Deep learning, Class 1

Anders Eklund

anders.eklund@liu.se

Department of Biomedical Engineering (IMT)
Department of Computer and Information Science (IDA)
Center for Medical Image Science and Visualization (CMIV)
Linköping University, Sweden

April 11, 2023

Outline

- Practical methodology
  - Deep learning frameworks
  - GPU hardware & acceleration
  - Setting up an Anaconda environment
  - Docker containers & Singularity
  - Estimating network uncertainty

Disclaimer

- This class covers some topics that will not be used in the course
- These topics are important to know about
  if you work with statistics & machine learning,
  and especially if you work with big data

Deep learning frameworks

- The most common deep learning frameworks are
  - Tensorflow (low level programming, more common for production)
  - Keras (high level programming, easier to learn)
  - Pytorch (low level programming, more common for research)
  - ...

Deep learning backends

- The backend performs the actual work / calculations
- Was possible to use Keras with different backends, today only Tensorflow?
  - Tensorflow
  - Microsoft CNTK
  - Amazon MXNet
  - Theano
  - ...
- Training time and accuracy differ between backends

GPU HARDWARE & ACCELERATION

- Virtually all deep learning (training) requires GPU acceleration
- DL frameworks can run on CPU, but will be slow
- Built-in support for Nvidia GPU acceleration in most DL frameworks
- Speedup for convolution? Probably 10 - 50 times compared to CPU
- AMD GPU acceleration? ROCm, HipCaffe

Why are graphics cards so powerful?

- Graphics cards are normally used for demanding computer games
- The gamers require more and more advanced computer graphics
- Today, graphics cards can be used for arbitrary calculations (physics simulations, image processing, statistics, machine learning)

1986

2016

WHY GPU COMPUTING?

CPU memory (RAM) 4 - 128 GB

Fast memory

CPU

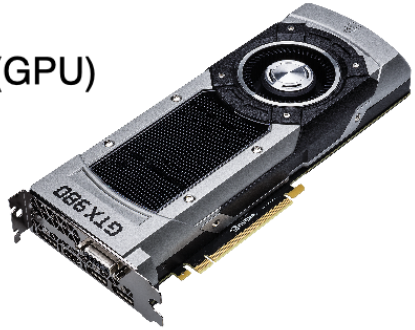
4 - 20 cores

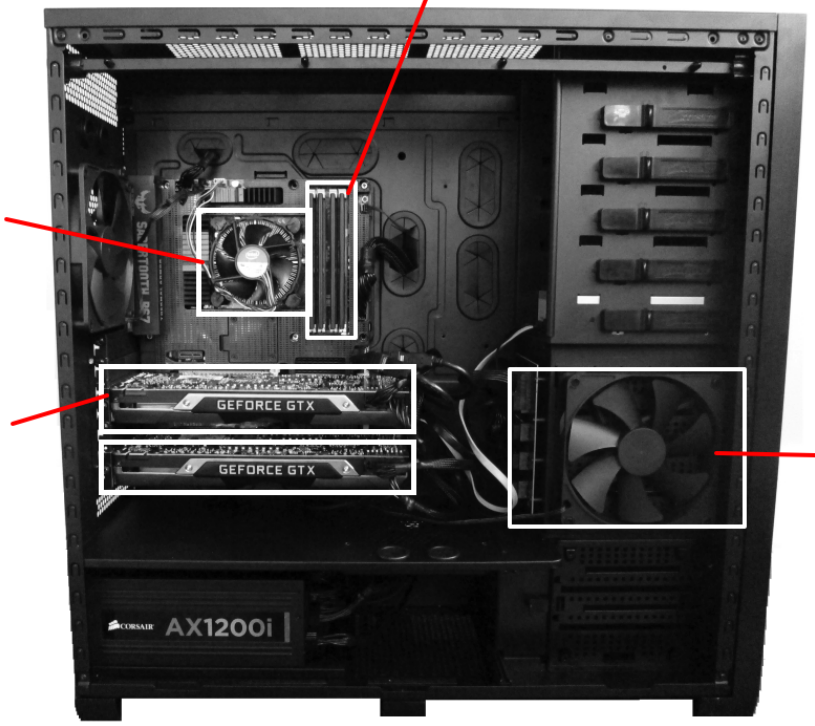
Graphics card (GPU)

2048 cores

2 - 32 GB

Superfast memory



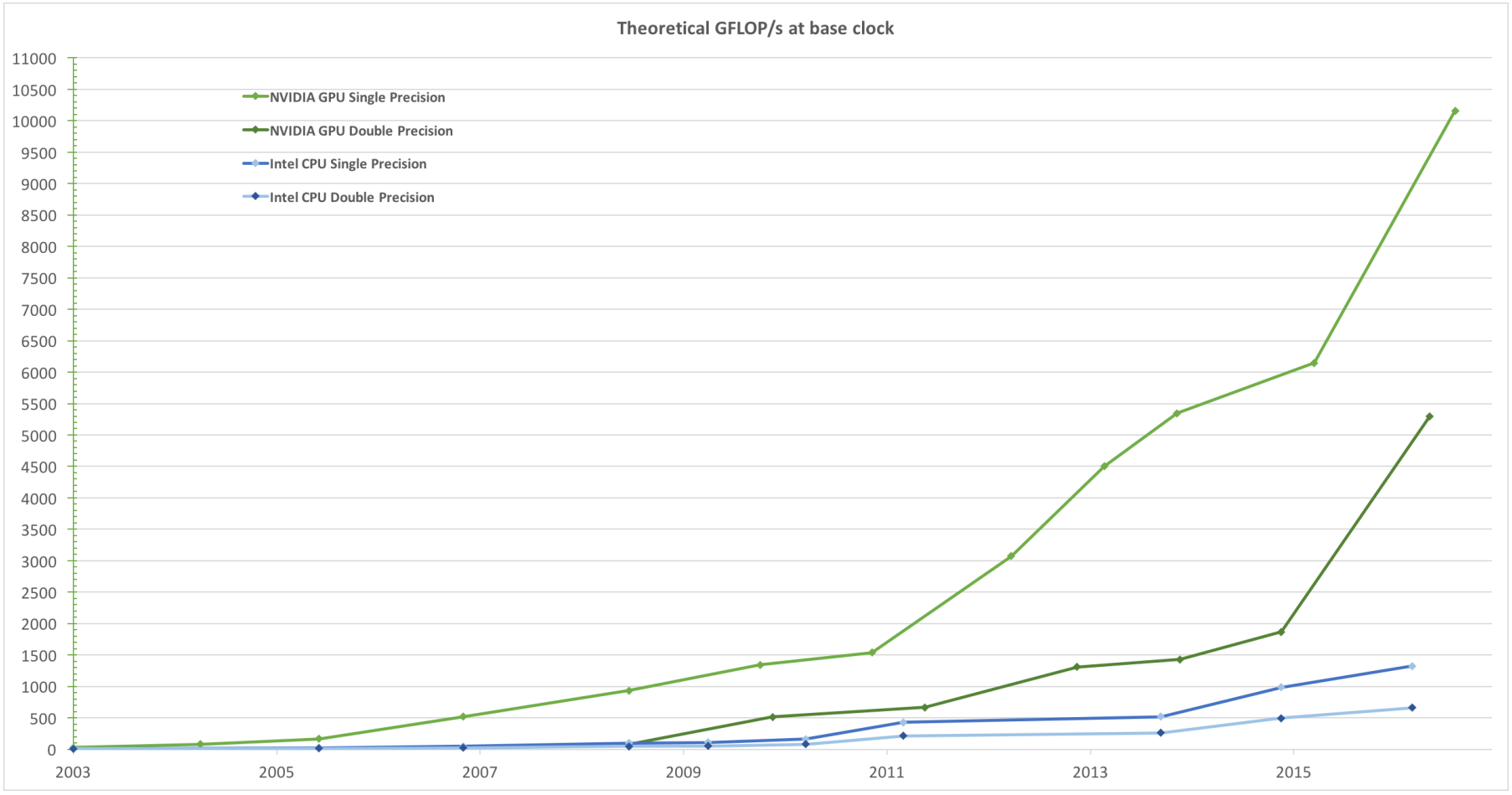


Hard drive

(2 - 20 TB)

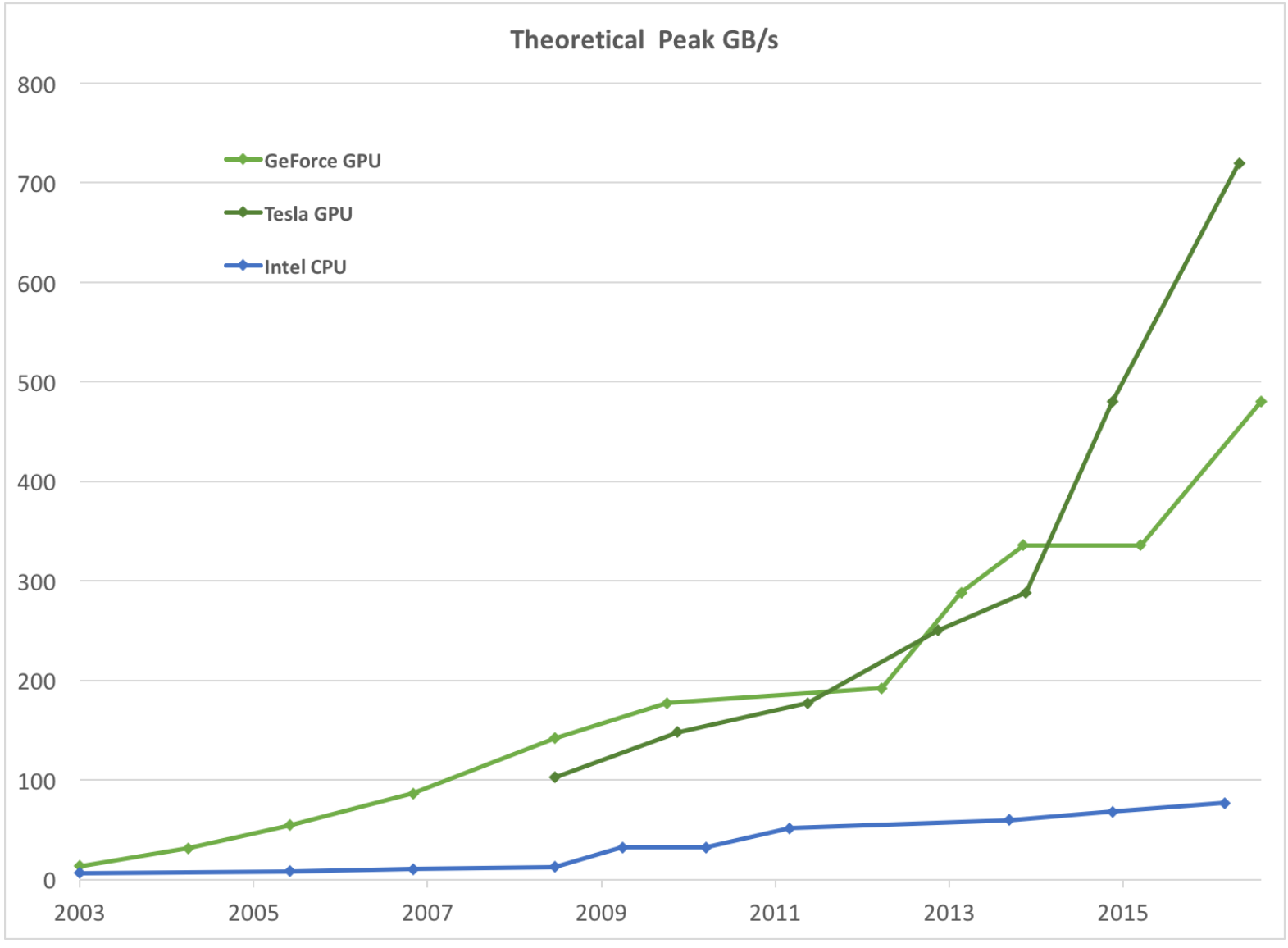
Slow memory

WHY GPU COMPUTING?



https://docs.nvidia.com/cuda/archive/10.0/cuda-c-programming-guide/index.html

WHY GPUS? MEMORY BANDWIDTH



CONSUMER GRAPHICS CARDS VS PROFESSIONAL CARDS

- Consumer graphics cards: e.g. Geforce GTX
  - Cheap (500 - 1000 USD)
  - Great performance for floats (single precision)
  - Not as good performance for double precision (1/2 to 1/24 compared to floats)
- Professional graphics cards: e.g. Tesla, Quadro (?)
  - Expensive ($>$ 5000 USD)
  - Great performance for double precision
  - ECC support (error correcting memory)
  - Bigger memory (up to 80 GB)

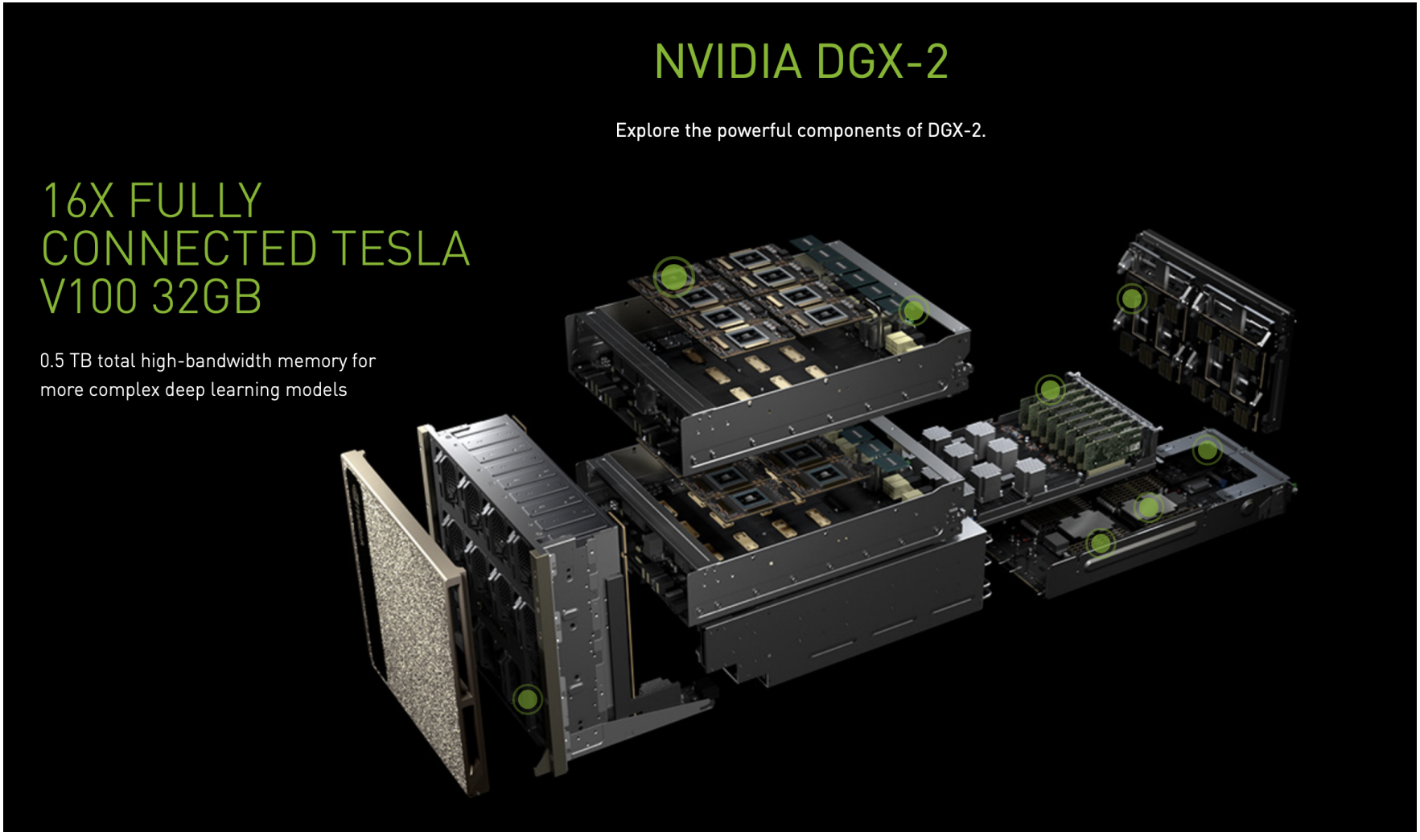
Example - 3090 card

- 10,496 processor cores, about 1400 MHz
- A GPU core cannot be compared to a CPU core
- Can run 40 000 - 50 000 threads in parallel
- 24 GB memory
- Memory bandwidth 936 GB / second
- About 15 000 - 20 000 SEK

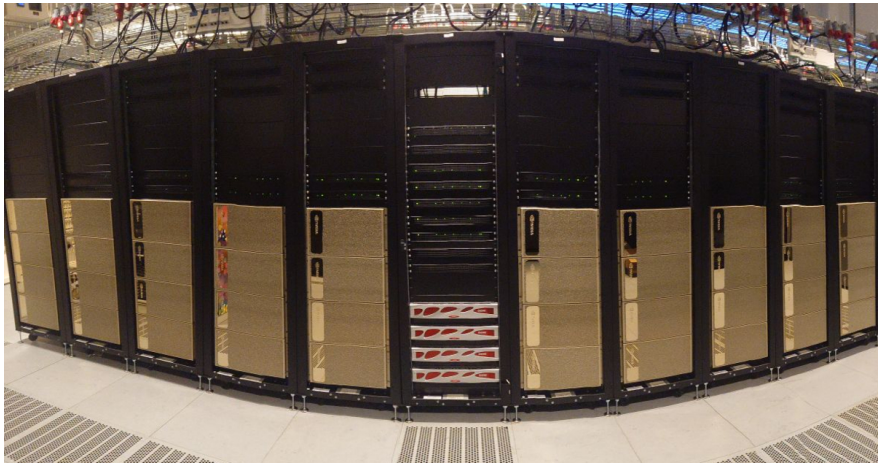
Example - A100 card

- Professional card
- 6912 processor cores
- 40 - 80 GB memory, ECC support
- Memory bandwidth 1500 - 2000 GB / second
- About 100 000 SEK

EXAMPLE - DGX-2 (16 X TESLA V100)



Example - Berzelius

- https://www.nsc.liu.se/systems/berzelius/
- 60 nodes
- Each node has
  - 1 TB RAM
  - 128 CPU cores
  - 8 Nvidia A100 graphics cards (40 GB)
- In total 60 * 8 = 480 graphics cards
- 

GPUs for deep learning

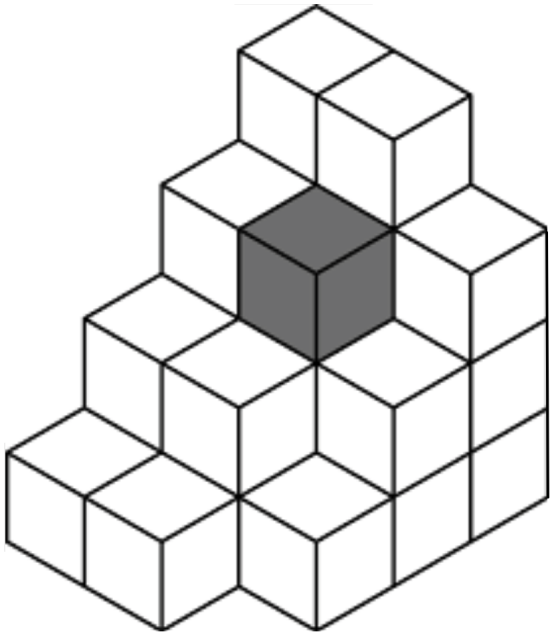
- How do we know how hard the GPU is working?
- nvidia-smi in terminal
- If the GPU is working less than 95%
  - Increase batch size (if there is enough memory)
  - Optimize code
  - Buy more RAM to avoid streaming from hard drive
  - Use TFrecords or numpy arrays instead of standard file formats (if streaming from hard drive)
  - Use SSD instead of mechanical drive (if streaming from hard drive)

GPUs FOR DEEP LEARNING

```
(base) lnx00193 [~]> nvidia-smi
Sun Apr  9 21:06:53 2023
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 525.105.17   Driver Version: 525.105.17   CUDA Version: 12.0     |
|-------------------------------+----------------------+----------------------+
| GPU  Name        Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Compute M. |
|                               |                      |               MIG M. |
|===============================+======================+======================|
|   0  NVIDIA GeForce ...  Off  | 00000000:01:00.0 Off |                  N/A |
| 61%   58C    P2   295W / 350W |  22950MiB / 24576MiB |    100%      Default |
|                               |                      |                  N/A |
+-------------------------------+----------------------+----------------------+

+-----------------------------------------------------------------------------+
| Processes:                                                                  |
|  GPU   GI   CI        PID   Type   Process name                  GPU Memory |
|        ID   ID                                                   Usage      |
|=============================================================================|
|    0   N/A  N/A      2037      G   /usr/bin/X                        1225MiB |
|    0   N/A  N/A      2744      G   /usr/bin/gnome-shell                33MiB |
|    0   N/A  N/A      3429      G   ...AAAAAAAAA= --shared-files        36MiB |
|    0   N/A  N/A      4139      C   python                           21636MiB |
|    0   N/A  N/A     13982      G   /usr/lib64/firefox/firefox          11MiB |
+-----------------------------------------------------------------------------+
```

How can GPUs be used for arbitrary calculations?



Voxel

- In computer games, the graphics card is used to render all images
- For a resolution of 2560 x 1440 pixels and 60 frames per second, the graphics card needs to render 221,184,000 pixels per second
- Medical volume data is for example 512 x 512 x 512 voxels (3D pixels), the graphics card can do calculations in parallel instead of rendering
- Write code for analyzing one pixel / voxel, graphics card does the rest, launches one thread per pixel

GPU PROGRAMMING - BEFORE CUDA

- Perform calculations through computer graphics languages
- OpenGL, Open Graphics Language
- Direct X
- HLSL, High Level Shading Language
- Cg, C for graphics
- GLSL, OpenGL Shading Language
- Hard to optimize performance and debug code

GPU PROGRAMMING - PRESENT

- CUDA released in 2007
- Compute Unified Device Architecture
- C / C++ programming of GPUs
- Possible to debug GPU code as regular C / C++ code
- Possible to improve performance by using tools like the Nvidia profiler

OpenCL

- OpenCL, Open Computing Language (CPUs, GPUs, FPGAs, ...)
- CUDA only works for Nvidia GPUs
- For AMD GPUs you can use OpenCL
- OpenCL can be used for any hardware with an OpenCL driver
- For Intel CPUs, the OpenCL compiler will vectorize the code
  (run on several threads, and each thread works on several data points)

CUDA LIBRARIES

- Optimized libraries by Nvidia
- CUBLAS, dense matrix vector operations
- CUSPARSE, sparse matrix vector operations
- CUFFT, fast Fourier transform
- CURAND, random numbers
- CUDNN, deep neural network,
  called by Tensorflow / Pytorch / Theano ...

Conjugate gradient using cublas and cusparse

- Example from research
- Sparse equation system of size 2,200,000 x 2,200,000
- 75 000 000 non-zero elements
- CUDA version of PCG 13 times faster than Matlab,
  1 minute saved per call to PCG,
  1000 calls = saved 17 hours

NON-PARAMETRIC PERMUTATION TESTS

- Do fMRI group analysis with a non-parametric permutation test (to correct for multiple comparisons over 50,000 voxels)
- BROCCOLI: 1,000 permutations in about 3-4 seconds
- FSL: 4 minutes on single CPU core (can use several cores)
- Used in "cluster failure" paper to run 384,000 permutation tests in 25 days (would have taken 5 - 10 years in FSL)
- Eklund, A., Nichols, T. E., & Knutsson, H. (2016). Cluster failure: Why fMRI inferences for spatial extent have inflated false-positive rates. Proceedings of the national academy of sciences, 113(28), 7900-7905.

PyCUDA & PyOpenCL

- Call GPU code from Python
- https://pypi.org/project/pycuda/
- https://pypi.org/project/pyopencl/
- Klöckner, A., Pinto, N., Lee, Y., Catanzaro, B., Ivanov, P., & Fasih, A. (2012). PyCUDA and PyOpenCL: A scripting-based approach to GPU run-time code generation. Parallel Computing, 38(3), 157-174.

PYCUDA - YOUR OWN KERNEL

```
import pycuda.autoinit
import pycuda.driver as drv
import numpy

from pycuda.compiler import SourceModule
mod = SourceModule("""
__global__ void multiply_them(float *dest, float *a, float *b)
{
  const int i = threadIdx.x;
  dest[i] = a[i] * b[i];
}
""")

multiply_them = mod.get_function("multiply_them")

a = numpy.random.randn(400).astype(numpy.float32)
b = numpy.random.randn(400).astype(numpy.float32)

dest = numpy.zeros_like(a)
multiply_them(
        drv.Out(dest), drv.In(a), drv.In(b),
        block=(400,1,1), grid=(1,1))

print dest-a*b
```

CUDA PROGRAMMING

- Writing your own CUDA kernels can be fun,
  but can take a lot of time to optimize code
- Code that is optimized for one specific graphics card
  will in most cases not be optimal for another graphics card
- Fortunately there are built in functions / packages
  for many computing languages such as Python, Matlab, R
- Note: you need rather large datasets / demanding algorithms
  to see a speeedup, will probably not see any speedup for a
  100 x 100 or 1000 x 1000 matrix

SCIKIT-CUDA

- https://scikit-cuda.readthedocs.io/en/latest/generated/skcuda.linalg.dot.html

```
>>> import pycuda.autoinit
>>> import pycuda.gpuarray as gpuarray
>>> import numpy as np
>>> import skcuda.linalg as linalg
>>> import skcuda.misc as misc
>>> linalg.init()
>>> a = np.asarray(np.random.rand(4, 2), np.float32)
>>> b = np.asarray(np.random.rand(2, 2), np.float32)
>>> a_gpu = gpuarray.to_gpu(a)
>>> b_gpu = gpuarray.to_gpu(b)
>>> c_gpu = linalg.dot(a_gpu, b_gpu)
>>> np.allclose(np.dot(a, b), c_gpu.get())
True
>>> d = np.asarray(np.random.rand(5), np.float32)
>>> e = np.asarray(np.random.rand(5), np.float32)
>>> d_gpu = gpuarray.to_gpu(d)
>>> e_gpu = gpuarray.to_gpu(e)
>>> f = linalg.dot(d_gpu, e_gpu)
>>> np.allclose(np.dot(d, e), f)
True
```

-

MATLAB

- Parallel computing toolbox
- https://se.mathworks.com/discovery/matlab-gpu.html
- gpuArray converts an array in the MATLAB workspace into a gpuArray with elements stored on the GPU device
- Many functions with built in support for gpuArray
- Also possible to make your own kernels

MATLAB FUNCTIONS FOR GPUARRAY

MATLAB Functions with gpuArray Arguments

Many MATLAB® built-in functions support `gpuArray` input arguments. Whenever any of these functions is called with at least one gpuArray as an input argument, the function executes on the GPU and generates a gpuArray as the result. You can mix inputs using both gpuArray and MATLAB arrays in the same function call; the MATLAB arrays are transferred to the GPU for the function execution. Supporting functions include the discrete Fourier transform (`fft`), matrix multiplication (`mtimes`), and left matrix division (`mldivide`).

The following functions and their symbol operators are enhanced to accept gpuArray input arguments so that they execute on the GPU:

abs
acos
acosd
acosh
acot
acotd
acoth
acsc
acscd
acsch
accumarray
all
and
angle
any
arrayfun
asec
asecd
asech
asin
asind
asinh
assert
atan
atan2
atan2d
atand
atanh
bandwidth
besselj
bessely
beta
betainc
betaincinv
betaln
bicg
bicgstab
bitand
bitcmp
bitget
bitor
bitset
bitshift
bitxor
blkdiag
bounds
bsxfun
cart2pol
cart2sph
cast
cat
cdf2rdf
ceil
cgs
chol
circshift
classUnderlying
colon
compan
complex
cond
conj
conv
conv2
convn
corrcoef
cos
cosd
cosh
cot
cotd
coth
cov
cross
csc
cscd
csch
ctranspose
cummax
cummin
cumprod
cumsum
deg2rad
del2
det
detrend
diag
diff
discretize
disp
display
dot
double
eig
eps
eq
erf
erfc
erfcinv
erfcx
erfinv
exp
expint
expm
expm1
eye
factorial
false
fft
fft2
fftn
fftshift
filter
filter2
find
fix
flip
fliplr
flipud
floor
fprintf
full
gamma
gammainc
gammaincinv
gammaln
gather
ge
gmres
gradient
gt
hankel
head
histcounts
horzcat
hsv2rgb
hypot
idivide
ifft
ifft2
ifftn
ifftshift
imag
ind2sub
Inf
inpolygon
int16
int2str
int32
int64
int8
interp1
interp2
interp3
interpn
intersect
inv
ipermute
isaUnderlying
isbanded
iscolumn
isdiag
isempty
isequal
isequaln
isfinite
isfloat
ishermitian
isinf
isinteger
islogical
ismatrix
ismember
ismembertol
isnan
isnumeric
isreal
isrow
issorted
issparse
issymmetric
istril
istriu
isvector
kron
ldivide
le
legendre
length
log
log10
log1p
log2
logical
lsqr
lt
lu
mat2str
max
median
mean
meshgrid
min
minus
mldivide
mod
mode
movmean
movstd
movsum
movvar
mpower
mrdivide
mtimes
NaN
ndgrid
ndims
ne
nextpow2
nnz
nonzeros
norm
normest
not
nthroot
null
num2str
numel
ones
or
orth
pagefun
pcg
perms
permute
pinv
planerot
plot (and related)
plus
pol2cart
poly
polyarea
polyder
polyfit
polyint
polyval
polyvalm
pow2
power
prod
psi
qmr
qr
rad2deg
rand
randi
randn
randperm
rank
rdivide
real
reallog
realpow
realsqrt
rectint
rem
repelem
repmat
rescale
reshape
rgb2hsv
roots
rot90
round
sec
secd
sech
setdiff
setxor
shiftdim
sign
sin
sind
single
sinh
size
sort
sortrows
svds
spconvert
spdiags
sph2cart
sprand
sprandn
sprandsym
spconvert
sph2cart
sprand
sprandn
sprandsym
sprintf
sqrt
squeeze
std
sub2ind
subsasgn
subsindex
subspace
subsref
sum
superiorfloat
svd
svds
swapbytes
tail
tan
tand
tanh
times
toeplitz
trace
transpose
trapz
tril
triu
true
typecast
uint16
uint32
uint64
uint8
uminus
union
unique
uniquetol
unwrap
uplus
vander
var
vertcat
xor
zeros

MATLAB FUNCTIONS FOR GPUARRAY

- a = randn(1000,1000);
- b = randn(1000,1000);
- a_GPU = gpuArray(a); % Copy to GPU
- b_GPU = gpuArray(b); % Copy to GPU
- c_GPU = a_GPU * b_GPU; % Matrix multiplication on GPU
- c = gather(c_GPU); % Copy back to CPU

GPU COMPUTING WITH R

- http://www.r-tutor.com/gpu-computing, rpud package
  http://www.r-tutor.com/gpu-computing/clustering/distance-matrix

The following measures the time spent on finding the distance matrix for a collection of 4,500 random vectors in a 120 dimension space. On a desktop computer with AMD Phenom II X4 CPU, it takes about 14 seconds to finish.

```
> test.data <- function(dim, num, seed=17) {
+     set.seed(seed)
+     matrix(rnorm(dim * num), nrow=num)
+ }
> m <- test.data(120, 4500)
> system.time(dist(m))
   user  system elapsed
 14.343   0.185  14.533
```

Now load the rpud package, and compute the same distance matrix using the rpuDist method. For rpud running on NVIDIA GTX 460 GPU, the execution time is about 1 second. Even better, with the rpudplus add-on, you can compute distance matrices for larger data sets that fit inside the system RAM available to R.

```
> library(rpud)                  # load rpud with rpudplus
> system.time(rpuDist(m))
   user  system elapsed
  0.674   0.305   0.980
```

Anaconda

- Easy to setup deep learning environment, https://www.anaconda.com/distribution/
- conda create -n mykeras python=3.7
- conda activate mykeras
- conda install keras-gpu
- Each install command guarantees the correct versions (CUDA driver, CUDA toolkit, CUDNN, Python, ...)
- Can easily switch between different environments

Operating systems

- What happens if you setup a Python / Anaconda environment in Linux and try to use the same nvironment in Windows?

Operating systems

- What happens if you setup a Python / Anaconda environment in Linux and try to use the same environment in Windows?
- The same Python libraries do not exist in Windows, Mac, Linux...

Docker containers

- Versions of Python, Tensorflow, graphics drivers, CUDNN, must match
- May get different results for different operating systems
- After upgrading the OS, things may stop working...
- Maybe need to test something 2 years later, with the same setup
- This is simplified with Docker containers, small virtual machines
- A Docker container can be setup with a text file (Dockerfile)
- You can also send a built container to someone
- www.docker.com

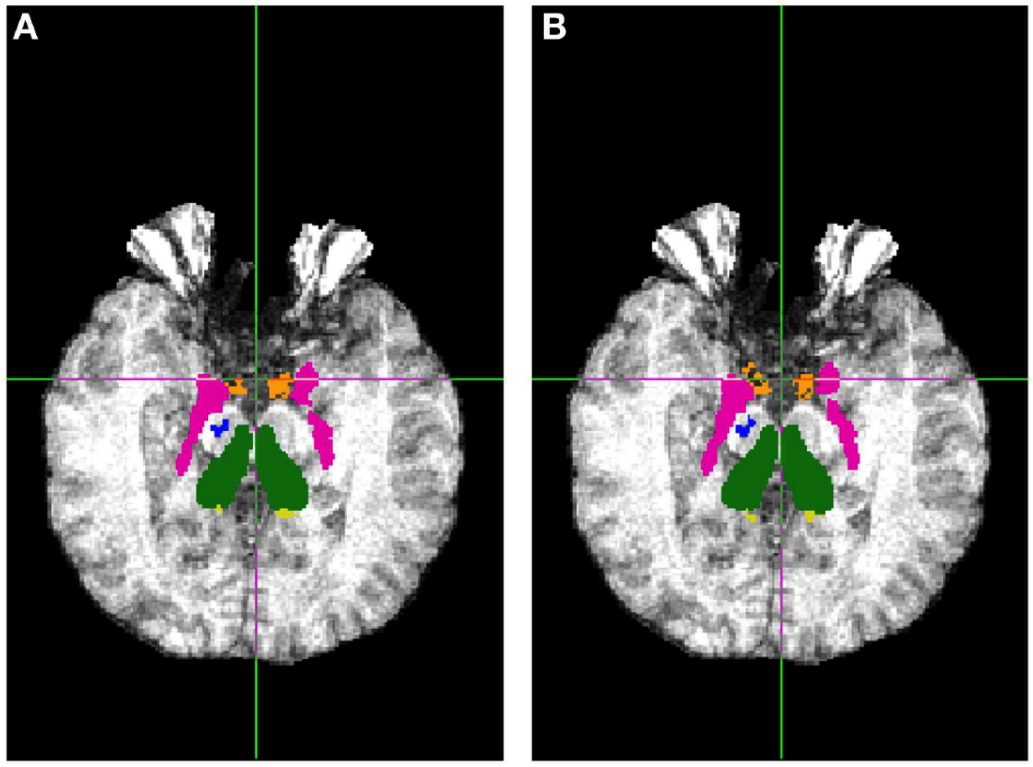
Reproducibility - example

- Glatard et al., Reproducibility of neuroimaging analyses across operating systems, Frontiers in Neuroinformatics, 2015

- Tested the softwares FSL, Freesurfer and CIVET, on two different clusters running CentOS 5.10 and Fedora 20

- "With FSL, most Dice coefficients between subcortical classifications obtained on different operating systems remain above 0.9, but values as low as 0.59 are observed. Independent component analyses (ICA) of fMRI data differ between operating systems in one third of the tested subjects, due to differences in motion correction. With Freesurfer and CIVET, in some brain regions we find an effect of build or operating system on cortical thickness."

REPRODUCIBILITY - EXAMPLE



Cluster **A** Cluster **B**

Structure	Color	Dice
L. and R. putamen	Magenta	0.92 and 0.95
R. pallidum	Dark blue	0.93
L. and R. thalamus	Green	0.97 and 0.93
L. and R. accumbens area	Orange	0.75 and 0.59
L. and R. hippocampus	Yellow	0.92 and 0.77

FIGURE 5 | Sample subcortical classifications with FSL FIRST: subject 260, $Z = 114$.

Docker containers

- Run CentOS 6.8 container on your machine

  docker run -it --rm centos:6.8

- Run CentOS 7 container on your machine

  docker run -it --rm centos:7

- Run Ubuntu 12

  docker run -it --rm ubuntu:12.04

- Run Ubuntu 14

  docker run -it --rm ubuntu:14.04

- The first time the container will be downloaded from docker hub (hub.docker.com)

Docker files

- Dockerfiles define how to setup a container
- A text file that can easily be distributed and built on any computer
- Example
- FROM centos:6.8

  RUN yum -y install wget && \
  yum -y install git && \
  yum -y install gcc-c++
- Need to install all dependencies, nothing included as default

Running a Docker container

- Built a container called 'nnimage-gpu'
- docker run --rm --gpus <all,0,1,...> -v /path/to/input/images:/in -v /path/to/output/segmentations:/out nnimage-gpu
- Setup Anaconda inside the container
- Runs segmentation on volumes in input path (nifti files), with GPU acceleration, saves results in output path
- Full example at https://github.com/wanderine/nnunetdocker

Docker containers - performance

- Why containers?
- Some performance optimizations are only available through containers
- Mixed precision - a mix of 16 and 32 bit floats for higher performance
- Improved performance for 3D convolutions in CUDNN
- Nvidia makes improvements, quickly released as container, integration with Tensorflow may take a much longer time
- ngc.nvidia.com

Singularity

- Docker containers typically require root access
- Several compute clusters do not allow Docker
- A Docker container can be converted into Singularity, which does not require root access
- Required for LiU computers and Berzelius
- Kurtzer, G. M., Sochat, V., & Bauer, M. W. (2017). Singularity: Scientific containers for mobility of compute. PloS one, 12(5), e0177459.

Comparing networks

- Network A: performance on test data is 92%
- Network B: performance on test data is 94%
- Which network is best?

Comparing networks

- Without knowing the uncertainty, hard to pick a network!
- Estimate uncertainty using cross-validation or dropout
- Network A: 92% $\pm$ 2%, Network B: 94% $\pm$ 4%
- Cannot really say that one network is better,
  confidence intervals overlap
- If performance is similar, pick network with lowest uncertainty

Uncertainty through cross validation

- Cross-validation can be used to estimate uncertainty
- Split the data into different folds and perform training + testing, fold 1: 74%, fold 2: 82%, fold 3: 68%, fold 4: 72%
- In deep learning running 10 folds takes 20 days
  if training the network takes 2 days...

Uncertainty through dropout

- Use dropout during testing, run the same data through the network N times, every result will be slightly different
- Much more efficient compared to re-training the network
- Monte Carlo dropout, uncertainty = standard deviation over N results
- Gal, Y., & Ghahramani, Z. (2016). Dropout as a Bayesian approximation: Representing model uncertainty in deep learning. In international conference on machine learning (pp. 1050-1059).

Uncertainty through dropout

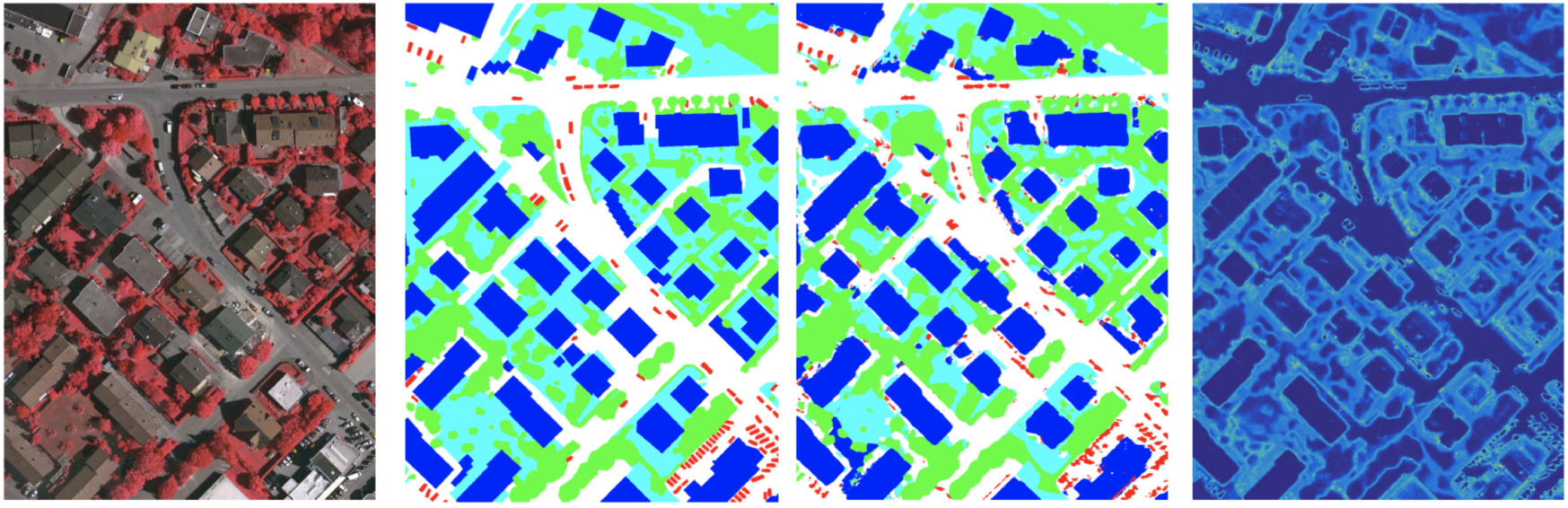


Figure 3: Results of the patch-based CNN. From left to right: One of the validation images, its ground truth, the results for the patch-based CNN and the uncertainty map.

Kampffmeyer et al. (2016). Semantic segmentation of small objects and modeling of uncertainty in urban remote sensing images using deep convolutional neural networks. IEEE conference on computer vision and pattern recognition workshops