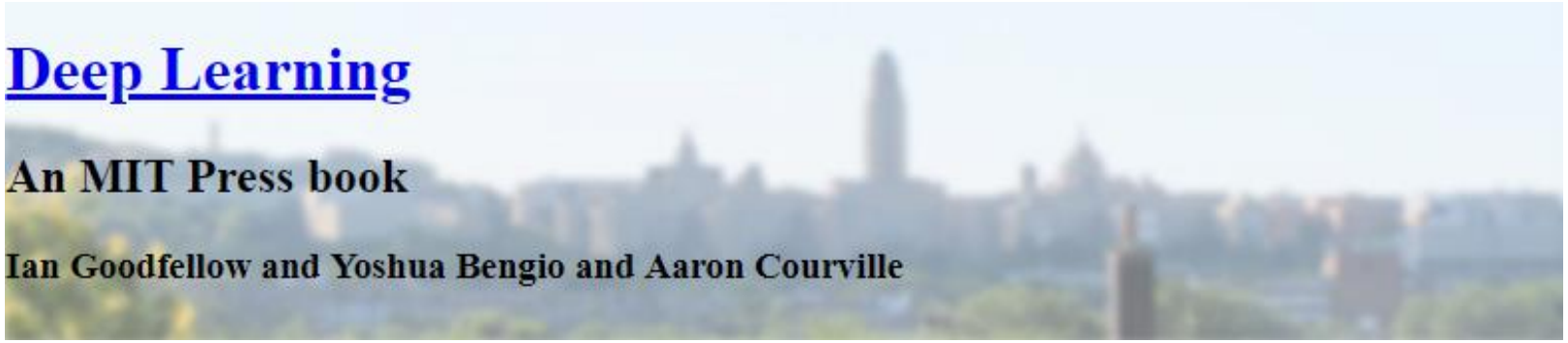
Review of Deep Learning and Convolutional Neural Networks

Pablo Mesejo and Víctor Vargas
Universidad de Granada
Departamento de Ciencias de la Computación e Inteligencia Artificial

UNIVERSIDAD DE GRANADA

Recommended Bibliography



https://www.deeplearningbook.org/ (2016)

LeCun, Y., Bengio, Y., & Hinton, G. (2015). *Deep learning.* Nature, *521*(7553), 436-444.

Schmidhuber, J. (2015). *Deep learning in neural networks: An overview.* Neural networks, 61, 85-117.

Bengio, Y.; Courville, A.; Vincent, P. (2013). *Representation Learning: A Review and New Perspectives*. IEEE Transactions on Pattern Analysis and Machine Intelligence. 35 (8): 1798–1828.

Bengio, Y. (2009). *Learning deep architectures for AI.* Foundations and Trends in Machine Learning, 2(1), 1-127.

Recommended online resources

- “CS231n: Convolutional Neural Networks for Visual Recognition” (Stanford): http://cs231n.stanford.edu/
- “Neural Networks for Machine Learning” (course by Geoffrey Hinton, University of Toronto/Coursera): https://www.youtube.com/watch?v=cbeTc-Urqak
- “Deep Learning Specialization” (course by AndrewNg, Stanford/deeplearning.ai): https://www.youtube.com/watch?v=CS4cs9xVecg&list=PLkDaE6sCZn6Ec-XTbcX1uRg2_u4xOEky0
- “Dive into Deep Learning. Interactive deep learning book with code, math, and discussions” (A. Zhang, Z.C. Lipton, M. Li, A.J. Smola): https://d2l.ai/
- “CS230 Deep Learning” (Stanford): https://cs230.stanford.edu/lecture/
- Andrej Karpathy’s blog: https://karpathy.github.io/
- Christopher Olah’s blog: https://colah.github.io/ (especially for RNNs)
- https://distill.pub/
- “Practical Deep Learning” (Jeremy Howard): https://www.fast.ai/
- “Neural Networks and Deep Learning” (Michael Nielsen): http://neuralnetworksanddeeplearning.com/

1. Introduction

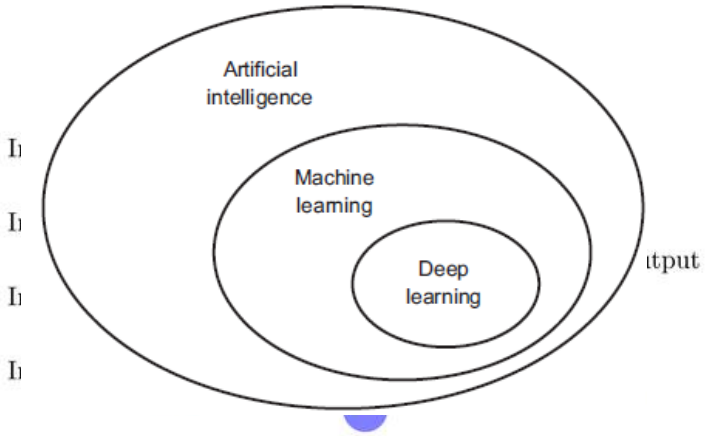
Main AI paradigms

Symbolic/classic

```
SI
        CARD_NUMBER= verified        &
        DATE       = no expired      &
        CODES      = correct         &
        ATTEMPTS   = no exceded      &
        BALANCE    = sufficient      &
        LIMIT      = no exceeded     &
THEN
        PAYMENT    = authorized
```

Example: rule-based systems.
It starts from knowledge (high-level information)

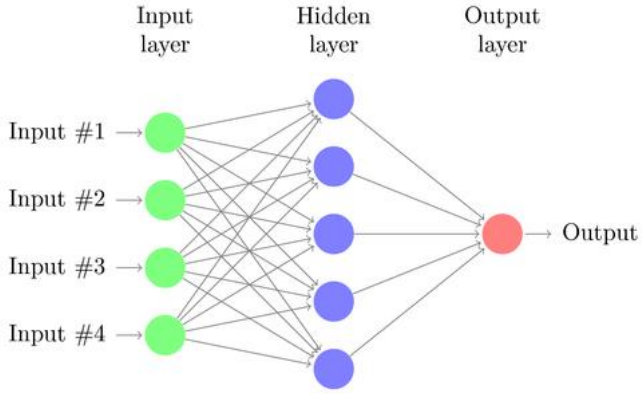
Sub-symbolic/conectionist



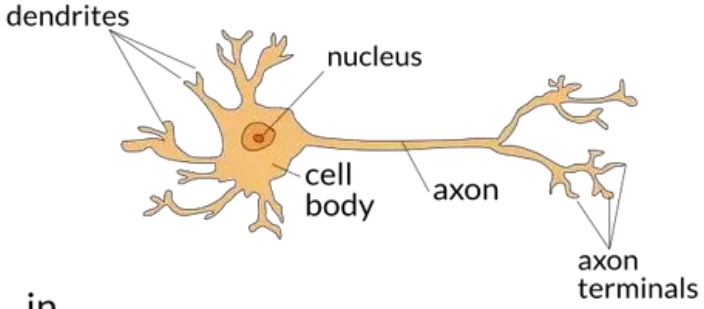
Example: artificial neural networks.
It starts fom data (low-level information)

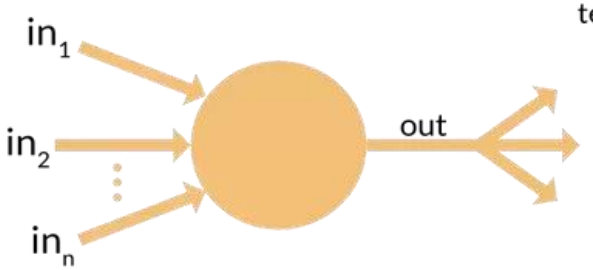
Neural Networks

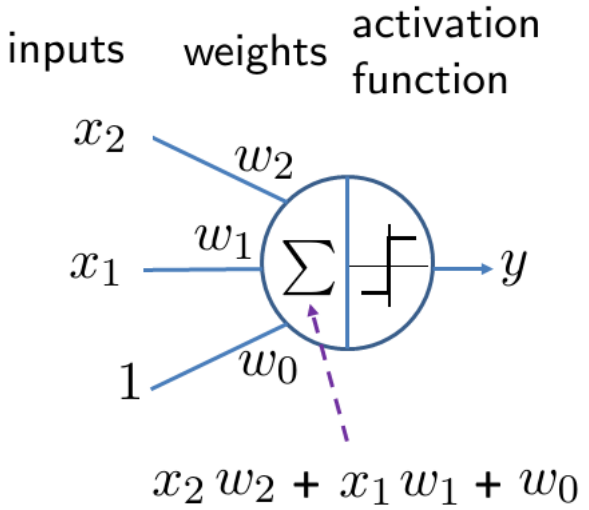
A set of simple processing units that are highly interconnected and, through a learning process, solve a specific task



What is an artificial neuron?

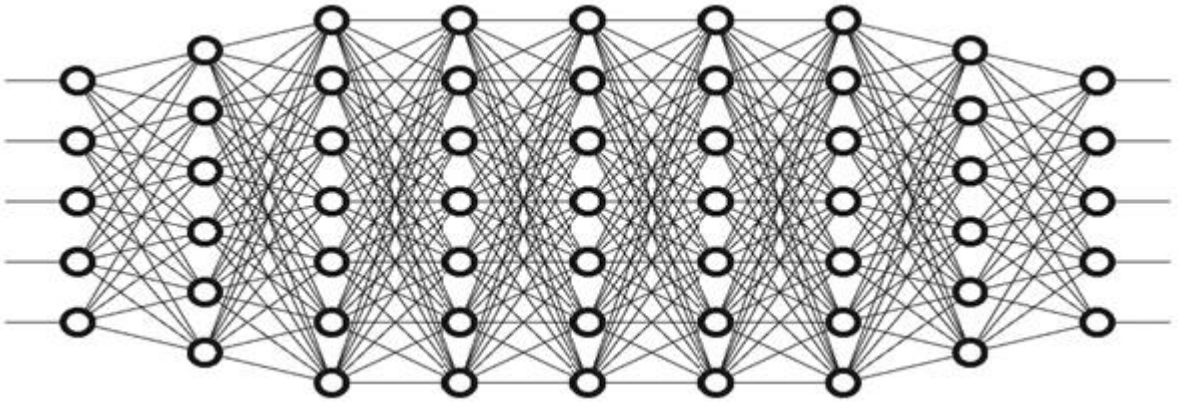




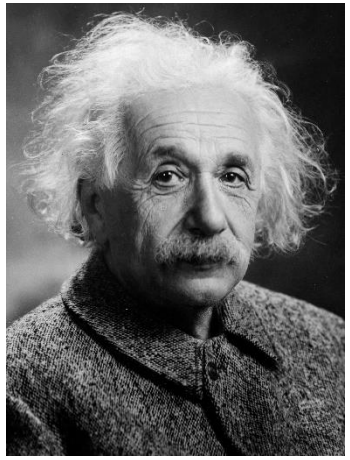


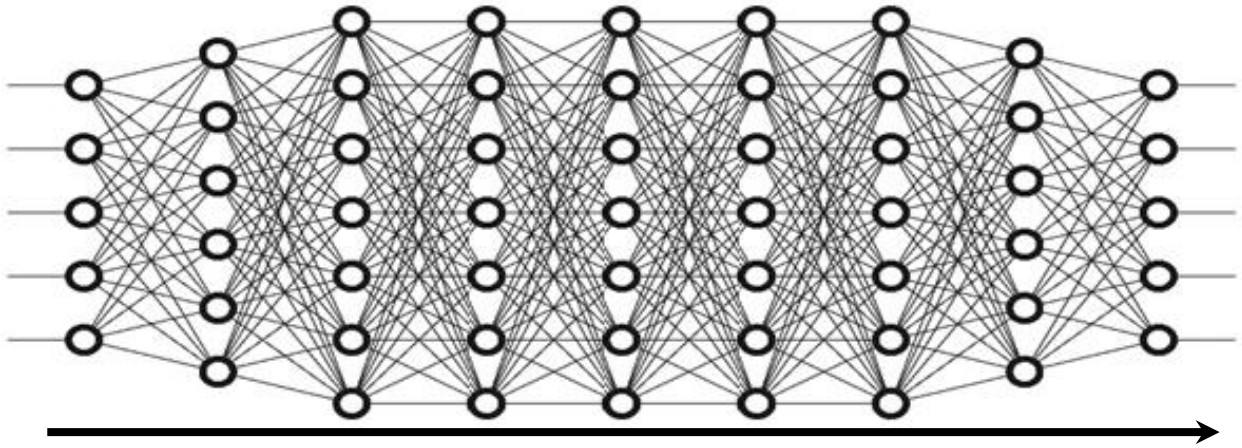
What are Deep Neural Networks?

- Official definition
  - computational models that are composed of multiple processing layers to learn representations of data with multiple levels of abstraction (LeCun et al., 2015)
- In practice:
  - Neural networks with "many" hidden layers
    - And that, as a consequence, can approximate more complex functions (i.e. solve more difficult problems)

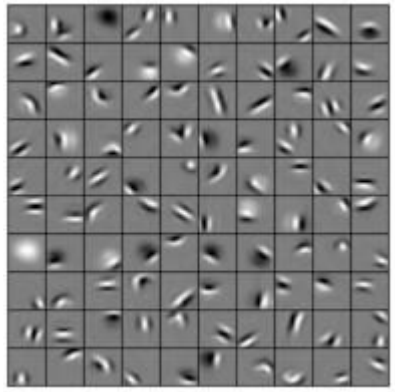


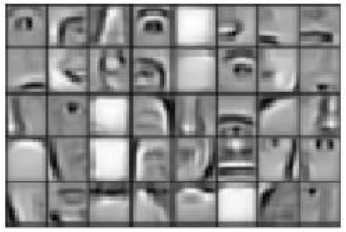
What are Deep Neural Networks?





“Albert Einstein”





They learn hierarchical representations of data (multiple levels of abstraction)

What are Deep Neural Networks?

François Chollet 
@fchollet

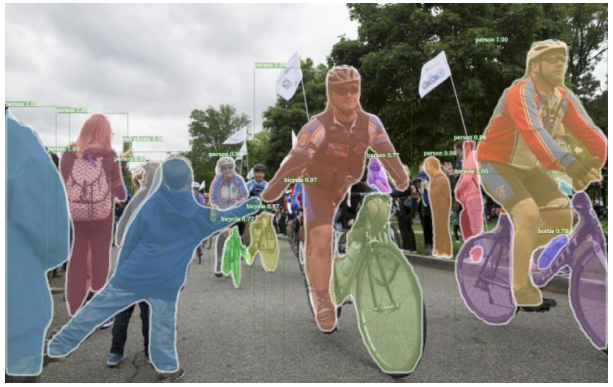
"Neural networks" are a sad misnomer. They're neither neural nor even networks. They're chains of differentiable, parameterized geometric functions, trained with gradient descent (with gradients obtained via the chain rule). A small set of highschool-level ideas put together

Traducir post

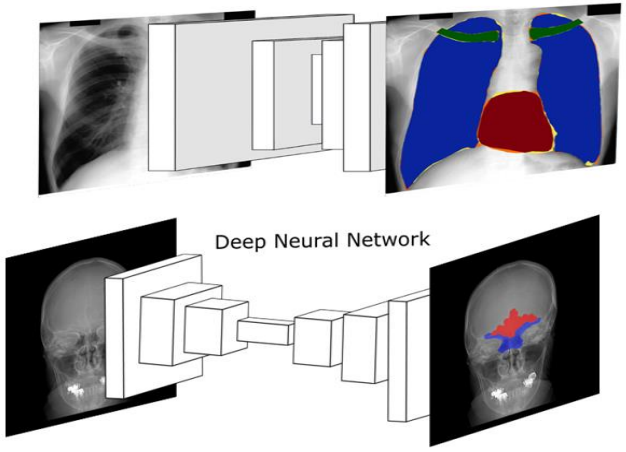
8:58 p. m. · 12 ene. 2018

Applications (1)

Object Recognition



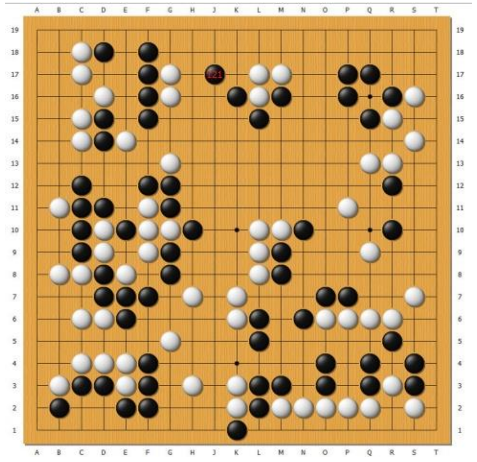
Mask R-CNN output from
https://github.com/facebookresearch/Detectron



"Realtime Multi-Person 2D Pose Estimation using Part Affinity Fields" (CVPR'17)

Reinforcement Learning in Games

AlphaGo

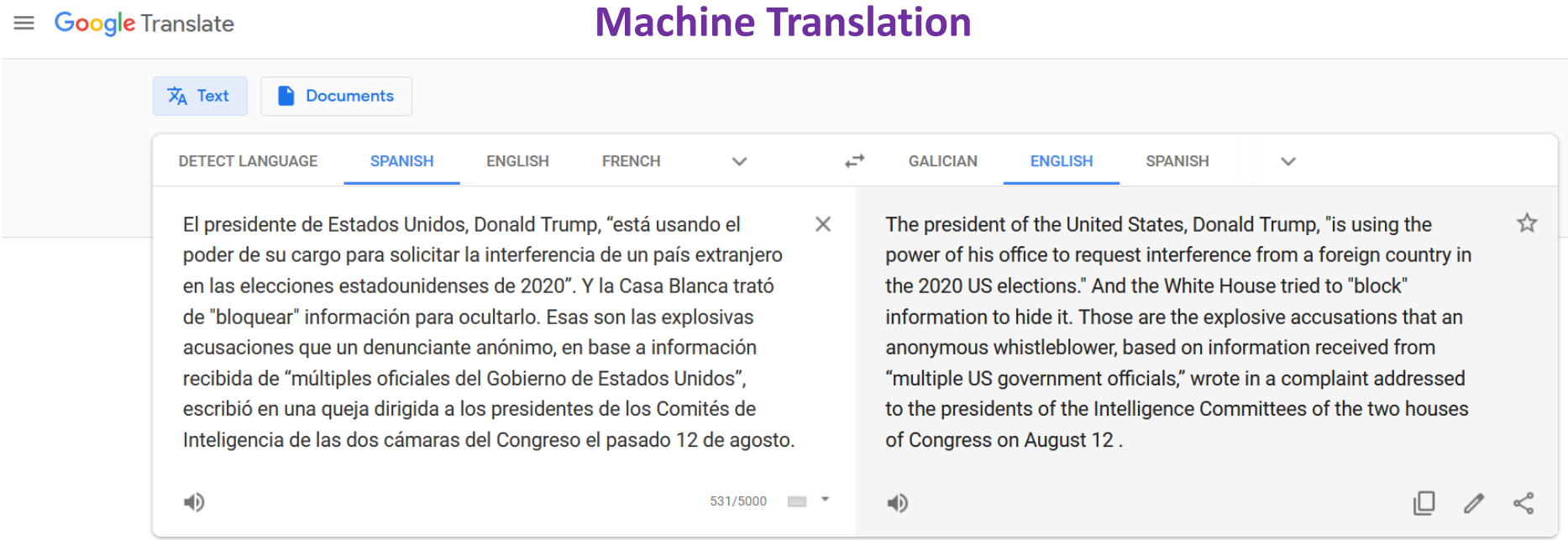


Applications (2)

Automatic Speech Recognition

Image taken from https://www.trzcacak.rs/imgb/iRoJxhx/

Machine Translation



Applications (3): style transfer

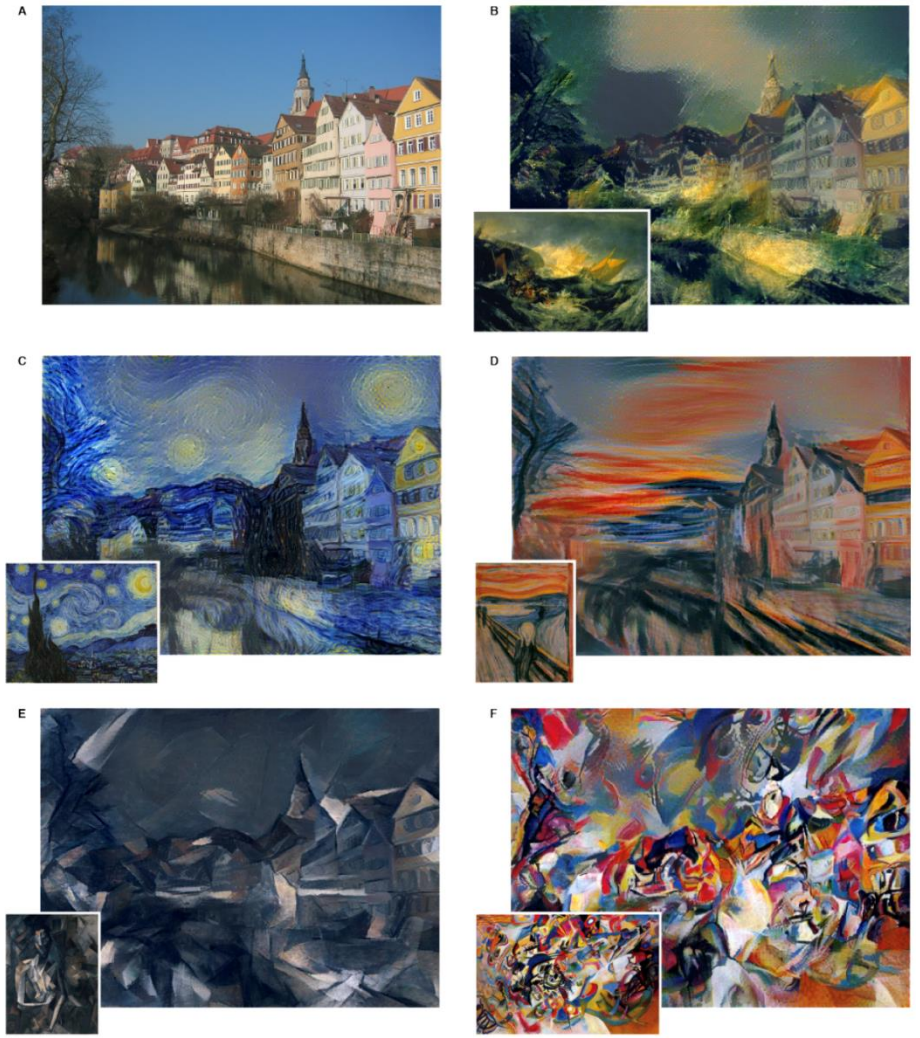


Image taken from “A Neural Algorithm of Artistic Style” (Gatys et al., 2015)

Applications (4): image colorization

Image taken from “Colorful Image Colorization” (Zhang et al., 2016)

Applications (5): image inpainting/reconstruction

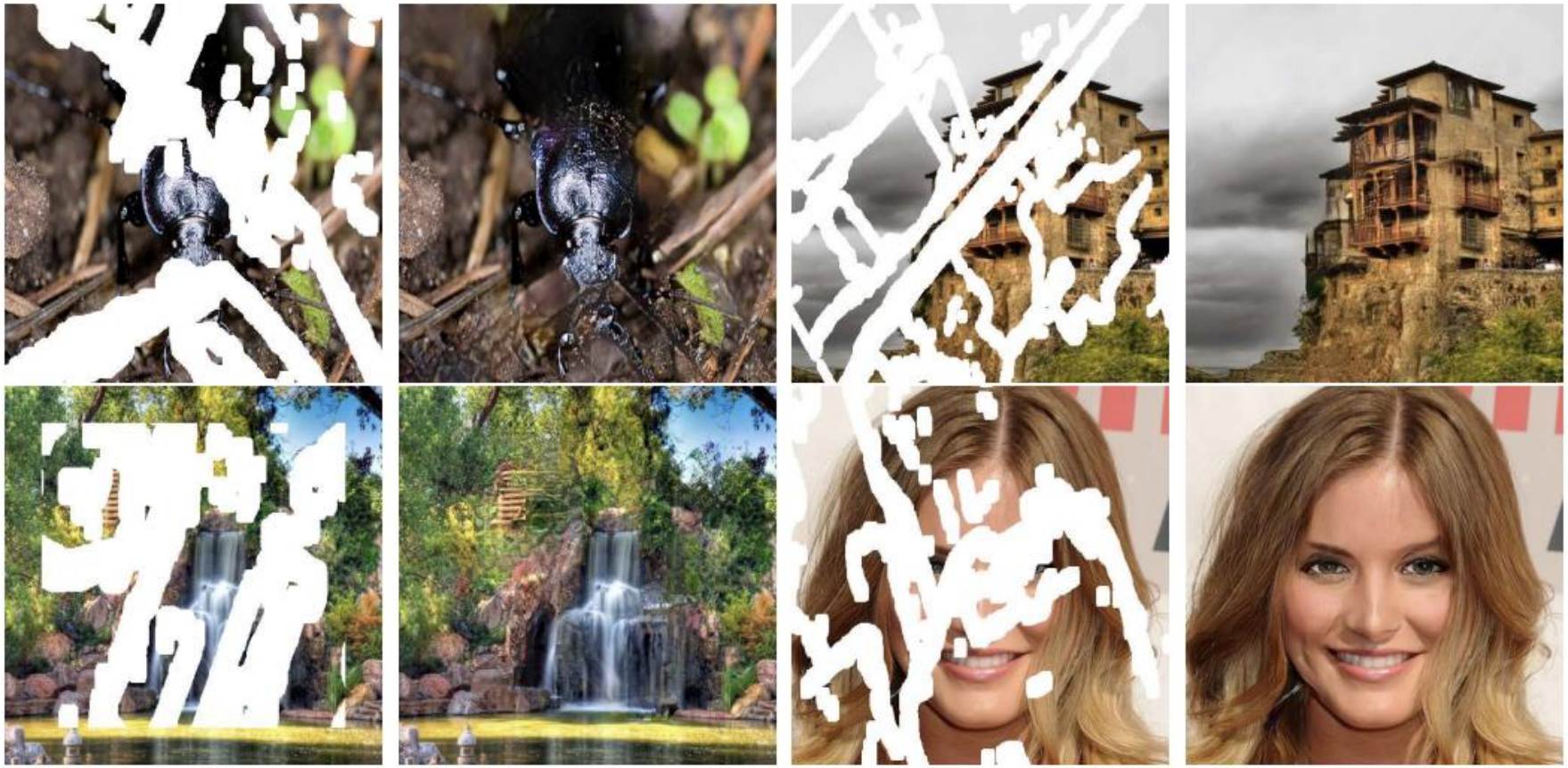


Image taken from “Image Inpainting for Irregular Holes Using Partial Convolutions” (Liu et al., 2018)

Applications (6): image synthesis

Image taken from “Progressive Growing of GANs for Improved Quality, Stability, and Variation” (Karras et al., 2017)”

Applications (and 7): image captioning

"man in black shirt is playing guitar."

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

"two young girls are playing with lego toy."

"boy is doing backflip on wakeboard."

"girl in pink dress is jumping in air."

"black and white dog jumps over bar."

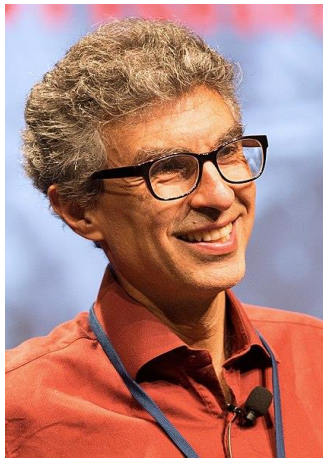
"young girl in pink shirt is swinging on swing."

"man in blue wetsuit is surfing on wave."

Karpathy, A., & Fei-Fei, L. (2015). Deep visual-semantic alignments for generating image descriptions. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (pp. 3128-3137).

Clear success and progress

So evident that three of the field leading researchers received in 2018 the **Turing Award** (“Nobel Prize in Computing Science”)



Yoshua Bengio
University of Montreal

Geoffrey Hinton
University of Toronto & Google

Yann LeCun
New York University & Facebook

Clear success and progress

Clear success and progress

Why now?

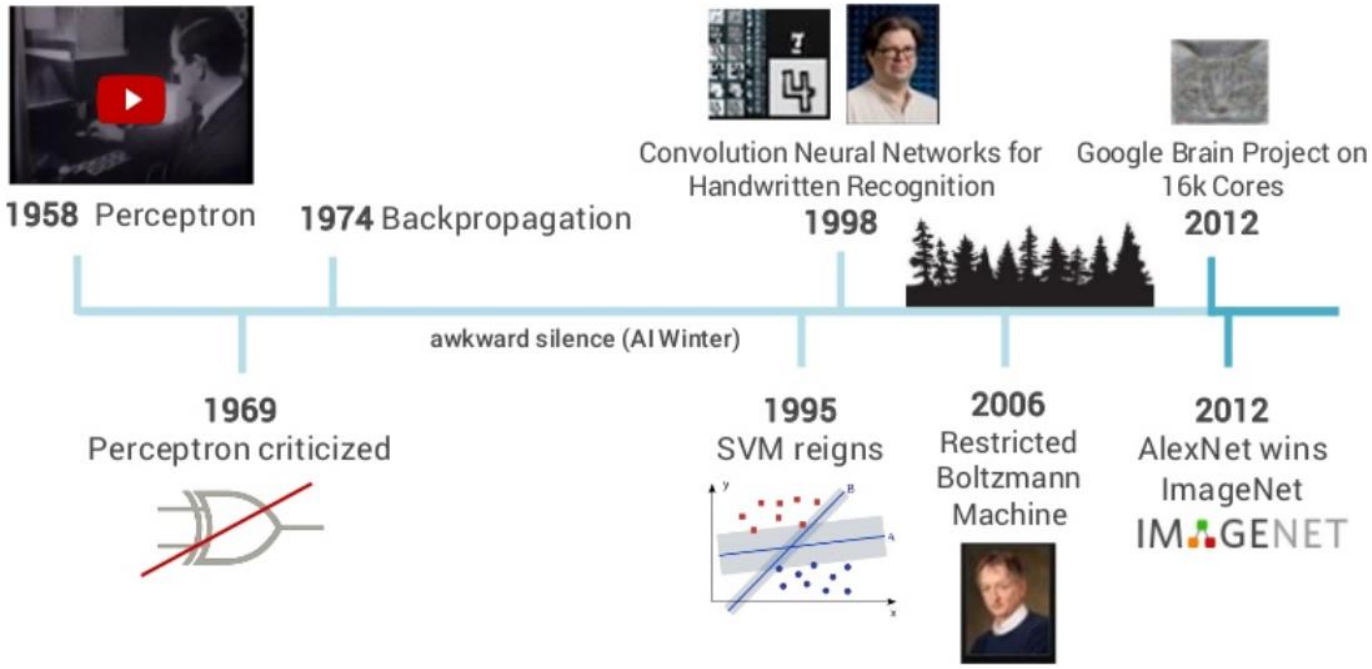
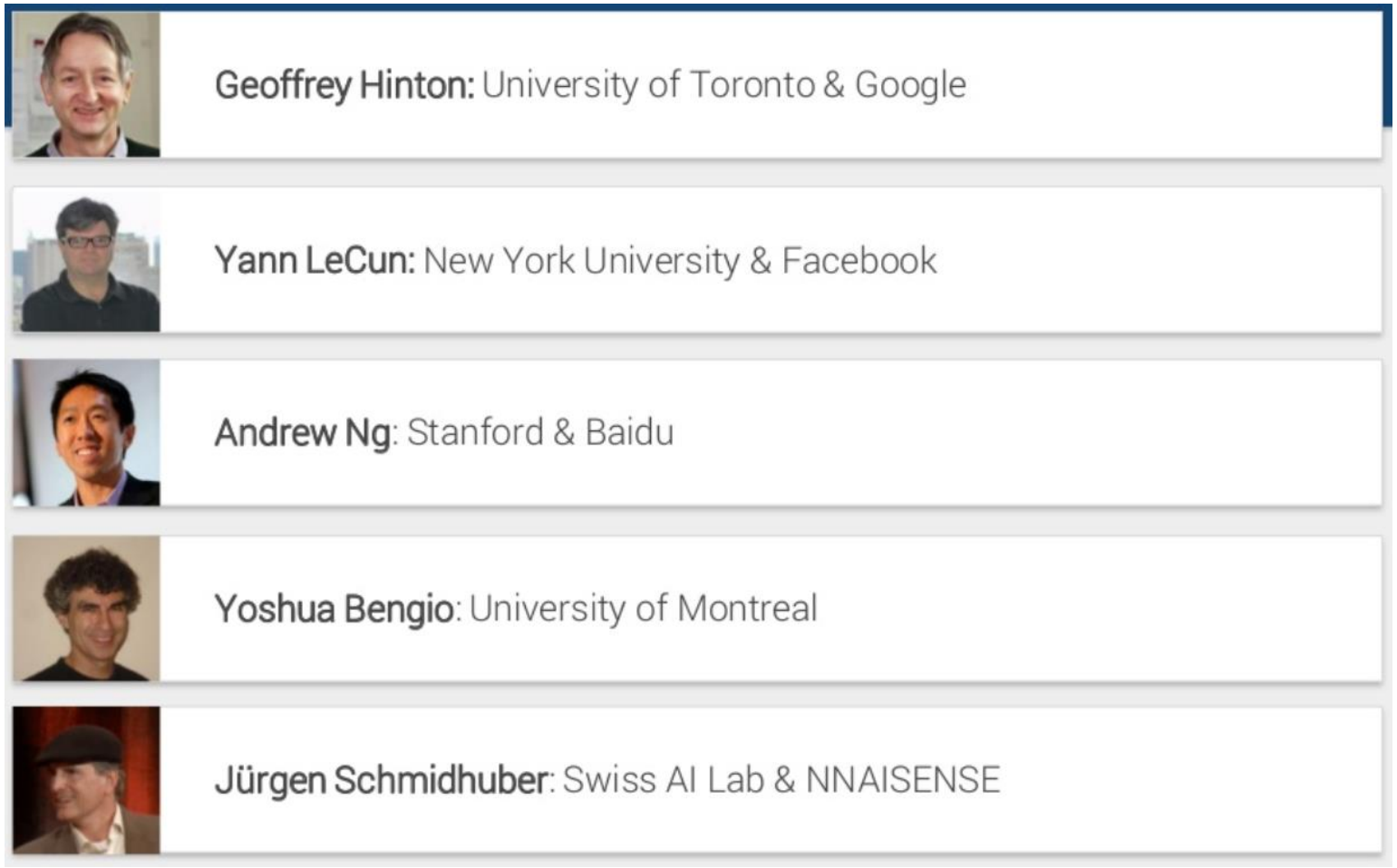


Image taken from https://www.slideshare.net/LuMa921/deep-learning-a-visual-introduction

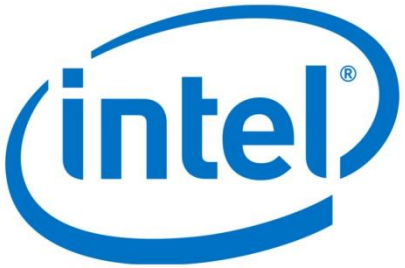
- Explosion since 2012 due to:
  - Huge amounts of labeled data (e.g. ImageNet)
  - High computational power (e.g. GPU-computing)
  - Algorithmic and architectural advances (e.g. ReLU, Dropout, Pretraining)

Main names

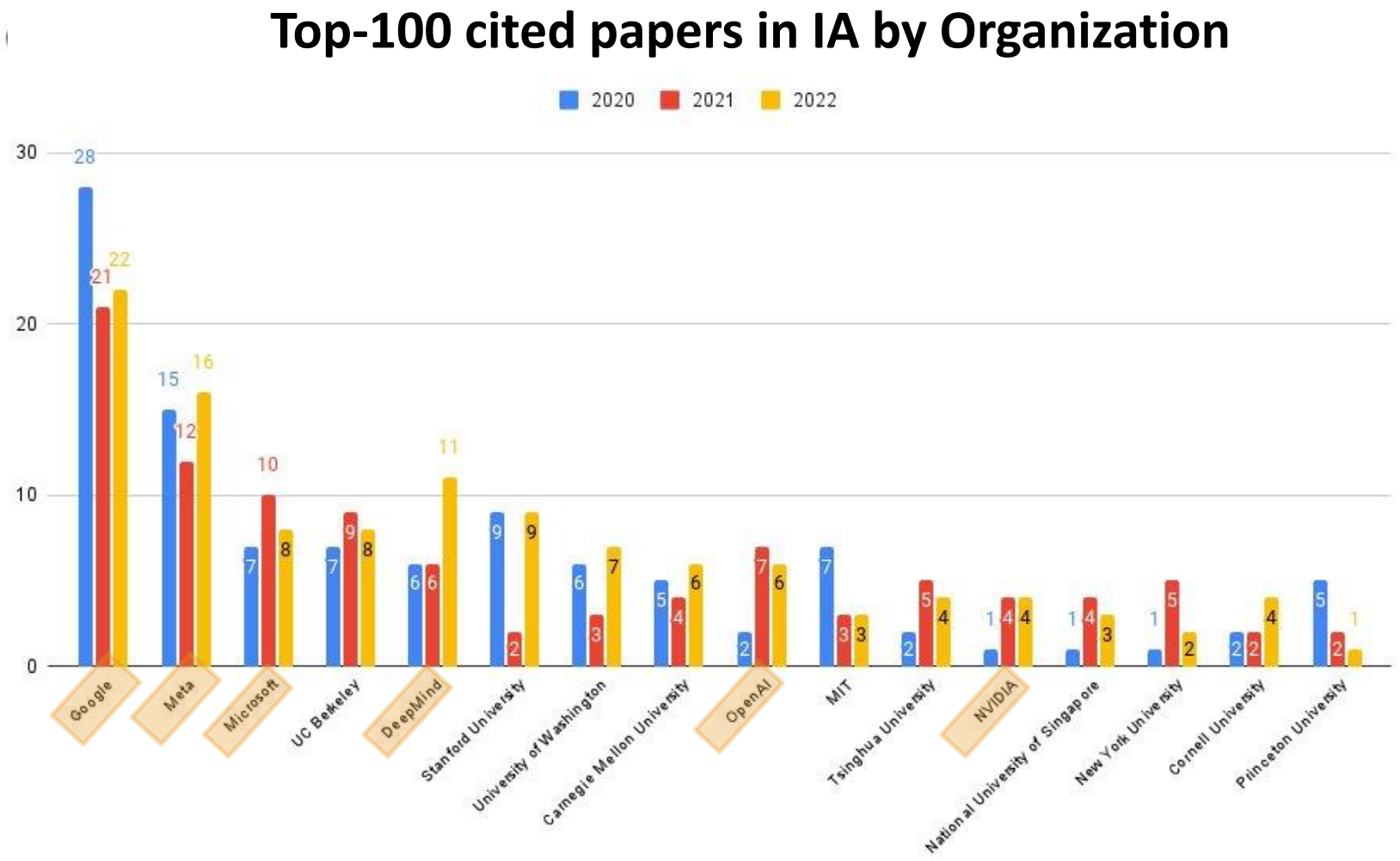


https://www.slideshare.net/LuMa921/deep-learning-a-visual-introduction

Main enterprises



Main enterprises



https://twitter.com/ZetaVector/status/1631590029926494211

2. Convolutional Neural Networks

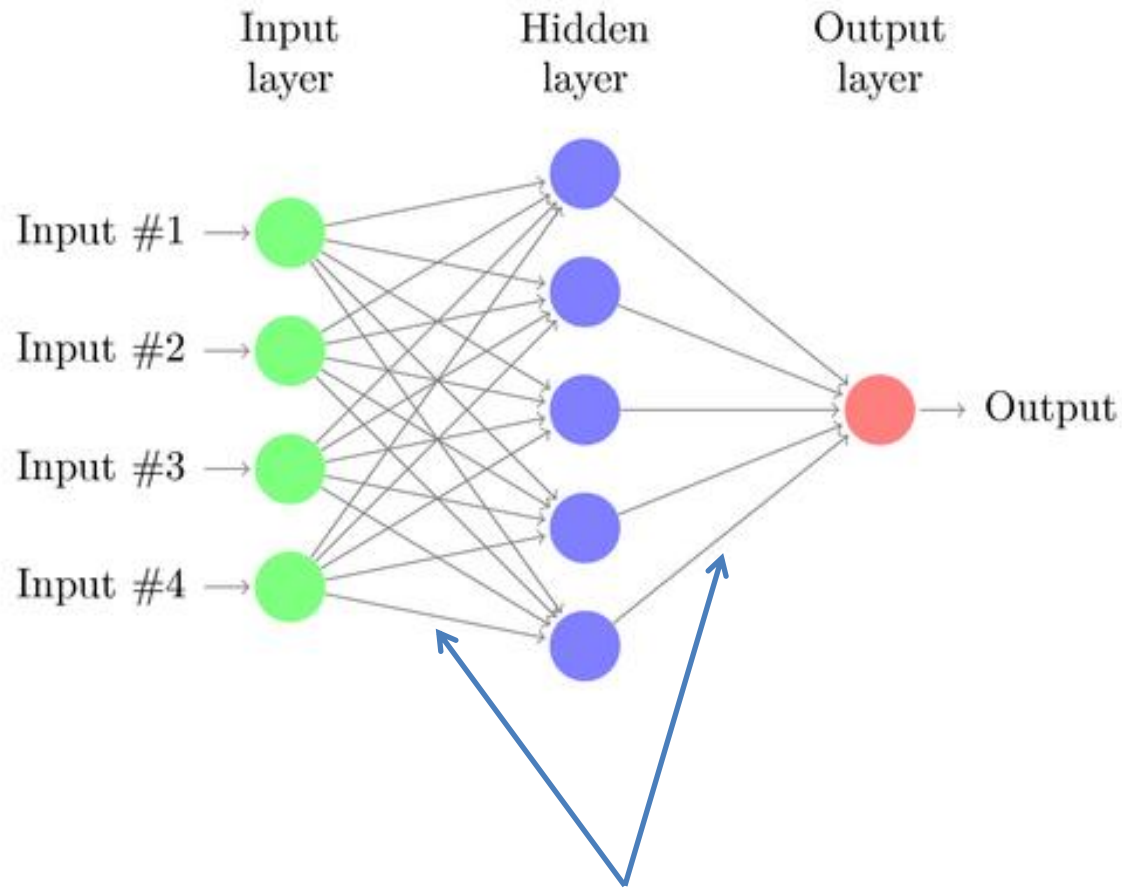
Recommended reading

(part of these slides are taken from this source)

- Highly recommended to consult the Stanford course:
  - CS231n: Convolutional Neural Networks for Visual Recognition (now called Deep Learning for Computer Vision) (http://cs231n.stanford.edu/2018/syllabus.html)

- In particular, section on Convolutional Neural Networks (ConvNets): http://cs231n.stanford.edu/slides/2018/cs231n_2018_lecture05.pdf
  https://cs231n.github.io/convolutional-networks/

Until now…



Shallow neural networks: “few” layers with weights.
In the example on this slide, two.

There were theoretical reasons for this…

Multi-layer perceptron with a single hidden layer of units

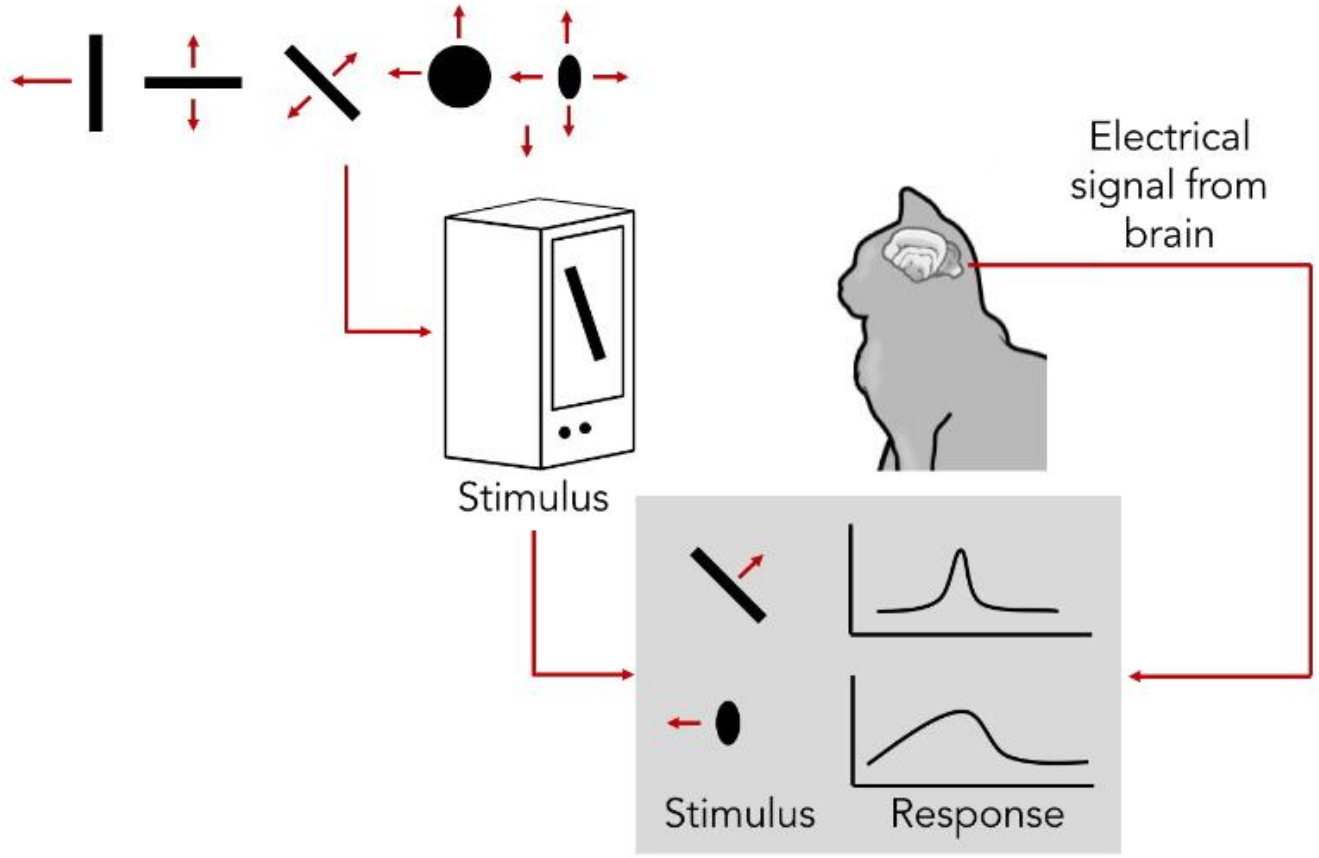
- Networks with two or more layers are universal approximators:
  - Any continuous function can be uniformly approximated to arbitrary accuracy, given enough hidden units.
  - This is true for many definitions of $h(\cdot)$, but excluding polynomials.
  - The key problem is how to find suitable parameter values given a set of training data.

https://project.inria.fr/deeplearning/files/2016/05/session1.pdf

Chapter 5. Bishop, Christopher M. Pattern recognition and machine learning. Springer, 2006.

Origin

- The origin of ConvNets can be found in the Neocognitron (Fukushima, 1980): a model inspired by the work of Hubel and Wiesel (50s and 60s) on the structure and function of the visual cortex.



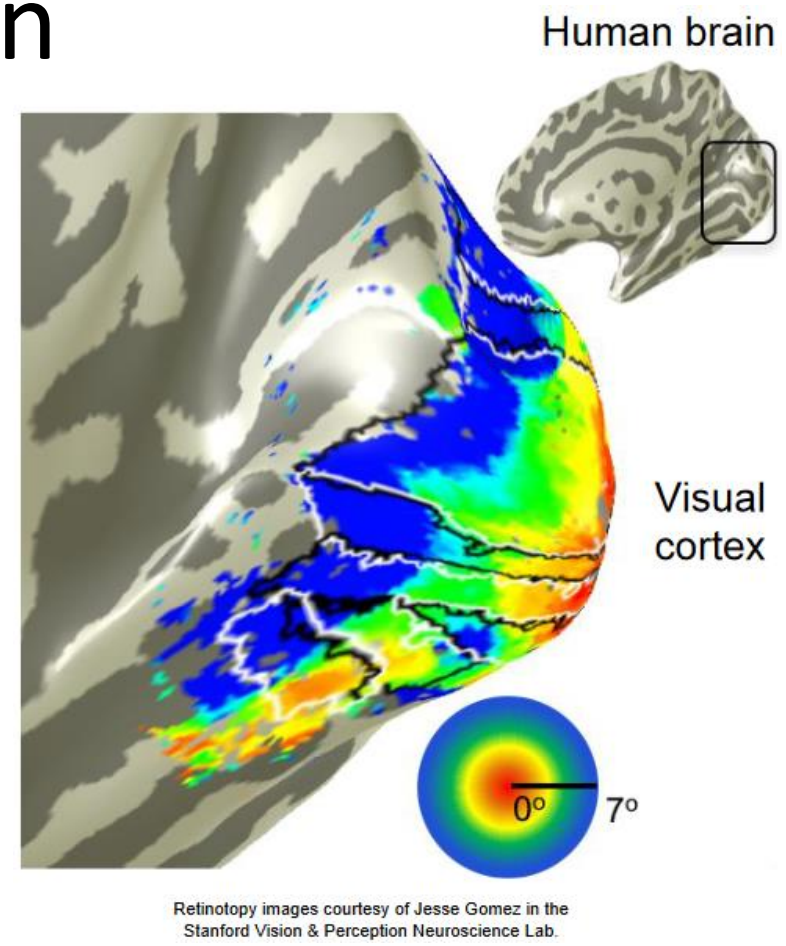
How do visual stimuli affect the activation of a neuron?

Slide credit: Fei-Fei Li, Justin Johnson and Serena Yeung. http://cs231n.stanford.edu/slides/2018/cs231n_2018_lecture05.pdf

Origin

- The visual cortex is topographically mapped:
  - Proximal cells in the cortex represent proximal regions in the visual field.

- There is a **hierarchical organization**:
  - Simple cells: respond to light orientation and position
  - Complex cells: respond to light orientation, motion and direction
  - Hypercomplex (or end-stopped) cells: respond to orientation, motion, direction and length.



Retinotopy images courtesy of Jesse Gomez in the Stanford Vision & Perception Neuroscience Lab.

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

Definition

- What is a Convolutional Neural Network (ConvNet or CNN)?
  - A neural network with a convolution in, at least, one of its layers.
  - Used in problems where information is presented in matrix/grid format (e.g. images)
- Typical architecture of a ConvNet:

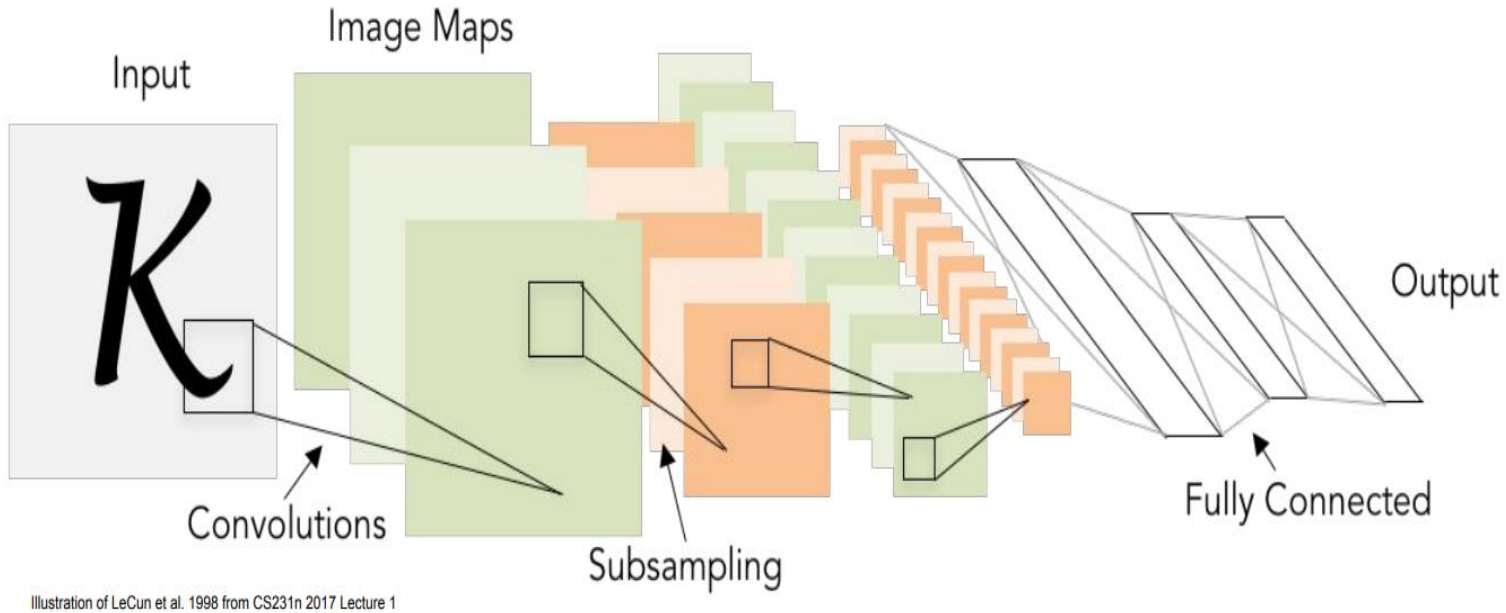


Illustration of LeCun et al. 1998 from CS231n 2017 Lecture 1

What is a convolution?

- **Local linear operation** with a mask.
  - The coefficients/values of the mask/filter determine the operation performed.

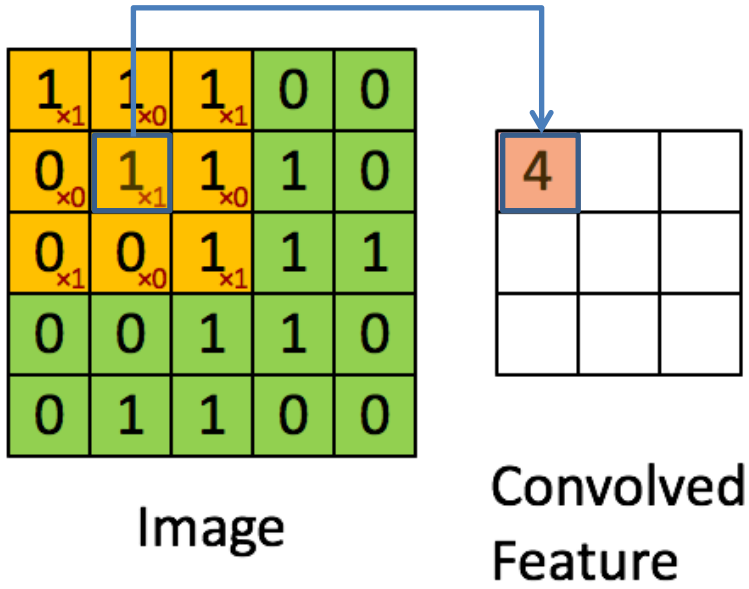
- Technically, a convolution is a cross-correlation where the filter/mask is rotated 180 degrees.
  - Unlike correlation, the convolution verifies (among others) the associative property, which allows us to build complex filters from simple ones.

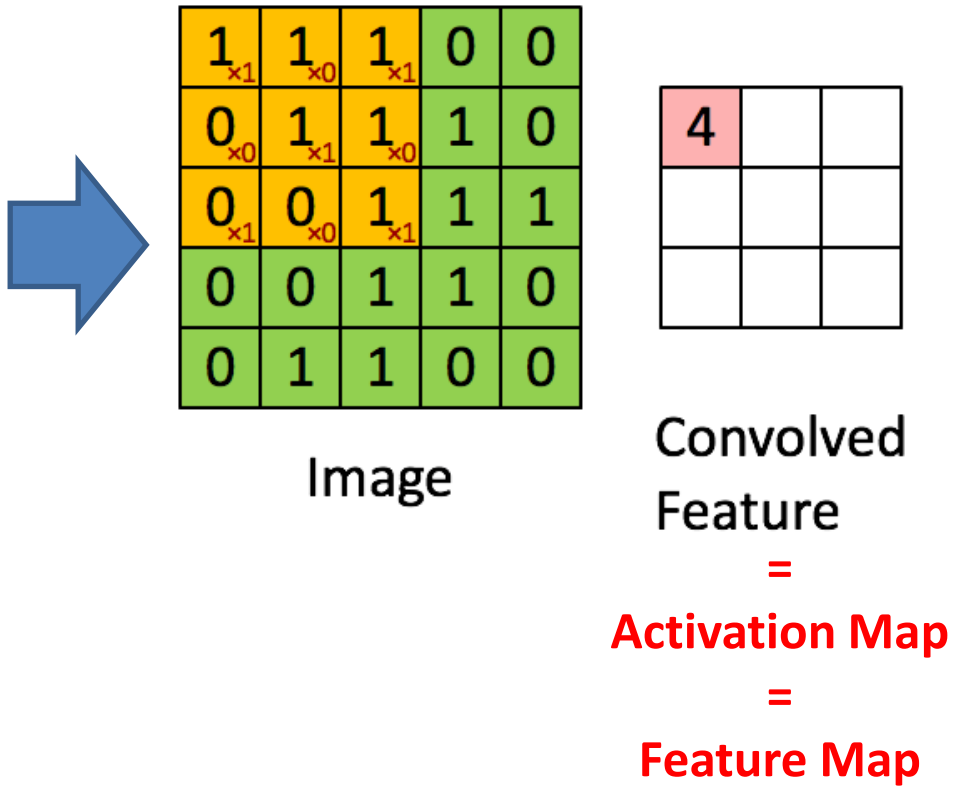
What is a convolution?

- **Local linear operation** with a mask.

The red numbers represent the values/coefficients of the filter/mask.

The filter is multiplied element-by-element with the image, the products are added, and the central position of the filter in the image is substituted.

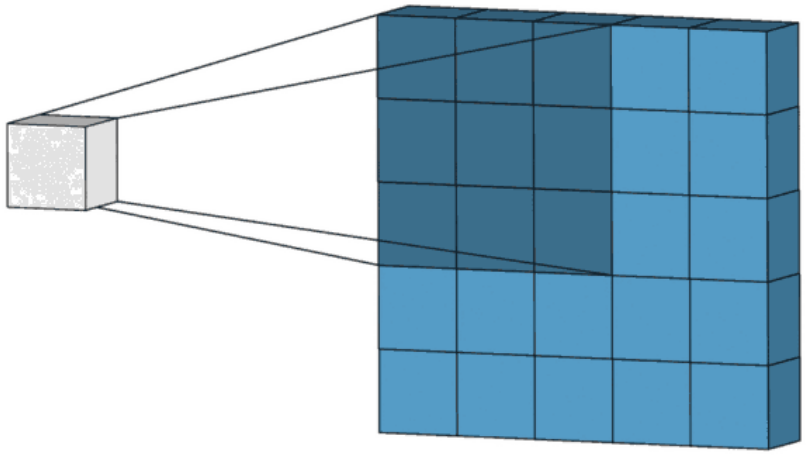


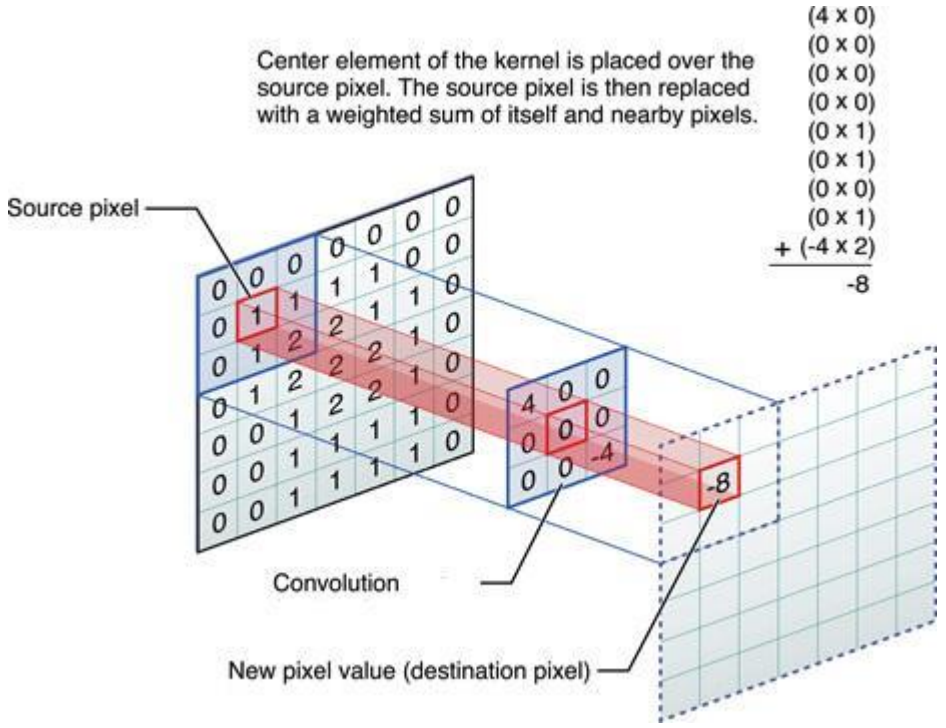


Images taken from https://glassboxmedicine.com/2019/07/26/convolution-vs-cross-correlation/

What is a convolution?

- Another way of looking at it…





https://towardsdatascience.com/intuitively-understanding-convolutions-for-deep-learning-1f6f42faee1

https://medium.com/@bdhuma/6-basic-things-to-know-about-convolution-daef5e1bc411

What is a convolution?

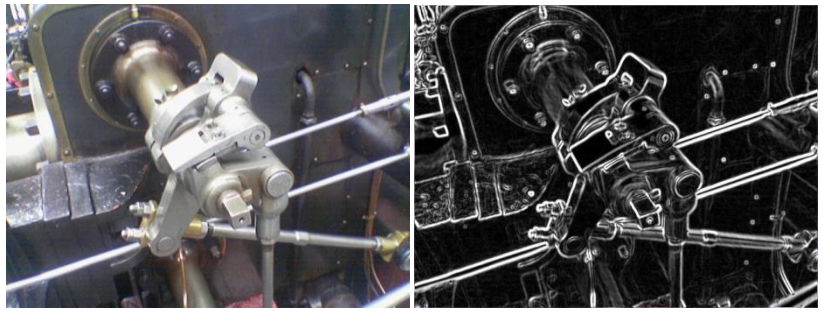
- **Local linear operation** with a mask.
  - The values of the mask determine the result (features to be extracted)

Gaussian filter (removes high frequencies → image smoothing)

$$\frac{1}{273}\begin{bmatrix} 1 & 4 & 7 & 4 & 1 \\ 4 & 16 & 26 & 16 & 4 \\ 7 & 26 & 41 & 26 & 7 \\ 4 & 16 & 26 & 16 & 4 \\ 1 & 4 & 7 & 4 & 1 \end{bmatrix}$$

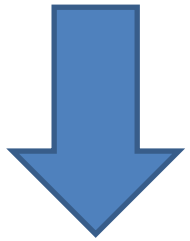
Sobel filter (removes low frequencies → edge enhancement)

$$\mathbf{G}_x = \begin{bmatrix} +1 & 0 & -1 \\ +2 & 0 & -2 \\ +1 & 0 & -1 \end{bmatrix} \quad \mathbf{G}_y = \begin{bmatrix} +1 & +2 & +1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}$$



Key idea

In ConvNets, these values/coefficients are learned! They are not selected by a human expert. Now, they are free parameters of the network and are trained like any other weight.



1989: LeCun et al. used **back-propagation to directly learn the coefficients of convolutional filters** from images of handwritten numbers.

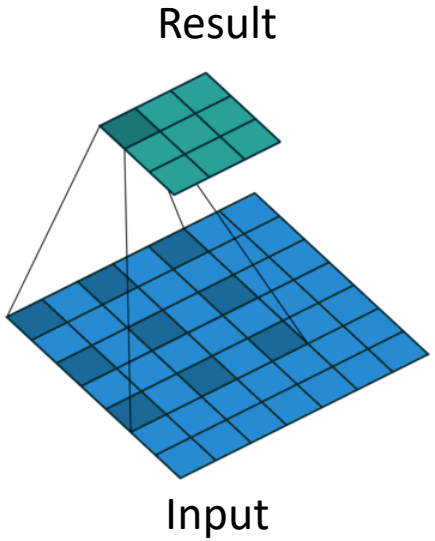
1998: LeCun et al. presented **LeNet-5**, ConvNet with 5 layers that could authomatically recognize handwritten numbers, and showed that **ConvNets outperformed all other techniques in this task**.

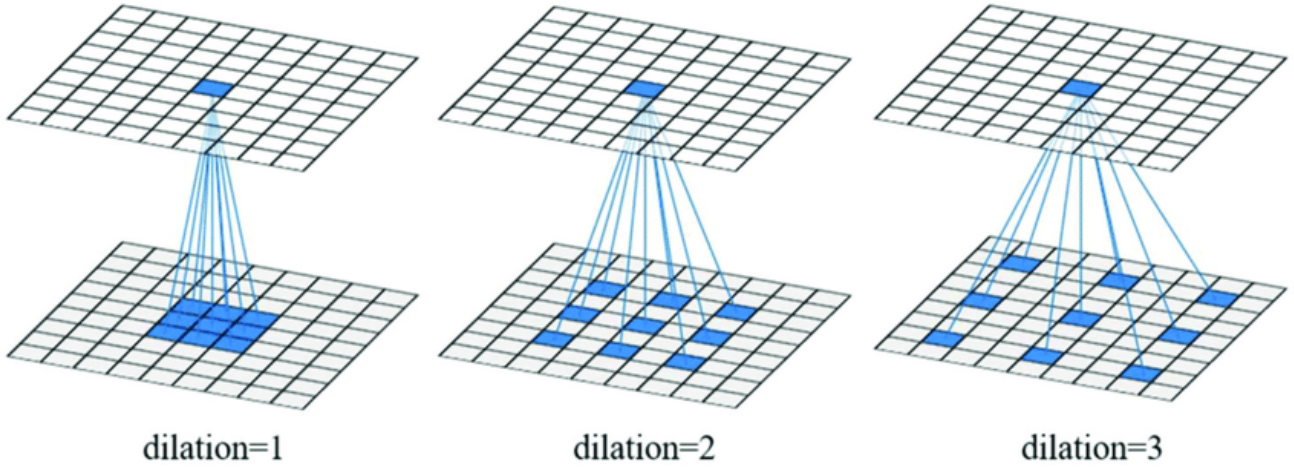
LeCun, Y., Boser, B., Denker, J. S., Henderson, D., Howard, R. E., Hubbard, W., & Jackel, L. D. (1989). Backpropagation applied to handwritten zip code recognition. Neural computation, 1(4), 541-551.

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

Other ways of convolution

- So far we have seen the classical/canonical form of convolution, but there are others. It is worth, at least, know about their existence.
- Dilated convolutions (atrous convolutions).
  - Popular in image segmentation.
  - A *dilation rate*, which defines a spacing between kernel/filter values, is established.
  - The receptive field is widen without increasing the computational cost.

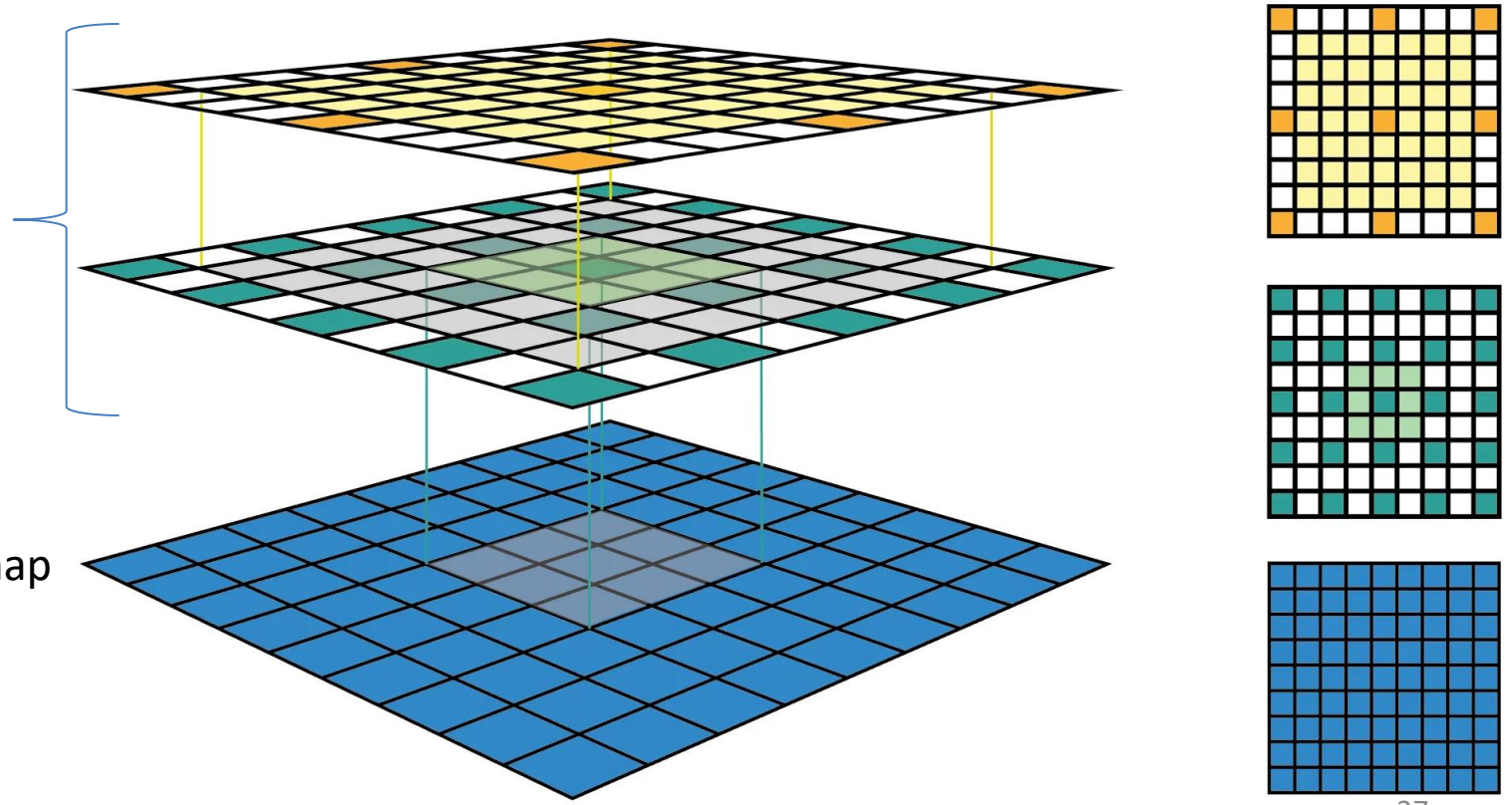




What is the receptive field?

– The region of the input that affects to a particular unit/*feature* of the neural network.

  • In other words, the region in the input space that a particular ConvNet feature is looking at.

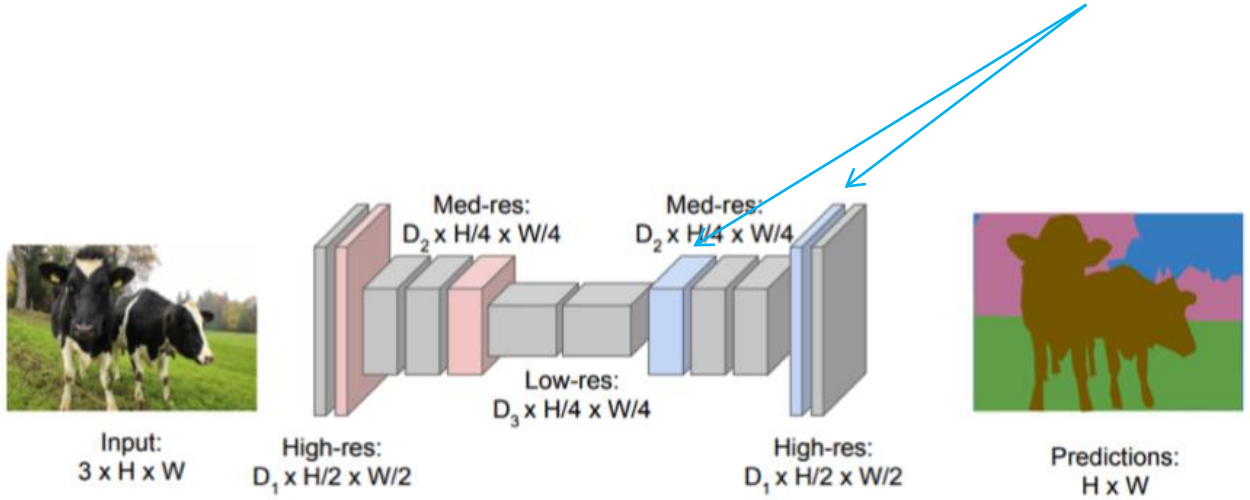


"large receptive fields are necessary for high-level recognition tasks"
https://distill.pub/2019/computing-receptive-fields/

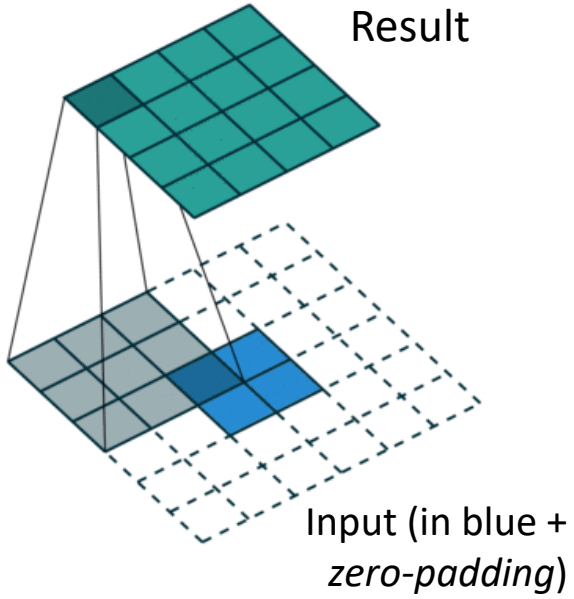
https://blog.mlreview.com/a-guide-to-receptive-field-arithmetic-for-convolutional-neural-networks-e0f514068807

Other ways of convolution

- Transposed convolutions (sometimes, erroneously referred to as *deconvolution*)
  - Popular in super-resolution and segmentation (in the *up-sampling* part) of images.



https://towardsdatascience.com/transposed-convolution-demystified-84ca81b4baba



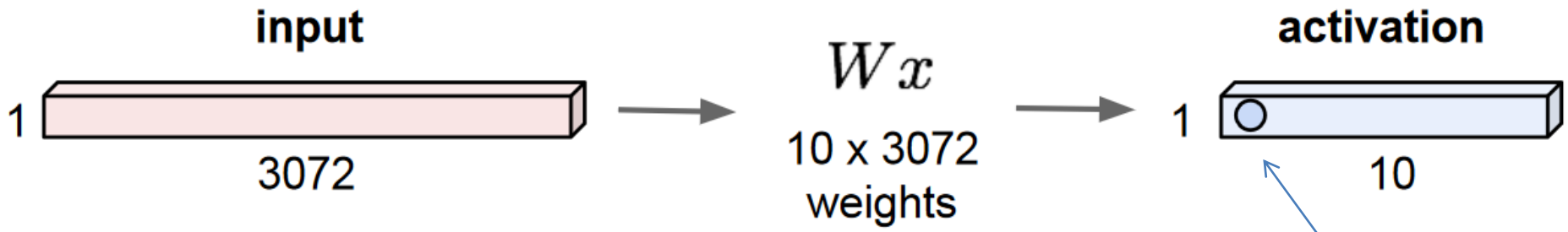
Interesting reading:

Vincent Dumoulin, Francesco Visin - A guide to convolution arithmetic for deep learning

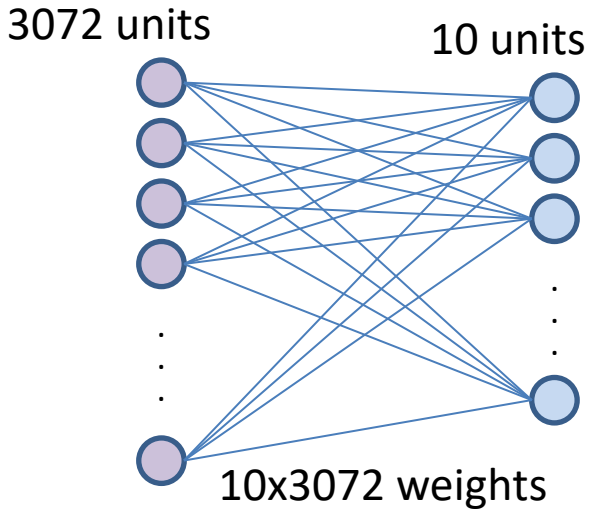
Here more visualizations: https://github.com/vdumoulin/conv_arithmetic

Fully-connected Layer

All units/neurons of a layer are connected (weights) with all units/neurons in the next layer.

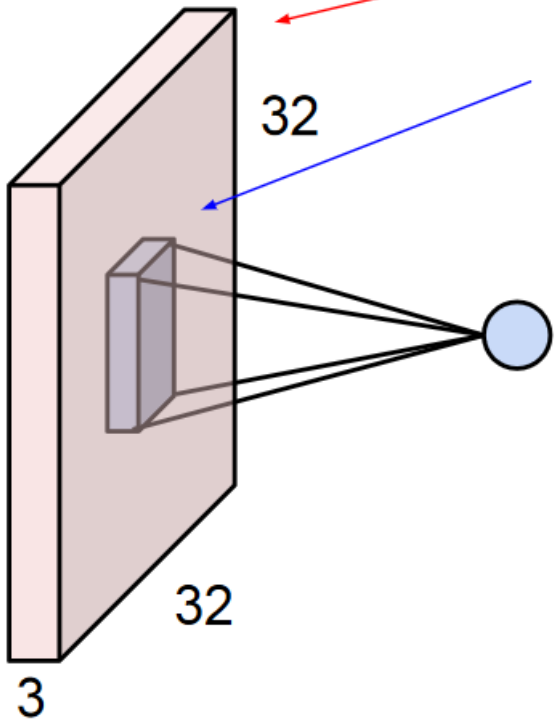


Other way of representing it:



1 number: result of applying the *dot product* between a row of W and the input

Convolutional Layer



32x32x3 image

5x5x3 filter w

The 3072 elements we saw before!

1 number:
the result of taking a dot product between the filter and a small 5x5x3 chunk of the image
(i.e. 5*5*3 = 75-dimensional dot product + bias)

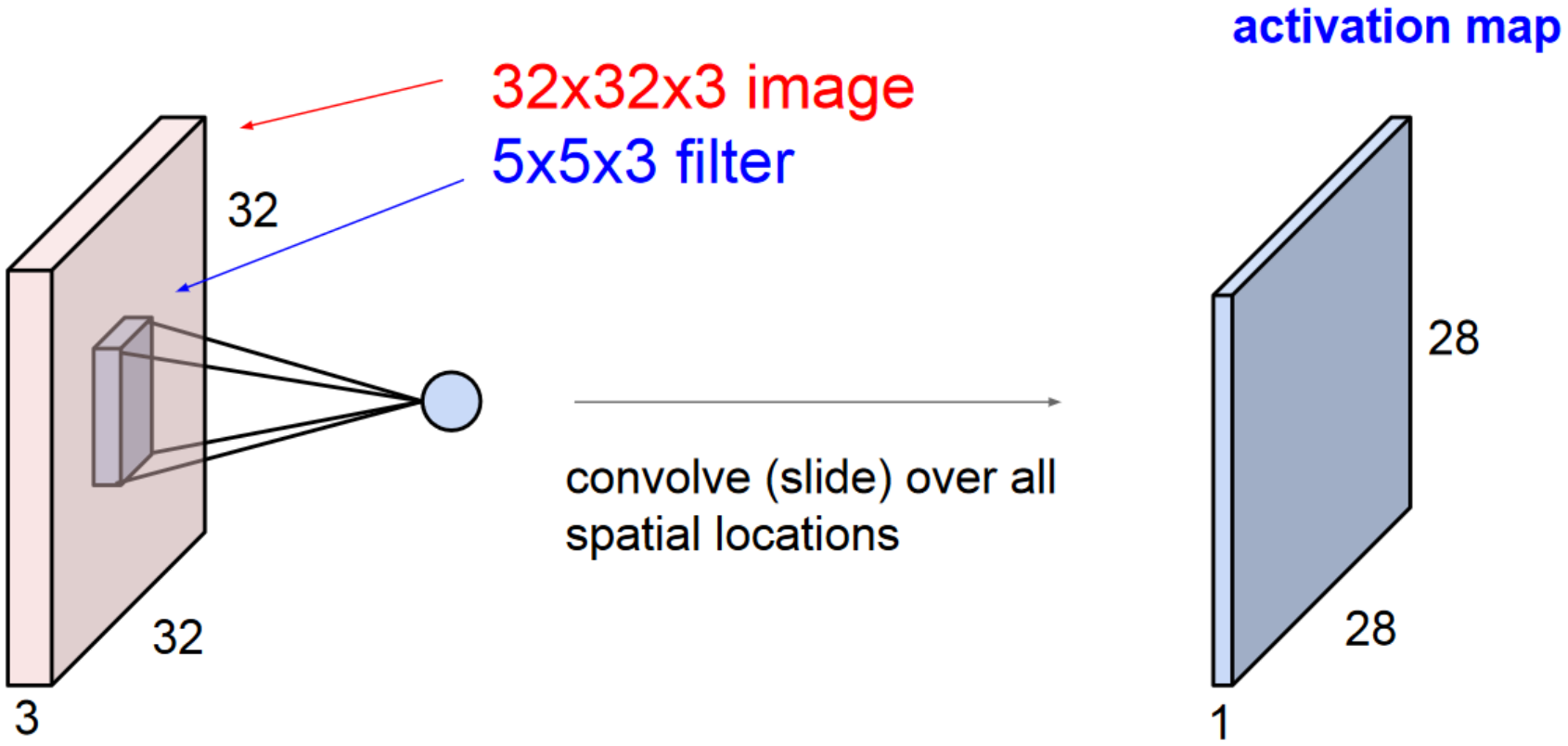
$$w^T x + b$$

Great advantage:
76 weights vs 30720!

Note: *bias* brings flexibility to learning (richer representations of the input space), allowing to translate/adapt the output of the unit/neuron. Simple and intuitive example: https://stackoverflow.com/questions/2480650/what-is-the-role-of-the-bias-in-neural-networks

Slide credit: Fei-Fei Li, Justin Johnson and Serena Yeung. http://cs231n.stanford.edu/slides/2018/cs231n_2018_lecture05.pdf

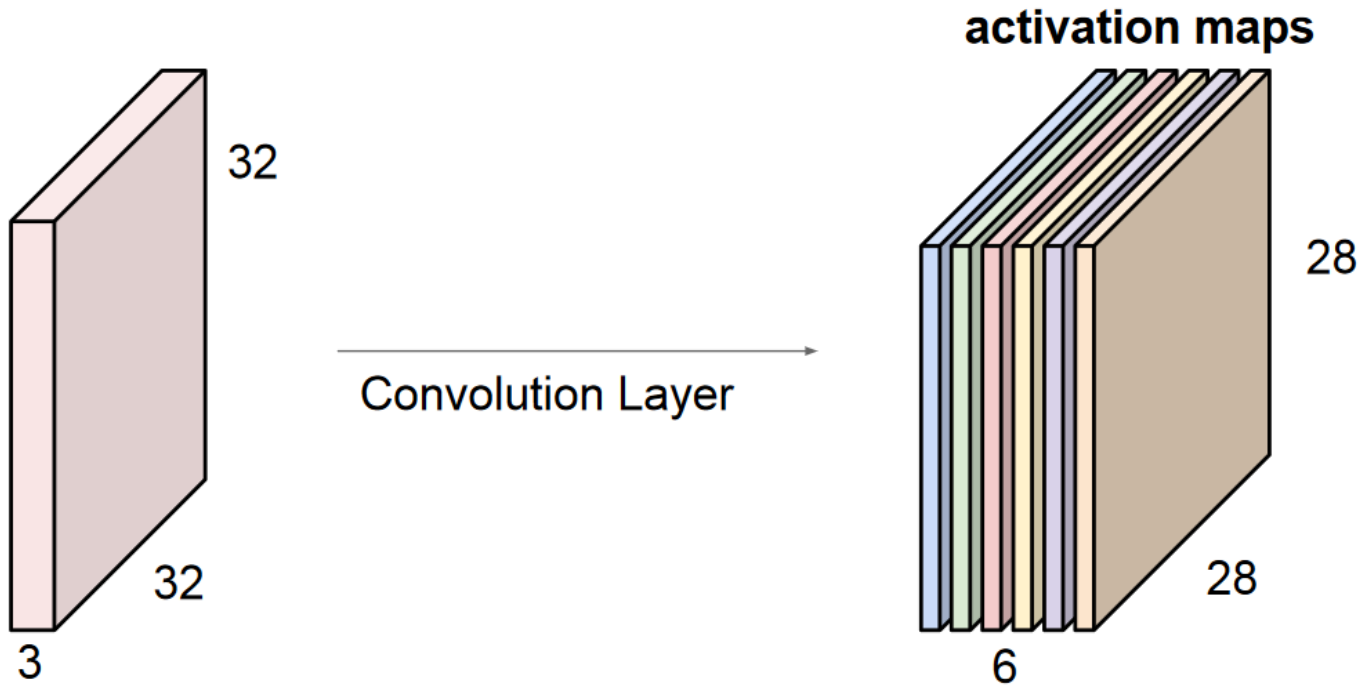
Convolutional Layer



Slide credit: Fei-Fei Li, Justin Johnson and Serena Yeung. http://cs231n.stanford.edu/slides/2018/cs231n_2018_lecture05.pdf

Convolutional Layer

For example, if we had 6 5x5 filters, we'll get 6 separate activation maps:



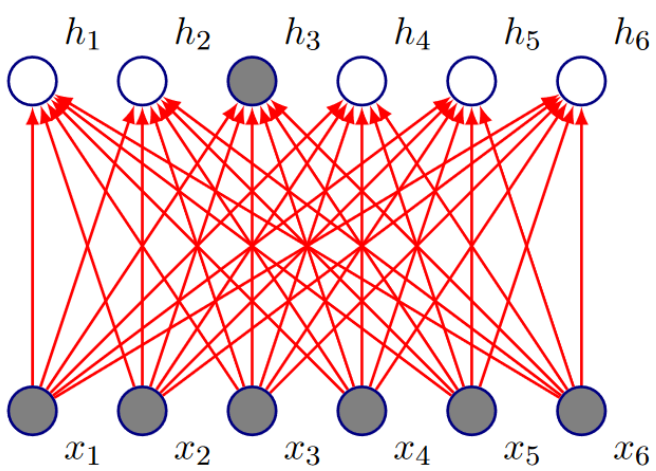
We stack these up to get a "new image" of size 28x28x6!

Even now the number of weights is much lower than what we had before:
456 weights vs 30720!

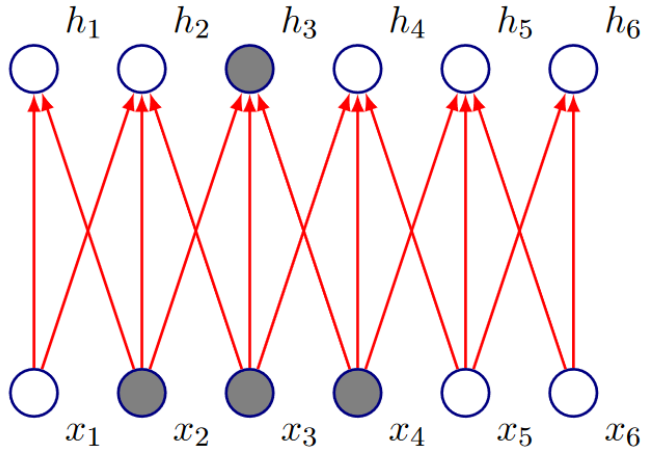
Slide credit: Fei-Fei Li, Justin Johnson and Serena Yeung. http://cs231n.stanford.edu/slides/2018/cs231n_2018_lecture05.pdf

Advantages of ConvNets

- **Sparse connectivity**
  - *Fully-connected layers* operate globally (each neuron sees the complete input), while *convolutional layers* operate locally → Fewer operations!



vs



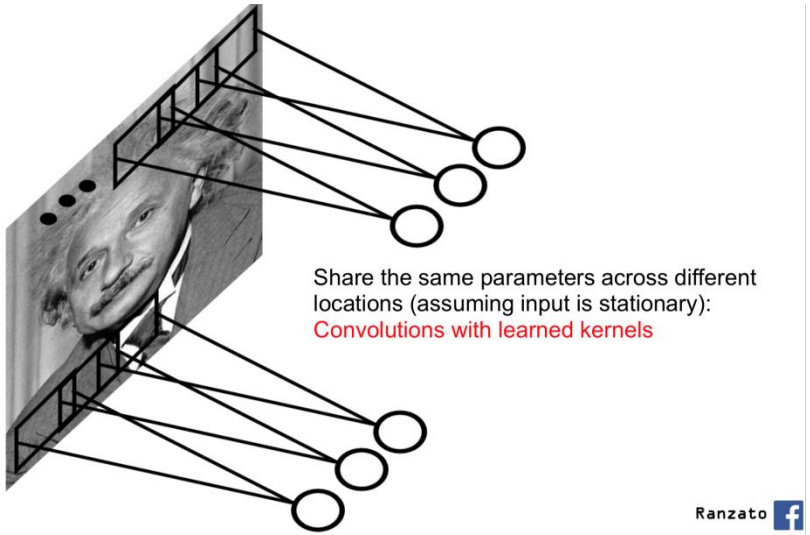
- Fully connected network: h_3 is computed by full matrix multiplication with no sparse connectivity

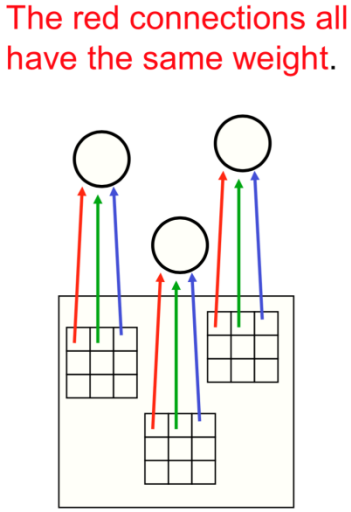
- Kernel of size 3, moved with stride of 1
- h_3 only depends on x_2, x_3, x_4

https://ttic.uchicago.edu/~shubhendu/Pages/Files/Lecture7_pauses.pdf

Advantages of ConvNets

- **Weight/Parameter Sharing**
  - The same filter is applied to the whole image.
    - Instead of learning a parameter for each localization in the image, only a reduced set of parameters (those corresponding to the filter) are learned.
    - The number of parameters to learn and store is largely reduced! → Regularization!





https://www.cs.toronto.edu/~lczhang/aps360_20191/lec/w03/convnet.html

Advantages of ConvNets

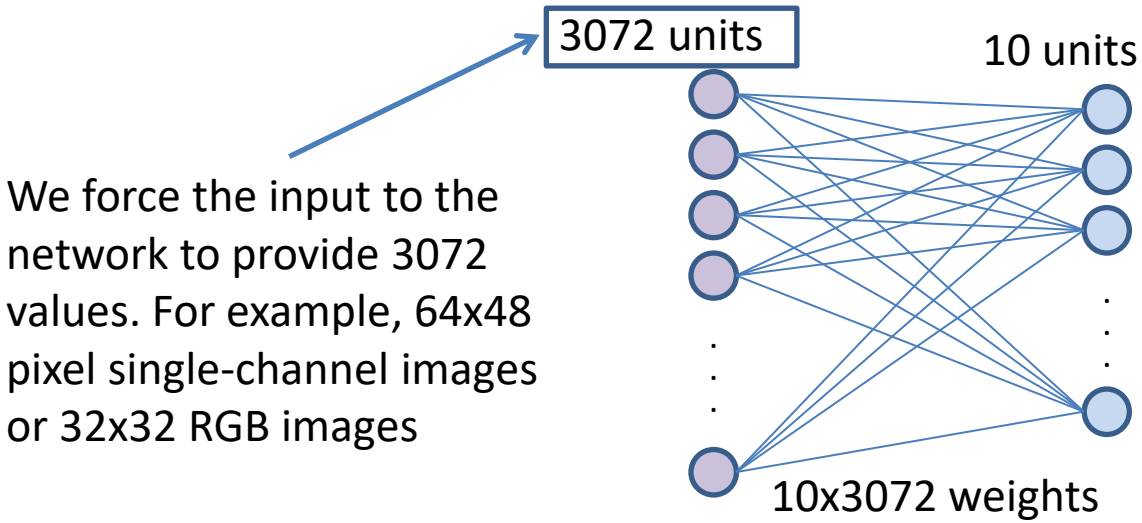
- **Equivariant representations**
  - Each convolutional layer is equivariant with respect to translation.

f is equivariant with respect to g if $f(g(\boldsymbol{x})) = g(f(\boldsymbol{x}))$

  - if we translate an object in the image, its representation will be translated the same distance in the output.
    - It allows to generalize the detection of edges, textures and shapes at different locations in the image!
  - Convolution is not equivariant with respect to scale or rotation.

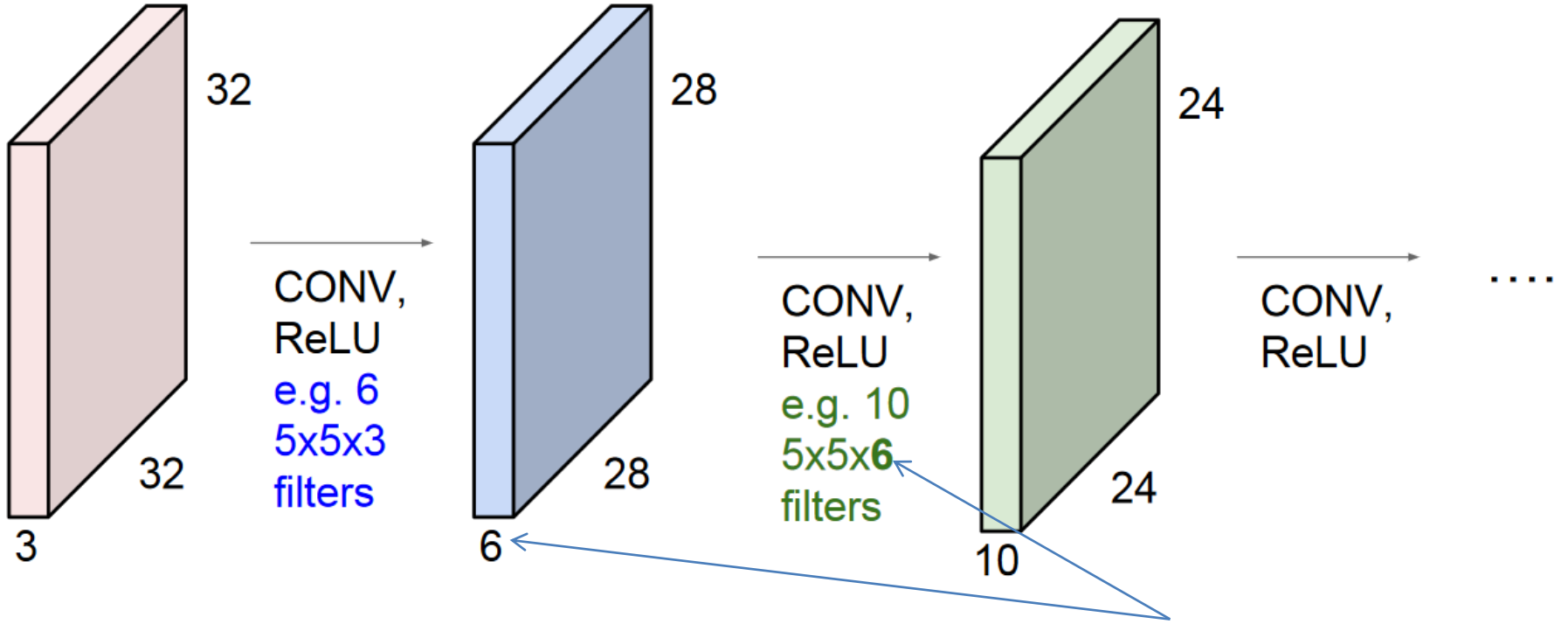
Advantages of ConvNets

- Ability to **operate on variable size inputs**
  - A *fully-connected layer* forces the input to have a certain size.



Convolutional Networks

- ConvNet is a sequence of Convolutional Layers, interspersed with activation functions.

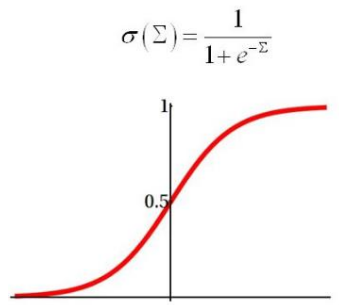


Notice that the filters always have the same depth as the previous convolutional block, i.e. they extend the full depth of the input volume/tensor.

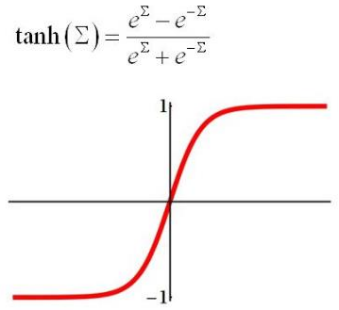
http://cs231n.stanford.edu/slides/2018/cs231n_2018_lecture05.pdf

Convolutional Networks

- Convolutional layers are interspersed with **(no linear)** activation functions.
  - Traditionally, the activation functions were sigmoids, such as the *logistic sigmoid* or the hyperbolic tangent.

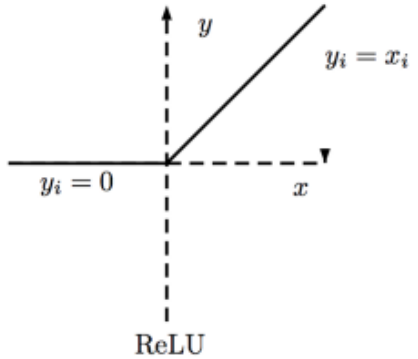


logistic (sigmoid, unipolar)

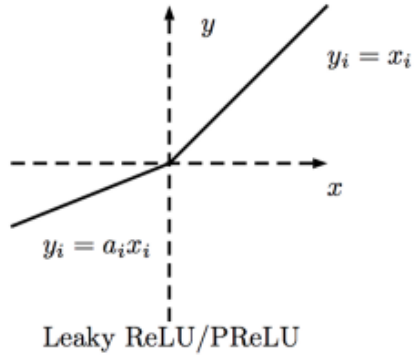


tanh (bipolar)

  - In deep networks it is generally recommended to use the rectified linear unit (ReLU), and its variants, among other reasons because it allows a faster training.



ReLU



Leaky ReLU/PReLU

See dying ReLU problem and vanishing gradient problem
("Yes, you should understand backprop" by Andrej Karpathy)

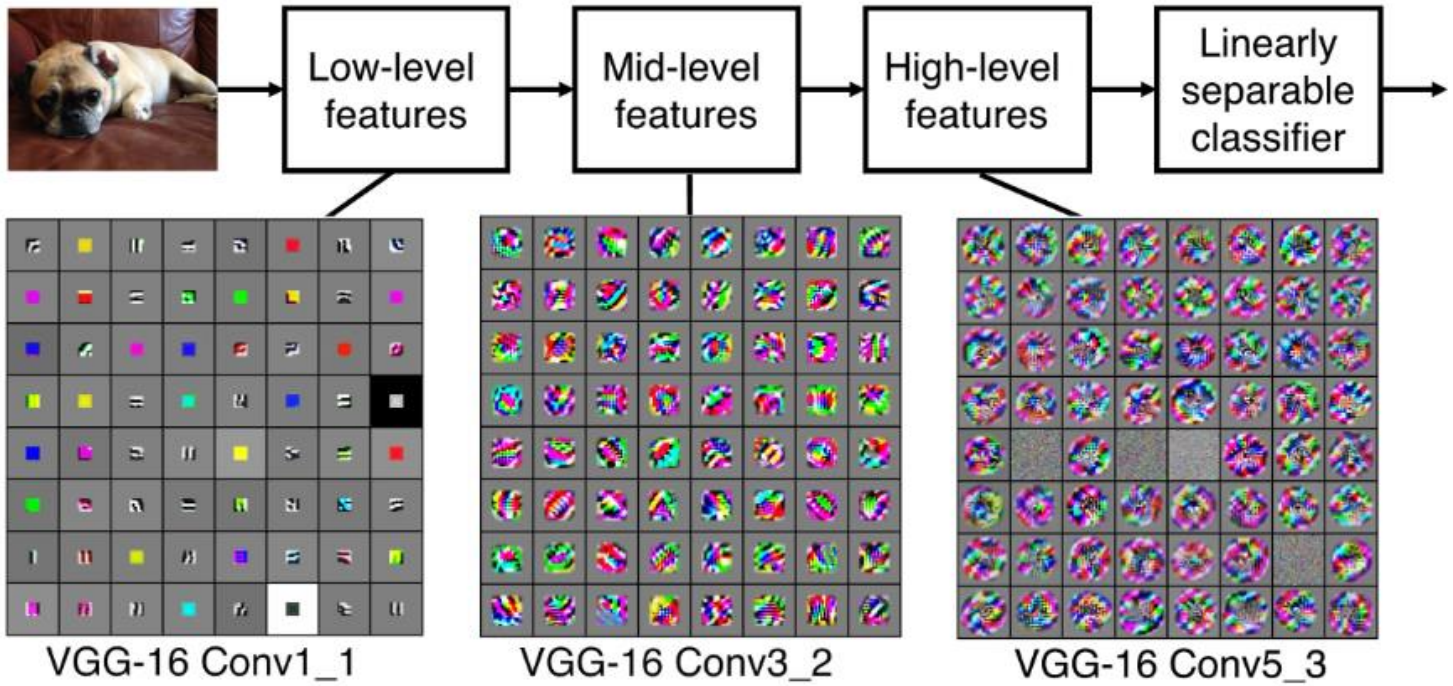
Convolutional Networks

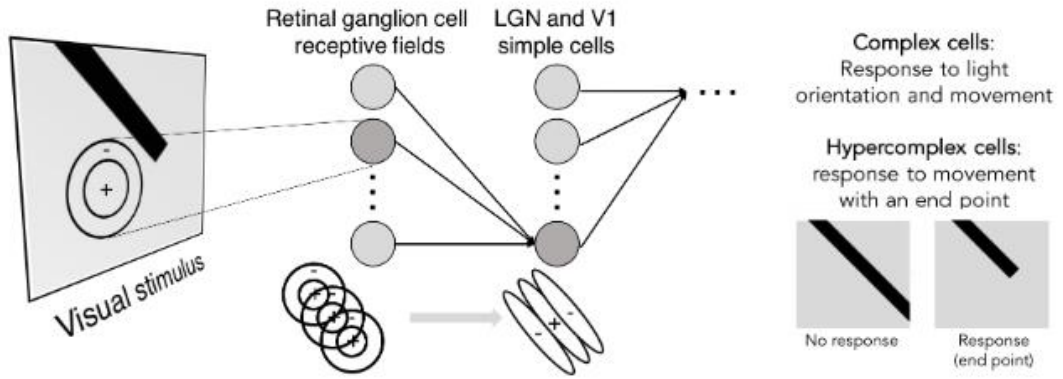
- Convolutional layers are interspersed with **(no linear)** activation functions.
  - This non-linearity is necessary to learn complex representations of the input data.
  - Otherwise, if only linear activation functions were used, the neural network would behave like a linear function.

If the activation functions of all the hidden units in a network are taken to be linear, then for any such network we can always find an equivalent network without hidden units.

"Pattern Recognition and Machine Learning" (C. M. Bishop, 2016) p.229

Convolutional Networks

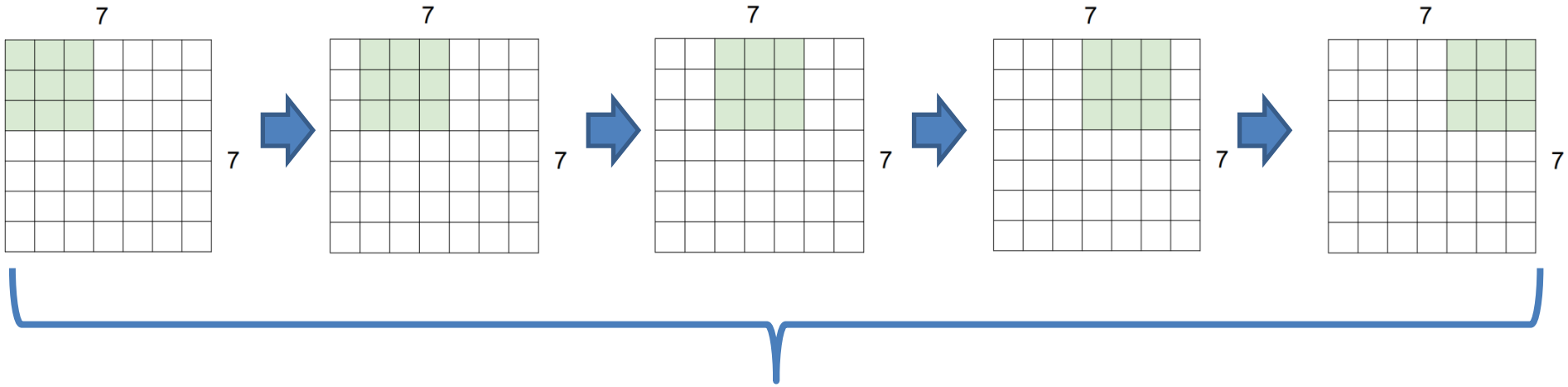




Slide credit: Fei-Fei Li, Justin Johnson and Serena Yeung. http://cs231n.stanford.edu/slides/2018/cs231n_2018_lecture05.pdf

Convolutional Networks

- Lets pay attention now to the dimensions
  - Example: 7x7 image and 3x3 filter

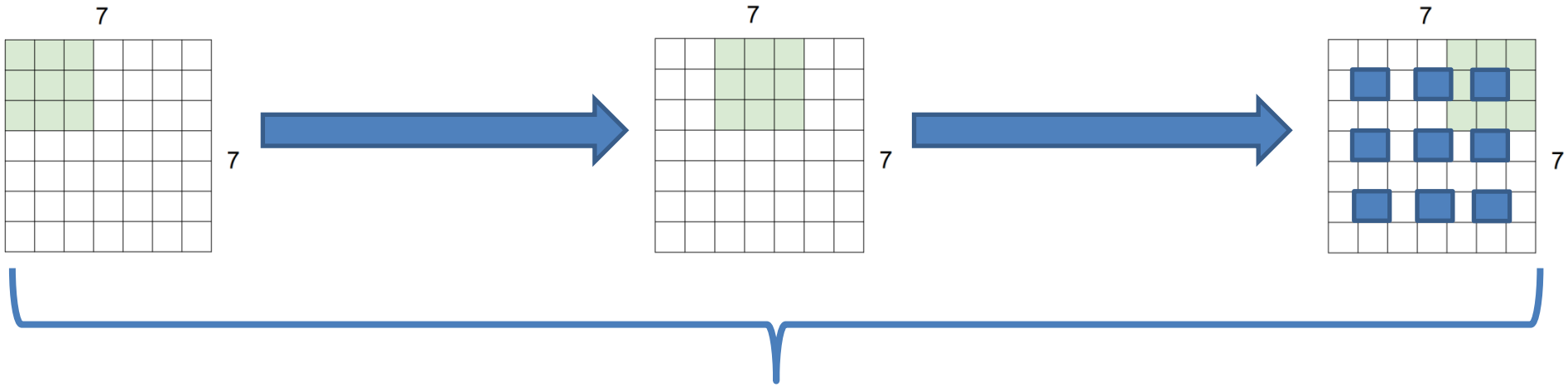


The output is a 5x5 image
(notice that we do not “replace” the edges)

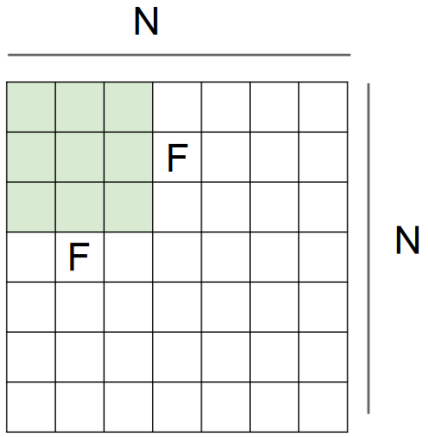
- The concepts of stride and padding appear here.

Convolutional Networks

- Lets pay attention now to the dimensions
  - Example: 7x7 image and 3x3 filter, with stride 2



The output is a 3x3 image



Output size:
(N - F) / stride + 1

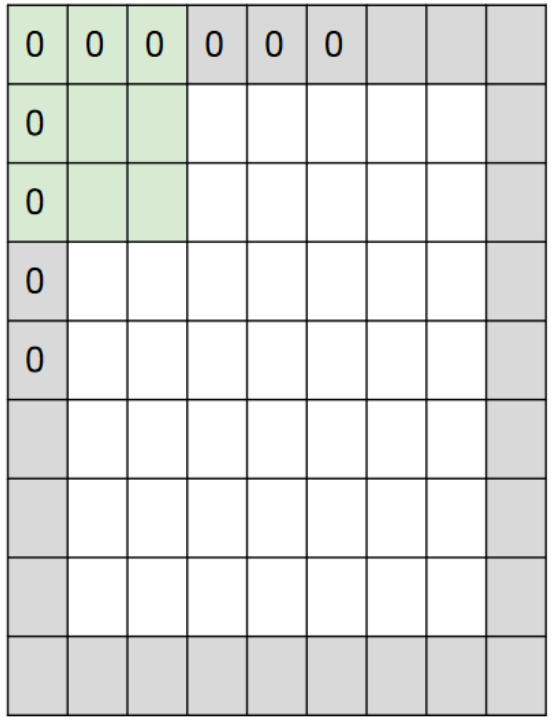
e.g. N = 7, F = 3:
stride 1 => (7 - 3)/1 + 1 = 5
stride 2 => (7 - 3)/2 + 1 = 3
stride 3 => (7 - 3)/3 + 1 = 2.33

Wrong! A priori, it is not possible to apply a 3x3 filter on a 7x7 input with stride 3!

http://cs231n.stanford.edu/slides/2018/cs231n_2018_lecture05.pdf

Convolutional Networks

- We progressively reduce the size of the maps/volumes.
  - If we want to keep the size → padding
  - Example: *zero-padding*



e.g. input 7x7
3x3 filter, applied with **stride 1**
pad with 1 pixel border => what is the output?

7x7 output!

In many cases it is good to use *padding*, as empirical evidence advises not to reduce dimensionality too quickly!

http://cs231n.stanford.edu/slides/2018/cs231n_2018_lecture05.pdf

Convolutional Networks

- Complete example:
  - Input volume: **16**x**16**x**3**
  - Filters used: **10** de **3**x**3** with stride **1** and padding **1**

Output_size = ((**N**+2*__P__-**F**)/**stride**) +1

  - Output volume?
    - (**16**+2*__1__-**3**)/**1** + 1 = 16 → 16x16x**10**

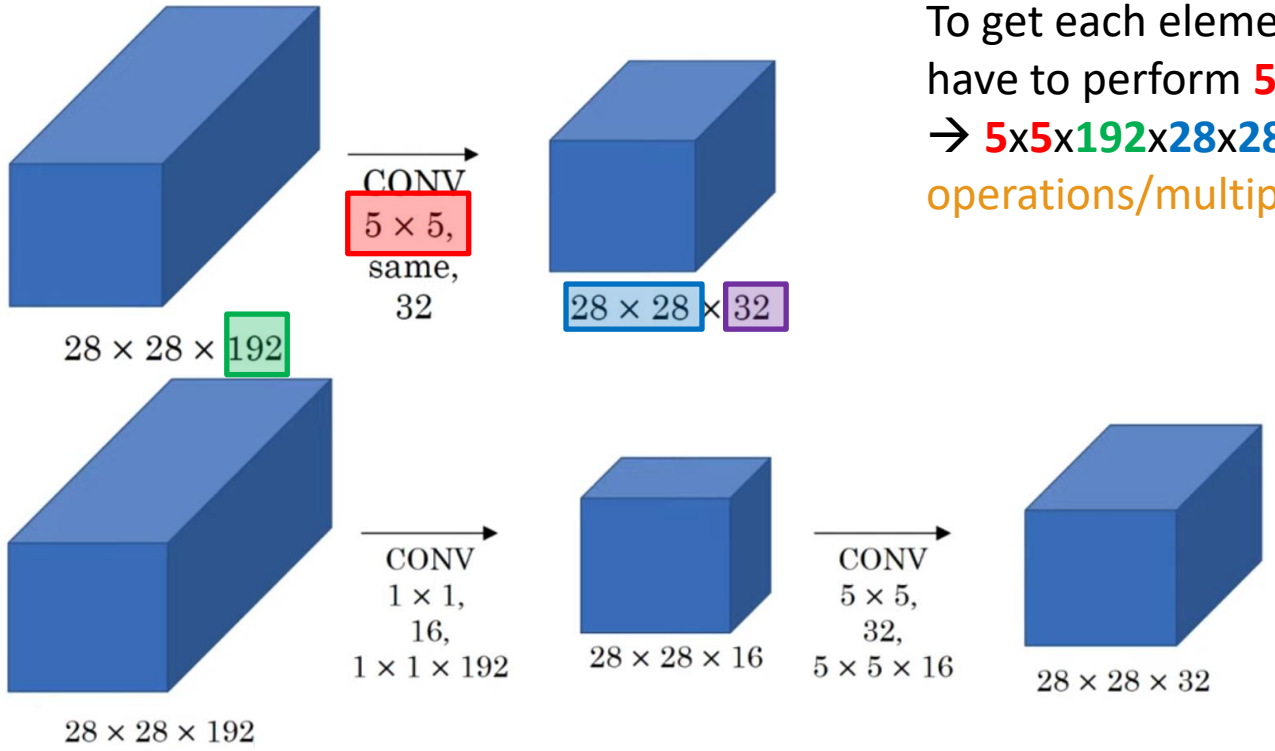
  - Number of parameters in that layer?
    - Each filter has **3**x**3**x**3** + 1 = 28 parameters

      → 28x**10** = 280 parameters

+1 by *bias*

Convolutional Networks

- **Convolutions 1x1**
  - They are used to linearly combine information from different channels and reduce (or increase) the number of filters (*feature maps*).
  - Reduces dimensionality and computational cost.



To get each element of the output volume we have to perform 5x5x192 operations (products)
→ 5x5x192x28x28x32 = ~120M operations/multiplications

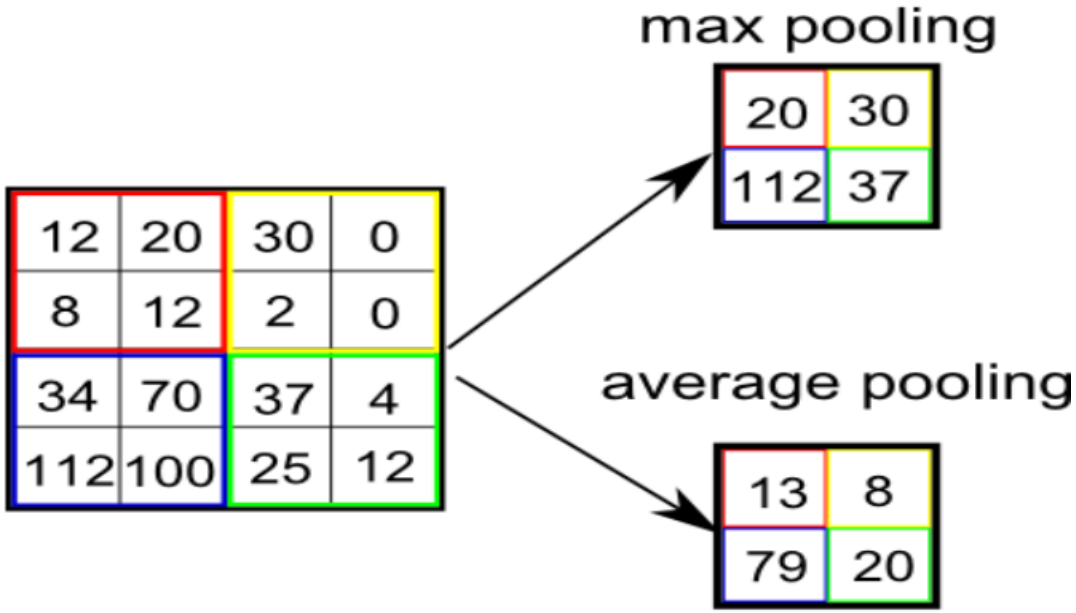
Now, we have to perform:
a) 1x1x192x28x28x16 = ~2.4M mult.
b) 5x5x16x28x28x32 = ~10M mult.

Total (a+b): ~12.4M operations

Convolutional Networks

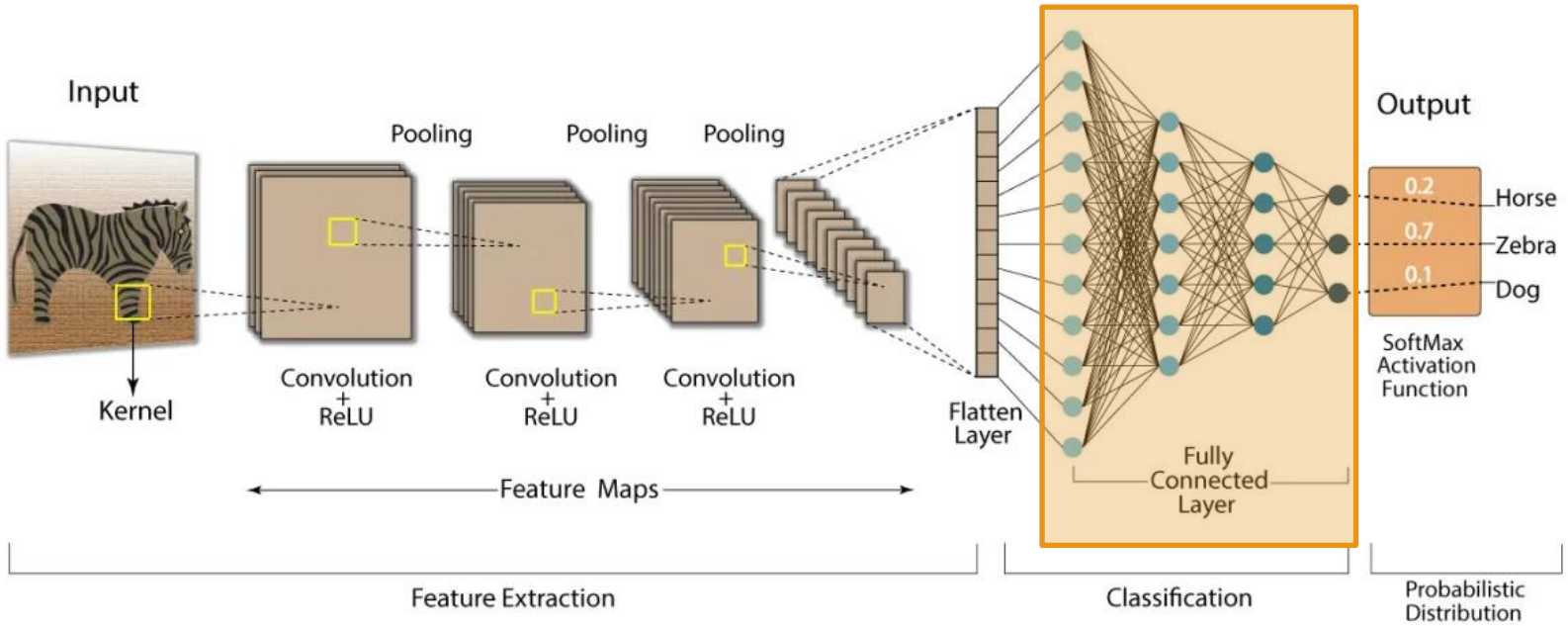
- **Pooling**. Reduces dimensionality and introduces certain invariance to (small) translations of the input
  - If the input is shifted by a small amount, most of the *pooling* output values do not change.

Example of *max* and *average pooling* with 2x2 filters and stride 2



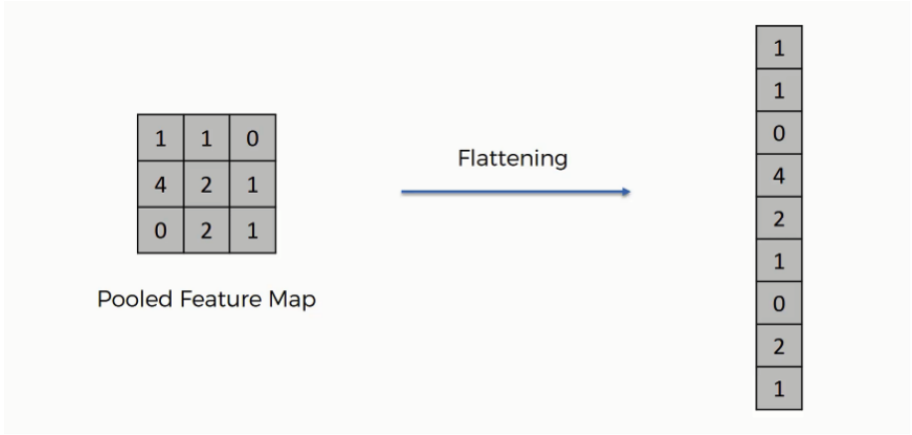
Convolutional Networks

- **Fully-connected or dense layers**.
  - Convolutional networks may or may not have fully-connected layers.
  - When these are present, they usually appear at the end of the network.
  - All units in one layer are connected to all units of the next layer.
  - They usually contain many parameters, so be careful!

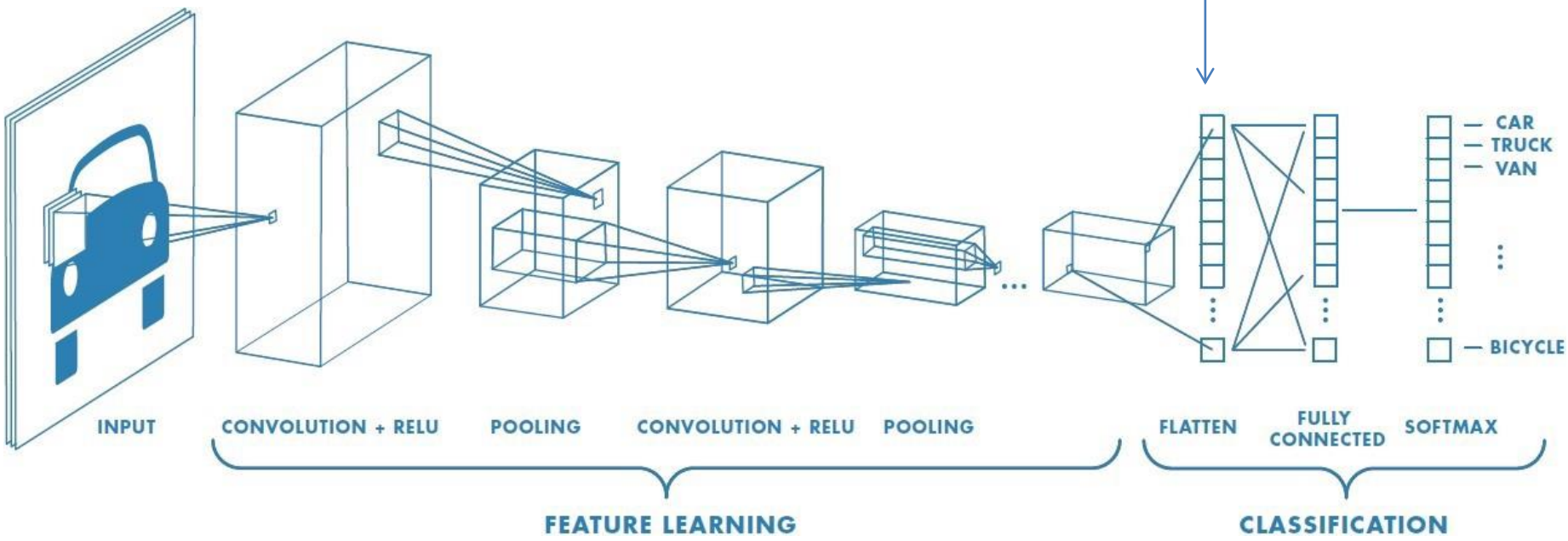


https://developersbreach.com/convolution-neural-network-deep-learning/

Combining everything



Flatten: Layer that "flattens" the *features*, allowing to pass from a volumen (set of feature map) to a vector (that can be processed by a fully-connected layer)



https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53

Combining everything

$$\sigma(\vec{z})_i = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}}$$

σ = softmax

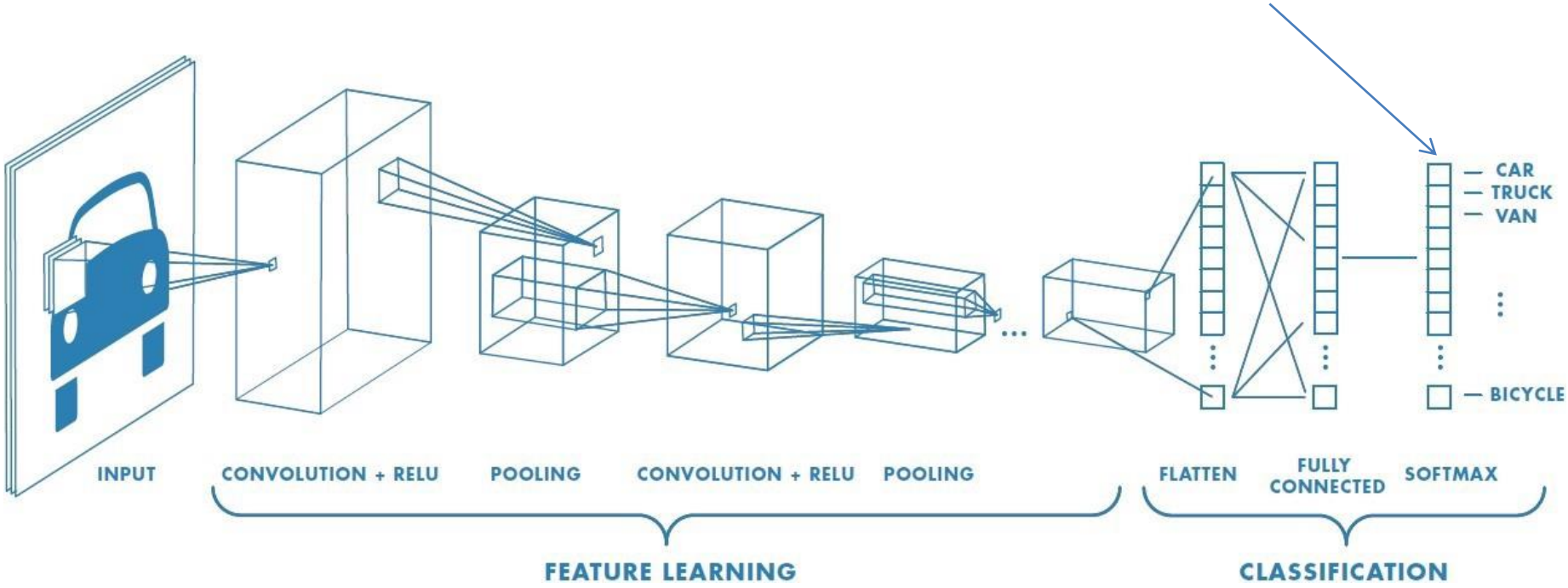
$\vec{z}$ = input vector

e^{z_i} = standard exponential function for input vector

K = number of classes in the multi-class classifier

e^{z_j} = standard exponential function for output vector

Softmax: activation function that generalizes the sigmoid function to multiple classes. It normalizes the output of the network, so that each prediction corresponds to the probability that the input belongs to that class.



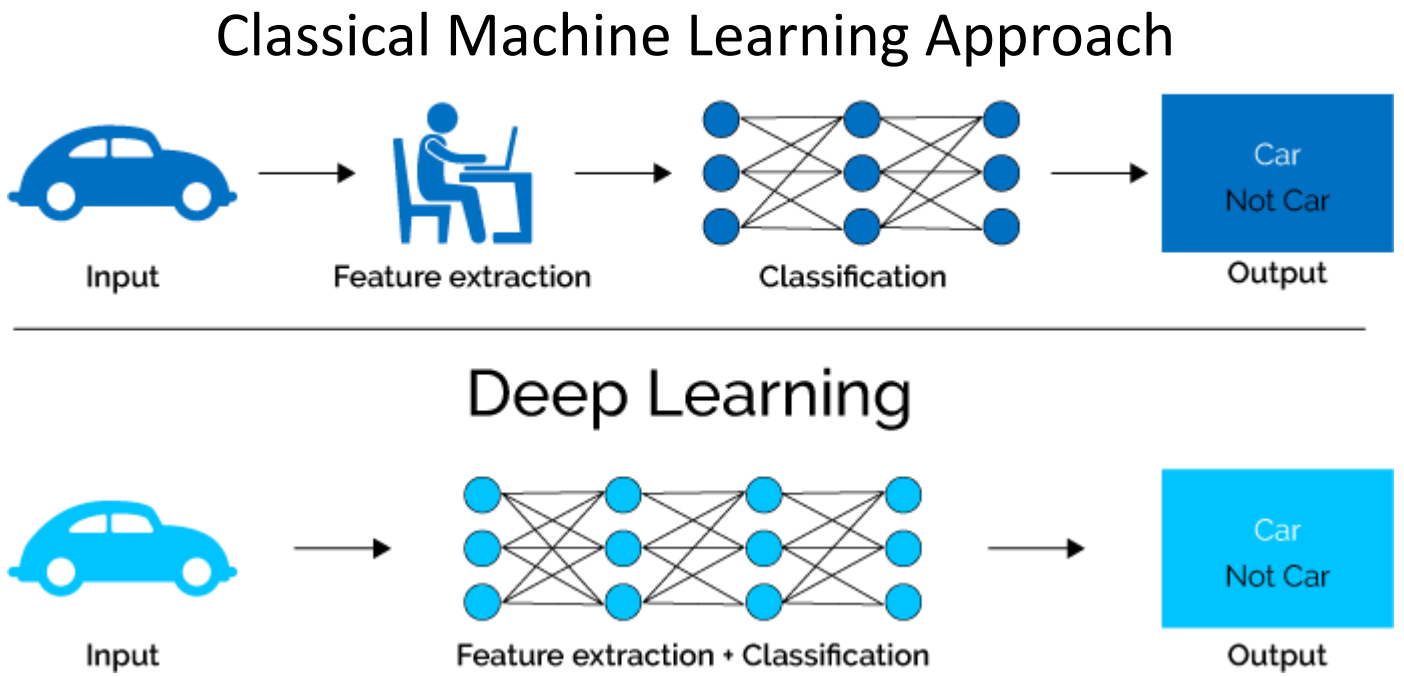
https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53

Visualizing how it works

- ConvNetJS. Deep Learning in your browser: https://cs.stanford.edu/people/karpathy/convnetjs/
- A Neural Network Playground: https://playground.tensorflow.org/
- An Interactive Node-Link Visualization of Convolutional Neural Networks: https://adamharley.com/nn_vis/
- CNN Explainer: https://poloclub.github.io/cnn-explainer/
- ConvNet Playground: https://convnetplayground.fastforwardlabs.com/
- Topological visualisation of a convolutional neural network: https://terencebroad.com/works/cnn-vis

Machine Learning vs Deep Learning

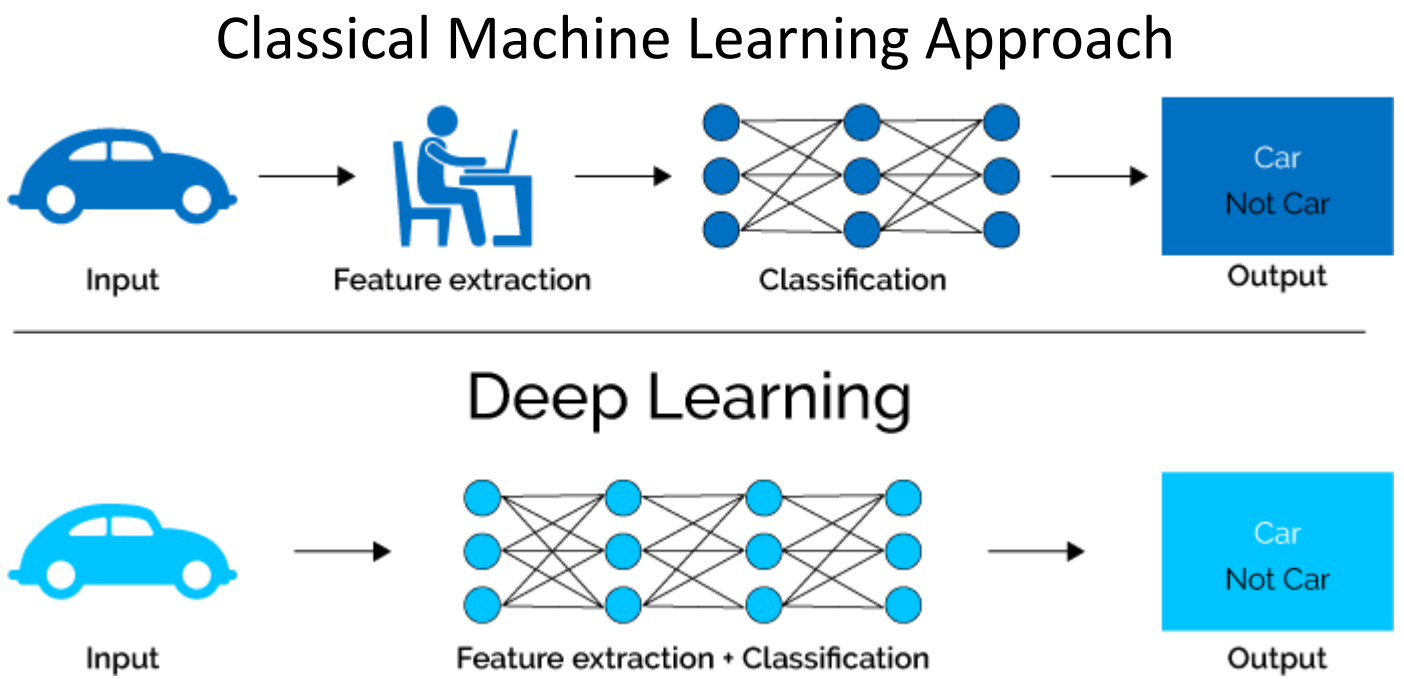
- Methodological level: ConvNets allow *feature learning* instead of designing feature extractors by hand (*feature engineering*).
- Empirical level: they provide superior results in many vision tasks (classification, regression, segmentation, etc.)



https://towardsdatascience.com/cnn-application-on-structured-data-automated-feature-extraction-8f2cd28d9a7e

Machine Learning vs Deep Learning

- Deep Learning blurs the boundaries between feature extraction and image classification (*end-to-end learning*).
- The key is to have a good internal representation of the input data.



https://towardsdatascience.com/cnn-application-on-structured-data-automated-feature-extraction-8f2cd28d9a7e

How are ConvNets trained?

- *Backpropagation* is used.
  - Parameter learning becomes an optimization problem.

- Do not confuse *backprop* and gradient descent!
  - Gradient descend uses backprop!

Optimization method for minimizing a loss function using the gradient

$$\mathbf{a}_{n+1} = \mathbf{a}_n - \gamma \nabla F(\mathbf{a}_n)$$

Method for calculating the gradient (i.e. the partial derivatives of the loss function with respect to the weights)

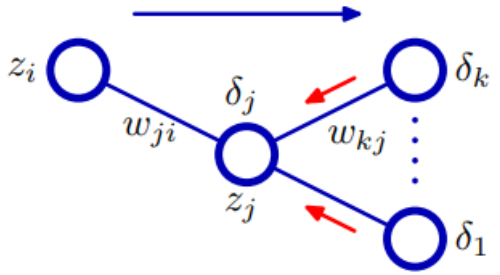
- *Backprop* is the chain rule!

Intuitive explanations of *Backprop*: https://www.jeremyjordan.me/neural-networks-training/ and https://www.youtube.com/watch?v=Ilg3gGewQ5U
Interesting post in https://karpathy.medium.com/yes-you-should-understand-backprop-e2f06eab496b
Neural Networks and Backpropagation: http://cs231n.stanford.edu/slides/2023/lecture_4.pdf (https://www.youtube.com/watch?v=i94OvYb6noo)

How are ConvNets trained?

Given a training example

1. Propagate it forward (*forward pass*), calculate activations and the output error
2. Go backwards (*backward pass*) calculating the "error" (δ_j) of each unit *j* on each layer
3. We update the parameters/weights using the calculated gradient. $\mathbf{w}^{(\tau+1)} = \mathbf{w}^{(\tau)} - \eta \nabla E(\mathbf{w}^{(\tau)})$



Batch gradient descent: we use the average of the gradients of all training examples to update the weights

Mini-batch gradient descent: we use the average of the gradients of a subset of the training examples to update the weights

Stochastic gradient descent: we use the gradient of a single training example to update the weights

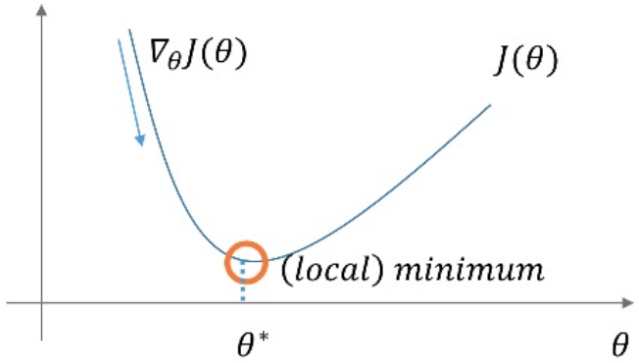
How are ConvNets trained?

- Batch: set in which the training set is divided to train and adjust the weights.
  - Batch Gradient Descent. Batch Size = Training Set Size
  - Stochastic Gradient Descent. Batch Size = 1
  - Mini-Batch Gradient Descent. 1 < Batch Size < Training Set Size
- Epoch: each time the network, during training, sees the entire training set.
- Iteration: number of batches required to complete an *epoch*.

How are ConvNets trained?

- Vanilla gradient descent has a single learning rate for all the weights and does not change throughout the training.

$$\theta = \theta - \eta \cdot \nabla_\theta J(\theta)$$

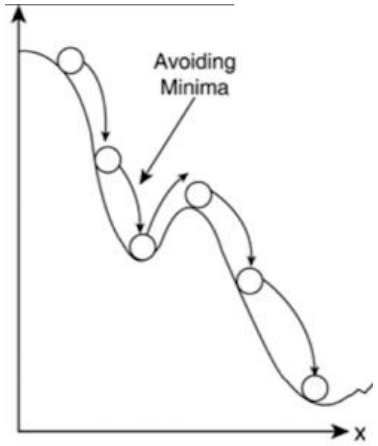


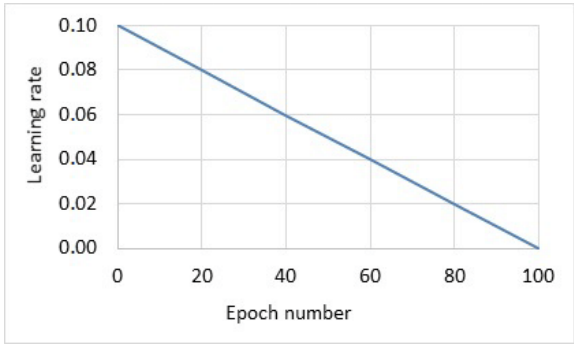
https://www.slideshare.net/SebastianRuder/optimization-for-deep-learning

- There are many other algorithms and optimization strategies (https://ruder.io/optimizing-gradient-descent/, https://d2l.ai/chapter_optimization/index.html)

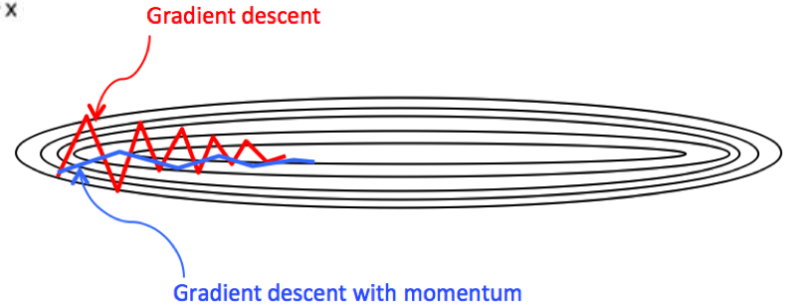
How are ConvNets trained?

- Learning rate decay
- SGD with momentum:
  - Accelerates the training (trajectory with less oscillations) and allows to skip local minima (https://distill.pub/2017/momentum/, https://www.youtube.com/watch?v=k8fTYJPd3_I)





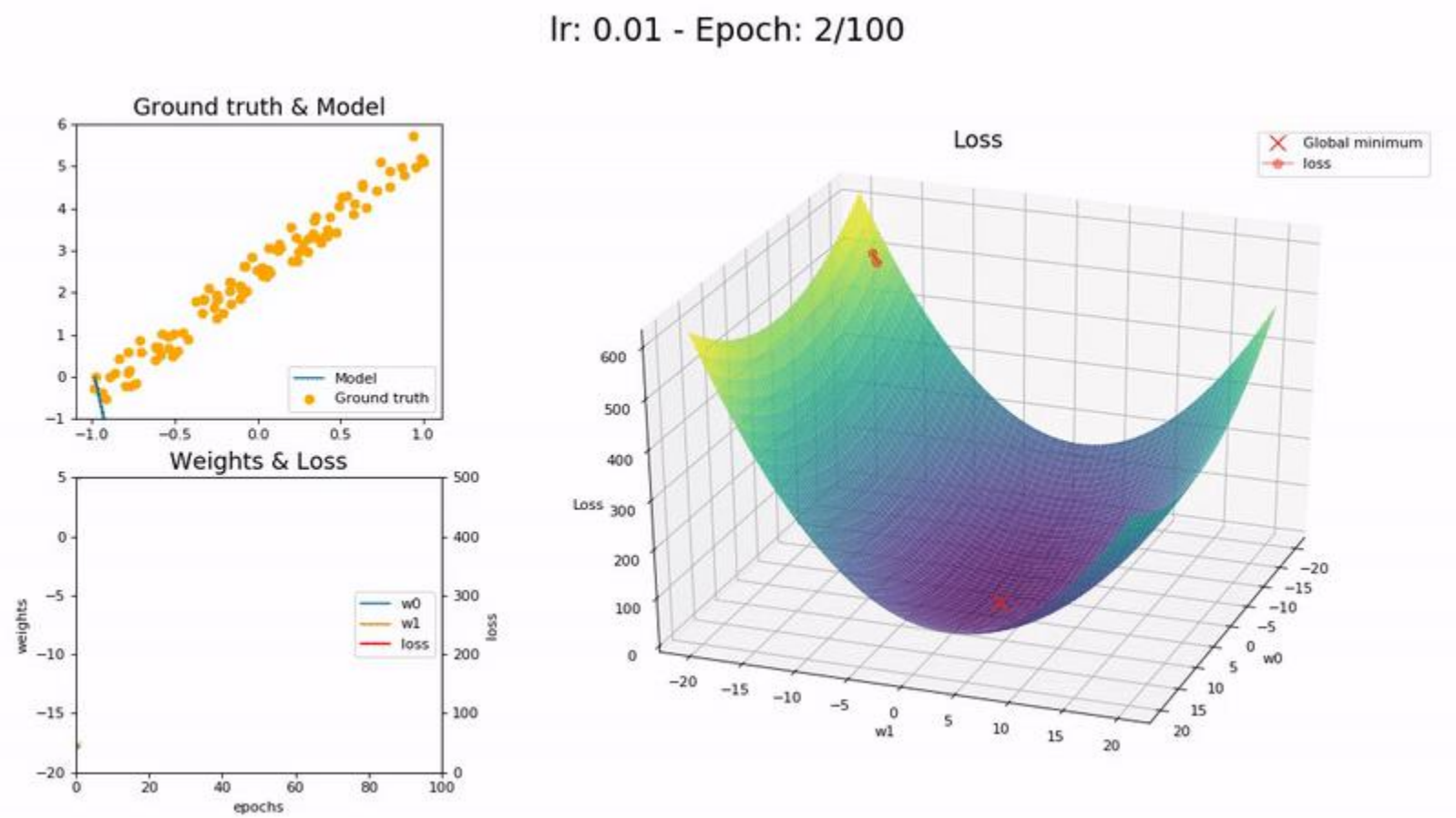
- Adam [Kingma and Ba, 2015]: RMSProp + momentum
  - Learning rate for each parameter
  - This learning rate is adaptive



- AdamW [Loshchilov and Hutter, 2019]: Adam + weight decay
  - Empirically provides models with better generalization
  - Corrects a bug in the way weight decay (L2 regularization) is commonly implemented in Adam (https://www.fast.ai/posts/2018-07-02-adam-weight-decay.html, https://towardsdatascience.com/why-adamw-matters-736223f31b5d)
    - Disconnect weight decay from gradient update

Importance of learning rate

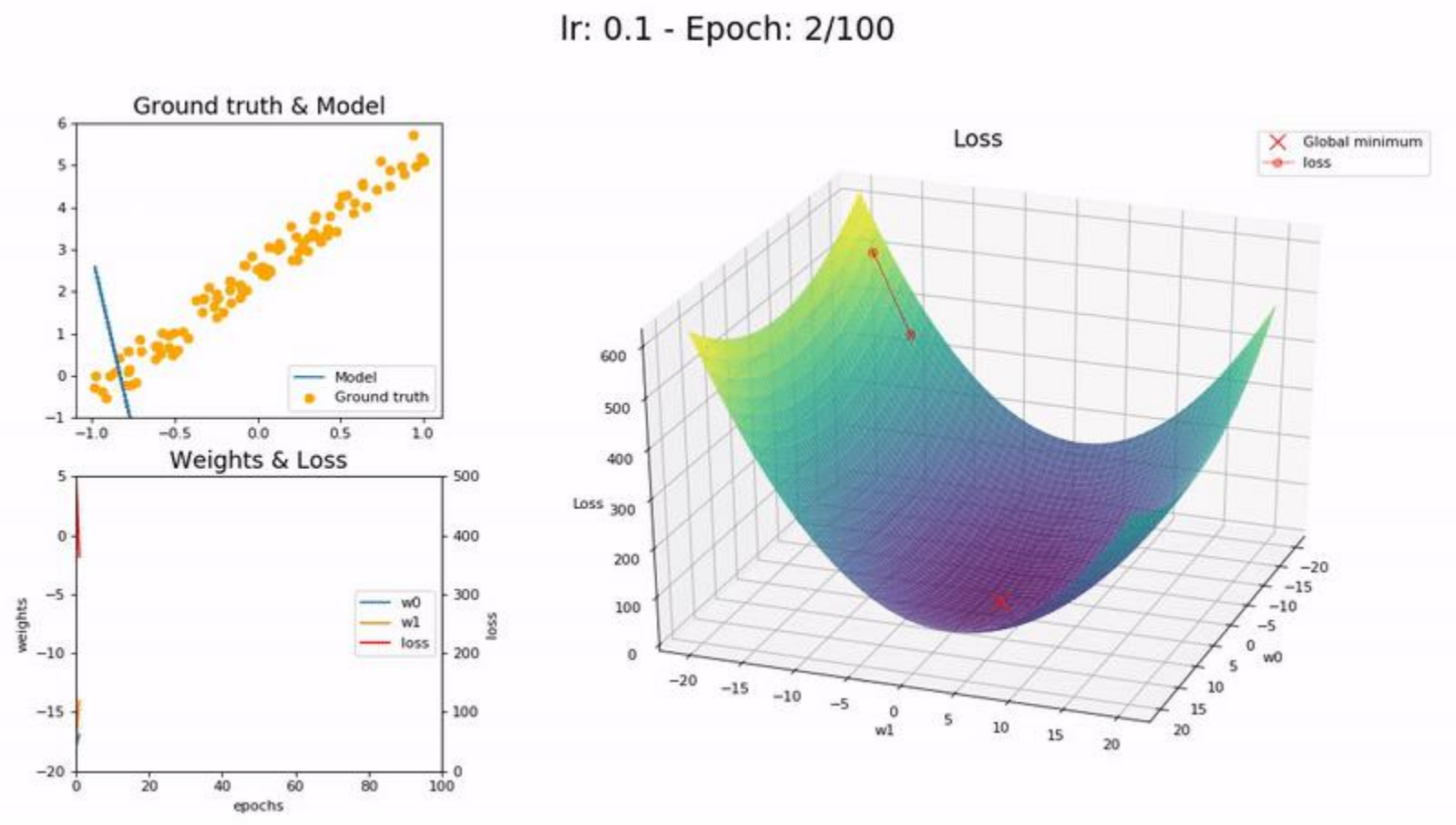
Learning rate too small: the optimization of the model progresses too slowly. In fact, it does not reach the optimum at the indicated epochs.



https://iconof.com/1cycle-learning-rate-policy/

Importance of learning rate

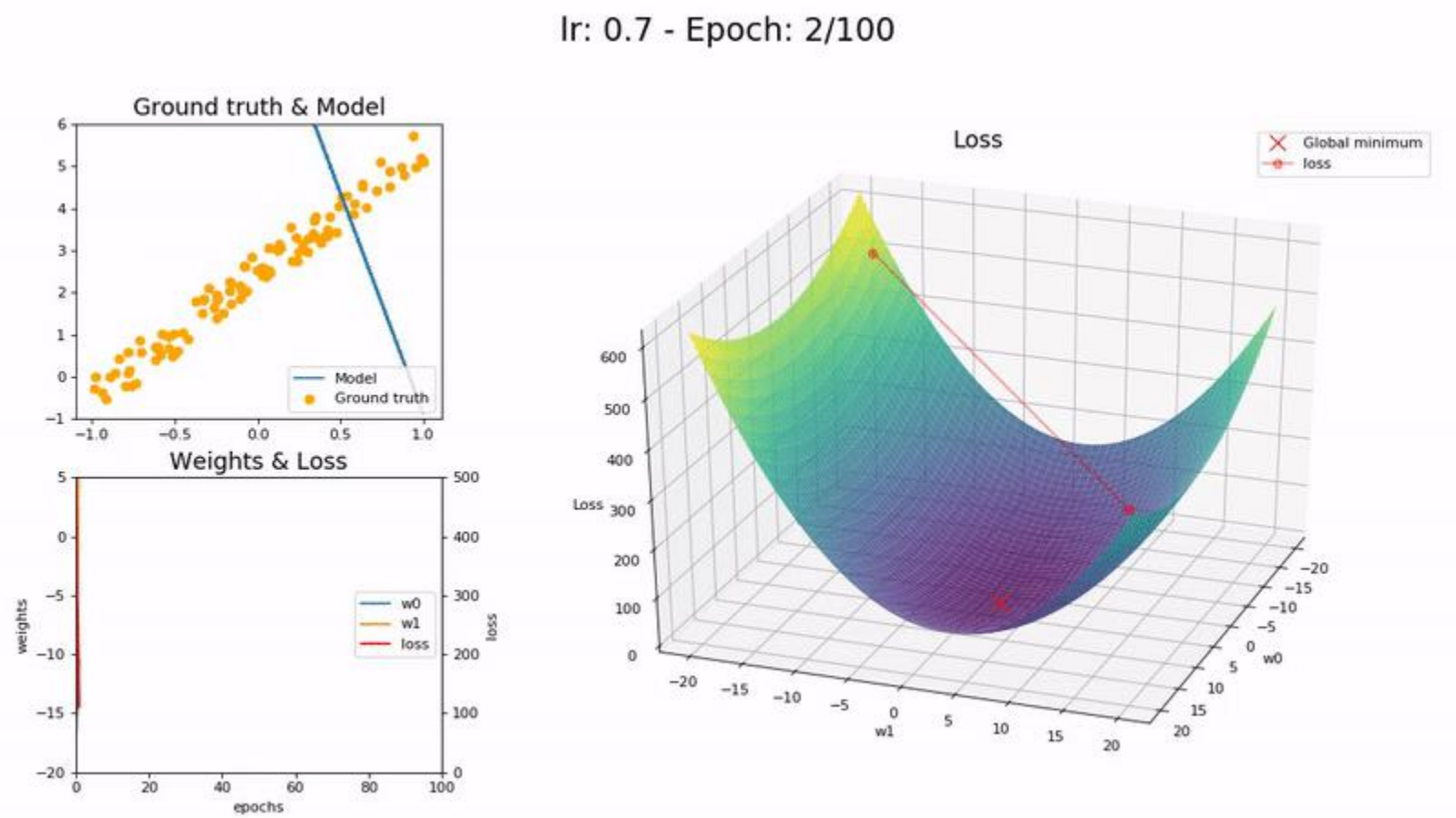
Adequate learning rate: the model converges within the indicated epochs.



https://iconof.com/1cycle-learning-rate-policy/

Importance of learning rate

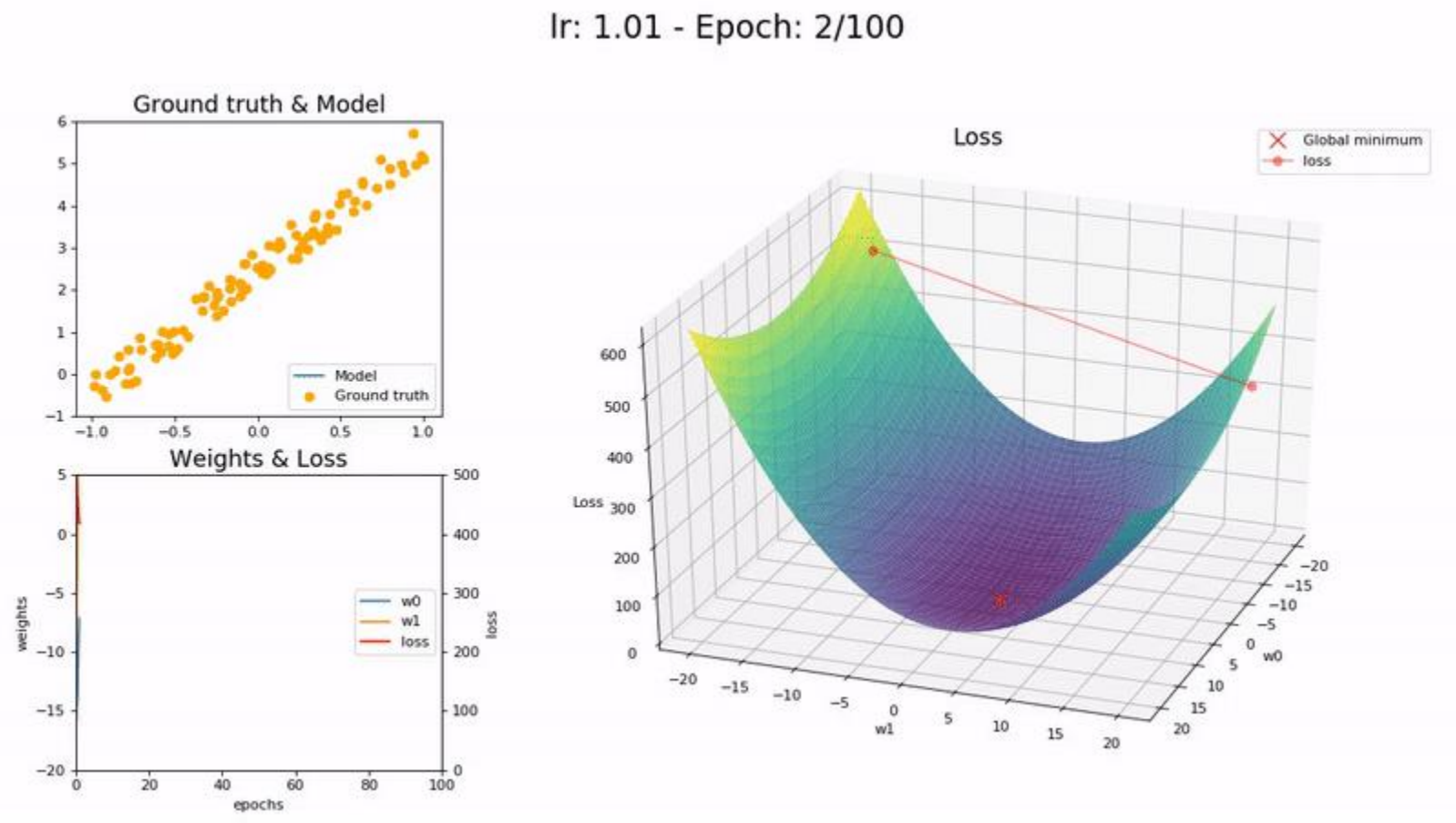
Optimal learning rate: the model reaches the optimum very quickly (less than 10 epochs).



https://iconof.com/1cycle-learning-rate-policy/

Importance of learning rate

Learning rate too high: the model diverges.



https://iconof.com/1cycle-learning-rate-policy/

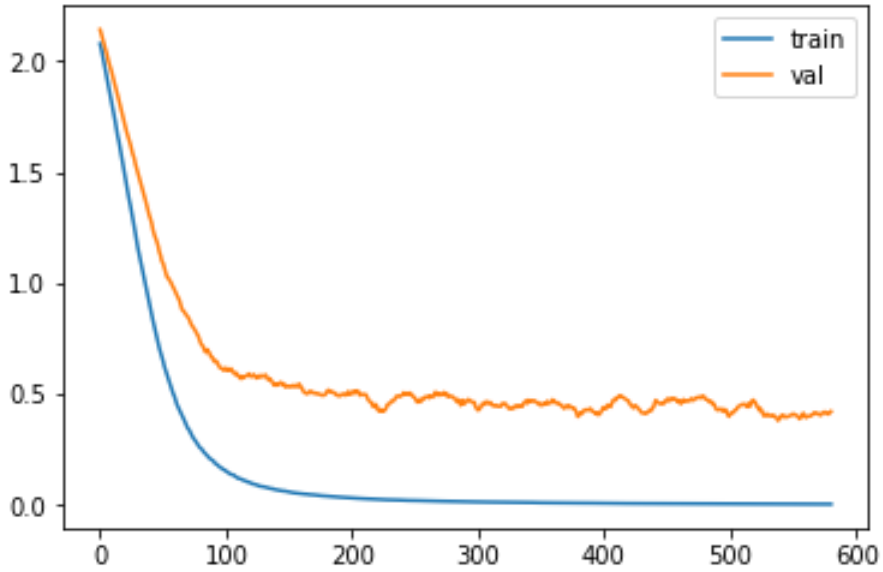
Ways to improve training

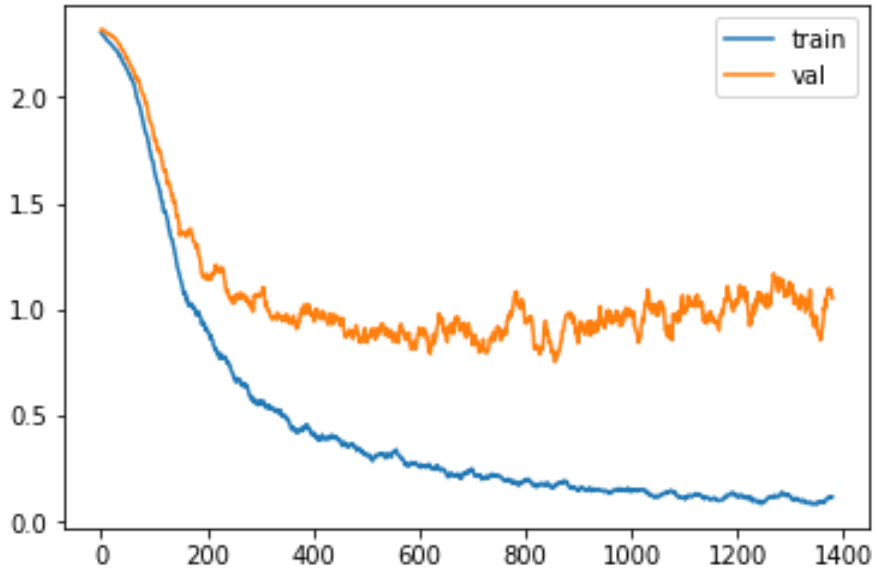
- Deep networks have many parameters → they need large amounts of data to train and have a high risk of overfitting.

- What can we do to improve our training (i.e. accelerate it and/or regularize it)?
  - Introduce loss penalties
  - Use a different optimizer or select a better learning rate
  - Early Stopping
  - Data augmentation
  - Batch normalization and other ways of normalization
  - Dropout

Overfitting

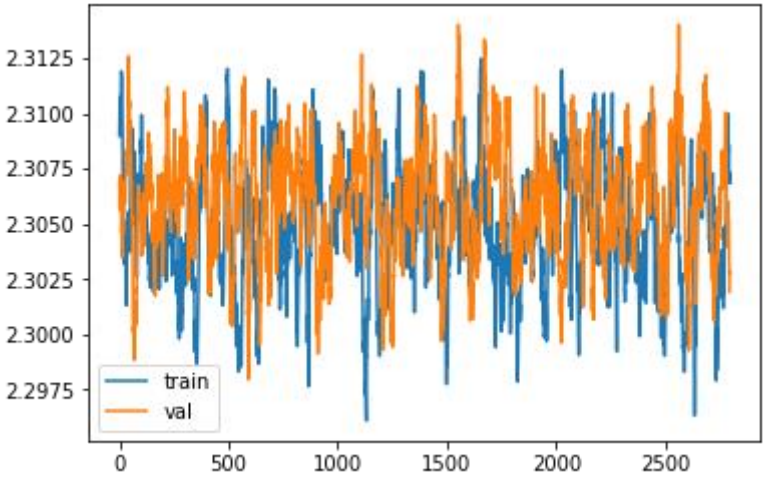
- The model does not generalize properly
- The model is too complex / we have too little data



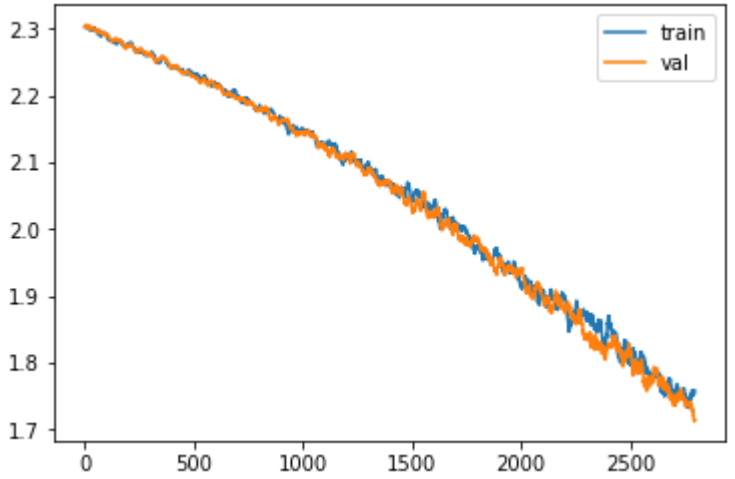


Look at the training curves because they give you a lot of information about what might be happening to you

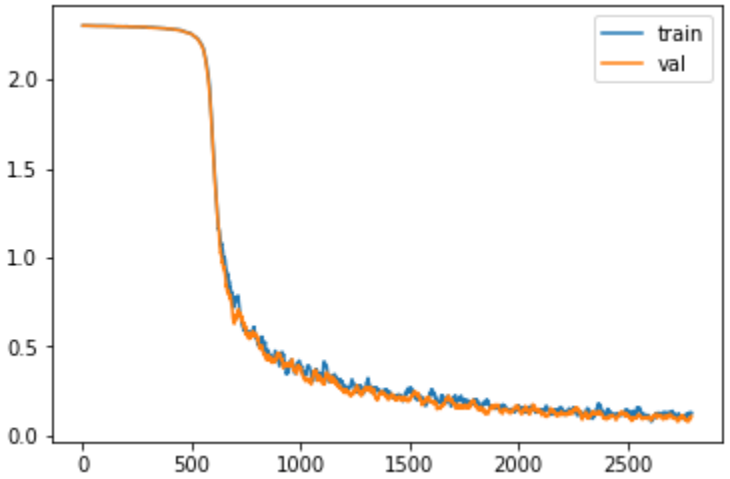
Looking at training curves



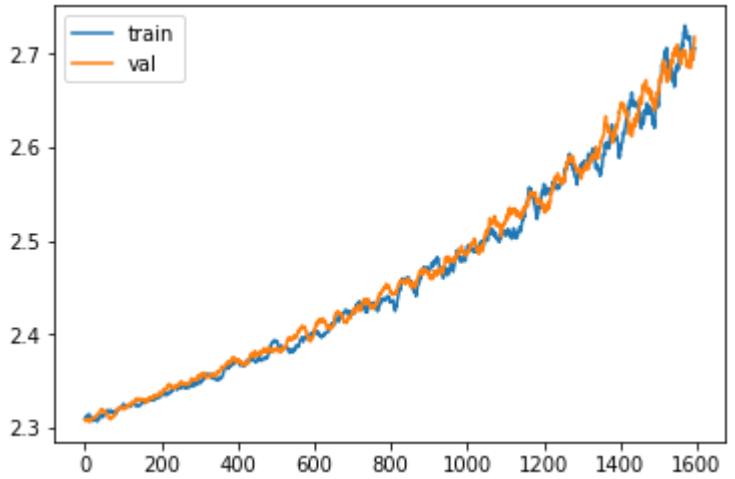
No learning: gradients not applied to weights



Not yet converged: needs more time



Slow start: improper initialization of weights

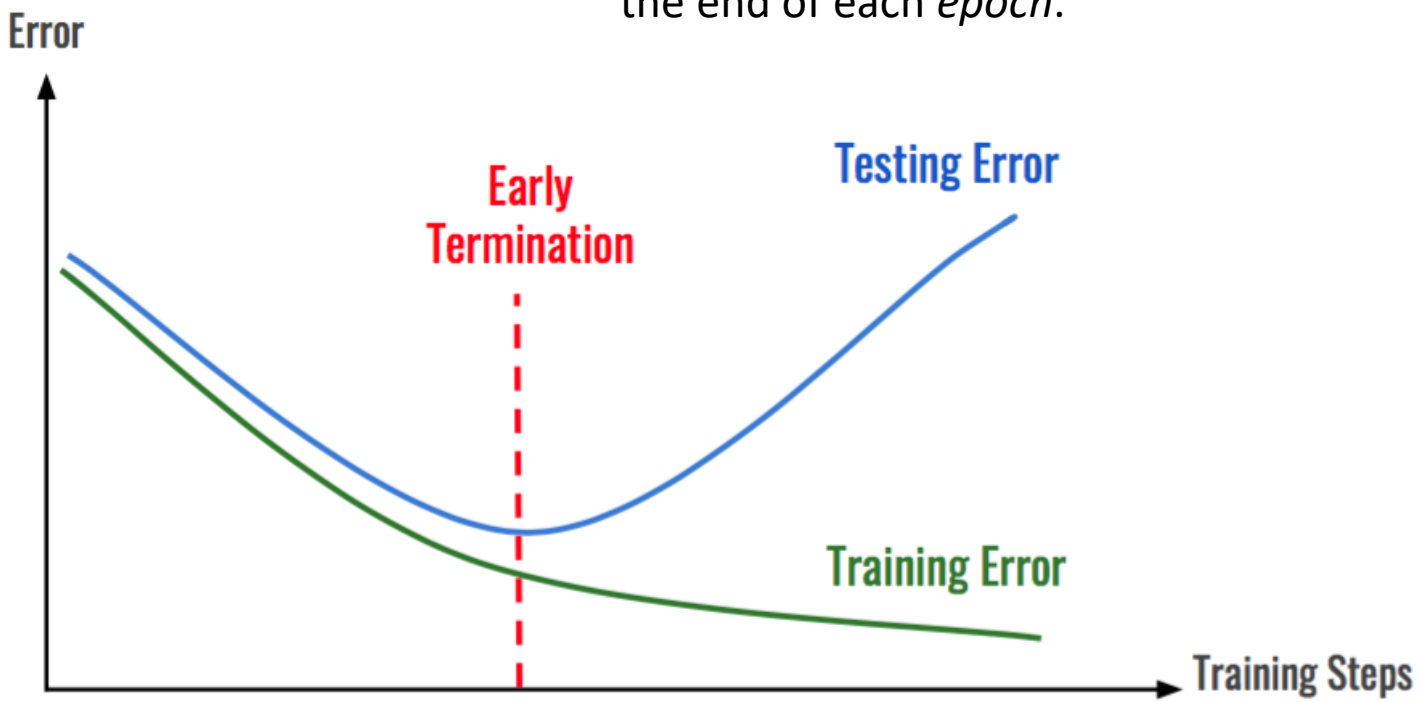


You are probably applying the negative of gradients

[Tips and tricks for tuning NNs (Fei & Nish)]

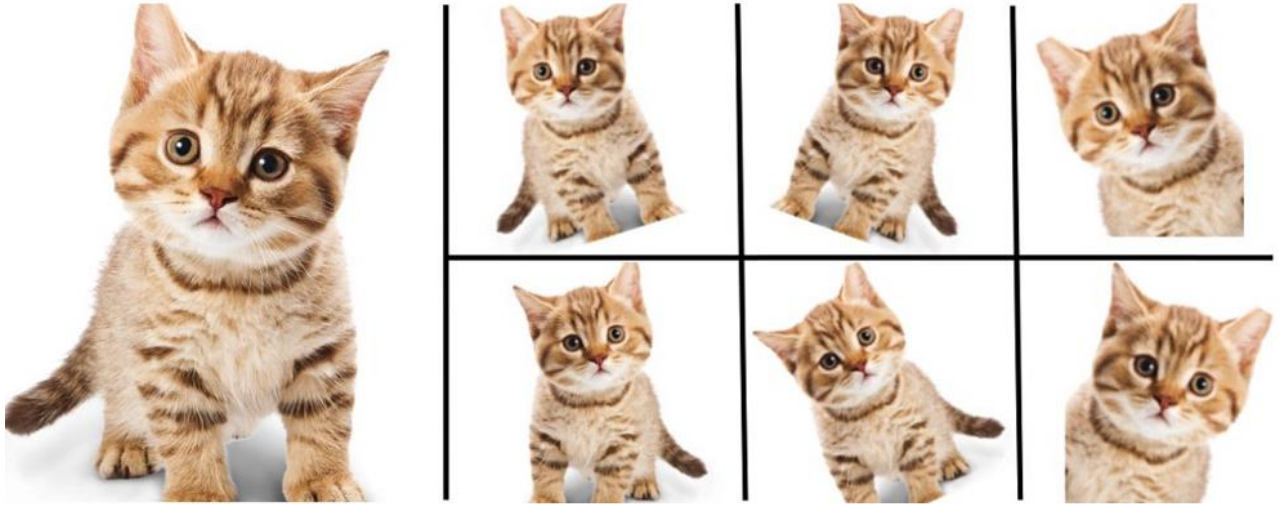
Early Stopping

Note: generally, the validation set is passed at the end of each *epoch*.

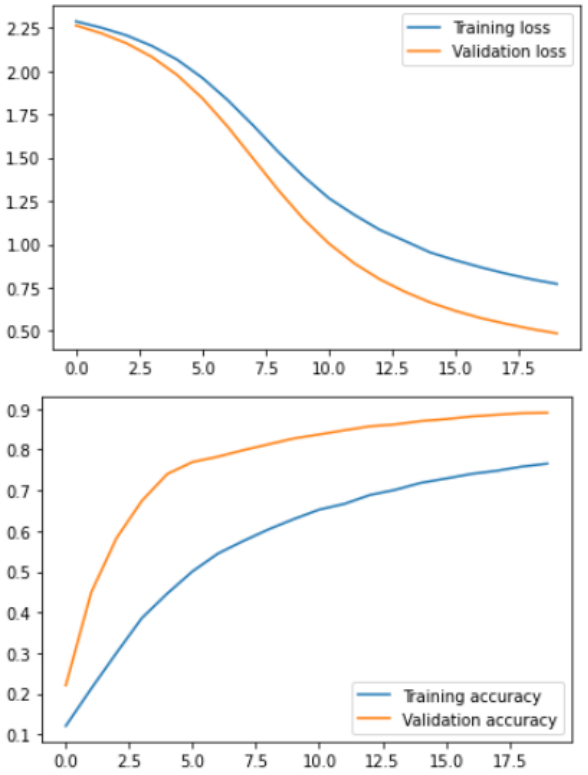


Data Augmentation

- We apply different transformation to the input data → we increase the training set
  - E.g. in images: rotations, contrast changes, noise insertion, *mirroring*,…
  - Depending on the problem, some transformations will be valid and others will not. E.g. if you want to classify digits and you rotate a 6 a lot, it could become a 9…



On the evolution of errors



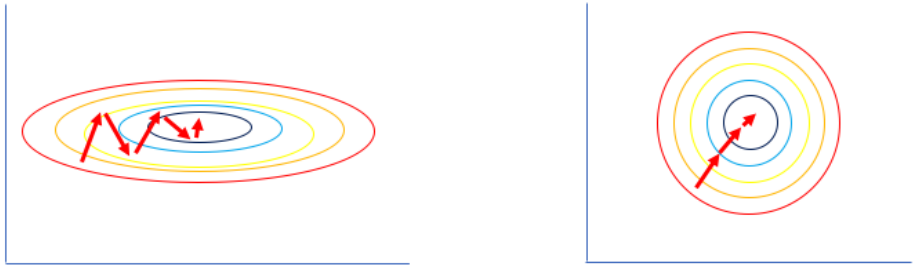
Infrequent (but possible): the network provides better results in validation than in training.

Possible reasons:

1) Error in visualization/implementation
2) The validation set is "easier" than the training set
   - This can be verified by testing with different partitions and percentages of data, and see if the behavior is the same.
3) Influence of the way the *framework* computes errors:
   - For example, if the *training loss* is the average of the *losses* of all *batches* within an *epoch* (note that at the beginning of the epoch errors are generally higher), while *validation* is only at the end of the epoch (with the trained model).
4) Influence of Dropout (or other regularizing elements, such as BN).
   - Dropout is activated in training and deactivated in test/validation. Training accuracy is sacrificed for robustness and generalization in validation.

Ways to improve training

- Batch normalization [Ioffe & Szegedy, 2015]
  - It is known that normalizing input accelerates training



  - The idea is to normalize the inputs to the hidden layers (usually just before applying the activ. func.).

Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1...m}\}$;
Parameters to be learned: γ, β
Output: $\{y_i = \mathrm{BN}_{\gamma,\beta}(x_i)\}$

$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \qquad \text{// mini-batch mean}$$

$$\sigma^2_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_{\mathcal{B}})^2 \qquad \text{// mini-batch variance}$$

$$\widehat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma^2_{\mathcal{B}} + \epsilon}} \qquad \text{// normalize}$$

$$y_i \leftarrow \gamma \widehat{x}_i + \beta \equiv \mathrm{BN}_{\gamma,\beta}(x_i) \qquad \text{// scale and shift}$$

Algorithm 1: Batch Normalizing Transform, applied to activation x over a mini-batch.

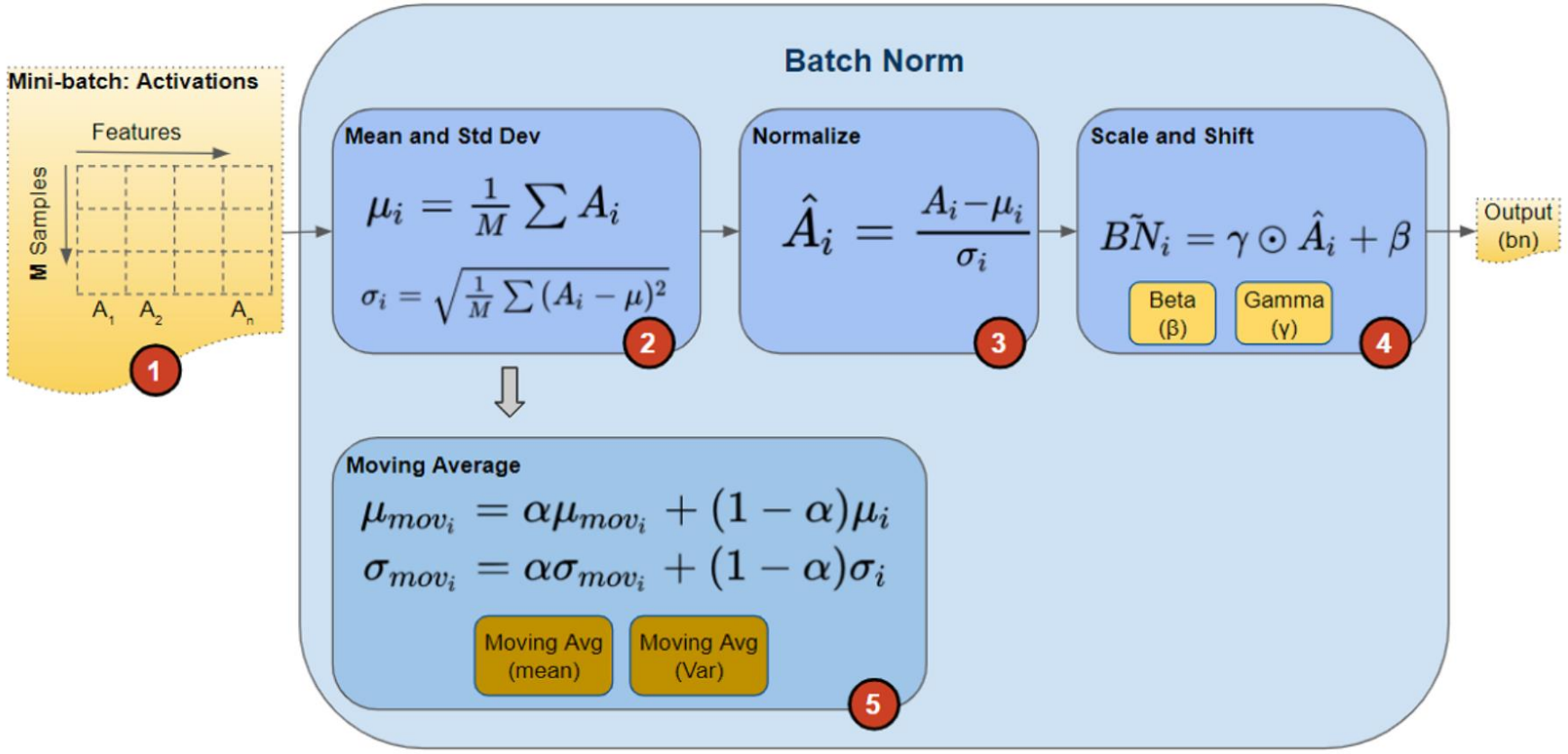
γ, β are parameters to be learned.
Statistics are calculated for each batch and each channel.

Ways to improve training

- Batch normalization [Ioffe & Szegedy, 2015]
  - γ, β are parameters to be learned (like any other weight).
  - statistics (μ, σ^2) are not learned, but are "stored" as part of the "state" of that BN layer for each channel.
    - Then, in test, an *exponential moving average* of statistics (μ, σ^2) is used.
  - These four parameters are for each BN layer.
    - If we have three hidden layers, each with BN, we will have 3 γ's, β's, μ's and σ^2's
      - Note that, if the imput has multiple channels, there will be one μ and σ^2 for each channel and BN layer.

Ways to improve training

- Batch normalization [Ioffe & Szegedy, 2015]



Intuitive visual explanation of how BN works: https://towardsdatascience.com/batch-norm-explained-visually-how-it-works-and-why-neural-networks-need-it-b18919692739

In this example BN is included after activation. What is called *Features* in the figure are the channels.

Ways to improve training

- Other ways of normalization

We normalize each batch, taking each channel separately

We normalize each batch image separately, including all channels

We normalize each channel of each image (of each batch) separately

We normalize a set of channels from each image (from each batch)

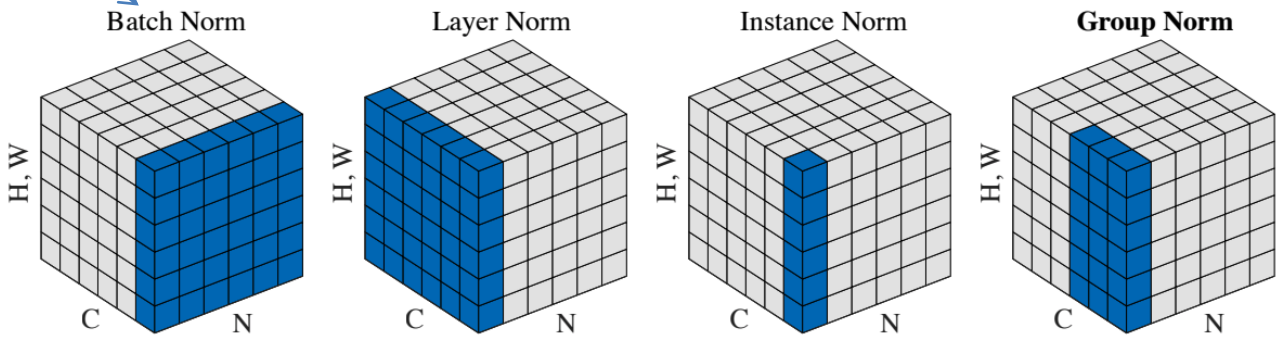
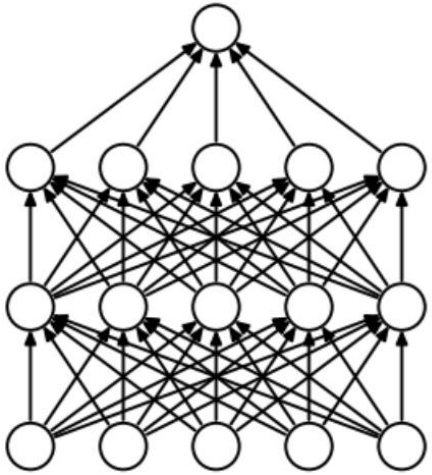


Figure 2. **Normalization methods**. Each subplot shows a feature map tensor, with N as the batch axis, C as the channel axis, and (H, W) as the spatial axes. The pixels in blue are normalized by the same mean and variance, computed by aggregating the values of these pixels.

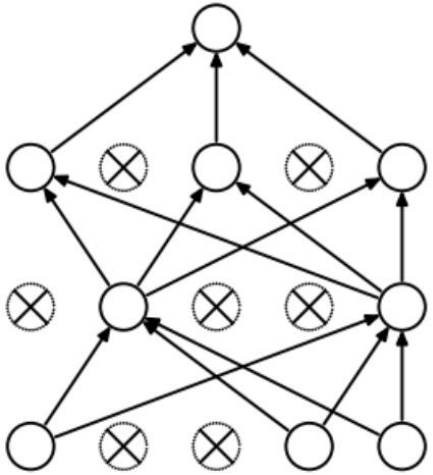
Image taken from Wu, Y. and He, K., 2018. Group normalization. https://arxiv.org/abs/1803.08494

Ways to improve training

- Dropout [Hinton et al., 2012]
  - We randomly eliminate hidden units during training.
  - It forces the "generalization" of units, decoupling them from each other.
  - Generally, it is applied in fully-connected layers, and with different units in each iteration.



(a) Standard Neural Net



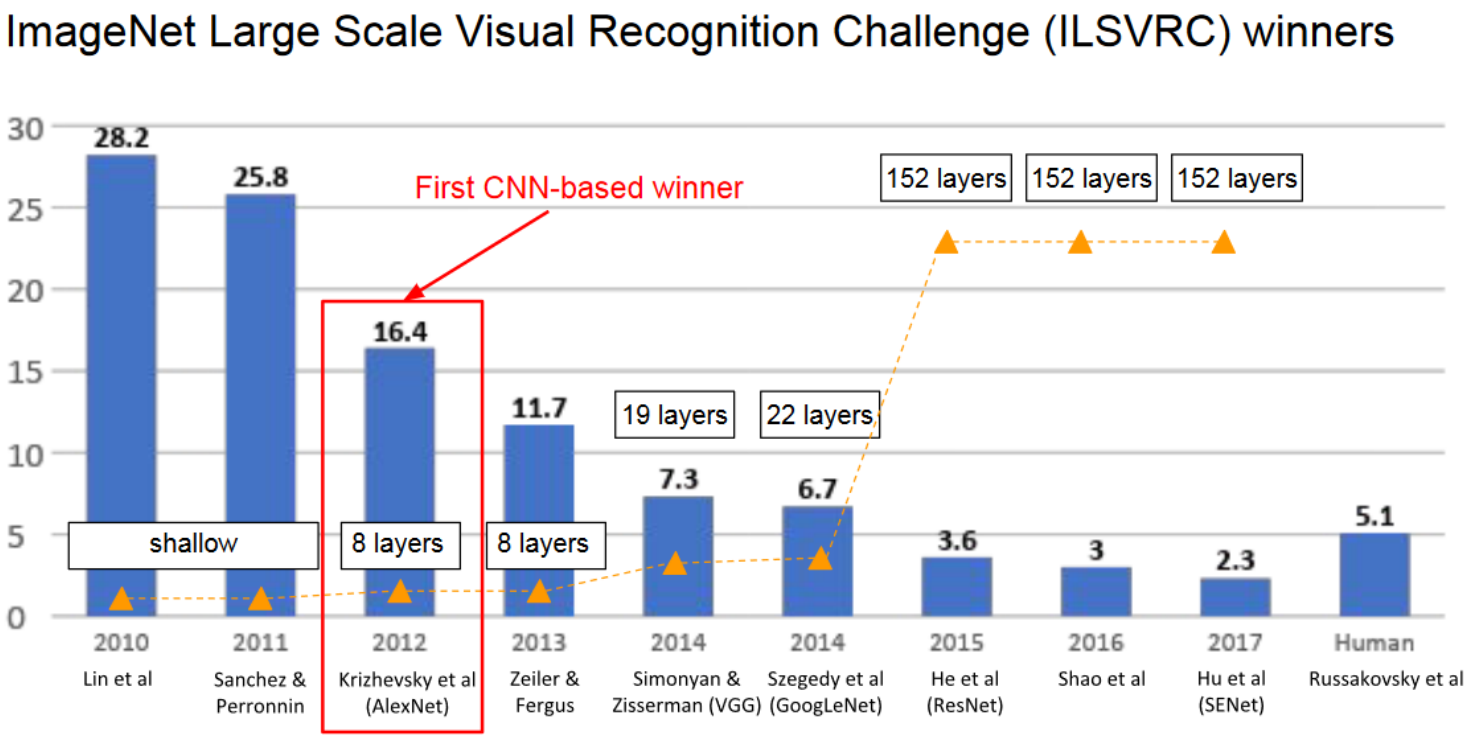
(b) After applying dropout.

On neural models stochasticity

- You will see that when training a model you get different results each time
  - Remember that neural netwoks are stochastic models
    - Random weights initialization
    - Random selection of batches for training
    - Neurons randomly “turned off” in Dropout
- In test, the weights are fixed, the model is static, and the results should always be the same.

Some popular models

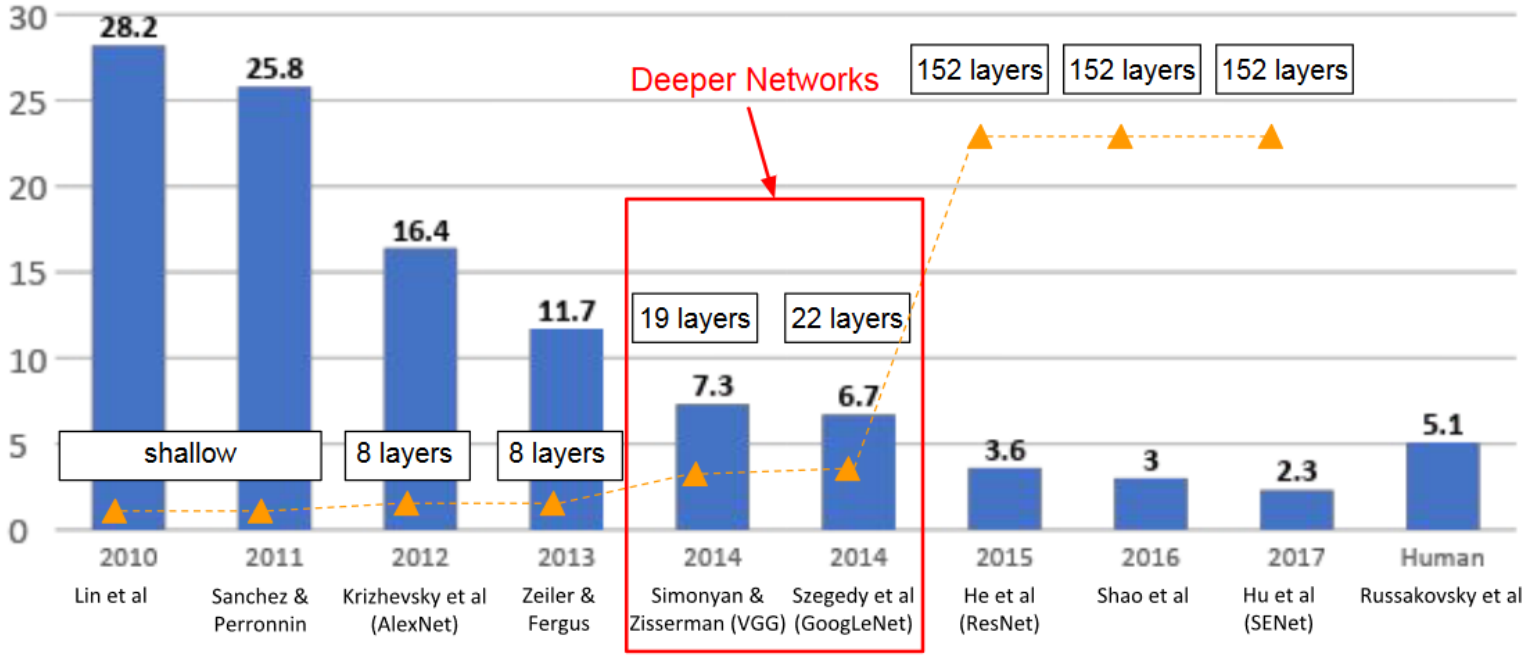
- LeNet-5 [LeCun et al., 1998]
- AlexNet [Krizhevsky et al., 2012]
  - First use of ReLU
  - It was the take off of deep learning



Slide credit: Fei-Fei Li, Justin Johnson and Serena Yeung. http://cs231n.stanford.edu/slides/2018/cs231n_2018_lecture09.pdf

Some popular models

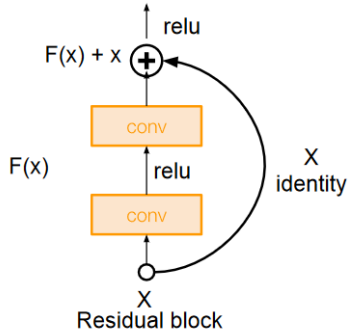
- VGGNet [Simonyan and Zisserman, 2014]
  - Smaller filters (3x3), deeper networks (more non-linearity)
- GoogLeNet [Szegedy et al., 2014]
  - Combines different receptive fields (Inception modules)
  - Uses 1x1 convolutions to reduce computational cost
  - Auxiliary intermediate outputs to “inject” additional gradient into lower layers

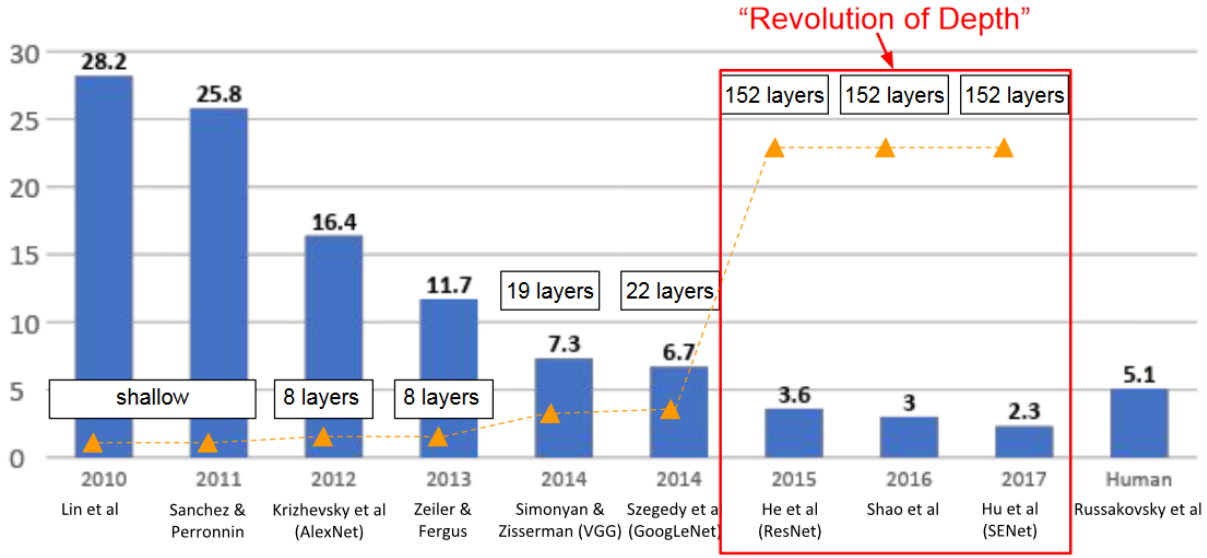


Slide credit: Fei-Fei Li, Justin Johnson and Serena Yeung. http://cs231n.stanford.edu/slides/2018/cs231n_2018_lecture09.pdf

Some popular models

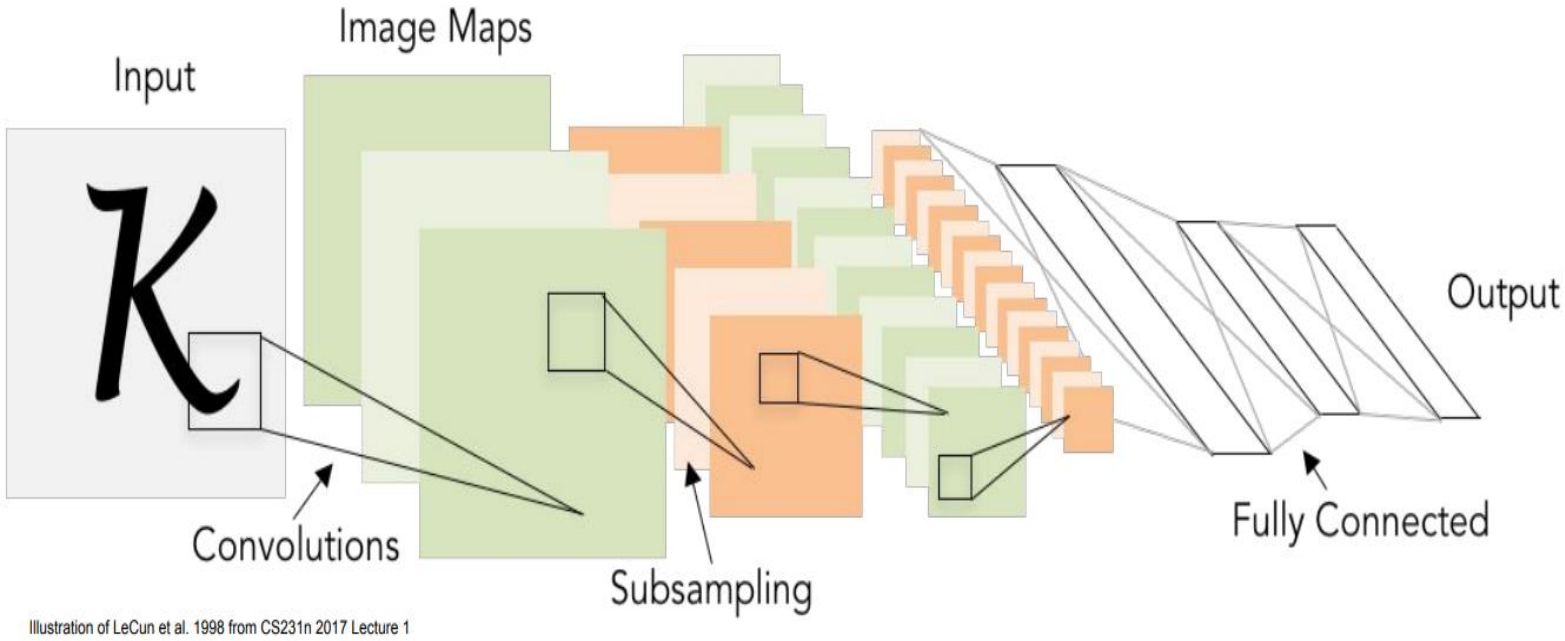
- ResNet [He et al., 2015]
  - Very Deep networks using residual connections (*skip connections* together with *batch norm and* 1x1 convolutions)
- ResNeXt [Xie et al., 2016]
  - ResNet + parallel routing strategy inspired by Inception module
- [Shao et al., 2016]: combination of models (Inception, ResNet,…)
- SENet [Hu et al., 2017]: feature recalibration module to learn how to re-weight *feature maps*.





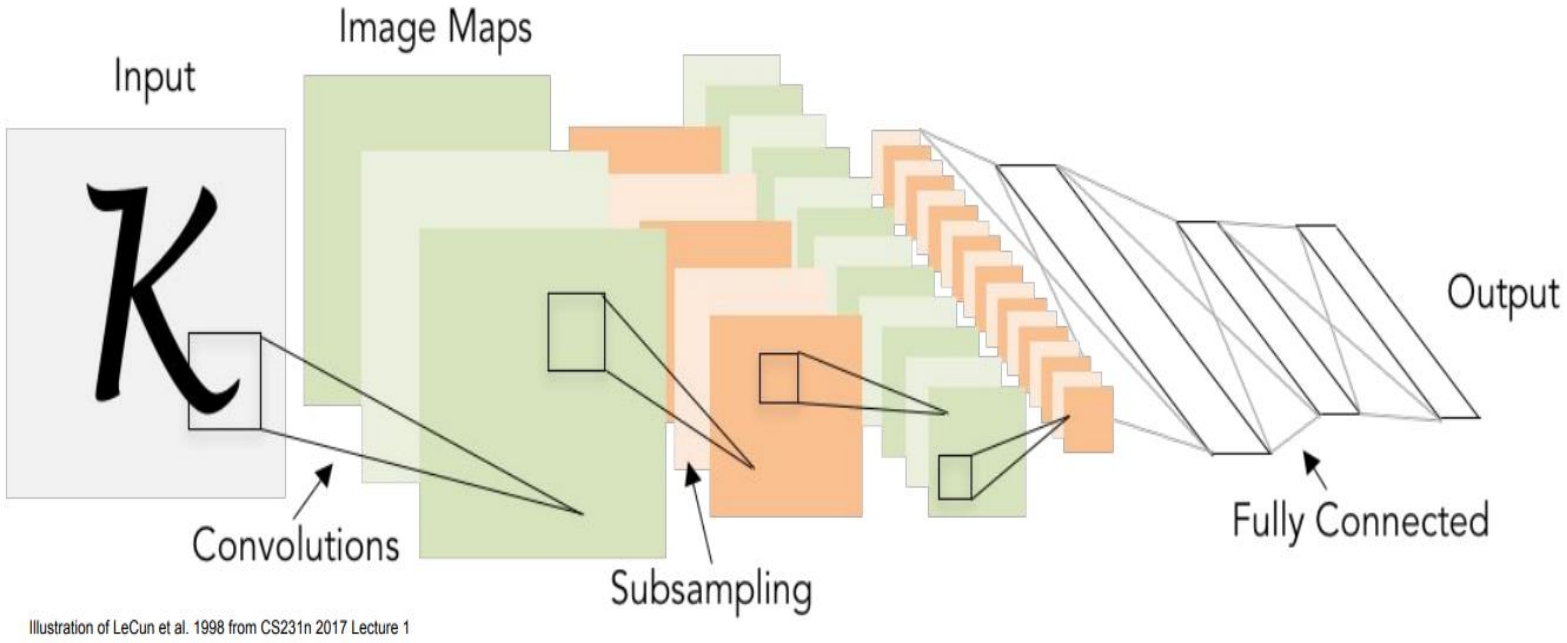
Slide credit: Fei-Fei Li, Justin Johnson and Serena Yeung. http://cs231n.stanford.edu/slides/2018/cs231n_2018_lecture09.pdf

Main conclusions



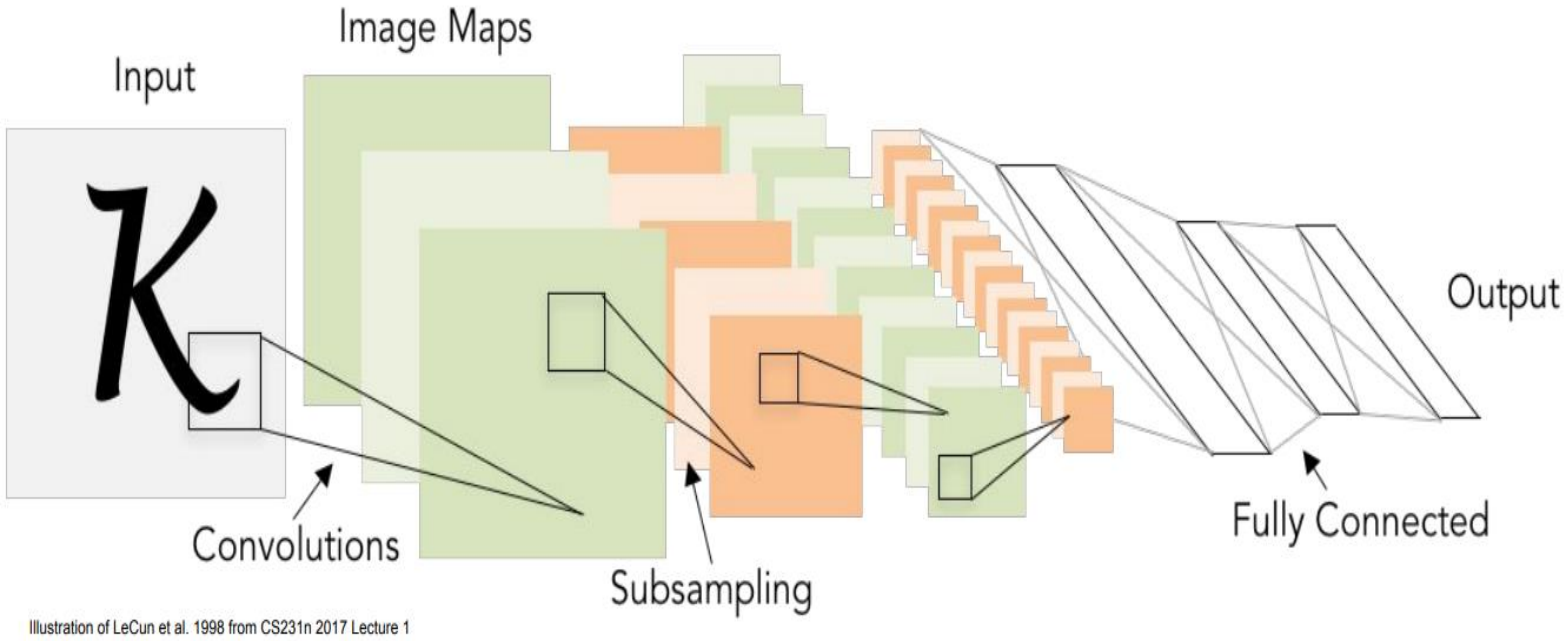
- Convolutional networks (ConvNets) are neural networks commonly applied to image analysis.
- They use convolutions to extract visual features (*feature maps*).
- Masks/kernels/filters are learned (network weights).

Main conclusions (2)



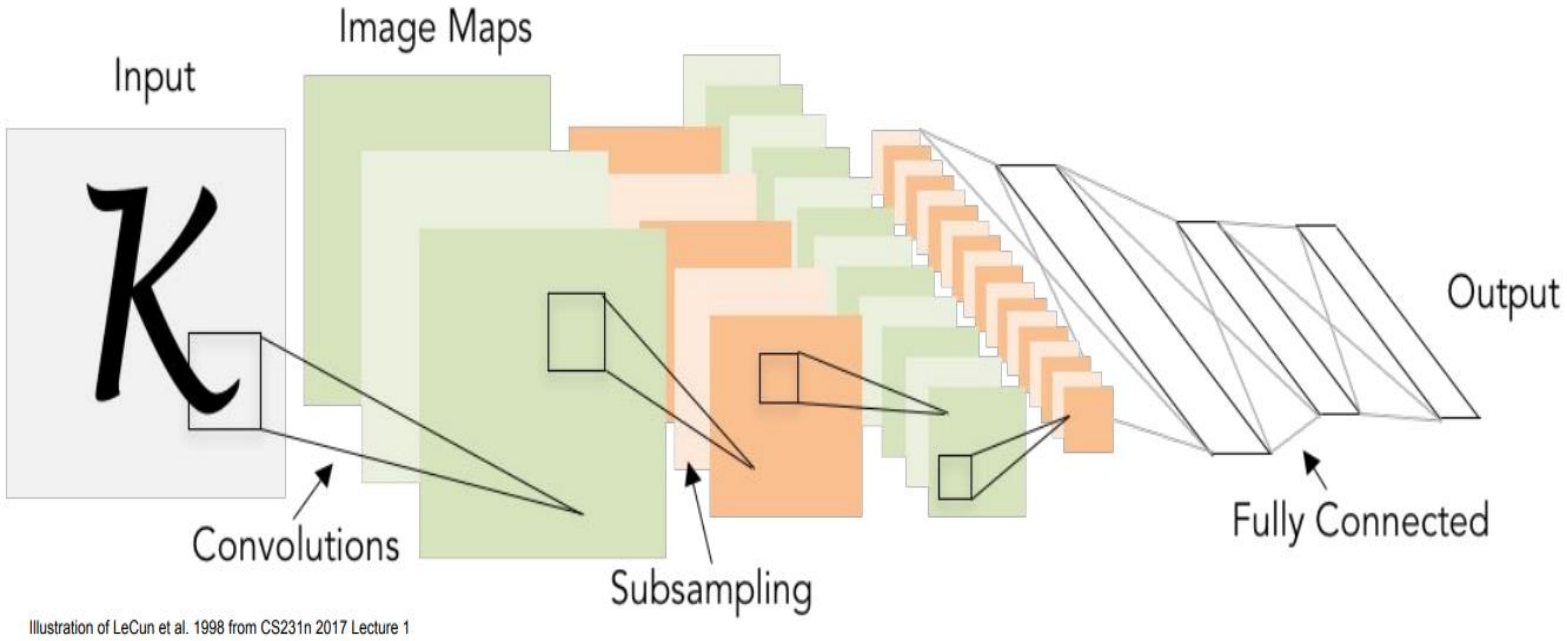
- They may or may not contain *fully connected layers*.
- They include *pooling* layers to reduce dimensionality (*subsampling*).

Main conclusions (3)



- They use the image itself (pixels directly) as input.
- Linear (convolution) and non-linear (activation function) operations are combined.
- Progressively more complex representations are learned.

Main conclusions (and 4)



- Training is *end-to-end,* minimizing one function (*loss*).
- When learning is supervised, it usually requires lots of data.

Some potentially useful references

- Montavon, Grégoire, Geneviève Orr, and Klaus-Robert Müller, eds. *Neural networks: tricks of the trade*. Vol. 7700. Springer, 2012.
  - Bengio, Yoshua. “Practical recommendations for gradient-based training of deep architectures”, pages 437-478.
  - Bottou, Léon. “Stochastic gradient descent tricks”, 421-436.
  - LeCun, Yann, et al. “Efficient backprop”, pages 9-50.
  - Neuneier, Ralph, and Zimmermann, Hans Georg. “How to train neural networks”, pages 369-418.
- Deep Learning Tips and Tricks cheatsheet: https://stanford.edu/~shervine/teaching/cs-230/cheatsheet-deep-learning-tips-and-tricks#

Review of Deep Learning and Convolutional Neural Networks

Pablo Mesejo and Víctor Vargas
Universidad de Granada
Departamento de Ciencias de la Computación e Inteligencia Artificial

UNIVERSIDAD DE GRANADA