Deep Learning

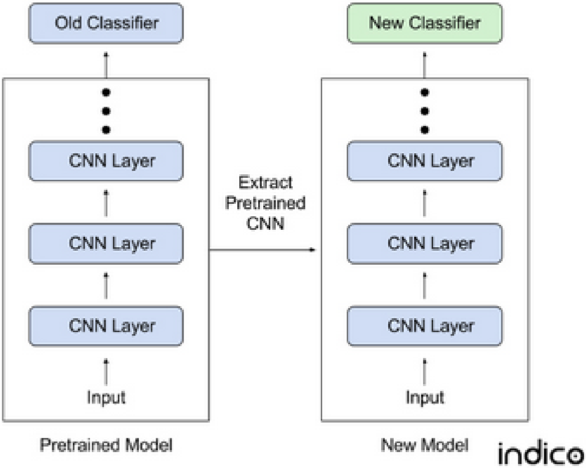
9. Practical aspects of deep learning. Transfer learning. One shot learning. Hardware. Frameworks.

Viacheslav Dudar

Taras Shevchenko National University of Kyiv

2018

Transfer learning



Transfer learning assumptions

We assume that:

- Features extracted at early layers are general for all visual data
- Convolutional network forms a hierarchy of features: features in higher levels become more and more abstract
- To recognize different types of objects we need to change only the last layer of the network

Tranfer learning use

- Small dataset, similar to original $->$ retrain last layer
- Large dataset, similar $->$ use original weights to initialize, finetune over the full net
- Small dataset, different $->$ train linear layer but earlier in the network
- Large dataset, different $->$ train from scratch (but weights initialization is still beneficial)
- If the input size is different, convolutional weights still can be used from another network
- Use smaller learning rates for finetuning

Example: cats/dogs classification

Small dataset: 1000 cats and 1000 dogs.

- Training from scratch: about 75-80% accuracy
- Finetuting fully connected layers of VGG: 90% accuracy
- Finetuting last convolutional and fully connected layers of VGG: 95%

NN architecture

Before NNs:

- Too small model $->$ underfitting
- Too big model $->$ overfitting
- Need to find optimal middle sized model and apply good regularization

NN architecture

- Divide into disjoint training, validation and test sets.
- Train on training set, check performance on validation set, modify model, repeat
- Use test set to estimate final performace

How to modify model?

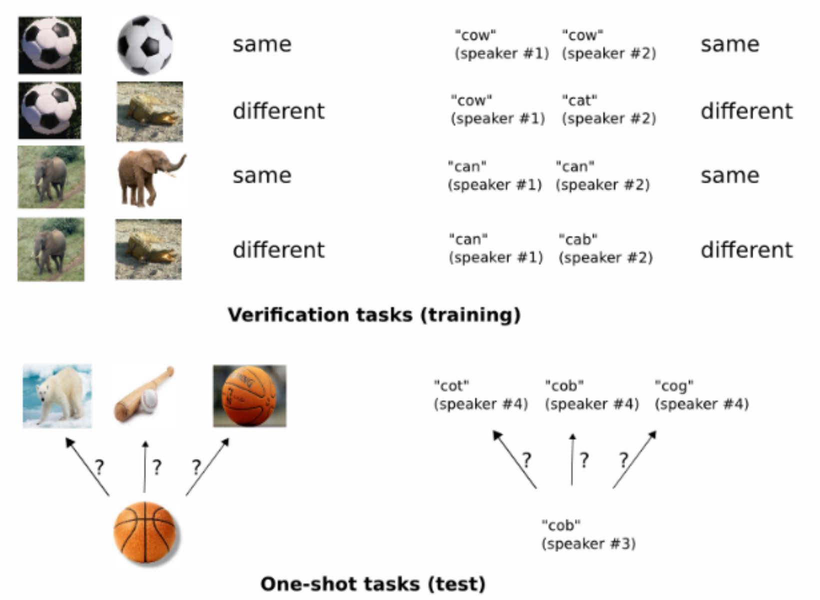
- High train error $->$ underfitting $->$ increase model size (increase layer size, add more layers)
- Low train error, high validation error $->$ overfitting $->$ increase regularization (dropout rate), use more data, use heavier data augmentation
- Low train error, low validation error $->$ done

One-shot learning

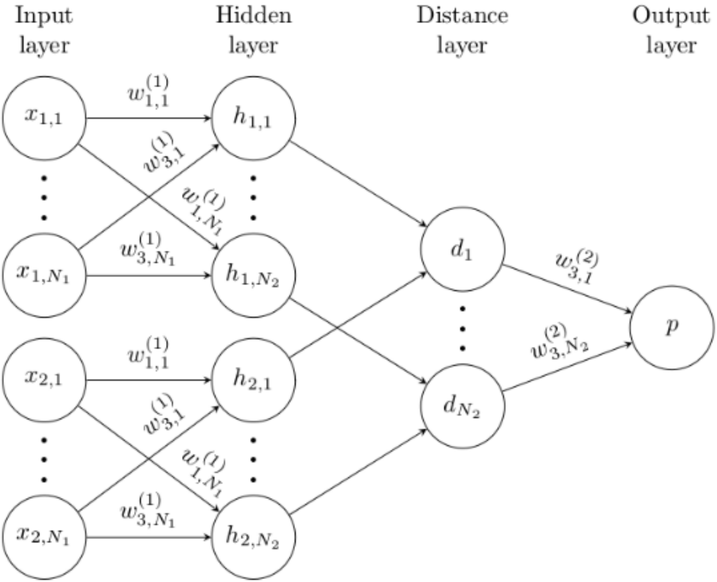
Motivation:

- Deep learning works for large datasets. What to do with small datasets?
- Can we classify objects if we have only one example for each class?
- Application: face recognition

Siamese network



Siamese network

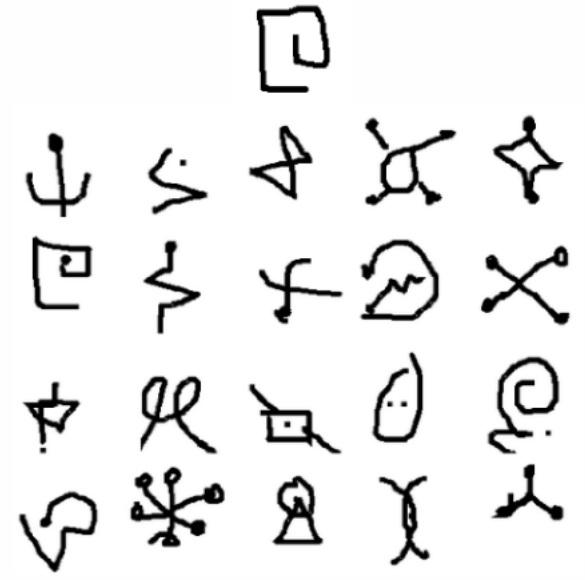


Siamese network

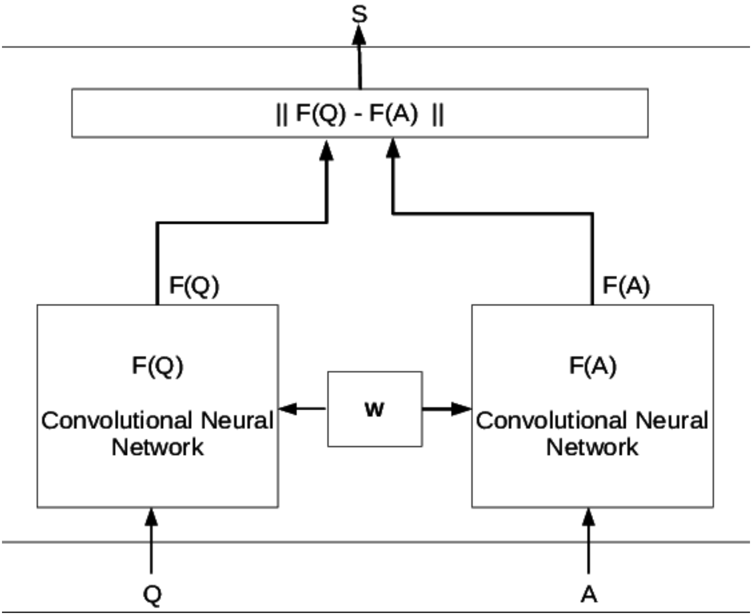
- Generate feature vector for input image with convolutional network
- Maximize distance between feature vectors for different classes and minimize for the same classes
- If there is only one example for each class, minimize distance betwwen images and their affine transformations, subimages, and so on

Example

Example dataset: Omniglot (alphabets over the world)



Siamese network



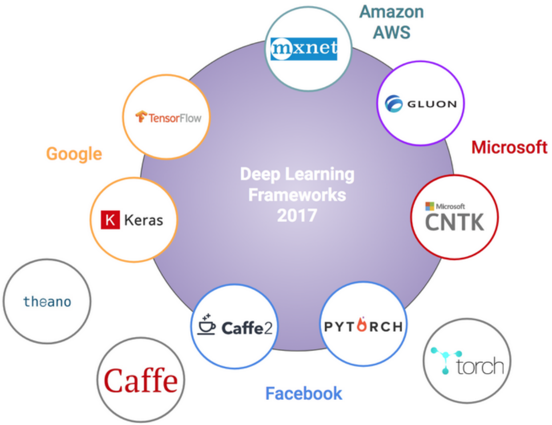
Omniglot results

Method	Test
Humans	95.5
Hierarchical Bayesian Program Learning	95.2
Affine model	81.8
Hierarchical Deep	65.2
Deep Boltzmann Machine	62.0
Simple Stroke	35.2
1-Nearest Neighbor	21.7
Siamese Neural Net	58.3
Convolutional Siamese Net	92.0

Deep learning frameworks

- Building blocks for designing and training neural nets
- High level programming interface
- Choose framework based on: ease of programming, running speed, support of GPUs
- Main function of most frameworks: automatic differentiation of the model

Deep learning frameworks



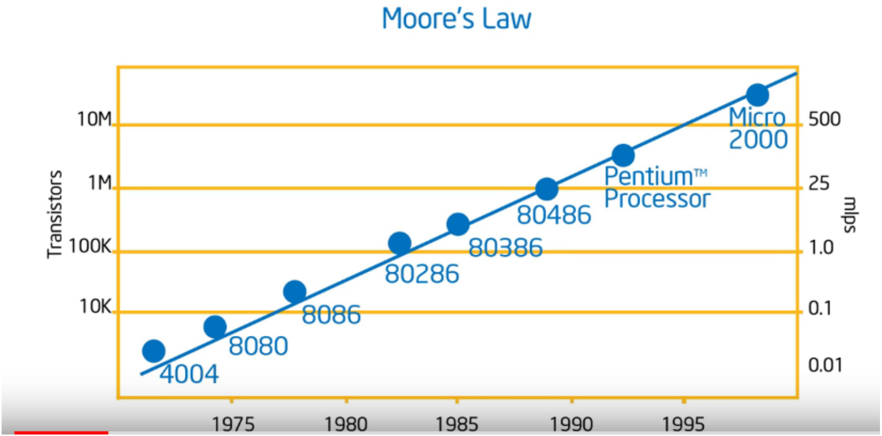
Many more...

Some of deep learning frameworks

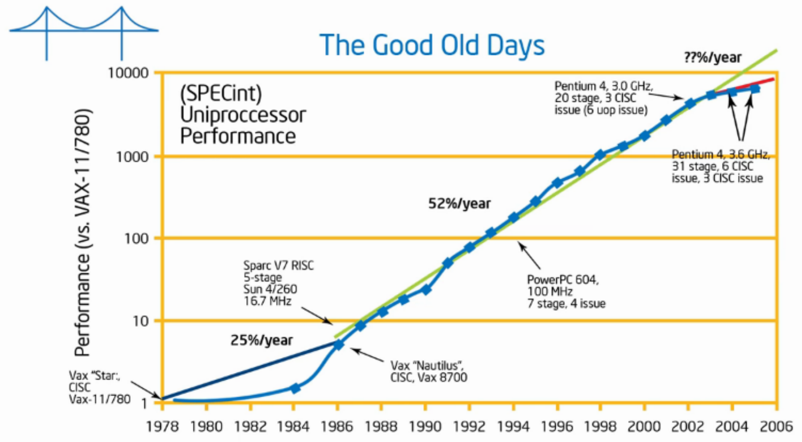
Feel free to explore wide variety of deep learning frameworks!
Keras (python): simple to use. Contains most popular layers, optimizers, pretrained networks. Good for exploiting already existing architectures.
PyTorch (python): (Python, former Torch, Lua language): able to create dynamical computational graphs. Good for trying own layers.
Theano: (Keras with Theano backend is faster on single GPU compared with Tensorflow backend)
Caffe: (C++, one of the fastest convnets implementations)
MXNET (Apache): supported by AWS, Azure
CNTK (Microsoft Cognitive Toolkit): optimized for Azure, integrated with Visual Studio.

Hardware for deep learning

Why parallel computing?

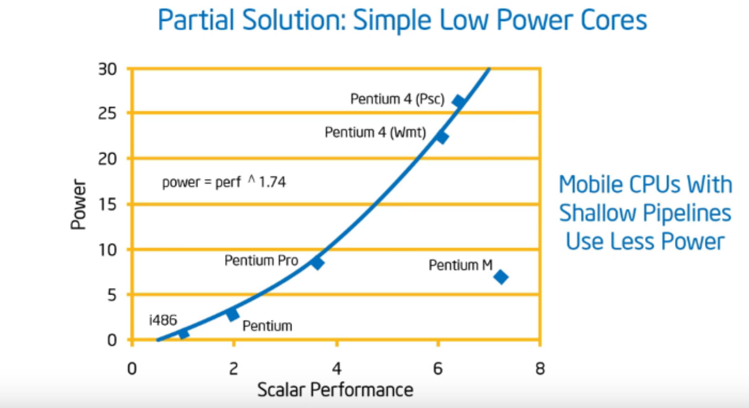


Good old days



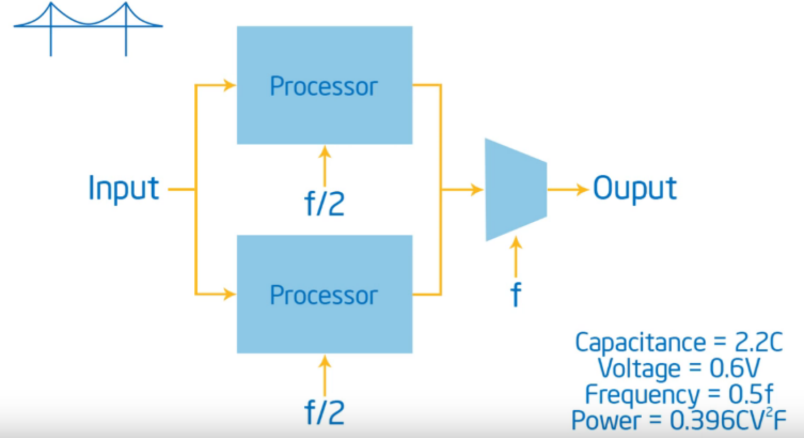
From Hennessy and Patterson, *Computer Architecture: A Quantitative Approach*, 4th edition, Sept. 15, 2006

Power wall


Partial Solution: Simple Low Power Cores
30
25
20
15
10
5
0
Power
0
2
4
6
8
Scalar Performance
power = perf ^ 1.74
Pentium 4 (Psc)
Pentium 4 (Wmt)
Pentium Pro
Pentium M
i486
Pentium
Mobile CPUs With Shallow Pipelines Use Less Power

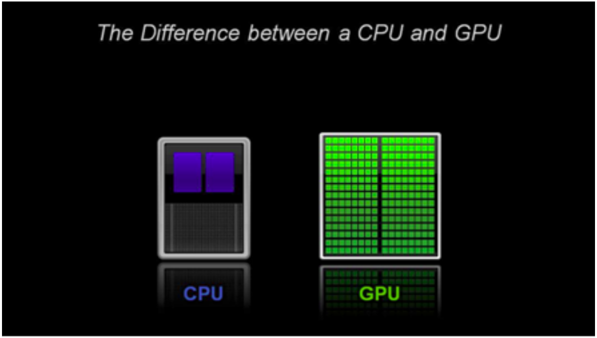
Power wall solution

Parallel architecture solves power problem at expense of need to parallelize computation.



Hardware for deep learning

CPU: efficient for few very complex instructions
GPU: efficient for many simple instructions

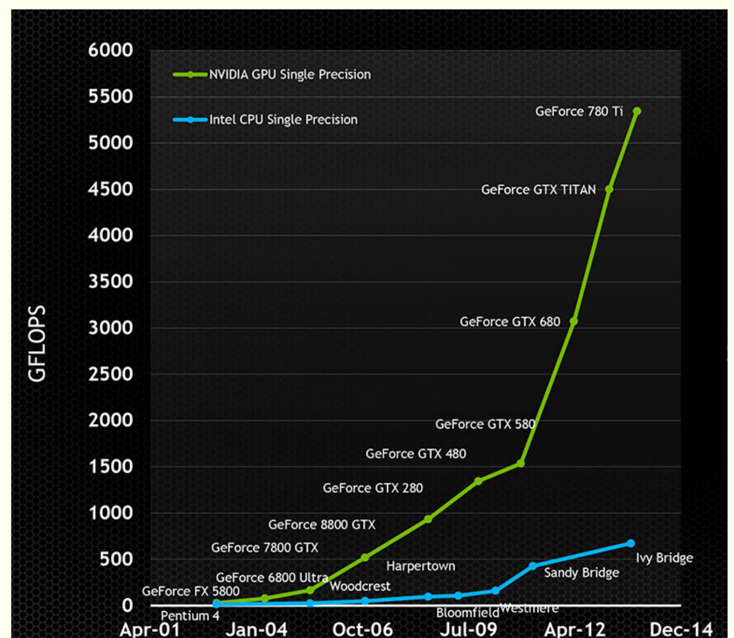


GPUs were used for rendering images on screen (performing in parallel many simple operations, like matrix multiplications) - this is exactly what we need for training neural net!

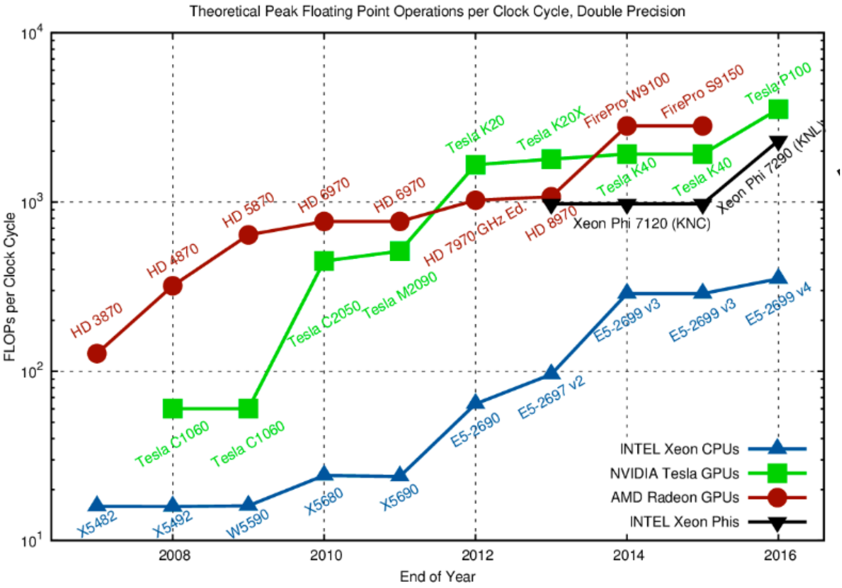
GPGPU

GPGPU: General-purpose computing on graphics processing units
OpenCL (Open Computing Language): open source GPGPU framework
CUDA: proprietary (NVIDIA) GPGPU framework (works faster)
Both are C-like.

GPU-CPU comparison



GPU - CPU servers comparison



Modern GPU

Modern GPU

2017: TESLA VOLTA V100

21B transistors
815 mm^2

80 SM
5120 CUDA Cores
640 Tensor Cores

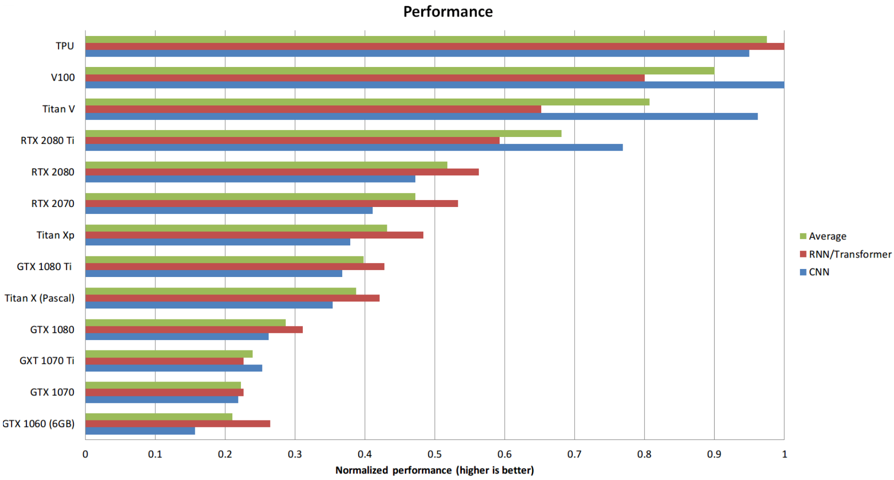
16 GB HBM2
900 GB/s HBM2
300 GB/s NVLink

*full GV100 chip contains 84 SMs

Modern GPU

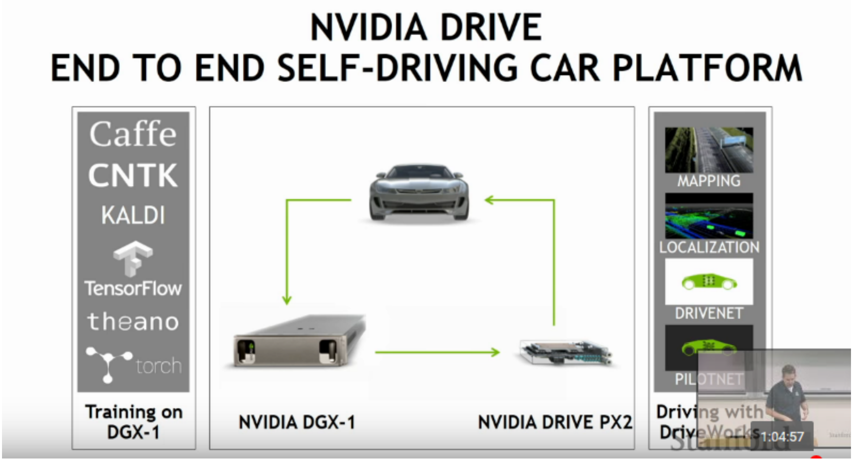
	Tesla V100 for NVLink	Tesla V100 for PCIe
PERFORMANCE with NVIDIA GPU Boost™	DOUBLE-PRECISION 7.8 teraFLOPS	DOUBLE-PRECISION 7 teraFLOPS
	SINGLE-PRECISION 15.7 teraFLOPS	SINGLE-PRECISION 14 teraFLOPS
	DEEP LEARNING 125 teraFLOPS	DEEP LEARNING 112 teraFLOPS
INTERCONNECT BANDWIDTH Bi-Directional	NVLINK 300 GB/s	PCIE 32 GB/s
MEMORY CoWoS Stacked HBM2	CAPACITY 32/16 GB HBM2	

GPU comparison



Deep learning in a box

GPU for Self driving

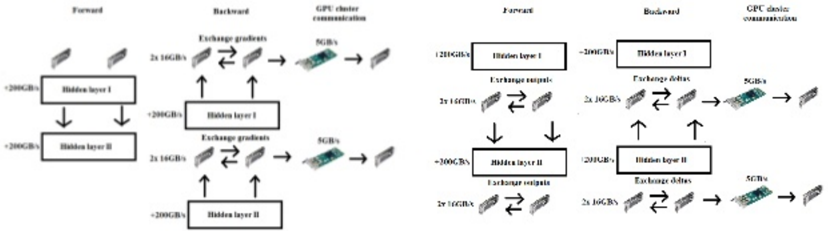


Data and model parallelism

Parallelism

DEVIEW
2015

Data Parallelization | Model Parallelization



The good : Easy to implement
The bad : Cost of sync increases with the number of GPU

The good : Larger network can be trained
The bad : Sync is necessary in all layers