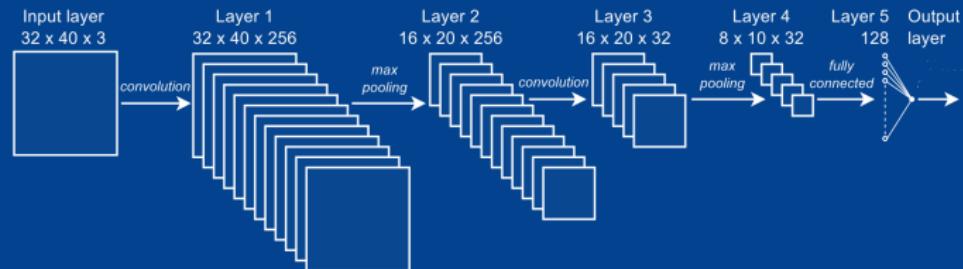




LESSON 09: Deep Learning + CNN's + Hardware

CARSTEN EIE FRIGAARD

SPRING 2021



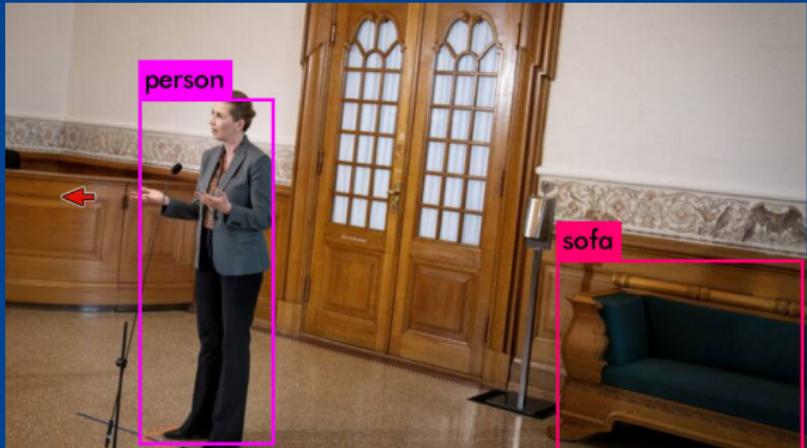
"A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P , if its performance at tasks in T , as measured by P , improves with experience E." — Mitchell (1997).

L09: DL + CNN's + HW, Agenda

- ▶ Spørge-minutter..
- ▶ Admin:
 - ▶ O3 opgavesæt afsluttet, Deadline 14/4, 18:00:
 - ▶ Opgave fra L07: `capacity_under_overfitting.ipynb`
 - ▶ Opgave fra L07: `generalization_error.ipynb`
 - ▶ Opgave fra L08: `regulizers.ipynb`
 - ▶ Opgave fra L08: `gridsearch.ipynb`
 - ▶ **'Cheat'-review** af O3: før aflevering, se beskrivelse i O1+3
'cheat'-review. Resume skal denne gang inkluderes i O3 afleveringen.
 - ▶ Ændringer i "Almindelige formelle journal krav":
 - ▶ NY: Ingen direkte genbrug af overskrifter ...
 - ▶ NY: Skriv kun på eet sprog igennem hele journalen..

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

CONVOLUTIONAL NEURAL NETWORKS



Deep-learning (DL)

Artificial Intelligence:

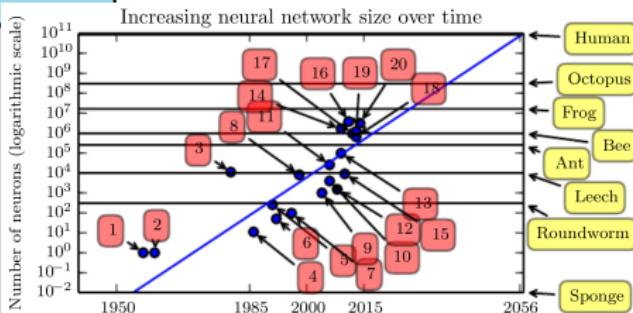
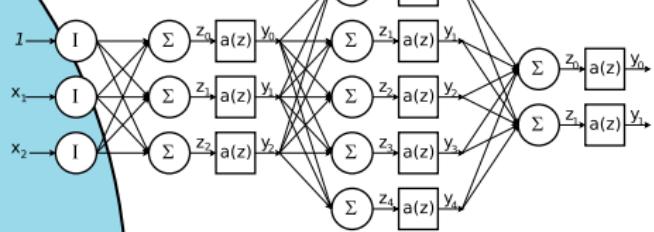
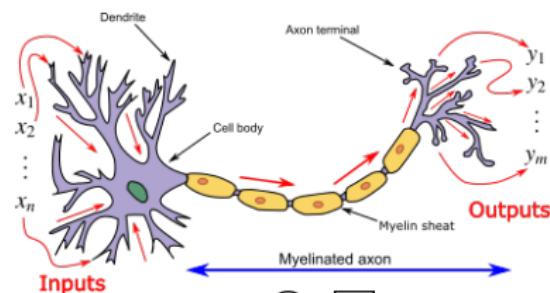
Mimicking the intelligence or behavioural pattern of humans or any other living entity.

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:

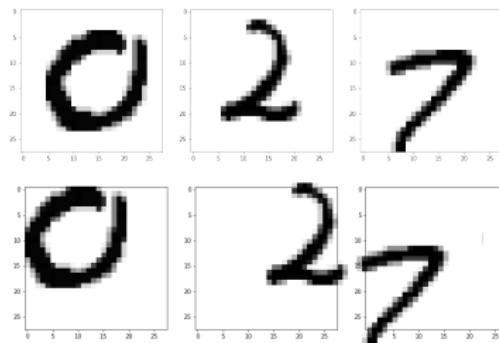
A technique to perform machine learning inspired by our brain's own network of neurons.



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?

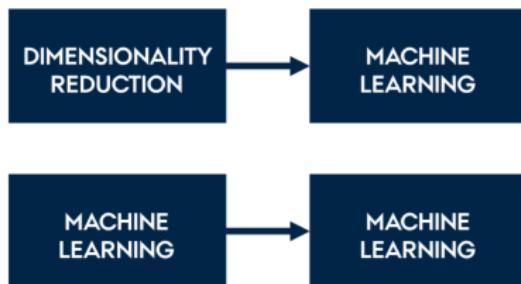
Introducing the Convolutional Neural Networks: trade off

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

Convolutional Neural Networks

Feature Extraction vs. Feature Learning

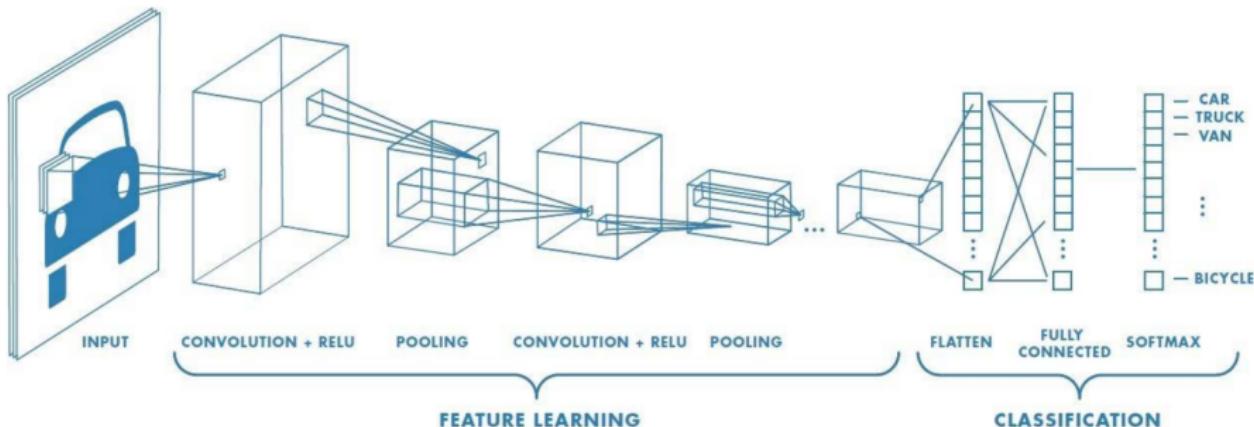
- ▶ Smart filtering:
the distinction bewteen dimensionality reduction and machine learning blurs..



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

Convolutional Neural Networks

Feature Extraction vs. Feature Learning

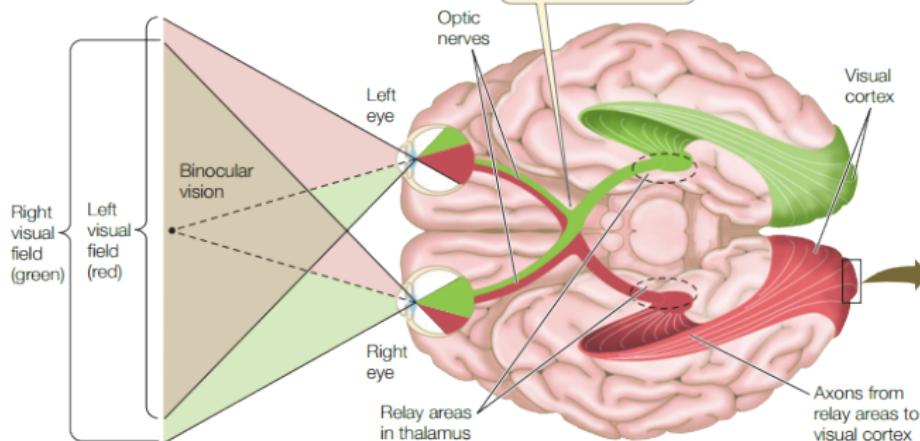


Convolutional Neural Networks

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

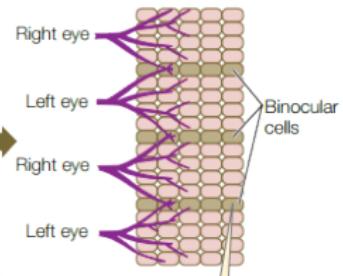
(A)

Human brain (viewed from underneath)



(B)

The visual cortex is organized in columns that receive input from the right eye and the left eye.



Binocular cells at the borders of columns receive input from both the right and left eyes.

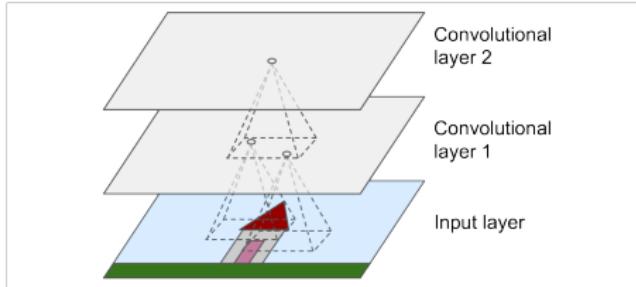


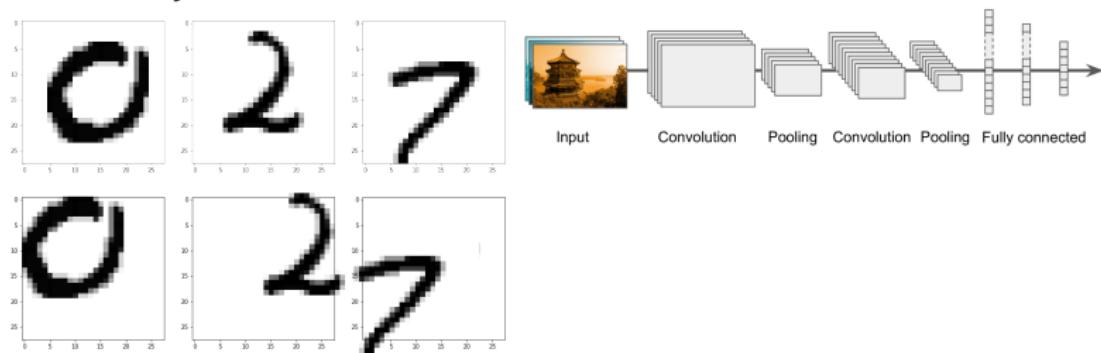
Figure 14-2. CNN layers with rectangular local receptive fields

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)

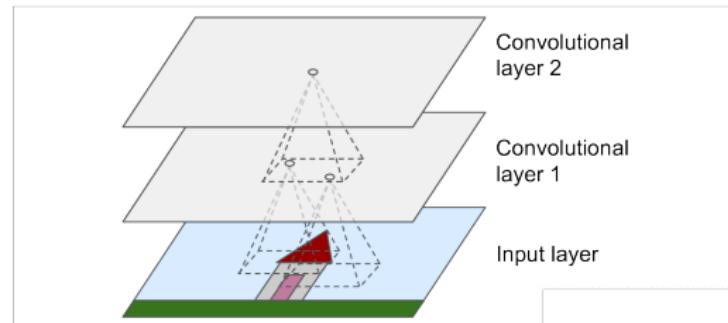


Figure 14-2. CNN layers with rectangular local receptive fields

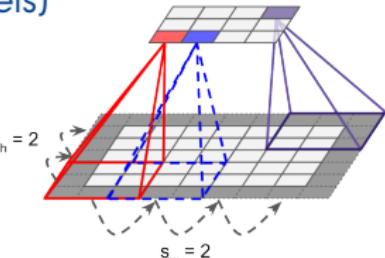
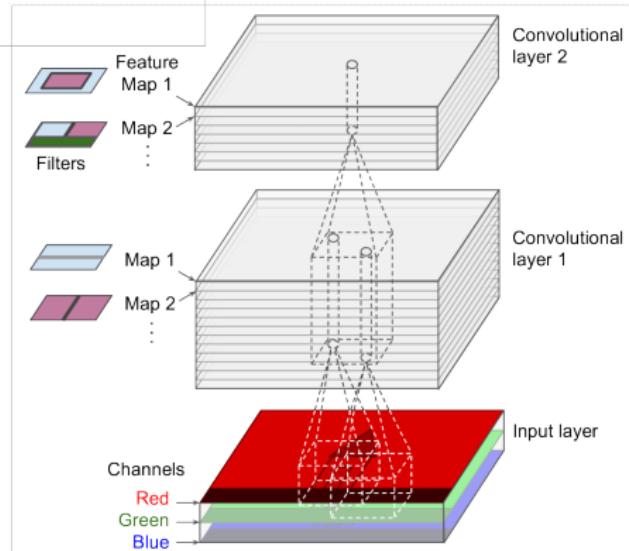


Figure 14-4. Reducing dimensionality using a stride of 2



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 |

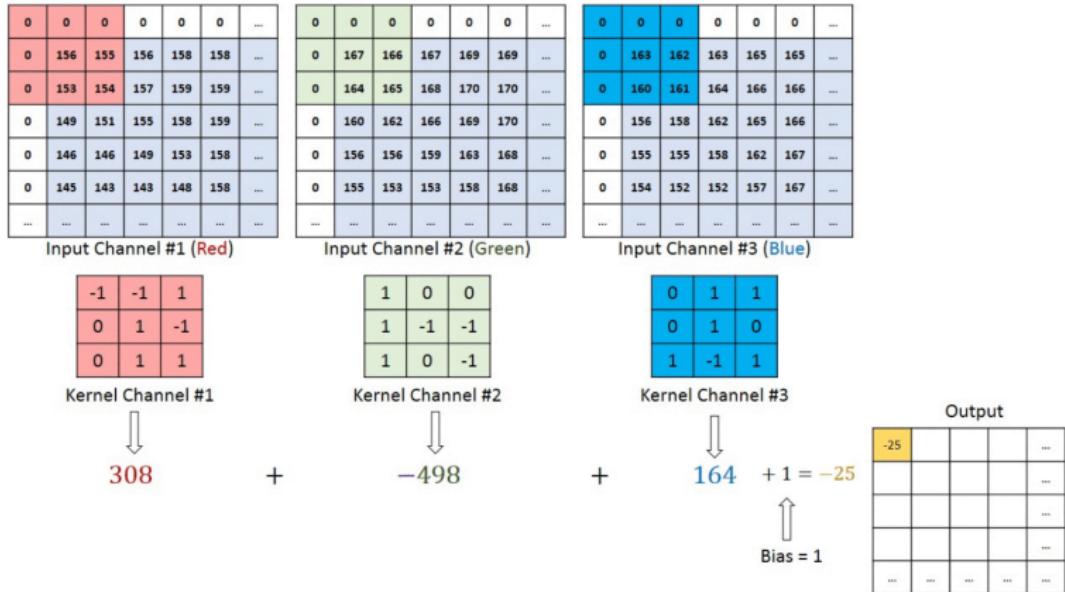
| | | |
|-----|------|-----|
| 0.0 | 0.0 | 0.0 |
| 0.8 | -0.5 | 0.8 |
| 0.0 | -0.2 | 0.0 |

| | | |
|------|------|------|
| 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



CNN Pooling

The Principle

Max Pooling

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

2 x 2
pool size

| | |
|-----|-----|
| 100 | 184 |
| 12 | 45 |

Average Pooling

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

2 x 2
pool size

| | |
|----|----|
| 36 | 80 |
| 12 | 15 |

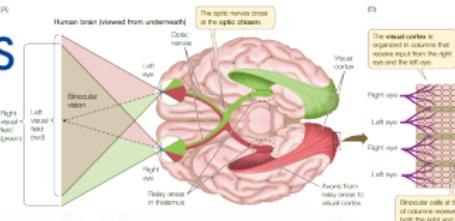
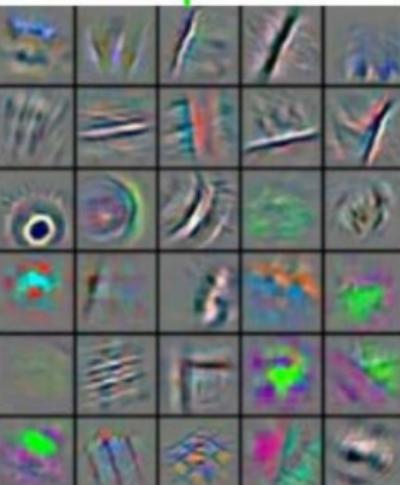
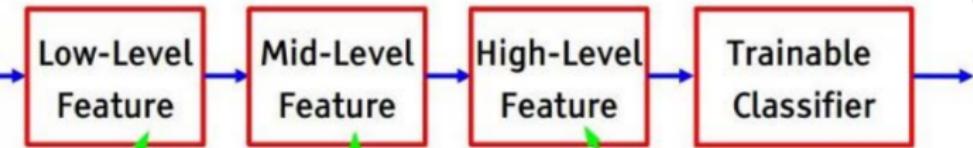
CNN Pooling

On real data (effectively subsampling)



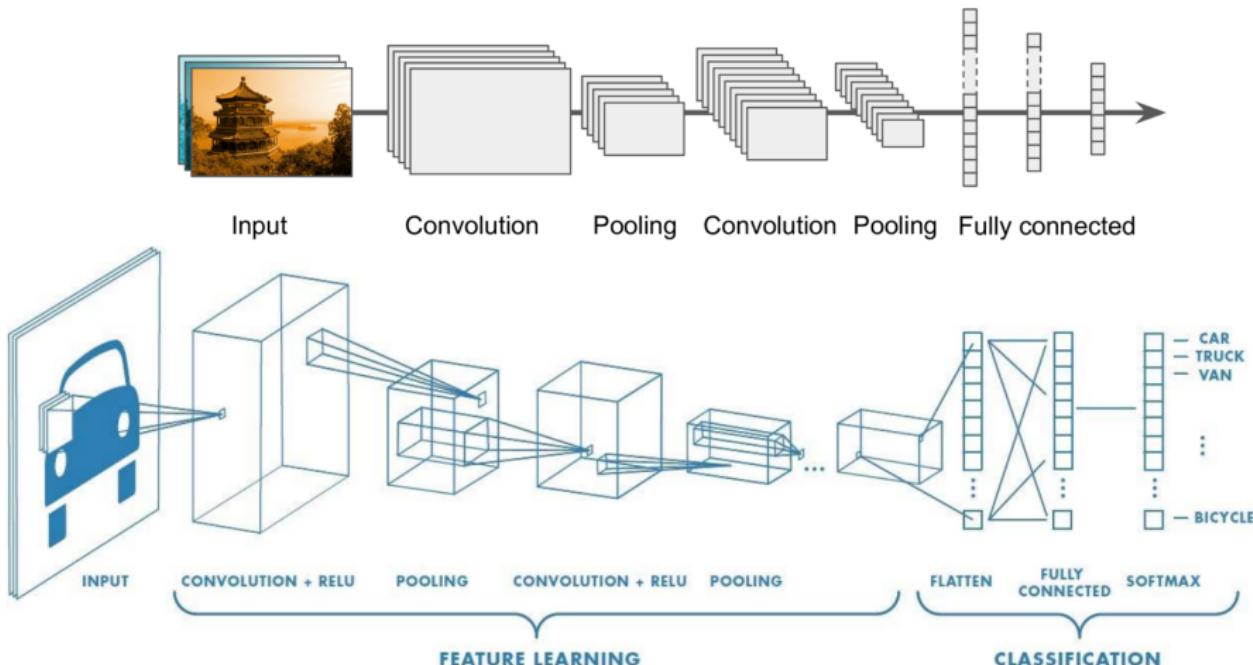
Convolutional Neural Networks

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



Convolutional Neural Networks

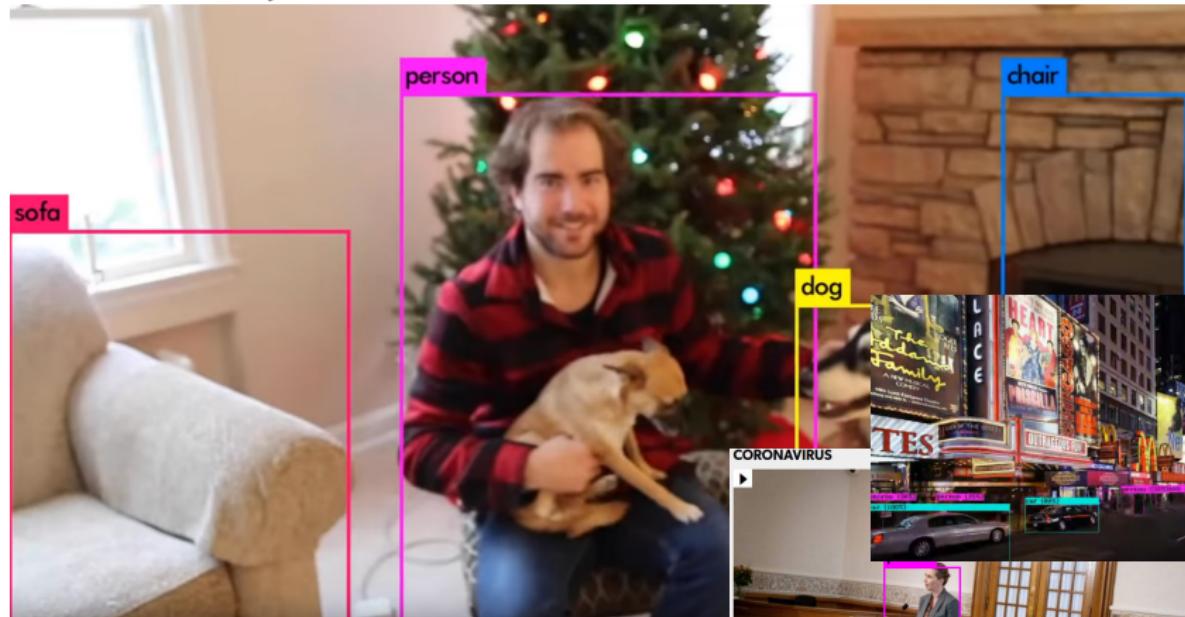
Stacking It All Up..



CNN's in Practice

YOLOv2+3+4 (You Only Look Once)

Real-time object detection, demo..



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

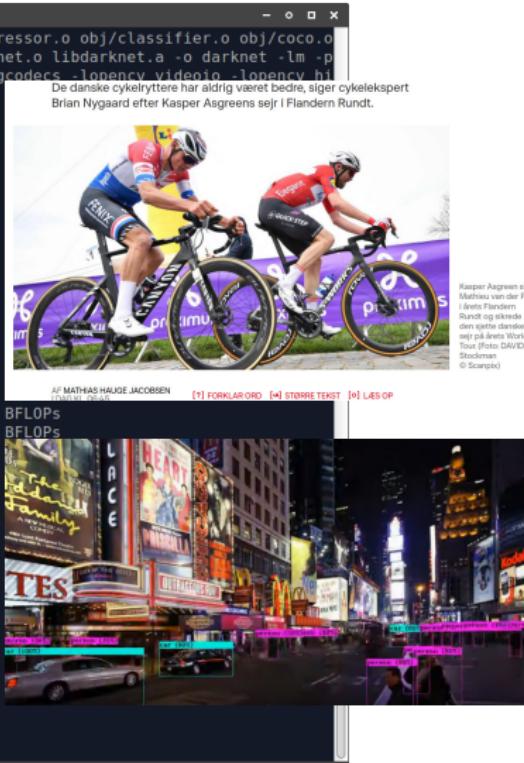
[<https://github.com/AlexeyAB/darknet>]

Mette Frederiksen: Uklogt og uholdbart at vi

CNN's in Practice

YOLOv2+3+4 (You Only Look Once)

```
[~] Terminal
o obj/art.o obj/tag.o obj/cifar.o obj/go.o obj/rnn.o obj/segmenter.o obj/regressor.o obj/classifier.o obj/coco.o
obj/yolo.o obj/detector.o obj/nightmare.o obj/instance-segmenter.o obj/darknet.o libdarknet.a -o darknet -lm -lpthread
-L/opt/opencv/opencv4/lib -lopencv_core -lopencv_imgproc -lopencv_imgcodecs -lopencv_videoio -lopencv_highgui
-ltiff -lstdc++ libdarknet.a
layer    filters   size      input           output
 0 conv     32  3 x 3 / 1  256 x 256 x  3  ->  256 x 256 x  32  0.113
 1 max      2 x 2 / 2  256 x 256 x 32  ->  128 x 128 x 32
 2 conv     64  3 x 3 / 1  128 x 128 x 32  ->  128 x 128 x 64  0.604
 3 max      2 x 2 / 2  128 x 128 x 64  ->  64 x 64 x 64
 4 conv    128  3 x 3 / 1  64 x 64 x 64  ->  64 x 64 x 128  0.604
 5 conv     64  1 x 1 / 1  64 x 64 x 128 ->  64 x 64 x 64  0.067
 6 conv    128  3 x 3 / 1  64 x 64 x 64  ->  64 x 64 x 128  0.604
 7 max      2 x 2 / 2  64 x 64 x 128 ->  32 x 32 x 128
 8 conv    256  3 x 3 / 1  32 x 32 x 128 ->  32 x 32 x 256  0.604
 9 conv    128  1 x 1 / 1  32 x 32 x 256 ->  32 x 32 x 128  0.067
10 conv   256  3 x 3 / 1  32 x 32 x 128 ->  32 x 32 x 256  0.604
11 max      2 x 2 / 2  32 x 32 x 256 ->  16 x 16 x 256
12 conv   512  3 x 3 / 1  16 x 16 x 256 ->  16 x 16 x 512  0.604
13 conv   256  1 x 1 / 1  16 x 16 x 512 ->  16 x 16 x 256  0.067
14 conv   512  3 x 3 / 1  16 x 16 x 256 ->  16 x 16 x 512  0.604
15 conv   256  1 x 1 / 1  16 x 16 x 512 ->  16 x 16 x 256  0.067
16 conv   512  3 x 3 / 1  16 x 16 x 256 ->  16 x 16 x 512  0.604
17 max      2 x 2 / 2  16 x 16 x 512 ->  8 x 8 x 512
18 conv   1024 3 x 3 / 1  8 x 8 x 512 ->  8 x 8 x 1024  0.604
19 conv   512  1 x 1 / 1  8 x 8 x 1024 ->  8 x 8 x 512  0.067
20 conv   1024 3 x 3 / 1  8 x 8 x 512 ->  8 x 8 x 1024  0.604
21 conv   512  1 x 1 / 1  8 x 8 x 1024 ->  8 x 8 x 512  0.067
22 conv   1024 3 x 3 / 1  8 x 8 x 512 ->  8 x 8 x 1024  0.604
23 conv   1000 1 x 1 / 1  8 x 8 x 1024 ->  8 x 8 x 1000  0.131
24 avg          8 x 8 x 1000 ->  1000
25 softmax
Loading weights from darknet19.weights...Done!
test.jpg: Predicted in 1.965434 seconds.
64.68%: mountain bike
16.00%: web site
12.87%: bicycle-built-for-two
1.05%: crash helmet
0.86%: alp
cef@leno:~/textmal/darknet$
```



CNN's in Practice

The LeNET-5 Architecture

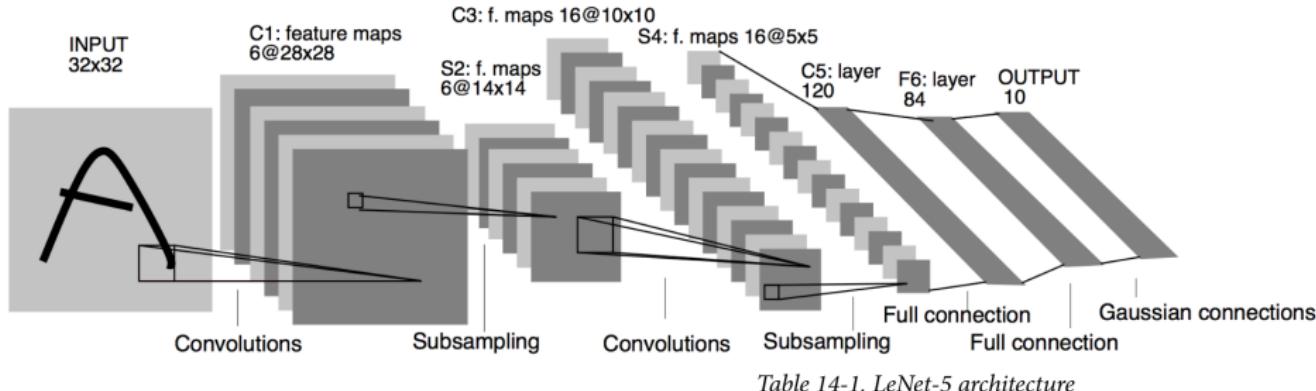


Table 14-1. LeNet-5 architecture

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()
```

Table 14-1. LeNet-5 architecture

| 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 | - | - | - |

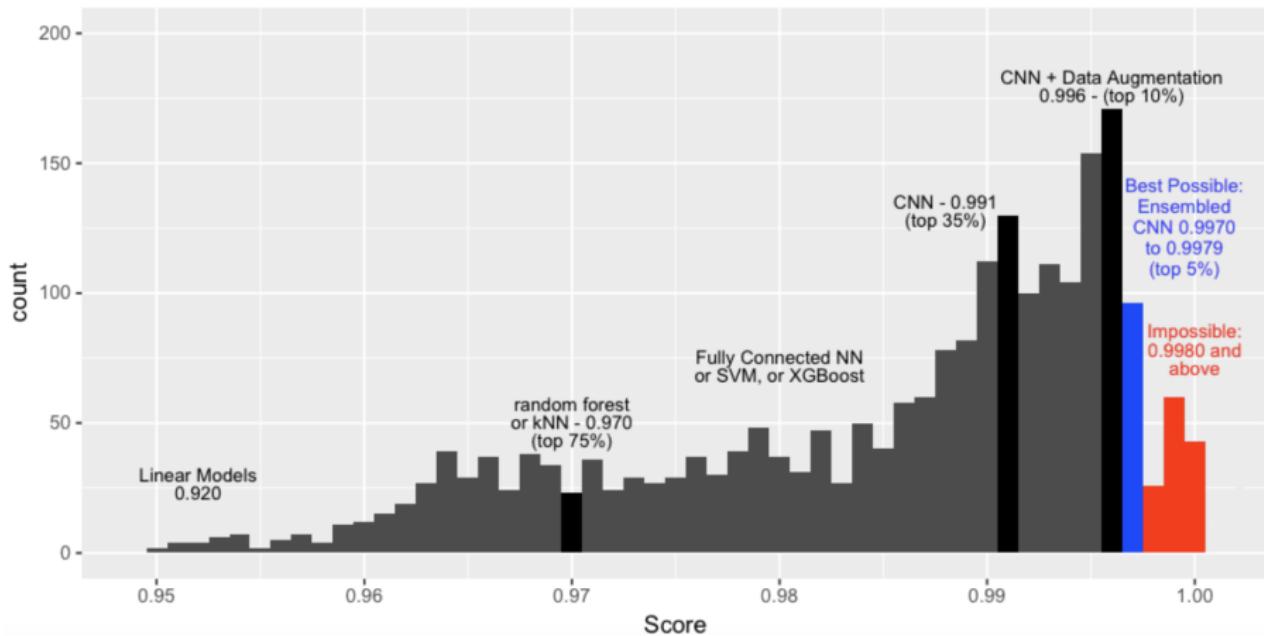
| Layer (type) | Output Shape | Param # |
|-------------------------------|--------------------|---------|
| conv2d_5 (Conv2D) | (None, 30, 30, 6) | 60 |
| average_pooling2d_5 (Average) | (None, 15, 15, 6) | 0 |
| conv2d_6 (Conv2D) | (None, 13, 13, 16) | 880 |
| average_pooling2d_6 (Average) | (None, 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 |
| ===== | | |
| Total params: 81,194 | | |
| Trainable params: 81,194 | | |
| Non-trainable params: 0 | | |

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, ...

Qd MNIST Search Quest II

Fra tidligere semester:

Highest score = 0.97369,

→ ITMALGpr28:

```
1 model=KNeighborsClassifier(  
2     algorithm='ball_tree',  
3     n_neighbors=3,  
4     p=4,  
5     weights='distance')
```

```
12/10 11:13> "ITMALGrp17: best: dat mnist, score=0.97206,  
model=KNeighborsClassifier(leaf_size=40, p=4)"  
08/10 22:56> "ITMALGrp09: dat=mnist, score=0.96973,  
model=KNeighborsClassifier(leaf_size=10, n_neighbors=3, weights='distance')"  
08/10 22:04> "ITMALGrp09: dat=mnist, score=0.93550,  
model=KNeighborsClassifier(leaf_size=10, n_neighbors=3, weights='distance')"  
08/10 10:02> "08/10 10:01 Is from 'ITMALGrp28'"  
08/10 10:01> "best: dat=mnist, score=0.97369,  
model=KNeighborsClassifier(algorithm='ball_tree', n_neighbors=4, weights='distance')"  
08/10 08:53> "ITMALGrp17: dat=mnist, score=0.47090, model=KNeighborsClassifier(algorithm='auto',  
n_neighbors=3, weights='distance', p=2, metric='minkowski', metric_params=None, n_jobs=1),  
score=0.97206"  
07/10 21:43> "ITMALGrp25: KNeighborsClassifier(algorithm='ball_tree', leaf_size=30,  
metric='minkowski', metric_params=None, n_jobs=None, n_neighbors=5, p=2, weights='uniform'),  
score=0.96771<2857142857"  
07/10 18:53> "ITMALGrp21: dat=mnist, score=0.96878,  
model=KNeighborsClassifier(algorithm='ball_tree', n_neighbors=3)"  
06/10 22:54> "ITMALGrp08: dat=mnist, score=0.96973,  
model=KNeighborsClassifier(algorithm='ball_tree', leaf_size=10, n_neighbors=3, weights='distance')"  
29/09 13:40> "ITMALGrp05: dat=mnist , score=0.97257,  
model=KNeighborsClassifier(n_neighbors=4,p=2,weights='distance')"  
24/09 13:55> "ITMALGrp02: dat=mnist, score=0.96810, model=RandomForestClassifier(n_estimators=1000)*
```

Training time?

```
12/10 11:13> "ITMALGrp17: score=0.97206, model=KNeighborsClassifier...  
08/10 22:56> "ITMALGrp09: score=0.96973, model=KNeighborsClassifier...  
08/10 22:04> "ITMALGrp09: score=0.93550, model=KNeighborsClassifier...  
08/10 10:02> "08/10 10:01 Is from 'ITMALGrp28'"  
08/10 10:01> " score=0.97369, model=KNeighborsClassifier...  
08/10 08:53> "ITMALGrp17: score=0.47090, model=KNeighborsClassifier...  
07/10 21:43> "ITMALGrp25: score=0.9677, model=KNeighborsClassifier...  
07/10 18:53> "ITMALGrp21: score=0.96878, model=KNeighborsClassifier...  
06/10 22:54> "ITMALGrp08: score=0.96973, model=KNeighborsClassifier...  
29/09 13:40> "ITMALGrp05: score=0.97257, model=KNeighborsClassifier...  
24/09 13:55> "ITMALGrp02: score=0.96810, model=RandomForestClassifier...
```

HARDWARE FOR MACHINE LEARNING



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).

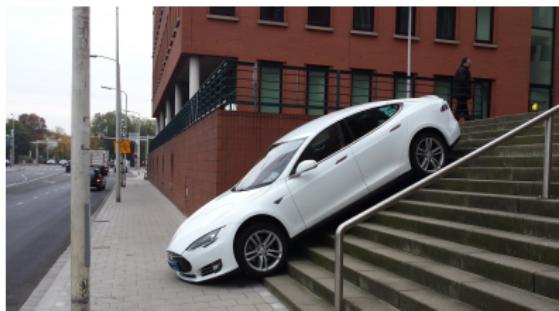
▶ Big-data:

▶ enable training on x-large data.

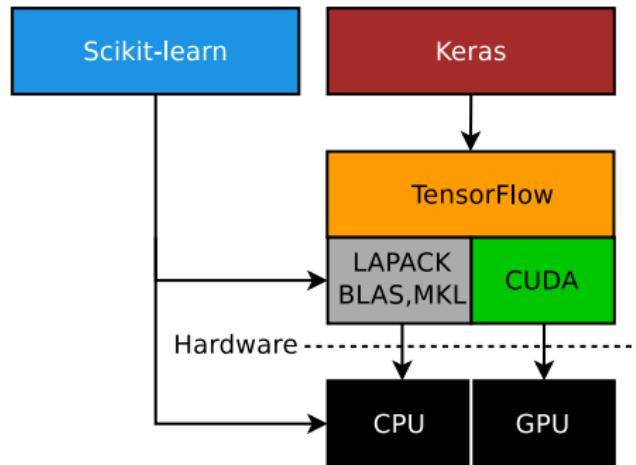
▶ Real-time constraints:

▶ 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
```

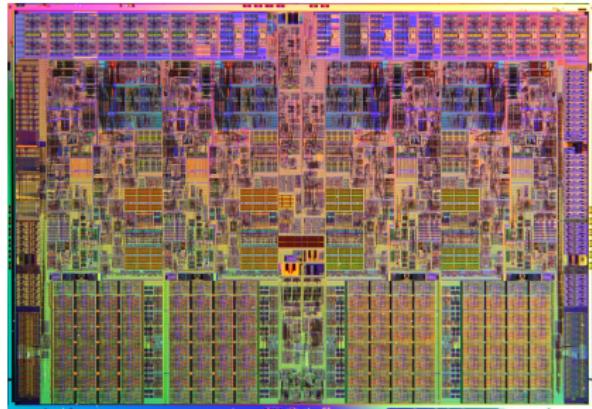
Howtos:

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

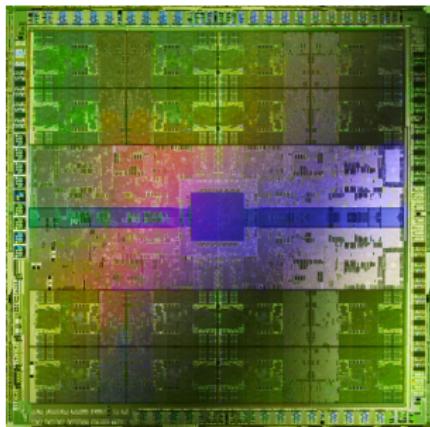
<https://www.pugetsystems.com/labs/hpc/Build-TensorFlow-CPU-with-MKL-and-Anaconda-Python-3-6-using-a-Docker-Container-1133>

<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?

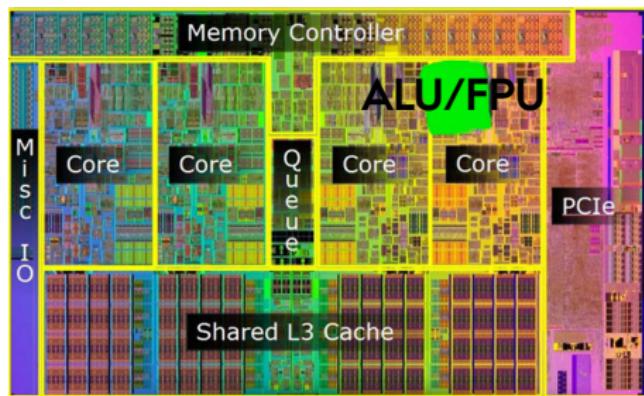
What makes GPUs such excellent HW for Machine Learning?

ALU: Arithmetic logic unit

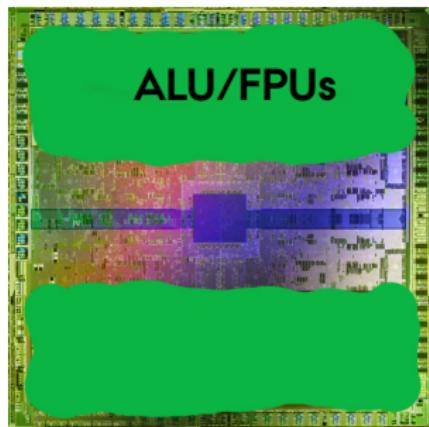
FPU: Floating-point unit

Memory Controller: six controllers on GPU,

CPU: lots of speculative execution; waste of transistors.



CPU (type?)
with one ALU/FPU marked



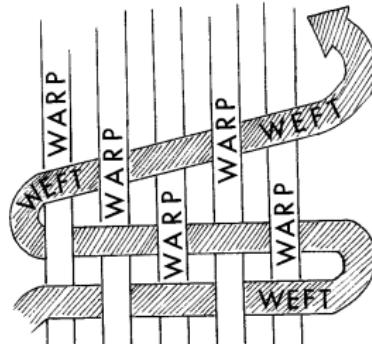
Fermi GPU
ALUs/FPUs all over

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:FPUs

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

2. generation

Tensor Cores

3. generation

Standard hukommelseskonfiguration

Hukommelsesgrænsefladens bredde

24 GB GDDR6X

384-bit

Tex

Tex

Tex

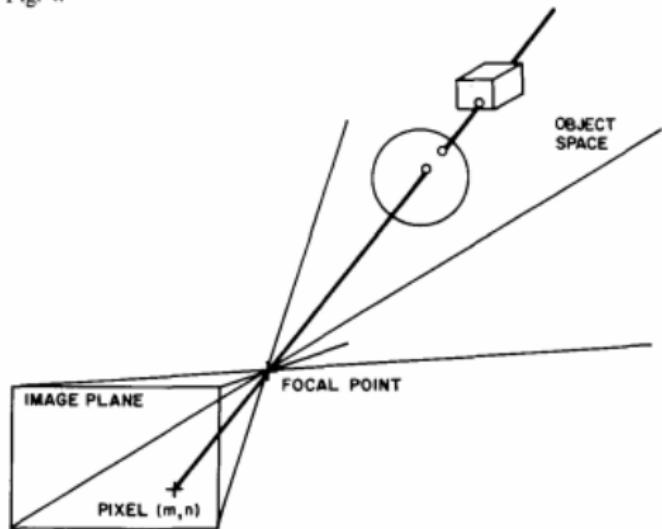
Tex

GPUs

Tensor cores: Raytracing vs Rasterization on GPUs

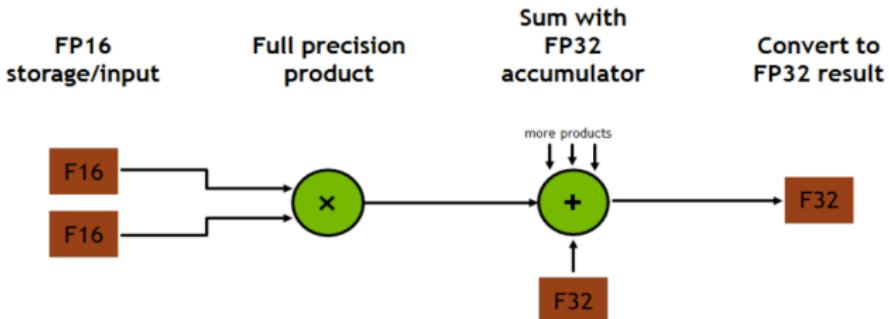


Fig. 4.



GPUs

Tensor Cores
(SKIP most)



$$D = \begin{pmatrix} A_{0,0} & A_{0,1} & A_{0,2} & A_{0,3} \\ A_{1,0} & A_{1,1} & A_{1,2} & A_{1,3} \\ A_{2,0} & A_{2,1} & A_{2,2} & A_{2,3} \\ A_{3,0} & A_{3,1} & A_{3,2} & A_{3,3} \end{pmatrix}_{\text{FP16 or FP32}} \begin{pmatrix} B_{0,0} & B_{0,1} & B_{0,2} & B_{0,3} \\ B_{1,0} & B_{1,1} & B_{1,2} & B_{1,3} \\ B_{2,0} & B_{2,1} & B_{2,2} & B_{2,3} \\ B_{3,0} & B_{3,1} & B_{3,2} & B_{3,3} \end{pmatrix}_{\text{FP16}} + \begin{pmatrix} C_{0,0} & C_{0,1} & C_{0,2} & C_{0,3} \\ C_{1,0} & C_{1,1} & C_{1,2} & C_{1,3} \\ C_{2,0} & C_{2,1} & C_{2,2} & C_{2,3} \\ C_{3,0} & C_{3,1} & C_{3,2} & C_{3,3} \end{pmatrix}_{\text{FP16 or FP32}}$$

Tesla V100's Tensor Cores deliver up to **125 Tensor 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():`

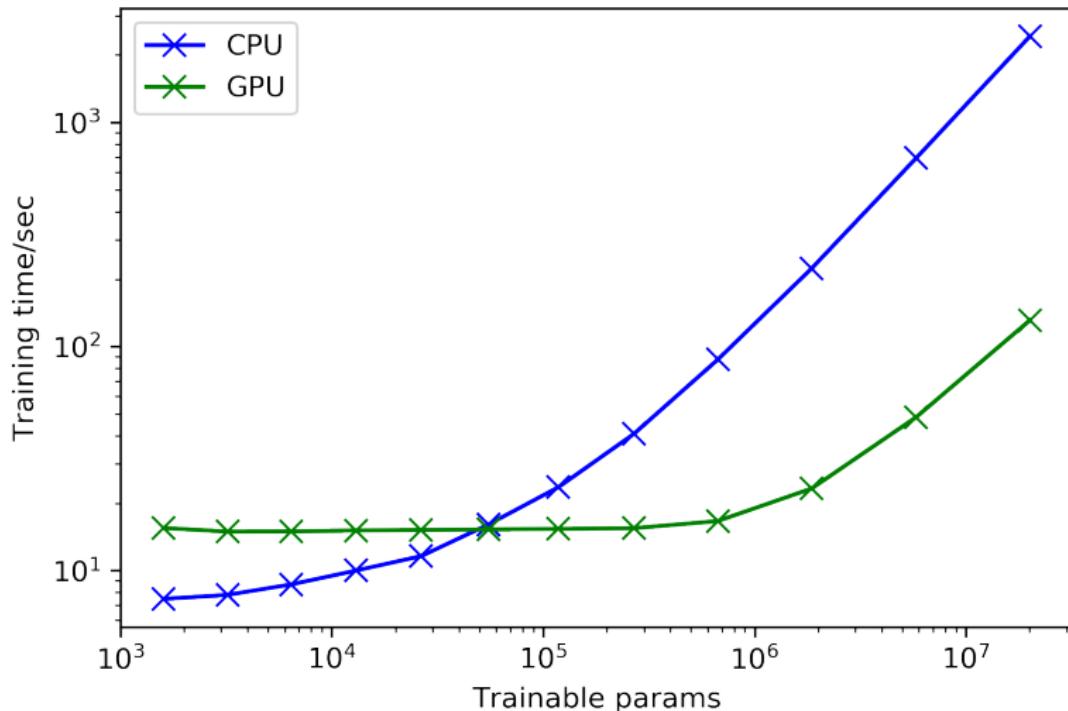
```
1 # i=12, n=4096
2 #
3 # Layer (type)          Output Shape       Param #
4 # =====
5 # dense_46 (Dense)     (None, 4096)        3215360
6 #
7 # dropout_31 (Dropout) (None, 4096)        0
8 #
9 # dense_47 (Dense)     (None, 4096)        16781312
10 #
11 # dropout_32 (Dropout) (None, 4096)        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...

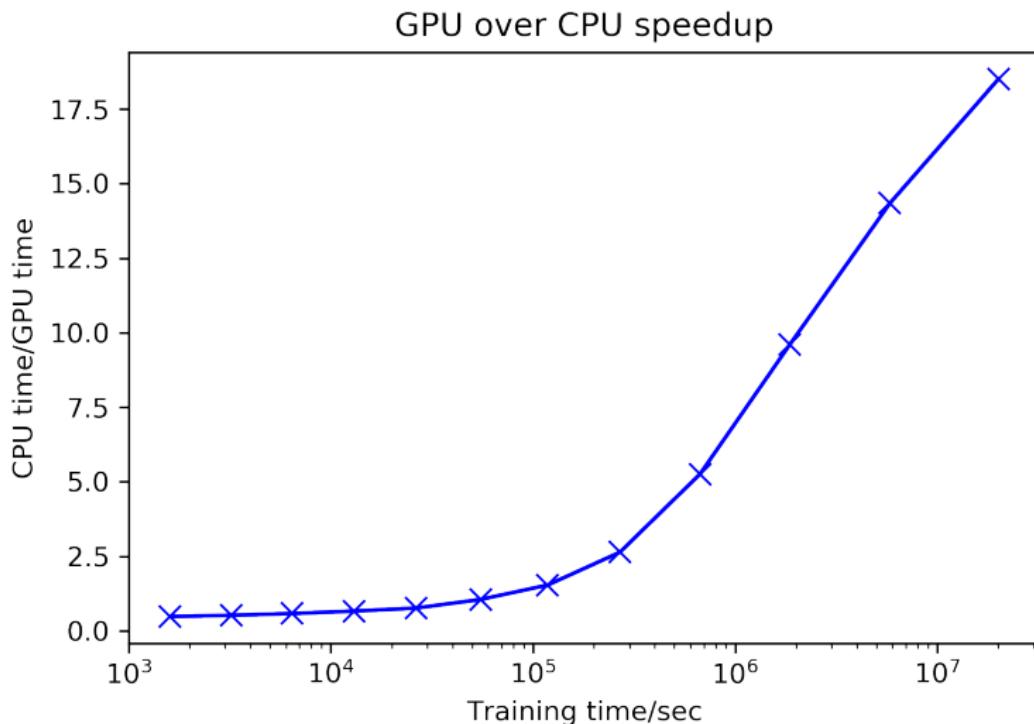
Training time-vs-Trainable params



GPUs

Actual test on the GPU-server

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



TPUs

Tensor Processing Units

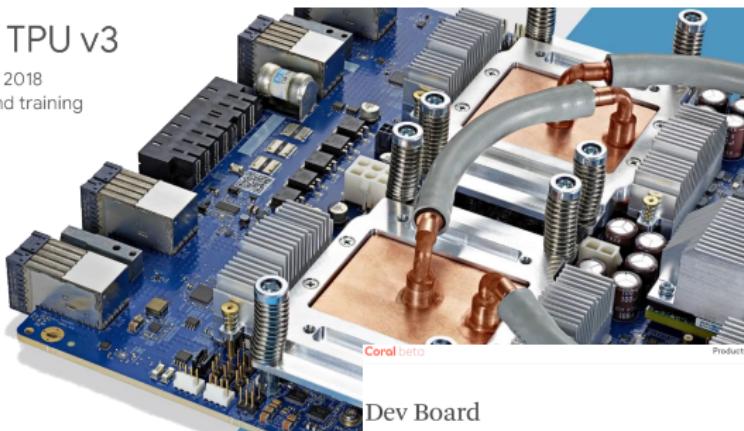
Custom ASICs by Google



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

Inference and training

Dev board with Google Edge TPU ML accelerator coprocessor

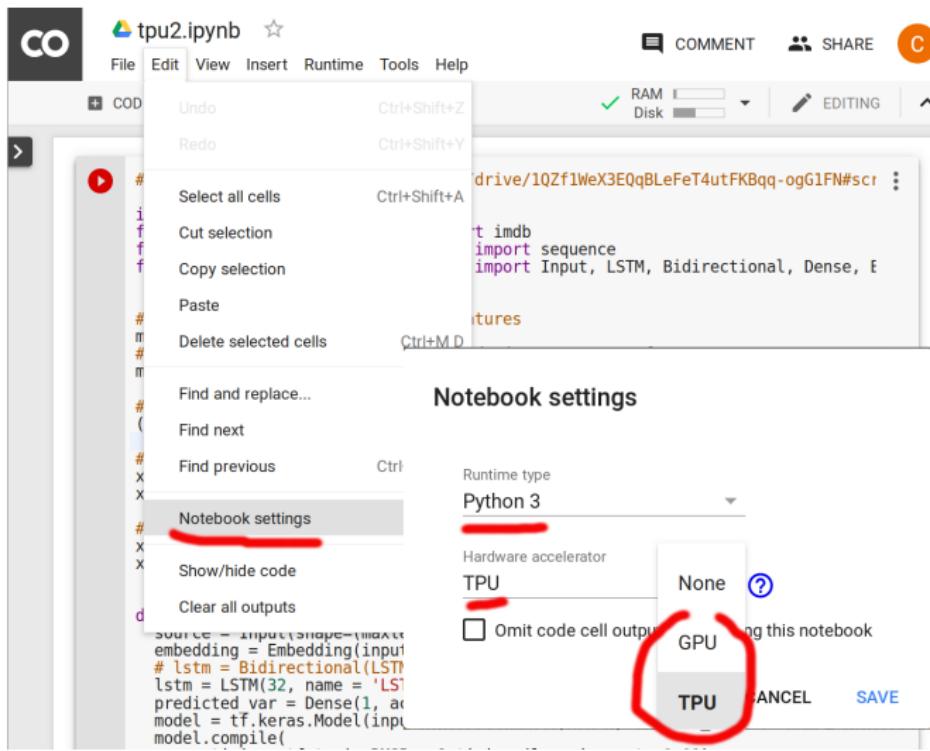
[\[https://coral.withgoogle.com/products/dev-board/\]](https://coral.withgoogle.com/products/dev-board/)



TPUs

Access to TPUs (SKIP)

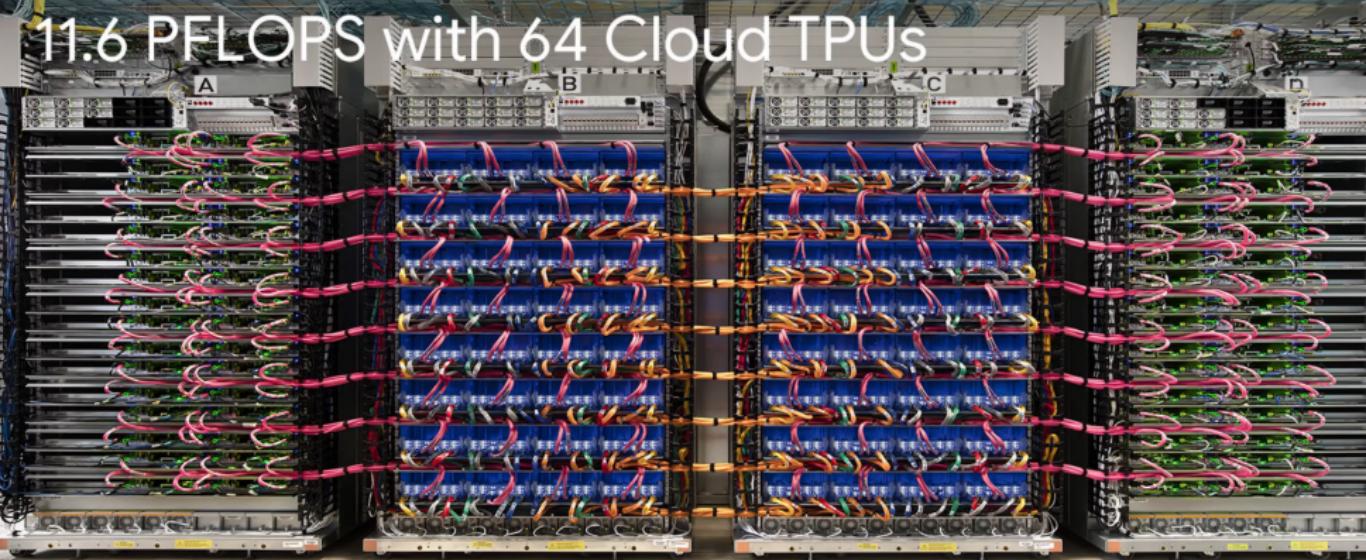
Free Jupyter Notebook environment with access to TPUs:
<https://colab.research.google.com>



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 |

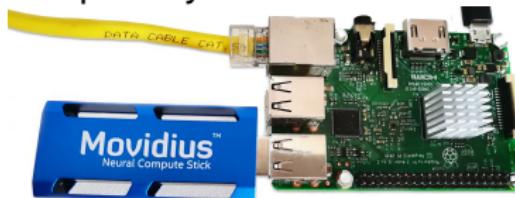
[<https://www.extremetech.com/computing/247403-nvidia-claims-pascal-gpus-challenge-googles-tensorflow-tpu-updated-benchmarks>]

Exotic Hardware

- ▶ Intel Phi multicore CPU, 64 i386 cores:



- ▶ Raspberry PIs + Intel Movidius stick



- ▶ FPGAs

