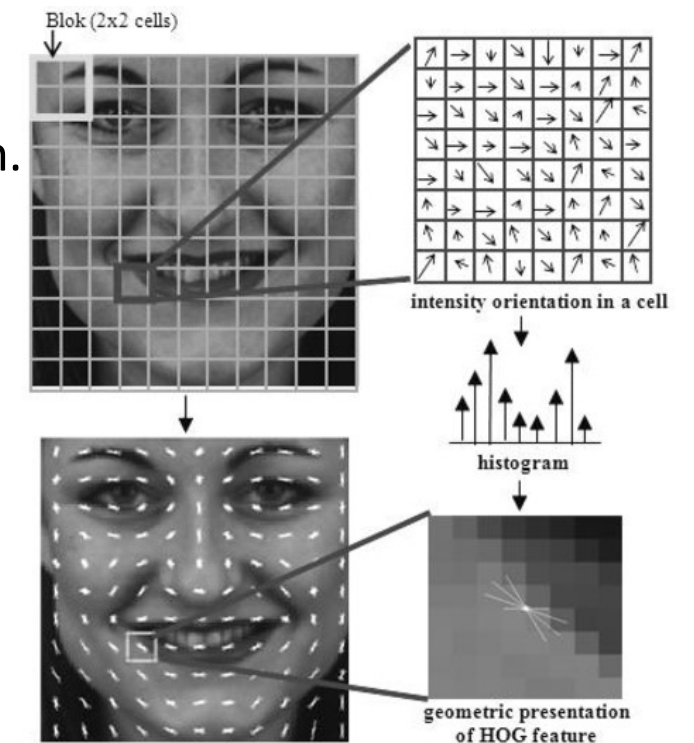
https://en.wikipedia.org/wiki/Histogram_of_oriented_gradients
<https://www.learnopencv.com/histogram-of-oriented-gradients/>

See also “*Haar Cascade Classifier*”

<http://www.willberger.org/cascade-haar-explained/>

These algorithms are all based on “sliding window” and “image pyramid”.

Given an input image we extract multiple “sliding” sub-pictures and we perform image classification based on the features extracted from it. Multi-resolution (image pyramid) is needed to support multiple size detection and it is a penalty for the complexity of the whole algorithm.



Classical object detection algorithms

Before CNN became popular few algorithms were common for feature detection, for example “*histogram of oriented gradients (HOG)*” used with “*scalar vector machine (SVM)*” for classification.

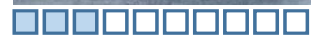
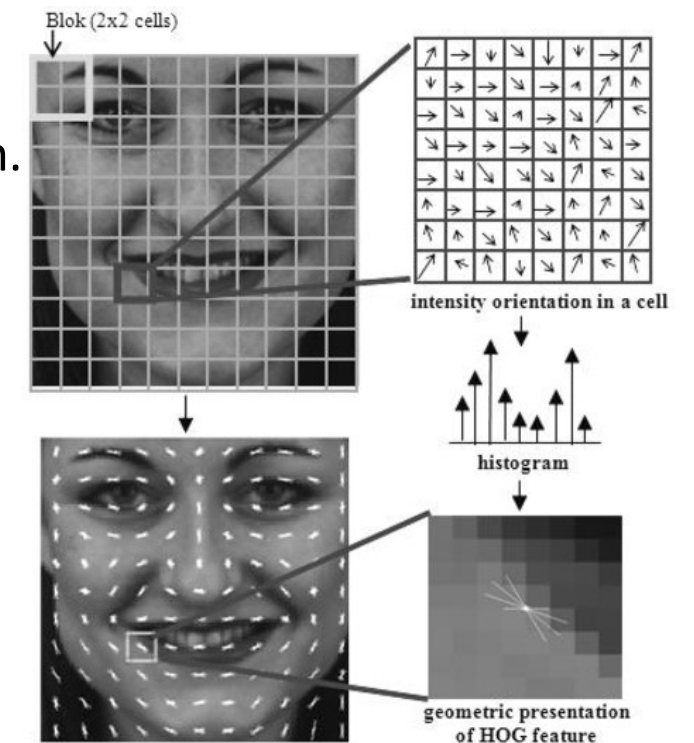
https://en.wikipedia.org/wiki/Histogram_of_oriented_gradients
<https://www.learnopencv.com/histogram-of-oriented-gradients/>

See also “*Haar Cascade Classifier*”

<http://www.willberger.org/cascade-haar-explained/>

These algorithms are all based on “sliding window” and “image pyramid”.

Given an input image we extract multiple “sliding” sub-pictures and we perform image classification based on the features extracted from it. Multi-resolution (image pyramid) is needed to support multiple size detection and it is a penalty for the complexity of the whole algorithm.



Classical object detection algorithms

Before CNN became popular few algorithms were common for feature detection, for example “*histogram of oriented gradients (HOG)*” used with “*scalar vector machine (SVM)*” for classification.

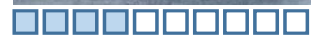
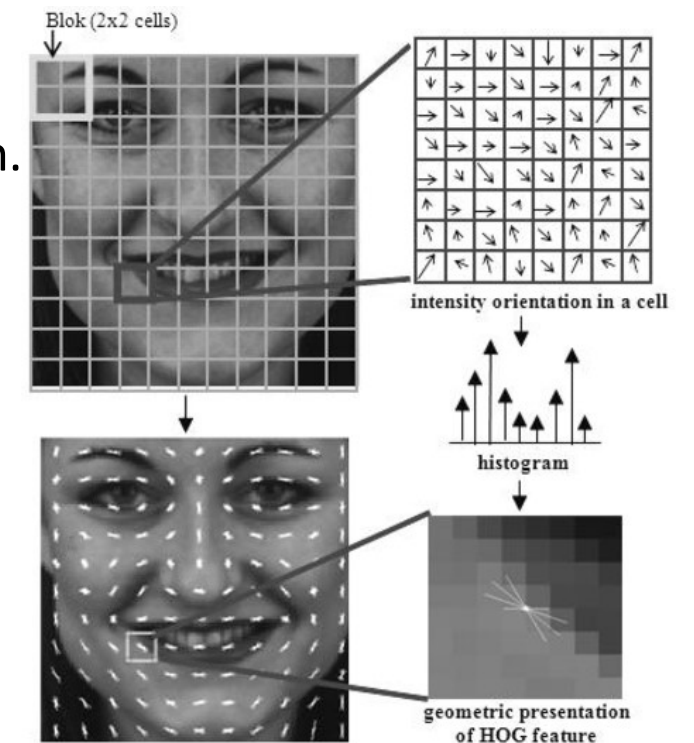
https://en.wikipedia.org/wiki/Histogram_of_oriented_gradients
<https://www.learnopencv.com/histogram-of-oriented-gradients/>

See also “*Haar Cascade Classifier*”

<http://www.willberger.org/cascade-haar-explained/>

These algorithms are all based on “sliding window” and “image pyramid”.

Given an input image we extract multiple “sliding” sub-pictures and we perform image classification based on the features extracted from it. Multi-resolution (image pyramid) is needed to support multiple size detection and it is a penalty for the complexity of the whole algorithm.



Classical object detection algorithms

Before CNN became popular few algorithms were common for feature detection, for example “*histogram of oriented gradients (HOG)*” used with “*scalar vector machine (SVM)*” for classification.

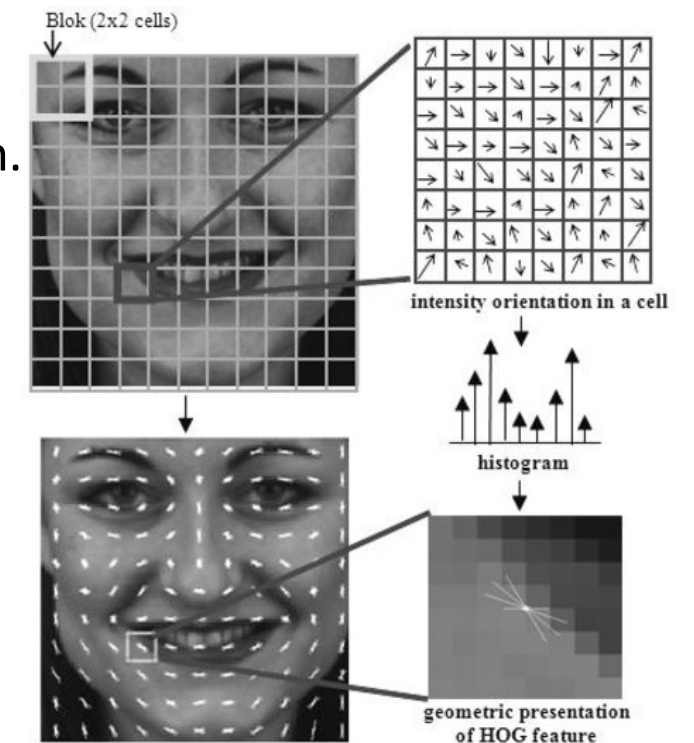
https://en.wikipedia.org/wiki/Histogram_of_oriented_gradients
<https://www.learnopencv.com/histogram-of-oriented-gradients/>

See also “*Haar Cascade Classifier*”

<http://www.willberger.org/cascade-haar-explained/>

These algorithms are all based on “sliding window” and “image pyramid”.

Given an input image we extract multiple “sliding” sub-pictures and we perform image classification based on the features extracted from it. Multi-resolution (image pyramid) is needed to support multiple size detection and it is a penalty for the complexity of the whole algorithm.



Classical object detection algorithms

Before CNN became popular few algorithms were common for feature detection, for example “*histogram of oriented gradients (HOG)*” used with “*scalar vector machine (SVM)*” for classification.

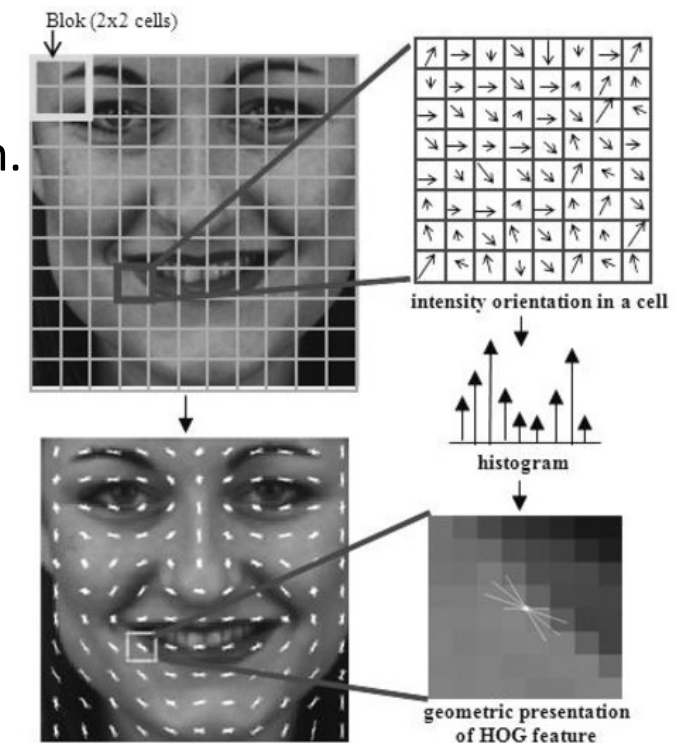
https://en.wikipedia.org/wiki/Histogram_of_oriented_gradients
<https://www.learnopencv.com/histogram-of-oriented-gradients/>

See also “*Haar Cascade Classifier*”

<http://www.willberger.org/cascade-haar-explained/>

These algorithms are all based on “sliding window” and “image pyramid”.

Given an input image we extract multiple “sliding” sub-pictures and we perform image classification based on the features extracted from it. Multi-resolution (image pyramid) is needed to support multiple size detection and it is a penalty for the complexity of the whole algorithm.



Classical object detection algorithms

Before CNN became popular few algorithms were common for feature detection, for example “*histogram of oriented gradients (HOG)*” used with “*scalar vector machine (SVM)*” for classification.

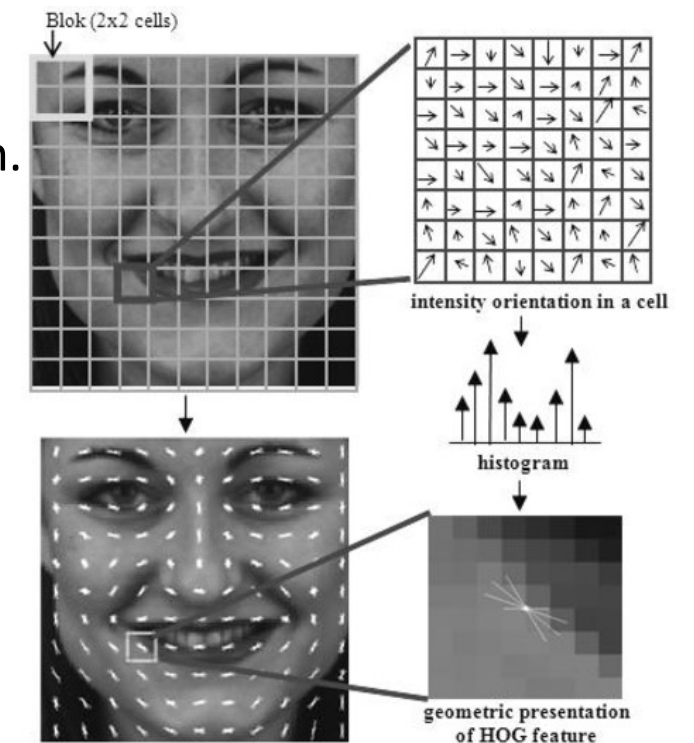
https://en.wikipedia.org/wiki/Histogram_of_oriented_gradients
<https://www.learnopencv.com/histogram-of-oriented-gradients/>

See also “*Haar Cascade Classifier*”

<http://www.willberger.org/cascade-haar-explained/>

These algorithms are all based on “sliding window” and “image pyramid”.

Given an input image we extract multiple “sliding” sub-pictures and we perform image classification based on the features extracted from it. Multi-resolution (image pyramid) is needed to support multiple size detection and it is a penalty for the complexity of the whole algorithm.



Classical object detection algorithms

Before CNN became popular few algorithms were common for feature detection, for example “*histogram of oriented gradients (HOG)*” used with “*scalar vector machine (SVM)*” for classification.

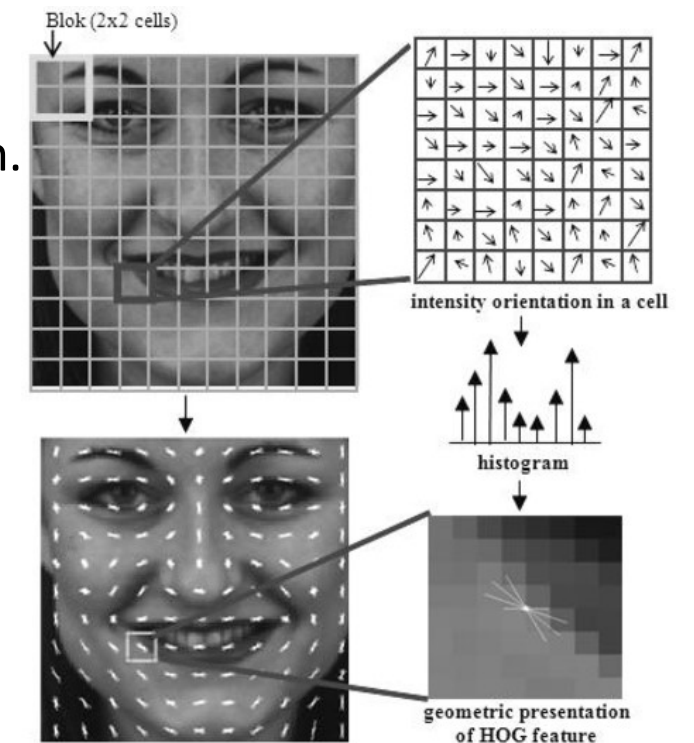
https://en.wikipedia.org/wiki/Histogram_of_oriented_gradients
<https://www.learnopencv.com/histogram-of-oriented-gradients/>

See also “*Haar Cascade Classifier*”

<http://www.willberger.org/cascade-haar-explained/>

These algorithms are all based on “sliding window” and “image pyramid”.

Given an input image we extract multiple “sliding” sub-pictures and we perform image classification based on the features extracted from it. Multi-resolution (image pyramid) is needed to support multiple size detection and it is a penalty for the complexity of the whole algorithm.



Lower resolution for bigger objects
obj yes/no = YES



Obj Detect
(HOG+SVM)



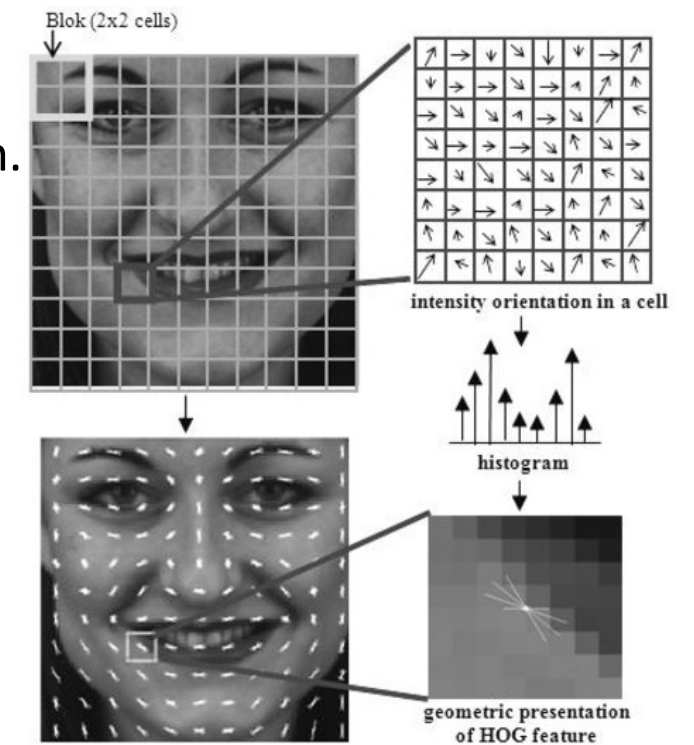
Classical object detection algorithms

Before CNN became popular few algorithms were common for feature detection, for example “*histogram of oriented gradients (HOG)*” used with “*scalar vector machine (SVM)*” for classification.

https://en.wikipedia.org/wiki/Histogram_of_oriented_gradients
<https://www.learnopencv.com/histogram-of-oriented-gradients/>

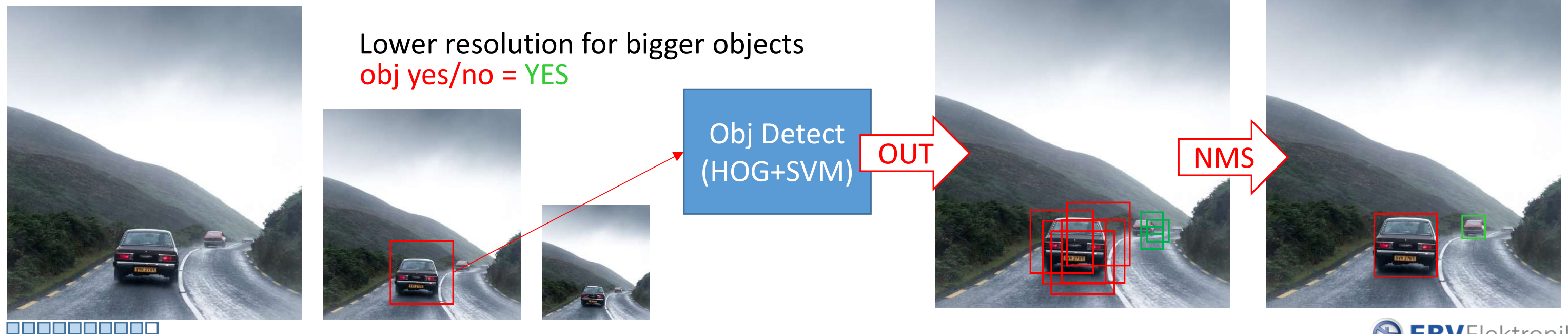
See also “*Haar Cascade Classifier*”

<http://www.willberger.org/cascade-haar-explained/>



These algorithms are all based on “sliding window” and “image pyramid”.

Given an input image we extract multiple “sliding” sub-pictures and we perform image classification based on the features extracted from it. Multi-resolution (image pyramid) is needed to support multiple size detection and it is a penalty for the complexity of the whole algorithm.



Classical object detection algorithms

Before CNN became popular few algorithms were common for feature detection, for example “*histogram of oriented gradients (HOG)*” used with “*scalar vector machine (SVM)*” for classification.

https://en.wikipedia.org/wiki/Histogram_of_oriented_gradients
<https://www.learnopencv.com/histogram-of-oriented-gradients/>

See also “*Haar Cascade Classifier*”

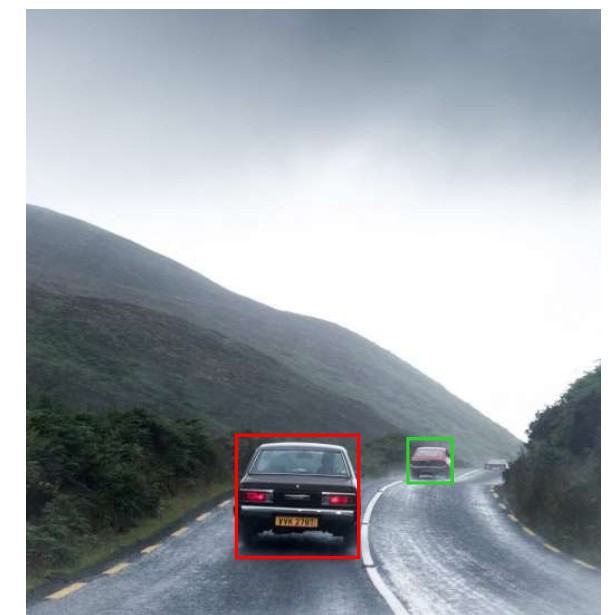
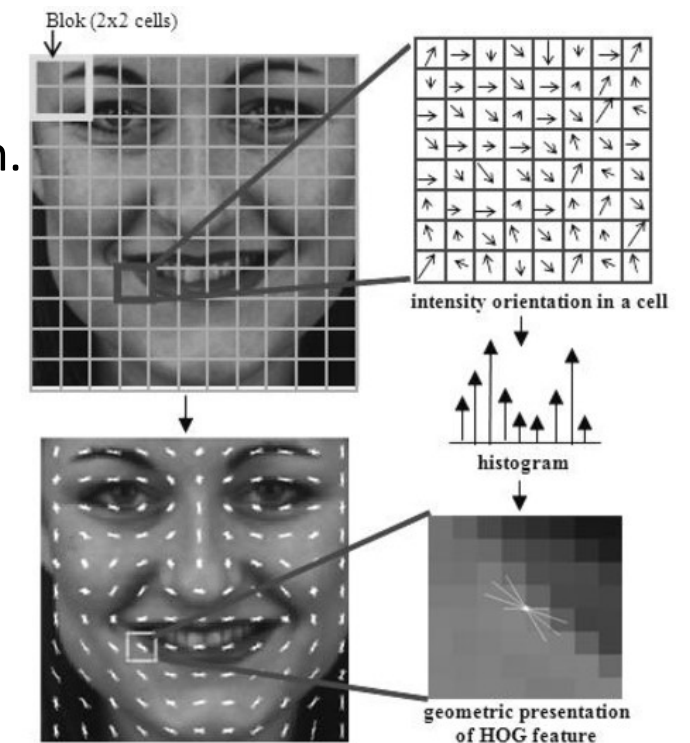
<http://www.willberger.org/cascade-haar-explained/>

These algorithms are all based on “sliding window” and “image pyramid”.

Given an input image we extract multiple “sliding” sub-pictures and we perform image classification based on the features extracted from it. Multi-resolution (image pyramid) is needed to support multiple size detection and it is a penalty for the complexity of the whole algorithm.

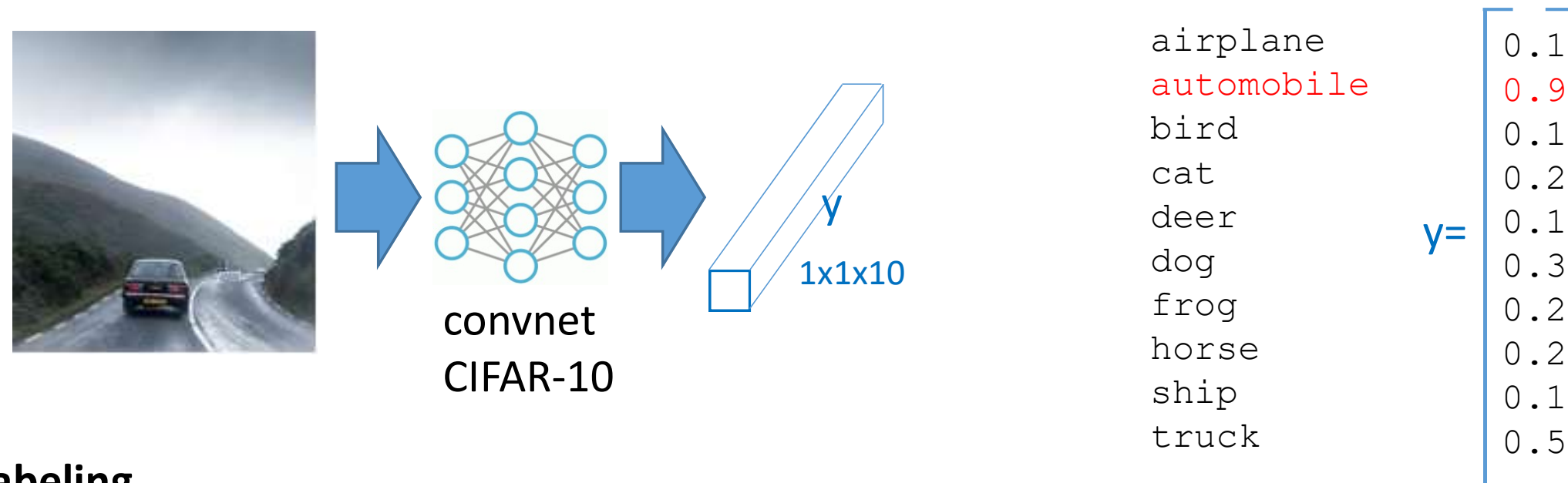
These methods have a strong penalty on performance because we have to run the same algorithm **multiple times** on different **overlapping** sub-pictures (loop of detections)

Computationally **not** efficient



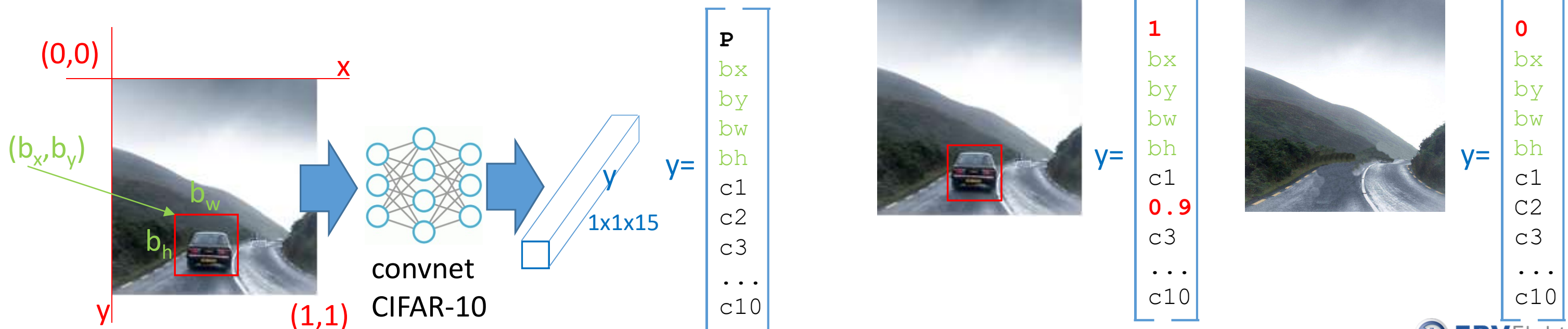
CNN: Object Localization

CNN like ResNet will classify “the whole picture” as if it does contain only one object (a cat or a dog). This is because we have trained the network to output only a *label* which is a “list of features” related to a “list of class”

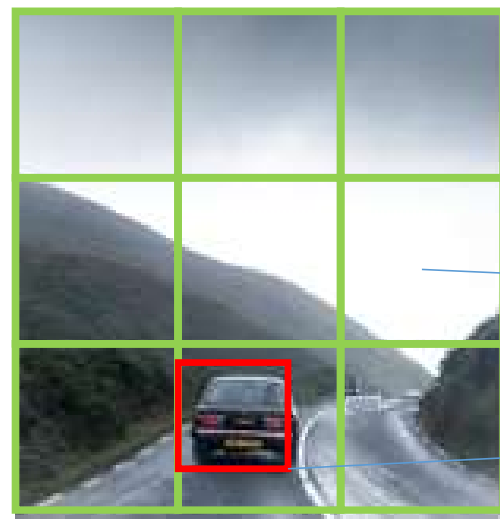


A better labeling..

Using a better labeling we can “add” features to our classification, for example localization of the object as a bounding box. Bounding box is identified by the absolute coordinates of origin, width, height. We also code the *probability of presence* P for the identified class. (note: coordinates are normalized)



CNN: bounding box ... better than sliding window



Bounding box does combine a “superimposed grid” with the image classification+localization as seen before. For each cell we define a label which contains 8 features: P , $\text{box}(x,y,w,h)$, $\text{class}(c_1, c_2, c_3)$

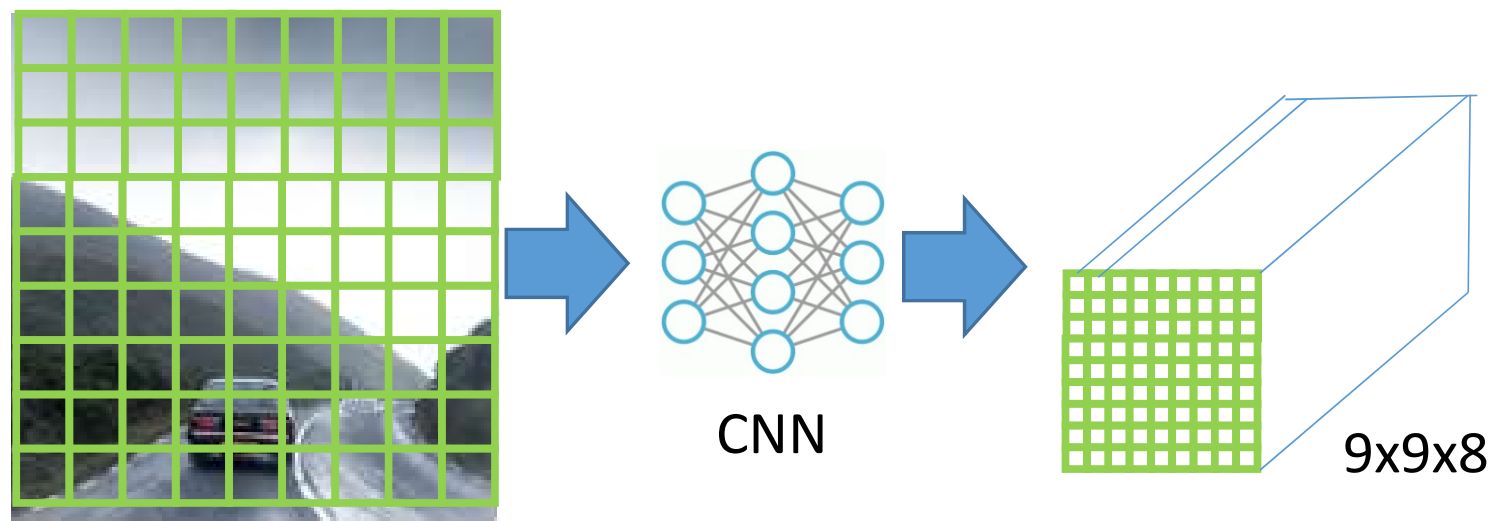
presence prob B.Box coordinates Classification result

Label : $Y = (P=0, bx, by, bw, bh, c_1, c_2, c_3 \dots c_n)$

Label : $Y = (P=1, 0.1, 0.3, 0.7, 0.8, c_1, c_2, c_3)$

Note: coord, size are always normalized and referred to the cell origin (upper left corner)

NOTE: The grid is NOT computed, it is defined as a coordinate system while creating labels for training.



CNN

9x9x8

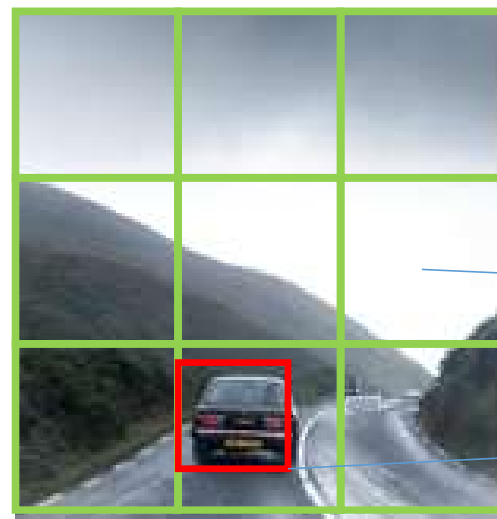
With only one pass into the CNN we obtain information for object presence, bounding box, category of an object inside a “virtual” grid cell.

Cells: $9 \times 9 = 81$

Label features: 8 (P , bbox, three classes)

Labeling on a 9x9 grid (finer detail)

CNN: bounding box ... better than sliding window



Bounding box does combine a “superimposed grid” with the image classification+localization as seen before. For each cell we define a label which contains 8 features: P , $\text{box}(x,y,w,h)$, $\text{class}(c_1, c_2, c_3)$

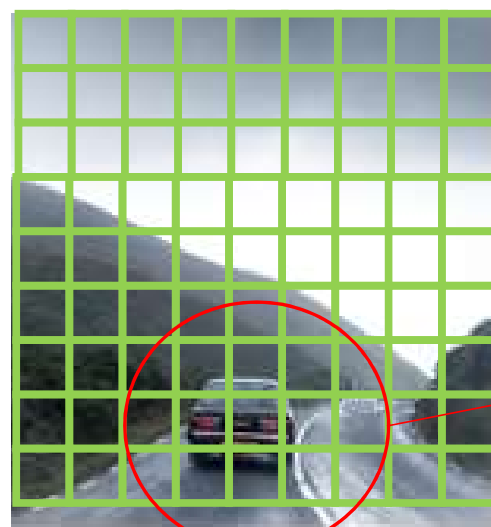
presence prob B.Box coordinates Classification result

Label : $Y = (P=0, bx, by, bw, bh, c_1, c_2, c_3 \dots c_n)$

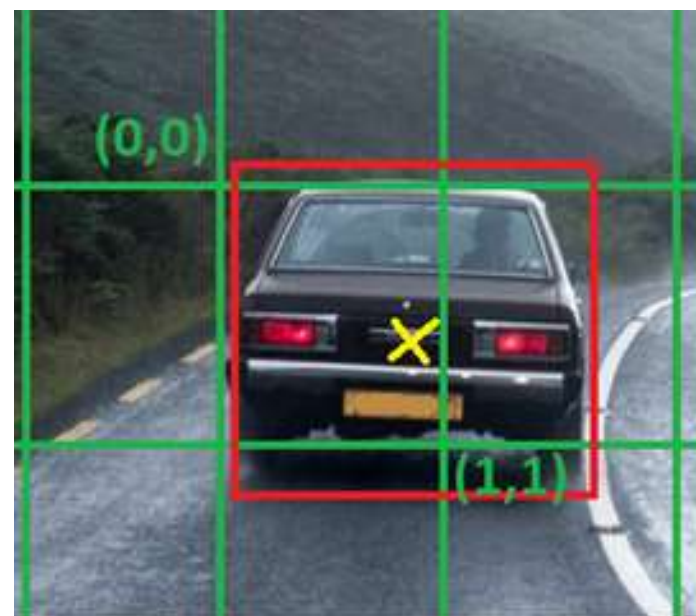
Label : $Y = (P=1, 0.1, 0.3, 0.7, 0.8, c_1, c_2, c_3)$

Note: coord, size are always normalized and referred to the cell origin (upper left corner)

NOTE: The grid is NOT computed, it is defined as a coordinate system while creating labels for training.



zoom



NOTE:

object is assigned (labeling) to the cell which contains the center point of the bounding box.

bx, by always between 0 and 1.

bw, bh always >0 but *can* be >1 .

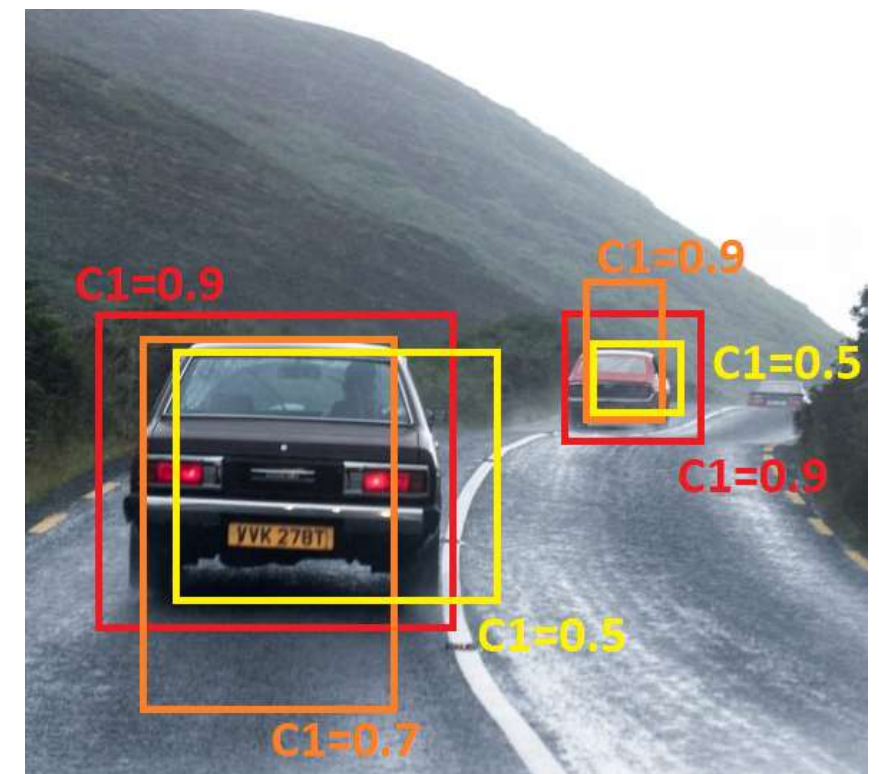
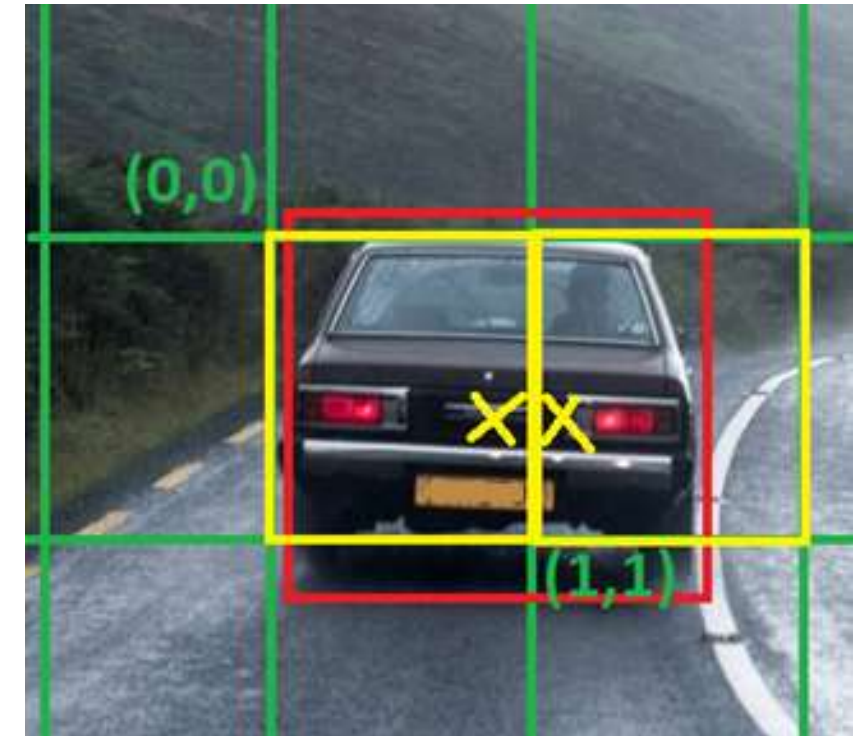
CNN: multiple detection and Non-Maxima suppression (NMS)

When running the CNN the output result will often have multiple detections for the same object especially when we have a *finer* grid.

To select only the best bounding box from all the ones provided on the grid we use IoU (intersection over union). Given two bounding box we compute the ratio between the area of intersection over the area of union



Example of NMS: When multiple bounding boxes are detected from different cells in the grid we *remove* the one with the maximum IoU referred to the one *classified* with the *higher* result. This is a post-processing step repeated over all the objects classes found in the input image.



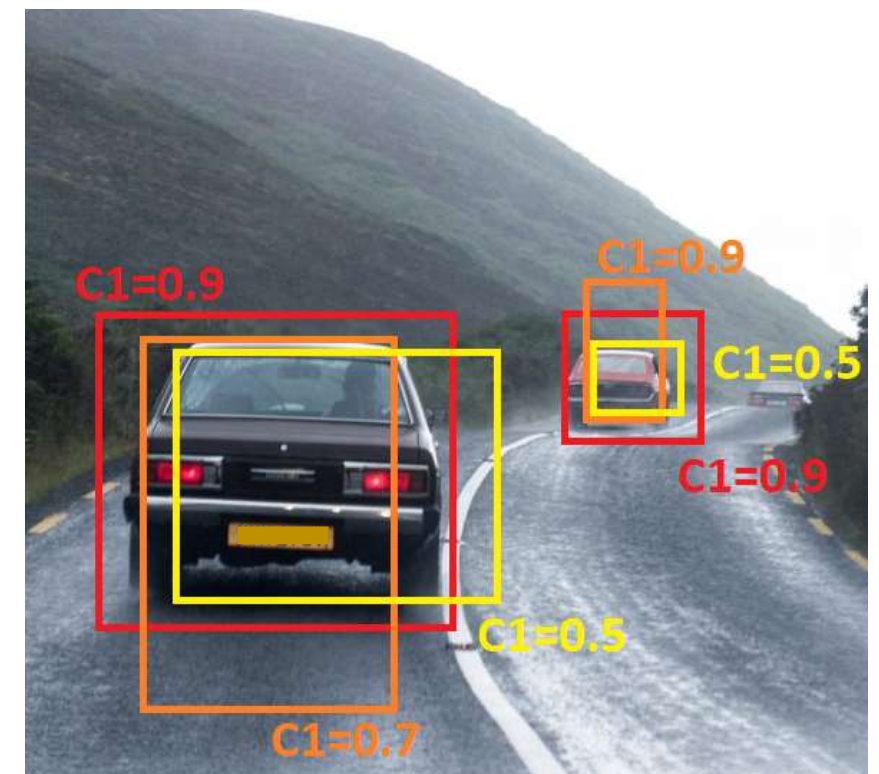
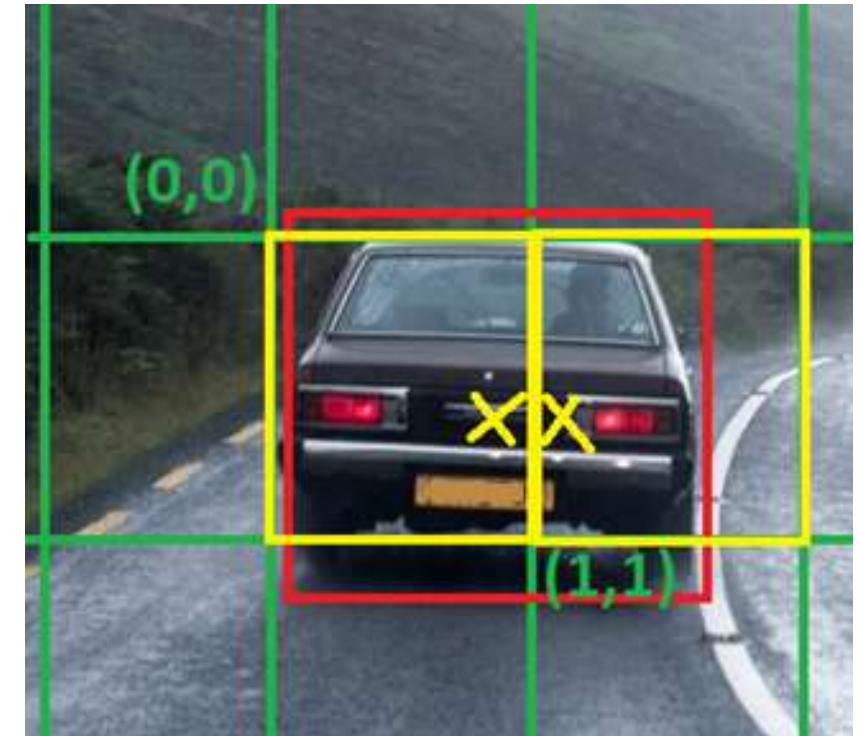
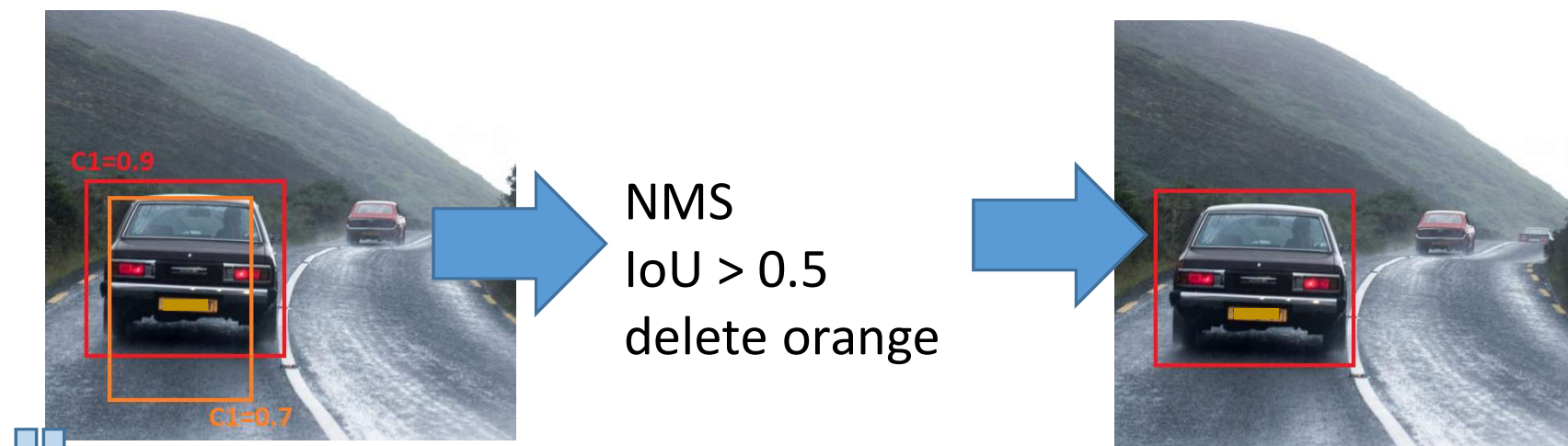
CNN: multiple detection and Non-Maxima suppression (NMS)

When running the CNN the output result will often have multiple detections for the same object especially when we have a *finer* grid.

To select only the best bounding box from all the ones provided on the grid we use IoU (intersection over union). Given two bounding box we compute the ratio between the area of intersection over the area of union



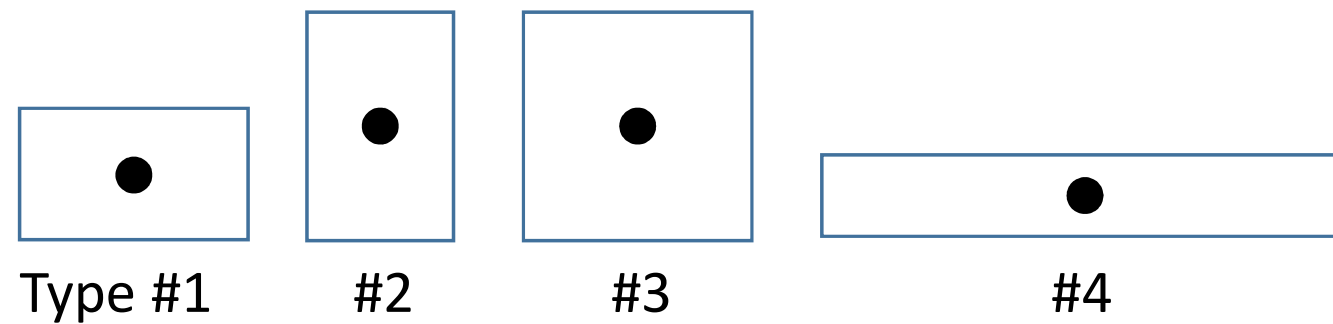
Example of NMS: When multiple bounding boxes are detected from different cells in the grid we *remove* the one with the maximum IoU referred to the one *classified* with the *higher* result. This is a post-processing step repeated over all the objects classes found in the input image.



CNN: overlapping objects and anchor boxes

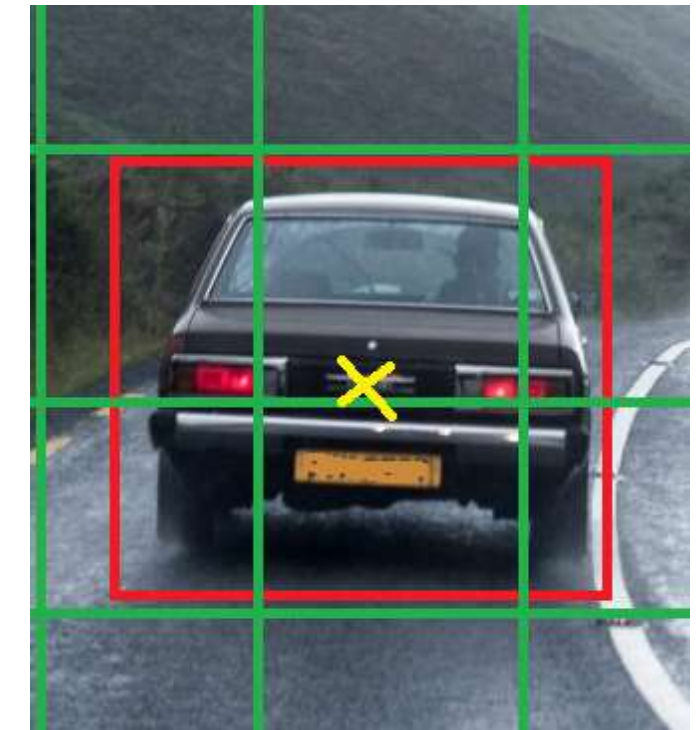
When multiple objects are in the same position they will overlap.
The grid cell algorithm is based on the definition of one bounding box for each grid cell assumed that there is only *one* object for a single cell

To be able to detect multiple (overlapping) objects for a single cell we define a feature called *anchor boxes* where for each cell of the grid we support multiple bounding boxes of different (pre defined) aspect ratio.



Each label y is now a longer vector with multiple bounding boxes where the order of labeling is based on the *shape* of the anchor box. For example with only n.2 anchors:

$$\text{Label: } Y = (\underbrace{P, bx, by, bw, bh, c_1, c_2, c_3 \dots c_n}_{\text{anchor box type \#1}}, \underbrace{P, bx, by, bw, bh, c_1, c_2, c_3 \dots c_n}_{\text{anchor box type \#2}})$$



YOLO (you look only once) <https://pjreddie.com/publications/>

Unified, Real-Time Object Detection by Joseph Redmon, Santosh Divvala, Ross Girshick and Ali Farhadi (2015).



[ref docs\\[OBJ\]yolo v1 1506.02640.pdf](#)
[ref docs\\[OBJ\]yolo v2 1612.08242.pdf](#)
[ref docs\\[OBJ\]yolo v3 1804.02767.pdf](#)

- Dataset is labeling datapoints as per **SxS** grid
- Dataset is labeled on **C** classes
- Each grid cell contains **B** bounding boxes, but only 1 object (YOLO_V1)
- CNN was trained on Pascal VOC for YOLO_V1 (20 class)
- V1 CNN will return a result Y with $[(P, bx, by, bw, bh) * B, C1...Cc] * (SxS)$
 (bx, by are now the *center* of the box relative to the grid cell, only one classification)

YOLO uses a custom Loss Function during training.

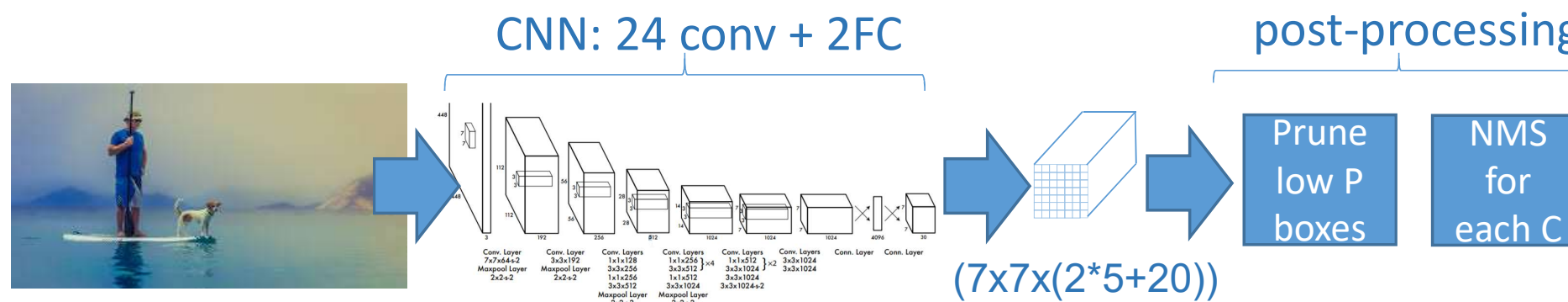
YOLO does post-process results by eliminating anchor boxes with low P

YOLO does NMS on the remaining results (one NMS for each class Cx)

Originally Pascal VOC dataset was used, S=7, B=2 and C=20.

CNN originally inspired by GoogLeNet.

| Name | Filters | Output Dimension |
|------------|------------------------|-------------------|
| Conv 1 | 7 x 7 x 64, stride=2 | 224 x 224 x 64 |
| Max Pool 1 | 2 x 2, stride=2 | 112 x 112 x 64 |
| Conv 2 | 3 x 3 x 192 | 112 x 112 x 192 |
| Max Pool 2 | 2 x 2, stride=2 | 56 x 56 x 192 |
| Conv 3 | 1 x 1 x 128 | 56 x 56 x 128 |
| Conv 4 | 3 x 3 x 256 | 56 x 56 x 256 |
| Conv 5 | 1 x 1 x 256 | 56 x 56 x 256 |
| Conv 6 | 1 x 1 x 512 | 56 x 56 x 512 |
| Max Pool 3 | 2 x 2, stride=2 | 28 x 28 x 512 |
| Conv 7 | 1 x 1 x 256 | 28 x 28 x 256 |
| Conv 8 | 3 x 3 x 512 | 28 x 28 x 512 |
| Conv 9 | 1 x 1 x 256 | 28 x 28 x 256 |
| Conv 10 | 3 x 3 x 512 | 28 x 28 x 512 |
| Conv 11 | 1 x 1 x 256 | 28 x 28 x 256 |
| Conv 12 | 3 x 3 x 512 | 28 x 28 x 512 |
| Conv 13 | 1 x 1 x 256 | 28 x 28 x 256 |
| Conv 14 | 3 x 3 x 512 | 28 x 28 x 512 |
| Conv 15 | 1 x 1 x 512 | 28 x 28 x 512 |
| Conv 16 | 3 x 3 x 1024 | 28 x 28 x 1024 |
| Max Pool 4 | 2 x 2, stride=2 | 14 x 14 x 1024 |
| Conv 17 | 1 x 1 x 512 | 14 x 14 x 512 |
| Conv 18 | 3 x 3 x 1024 | 14 x 14 x 1024 |
| Conv 19 | 1 x 1 x 512 | 14 x 14 x 512 |
| Conv 20 | 3 x 3 x 1024 | 14 x 14 x 1024 |
| Conv 21 | 3 x 3 x 1024 | 14 x 14 x 1024 |
| Conv 22 | 3 x 3 x 1024, stride=2 | 7 x 7 x 1024 |
| Conv 23 | 3 x 3 x 1024 | 7 x 7 x 1024 |
| Conv 24 | 3 x 3 x 1024 | 7 x 7 x 1024 |
| FC 1 | - | 4096 |
| FC 2 | - | 7 x 7 x 30 (1470) |



YOLO (you *look* only once) [https://en.wikipedia.org/wiki/YOLO_\(aphorism\)](https://en.wikipedia.org/wiki/YOLO_(aphorism))

1



YOLO improved by changing CNN (from GoogLeNet to Darknet), by adding anchor boxes (V2), by changing the training methodology and (multi scale) and by using a finer grid ...

YOLO_V3 uses DarkNet (106 layers) to perform detection at THREE scales of resolution. Also uses 9 anchor boxes (3 for each scale)

YOLO (you *look* only once) [https://en.wikipedia.org/wiki/YOLO_\(aphorism\)](https://en.wikipedia.org/wiki/YOLO_(aphorism))

1

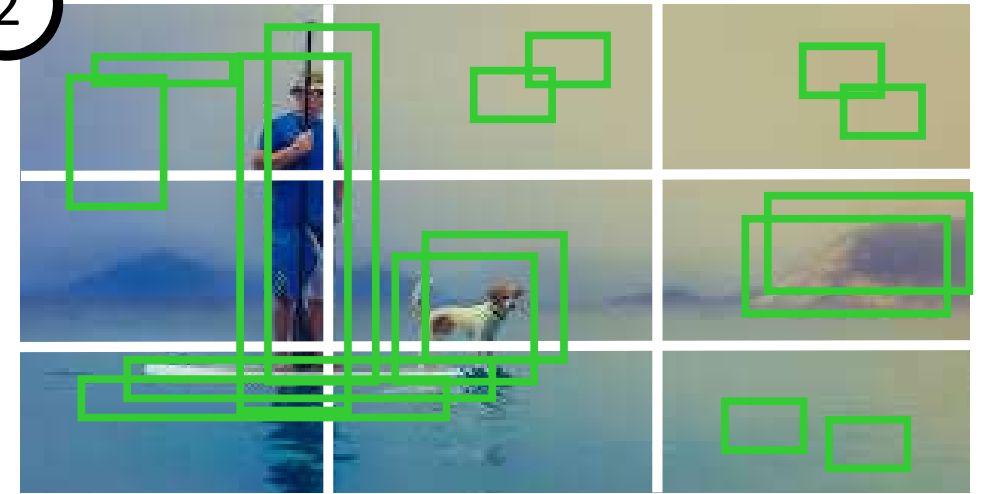


2) Given a 3x3 grid, the CNN is trained to produce 2 b.boxes results for **each** cell.

$Y = [[P, bx, by, bw, bh, C1..C20]$
 $[P, bx, by, bw, bh, C1..C20]]$

P is our "confidence" for obj being present in the cell.

2



YOLO improved by changing CNN (from GoogLeNet to Darknet), by adding anchor boxes (V2), by changing the training methodology and (multi scale) and by using a finer grid ...

YOLO_V3 uses DarkNet (106 layers) to perform detection at THREE scales of resolution. Also uses 9 anchor boxes (3 for each scale)

YOLO (you *look* only once) [https://en.wikipedia.org/wiki/YOLO_\(aphorism\)](https://en.wikipedia.org/wiki/YOLO_(aphorism))

1



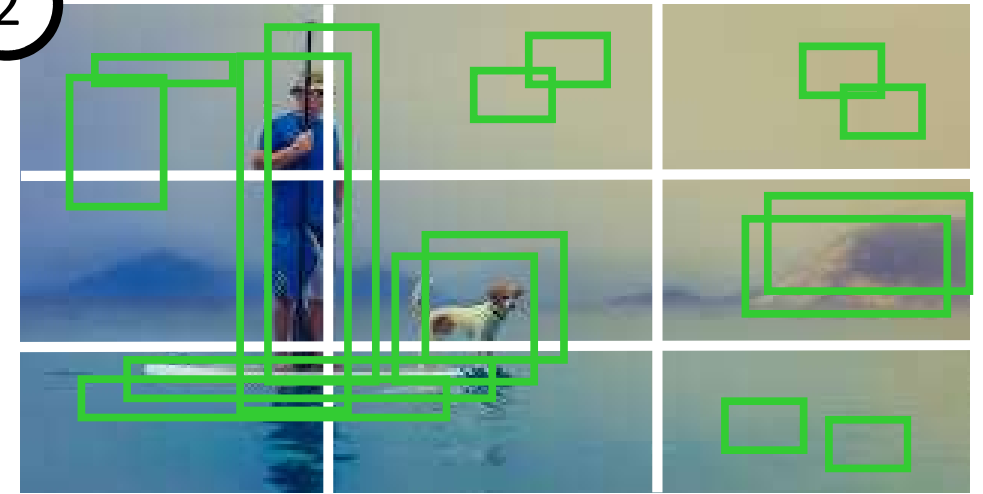
2) Given a 3x3 grid, the CNN is trained to produce 2 b.boxes results for **each** cell.

$Y = [[P, bx, by, bw, bh, C1..C20]$
 $[P, bx, by, bw, bh, C1..C20]]$

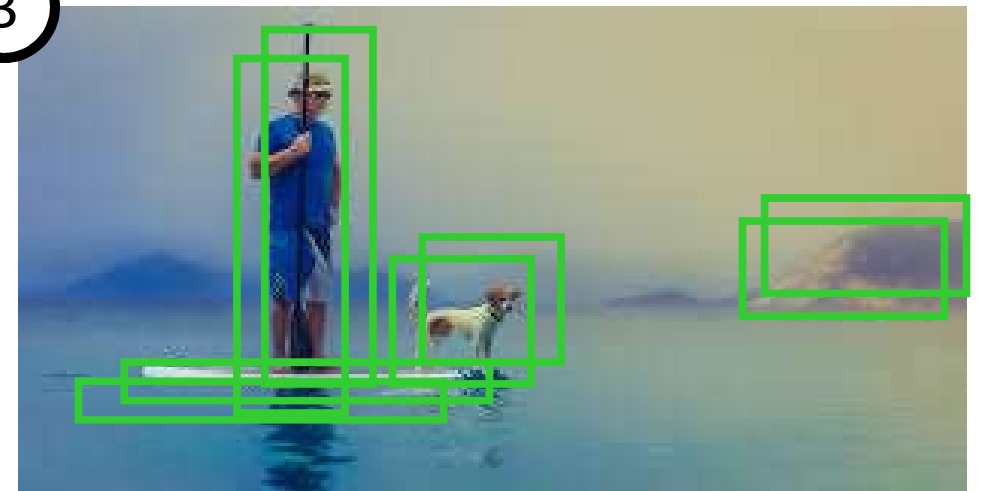
P is our “confidence” for obj being present in the cell.

3) Based on the confidence level being “too low” to be an object we eliminate many *false* boxes while post processing the CNN results

2



3



YOLO improved by changing CNN (from GoogLeNet to Darknet), by adding anchor boxes (V2), by changing the training methodology and (multi scale) and by using a finer grid ...

YOLO_V3 uses DarkNet (106 layers) to perform detection at THREE scales of resolution. Also uses 9 anchor boxes (3 for each scale)

YOLO (you look only once) <https://en.wikipedia.org/wiki/YOLO> (aphorism)

1



YOLO improved by changing CNN (from GoogLeNet to Darknet), by adding anchor boxes (V2), by changing the training methodology and (multi scale) and by using a finer grid ...

YOLO_V3 uses DarkNet (106 layers) to perform detection at THREE scales of resolution. Also uses 9 anchor boxes (3 for each scale)

2) Given a 3x3 grid, the CNN is trained to produce 2 b.boxes results for **each** cell.

$$Y = [[P, bx, by, bw, bh, C1..C20] \\ [P, bx, by, bw, bh, C1..C20]]$$

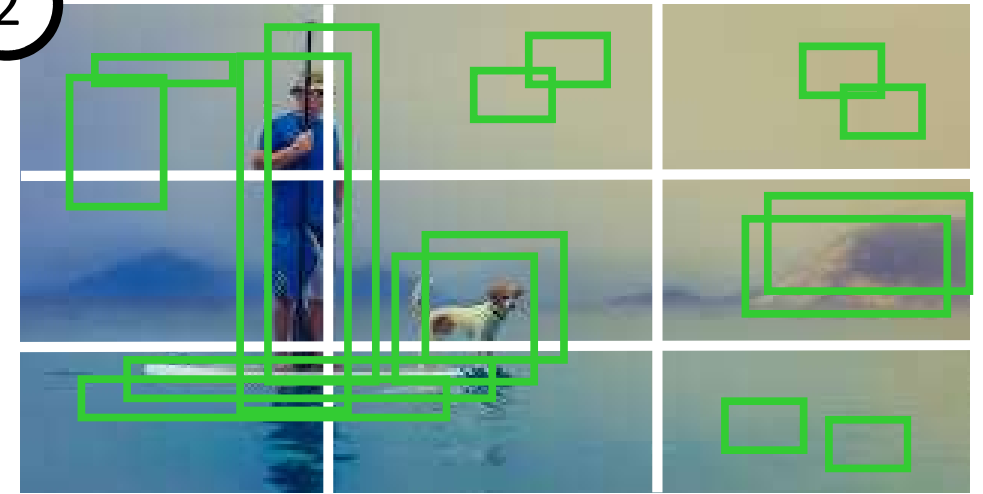
P is our "confidence" for obj being present in the cell.

3) Based on the confidence level being "too low" to be an object we eliminate many *false* boxes while post processing the CNN results

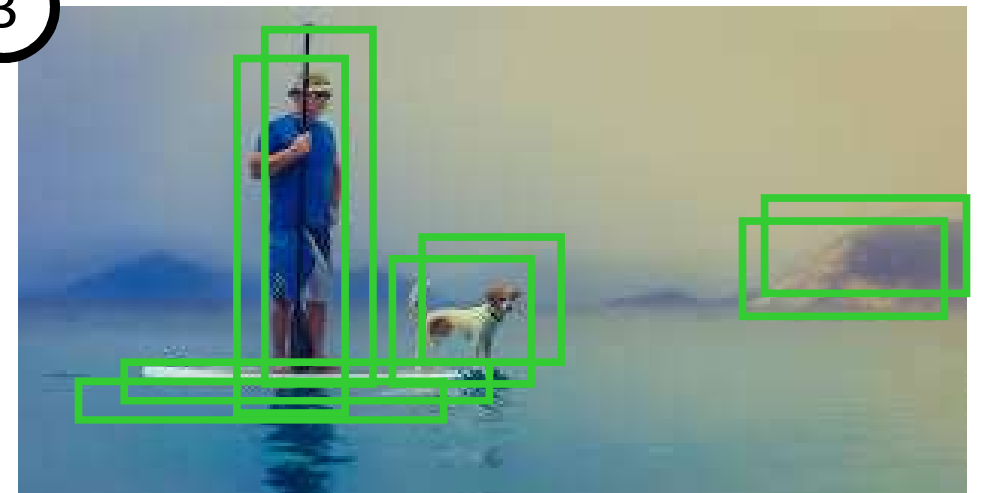
4) The last step of post processing will run NMS once for each class C (i.e. 20 times in this example) to keep only the most *confident* result.

$$class\ confidence = P(class_i) * IoU$$

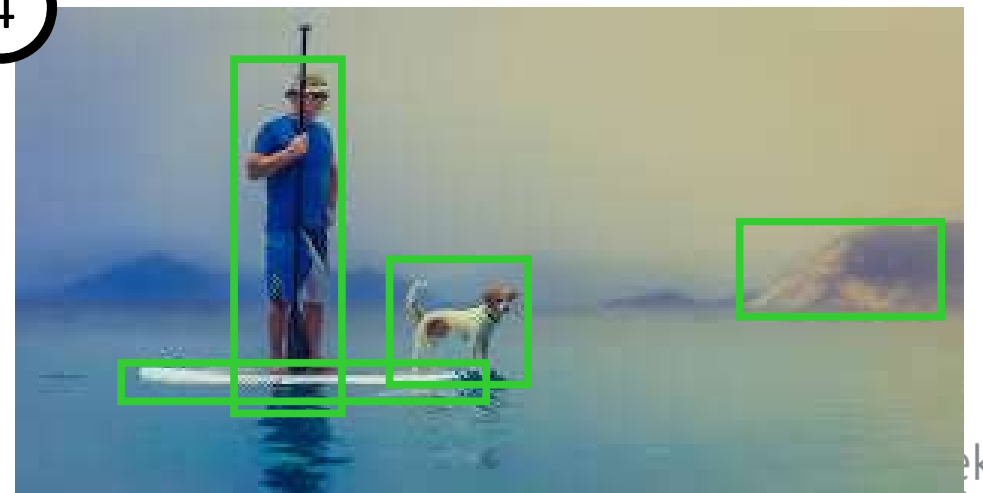
2



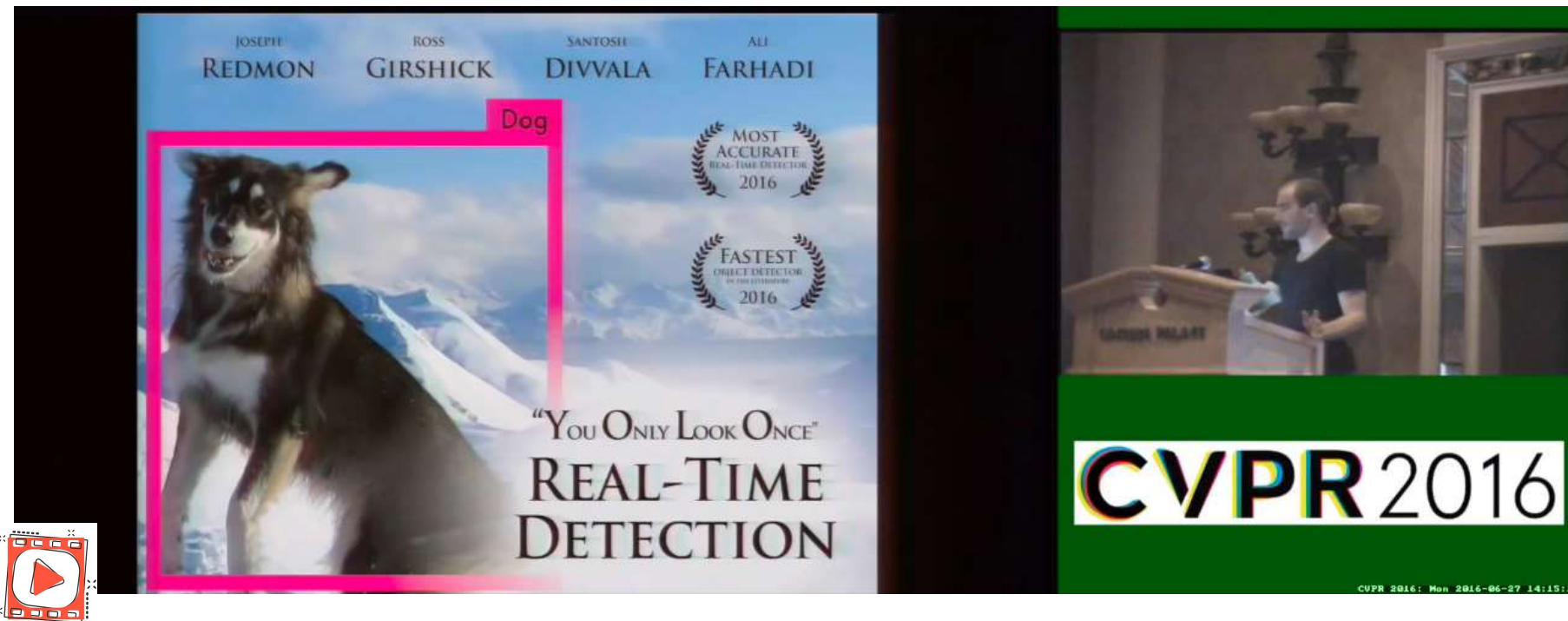
3



4



YOLO (you look only once) Computer Vision & Pattern Recognition (CVPR2016)



YoLo: the fastest algorithm at the time of publication still *not* a lightweight model for “IoT/Edge-Ai” devices

Cortex A-53@1.2Ghz : ~1 fps

Cortex A-72@1.5Ghz : ~2 fps

- Only ARM sw (no NN acceleration, no GPU)
- Image resolution 416x416 px
- Arm NEON optimized

<https://github.com/Tencent/ncnn/tree/master/benchmark>

There are simplified versions of YoLo which runs *faster* with a *lower* accuracy: see Tiny_Yolo, Yolo_Lite (10x speedup)

- Removal of few conv layers
- Removal of batch normalization
- Removal of pooling layers in favor of conv+stride
- Use of faster classification networks (MobileNet)

Useful Links:

<https://towardsdatascience.com/yolo-v3-object-detection-53fb7d3bfe6b>

<https://hackernoon.com/understanding-yolo-f5a74bbc79> 67

https://medium.com/@jonathan_hui/real-time-object-detection-with-yolo-yolov2-28b1b93e2088

<https://qengineering.eu/deep-learning-with-raspberry-pi-and-alternatives.html>

<https://towardsdatascience.com/retinanet-how-focal-loss-fixes-single-shot-detection-cb320e3bb0de>

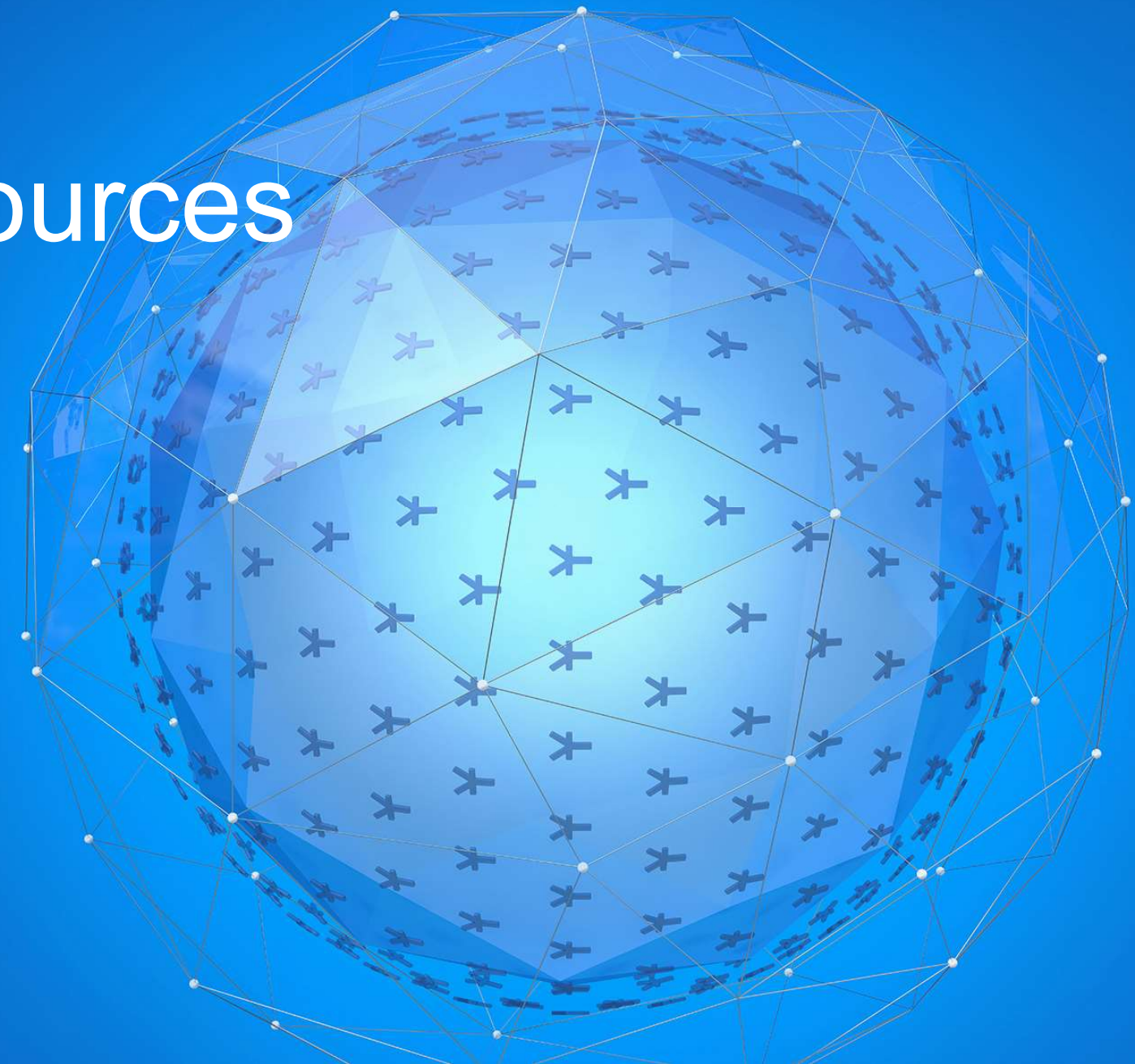


YOLO-LITE

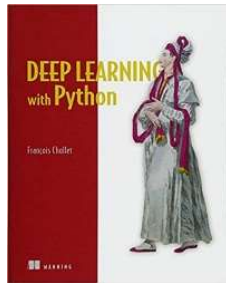
[ref_docs\\[CNN\\]Yolo_Lite_1811.05588.pdf](#)

“ ... reducing the input image size by a half can more than double the speed of the network (6.94 FPS vs 2.4 FPS) but will also effect the mAP (30.24% vs 40.48%). Reducing the input image size means that less of the image is passed through the network. This allows the network to be leaner, but also means that some data was lost.”

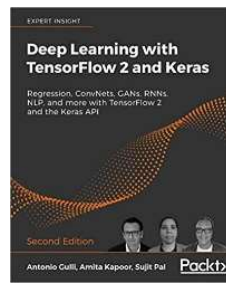
Books, Docs and Web resources



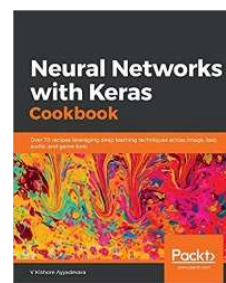
Books (as a quick start...)



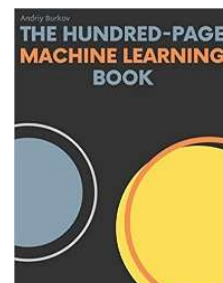
Deep Learning with Python
F.Chollet



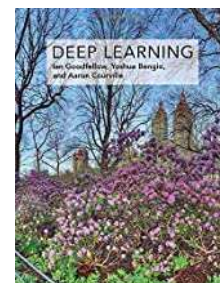
Deep Learning with TF 2.0 and Keras
A.Gulli



Neural Networks with Keras Cookbook



The 100Page Machine Learning,
A. Burkov



Deep Learning
Ian Goodfellow ,
Yoshua Bengio

Online Courses: Coursera by Andrew Ng



Machine Learning

Stanford University

COURSE

★★★★☆ 4.9 (120,9)

Mixed

<https://www.coursera.org/learn/machine-learning>



Deep Learning

deeplearning.ai

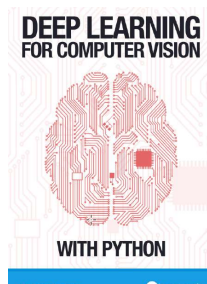
SPECIALIZATION

★★★★☆ 4.8

Intermediate

<https://www.coursera.org/specializations/deep-learning>

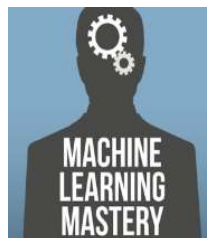
Online Courses + Books + Tutorials + Blog



<https://www.pyimagesearch.com/>

By Adrian Rosebrock, PhD

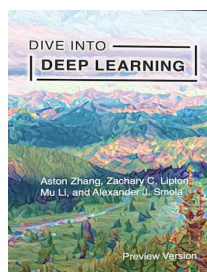
NOTE: one of the *most* comprehensive, updated, learning source for CNNs & image processing.



<https://machinelearningmastery.com/>

By Jason Brownlee, PhD

NOTE: not only focused on computer vision, excellent source also for RNN, LSTM etc.



<https://d2l.ai/index.html>

Dive into Deep Learning

An interactive deep learning book with code, math, and discussions, based on the NumPy interface.

Online Courses: by Fast.Ai



<https://www.fast.ai/>

Online Courses: by Stanford Univ.



<http://cs231n.stanford.edu/>

<http://cs231n.github.io/convolutional-networks/>



ConvNetJS : by A. Karpathy

Deep Learning in your browser

<https://cs.stanford.edu/people/karpathy/convnetjs/>

Thank You.

ai@ebv.com

Gianluca Filippini

EBV / FAE - ML Specialist

Ulrich Schmidt

EBV / Segment Manager - Hi-End Processing



**TECHNOLOGY.
PASSION.
EBV.**