Artificial Neural Networks

Machine Learning for Robotics and Computer Vision SS16

Caner Hazırbaş

hazirbas@cs.tum.edu

Technische Universität München
Department of Informatics
Computer Vision Group

May 13, 2016

Outline

Outline

Introduction

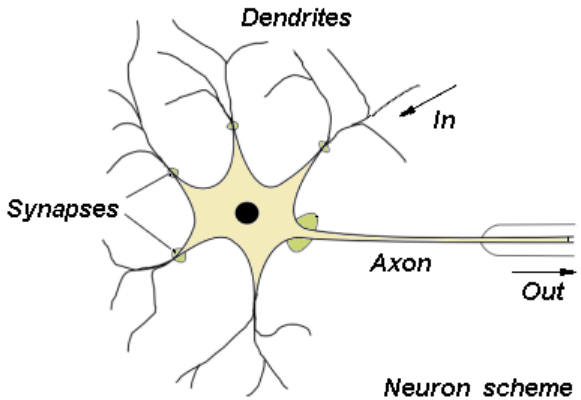
- Neural networks are inspired by the biological neurons
  - The human brain (10^{10} cells) is the archetype of neural networks
- Most commonly used classifiers in machine learning
- Easily adaptable to regression and multi-class problems
- Representation learning method
- History
  - Computational model in 1943 [McCulloch and Pitts, 1943]
  - Backpropagation in 1975 [Werbos, 1974]
  - Neocognitron in 1980 [Fukushima, 1980]
  - Convolutional Neural Networks in 1998 [Lecun et al., 1998])
  - AlexNet in 2012 [Krizhevsky et al., 2012]
  - VGG-Net in 2014 [Simonyan and Zisserman, 2015]
  - ResNets: Deep Residual Networks [He et al., 2015]
    - ResNet-50, ResNet-101, and ResNet-152

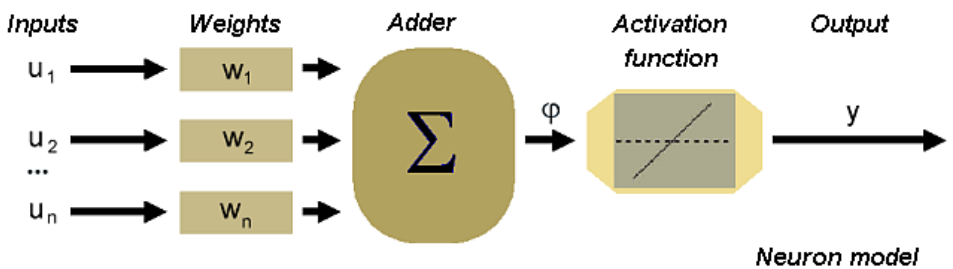
Outline

Perceptron

- One-neuron classifier
  - *linear* classifier
  - similar to SVMs
  - finds a hyperplane between classes
- Computational Model*

- Neuron scheme*





$$\varphi = \left(\sum_{i=1}^{n} w_i u_i + b \right)$$

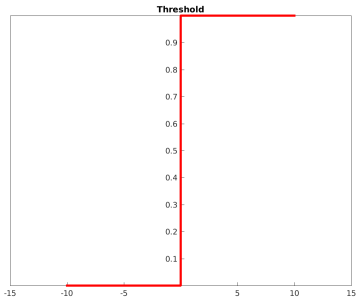
$$y = \sigma(\varphi)$$

* http://home.agh.edu.pl/~vlsi/AI/intro/

Activation Functions (σ)

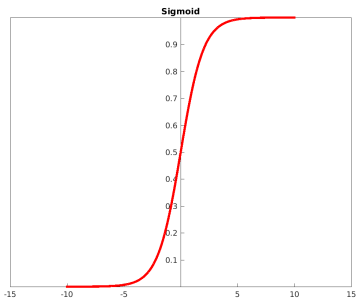
- Threshold Activation

$$\sigma(\varphi) = \begin{cases} 0 & \varphi \leq 0 \\ 1 & \varphi > 0 \end{cases}$$



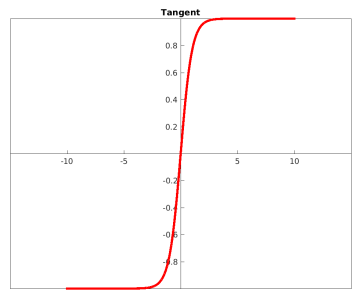
- Sigmoid Activation

$$\sigma(\varphi) = \frac{1}{1 + \exp^{-\varphi}}$$



- Tangent Activation

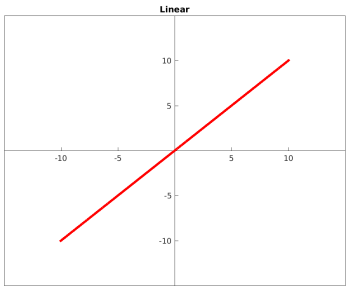
$$\sigma(\varphi) = \frac{\exp^{\varphi} - \exp^{-\varphi}}{\exp^{\varphi} + \exp^{-\varphi}}$$



Activation Functions (σ)

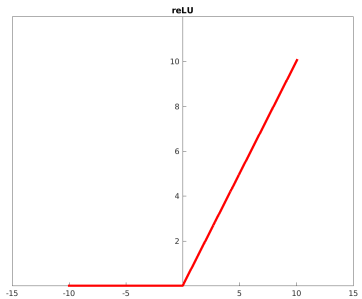
- Linear Activation

$$\sigma(\varphi) = \varphi$$



- Rectified Linear Unit (reLU)

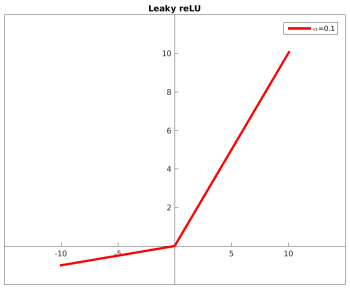
$$\sigma(\varphi) = \max(0, \varphi)$$



- Parametric reLU

$$\sigma(\varphi) = \max(\varphi, \alpha\,\varphi), \;\; \alpha < 1$$

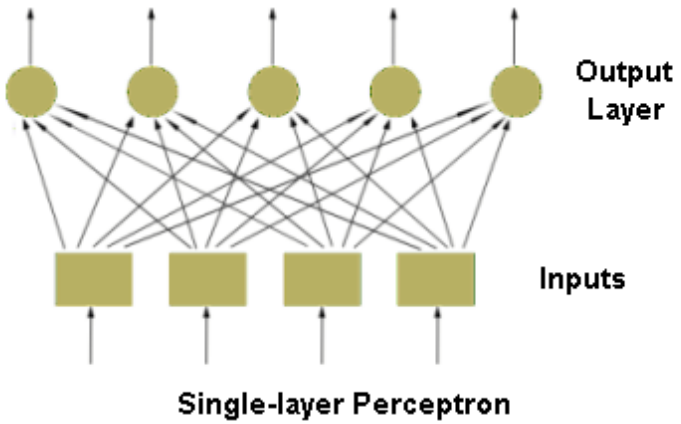
  - Leaky reLU when α is fixed



Outline

Single-layer Perceptron

- Single-layer Perceptron*
  - multi-class classification
  - each output neuron is connected to each input neuron: ***fully-connected***



Single-layer Perceptron

$$y_j = \sigma(\varphi_j) = \left(\sum_{i=1}^{n} w_{ij}\ u_i + b_j \right)$$

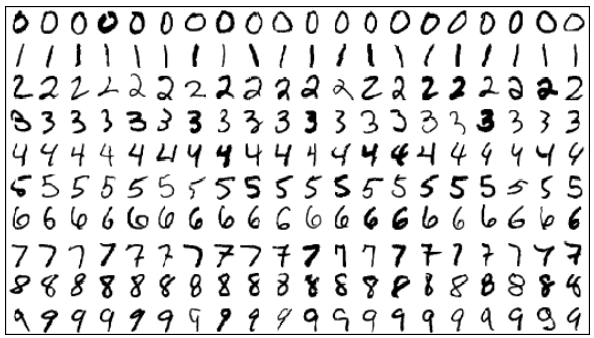
* http://home.agh.edu.pl/~vlsi/AI/intro/

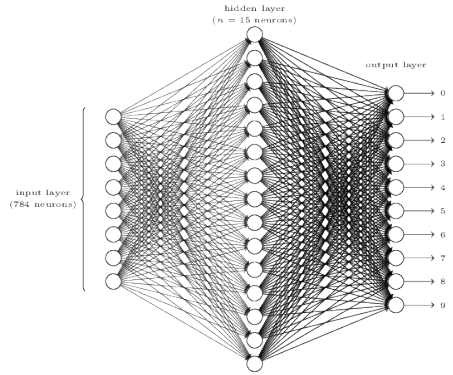
Multi-layer Perceptron

- Multilayer Perceptron
  - Stacked layers one after another
  - One or more hidden layers except input/output layers

$$y_j^{l+1} = \sigma(\varphi_j^l) = \left(\sum_1^n w_{ij}^l \, y_i^l + b_j^{l+1} \right)$$

  - $l > 0$ is the layer index, $y_i^1 = u_i$
- MNIST Digit Classification with 2-layers neural networks

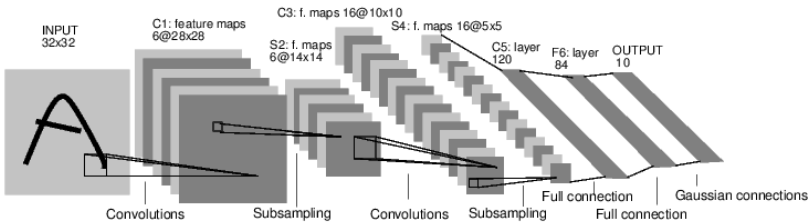




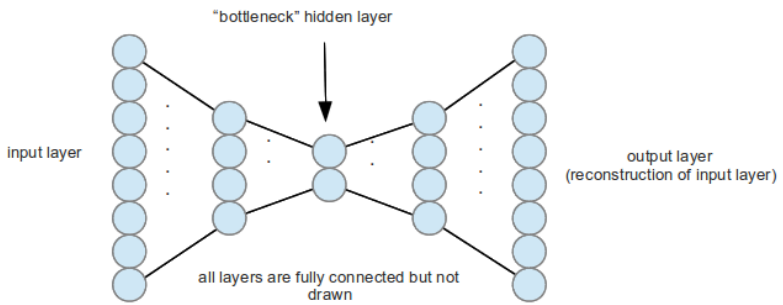
∗ `http://neuralnetworksanddeeplearning.com/`

Applications of Neural Networks

- Classification
  [Lecun et al., 1998]

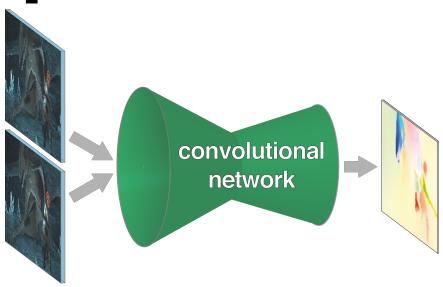


- Auto-encoders

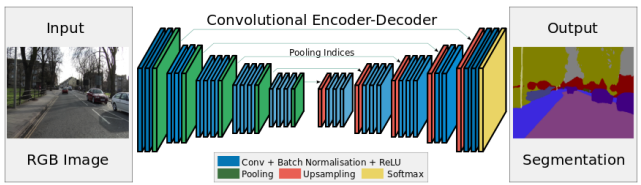


http://nghiaho.com/?p=1765

- Regression
  [Fischer et al., 2015]



- Transfer Learning
  [Kendall et al., 2015]



Outline

Gradient Descent

- Minimize a cost function. Let $\hat{y}(u)$ be the prediction and $y^*(u)$ be the expected output (ground-truth):

$$C(w,b) \equiv \frac{1}{2N}\sum_{j}^{N} ||\hat{y}(u_j) - y^*(u_j)||^2$$

- Highly non-convex respect to the parameter set (w, b). Thus no closed-form solution exist.
- Solution: Gradient Descent
  - Move in the opposite direction of the gradient
  - Let t be the iteration and η be learning rate, update rule for all w and b is then

$$w_{t+1} = w_t - \frac{\eta}{N}\sum_{j}^{N}\frac{\partial C_{U_j}}{\partial w_t}, \quad b_{t+1} = b_t - \frac{\eta}{N}\sum_{j}^{N}\frac{\partial C_{U_j}}{\partial b_t}$$

Backpropagation

- Minimize a cost function. Let $\hat{y}(u)$ be the prediction and $y^*(u)$ be the expected output (ground-truth):

$$C(w,b) \equiv \frac{1}{2N}\sum_{j}^{N} ||\hat{y}(u_j) - y^*(u_j)||^2$$

- Highly non-convex respect to the parameter set (w, b). Thus no closed-form solution exist.
- Backpropagation
  - forward pass all the inputs and compute the loss
  - propagate back the error through the layers
    - Take the derivative of the output of a layer w.r.t.its input and multiply with the error propagated down $\rightarrow$ ***chain-rule***
    - Update the parameters of the layer
  - repeat until convergence, *e.g.*, saturated-loss
- $\rightarrow$ example derivation: Exercise

Stochastic Gradient Descent (SGD)

- Gradient Descent
  - intractable for large datasets
  - high computational expense to compute the cost and derivatives for the entire dataset
  - not easily adaptable to 'online' setting
- Solution: SGD
  - compute the cost and derivatives over a batch of images
  - mini-batch ($m \ll N$) reduces the variance in the parameter update and can lead to more stable convergence

$$w_{t+1} = w_t - \frac{\eta}{m} \sum_j^m \frac{\partial C_{U_j}}{\partial w_t}, \quad b_{t+1} = b_t - \frac{\eta}{m} \sum_j^m \frac{\partial C_{U_j}}{\partial b_t}$$

  - use momentum for faster convergence

$$v_t \rightarrow v_{t+1} = \mu v_t + \frac{\eta}{m} \sum_j^m \frac{\partial C_{U_j}}{\partial w_t}, \quad w_t \rightarrow w_{t+1} = w_t + v_{t+1}$$

Other SGD-based Methods

- Adaptive Delta (AdaDelta)
- Adaptive Gradient (AdaGrad)
- Nesterovs' Accelerated Gradient (NAG)
- RMSProb
- Adaptive Moment Estimation (ADAM)
  - Less sensitive to initial learning rate and momentum

Back to Activation Functions

- Sigmoid Activation

$$\sigma(\varphi) = \frac{1}{1 + \exp^{-\varphi}}$$

  - ✗ Dying gradients (saturated activation)
  - ✗ Non-zero centered

- Tangent Activation

$$\sigma(\varphi) = \frac{\exp^{\varphi} - \exp^{-\varphi}}{\exp^{\varphi} + \exp^{-\varphi}}$$

  - ✗ Dying gradients (saturated activation)
  - ✗ Non-zero centered

- Rectified Linear Unit (reLU)

$$\sigma(\varphi) = \max(0, \varphi)$$

  - ✓ Greatly accelerated convergence
  - ✓ Computationally cheap
  - ✗ Can be fragile during training $\rightarrow$ use Leaky-reLU

Lets play around...

www.cs.stanford.edu/people/karpathy/convnetjs

Quiz

Website

www.onlineted.com

Bibliography I

[Fischer et al., 2015] Fischer, P., Dosovitskiy, A., Ilg, E., Häusser, P., Hazırbaş, C., Golkov, V., van der Smagt, P., Cremers, D., and Brox, T. (2015).
FlowNet: Learning Optical Flow with Convolutional Networks.
In *IEEE International Conference on Computer Vision (ICCV)*.

[Fukushima, 1980] Fukushima, K. (1980).
Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position.
Biological Cybernetics, 36(4):193--202.

[He et al., 2015] He, K., Zhang, X., Ren, S., and Sun, J. (2015).
Deep residual learning for image recognition.
arXiv preprint arXiv:1512.03385.

[Kendall et al., 2015] Kendall, A., Badrinarayanan, V., , and Cipolla, R. (2015).
Bayesian segnet: Model uncertainty in deep convolutional encoder-decoder architectures for scene understanding.
arXiv preprint arXiv:1511.02680.

[Krizhevsky et al., 2012] Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2012).
Imagenet classification with deep convolutional neural networks.
In Pereira, F., Burges, C. J. C., Bottou, L., and Weinberger, K. Q., editors, *Advances in Neural Information Processing Systems 25*, pages 1097--1105. Curran Associates, Inc.

[Lecun et al., 1998] Lecun, Y., Bottou, L., Bengio, Y., and Haffner, P. (1998).
Gradient-based learning applied to document recognition.
Proceedings of the IEEE, 86(11):2278--2324.

Bibliography II

[McCulloch and Pitts, 1943] McCulloch, W. S. and Pitts, W. (1943).
A logical calculus of the ideas immanent in nervous activity.
The bulletin of mathematical biophysics, 5(4):115--133.

[Simonyan and Zisserman, 2015] Simonyan, K. and Zisserman, A. (2015).
Very deep convolutional networks for large-scale image recognition.
ICLR, abs/1409.1556.

[Werbos, 1974] Werbos, P. J. (1974).
Beyond Regression: New Tools for Prediction and Analysis in the Behavioral Sciences.
PhD thesis, Harvard University.

Convolutional Neural Network in A Netshell

Lingni Ma
lingni.ma@tum.de

Technische Universität München
Computer Vision Group

Summer Semester 2016

Unless otherwise stated, this slide takes most references from Standford lecture cs231n: "convolutional neural networks for visual recognition" (`http://cs231n.stanford.edu/`). The mathematical details are referenced from C. M. Bishop, "Pattern recognition and machine learning".

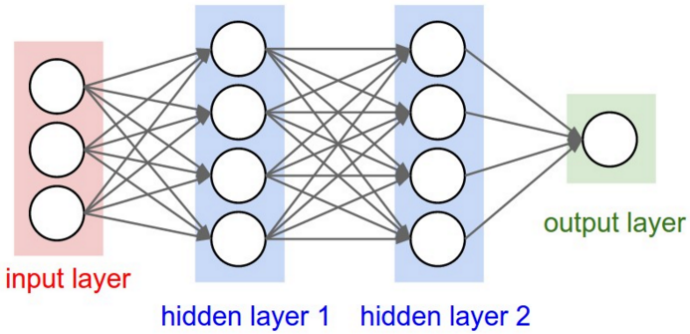
Outline

- Overview of convolutional neural network (CNN)
- Building CNN: basic layers
- Typical CNN architecture
- Training CNN
- Applications of CNN

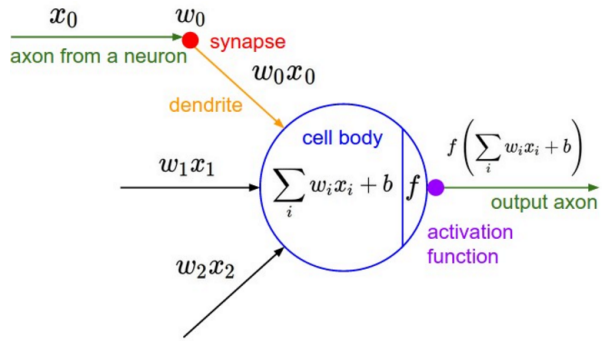
Overview CNN vs. regular neural networks

recall regular neural networks

- stack of layers of neurons
  - neurons within a layer has no particular structure
- fully-connection
  - each neuron is connected to all neurons in previous layer
- estimation capacity: universal approximator
- do not scale well to large input
- easily overfit



regular neural network



single neuron

Overview CNN vs. regular neural networks

convolutional neural networks

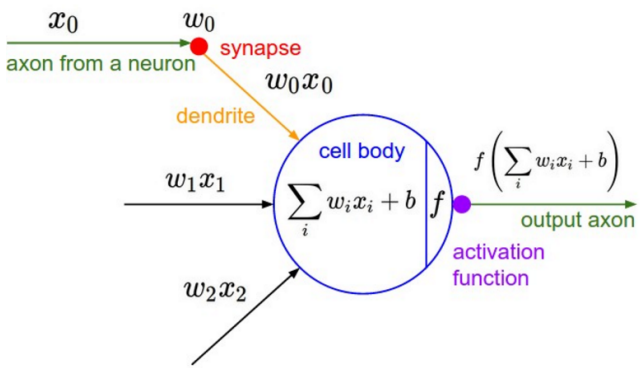
- stack of layers of neurons
- structure neurons into 3D volume
  - a lot of data do have a structure, e.g. images, CT scanes
- introduce convolution as the special type of layer
- also some other new types of layers
- all layers operate on structured neurons and keep neurons structured
- feed-forward architecture: NO LOOPS

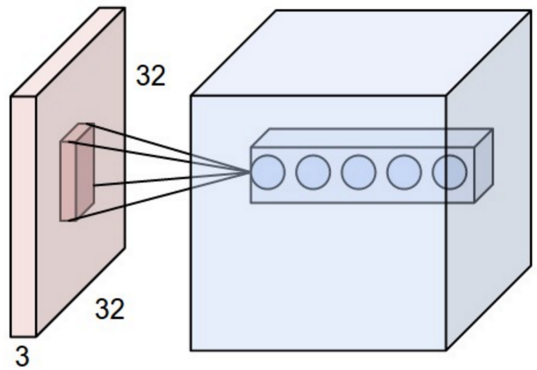
Building CNN Convolution layer

- intuition:
  - structured data have stronger dependency locally than globally
  - if a filter is useful at one location, it is likely to be useful at other locations as well
- connect neurons locally
- share weights
- solution: convolve a small filter over the structured neurons

Building CNN Convolution layer

- a set of learnable filters (or kernels) that performs $\sum w_i x_i + b$
- local fully-connected: each filter is small in width and height, but go through the full depth of input volume
- parameter sharing: convolve filter over input neurons volume
- feature map: 2D output of one filter
- layer configuration
  - receptive field (F): filter spatial dimension
  - depth (D): number of filters
  - padding (P): zero-padding on boundary
  - stride (S): control the overlap of receptive field





input volume: $D_1 \times H_1 \times W_1$
conv layer: receptive field F, depth D, padding P, stride S

- how many parameters does this layer contain?

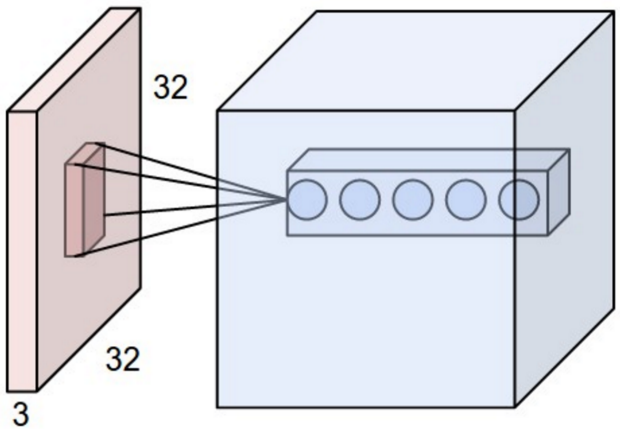
  $$(F \times F \times D_1 + 1) \times D$$

- what is the output volume shape?

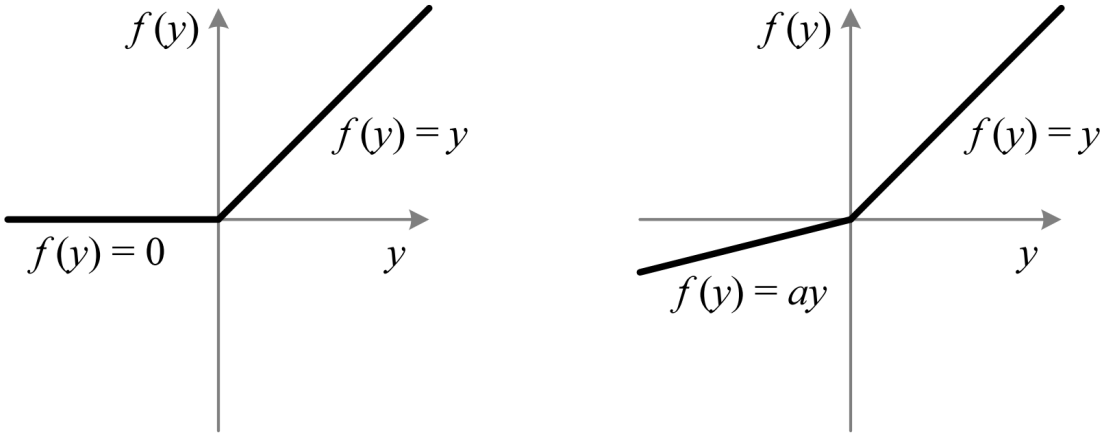
  $$H_2 = (H_1 + 2P - F)/S + 1$$
  $$W_2 = (W_1 + 2P - F)/S + 1$$
  $$D_2 = D$$

- how to preserve spatial resolution of the input?

  $S = 1$ and $F = 2P + 1$



- regular element-wise activation
- the structure of input neuron volume do not change
- no hyper-parameter required
- ReLU[1] = $\max(0, y)$
- PReLU[2] = $\max(0, y) + \alpha \min(0, y)$



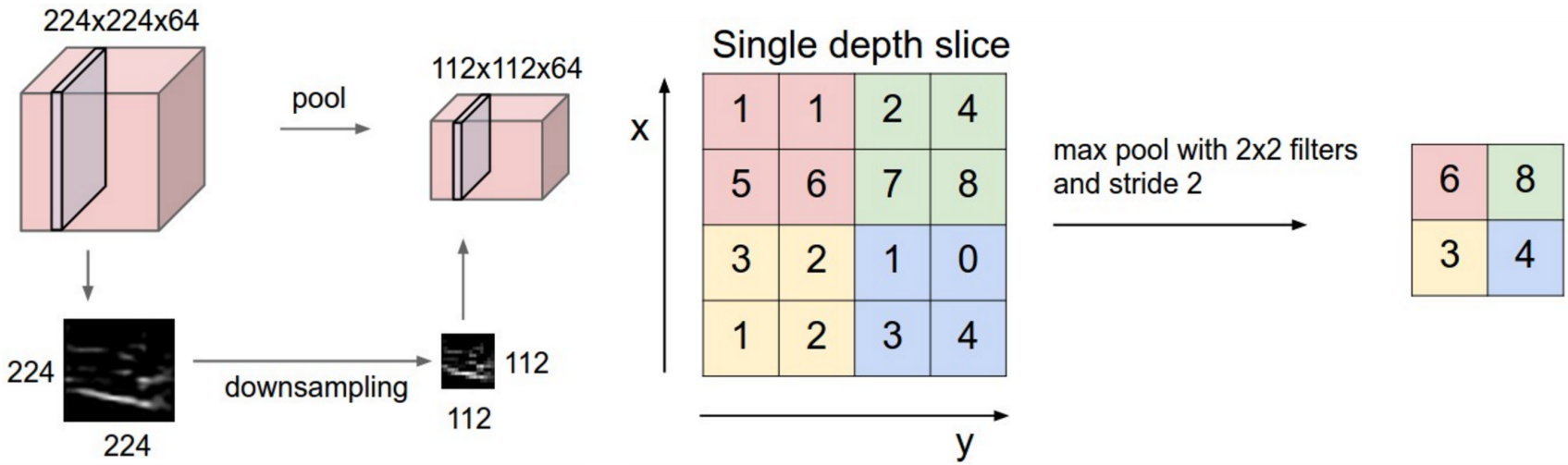
[1] Alex Krizhevsky, Ilya Sutskever and Geoff Hinton, "ImageNet classification with deep convolutional neural networks", NIPS 2012
[2] He et al. ICCV 2015, "Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification"

Building CNN Pooling layer

- extract salient information, such that parameters are drastically reduced and over-fitting can be controlled
- convolve pooling window over each feature map
- MAX pooling vs. Average pooling
- common pooling layer specification

  $F = 2, S = 2, P = 0$, non-overlapping, most common

  $F = 3, S = 2, P = 0$, overlap pooling



Building CNN Fully Connected layer

- contain most of network parameters
- combine features to compute classification score $\Rightarrow$ act as SVM classifier
- net surgery: convert FC layer to Conv layer

  FC layer: $K = 4096$ operate on $7 \times 7 \times 512$ volume of neuron

  FC: 4096 inner product $\mathbf{w}^{\mathsf{T}}\mathbf{x}$, where $\mathbf{w}, \mathbf{x} \in \mathbb{R}^{7\times7\times512}$

  Conv: $D = 4096, F = 7, P = 0, S = 1$

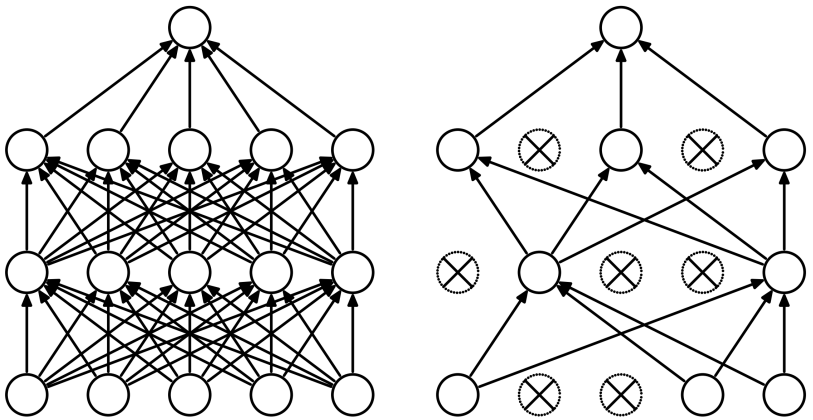
- practical advantages with Conv layer: efficiency

  input 224×224 image, downsample 32 times before FC.

  What if input 384×384?

  evaluating 224×224 crops of the 384×384 image in strides of 32 pixels is identical to forwarding the converted ConvNet one time

- batch normalization[1]
  - accelerate training and increase accuracy
  - append after activation
  - $y_i = \gamma \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}} + \beta$, with $\mu_{\mathcal{B}} = \frac{1}{m}\sum_i^m x_i$, $\sigma_{\mathcal{B}}^2 = \frac{1}{m}\sum_i^m (x_i - \mu_{\mathcal{B}})^2$
- dropout[2]
  - randomly mask out neuros
  - control over-fitting
  - mostly append to FC layers



[1] Ioffe et al., "Batch normalization: accelerating deep network training by reducing internal covariate shift", NIPS 2015
[2] Srivastava et al., "Dropout: a simple way to prevent neural networks from overfitting", JMLR 2014

Loss layer is only used during training. The loss of CNN defines the optimization problem for parameter learning.

- regression with Euclidean loss

$$E(\mathbf{w}) = \frac{1}{2N}\sum_i \|f(\mathbf{w}, x_i) - y_i\|^2$$

- classification with hinge loss

$$E(\mathbf{w}) = \frac{1}{N}\sum_{\mathbf{x}_n}\sum_{j\neq k} \max\left(0, y_j(\mathbf{w}, \mathbf{x}_n) - y_k(\mathbf{w}, \mathbf{x}_n) + \Delta\right)$$

- classification with cross-entropy loss

$$H(p, q) = \sum_x -p(x)\log q(x)$$

minimize KL-divergence between true distribution $p(x)$ and estimated distribution $q(x)$

$E(\mathbf{w}) = E_{\text{data}}(\mathbf{w}) + \lambda E_{\text{regu}}(\mathbf{w})$

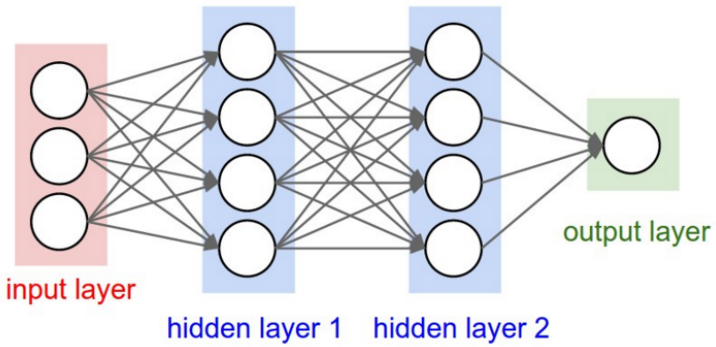
- regularizer is also known as weight decay
- control over-fitting by penalize parameter values
- $L_1 := \|\mathbf{w}\|_1$, not strictly differetiable
- $L_2 := \|\mathbf{w}\|_2^2$, differentiable, zero-mean Gaussian prior distribution

Building CNN summary

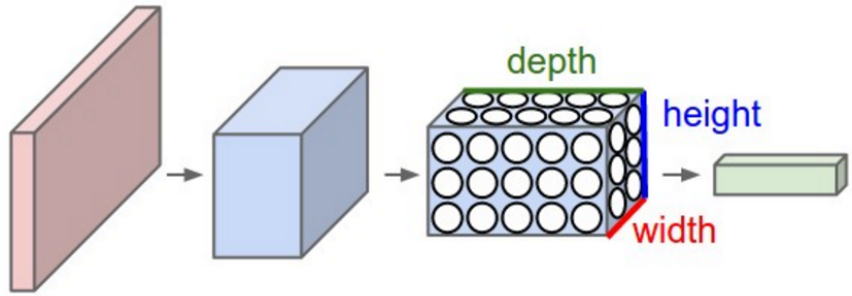
- inferene: feed-forward
- parameter training: error backpropagation

$$y_k^{l+1}(\mathbf{w},\mathbf{x}) = f\left(\sum_{j=1}^{M} w_{kj}^{l+1} g\left(\sum_{i=1}^{D} w_{ji}^{l} x_i + b_j^l\right) + b_k^{l+1}\right)$$

- neurons are structured into 3D volume, layers operates on structured neurons and keep the neurons structured
- convolution layer: neurons are local fully-connected, small filters and parameter sharing

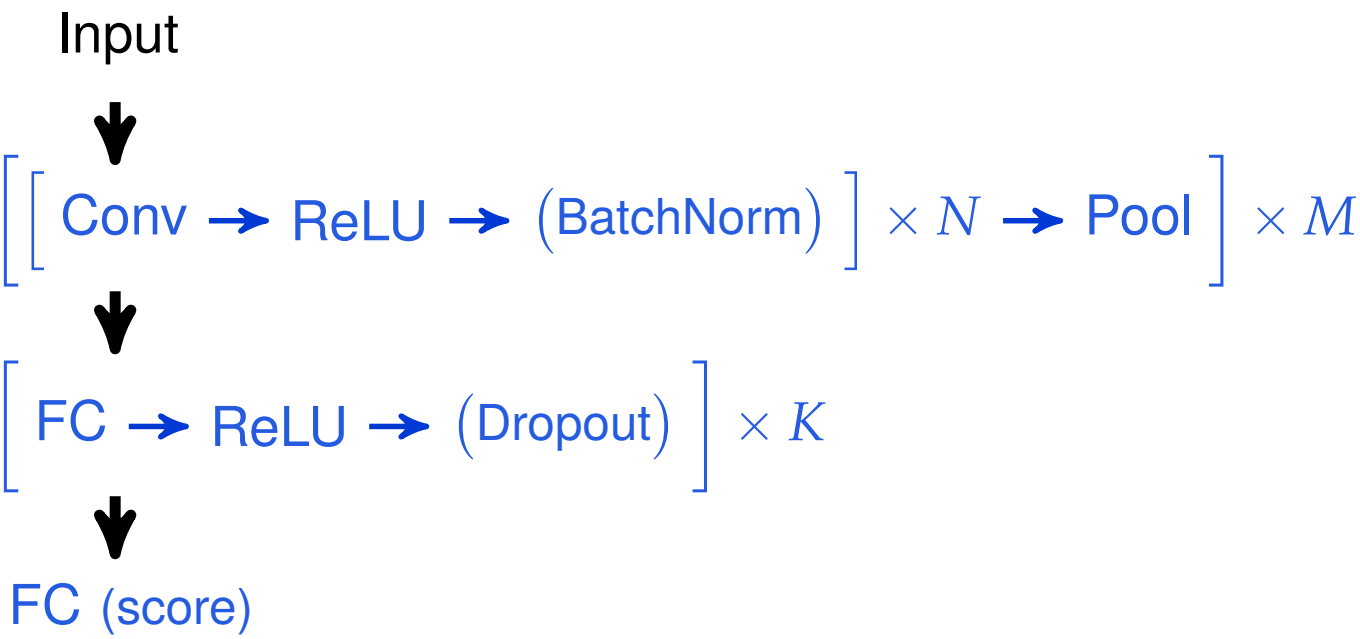


regular neural network



convolutional neural network

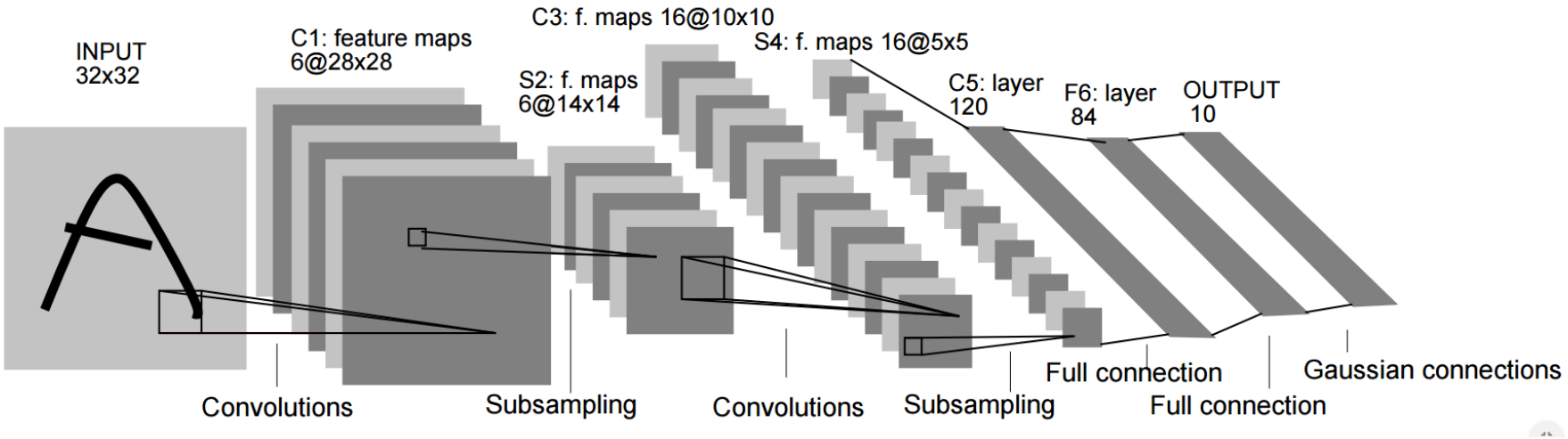
Building CNN patterns of layers and filters



- usually $N \in (0, 3), M \geq 0$
- shape-preserve conv $F = 2P + 1, S = 1$, downsample with pooling
- better stack small conv filter to obtain large receptive field
  - stack three $F = 3$ filters vs. one $F = 7$ filter
- $K \in (0, 3)$

Classical CNN Architectures LeNet[1]

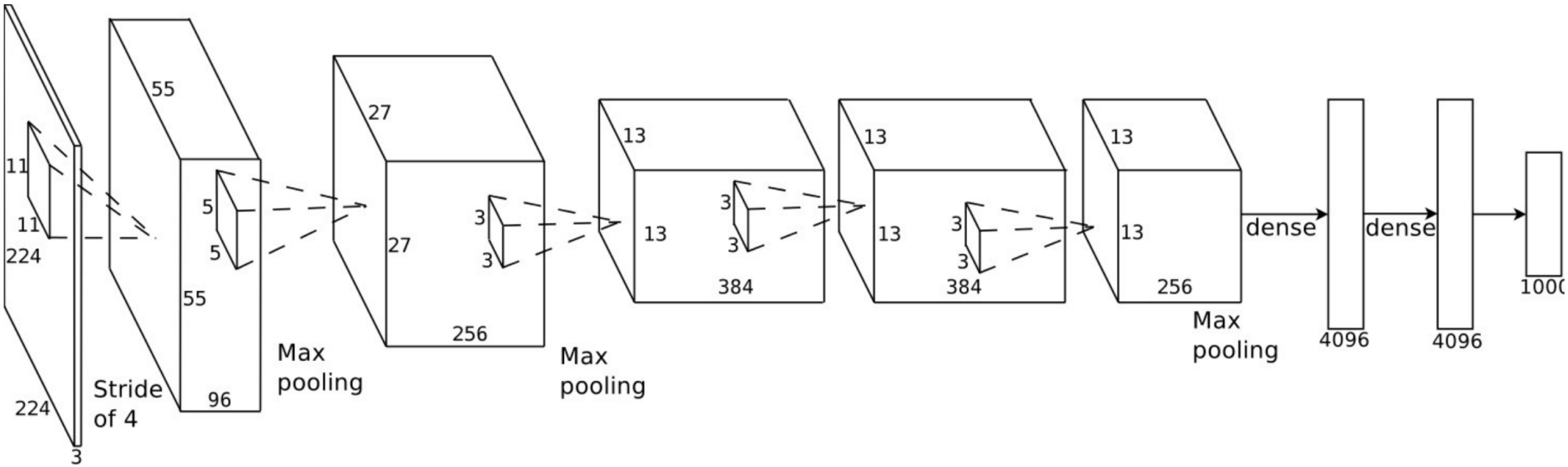
- first successful CNN example
- digit and letters recognition
- 3 Conv, 1 FC, 60K parameters



[1] Yann LeCun et al., "Gradient-based learning applied to document recognition", IEEE proceedings, 1998

Classical CNN Architectures AlexNet[1]

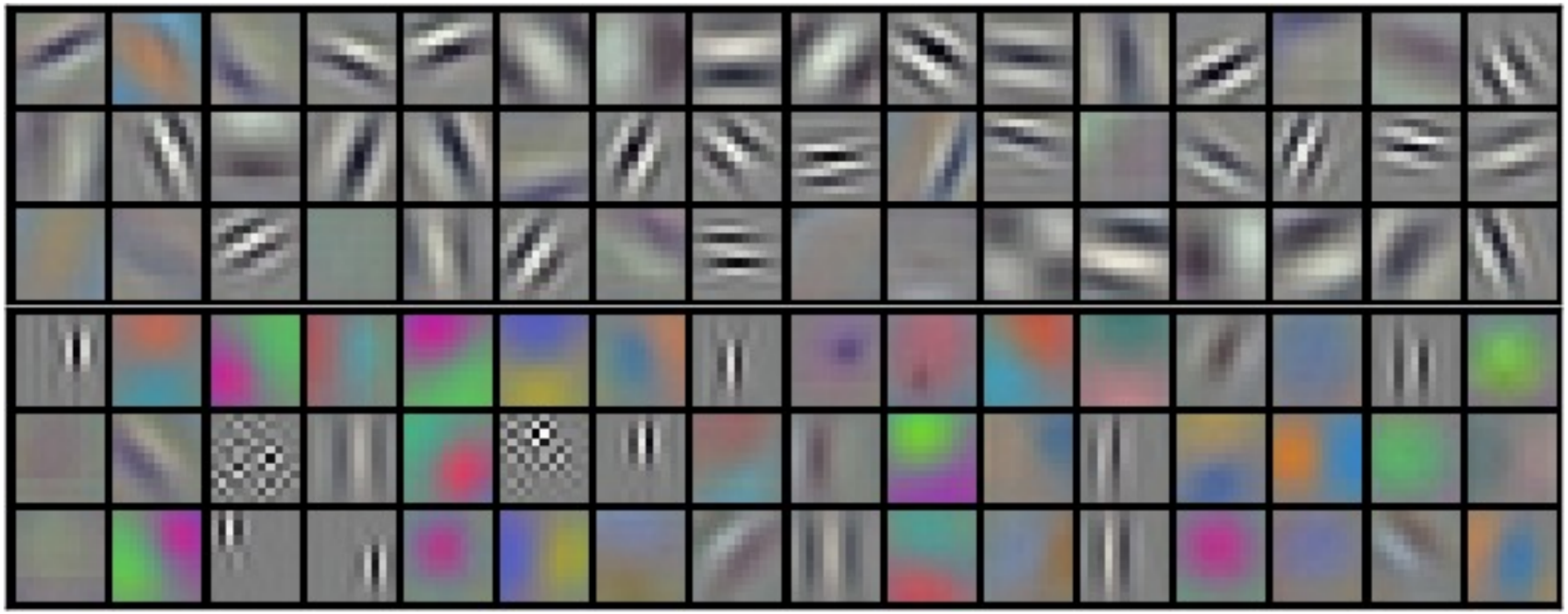
- first work to popularize CNN
- 5 Conv layers, 3 FC layers, 650K neurons, 60M parameters



[1] Alex Krizhevsky, Ilya Sutskever and Geoff Hinton, "ImageNet classification with deep convolutional neural networks", NIPS 2012

Classical CNN Architectures AlexNet[1]

- first work to popularize CNN
- 5 Conv layers, 3 FC layers, 650K neurons, 60M parameters
- first 96 filters learned



[1] Alex Krizhevsky, Ilya Sutskever and Geoff Hinton, "ImageNet classification with deep convolutional neural networks", NIPS 2012

Classical CNN Architectures VGGNet[1]

- deep: 16 or 19 layers
- 13 or 16 Conv layers, small filters $F = 3, S = 1, P = 1$
- 5 max pooling layer, downsample by 32
- 240K neurons, 138M parameters

[1] Karen Simonyan and Andrew Zisserman, 2014 ICLR, "Very deep convolutional networks for large-scale image recognition"

Classical CNN Architectures

GoogLeNet[1]

- 22 layers
- inception module: combine convolutional layers

ResNet[2]

- very deep, e.g. 152 layers

Do networks get better simply by stacking more layers?

[1] Christian Szegedy et al, 2015 CVPR, "Going deeper with convolutions"
[2] He et al, 2015, "Deep Residual Learning for Image Recognition"

Training CNN

data preparation

- mean substraction, normalization
- augmentation

weight initialization

- Gaussain, xavier[1], msra[2]
- trainsfer learning

optimization algorithm

- SGD
- shuffle input

hyper-parameter

- learning rate policy
- batch size

[1] Xavier Glorot et al. "Understanding the difficulty of training deep feedforward neural networks", 2010

[2] He et al., "Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification", ICCV 2015

Training CNN Framework and Packages

Caffe[1]

- C++/CUDA, with Python and Matlab bindings
- (+) good for feedforward network (+) finetune (+) easy to start
- (-) bad at recurrent network (-) poor support for visualization

Torch[2]

- C and Lua
- used a lot by Facebook, DeepMind

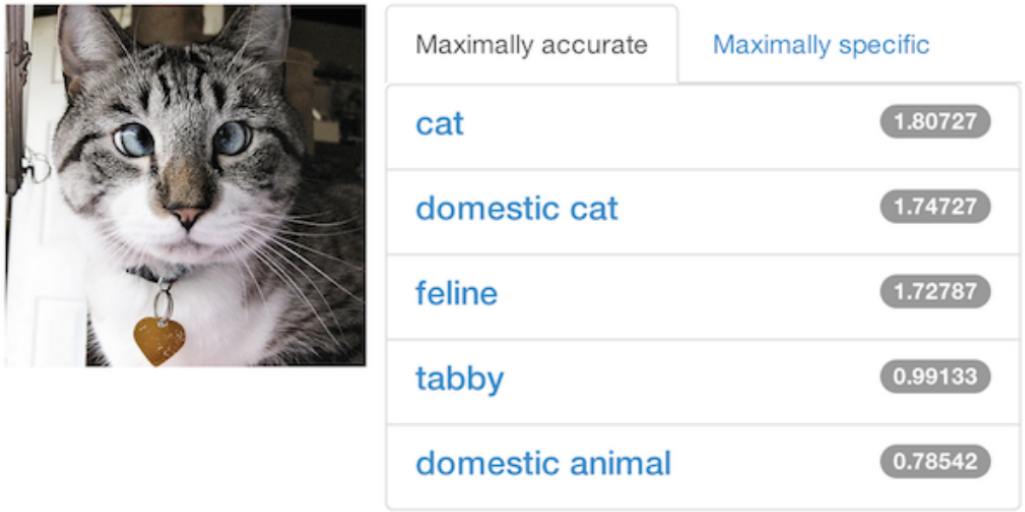
TensorFlow[3]

- Google
- Python, source code not readable
- not limited for CNN, also good at RNN

[1] http://caffe.berkeleyvision.org/
[2] https://torch.ch
[3] https://www.tensorflow.org/

Applications What CNN is good at

- classification and recognition



Applications What CNN is good at

- classification and recognition
- semantic segmentation

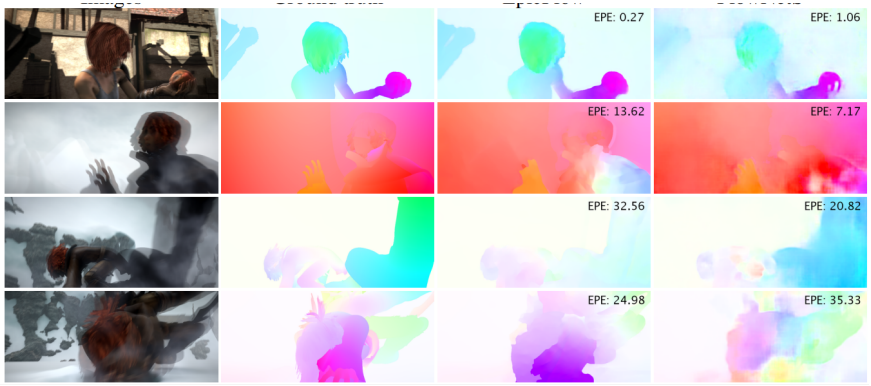
Applications What CNN is good at

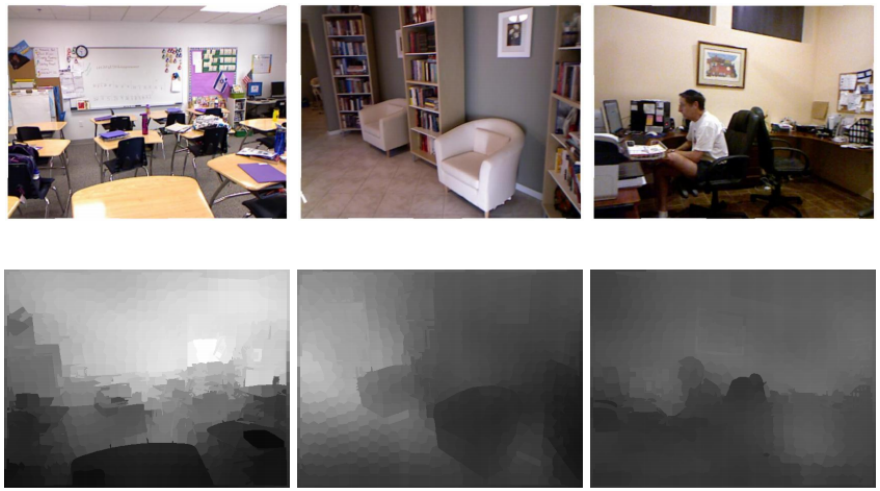
- classification and recognition
- semantic segmentation
- feature extraction, encoding



Applications What CNN is good at

- classification and recognition
- semantic segmentation
- feature extraction, encoding
- regression, estimation, reconstruction





Applications What CNN is good at

- classification and recognition
- semantic segmentation
- feature extraction, encoding
- regression, estimation, reconstruction

Applications What CNN is good at

- classification and recognition
- semantic segmentation
- feature extraction, encoding
- regression, estimation, reconstruction
- unsupervised problem, reinforcement learning

Question ?