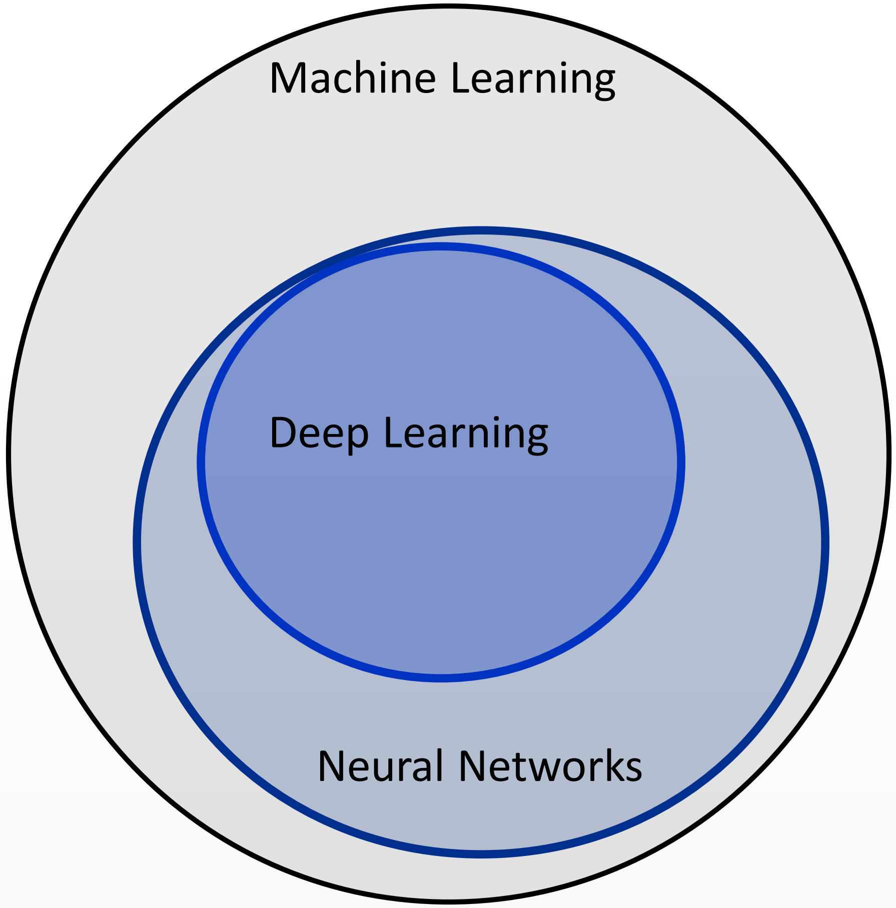
Basic Image Processing Algorithms

PPKE-ITK

Lecture 13.

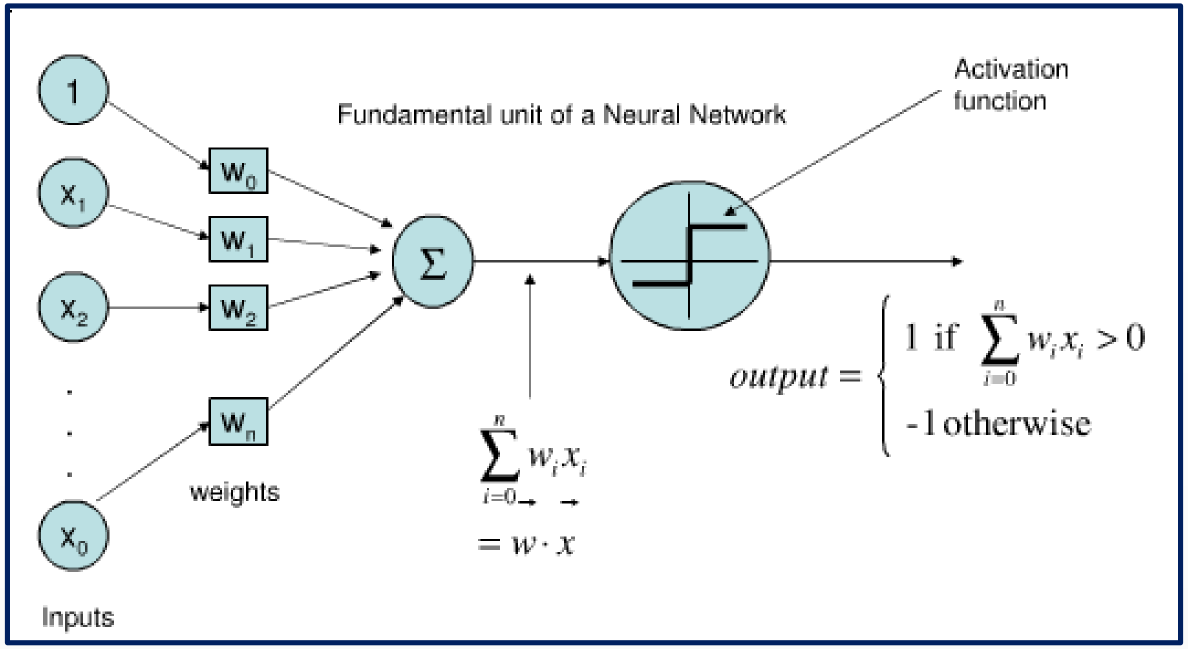
Introduction to Deep Learning

What is Deep Learning?



History of Neural Networks

- 1943: McCulloch and Pitts proposed a model to mimic how the brain operates.
- 1958: Rosenblatt introduced the Perceptron



Fundamental unit of a Neural Network
Activation function
1
x_1
x_2
x_0
Inputs
w_0
w_1
w_2
w_n
weights
Σ
$\sum_{i=0}^{n} w_i x_i = \vec{w} \cdot \vec{x}$
$output = \begin{cases} 1 \text{ if } \sum_{i=0}^{n} w_i x_i > 0 \\ -1 \text{ otherwise} \end{cases}$

Perceptron Model

History of Neural Networks

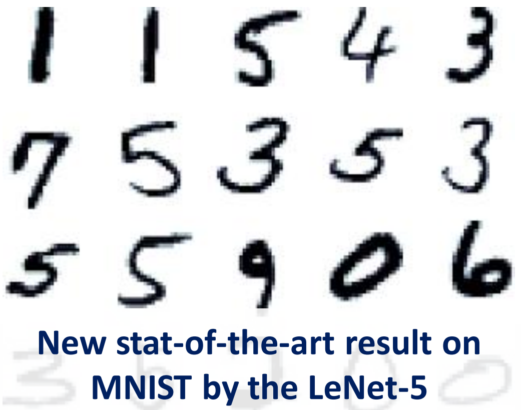
- Thanks to the success of the Perceptron model, there was a big hype around AI (not unlike now!!)
- 1969: It was shown that a perceptron may fail to separate seemingly simple patterns (e.g. cannot learn the XOR function).
- Research in the area nearly stopped completely (AI winter)
- 1974: Paul J. Werbos introduced backpropagation algorithm: efficient training of multi layer networks.

History of Neural Networks

- 1989-98: Convolutional Neural Networks in action:

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, Yann; Léon Bottou; Yoshua Bengio; Patrick Haffner (1998). "Gradient-based learning applied to document recognition„ Proceedings of the IEEE.86 (11): 2278–2324.

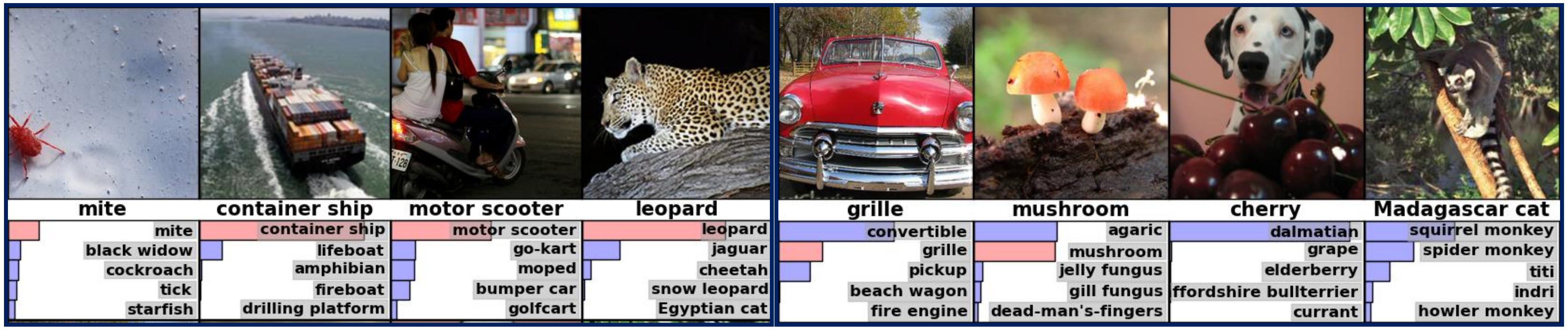

New stat-of-the-art result on MNIST by the LeNet-5

- Since 2012 there is a huge buzz around neural networks again.
- Recap: What happened in 2012?
  - The ImageNet challenge was won by a deep neural network architecture, the AlexNet:

Krizhevsky, A., Sutskever, I. and Hinton, G. E. „*ImageNet Classification with Deep Convolutional Neural Networks*” NIPS 2012: Neural Information Processing Systems, Lake Tahoe, Nevada

ImageNet Challenge 2012

- Task: For each image produce a list of at most 5 object categories in the descending order of confidence.



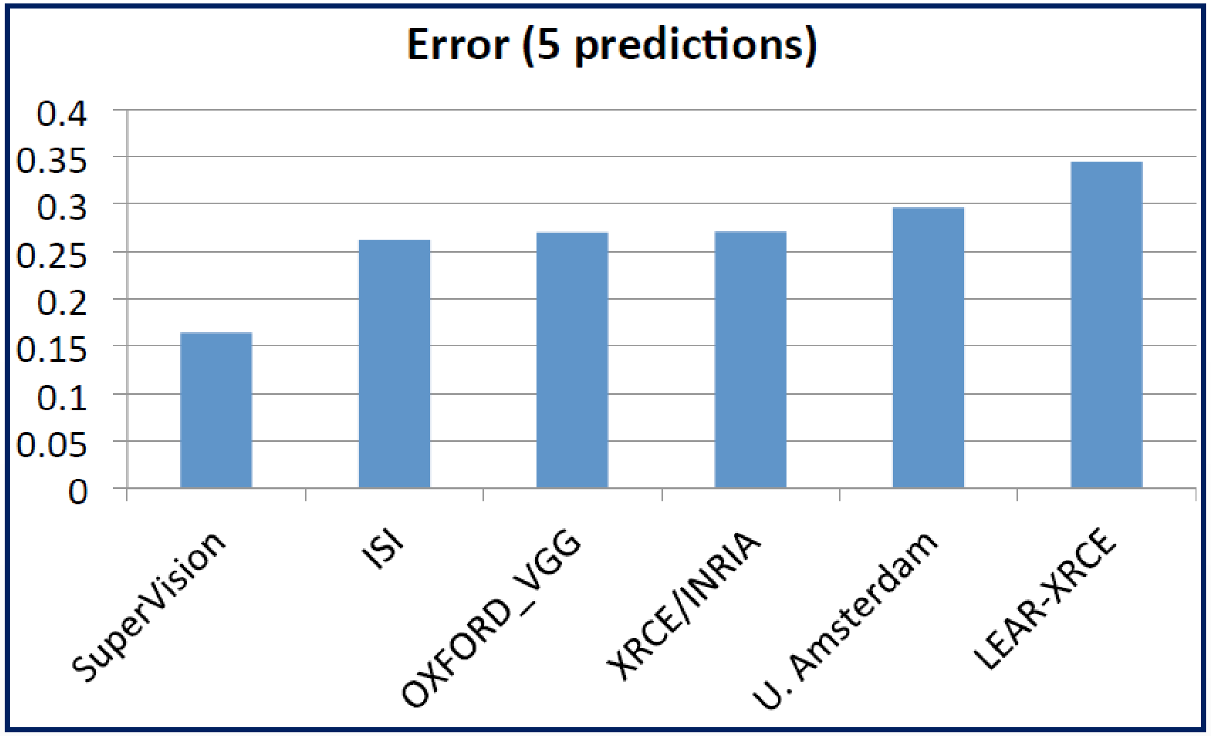
- The training data: 1.2 million images 1000 categories
- Summary of the Challenges:
  - A large number of images in training
  - A large number of classes
  - Diversity of classes
  - Diversity of images within classes

ImageNet Challenge 2012

Test images of „Hammer”

ImageNet Challenge 2012

- Results:

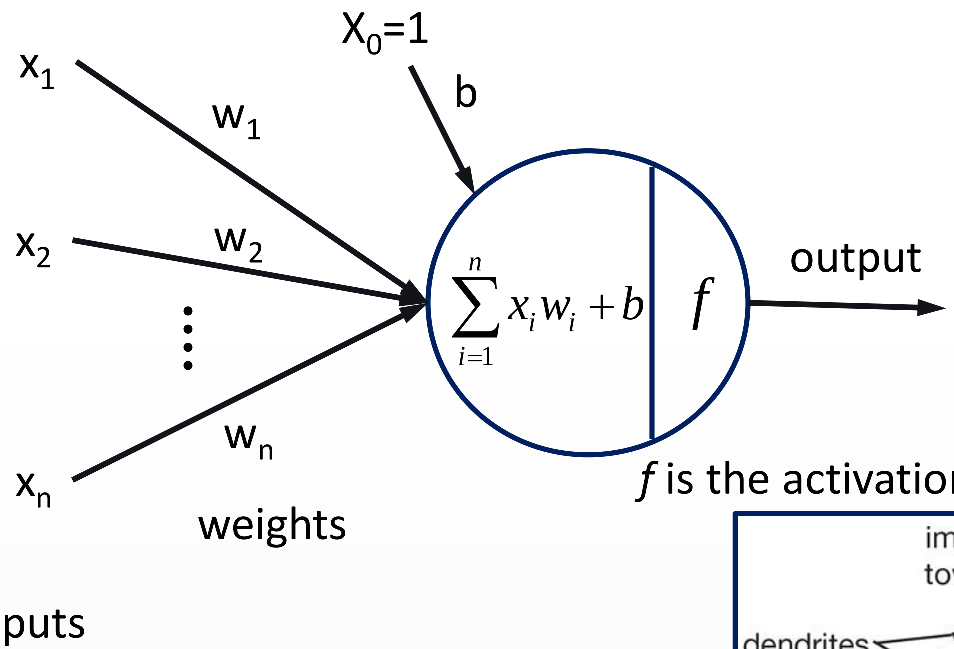


- Since then it is widely used in computer vision (and also in many other fields)

Artificial Neural Networks

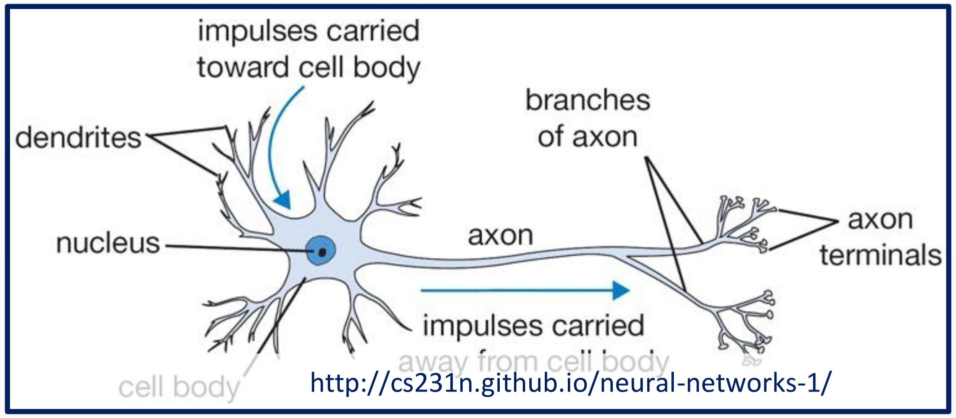
Artificial Neural Networks

- Simplest ANN: One Neuron



$$y = f(z) = f\left(\sum_{i=1}^{n} x_i w_i + b\right) = f(xw+b)$$

f is the activation function



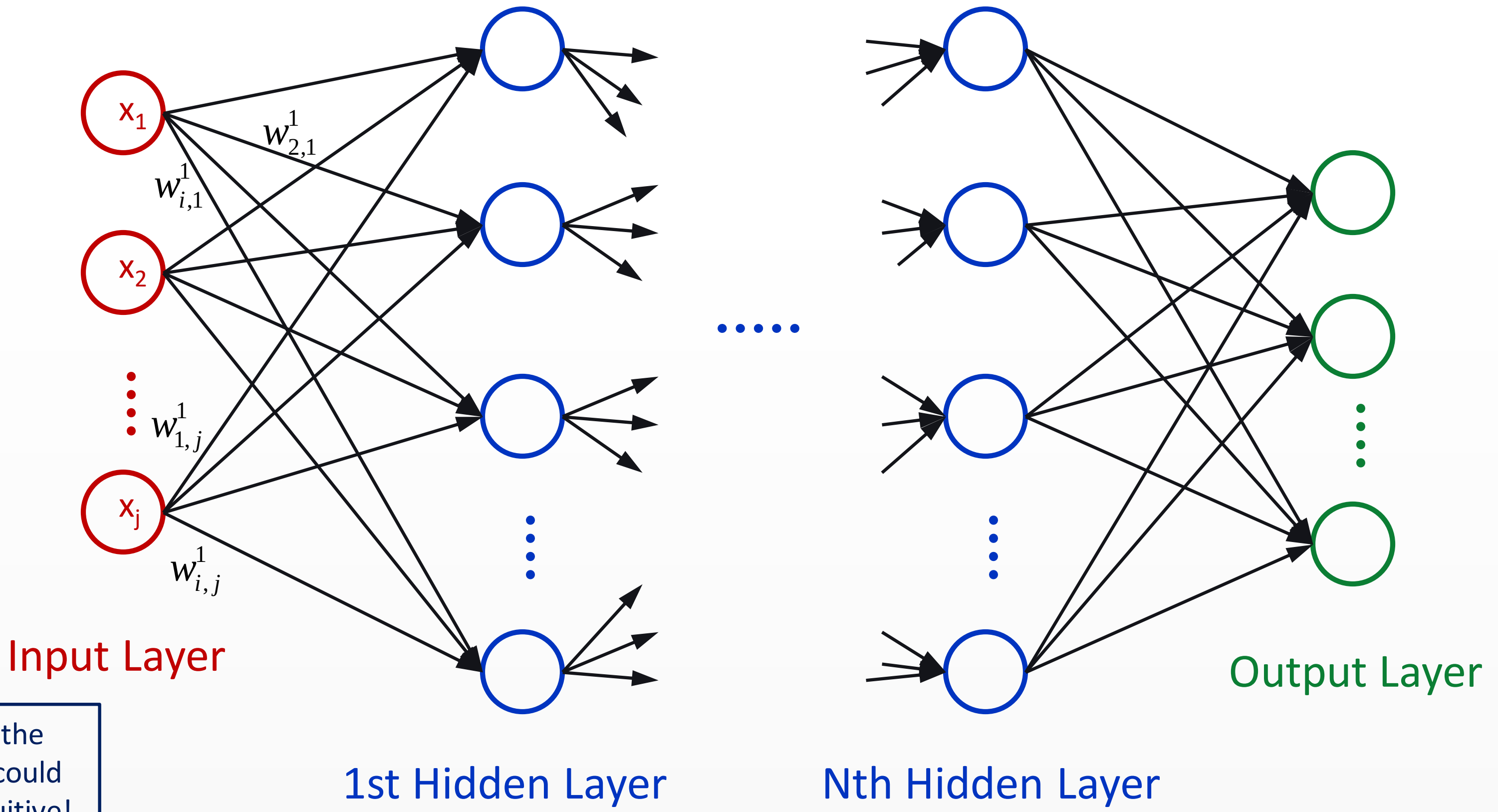
impulses carried toward cell body
dendrites
branches of axon
nucleus
axon
axon terminals
impulses carried away from cell body
cell body

http://cs231n.github.io/neural-networks-1/

Note: If *f* is a sigmoid function then this is exactly the logistic regression.

Artificial Neural Networks

- These simple computational units (the artificial neurons) can be organized into networks. (Like the „real” neurons in the human nervous system)



Watch out for the indexation, it could be counterintuitive!

Artificial Neural Networks

- Feed-Forward network
  - Contains only forward connections: the neurons in layer *l* are only connected to the neurons in layer *l+1*.
  - No backward or within-layer connections.
- Fully Connected layers:
  - ***All*** units in layer *l* are connected to ***all*** units in layer *l+1*.
- Each neuron has a bias connection:
  - Acts as a connection where the input is always 1.
- ANNs are universal function approximators [Cybenko 1989]

  „The universal approximation theorem states that a feed-forward network with a single hidden layer containing a finite number of neurons (i.e., a multilayer perceptron), can approximate continuous functions on compact subsets of R^n." (Wikipedia)

Artificial Neural Networks - Forward Pass

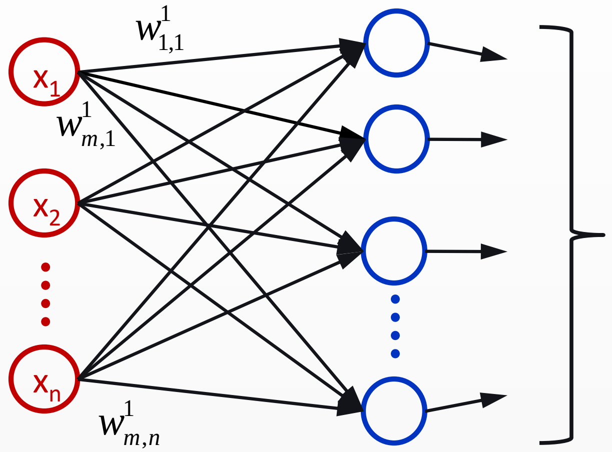
- The *Forward Pass* is the calculation of the response of the network to an input.
- Assuming you have a trained network, all trainable parameters (θ) are tuned for the task:

$$F\left(\cdot , \theta \right) = \begin{bmatrix} 0.91 \\ 0.03 \\ 0.06 \end{bmatrix}$$

$$F\left(\cdot , \theta \right) = \begin{bmatrix} 0.04 \\ 0.95 \\ 0.01 \end{bmatrix}$$

$$F\left(\cdot , \theta \right) = \begin{bmatrix} 0.15 \\ 0.08 \\ 0.77 \end{bmatrix}$$

- It can be calculated a series of matrix-vector operation:



$$\begin{bmatrix} a^1_1 \\ a^1_2 \\ \vdots \\ a^1_m \end{bmatrix} = f\left(\begin{bmatrix} z^1_1 \\ z^1_2 \\ \vdots \\ z^1_m \end{bmatrix} \right) = f\left(\begin{bmatrix} - & w^1_1 & - \\ - & w^1_2 & - \\ & & \\ - & w^1_m & - \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} + \begin{bmatrix} b^1_1 \\ b^1_2 \\ \vdots \\ b^1_m \end{bmatrix} \right)$$

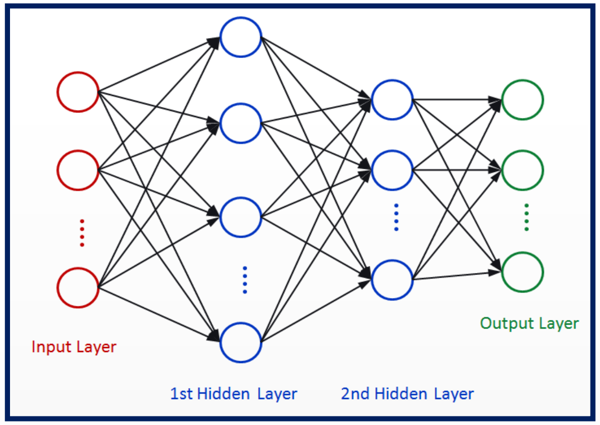
$$a^{(1)} = f\left(z^{(1)}\right) = f\left(W^{(1)}x + b^{(1)}\right)$$

Artificial Neural Networks - Forward Pass

$I \in$ set of images

Extracted Features: HOG/SIFT/Pixels

$x = \xi(I)$



$$a^{(1)} = f\left(z^{(1)}\right) = f\left(W^{(1)}x + b^{(1)}\right)$$
$$a^{(2)} = f\left(W^{(2)}a^{(1)} + b^{(2)}\right)$$
$$z^{(3)} = W^{(3)}a^{(2)} + b^{(3)}$$

Predicted Labels: Cat/Car/Dog/Truck

$\tilde{y} = a^{(3)} = g\left(z^{(3)}\right)$

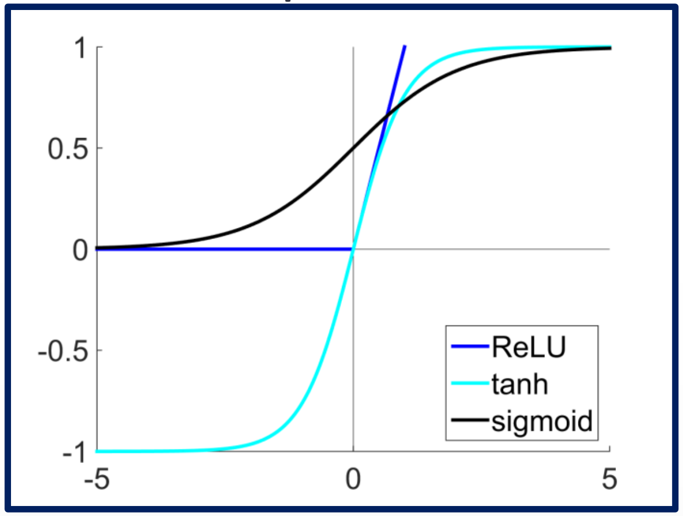
g():
can be a softmax function for classification,
or it can be omitted in case of regression.

Artificial Neural Networks - Activation Functions

- The activation function is a non-linear function applied in each neuron on ***z***, the weighted sum of the neuron's input.
- Commonly used activation functions:
  - Sigmoid
  - Hyperbolic tangent (tanh)
  - **Rectified Linear Unit (ReLU,** $y = \max(0, x)$**)**
  - ...

↑

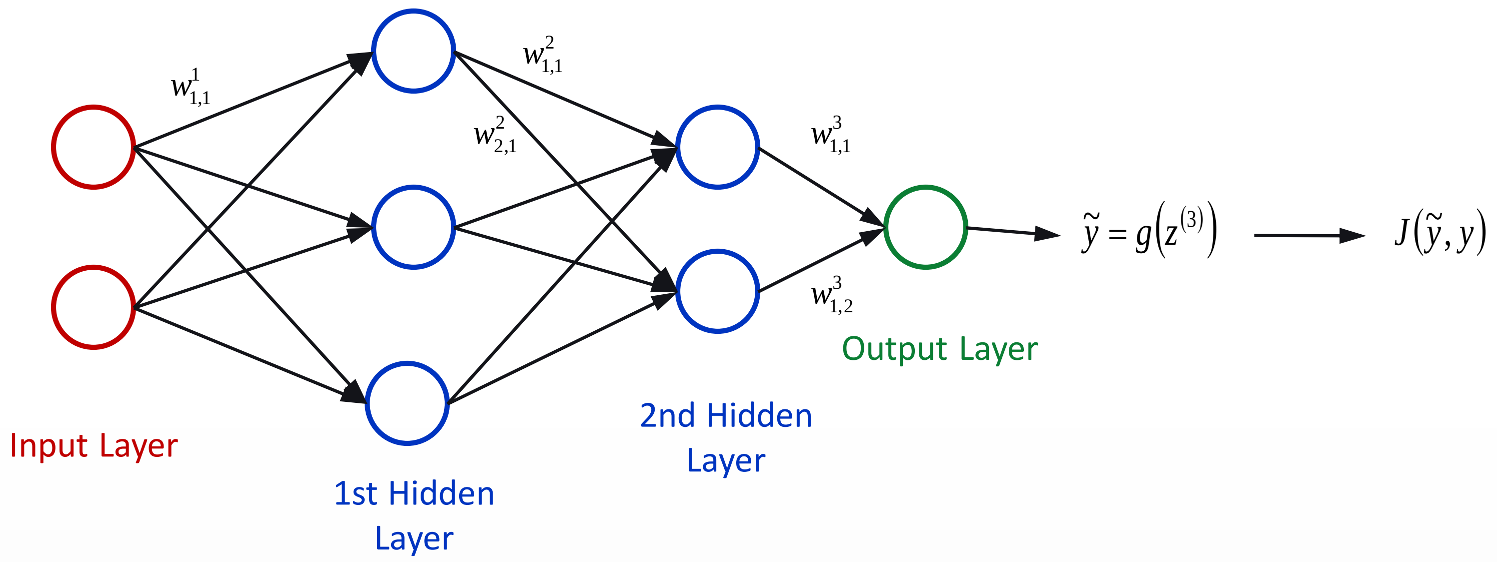
Lately ReLU is the default choice for deep nets.



  - What happens if we don't use activation function? (== If the activation function is linear?)
    - The composition of linear functions is a linear function. It would be equivalent to a 1-layer logistic/linear regression.

Network Training

Training of the Network



$$W^{(1)} = \begin{bmatrix} w_{1,1}^{1} & w_{1,2}^{1} & w_{1,3}^{1} \\ w_{2,1}^{1} & w_{2,2}^{1} & w_{2,3}^{1} \end{bmatrix}$$

$$b^{(1)} = \begin{bmatrix} b_1^1 & b_2^1 & b_3^1 \end{bmatrix}$$

- Forward propagation: $\tilde{y} = g\left(W^{(3)} f\left(W^{(2)} f\left(W^{(1)} x + b^{(1)}\right) + b^{(2)}\right) + b^{(3)}\right)$
- Loss function:
  - For classification, we can use cross-entropy loss (see softmax)
  - For regression, we can use L2 loss (see linear regression)
- How can we minimize the loss?

Training of the Network

- To optimize the weights and biases so that the *loss is minimal* we use ***Gradient Descent*** algorithm:
  - The weights and biases are initialized as small random numbers.
  - Each parameter (weights and biases) are modified simultaneously in each iteration:

$$w_{i,j}^{(l)} = w_{i,j}^{(l)} - \alpha \frac{\partial}{\partial w_{i,j}^{(l)}} J \qquad b_i^{(l)} = b_i^{(l)} - \alpha \frac{\partial}{\partial b_i^{(l)}} J$$

- How can we compute the *gradient of the loss wrt each parameter*?
- With ***Backpropagation algorithm***:
  - It is based on the application of the *chain rule:*

$$F(x) = f(g(x)) \;\Rightarrow\; F'(x) = f'(g(x))g'(x)$$

  - This is the most efficient way to compute the exact gradients.

Training of the Network

- Practical considerations:
  - Initialization with small random numbers (instead of all zeros) to *break the symmetry*, otherwise all the hidden units would learn the same function of the input.
  - Training Strategies:
    - Adjust the weights based on the ...
      - ...error of one training sample (***Stochastic Gradient Descent***)
      - ...average error of all the training samples (***Batch Gradient Descent***)
      - ...average error of a few dozens/hundreds of training samples (***Mini-Batch Gradient Descent***)
        - The average error on the mini-batch usually approximates well the average error on all the training samples, but it is much faster.
        - Still we will use all the training samples many times: after we go through all mini-batches (== we complete one epoch), we reshuffle the samples, divide them into mini-batches again and start the next epoch.

Training of the Network

- Regularization:
  - The goal is to prevent the network from overfitting.
  - Commonly used types of regularization:
    - L2 regularization:
      - An extra term is added to the cost to penalize peaky weights:

$$\frac{\lambda}{2}\|w\|_{L2} = \frac{\lambda}{2}\sum_{j=1}^{n} w_j^2 \longleftarrow \text{L2-norm of the weights}$$

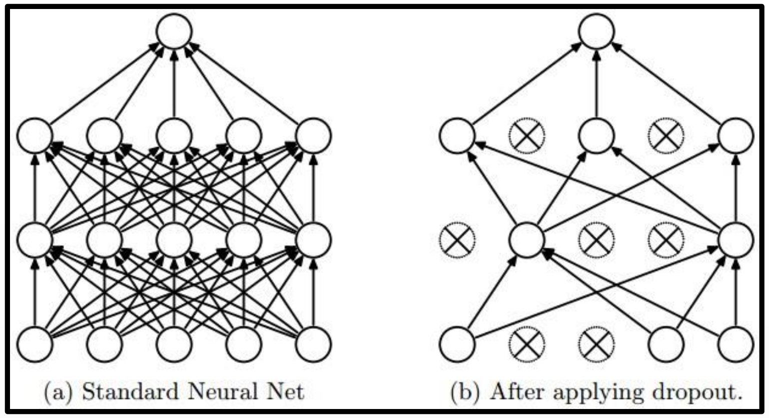
      - This regularization prefers diffuse weights.
    - L1 regularization:
      - Regularization term:

$$\frac{\lambda}{2}\|w\|_{L1} = \frac{\lambda}{2}\sum_{j=1}^{n} |w_j| \longleftarrow \text{L1-norm of the weights}$$

      - Favours sparse weight vectors. (Sparse means that only a few elements in the vector are non-zero)

Training of the Network

- Regularization:
  - Dropout:
    - During training each neuron can be deactivated with a certain probability.
    - In each iteration of the training, a different sub-network is optimized.
    - In test time there is no dropout!



http://www.cs.toronto.edu/~rsalakhu/papers/srivastava14a.pdf

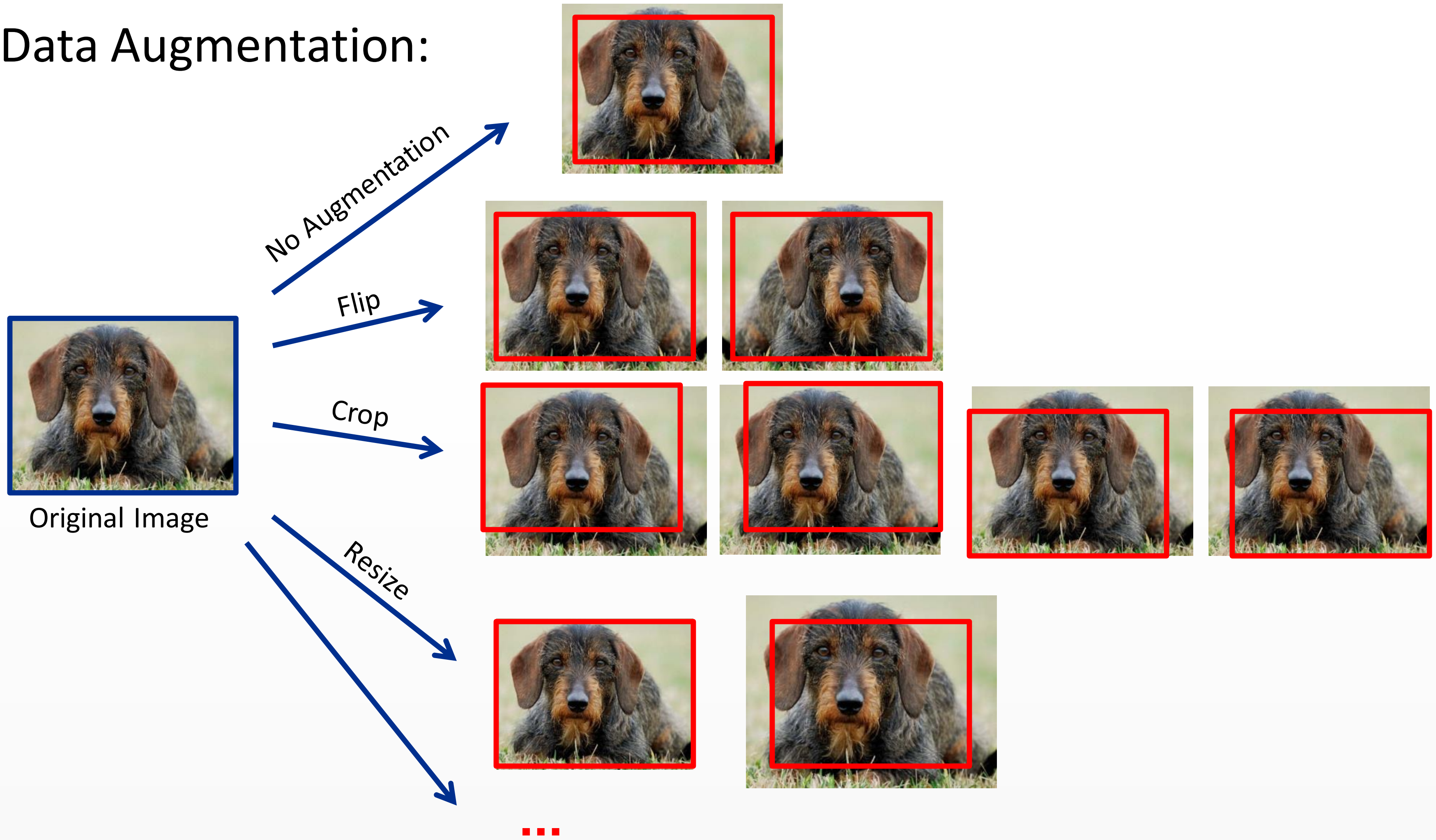
  - Early stopping:
    - Monitor the training:
      calculate performance metrics
      (accuracy, f1-score, ROC-AUC, etc.)
      on a separate validation set after each round.
    - Stop the training if the performance drops on the validation set!

Training of the Network

- In general we can say that the more parameters we use the larger the training dataset needs to be to be able to train without overfitting.
- Creating a large dataset takes time and expensive.
- Have to get the most out of the available data!
- Data Augmentation:
  - Increasing the size of the available training dataset by adding modified versions of the original samples.
- The augmentation has to be task specific (e.g. for handwritten digit recognition, flip is not a good idea..)

Training of the Network

Data Augmentation:

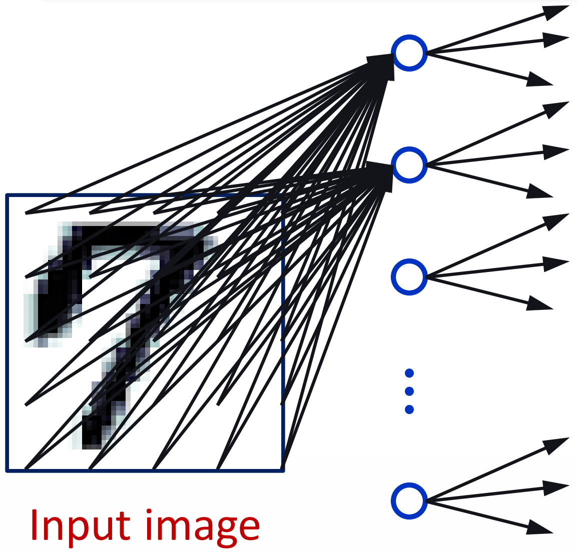


Convolutional Neural Networks

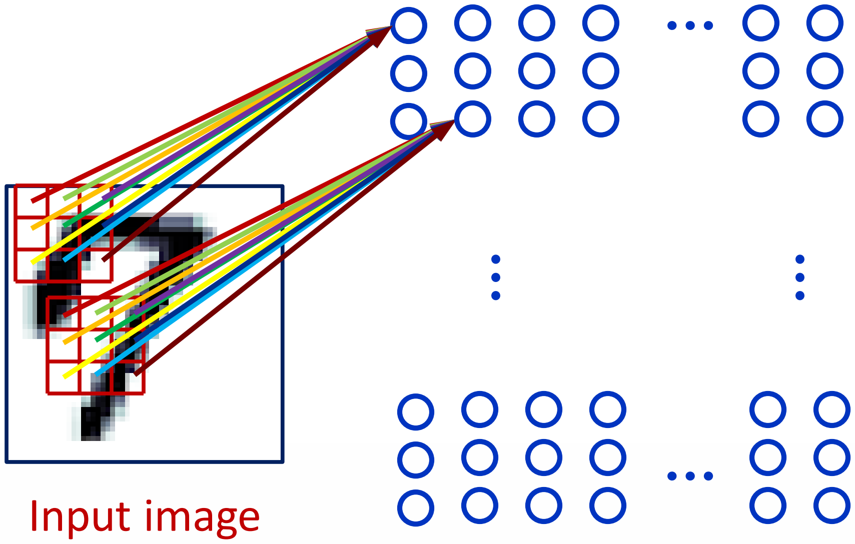
Convolutional Neural Network

- Fully connected layers on the raw pixels are not efficient. Why?
  - Regular neural nets don’t scale well to full images:
    - For a small image of 128x128 we have 16384 pixels.
    - If in the first hidden layer we have 100 neurons that is already 1.6 million parameters!!
  - To tune this many parameters we would need a lot of training samples to avoid overfitting, and it would be slow.
- Natural images...
  - contain strong local correlation, we should take it into account
  - have similar local statistics over the image, we could share the used parameters.
- Convolutional Neural Networks look at small parts of the image, one at a time using the same set of weights for each part.
- For CNN the weights are organized in 3D (width, height, depth).

Convolutional Neural Network



Fully Connected Layer

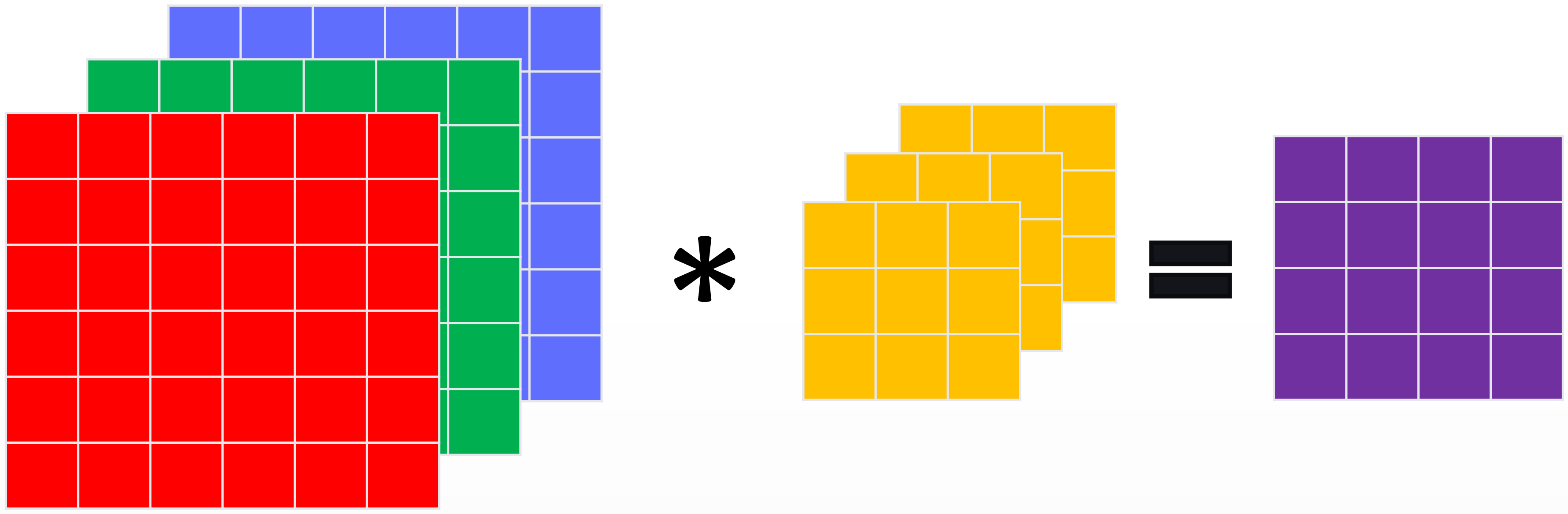


Convolutional Layer

Each pixel is one input for the network -> the number of parameters can be very high even for a medium sized image.

- The weights are reused in different parts of the image -> much less parameters.
- Practically convolutional kernels will be learned. Each kernel has 3 dimensions (only 2 are visualized above)

Convolution on volumes

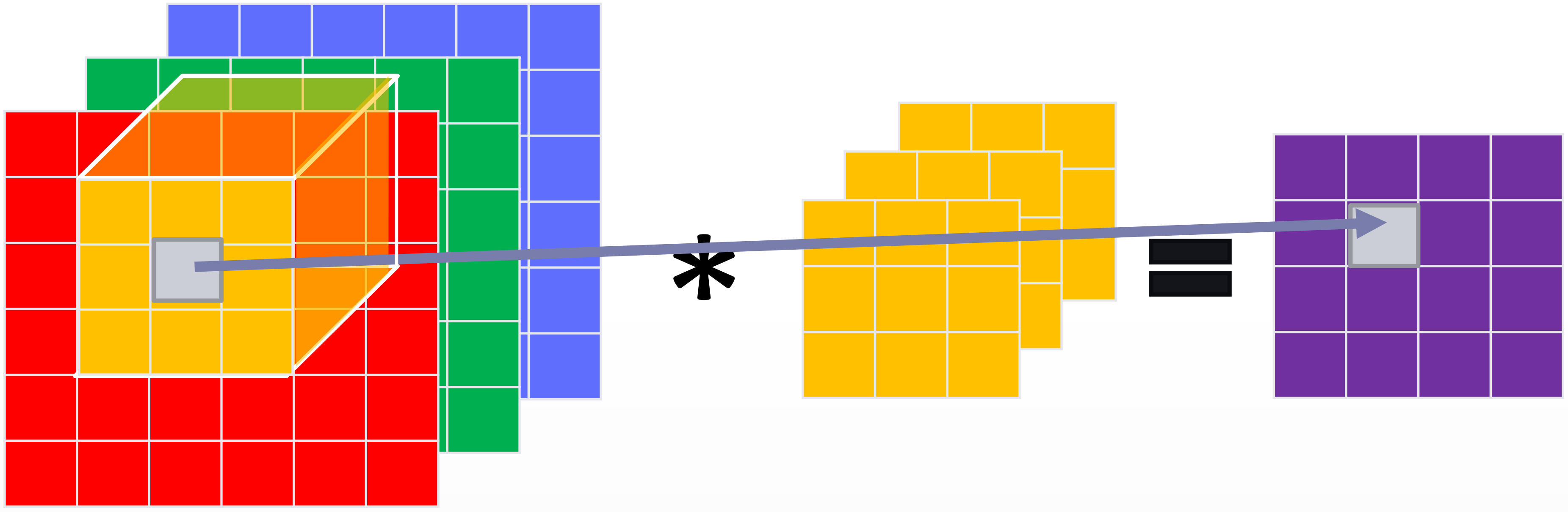


Input: n-channel array: $h \times w \times n$

Filter: n-channel kernel of size $k_h \times k_w \times n$

Output: 1-channel $(h - k_h + 1) \times (w - k_w + 1) \times 1$

Convolution on volumes



Input: n-channel array:

$h \times w \times n$

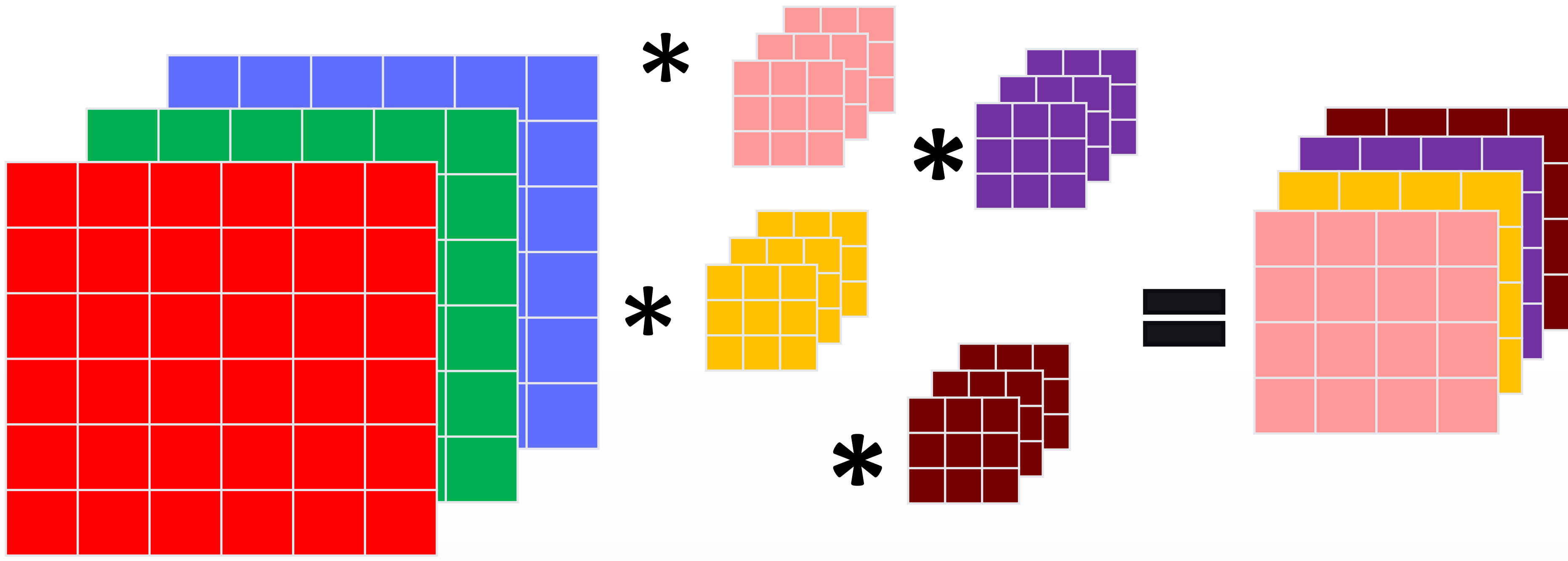
Filter: n-channel kernel of size

$k_h \times k_w \times n$

Output: 1-channel

$(h - k_h + 1) \times$
$(w - k_w + 1) \times 1$

A convolution layer with $f = 4$ filters



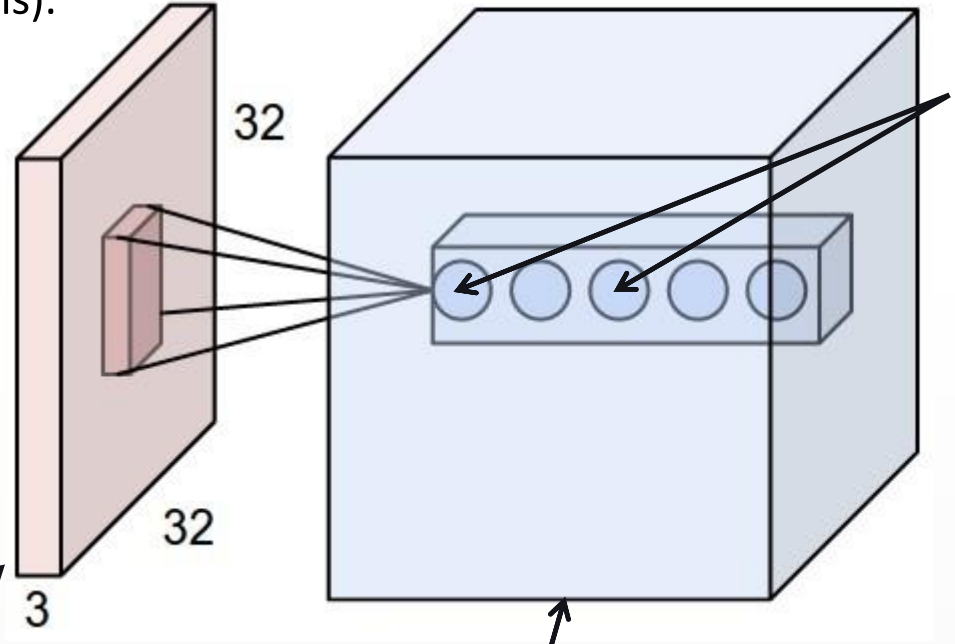
Input: n-channel array: $h \times w \times n$

f filter: each one is an n-channel kernel of size $k_h \times k_w \times n$

Output: f-channel array of size $(h - k_h + 1) \times (w - k_w + 1) \times f$

Convolutional Neural Network

Each neuron in the convolutional layer is connected only to a local region in the input volume spatially, but to the full depth (i.e. all color channels).



The neurons along the depth are all looking at the same region of the image. But they learn a different set of weights.

3-channel (R,G,B) input image with 32x32 pixels

Volume of neurons calculating 5 different convolutions

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

Convolutional Neural Network

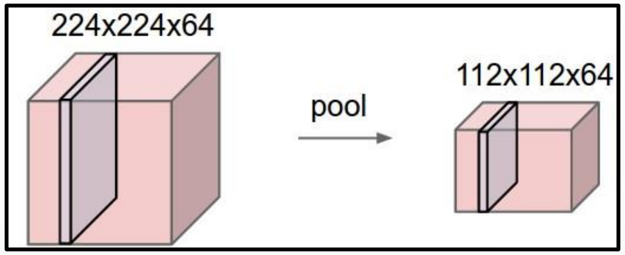
- Convolutional layers:
  - Has learnable weights and biases.
  - Has an activation function in the neurons.
  - The performed computation is a differentiable function.
  - It assumes that the input is an *image*:
    - Using this assumption it can be more efficient with less parameters

Pooling Layer

- It is common practice to insert a Pooling layer in-between successive convolutional layers.
- The goal is
  - to gain robustness against small changes in the location of a feature

The algorithm should recognize the wolf on both images, regardless of its location on the image.

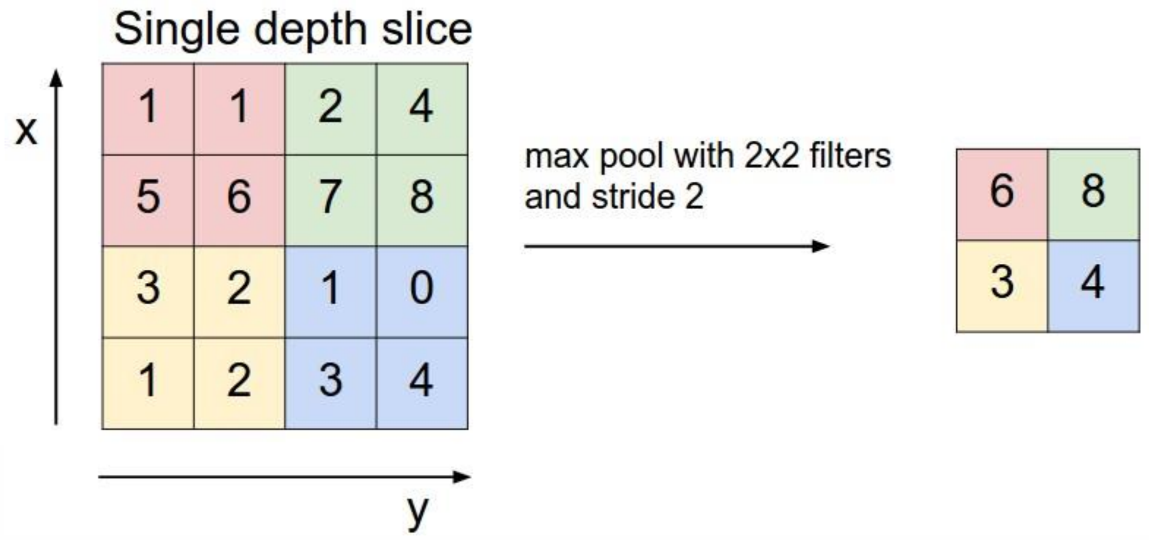
  - to reduce dimensionality (downsample along the spatial dimensions)



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

How mutch the feature maps are downsampled is defined by the parameters of the pooling.

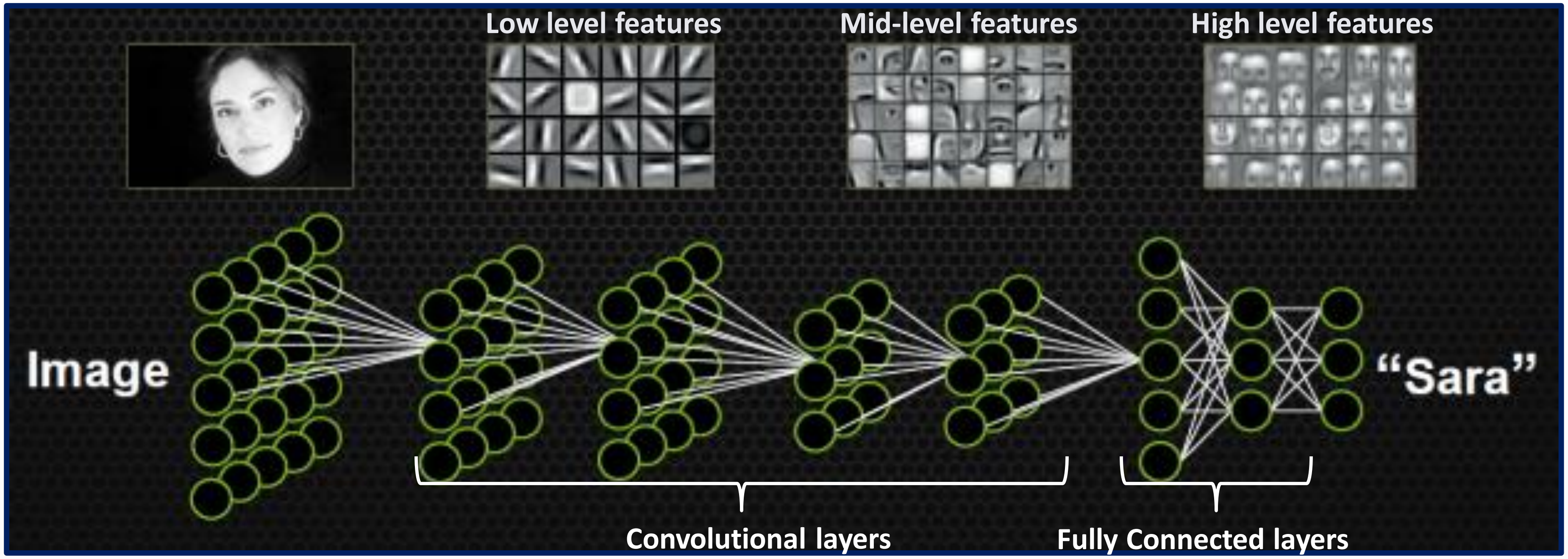
Pooling Layer



- Parameters of the pooling:
  - Filter size: usual sizes are 2x2 or 3x3
  - Stride: defines the spatial shift of the filter
- Introduces zero parameters since it computes a fixed function of the input.
- Types of pooling:
  - Max pooling
  - Average pooling

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

What does the CNN Learn?



Source of the image:
https://devblogs.nvidia.com/parallelforall/accelerate-machine-learning-cudnn-deep-neural-network-library/
H. Lee, R. Grosse, R. Ranganath, and A. Y. Ng. "Convolutional deep belief networks for scalable unsupervised learning of hierarchical representations." In ICML 2009.

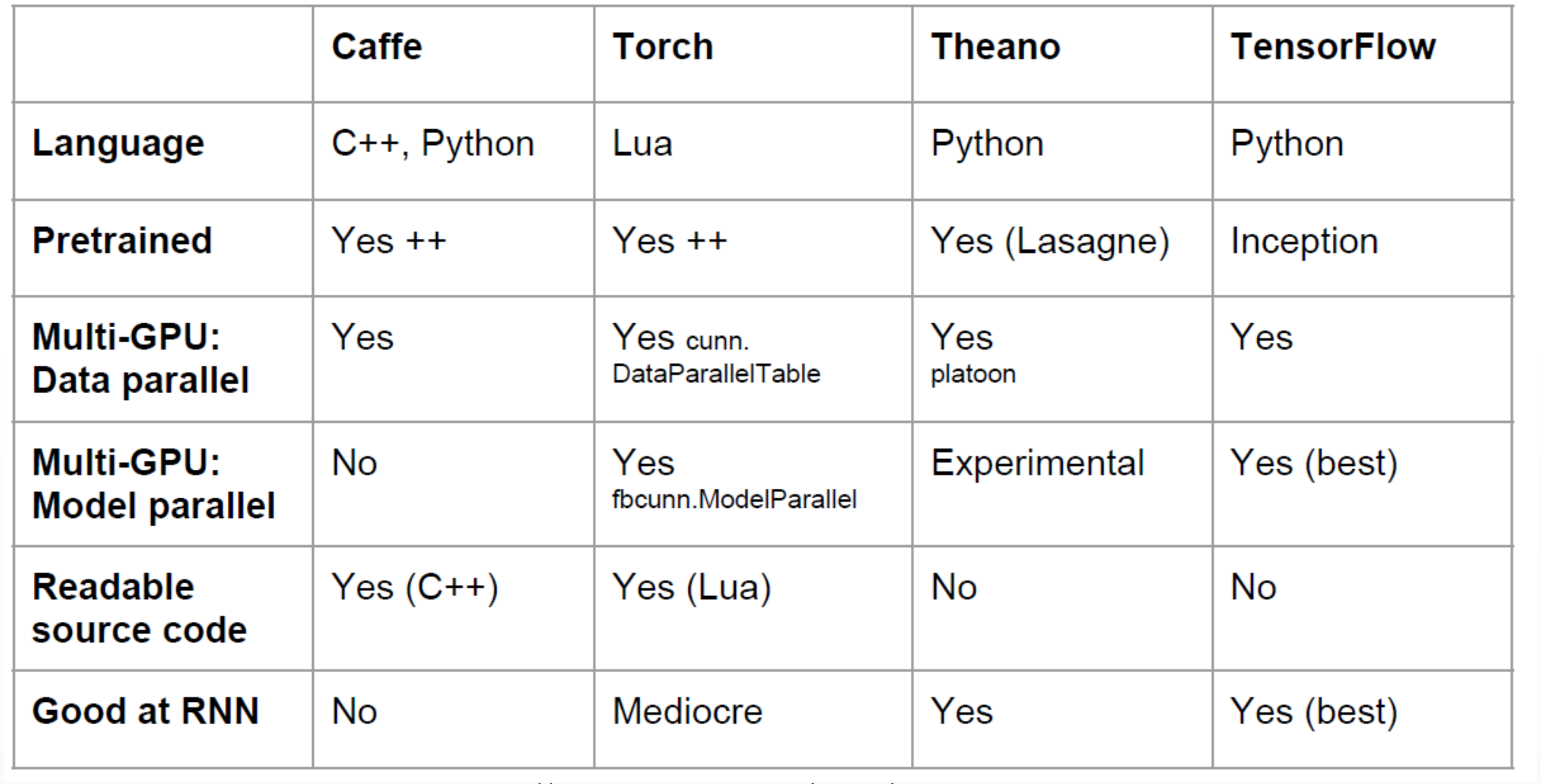
Transfer Learning

- For natural images the low level features are similar, hence the learned features will be very similar for different tasks.
- Big databases are not so easy to come by.
- The models that were trained on a big dataset could be partly reused on other tasks:
  - Reuse a convnet as *fixed feature extractor*: Take a trained model, remove its final fully connected layer and use the rest as a fixed feature extractor for a classifier (like a linear SVM or a softmax).
  - *Fine tuning*: use the network pre-trained on an other data and use it as initialization for the training on the data of interest. Usually this fine tuning is done with low learning rate, or even with the first few layers kept fixed.
- Using pre-trained models also helps against over fitting!

More info: http://cs231n.github.io/transfer-learning/

Deep Learning Frameworks

Frameworks



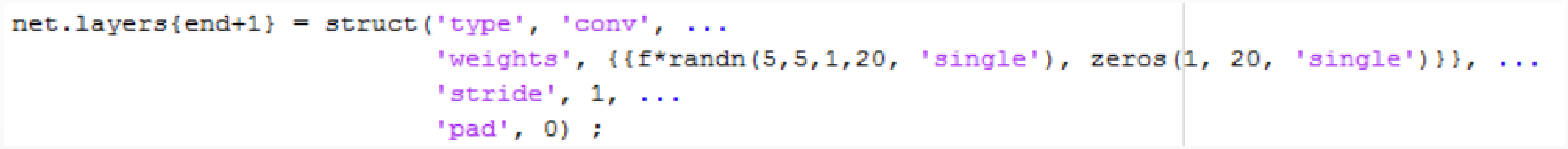
	Caffe	Torch	Theano	TensorFlow
Language	C++, Python	Lua	Python	Python
Pretrained	Yes ++	Yes ++	Yes (Lasagne)	Inception
Multi-GPU: Data parallel	Yes	Yes cunn. DataParallelTable	Yes platoon	Yes
Multi-GPU: Model parallel	No	Yes fbcunn.ModelParallel	Experimental	Yes (best)
Readable source code	Yes (C++)	Yes (Lua)	No	No
Good at RNN	No	Mediocre	Yes	Yes (best)

Source: http://cs231n.stanford.edu/slides/winter1516_lecture12.pdf

For more info: https://en.wikipedia.org/wiki/Comparison_of_deep_learning_software

Frameworks

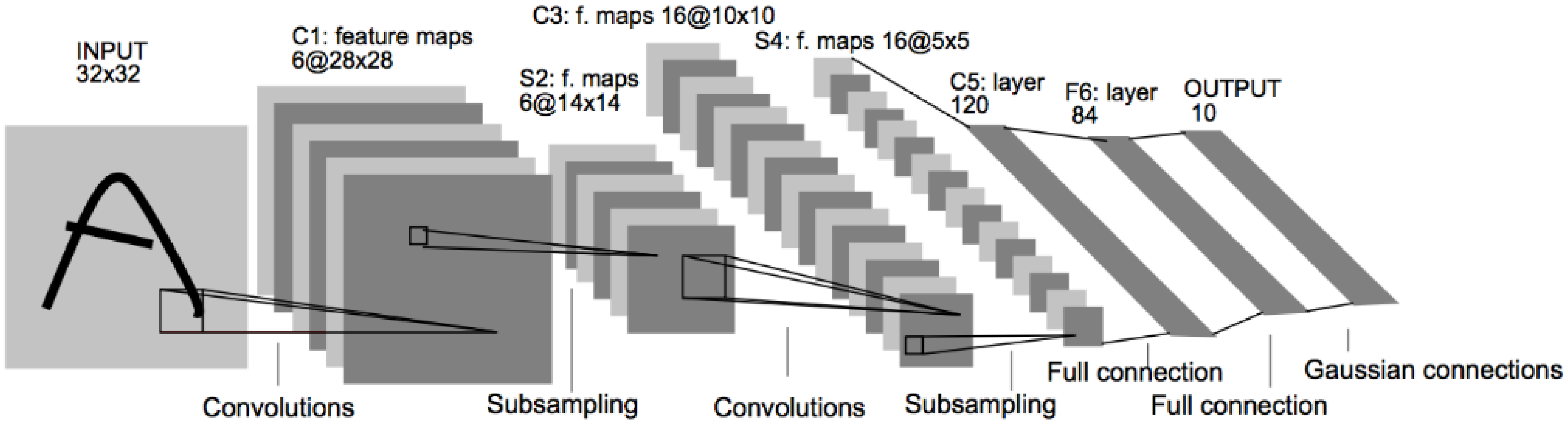
- MatConvNet:
  - Developed in Oxford for CNN training.
  - The network architecture is defined in a cell array of structs:
    - Each element in the array is one layer of the network:
      - Convolutional layer
      - Pooling layer
      - Activation layer
      - Dropout layer
      - Normalization layer
      - ...
    - Each layer is defined in one struct:



```
net.layers{end+1} = struct('type', 'conv', ...
                           'weights', {{f*randn(5,5,1,20, 'single'), zeros(1, 20, 'single')}}, ...
                           'stride', 1, ...
                           'pad', 0) ;
```

CNN Architectures

LeNet-5 Architecture



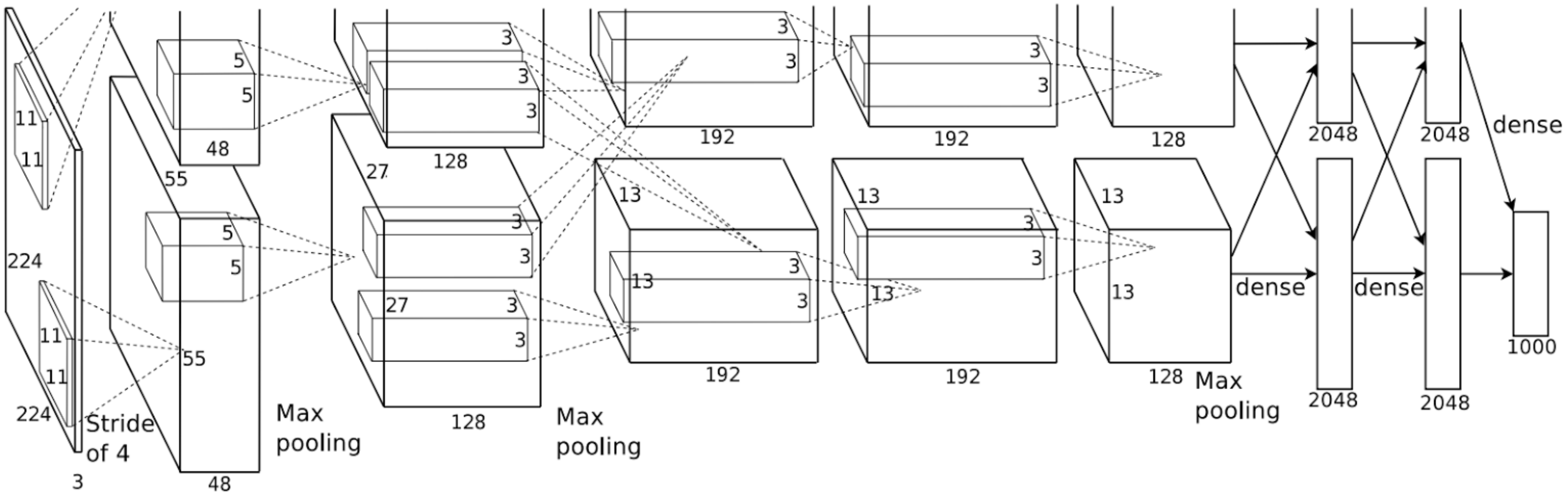
CNN called LeNet by Yann LeCun (1998)

- The first successfull application of convolutional networks.
- It was used for handwritten digit/zip code recognition.

http://yann.lecun.com/exdb/publis/pdf/lecun-98.pdf

Alexnet

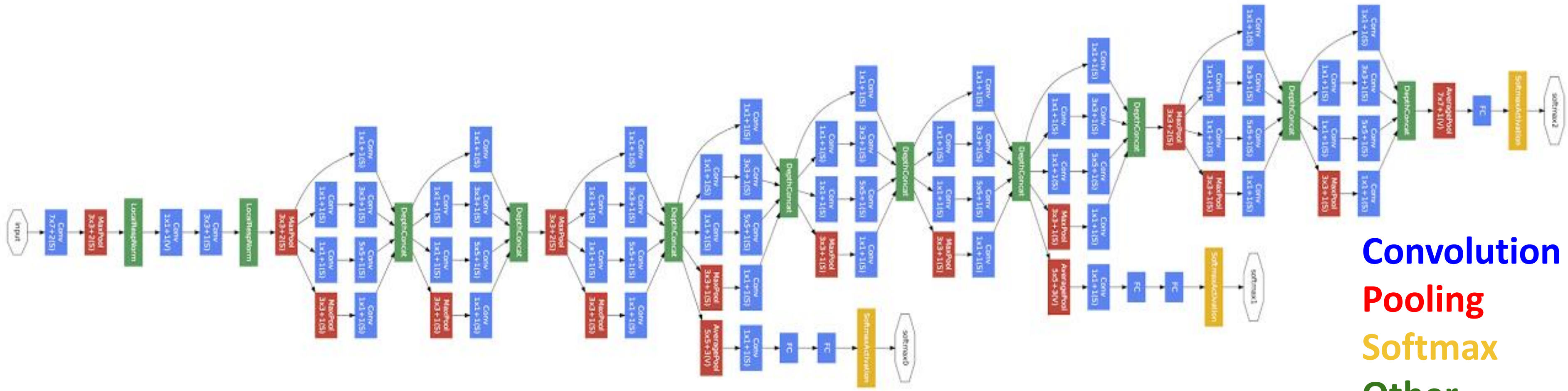
- Alex Krizhevsky's architecture that won the Imagenet in 2012.
- Similar to the LeNet architecture, but deeper.
- #parameters: 60 million



Krizhevsky, Alex, Ilya Sutskever, and Geoffrey E. Hinton. "Imagenet classification with deep convolutional neural networks." Advances in neural information processing systems. 2012

GoogLeNet

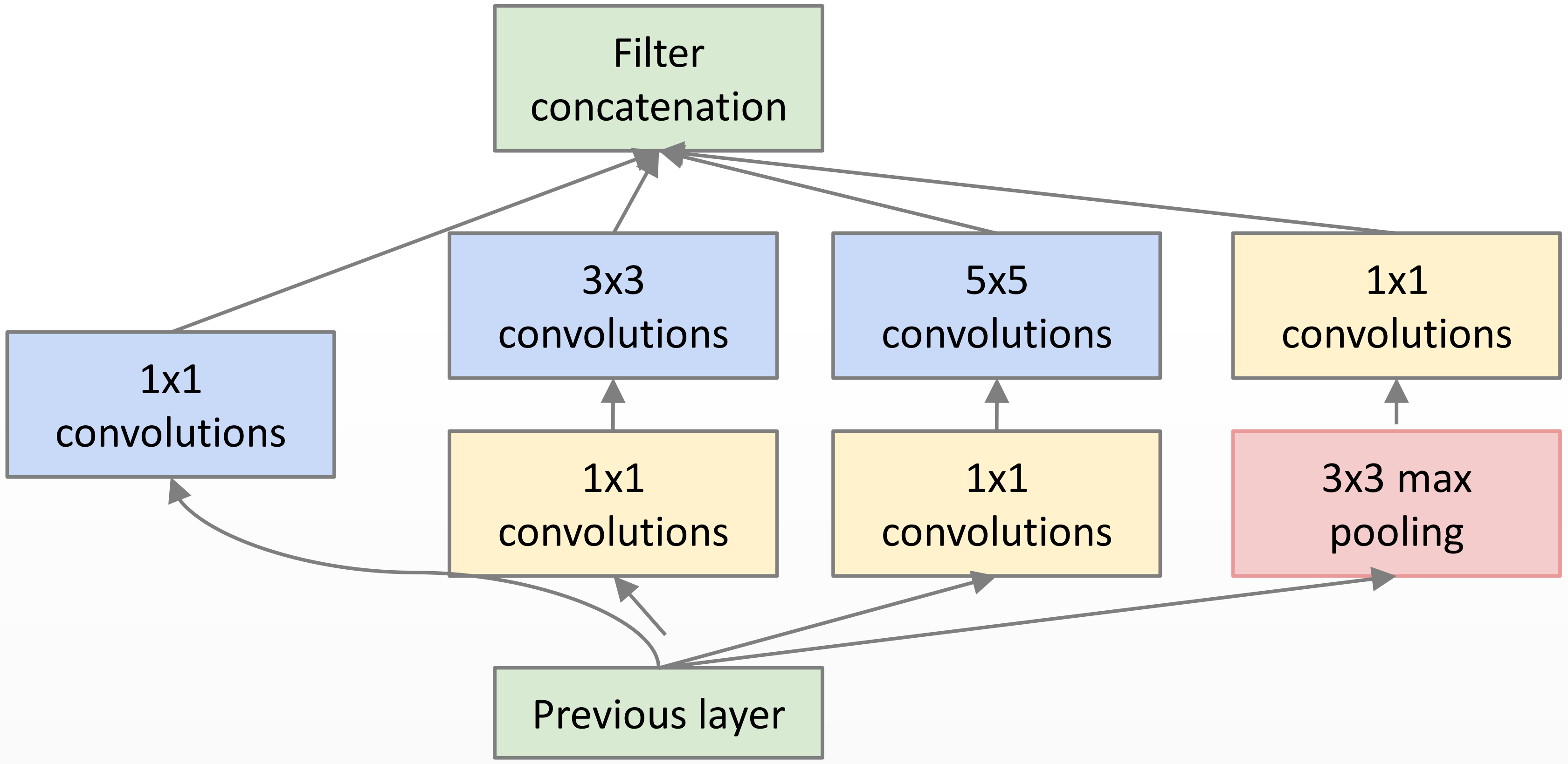
- Winner of the 2014 Imagenet challenge.
- Introduced the Inception module
- #parameters: 5 million



Convolution
Pooling
Softmax
Other

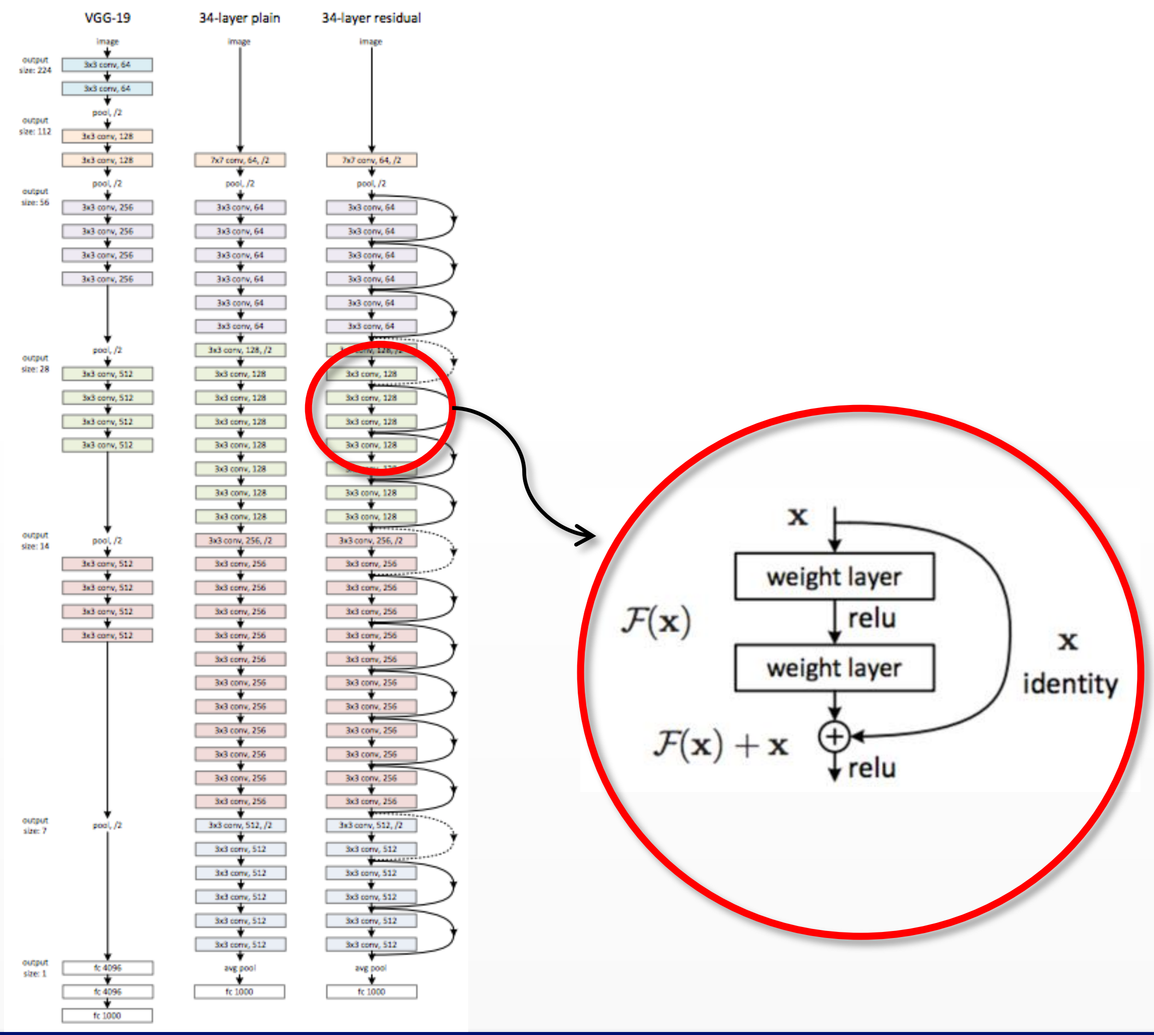
Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, Andrew Rabinovich, „Going Deeper withConvolution," Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2015.

GoogLeNet – Inception Module



Deep Residual Networks

- Winner in 2015.
- 152 layers
- Introduction of the Residual blocks
- It can be regarded as a special case of Highway networks.



Kaiming He, Xiangyu Zhang, Shaoqing Ren, Jian Sun. Deep Residual Learning for Image Recognition. 2015.

Further Reading

- http://cs231n.stanford.edu/
- http://deeplearning.stanford.edu/tutorial/
- http://neuralnetworksanddeeplearning.com/
- http://deeplearning.net/
- http://www.deeplearningbook.org/
- https://www.cs.toronto.edu/~hinton/absps/NatureDeepReview.pdf
- http://www.computervisionblog.com/2015/01/from-feature-descriptors-to-deep.html
- http://lamda.nju.edu.cn/weixs/project/CNNTricks/CNNTricks.html