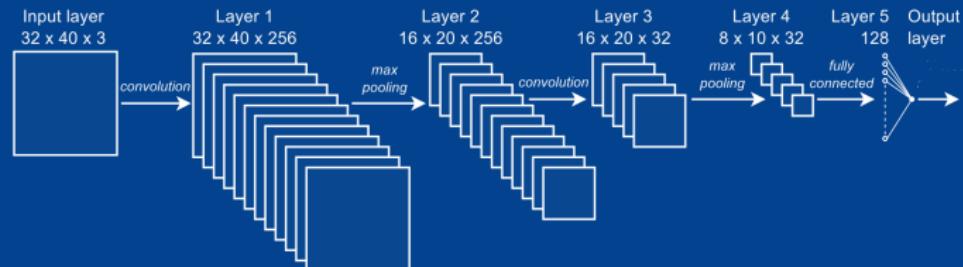




LESSON 09: Deep Learning + CNN's + Hardware

CARSTEN EIE FRIGAARD

AUTUMN 2020



"A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P , if its performance at tasks in T , as measured by P , improves with experience E." — Mitchell (1997).

L09: DL + CNN's + HW, Agenda

- ▶ Feedback på O2, Peter.
-
- ▶ Convolutional Neural Networks (CNN's),
 - ▶ and Deep-learning (DL).
- ▶ Hardware for Machine Learning,
 - ▶ (CPUs), GPUs, TPUs, Exotic Hardware.

Deep-learning (DL)

Artificial Intelligence:

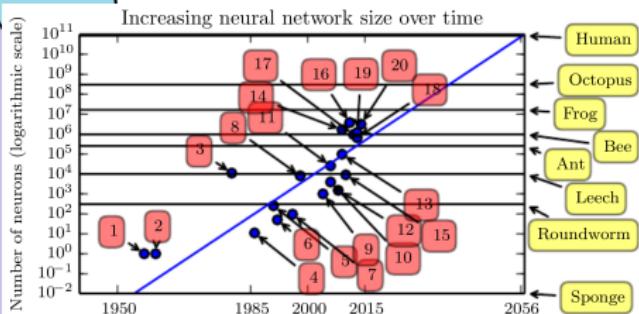
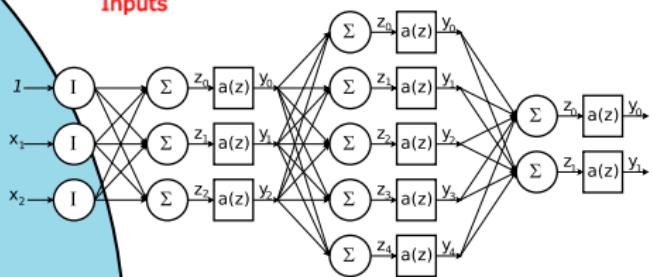
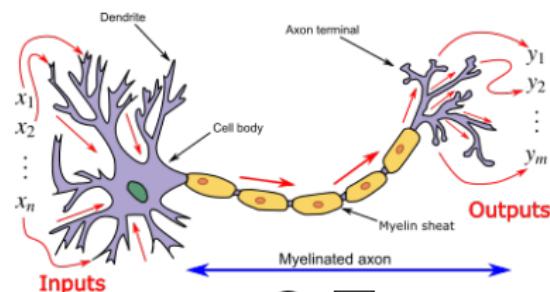
Mimicking the intelligence or behavioural pattern of humans or any other living entity.

Machine Learning:

A technique by which a computer can "learn" from data, without using a complex set of different rules. This approach is mainly based on training a model from datasets.

Deep Learning:

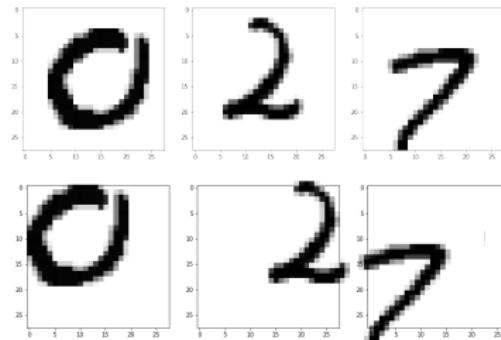
A technique to perform machine learning inspired by our brain's own network of neurons.



Definition of DL=??

Preprocessing/Feature-extraction + Machine Learning

Can your ML model handle simple image translation (rotation and scaling)?



..no problem for Your visual cortex, right?

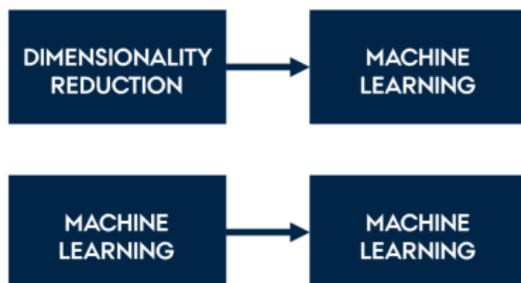
Introducing the Convolutional Neural Networks: trade off

- ▶ preprocessing/feature-extraction + classification with:
- ▶ feature learning (CNN kernels) + classification (fully connected NN)

Convolutional Neural Networks

Feature Extraction vs. Feature Learning

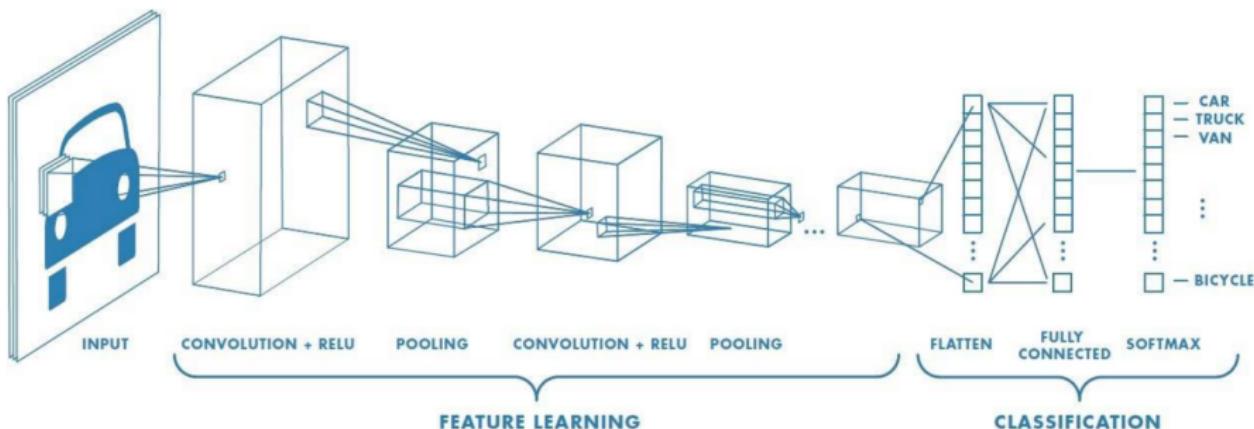
- ▶ Smart filtering:
the distinction between dimensionality reduction and machine learning blurs..



- ▶ Fundamental problem of filtering:
What is noise, what is signal?

Convolutional Neural Networks

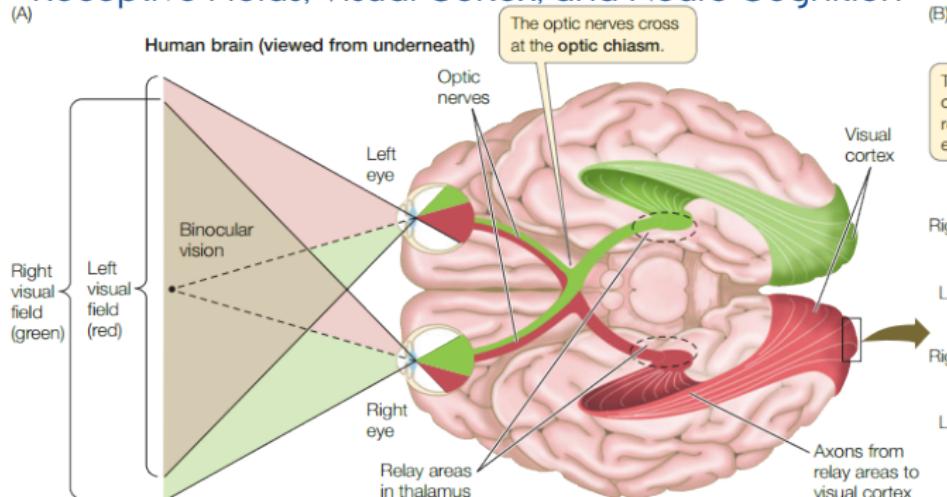
Feature Extraction vs. Feature Learning



Convolutional Neural Networks

Human Eye and Brain Image Processing: Receptive Fields, Visual Cortex, and Neuro Cognition

(A)



(B)

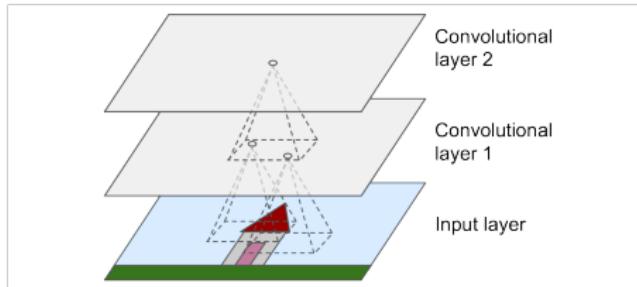
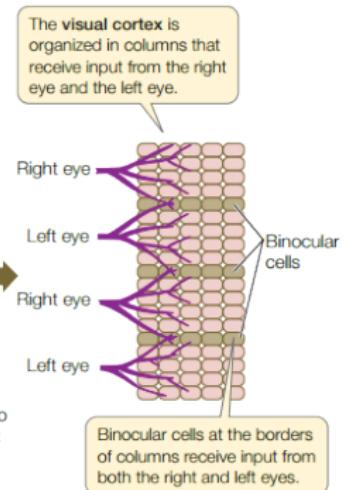


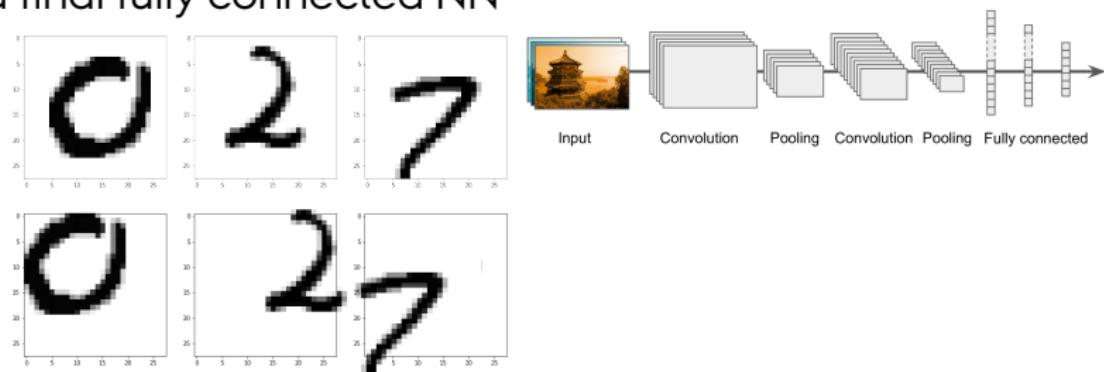
Figure 14-2. CNN layers with rectangular local receptive fields

Feature Learning + Machine Learning

CNN principle

Translation, rotation, and scaling invariant

- ▶ automatic image **feature extraction** that is **feature learning** via
 - ▶ 'convolutional', 'pooling' CNN layers, then a final fully connected NN



..no problem for Your CNN ML model, right?

Feature Learning + Machine Learning

CNN principle: Convolutional Layer (via CNN kernels)

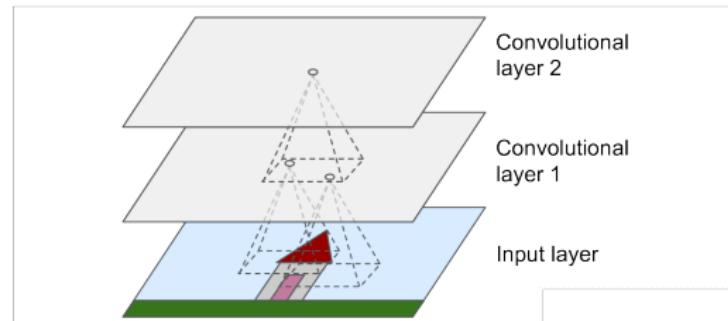


Figure 14-2. CNN layers with rectangular local receptive fields

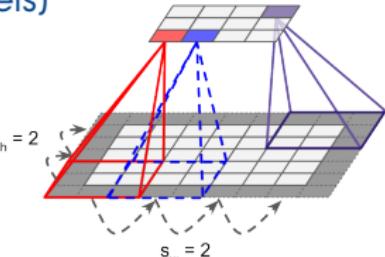
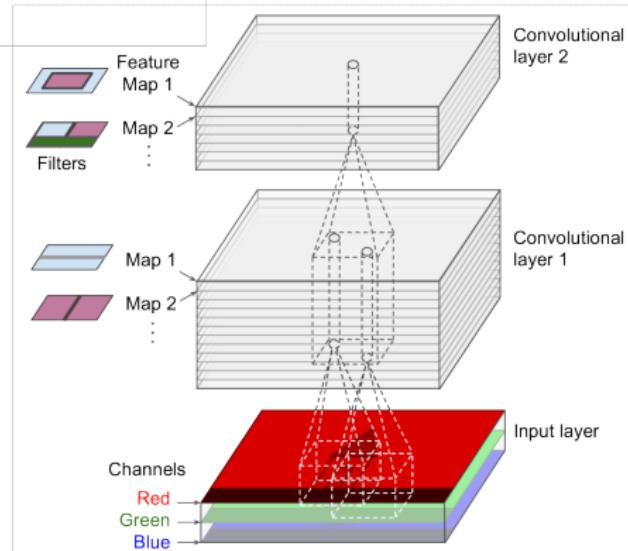


Figure 14-4. Reducing dimensionality using a stride of 2



Feature map 1 => with kernel 1

Feature map 2 => with kernel 2

Figure 14-6. Convolution layers with multiple feature maps, and images with three color channels

CNN Kernels

2D Convolution with a Kernel: Principle

0	0	0	0	0	0	0	0
0	60	113	56	139	85	0	0
0	73	121	54	84	128	0	0
0	131	99	70	129	127	0	0
0	80	57	115	69	134	0	0
0	104	126	123	95	130	0	0
0	0	0	0	0	0	0	0

Kernel

0	-1	0
-1	5	-1
0	-1	0

114				

$$5*60 + (-1)*113 + (-1)*73 = 114$$

CNN Kernels

2D Convolution with a Kernel: Different Kernels

-0.2	0.0	0.5
1.0	0.3	-0.6
0.0	0.0	0.8

0.0	0.0	0.0
0.8	-0.5	0.8
0.0	-0.2	0.0

0.4	0.2	-0.2
-0.8	0.0	0.8
0.0	-0.5	0.2



CNN Kernels in 3D

3D Convolution with a Kernel: Principle

0	0	0	0	0	0	...
0	156	155	156	158	158	...
0	153	154	157	159	159	...
0	149	151	155	158	159	...
0	146	146	149	153	158	...
0	145	143	148	148	158	...
...

Input Channel #1 (Red)

0	0	0	0	0	0	...
0	167	166	167	169	169	...
0	164	165	168	170	170	...
0	160	162	166	169	170	...
0	156	156	159	163	168	...
0	155	153	153	158	168	...
...

Input Channel #2 (Green)

0	0	0	0	0	0	...
0	163	162	163	165	165	...
0	160	161	164	166	166	...
0	156	158	162	165	166	...
0	155	155	158	162	167	...
0	154	152	152	157	167	...
...

Input Channel #3 (Blue)

-1	-1	1
0	1	-1
0	1	1

Kernel Channel #1



308

+

1	0	0
1	-1	-1
1	0	-1

Kernel Channel #2



-498

0	1	1
0	1	0
1	-1	1

Kernel Channel #3



164

+

-25				...
				...
				...
				...
...

Output

Bias = 1

CNN Pooling

The Principle

Max Pooling

29	15	28	184
0	100	70	38
12	12	7	2
12	12	45	6

Average Pooling

31	15	28	184
0	100	70	38
12	12	7	2
12	12	45	6

2 x 2
pool size

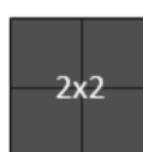
100	184
12	45

2 x 2
pool size

36	80
12	15

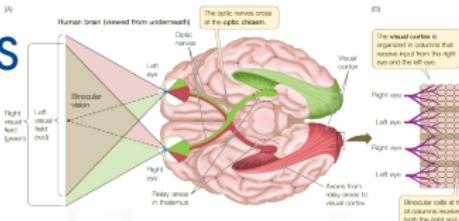
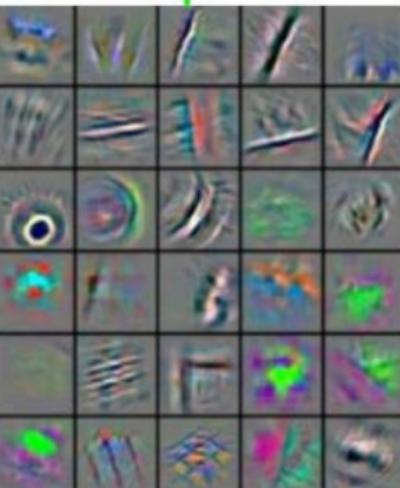
CNN Pooling

On real data (effectively subsampling)



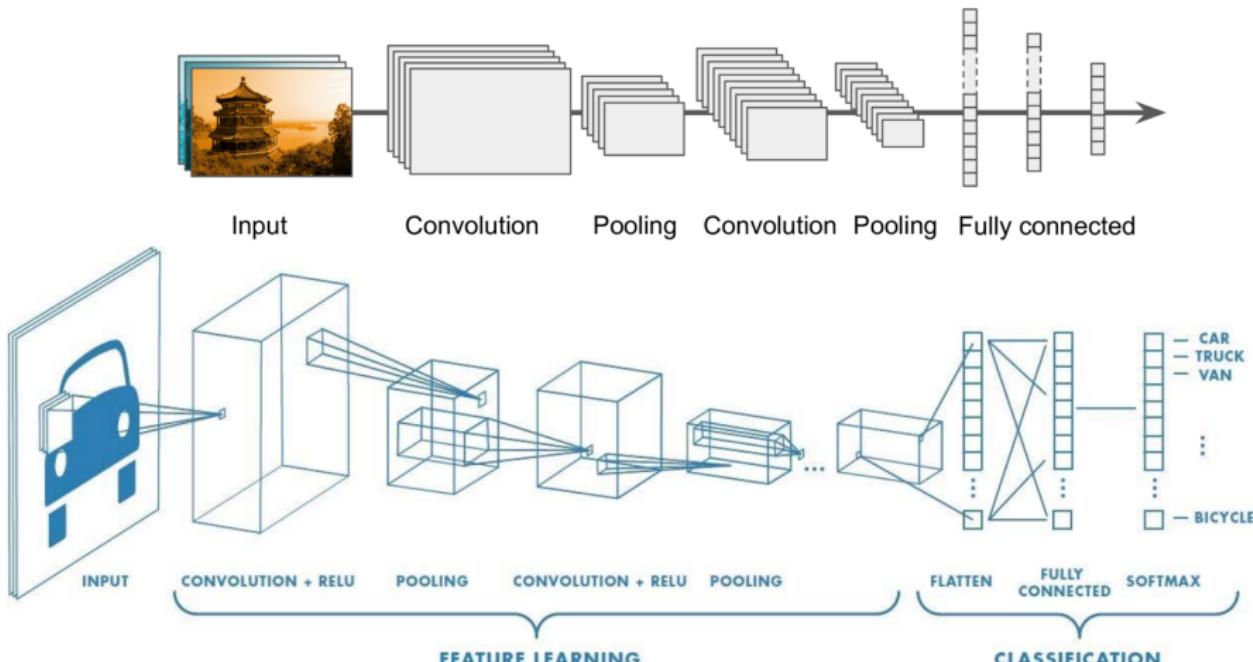
Convolutional Neural Networks

Low-, Mid-, and High-Level Feature Extraction



Convolutional Neural Networks

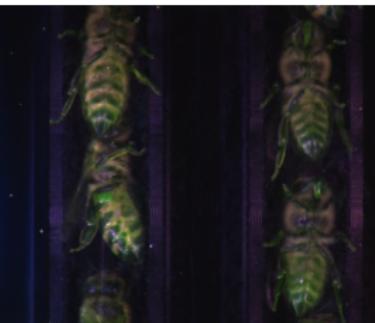
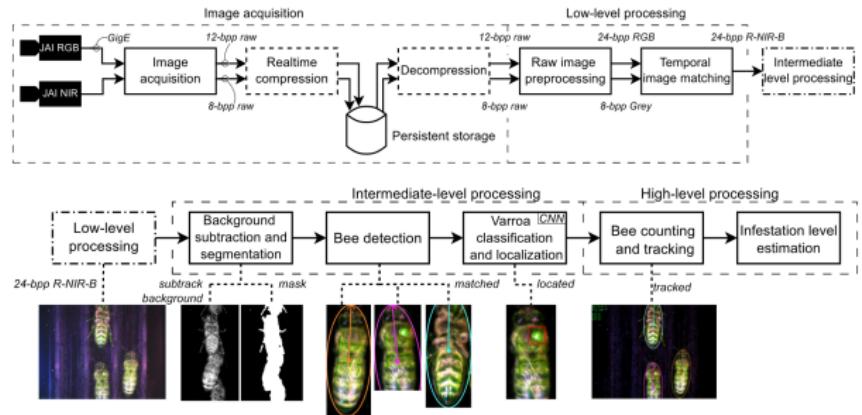
Stacking It All Up..



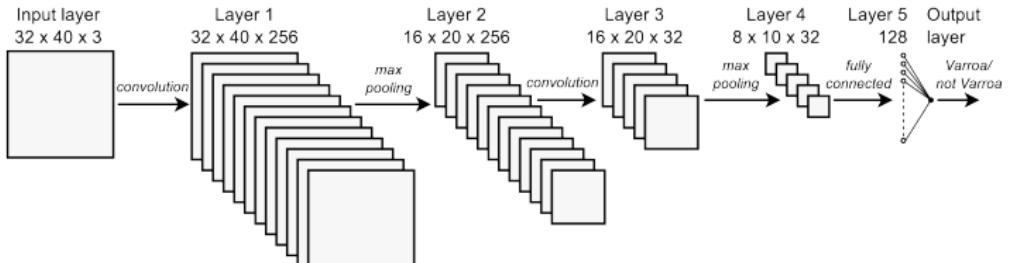
CNN's in Practice

Varroa mite detector—with a CNN somewhere in the pipeline

Image processing pipeline



CNN architecture



A computer vision system to monitor the infestation level of Varroa
distractor in a honeybee colony

Ren Wang^{a,*}, Charles De Paoli^b, Peter Hugh Mikheyev^b, Thomas Roby Nelson^c, Michael Meixner^c,
Pete Riggs^c

^aCollege of Engineering, Arctic University of Norway, Tromsø, Norway; ^bDepartment of Biology, University of Alberta, Edmonton, Alberta, Canada; ^cDepartment of Biological Sciences, York University, North York, Ontario, Canada

Abstract
Varroa mites are a notable parasite of honeybees, threatening the global survival of the species. Honeybees are often monitored by beekeepers using a labor-intensive visual inspection method. However, this method is time-consuming and prone to error. Accordingly, we developed a computer vision system to automatically detect the presence and number of Varroa mites on honeybees. The system consists of three main stages: low-level processing, intermediate-level processing, and high-level processing. The algorithm that was used to determine the number of Varroa mites is presented. Based on the experimental results, the proposed system can detect Varroa mites on honeybees with a 95% accuracy and a 95% completeness in a general case of 1000 bees. The algorithm has a high false alarm rate (less than 1%). The proposed system can be used to monitor the infestation level of Varroa mites on honeybees. The proposed system also provides a fast and efficient way to monitor the infestation level of Varroa mites on honeybees, which requires less labor and time.

Received 20 May 2017; revised 20 June 2017; accepted 20 June 2017
Handling Editor: Brian Lovett; Reviewer: Naveen Kasturi, Amrit Anand, Mohit Srivastava

Keywords: Varroa mite; computer vision; honeybee; infestation level; detection

© 2017 Society for Biotechnology, Biophysics and Bioengineering. All rights reserved. Published by Elsevier B.V. on behalf of Society for Biotechnology, Biophysics and Bioengineering. All rights reserved.

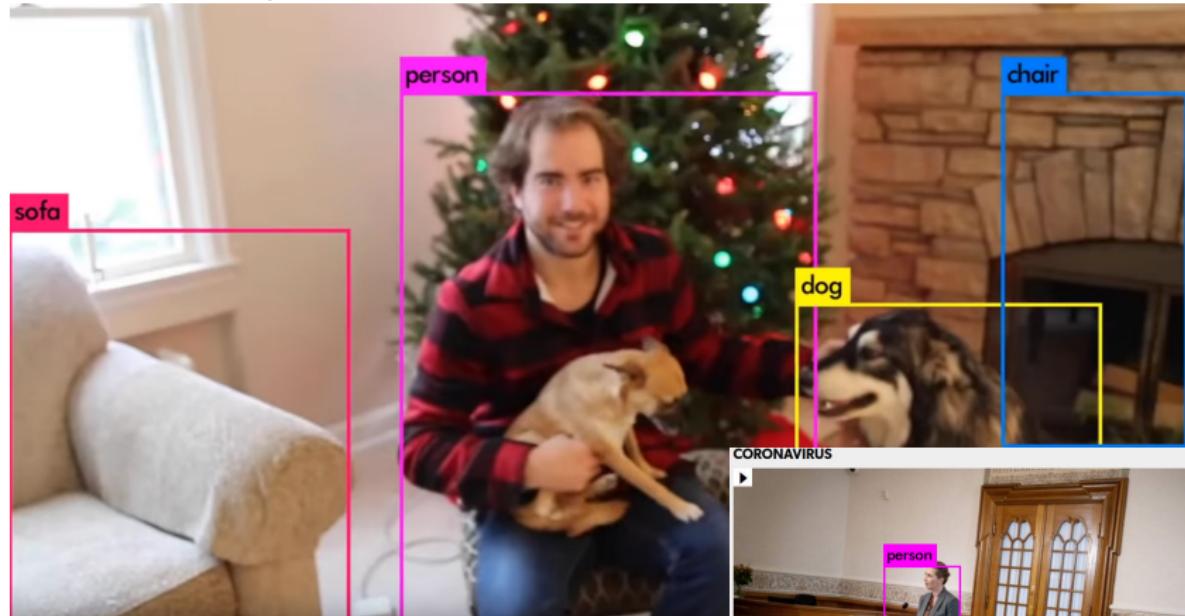
This is an open access article under the terms of the Creative Commons Attribution License (<http://creativecommons.org/licenses/by/4.0/>), which permits unrestricted use, distribution, and reproduction in any medium, provided the original author and source are credited.

Published online 20 July 2017. Available online 20 July 2017

CNN's in Practice

YOLOv3 (You Only Look Once)

Real-time object detection, demo..



[<https://holm.info/yolodemo>]



Mette Frederiksen: Uklogt og uholdbart at vi

CNN's in Practice

The LeNET-5 Architecture

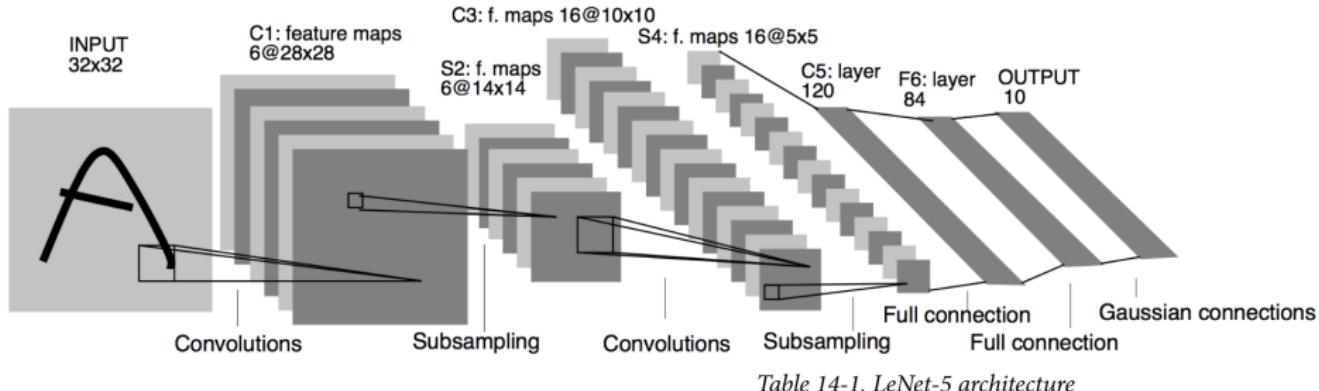


Table 14-1. LeNet-5 architecture

Other famous CNN-architectures:

- ▶ AlexNet,
- ▶ GoogLeNet (inception),
- ▶ ResNet (152 layers,
skip-connections),
- ▶ VGGNet,
- ▶ Inception-v4 (GoogLeNet + ResNet),
- ▶ ...

Layer	Type	Maps	Size	Kernel size	Stride	Activation
Out	Fully Connected	-	10	-	-	RBF
F6	Fully Connected	-	84	-	-	tanh
C5	Convolution	120	1 × 1	5 × 5	1	tanh
S4	Avg Pooling	16	5 × 5	2 × 2	2	tanh
C3	Convolution	16	10 × 10	5 × 5	1	tanh
S2	Avg Pooling	6	14 × 14	2 × 2	2	tanh
C1	Convolution	6	28 × 28	5 × 5	1	tanh
In	Input	1	32 × 32	-	-	-

CNN's in Practice

A LeNET-5 'Like' Architecture

```
In [9]: 1 import keras
2 from keras import layers
3
4 model = keras.Sequential()
5
6 model.add(layers.Conv2D(filters=6, kernel_size=(3, 3),
7                         activation='relu', input_shape=(32,32,1)))
8 model.add(layers.AveragePooling2D())
9
10 model.add(layers.Conv2D(filters=16, kernel_size=(3, 3), activation='relu'))
11 model.add(layers.AveragePooling2D())
12
13 model.add(layers.Flatten())
14
15 model.add(layers.Dense(units=120, activation='relu'))
16 model.add(layers.Dense(units=84, activation='relu'))
17 model.add(layers.Dense(units=10, activation='softmax'))
18
19 model.summary()
```

Layer (type)	Output Shape	Param #
conv2d_5 (Conv2D)	(None, 30, 30, 6)	60
average_pooling2d_5 (Average)	(None, 15, 15, 6)	0
conv2d_6 (Conv2D)	(None, 13, 13, 16)	880
average_pooling2d_6 (Average)	(None, 6, 6, 16)	0
flatten_3 (Flatten)	(None, 576)	0
dense_7 (Dense)	(None, 120)	69240
dense_8 (Dense)	(None, 84)	10164
dense_9 (Dense)	(None, 10)	850
<hr/>		
Total params: 81,194		
Trainable params: 81,194		
Non-trainable params: 0		

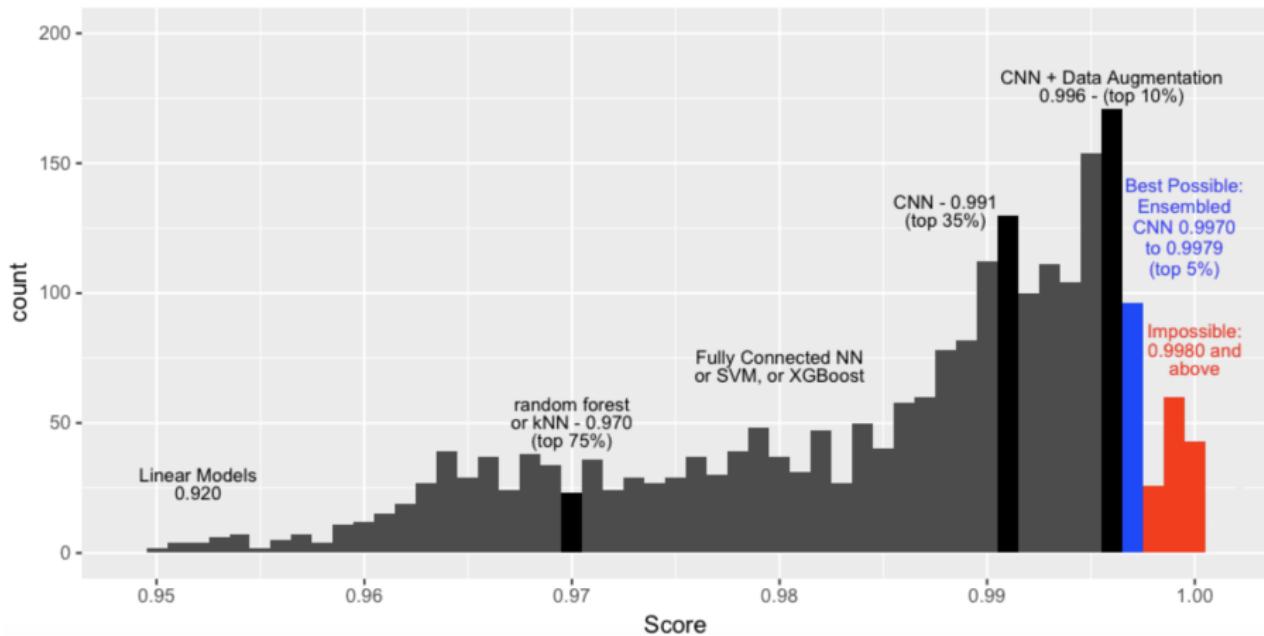
Table 14-1. LeNet-5 architecture

CNN's in Practice

The LeNET-5 Architecture on MNIST

Histogram of Kaggle MNIST

public leaderboard scores, July 15 2018



- ▶ using pre-trained models => Transferred Learning,
- ▶ object detection,
- ▶ semantic segmentation,
- ▶ time series => RNN's, ...

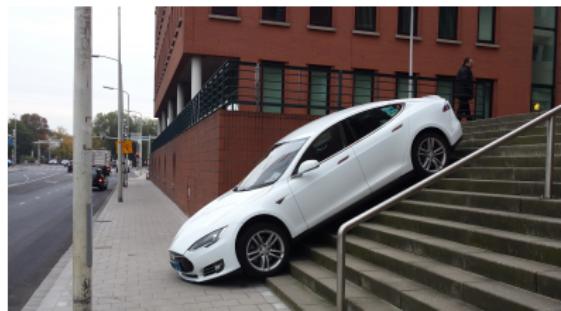
Hardware for Machine Learning)

Methods and Terminology (SKIP most except μ, η)

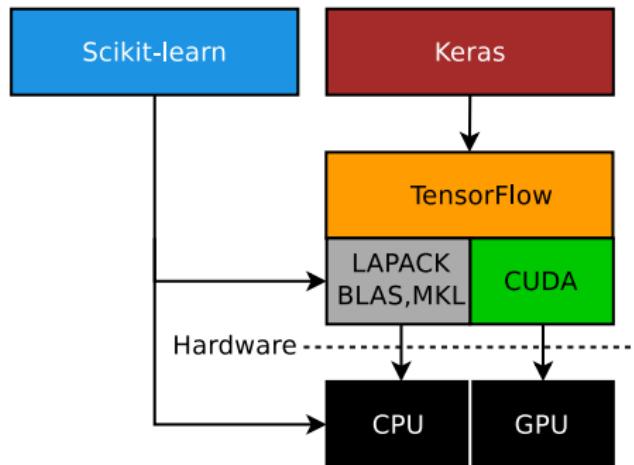
Objective:

Why optimize using 'application specific' hardware?

- ▶ Effectiveness:
 - ▶ cost of purchasing/operating systems,
 $\mu = \text{FLOPS}/\$$
 - $\eta = \text{FLOPS}/\text{Watt}$
 - ▶ cut-down developer waiting time,
 - ▶ make modelling iterations fast (say minutes).
- ▶ Big-data:
 - ▶ enable training on x-large data.
- ▶ Real-time constraints:
 - ▶ inference (on visual data) in real-time,
 - ▶ low-power constraints.



RESUMÉ: Keras and Tensorflow



GP-GPU: General-Purpose Graphics Processing Unit...or just **GPU**.

CUDA: Compute Unified Device Architecture, API for SIMD/SIMT on GPU,

CPUs

Build Tensorflow from source (SKIP)

- ▶ for specific architecture, say ARM,
- ▶ or for HPC optimization for all CPU feature
- > lscpu

```
Architecture: x86_64
Model name:  Intel(R) Core(TM) i7-6600U CPU @ 2.60GHz
Flags:          fpu vme de pse tsc msr pae mce cx8 apic sep mtrr
               pge dts acpi mmx fxsr fma .sse sse2..sse4_1sse4_2..
```



Using Docker and pulling TF from GIT + a lot of scripting!

```
> git clone https://github.com/tensorflow/tensorflow
> git checkout 1.12
(lots of scripting and pain..)
> bazel build -copt=-mfma -copt=-msse4.2 tensorflow
```

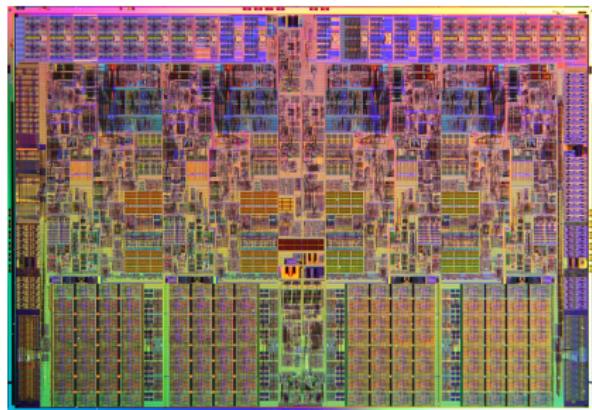
Howtos:

<https://www.tensorflow.org/install/source>

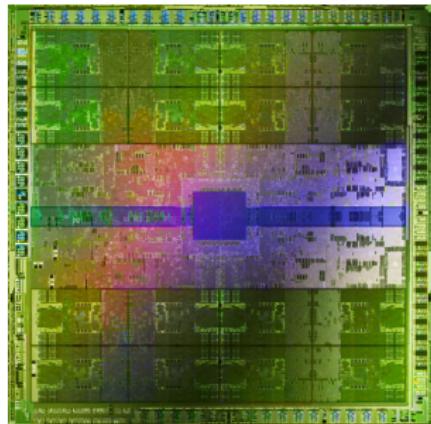
<https://www.pugetsystems.com/labs/hpc/Build-TensorFlow-CPU-with-MKL-and-Anaconda-Python-3-6-using-a-Docker-Container-1133>

<https://www.pugetsystems.com/labs/hpc/Build-TensorFlow-GPU-with-CUDA-9-1-MKL-and-Anaconda-Python-3-6-using-a-Docker-Container-1134>

CPUs vs GPUs



Nehalem CPU
die size: $\sim 700 \text{ mm}^2$
transistors: $\sim 2.3 \cdot 10^9$



Fermi GPU
die size: 520 mm^2
transistors: $\sim 3 \cdot 10^9$

Nvidia arch. transistor counts

Pascal	$\sim 15 \cdot 10^9$
Turing	$\sim 19 \cdot 10^9$
Volta	$\sim 21 \cdot 10^9$
Ampere	$\sim 28 \cdot 10^9$ (GPU 3090, 8nm)

CPUs vs GPUs

So many transistors, but how many for the ALUs/FPUs?

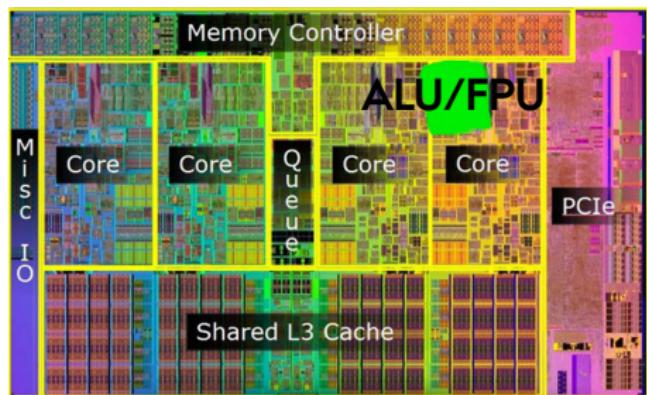
What makes GPUs such excellent HW for Machine Learning?

ALU: Arithmetic logic unit

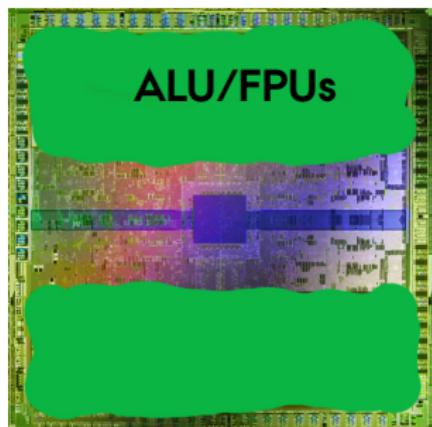
FPU: Floating-point unit

Memory Controller: six controllers on GPU,

CPU: lots of speculative execution; waste of transistors.



CPU (type?)
with one ALU/FPU marked



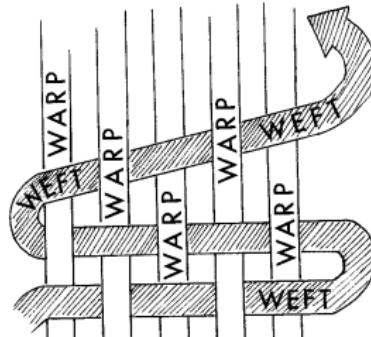
Fermi GPU
ALUs/FPUs all over

GPUs

Fundamental Problems with the GPUs Hardware (SKIP most, except WARP)

GPUs had several *Achilles heels* related to its hardware, many of them addressed in the latest Vola V100 architecture:

- ▶ coding problems graphically, now CUDA,
- ▶ no STACK, now added,
- ▶ no CACHE (or Texture only), now both L₁ and L₂ cache,
- ▶ distinct GPU memory, now UNIFIED memory,
- ▶ SIMT WARP-bunch of 32-threads, now true SIMD,



GPUs

GPU architecture: Core Design, Streaming Multiprocessors



GPUs

Core Design for a SM
(Streaming Multiprocessor)

Volta SM design
(new gen. GPU):

FP64/32:FPU
INT: ALUs
TENSOR CORES: ?

Ampere, RTX, 3090:
Raytracingkerner: ?



GEFORCE RTX 3090

NVIDIA CUDA® kerner

10496

Høj CPU-hastighed (GHz)

1.70

Normal CPU-hastighed (GHz)

1.40

Raytracingkerner

Tensor Cores

3. generation

Standard hukommelseskonfiguration

Hukommelsesgrænsefladens bredde

24 GB GDDR6X

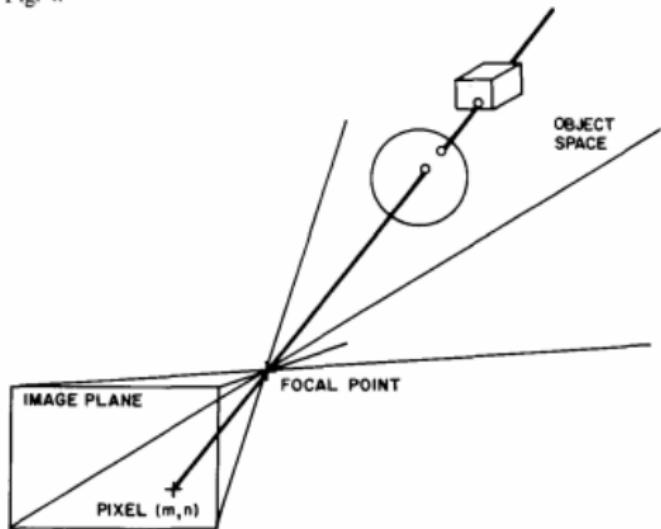
384-bit

GPUs

Tensor cores: Raytracing vs Rasterization on GPUs

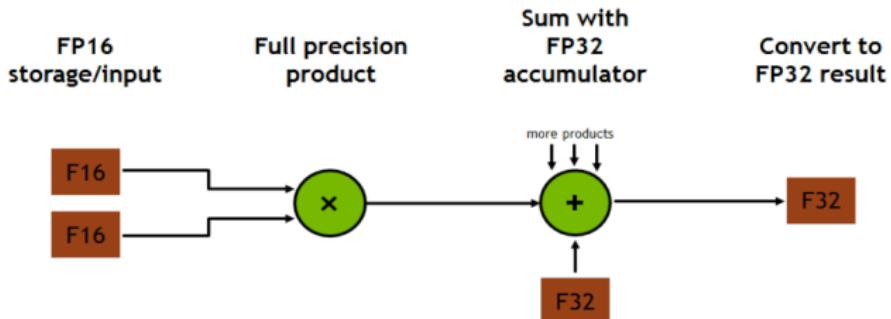


Fig. 4.



GPUs

Tensor Cores
(SKIP most)



$$D = \left(\begin{array}{cccc} A_{0,0} & A_{0,1} & A_{0,2} & A_{0,3} \\ A_{1,0} & A_{1,1} & A_{1,2} & A_{1,3} \\ A_{2,0} & A_{2,1} & A_{2,2} & A_{2,3} \\ A_{3,0} & A_{3,1} & A_{3,2} & A_{3,3} \end{array} \right) \left(\begin{array}{cccc} B_{0,0} & B_{0,1} & B_{0,2} & B_{0,3} \\ B_{1,0} & B_{1,1} & B_{1,2} & B_{1,3} \\ B_{2,0} & B_{2,1} & B_{2,2} & B_{2,3} \\ B_{3,0} & B_{3,1} & B_{3,2} & B_{3,3} \end{array} \right) + \left(\begin{array}{cccc} C_{0,0} & C_{0,1} & C_{0,2} & C_{0,3} \\ C_{1,0} & C_{1,1} & C_{1,2} & C_{1,3} \\ C_{2,0} & C_{2,1} & C_{2,2} & C_{2,3} \\ C_{3,0} & C_{3,1} & C_{3,2} & C_{3,3} \end{array} \right)$$

FP16 or FP32 FP16 FP16 or FP32

Tesla V100's Tensor Cores deliver up to **125 TFLOPS** for training and inference applications.

[volta-architecture-whitepaper.pdf]

Analog: DSD: half-adder, full-adder, ripple-carry-adder with n-bit multiplier based on adders (scales n-bit²)..

HPC Top500

Best High-Performance Computer

[<https://www.top500.org/list/2007/06/>]

[https://en.wikipedia.org/wiki/List_of_Nvidia_graphics_processing_units]



HOME	LISTS ▾	STATISTICS ▾	RESOURCES
------	---------	--------------	-----------

Home » Lists » TOP500 » June 2007

JUNE 2007

Rank	System	Cores	Rmax [TFlop/s]	Rpeak [TFlop/s]	Power (kW)
1	BlueGene/L - eServer Blue Gene Solution, IBM DOE/NNSA/LLNL United States	131,072	280.6	367.0	1,433
2	Jaguar - Cray XT4/XT3, Cray/HPE DOE/SC/Oak Ridge National Laboratory United States	23,016	101.7	119.3	
3	Red Storm - Sandia/ Cray Red Storm, Opteron 2.4 GHz dual core, Cray/HPE NNSA/Sandia National Laboratories United States	26,544	101.4	127.4	

GPUs

When is the GPU faster than the CPU for NN?

GPU slower for CPU for a three-layer NN + MNIST, why?

- ▶ GPU needs a reasonable amount of trainable parameters + data to beat the CPU!

`model.summary():`

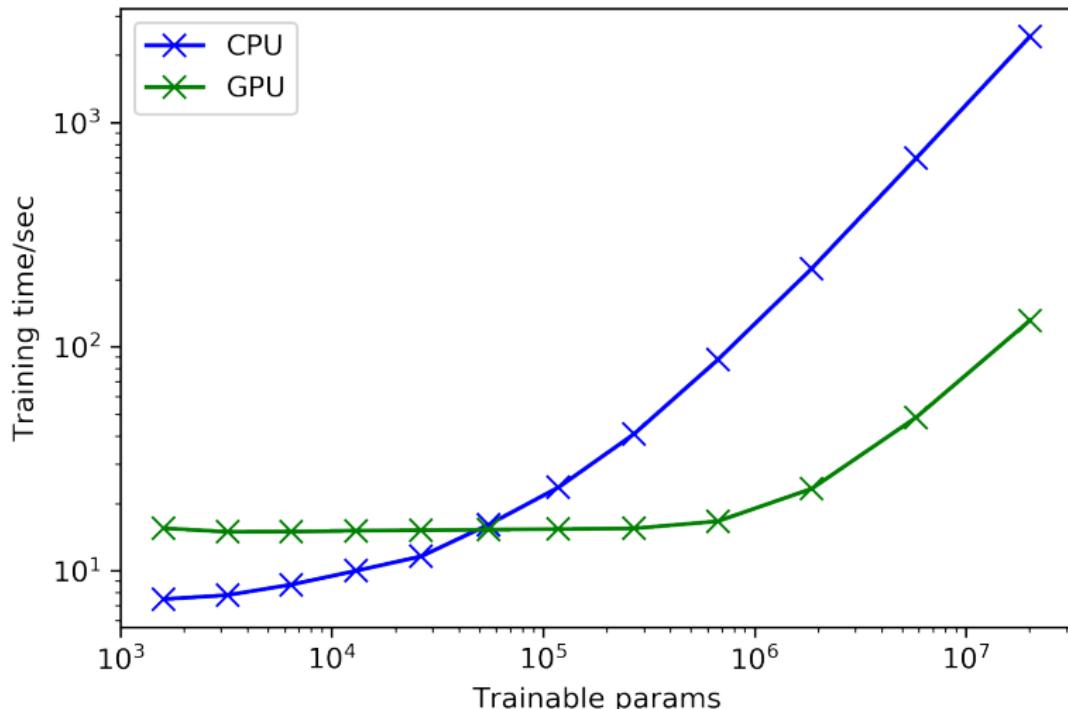
```
1 # i=12, n=4096
2 #
3 # Layer (type)          Output Shape       Param #
4 # =====
5 # dense_46 (Dense)     (None, 4096)        3215360
6 #
7 # dropout_31 (Dropout) (None, 4096)        0
8 #
9 # dense_47 (Dense)     (None, 4096)        16781312
10 #
11 # dropout_32 (Dropout) (None, 4096)        0
12 #
13 # dense_48 (Dense)     (None, 10)          40970
14 # =====
15 # Total params: 20,037,642
16 # Trainable params: 20,037,642
17 # Non-trainable params: 0
```

GPUs

Actual test on the GPU-server

CPU vs GPU on MNIST for a three layer NN with dropout...

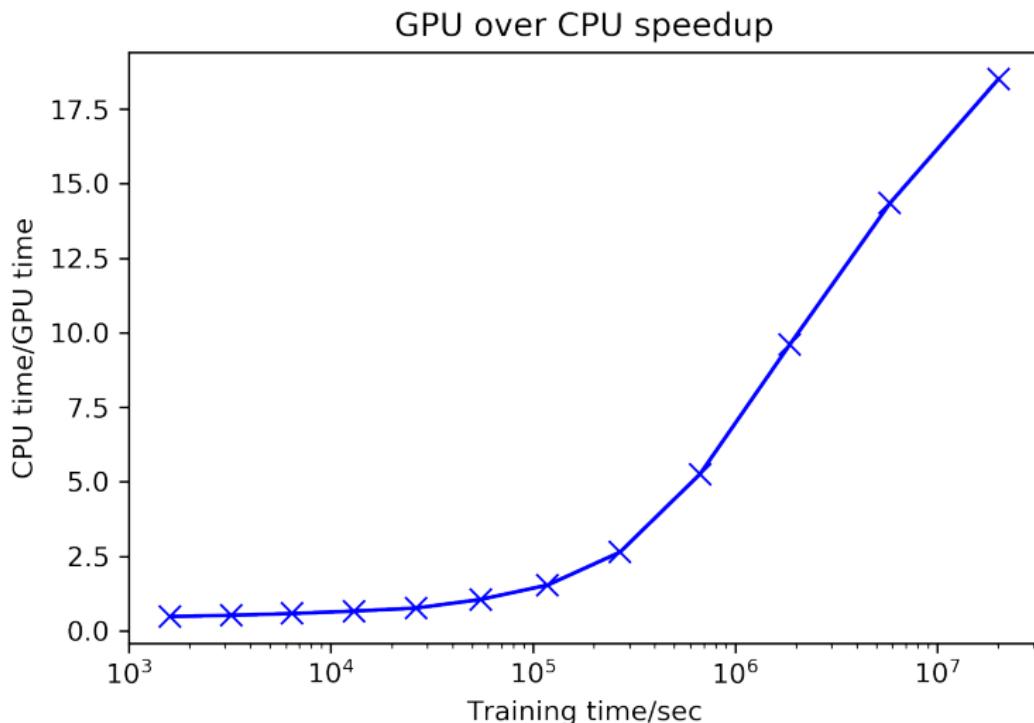
Training time-vs-Trainable params



GPUs

Actual test on the GPU-server

CPU vs GPU on MNIST for a three layer NN with dropout...



TPUs

Tensor Processing Units

Custom ASICs by Google



Cloud TPU v3

Launched in 2018

Inference and training



Dev Board

A development board to quickly prototype on-device ML products. Scale from prototype to production with a removable system-on-module (SoM).

→ Datasheet
→ Get started guide

\$149.99

Buy



TPU v2

Launched in 2015

Inference only

Launched in 2017
Inference and training

Dev board with Google Edge TPU ML accelerator coprocessor

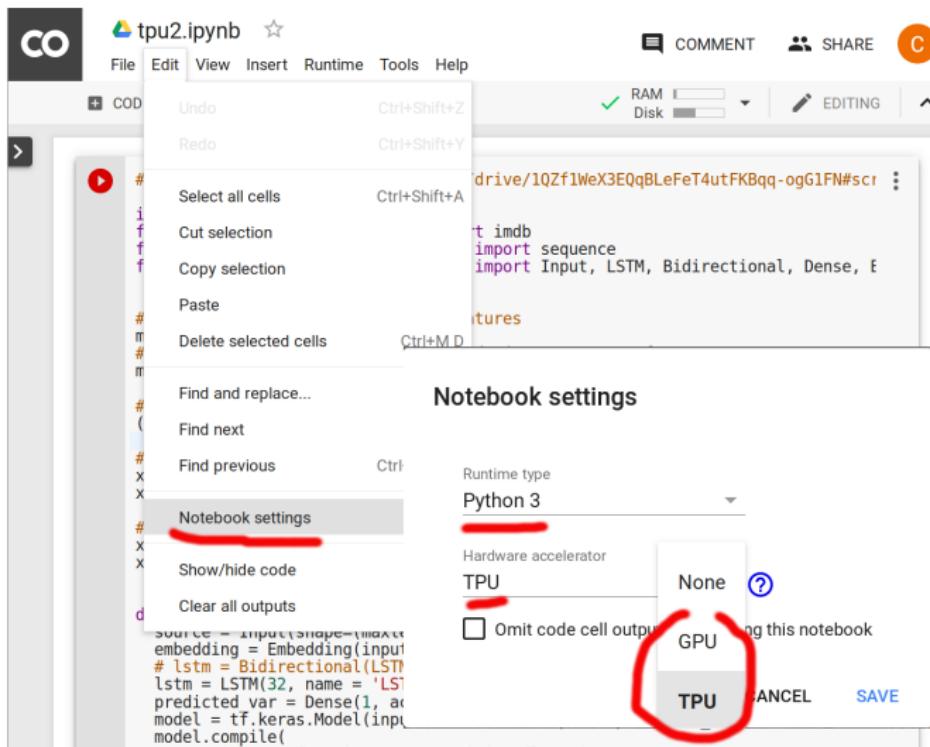
[\[https://coral.withgoogle.com/products/dev-board/\]](https://coral.withgoogle.com/products/dev-board/)



TPUs

Access to TPUs (SKIP)

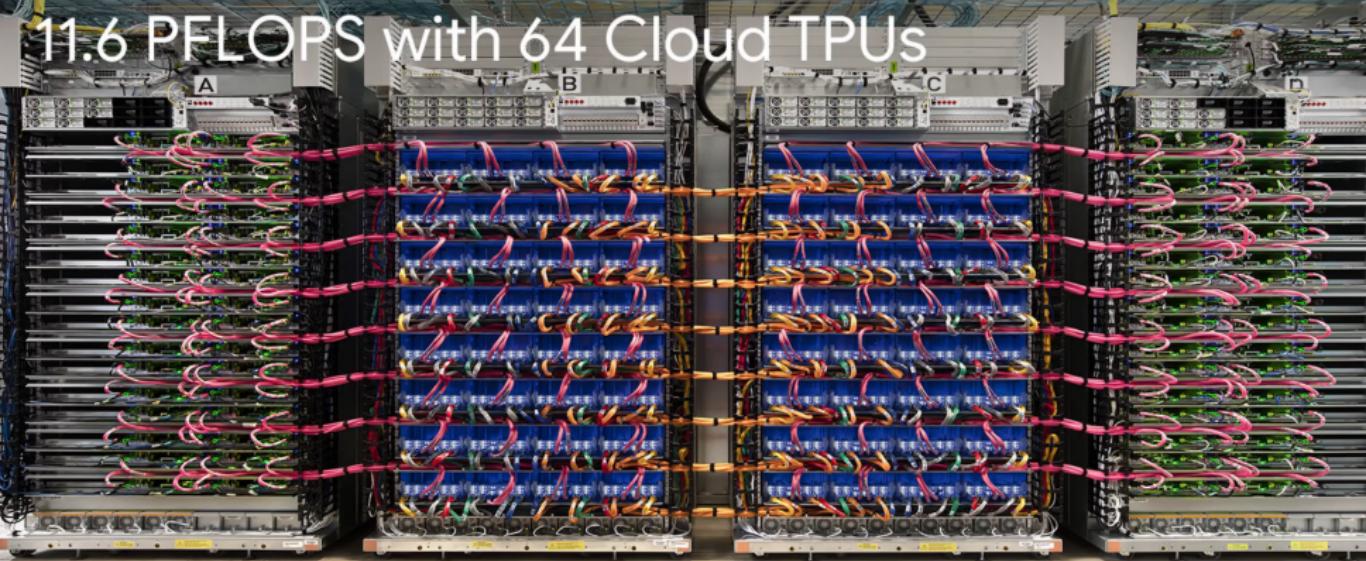
Free Jupyter Notebook environment with access to TPUs:
<https://colab.research.google.com>



TPUs

TPU Cloud

TPU v2 Pod: Google's HPC cluster for ML
11.6 PFLOPS with 64 Cloud TPUs



[<https://storage.googleapis.com/nexttpu/index.html>]

TPUs vs GPUs

Performance, TPUs vs GPUs, who wins? (SKIP most)

- ▶ Huge advantage for TPU performance-per-watt,
- ▶ Colab performance:
inconclusive (TPU part does not work yet),
- ▶ TPU only for inference?

	K80 2012	TPU 2015	P40 2016
Inferences/Sec <10ms latency	1/13 TH	1X	2X
Training TOPS	6 FP32	NA	12 FP32
Inference TOPS	6 FP32	90 INT8	48 INT8
On-chip Memory	16MB	24 MB	11 MB
Power	300W	75W	250W
Bandwidth	320 GB/S	34 GB/S	350 GB/S

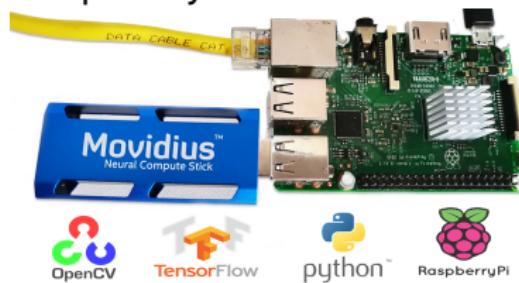
[<https://www.extremetech.com/computing/247403-nvidia-claims-pascal-gpus-challenge-googles-tensorflow-tpu-updated-benchmarks>]

Exotic Hardware

- ▶ Intel Phi multicore CPU, 64 i386 cores:



- ▶ Raspberry PIs + Intel Movidius stick



- ▶ FPGAs

