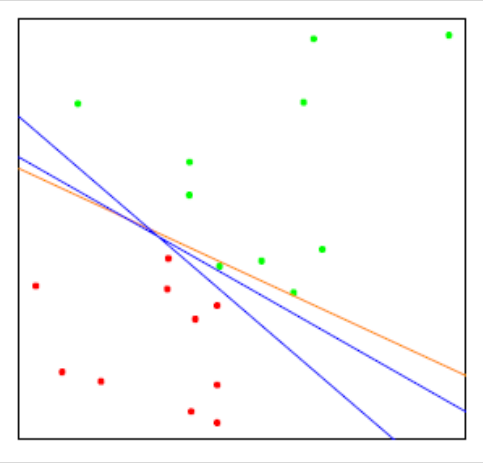


Deep Learning

Using a Convolutional Neural Network

Dr. – Ing. Morris Riedel
Adjunct Associated Professor
School of Engineering and Natural Sciences, University of Iceland
Research Group Leader, Juelich Supercomputing Centre, Germany

LECTURE 3

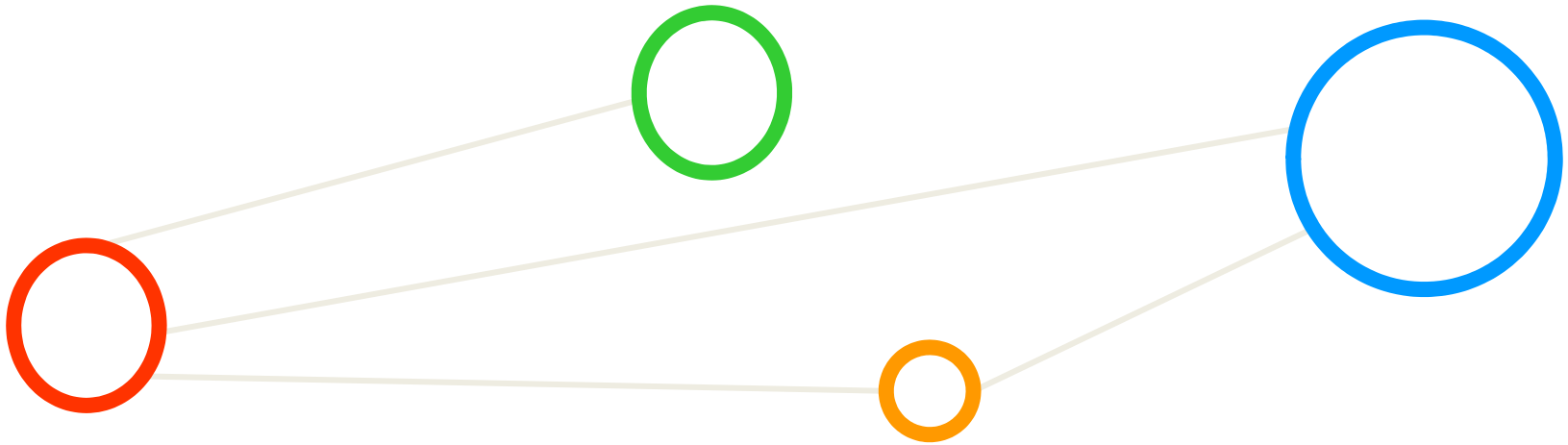
Convolutional Neural Networks Applications

November 30th, 2017
Ghent, Belgium

UNIVERSITY OF ICELAND
SCHOOL OF ENGINEERING AND NATURAL SCIENCES
FACULTY OF INDUSTRIAL ENGINEERING, MECHANICAL ENGINEERING AND COMPUTER SCIENCE

JÜLICH
FORSCHUNGSZENTRUM

Outline



Outline of the Course

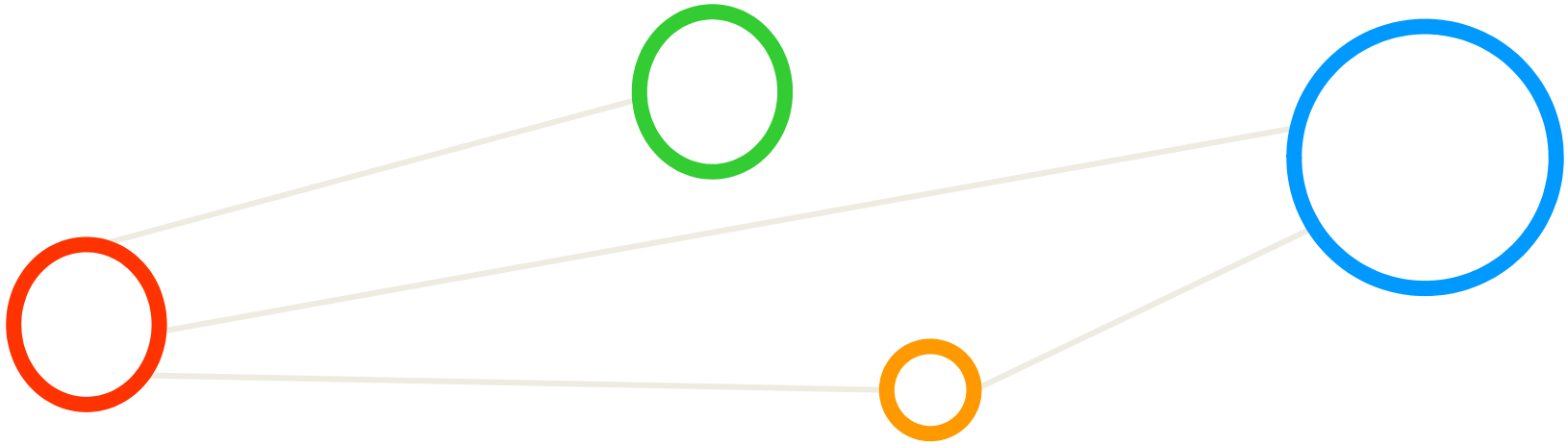
1. Deep Learning Fundamentals & GPGPUs
2. Convolutional Neural Networks & Tools
3. Convolutional Neural Network Applications
4. Convolutional Neural Network Challenges
5. Transfer Learning Technique
6. Other Deep Learning Models & Summary

Outline

- Remote Sensing Applications
  - Introduction to Application Domain
  - Rome and Indian Pines Dataset
  - Traditional Classifier Accuracy using SVMs
  - LibSVM Format & Cross-Validation
  - Indian Pines Dataset in HDF5 Format

- CNN Architecture for Application
  - Window Tensor Approach
  - Job Submission & Scripts on GPUs
  - CNN ‘Standard Model‘ in Keras
  - Experimental Setup & CNN Parameters
  - Selected Results & Parameter Changes

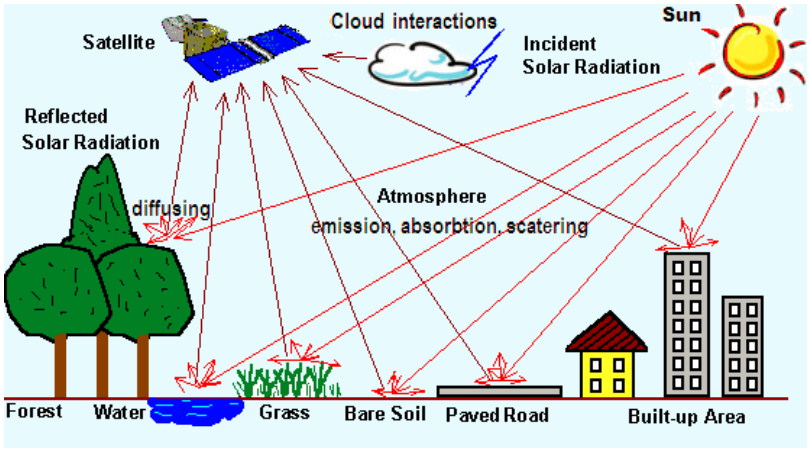
Remote Sensing Application Domain

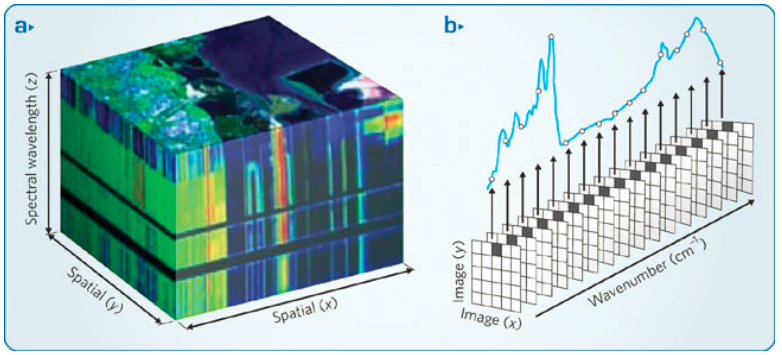


Introduction to Application Field

- **Remote sensing is the acquisition of information about an object or phenomenon without making physical contact with an object**

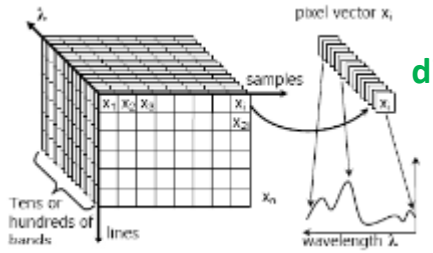
[1] Wikipedia on 'Remote Sensing'





- The overall system is complex:
  - Scattering or emission of energy from the earth's surface
  - Transmission through the atmosphere to instruments mounted on the remote sensing platform
  - Sending data back to the earth's surface
  - Processing into images products ready for application by the user

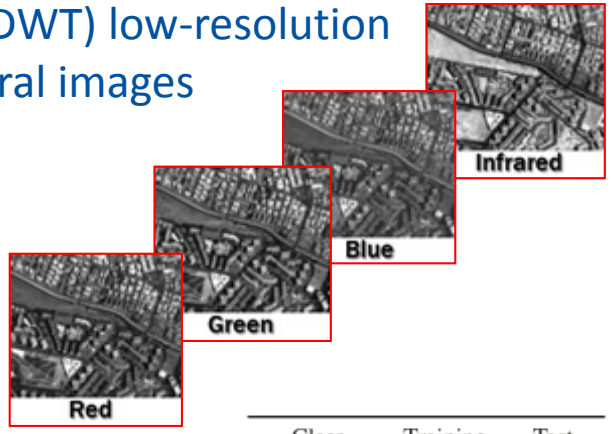
Supervised Learning Application – Labelled Data

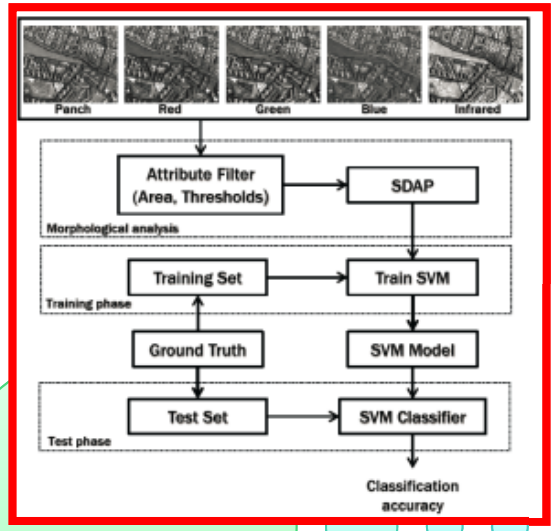


(high dimension)

Pansharpened (UDWT) low-resolution (2.4m) multispectral images

(Quickbird)





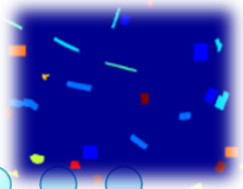
Class	Training	Test
Buildings	18126	163129
Blocks	10982	98834
Roads	16353	147176
Light Train	1606	14454
Vegetation	6962	62655
Trees	9088	81792
Bare Soil	8127	73144
Soil	1506	13551
Tower	4792	43124
Total	77542	697859

Sattelite Data

Groundtruth

Model & Algorithm

Classification Study of Land Cover Types



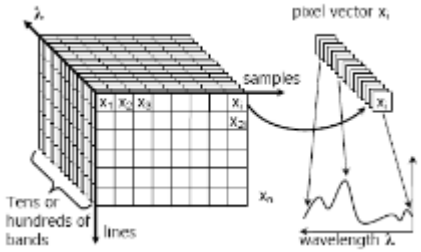
[2] G. Cavallaro & M. Riedel et al., 2014

Remote Sensing Application – The Dataset

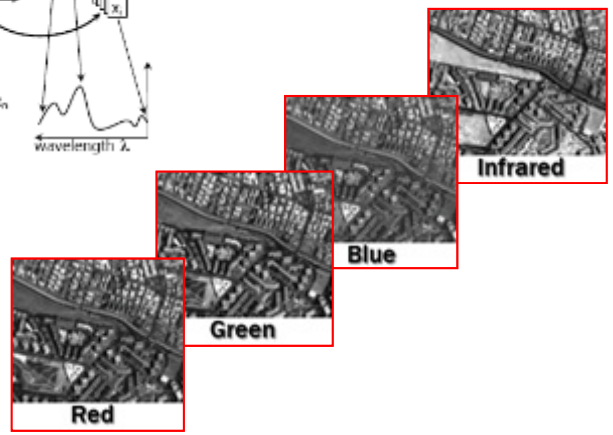
- Example dataset: Geographical location: Image of Rome, Italy
  - Remote sensor data obtained by Quickbird satellite

- High-resolution (0.6m) panchromatic image

Pansharpened (UDWT) low-resolution (2.4m) multispectral images



(high dimension)



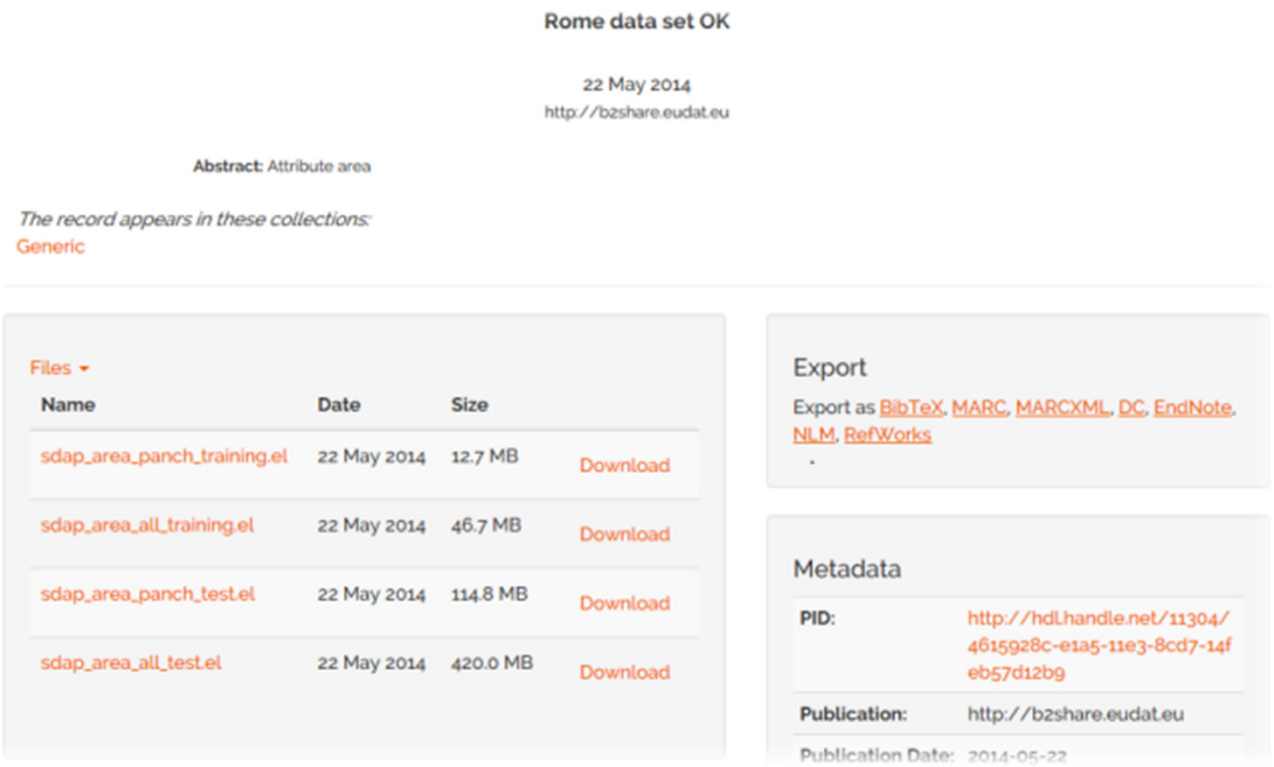
(Reasoning for picking SVM: Good classification accuracies on high dimensional datasets, even with a small ‚rare' number of training samples)

[3] Rome Image dataset

Inspect and Understanding the Data – Rome, Italy

- Data is publicly available in EUDAT B2SHARE tool

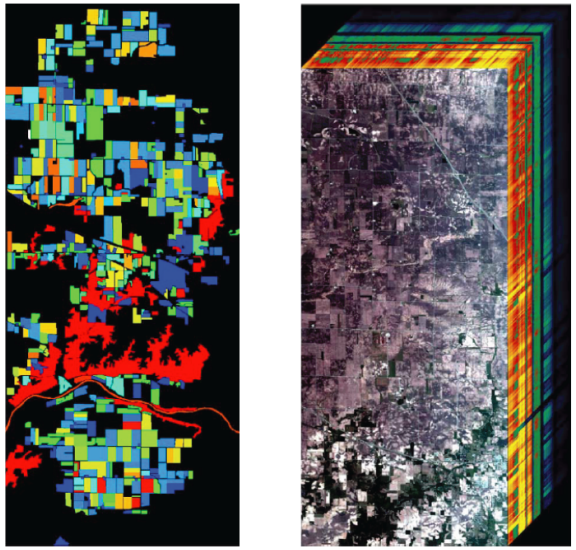
[3] Rome Image dataset



(persistent handle link for publication into papers)

Remote Sensing Dataset – Indian Pines – Metadata

- Relevant Metadata
  - Image itself was taken by Nasa's Airborne Visible/Infrared Imaging Spectrometer (AVIRIS) on June 12, 1992
  - Covers a mostly agricultural and wooded area in the west of the Purdue University in Indiana (USA)
  - The image consists of 1417 × 617 pixels with a spatial resolution of around 20 m
  - For each pixel the image provides 220 spectral channels covering wavelengths in the range of 0.4 μm to 2.5 μm (i.e. 'cube')
  - The pixel-wise labelling was done by M. Baumgardner and her studends
  - This resulted in a label-map of the dataset

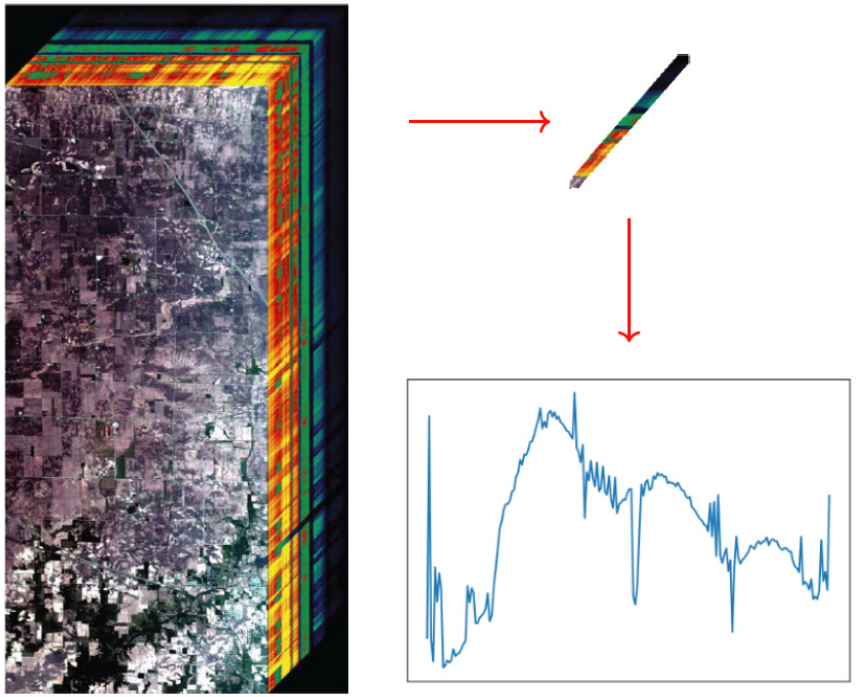


[8] P. U. R. Repository

[9] M.F. Baumgardner et al.

Remote Sensing Dataset – Indian Pines – Approaches

- Full dataset
  - 58 different classes
  - Distribution of the number of samples per class is highly unbalanced: Biggest class contains more than 60.000 pixels whereas there are several classes with less than 100 pixels
- Approaches
  - Some of the under-represented classes are excluded, e.g. six classes containing less than 100 pixels
  - Some of the spectral channels removed (e.g. only 200)



Representation of the dataset together with the intensity spectrum given for every pixel

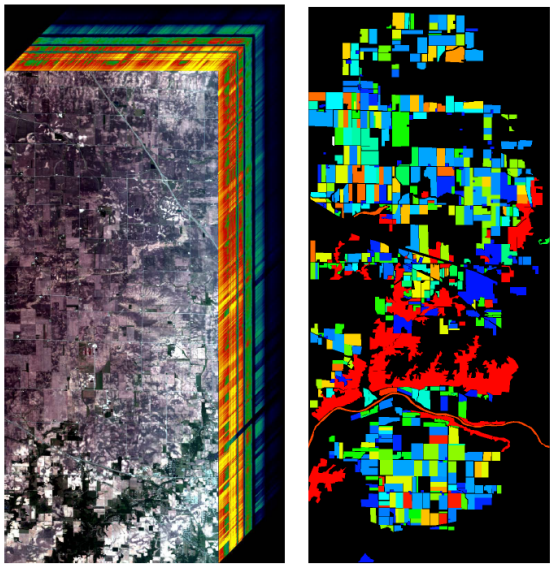
[4] G. Cavallaro, M. Riedel, J.A. Benediktsson et al., Journal of Selected Topics in Applied Earth Observation and Remote Sensing, 2015

[10] A. Romero et al., 'Unsupervised deep feature extraction for remote sensing image classification'

Advanced Supervised Learning Application – Indian Pines

- Challenging dataset
  - 52 classes (out of 58); rare groundtruth; mixed pixels; high dimensions

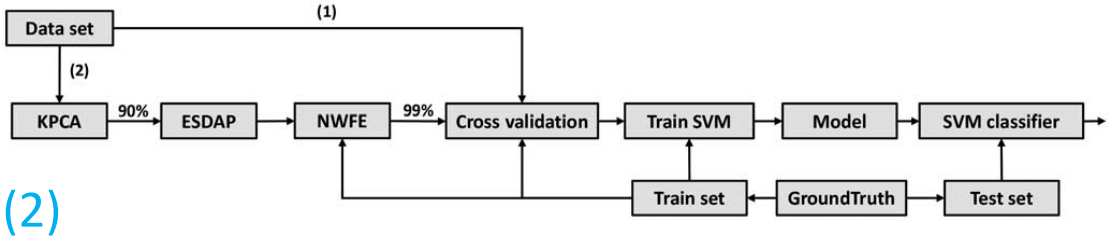
Class		Number of samples		Class		Number of samples	
Number	Name	Training	Test	Number	Name	Training	Test
1	Buildings	1720	15 475	27	Pasture	1039	9347
2	Corn	1778	16 005	28	pond	10	92
3	Corn?	16	142	29	Soybeans	939	8452
4	Corn-EW	51	463	30	Soybeans?	89	805
5	Corn-NS	236	2120	31	Soybeans-NS	111	999
6	Corn-CleanTill	1240	11 164	32	Soybeans-CleanTill	507	4567
7	Corn-CleanTill-EW	2649	23 837	33	Soybeans-CleanTill?	273	2453
8	Corn-CleanTill-NS	3968	35 710	34	Soybeans-CleanTill-EW	1180	10 622
9	Corn-CleanTill-NS-Irrigated	80	720	35	Soybeans-CleanTill-NS	1039	9348
10	Corn-CleanTilled-NS?	173	1555	36	Soybeans-CleanTill-Drilled	224	2018
11	Corn-MinTill	105	944	37	Soybeans-CleanTill-Weedy	54	489
12	Corn-MinTill-EW	563	5066	38	Soybeans-Drilled	1512	13 606
13	Corn-MinTill-NS	886	7976	39	Soybeans-MinTill	267	2400
14	Corn-NoTill	438	3943	40	Soybeans-MinTill-EW	183	1649
15	Corn-NoTill-EW	121	1085	41	Soybeans-MinTill-Drilled	810	7288
16	Corn-NoTill-NS	569	5116	42	Soybeans-MinTill-NS	495	4458
17	Fescue	11	103	43	Soybeans-NoTill	216	1941
18	Grass	115	1032	44	Soybeans-NoTill-EW	253	2280
19	Grass/Trees	233	2098	45	Soybeans-NoTill-NS	93	836
20	Hay	113	1015	46	Soybeans-NoTill-Drilled	873	7858
21	Hay?	219	1966	47	Swampy Area	58	525
22	Hay-Alfalfa	226	2032	48	River	311	2799
23	Lake	22	202	49	Trees?	58	522
24	NotCropped	194	1746	50	Wheat	498	4481
25	Oats	174	1568	51	Woods	6356	57 206
26	Oats?	34	301	52	Woods?	14	130

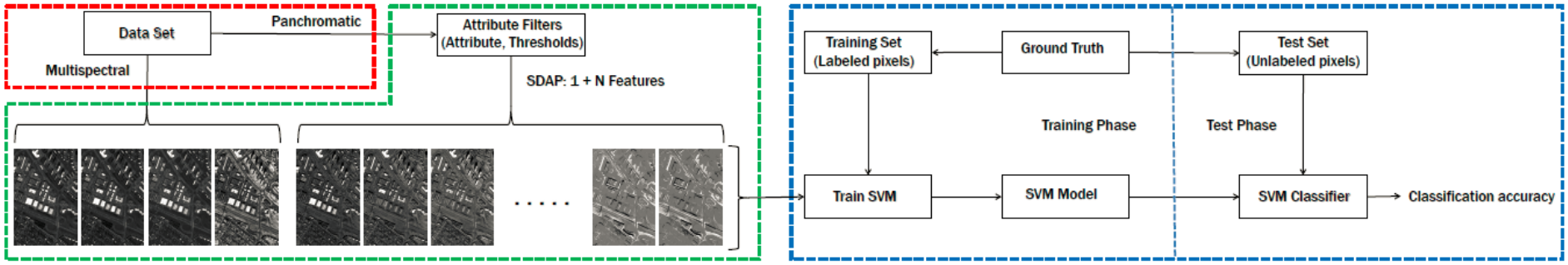


[4] G. Cavallaro, M. Riedel, J.A. Benediktsson et al., Journal of Selected Topics in Applied Earth Observation and Remote Sensing, 2015

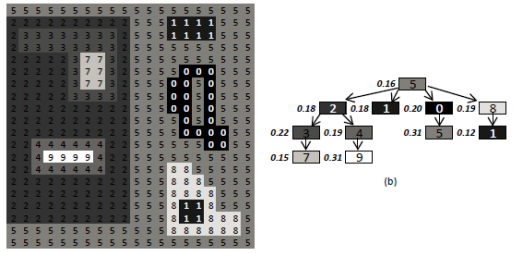
Importance of Features Engineering & Process Overview

- Application
  - Using dataset raw (1)
  - Using dataset processed (2)





- Feature Enhancement & Selection
  - Kernel Principle Component Analysis (KPCA)
  - Extended Self-Dual Attribute Profile (ESDAP)
  - Nonparametric weighted feature extraction (NWFE)



[4] G. Cavallaro, M. Riedel, J.A. Benediktsson et al.,
Journal of Selected Topics in Applied Earth Observation and Remote Sensing, 2015

Publicly Available Datasets – Location

- *Indian Pines Dataset Raw and Processed*

Abstract: 1) Indian raw: 1417x614x200 (training 10% and test)
2) Indian processed:1417x614x30 (training 10% and test)

[5] Indian Pine Image dataset

B2SHARE
Store and Share Research Data

Files

Name	Date	Size	
indian_processed_training.el	05 Feb 2015	11.7 MB	Download
indian_raw_test.el	05 Feb 2015	747.1 MB	Download
indian_raw_training.el	05 Feb 2015	83.0 MB	Download
indian_processed_test.el	05 Feb 2015	105.6 MB	Download

```
[morris@login-0-1 172-IndianPines]$ ls -al
insgesamt 925280
drwxrwxr-x  2 morris morris      4096 19. Nov 14:04 .
drwxrwxr-x 11 morris morris      4096 19. Nov 13:51 ..
-rw-rw-r--  1 morris morris 105594346  5. Feb 2015  indian_processed_test.el
-rw-rw-r--  1 morris morris  11732509  5. Feb 2015  indian_processed_training.el
-rw-rw-r--  1 morris morris 747125597  5. Feb 2015  indian_raw_test.el
-rw-rw-r--  1 morris morris  83014311  5. Feb 2015  indian_raw_training.el
```

Available Datasets – Training Data Example

- *Indian Pines Dataset Processed – Training*
  - Indian_processed_training.el
  - LibSVM data format: class feature1:value1 feature2:value2

[5] Indian Pine Image dataset

```
[morris@login-0-1 pisvmexamples]$ head /home/morris/BIGDATA/172-IndianPines/indian_processed_training.el
48 1:0.69021 2:0.599122 3:0.864571 4:0.265246 5:0.566572 6:0.561181 7:0.665698 8:0.430511 9:0.717402 10:0.514414 11:0.335391 12:0.486979 13:
0.58382 14:0.503849 15:0.420948 16:0.624377 17:0.43594 18:0.324333 19:0.614007 20:0.431993 21:0.600198 22:0.577696 23:0.528974 24:0.458141 2
5:0.543731 26:0.382421 27:0.33752 28:0.543062 29:0.5737 30:0.467784
48 1:0.675361 2:0.585579 3:0.85587 4:0.249317 5:0.579717 6:0.5633 7:0.634459 8:0.434777 9:0.719665 10:0.510279 11:0.330765 12:0.478097 13:0.
559455 14:0.505097 15:0.396293 16:0.627475 17:0.425701 18:0.32044 19:0.637625 20:0.437285 21:0.620149 22:0.535474 23:0.525151 24:0.511439 25
:0.504738 26:0.385925 27:0.303322 28:0.572963 29:0.570833 30:0.491508
51 1:0.554975 2:0.499039 3:0.578284 4:0.648481 5:0.867561 6:0.17041 7:0.841141 8:0.54466 9:0.746417 10:0.459481 11:0.356237 12:0.420116 13:0
.248721 14:0.541081 15:0.444207 16:0.286447 17:0.364712 18:0.522284 19:0.647555 20:0.538008 21:0.51632 22:0.437087 23:0.402312 24:0.574872 2
5:0.633603 26:0.426417 27:0.397177 28:0.400566 29:0.470729 30:0.51207
51 1:0.561296 2:0.495371 3:0.330829 4:0.513766 5:0.556074 6:0.387238 7:0.932418 8:0.643317 9:0.735262 10:0.216349 11:0.284832 12:0.499805 13
:0.272644 14:0.289776 15:0.699631 16:0.142032 17:0.161307 18:0.657442 19:0.72633 20:0.446231 21:0.542728 22:0.526951 23:0.433363 24:0.647875
 25:0.679122 26:0.760654 27:0.302971 28:0.515992 29:0.602717 30:0.34428
46 1:0.870891 2:0.768472 3:0.783699 4:0.28838 5:0.632466 6:0.418282 7:0.508729 8:0.437482 9:0.610721 10:0.480019 11:0.291032 12:0.427472 13:
0.428984 14:0.447632 15:0.452797 16:0.527423 17:0.583162 18:0.42837 19:0.557041 20:0.290409 21:0.547233 22:0.642817 23:0.535524 24:0.4478 25
:0.601117 26:0.405341 27:0.315288 28:0.488306 29:0.659466 30:0.469213
46 1:0.87119 2:0.779909 3:0.769342 4:0.292747 5:0.643116 6:0.416394 7:0.530679 8:0.4283 9:0.61224 10:0.474311 11:0.325682 12:0.468692 13:0.4
2888 14:0.461495 15:0.471023 16:0.524454 17:0.527418 18:0.398571 19:0.555335 20:0.35517 21:0.557823 22:0.588179 23:0.564881 24:0.444141 25:0
.576903 26:0.495859 27:0.344496 28:0.541081 29:0.539253 30:0.494238
15 1:0.869845 2:0.781471 3:0.412471 4:0.318165 5:0.414446 6:0.320591 7:0.605434 8:0.726664 9:0.590541 10:0.457315 11:0.355967 12:0.429357 13
:0.509808 14:0.421272 15:0.5406 16:0.532659 17:0.568507 18:0.404705 19:0.523209 20:0.52817 21:0.575417 22:0.570611 23:0.578825 24:0.452228 2
5:0.484941 26:0.542836 27:0.335545 28:0.491274 29:0.656853 30:0.546783
15 1:0.892436 2:0.790123 3:0.485313 4:0.318438 5:0.478488 6:0.329743 7:0.602466 8:0.581215 9:0.597853 10:0.442717 11:0.409894 12:0.479275 13
:0.51758 14:0.445422 15:0.483192 16:0.549861 17:0.49607 18:0.469129 19:0.475024 20:0.638295 21:0.578851 22:0.586814 23:0.579959 24:0.463105
25:0.550809 26:0.490199 27:0.376347 28:0.468157 29:0.55987 30:0.42689
15 1:0.809988 2:0.508887 3:0.782026 4:0.284838 5:0.682891 6:0.361665 7:0.243857 8:0.330213 9:0.456187 10:0.385119 11:0.230994 12:0.403049 13
:0.333648 14:0.473923 15:0.570014 16:0.45283 17:0.314596 18:0.574783 19:0.708582 20:0.292825 21:0.531681 22:0.616414 23:0.506682 24:0.493297
 25:0.396451 26:0.599563 27:0.329084 28:0.541152 29:0.56897 30:0.466168
8 1:0.843723 2:0.734137 3:0.686274 4:0.335633 5:0.6219 6:0.250049 7:0.575193 8:0.50452 9:0.623444 10:0.496143 11:0.263468 12:0.482361 13:0.5
46293 14:0.402207 15:0.730217 16:0.378631 17:0.433787 18:0.654169 19:0.513311 20:0.202298 21:0.530649 22:0.696446 23:0.631684 24:0.447328 25
:0.425982 26:0.40432 27:0.568027 28:0.461795 29:0.411903 30:0.428683
```

Parallelization Benefit: Parallel 10-Fold Cross-Validation

- Example: 10-fold cross-validation, Support Vector Machine (SVM)
  - 2 x benefits of parallelization possible in a so-called 'gridsearch'
  - (1) Compute parallel; (2) Do all cross-validation runs in parallel (all cells)
  - Evaluation between Matlab (aka 'serial laptop') & parallel (80 cores)

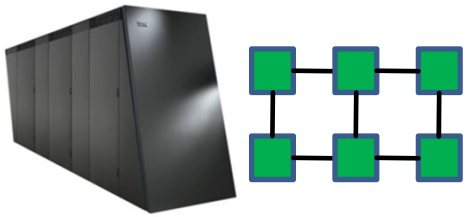
(2) Scenario 'pre-processed data', 10xCV **serial**: accuracy (min)

γ/C	1	10	100	1000	10 000
2	48.90 (18.81)	65.01 (19.57)	73.21 (20.11)	75.55 (22.53)	74.42 (21.21)
4	57.53 (16.82)	70.74 (13.94)	75.94 (13.53)	76.04 (14.04)	74.06 (15.55)
8	64.18 (18.30)	74.45 (15.04)	**77.00** (14.41)	75.78 (14.65)	74.58 (14.92)
16	68.37 (23.21)	76.20 (21.88)	76.51 (20.69)	75.32 (19.60)	74.72 (19.66)
32	70.17 (34.45)	75.48 (34.76)	74.88 (34.05)	74.08 (34.03)	73.84 (38.78)

(1) First Result: best parameter set from 14.41 min to 1.02 min

(2) Second Result: all parameter sets from ~9 hours to ~35 min

[4] G. Cavallaro, M. Riedel, J.A. Benediktsson et al., Journal of Selected Topics in Applied Earth Observation and Remote Sensing, 2015



'(1) each cell inherent parallel'

'(2) all cells in parallel'

(2) Scenario 'pre-processed data', 10xCV **parallel**: accuracy (min)

γ/C	1	10	100	1000	10 000
2	75.26 (1.02)	65.12 (1.03)	73.18 (1.33)	75.76 (2.35)	74.53 (4.40)
4	57.60 (1.03)	70.88 (1.02)	75.87 (1.03)	76.01 (1.33)	74.06 (2.35)
8	64.17 (1.02)	74.52 (1.03)	**77.02 (1.02)**	75.79 (1.04)	74.42 (1.34)
16	68.57 (1.33)	76.07 (1.33)	76.40 (1.34)	75.26 (1.05)	74.53 (1.34)
32	70.21 (1.33)	75.38 (1.34)	74.69 (1.34)	73.91 (1.47)	73.73 (1.33)

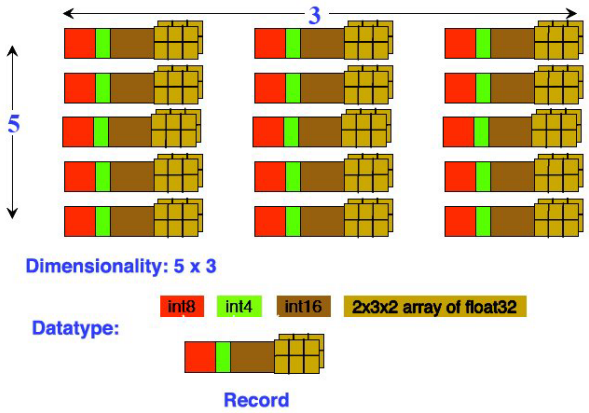
- **10-fold cross-validation achieves parallelization benefits (1) in each grid cell and (2) across all cells**

Hierarchical Data Format (HDF) often used in HPC

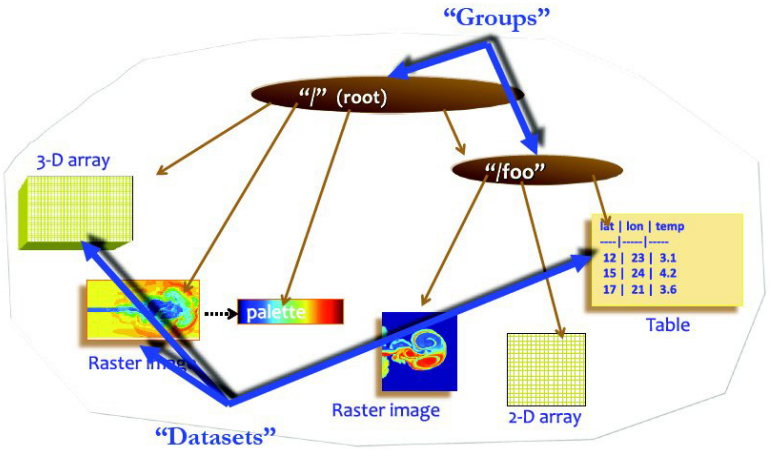
- **HDF is a technology suite that enables the work with extremely large and complex data collections**

[7] HDF@ I/O workshop

- Simple ‘compound type‘ example:
  - Array of data records with some descriptive information (5x3 dimension)
  - HDF5 data structure type with int(8); int(4); int(16); 2x3x2 array (float32)



‘HDF5 file is a container’ to organize data objects



Exercises

Available Datasets – Training & Testing Data Example

- *Indian Pines Dataset Raw Images*
  - Already available for the tutorial
  - HDF5 data format
- Full Dataset (USE AFTER TUTORIAL ONLY – LIMITED # GPUs)
  - All 58 classes; 10% training set / 90 % test set

```
[vsc42544@gligar01 full]$ pwd
/apps/gent/tutorials/deep_learning/IndianPines/data/full
[vsc42544@gligar01 full]$ ls -al
total 23271616
drwxr-xr-x 2 vsc40003 vsc40003        4096 Nov 29 11:34 .
drwxr-xr-x 4 vsc40003 vsc40003        4096 Nov 29 11:34 ..
-rw-r--r-- 1 vsc40003 vsc40003 11915039575 Nov 29 11:26 cnn_in_9x9_random_0.1_1.hdf5
```

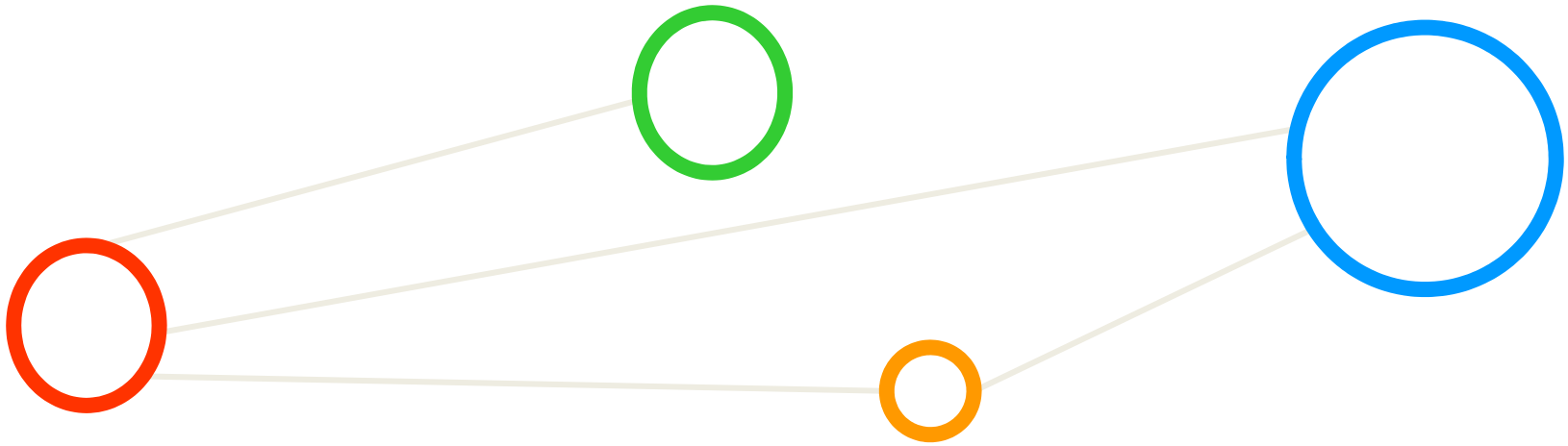
- Small Dataset (PLEASE USE THIS DATASET IN THE TUTORIAL)
  - 16 classes; crop of original image (145 *145 pixels); 10% train / 90% test

```
[vsc42544@gligar01 small]$ pwd
/apps/gent/tutorials/deep_learning/IndianPines/data/small
[vsc42544@gligar01 small]$ ls -al
total 721728
drwxr-xr-x 2 vsc40003 vsc40003      4096 Nov 29 11:34 .
drwxr-xr-x 4 vsc40003 vsc40003      4096 Nov 29 11:34 ..
-rw-r--r-- 1 vsc40003 vsc40003 369523607 Nov 29 11:33 cnn_in_9x9_random_0.1_1.hdf5
```

[Video] Remote Sensing

[6] YouTube Video, 'Remote Sensing'

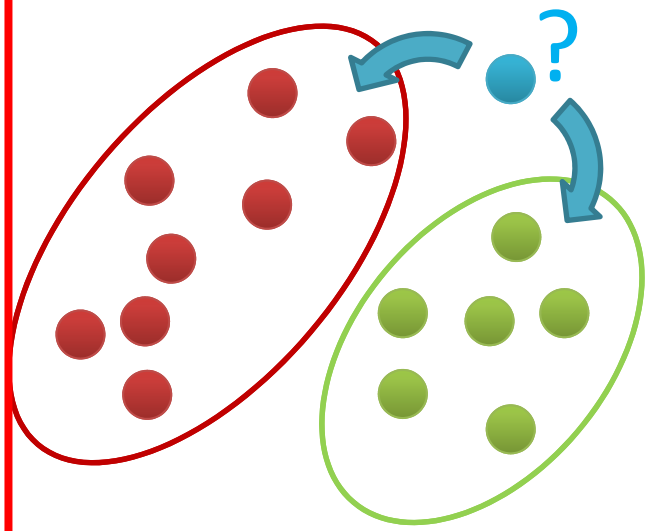
Remote Sensing Application Domain



Methods Overview

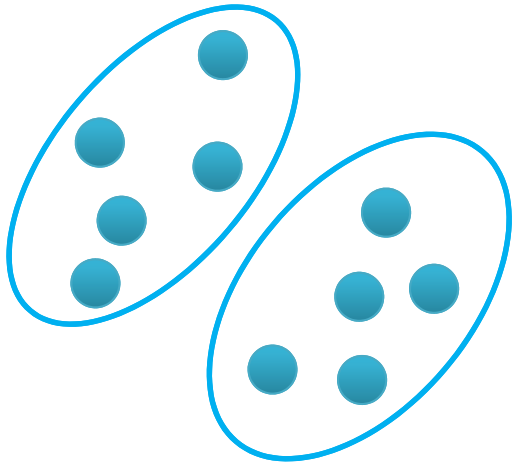
- **Machine learning methods can be roughly categorized in classification, clustering, or regression augmented with various techniques for data exploration, selection, or reduction**

Classification



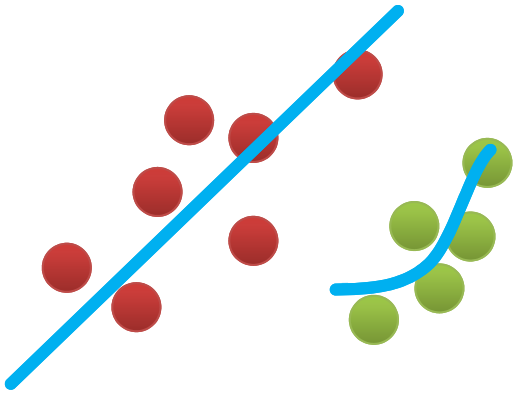
- Groups of data exist
- New data classified to existing groups

Clustering



- No groups of data exist
- Create groups from data close to each other

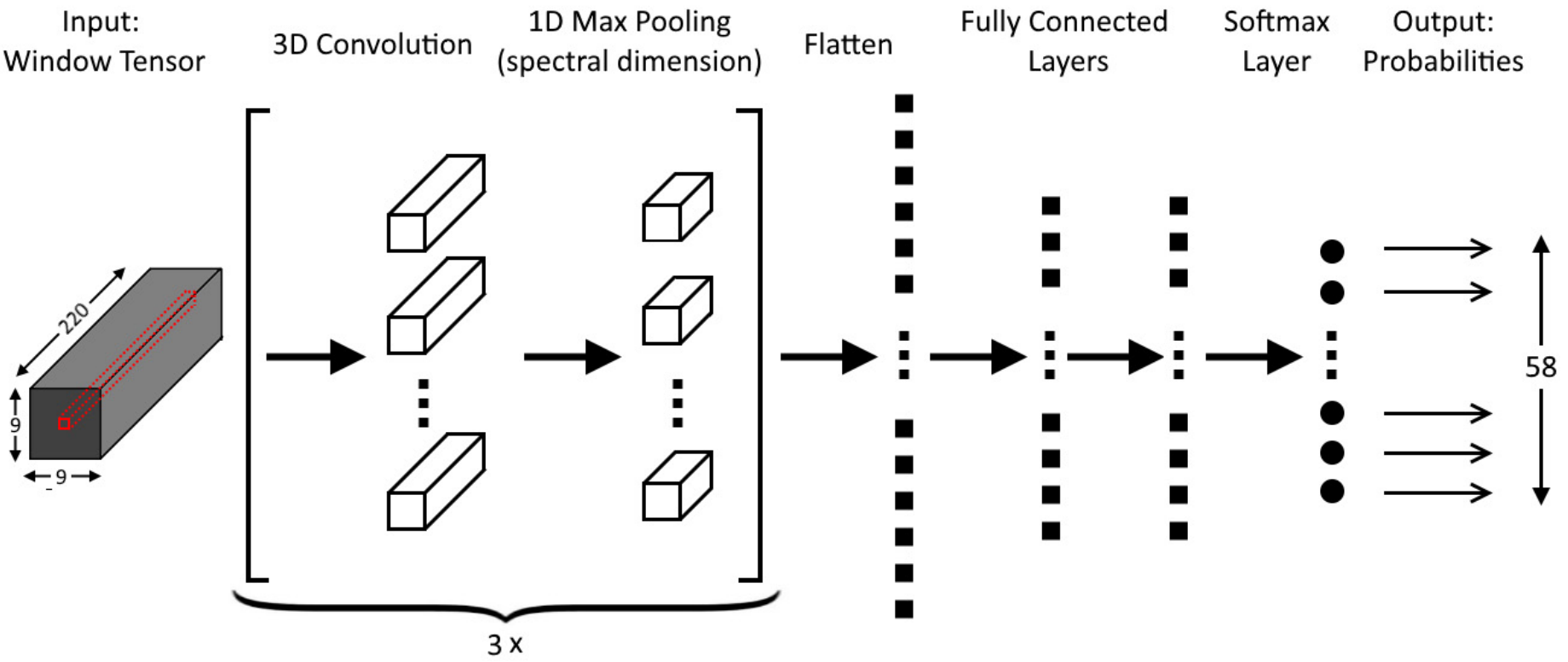
Regression



- Identify a line with a certain slope describing the data

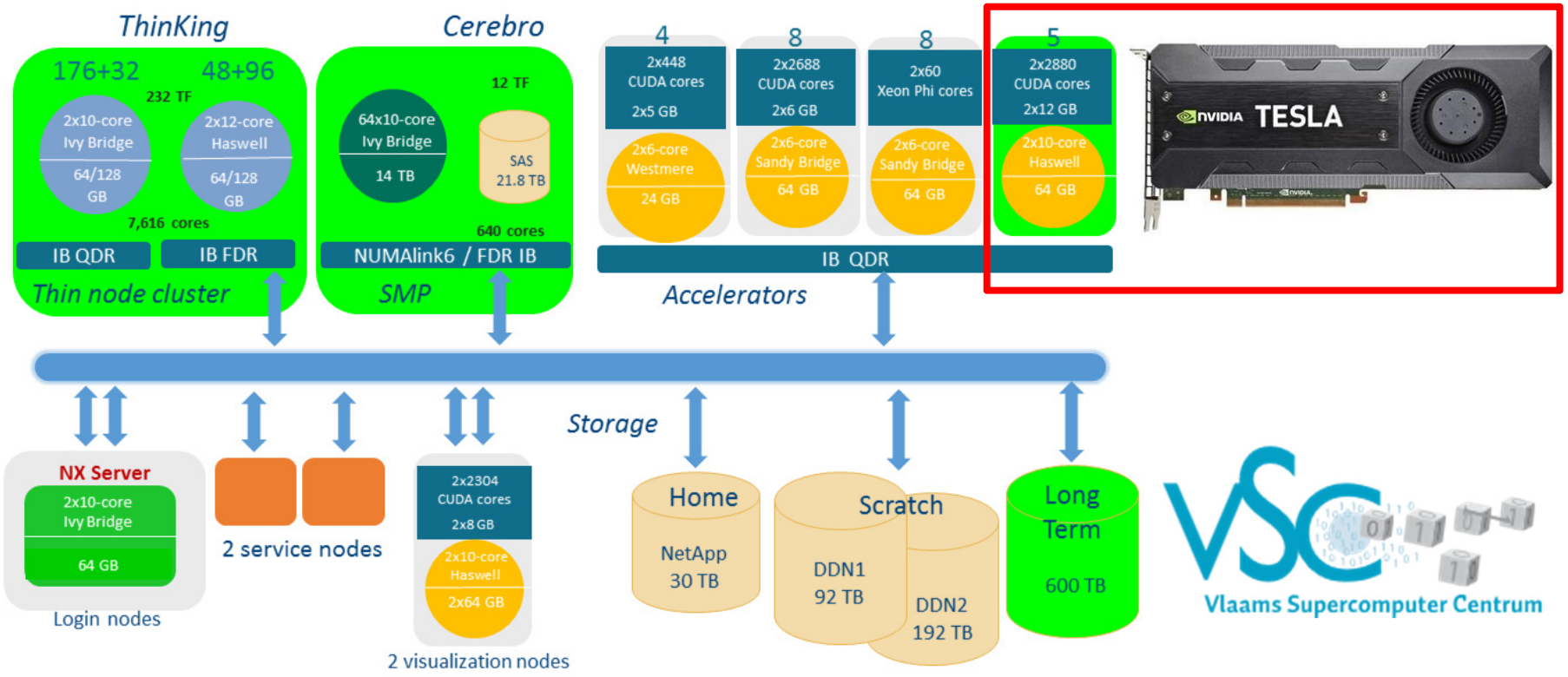
CNN Architecture for Application – Overview

- **Classify pixels in a hyperspectral remote sensing image having groundtruth/labels available**
- **Created CNN architecture for a specific hyperspectral land cover type classification problem**
- **Used dataset of Indian Pines (compared to other approaches) using all labelled pixels/classes**
- **Performed no manual feature engineering to obtain good results (aka accuracy)**



HPC System KU Leuven – GPUs – Careful Usage Please

- Accelerators
  - Nodes with two 10-core "Haswell" Xeon E5-2650v3 2.3GHz CPUs, 64 GB of RAM and 2 GPUs Tesla K40



modified from [11] HPC System KU Leuven

Exercises – Login into the KU Leuven System from Ghent

```
[vsc42544@gligar03 ~]$ ssh login.hpc.kuleuven.be
```

Remote Sensing Small Data CNN – Script Locations

```
[vsc42544@gligar01 scripts]$ pwd
/apps/gent/tutorials/deep_learning/IndianPines/scripts
[vsc42544@gligar01 scripts]$ ls -al
total 256
drwxr-xr-x 2 vsc40003 vsc40003  4096 Nov 29 14:18 .
drwxr-xr-x 4 vsc40003 vsc40003  4096 Nov 29 11:38 ..
-rw-r--r-- 1 vsc40003 vsc40003 15181 Nov 29 14:19 3Dcnn_full_2gpu.py
-rw-r--r-- 1 vsc40003 vsc40003 15186 Nov 29 13:44 3Dcnn_small_2gpu.py
-rw-r--r-- 1 vsc40003 vsc40003 15190 Nov 29 12:07 3Dcnn_small.py
-rw-r--r-- 1 vsc40003 vsc40003  3665 Nov 29 12:05 JLCallbacks.py
-rwxr--r-- 1 vsc40003 vsc40003  1914 Nov 29 14:27 job_full_2gpu.sh
-rwxr--r-- 1 vsc40003 vsc40003  1819 Nov 29 14:07 job_small_2gpu.sh
-rwxr--r-- 1 vsc40003 vsc40003   835 Nov 29 13:27 job_small.sh
-rw-r--r-- 1 vsc40003 vsc40003 21145 Nov 29 12:05 RS_data.py
```

Remote Sensing Small Data CNN – GPU Job Script – Part 1

```
#!/bin/bash
#PBS -N IndianPines_small_2GPU
#PBS -l nodes=1:ppn=12
#PBS -l walltime=4:0:0
#PBS -l partition=gpu
#PBS -A ldeeplearning
#PBS -l advres=DeepLearning-course.273656

###############################################################################
# IMPORTANT NOTE: to change after Deep Learning course:
#
# * change the '-A ldeeplearning' line to use the right credits
#   (see http://hpc.ugent.be/userwiki/index.php/Tips:Software:GPGPU and
#    https://www.vscentrum.be/cluster-doc/running-jobs/credit-system-basics)
#
# * remove the '-l advres=...' line to avoid submitting to a non-existing reservation!
###############################################################################

# make modules installed for GPGPU nodes in Leuven (CentOS6, K20/K40 Kepler GPUs)
module use /apps/gent/SL6/${VSC_ARCH_LOCAL}-kepler/modules/all

module load Tensorflow/1.4.0-intel-2017b-gpu-Python-3.6.3
module load Keras/2.1.1-intel-2017b-Python-3.6.3
module load scikit-learn/0.19.1-intel-2017b-Python-3.6.3
module load libgpuarray/0.7.5-intel-2017b-Python-3.6.3

# seems to be required to make sure that libcuda.so.1 is found
export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/usr/lib64

# prepare working directory
export WORKDIR=$VSC_SCRATCH/${PBS_JOBNAME}_${PBS_JOBID}
mkdir -p $WORKDIR
```

Remote Sensing Small Data CNN – GPU Job Script – Part 2

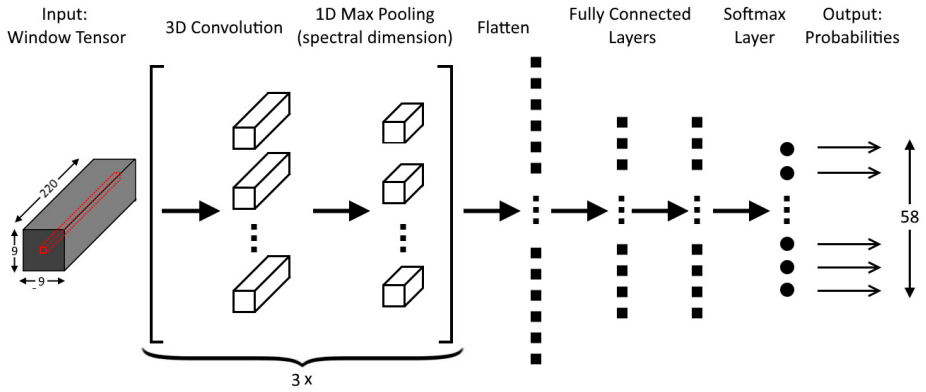
```
# copy Python scripts from location where job was submitted from
cp $PBS_O_WORKDIR/*.py .

# copy input file (small: 350MB, full: 12GB)
input=cnn_in_9x9_random_0.1_1.hdf5
cp -a /apps/gent/tutorials/deep_learning/IndianPines/data/small/$input .

# make sure Keras is using TensorFlow backend (Theano is a lot slower on GPU)
export KERAS_BACKEND=tensorflow

# dump info on available GPUs
nvidia-smi

# run!
time python 3Dcnn_small_2gpu.py $input model.h5 train.txt test.txt results.txt

echo "Results in $WORKDIR"
```

Remote Sensing Small Data CNN – GPU Python

```
[vsc42544@gligar02 scripts]$ diff -u 3Dcnn_small.py 3Dcnn_small_2gpu.py
--- 3Dcnn_small.py      2017-11-29 12:07:41.286370000 +0100
+++ 3Dcnn_small_2gpu.py 2017-11-29 13:44:57.511959496 +0100
@@ -20,7 +20,6 @@



-
 # standard model that achieved 85% accuracy on test in best of 5
 # The pool_size is adapted to 220 channels
 # The first two dimensions of kernel_size are adapted to 9x9 window tensors
@@ -170,8 +169,8 @@
     # compile either the built or the loaded model:
     model.compile(optimizer=optimizer, loss=loss, metrics=['accuracy'])
     ###### multiple gpus
-    #parallel_model = multi_gpu_model(model, gpus=4)
-    #parallel_model.compile(optimizer=optimizer, loss=loss, metrics=['accuracy'])
+    parallel_model = multi_gpu_model(model, gpus=2)
+    parallel_model.compile(optimizer=optimizer, loss=loss, metrics=['accuracy'])
     ######
     #   -   -   -   -   -   -   -   -   load and pre process TEST data  -   -   -   -   -   -   -   -   -   -   -   -
     # analoguously to the loading of the training data
@@ -207,10 +206,10 @@
         # It tests the network on the test-set and saves the accuracy and loss.
         # If the accuracy is better than before, it saves the models parameters.
         cb_save = TestAndSaveEveryN(x_test, y_test, model_output, test_output)
-        history = model.fit(x_train, y_train, batch_size, epochs, shuffle='batch', callbacks=[cb_hist, cb_save])
+        #history = model.fit(x_train, y_train, batch_size, epochs, shuffle='batch', callbacks=[cb_hist, cb_save])

         ###### multiple gpus
-        #history = parallel_model.fit(x_train, y_train, batch_size, epochs, shuffle='batch', callbacks=[cb_hist, cb_save])
+        history = parallel_model.fit(x_train, y_train, batch_size, epochs, shuffle='batch', callbacks=[cb_hist, cb_save])
         ######

     print(' > Time needed for learning and testing: {} hours'.format((timer() - t_start)/(60*60)))
```

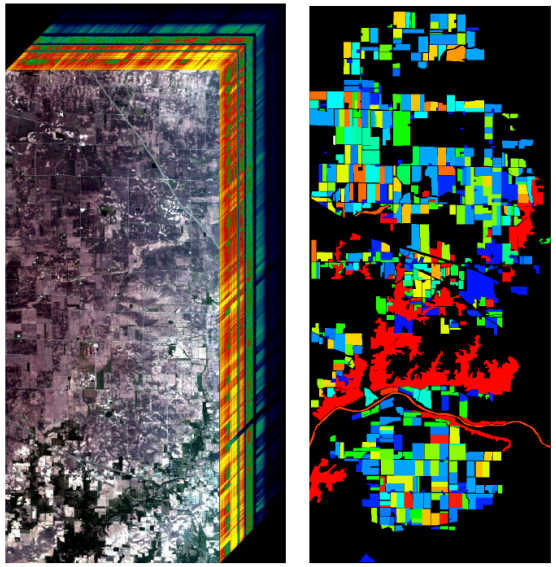
Keras – Remote Sensing CNN ‘Standard‘ Model

```
# standard model that achieved 85% accuracy on test in best of 5
# The pool_size is adapted to 220 channels
# The first two dimensions of kernel_size are adapted to 9x9 window tensors
def build_model_standard(input_shape, activation, num_classes):
    model = Sequential()
    model.add(Conv3D(48, kernel_size=(3, 3, 5), activation=activation, input_shape=input_shape))
    model.add(MaxPooling3D(pool_size=(1, 1, 3)))
    model.add(ZeroPadding3D((0, 0, 2), data_format=None))
    model.add(Conv3D(32, kernel_size=(3, 3, 5), activation=activation))
    model.add(MaxPooling3D(pool_size=(1, 1, 3)))
    model.add(ZeroPadding3D((0, 0, 2), data_format=None))
    model.add(Conv3D(32, kernel_size=(3, 3, 5), activation=activation))
    model.add(MaxPooling3D(pool_size=(1, 1, 2)))
    model.add(Flatten())
    model.add(Dense(128, activation=activation))
    model.add(Dense(128, activation=activation))
    model.add(Dense(num_classes, activation='softmax'))
    return model
```



Remote Sensing Dataset – Training and Testing Images (1)

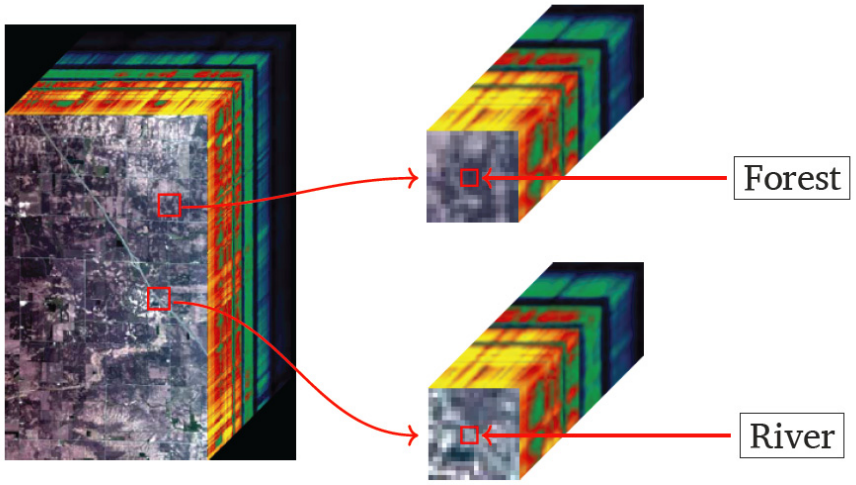
- LibSVM format approach
  - One possibility is to convert the data saved as image into the often used LibSVM format
  - One file for the training and one for the test data, each containing rows started by the label of the pixel and then containing the channel contents
  - Easy way to read in the samples
  - Disadvantage of loosing the information where the pixel was located and this way any spatial information



```
[morris@login-0-1 pisvmexamples]$ head /home/morris/BIGDATA/172-IndianPines/indian_processed_training.el
48 1:0.69021 2:0.599122 3:0.864571 4:0.265246 5:0.566572 6:0.561181 7:0.665698 8:0.430511 9:0.717402 10:0.
0.58382 14:0.503849 15:0.420948 16:0.624377 17:0.43594 18:0.324333 19:0.614007 20:0.431993 21:0.600198 22:
5:0.543731 26:0.382421 27:0.33752 28:0.543062 29:0.5737 30:0.467784
48 1:0.675361 2:0.585579 3:0.85587 4:0.249317 5:0.579717 6:0.5633 7:0.634459 8:0.434777 9:0.719665 10:0.51
559455 14:0.505097 15:0.396293 16:0.627475 17:0.425701 18:0.32044 19:0.637625 20:0.437285 21:0.620149 22:0
```

Remote Sensing Dataset – Training and Testing Images (2)

- Pixel with surroundings in HDF5 (9x9)
  - Store not only every pixel and its label but also the surrounding pixels (with all the spectral information, but without label) and also the pixel coordinates
  - Redundant way of saving the data but brings the advantage of using not only the spectral, but also the spatial information in the classification process
  - Increases the data size by a factor determined by the number of neighbouring pixels saved
  - HDF5 to decrease the data size, and in order to keep the I/O routines later on simple
  - Allows to store arrays of arbitrary dimensions in an optimized way



Label only describes the central pixel

Keras – Remote Sensing CNN – Imports

```
#   -   -   -   -   -   -   -   -   imports -   -   -   -   -   -   -   -   -   -   -
import keras
from keras import optimizers, regularizers
from keras.layers import Conv3D, Dense, MaxPooling3D, Flatten, ZeroPadding3D, Dropout
from keras.models import Sequential

import RS_data as rsd
from JLCallbacks import SaveHistory, TestAndSaveEveryN
import numpy as np
from timeit import default_timer as timer

import tensorflow as tf
from keras.applications import Xception
from keras.utils import multi_gpu_model

from sklearn.metrics import confusion_matrix
from sklearn.metrics import cohen_kappa_score
from sklearn import metrics
from keras.models import load_model
```

Keras – CNN Main Program & Parameters

```
def main(arguments):
    data_input = arguments[1]   # input data (e.g., cnn_in_9x9_random_0.1.hdf5 )
    model_output = arguments[2] # output model (in the current settings, it is saved every 10 epochs, file.h5)
    train_output = arguments[3] # training error and accuracy of each epoch (text file)
    test_output = arguments[4]  # test error and accuracy at each 10 epoch (text file)
    results_output=arguments[5] # classification results (OA,AA,kappa,F1,confusion matrix, text file)
    print(' >>> Start Program! <<< ')

    file = open(results_output, "w")
    file.write("Start loading \n")
    #   -   -   -   -   -   -   -   -   switches for program flow   -   -   -   -   -   -   -   -   -   -   -
    switch_test = 1         # loading test data?
    switch_loadweights = 0  # loading weights? (to reload trained model)


    #   -   -   -   -   -   -   -   -   set all the parameters  -   -   -   -   -   -   -   -   -   -   -   -
    num_classes = 58
    channels = 220
    window_size = 9
    batch_size = 50
    epochs = 2000
    learning_rate = 1
    momentum = 0
    decay = 0.000005
    reg_factor = 0.0        # for L2-regularization (see other models at the end of the program)
    dropout_frac = 0.0      # for dropout           (see other models at the end of the program)
    activation = 'relu'
    loss = 'mean_squared_error'
    optimizer = optimizers.SGD(lr=learning_rate, momentum=momentum, decay=decay)
```

Keras – CNN Load and Preprocess Training Data

```
#   -   -   -   -   -   -   -   -   load and pre process TRAINING data  -   -   -   -   -   -   -   -   -   -   -
print(' > Loading training data ... ')
t_start = timer()
x_train, y_train = rsd.load_train_from_hdf5(data_input)
x_train = np.array(x_train, np.float32)                                  # convert to numpy array
y_train = np.array(y_train)                                              # convert to numpy array
mean, std_dev, excluded = rsd.load_info_from_hdf5(data_input)
mean = np.array(mean, np.float32)                                        # convert to numpy array
std_dev = np.array(std_dev, np.float32)                                  # convert to numpy array
# normalize x with mean-deviation:
factor = 1 / std_dev
x_train = (x_train - mean) * factor
# transform y to categorical with length num_classes:
y_train = keras.utils.to_categorical(y_train-1, num_classes)             # convert digit to categorical vector (-1, because python starts indexing with 0)
# reshape x for CNN3D:
x_train = x_train.reshape(x_train.shape[0], x_train.shape[1], x_train.shape[2], x_train.shape[3], 1)
print(' > Time needed for reading and converting the training data: {} seconds'.format(timer() - t_start))
# specify input shape, because it is needed for the first layer:
input_shape = (x_train.shape[1], x_train.shape[2], x_train.shape[3], 1)
```

Keras – Remote Sensing CNN – Build the model

```
#   -   -   -   -   -   -   -   -   build the model -   -   -   -   -   -   -   -   -   -   -   -   -   -
print(' > Building the model ... ')
# choose from the functions at the end, also possible to specify new models:
###### multiple gpus
#with tf.device('/device:CPU:0'):
######
model = build_model_standard(input_shape, activation, num_classes)

# load weights in case it is enabled (the model has to be saved at "<data_output>/cnn3D_model_to_load.h5"):
if switch_loadweights == 1:
    model.load_weights('/path_to_saved_model.h5')
    print(' > Weights loaded! ')
else:
    print(' > Randomly initialised weights!')

# compile either the built or the loaded model:
model.compile(optimizer=optimizer, loss=loss, metrics=['accuracy'])
###### multiple gpus
#parallel_model = multi_gpu_model(model, gpus=4)
#parallel_model.compile(optimizer=optimizer, loss=loss, metrics=['accuracy'])
######
#   -   -   -   -   -   -   -   -   load and pre process TEST data  -   -   -   -   -   -   -   -   -   -
# analoguously to the loading of the training data
if switch_test == 1:
    print(' > Loading test data ... ')
    t_start = timer()
    x_test, y_test = rsd.load_test_from_hdf5(data_input)
    x_test = np.array(x_test, np.float32)
    y_test = np.array(y_test)
    # normalize x with mean-deviation:
    x_test = (x_test - mean) * factor
    # transform y to categorial with length num_classes:
    y_test = keras.utils.to_categorical(y_test-1, num_classes)
    # reshape x for CNN3D:
    x_test = x_test.reshape(x_test.shape[0], x_test.shape[1], x_test.shape[2], x_test.shape[3], 1)
    print(' > Time needed for reading and converting the test data: %g seconds' % (timer() - t_start))
```

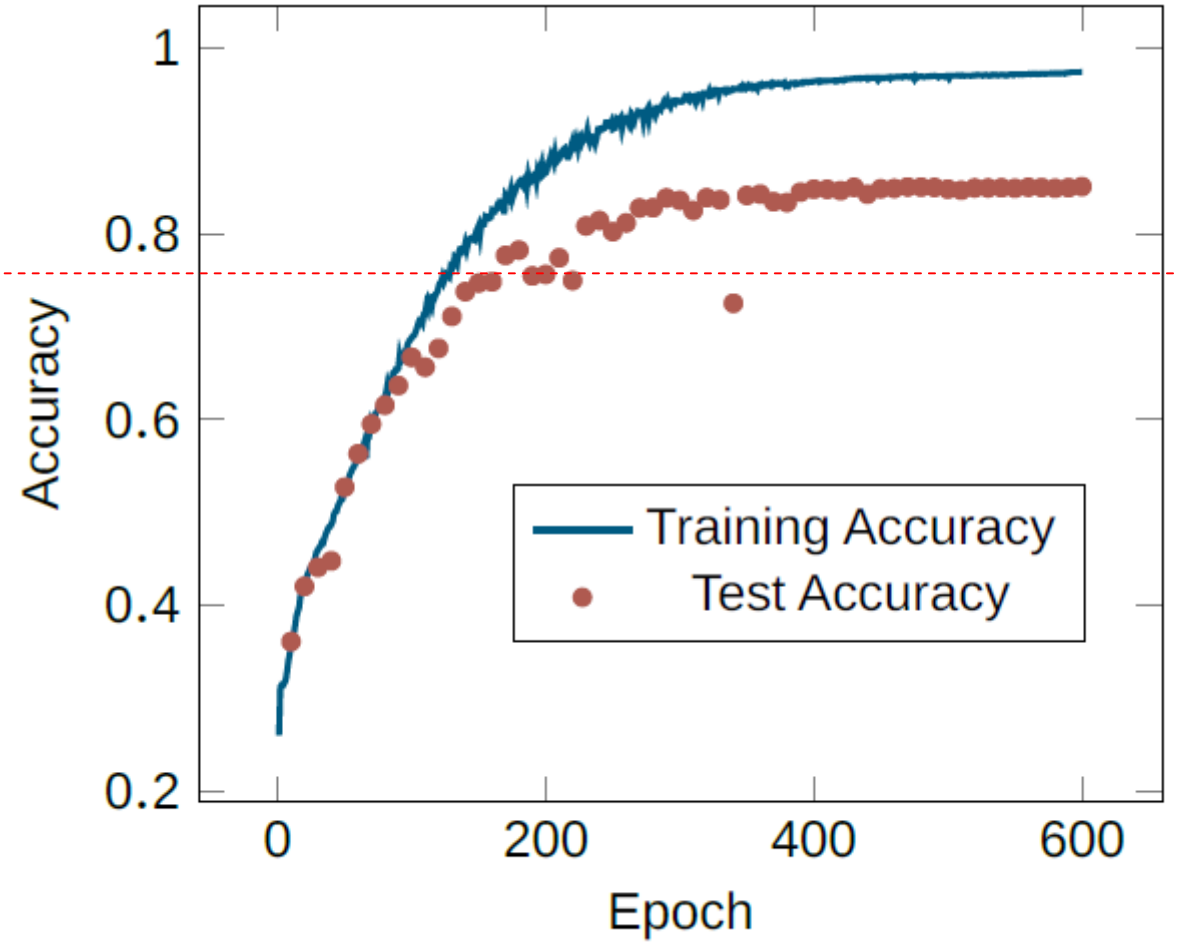
Experimental Setup @ Juelich Supercomputing Centre

- CNN Setup
  - Table overview
- HPC Machines used
  - Systems JURECA and JURON
- GPUs
  - NVIDIA Tesla K80 (JURECA)
  - NVIDIA Tesla P100 (JURON)
  - While Using MathWorks' Matlab for the
- Frameworks
  - Keras library (2.0.6) was used
  - Tensorflow (0.12.1 on Jureca, 1.3.0rc2 on Juron) as back-end
  - Automated usage of the GPU's of these machines via Tensorflow

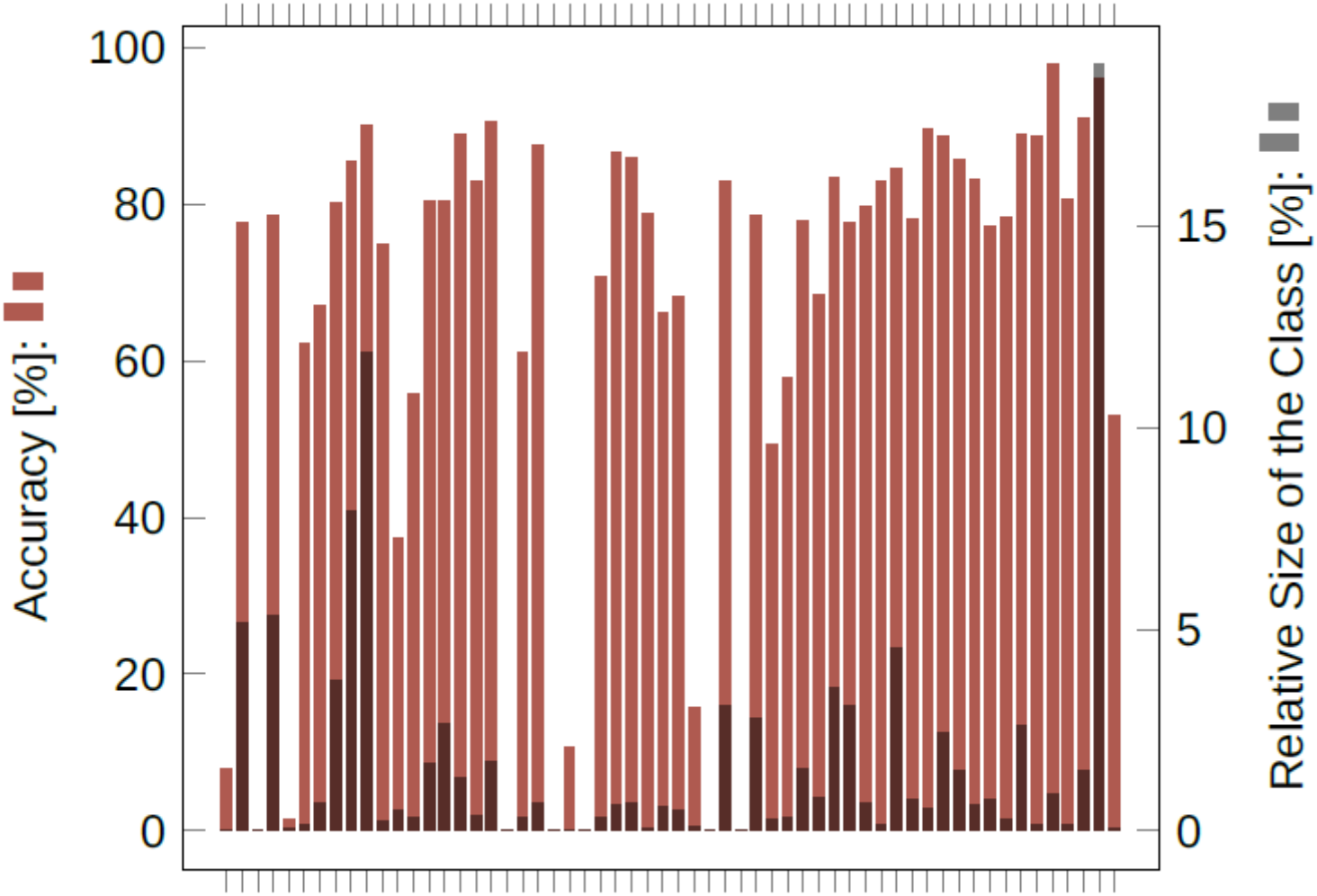
Feature	Representation / Value
Conv. Layer Filters	48, 32, 32
Conv. Layer Filter size	$(3,3,5)$, $(3,3,5)$, $(3,3,5)$
Dense Layer Neurons	128, 128
Optimizer	SGD
Loss Function	mean squared error
Activation Functions	ReLU
Training Epochs	600
Batch Size	50
Learning Rate	1
Learning Rate Decay	5×10^{-6}

Experimental Setup – Results – Full Dataset – Accuracy

SVM comparison ~ 77% with manual feature engineering

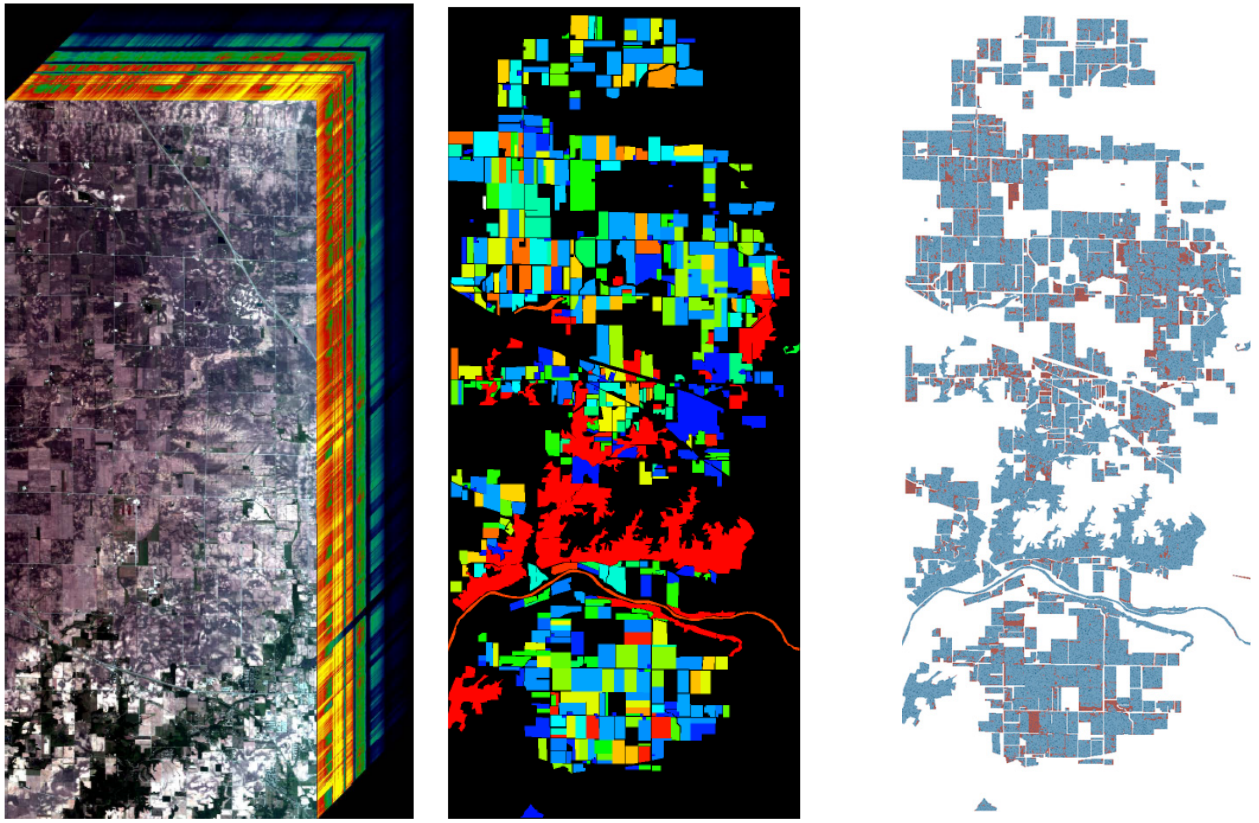


Experimental Setup – Results – Full Dataset – Class Checks



Experimental Setup – Results – Full Dataset – Class Checks

- Blue: correctly classified / training data
- Red: incorecctly classified



Exercises – Check GPU Job Runs

Small Data – Outputs

```
- - - - - - - Begin - Information - - - - - - -

--> Data:
> Number of classes: 16, HS-channels: 220, Window-size: 9
> Mean: 2524.4013671875 , Standard deviation: 1603.017822265625
> Excluded labels: []
> Number of training samples: 1036
> Number of test samples: 9330
--> Learning:
> Epochs: 1000, Batch size: 50
> LR: 1, Momentum: 0, LR-decay: 5e-06
> Activation Functions: relu, Loss Function: mean_squared_error
> Optimizer: SGD
--> Regularization:
> Dropout: 0.0
> L2 regularization with factor: 0.0

- - - - - - - End - Information - - - - - - -
```

Layer (type)	Output Shape	Param #
conv3d_1 (Conv3D)	(None, 7, 7, 216, 48)	2208
max_pooling3d_1 (MaxPooling3	(None, 7, 7, 72, 48)	0
zero_padding3d_1 (ZeroPaddin	(None, 7, 7, 76, 48)	0
conv3d_2 (Conv3D)	(None, 5, 5, 72, 32)	69152
max_pooling3d_2 (MaxPooling3	(None, 5, 5, 24, 32)	0
zero_padding3d_2 (ZeroPaddin	(None, 5, 5, 28, 32)	0
conv3d_3 (Conv3D)	(None, 3, 3, 24, 32)	46112
max_pooling3d_3 (MaxPooling3	(None, 3, 3, 12, 32)	0
flatten_1 (Flatten)	(None, 3456)	0
dense_1 (Dense)	(None, 128)	442496
dense_2 (Dense)	(None, 128)	16512
dense_3 (Dense)	(None, 16)	2064

Total params: 578,544

```
loss: 0.027155295770896957 - acc: 0.7379421221609412
New loss is bigger --> dont save model

1036/1036 [==============================] - 6s 5ms/step - loss: 8.0197e-04 - acc: 0.9894
 > Time needed for learning and testing: 0.49032413981337514 hours
```

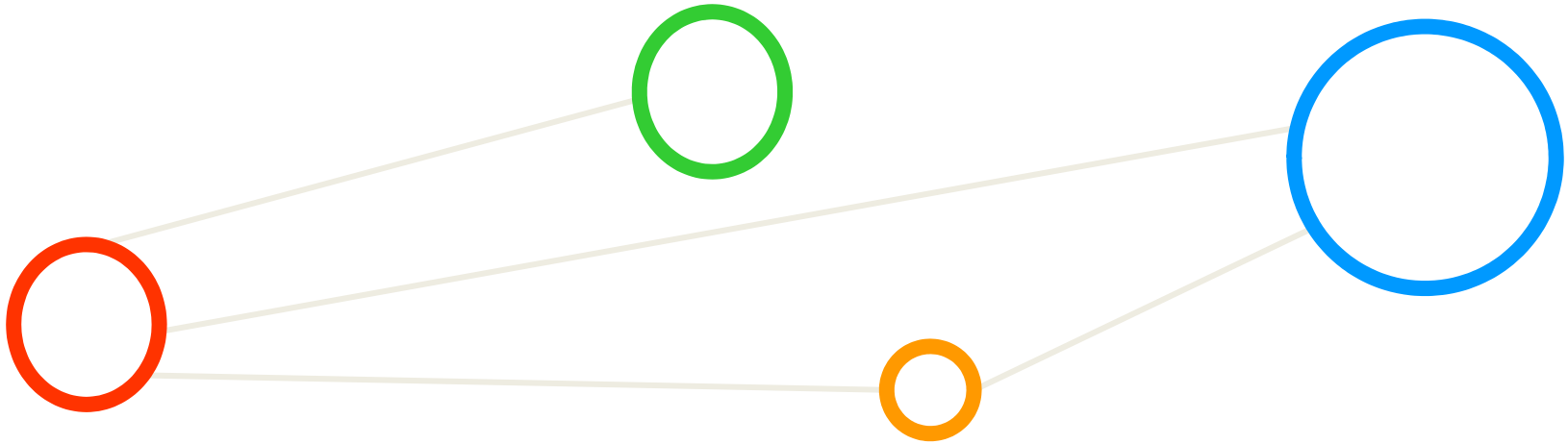
Full Data – Output (1)

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv3d_1 (Conv3D)            (None, 7, 7, 216, 48)     2208
_________________________________________________________________
max_pooling3d_1 (MaxPooling3 (None, 7, 7, 72, 48)      0
_________________________________________________________________
zero_padding3d_1 (ZeroPaddin (None, 7, 7, 76, 48)      0
_________________________________________________________________
conv3d_2 (Conv3D)            (None, 5, 5, 72, 32)      69152
_________________________________________________________________
max_pooling3d_2 (MaxPooling3 (None, 5, 5, 24, 32)      0
_________________________________________________________________
zero_padding3d_2 (ZeroPaddin (None, 5, 5, 28, 32)      0
_________________________________________________________________
conv3d_3 (Conv3D)            (None, 3, 3, 24, 32)      46112
_________________________________________________________________
max_pooling3d_3 (MaxPooling3 (None, 3, 3, 12, 32)      0
_________________________________________________________________
flatten_1 (Flatten)          (None, 3456)              0
_________________________________________________________________
dense_1 (Dense)              (None, 128)               442496
_________________________________________________________________
dense_2 (Dense)              (None, 128)               16512
_________________________________________________________________
dense_3 (Dense)              (None, 58)                7482
=================================================================
Total params: 583,962
Trainable params: 583,962
Non-trainable params: 0
_________________________________________________________________

 >>> Program End <<<
```

Full Data – Output (2)

```
[vsc42544@gligar02 .deep_learning_private]$ sed -n '906765,906824p' IndianPines_full_2GPU.o20657803
300821/300821 [==============================] - 137s 457us/step
loss: 0.004567355140805838 - acc: 0.8353838329111958
New loss is bigger --> dont save model

33424/33424 [==============================] - 182s 5ms/step - loss: 4.8412e-04 - acc: 0.9762
 > Time needed for learning and testing: 16.111324077533144 hours

 -   -   -   -   -   -   -   Begin - Information -   -   -   -   -   -   -

 --> Data:
 > Number of classes: 58, HS-channels: 220, Window-size: 9
 > Mean: 2424.492431640625 , Standard deviation: 1431.9654541015625
 > Excluded labels: []
 > Number of training samples: 33424
 > Number of test samples: 300821
 --> Learning:
 > Epochs: 1000, Batch size: 50
 > LR: 1, Momentum: 0, LR-decay: 5e-06
 > Activation Functions: relu, Loss Function: mean_squared_error
 > Optimizer: SGD
 --> Regularization:
 > Dropout: 0.0
 > L2 regularization with factor: 0.0

 -   -   -   -   -   -   -   End - Information   -   -   -   -   -   -   -
```

Lecture Bibliography



Lecture Bibliography (1)

- [1] Wikipedia on 'Remote Sensing',
  Online: http://en.wikipedia.org/wiki/Remote_sensing
- [2] G. Cavallaro and M. Riedel, 'Smart Data Analytics Methods for Remote Sensing Applications', 35th Canadian Symposium on Remote Sensing (IGARSS), 2014, Quebec, Canada
- [3] Rome Dataset, B2SHARE,
  Online: http://hdl.handle.net/11304/4615928c-e1a5-11e3-8cd7-14feb57d12b9
- [4] G. Cavallaro, M. Riedel, J.A. Benediktsson et al., 'On Understanding Big Data Impacts in Remotely Sensed Image Classification using Support Vector Machine Methods', IEEE Journal of Selected Topics in Applied Earth Observation and Remote Sensing, 2015
- [5] Indian Pine Image Dataset, B2SHARE,
  Online: http://hdl.handle.net/11304/7e8eec8e-ad61-11e4-ac7e-860aa0063d1f
- [6] YouTube Video, 'What is Remote Sensing',
  Online: https://www.youtube.com/watch?v=nU-CjAKry5c
- [7] Michael Stephan,'Portable Parallel IO - 'Handling large datasets in heterogeneous parallel environments',
  Online: http://www.fz-juelich.de/SharedDocs/Downloads/IAS/JSC/EN/slides/parallelio-2014/parallel-io-hdf5.pdf?__blob=publicationFile
- [8] P. U. R. Repository. 220 band aviris hyperspectral image data set: June 12, 1992 indian pine test site 3, 2015,
  Online: https://purr.purdue.edu/publications/1947/about?v=1
- [9] M. F. Baumgardner, L. L. Biehl, and D. A. Landgrebe. 220 band aviris hyperspectral, image data set: June 12, 1992 indian pine test site 3. *Purdue University Research Repository*, 2015.
- [10] A. Romero, C. Gatta, and G. Camps-Valls. Unsupervised deep feature extraction for remote sensing image classification. *IEEE Transactions on Geoscience and Remote Sensing*, 54(3):1349–1362, 2016.

Lecture Bibliography (2)

- [11] HPC System KU Leuven,
  Online: https://www.vscentrum.be/infrastructure/hardware/hardware-kul


services
measurement
within
fusion
new
funding
Policy-based
concepts
forms
cross-disciplinary
resources
device
Analysis
Centre
climate
Computer
expertise
year
like
Computational
Test
E.g
Computing
disciplines
results
Approaches
construction
Fusion
Enable
Cross-Disciplinary
methods
big
nodes
PB
community
Environment
Modelling
different
EU
Science
High
centers
Key
scientific
important
etc
DKRZ
Use
storage
basis
Infrastructure
files
simulations
ENES
technologies
SMAQ
international
model
engineering
References
formats
brain
Big
Project
Data
increasing
via
EUDAT
physical
processing
IPCC
Common
TB
Methods
MapReduce
create
computing
Using
European
databases
codes
GB
Supercomputing
Work
images
analysis
DLCL
stored
Resources
analyze
Large
Summary
data-intensive
Service
Scientific
levels
often
use
Level
Research
Matters
Concrete
external
performance
JSC
computational
research
e.g
science
HPC
large-scale
Juelich
system
datasets
acquisition
compute
Landscape
management
System
Climate
modelling
AR5
Hardware
distributed
cores
access
current
hundreds
Nutshell
Services
Technologies
Earth
Structure
network
DLCLs
Understanding
structures
directory
project
General
Germany
Simulation
Processing
Provide
NASA
Energy
systems
day
Health