

Detecting and Analyzing Solar Panels in Switzerland using Aerial Imagery (SolAI)

Adrian Meyer
Institute Geomatics
University of Applied Sciences
Northwestern Switzerland

Team

Institute Geomatics @ FHNW



Adrian Meyer

Data Scientist



Prof. Denis Jordan

Statistics & Mathematics



Prof. Martin Christen

Geoinformation & Computer Graphics

Project Partners



Schweizerische Eidgenossenschaft
Confédération suisse
Confederazione Svizzera
Confederaziun svizra

In collaboration with the cantons

Martin Hertach


Federal Office for Energy (BFE)

Peter Barmet



Energy Department of Canton Aargau (AG)

SolAI – Detection of Solar Systems

IGEO/FHNW and Federal Office for Energy (BFE)

 Schweizerische Eidgenossenschaft
 Confédération suisse
 Confederaziun svizra
 Confederaziun svizra

Bundesamt für Energie BFE
 Bundesamt für Meteorologie und Klimatologie MeteoSchweiz
 Bundesamt für Landestopografie swisstopo

 Dach
  Fassade

Wie viel Strom oder Wärme kann mein Dach produzieren?

Suchen Sie Ihre Adresse...

...ODER LOKALISIEREN SIE SICH ↗

Noch nicht lokalisiert

Bitte lokalisieren Sie sich, suchen Sie eine Adresse oder klicken Sie in der Karte auf ein Dach.

Suchen Sie Ihre Adresse...



Vollbild | Problem melden

CNES, Spot Image, swisstopo, NPOC | BFE

 Schweizerische Eidgenossenschaft
 Confédération suisse
 Confederaziun svizra
 Confederaziun svizra
 In Zusammenarbeit mit den Kantonen

Ort suchen oder Karte hinzufügen:

z.B. Bundesplatz 1 Bern, 46.7.7.6, Lärmkarte ...

Teilen
 Drucken
 Zeichnen & Messen auf der Karte
 Erweiterte Werkzeuge
 Energie Thema wechseln

Dargestellte Karten

Solarenergie: Eignung Dächer

Nach weiteren Karten suchen?

Menü schließen

Objekt-Information

Eignung von Hausdächern für die Nutzung von Sonnenenergie (Bundesamt für Energie)

Eignung	Sehr gut
Dachfläche [m ²]	594
Ausrichtung [°]	0
Neigung [°]	-
Finanzieller Ertrag [CHF]	9990.0
Weitere Information	solarerdach.ch

Eignung von Hausdächern für die Nutzung von Sonnenenergie (Bundesamt für Energie)

Eignung	Sehr gut
Dachfläche [m ²]	29
Ausrichtung [°]	0
Neigung [°]	-
Finanzieller Ertrag [CHF]	500.0
Weitere Information	solarerdach.ch

20 m CH1903+ / LV95

© Daten: CNES, Spot Image, swisstopo, NPOC, BFE
 www.nsp.energie.admin.ch Copyright & Datenschutzerklärung

sonnendach.ch



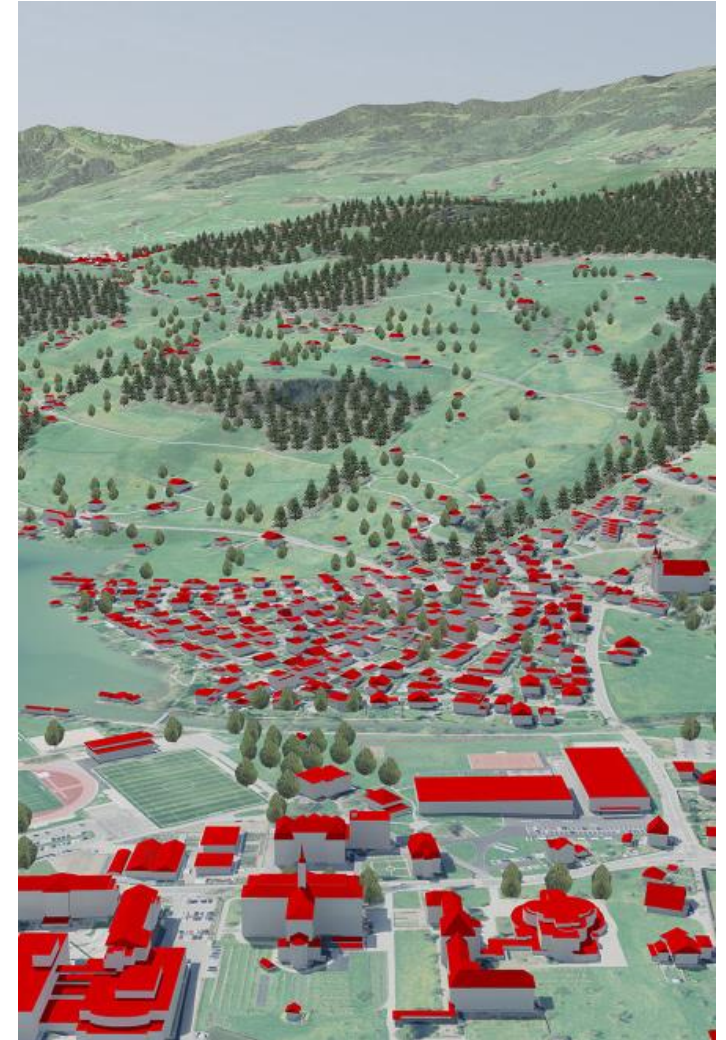
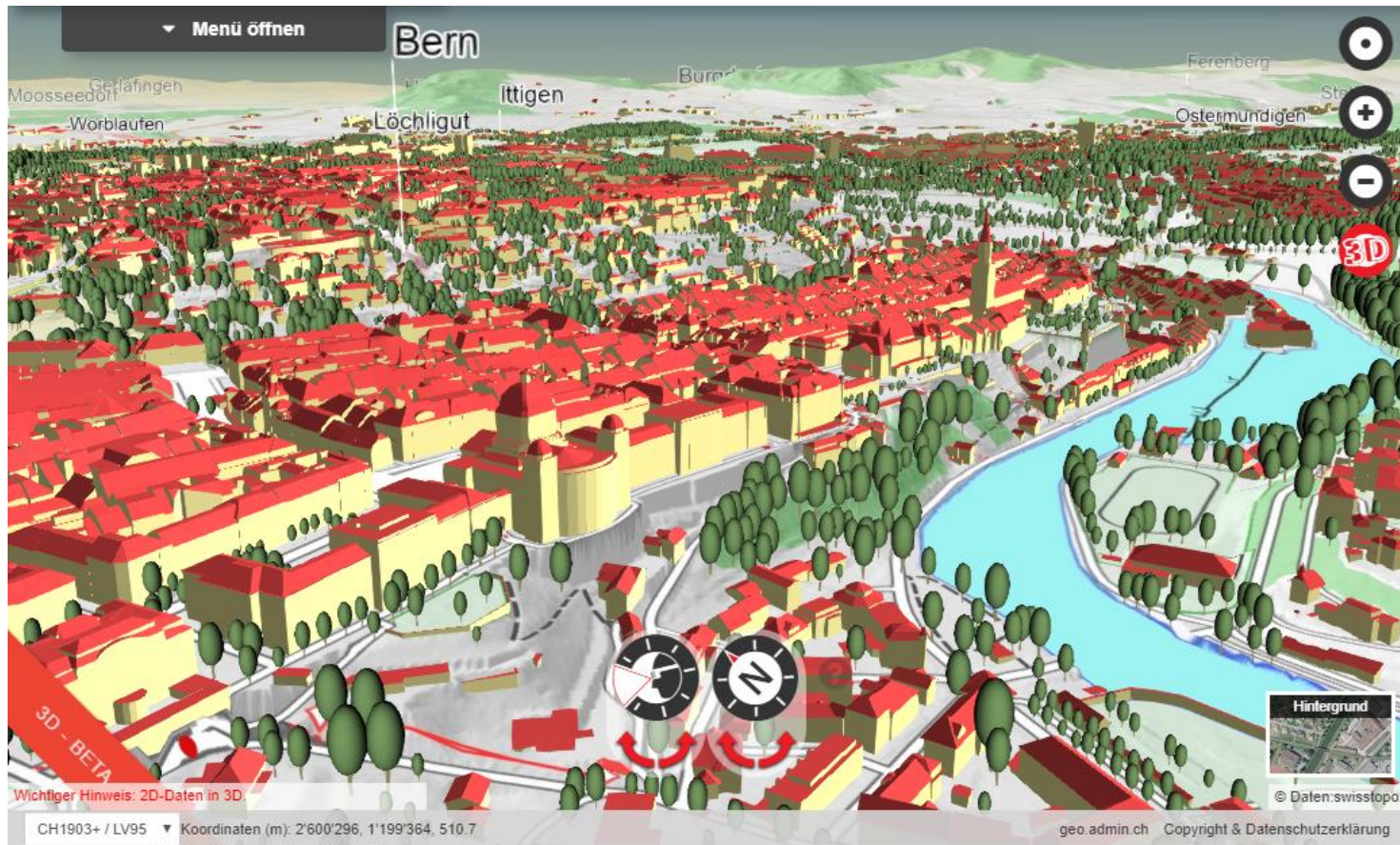
Object information

Suitability of roofs for the use of solar energy (Swiss Federal Office of Energy)

Suitability	Mean
Roof area [m2]	282
Orientation [°]	25
Inclination [°]	32
Earning [CHF]	3100.0
More info	sonnendach.ch

Riehe

Swiss Buildings 3D Dataset



Not for release

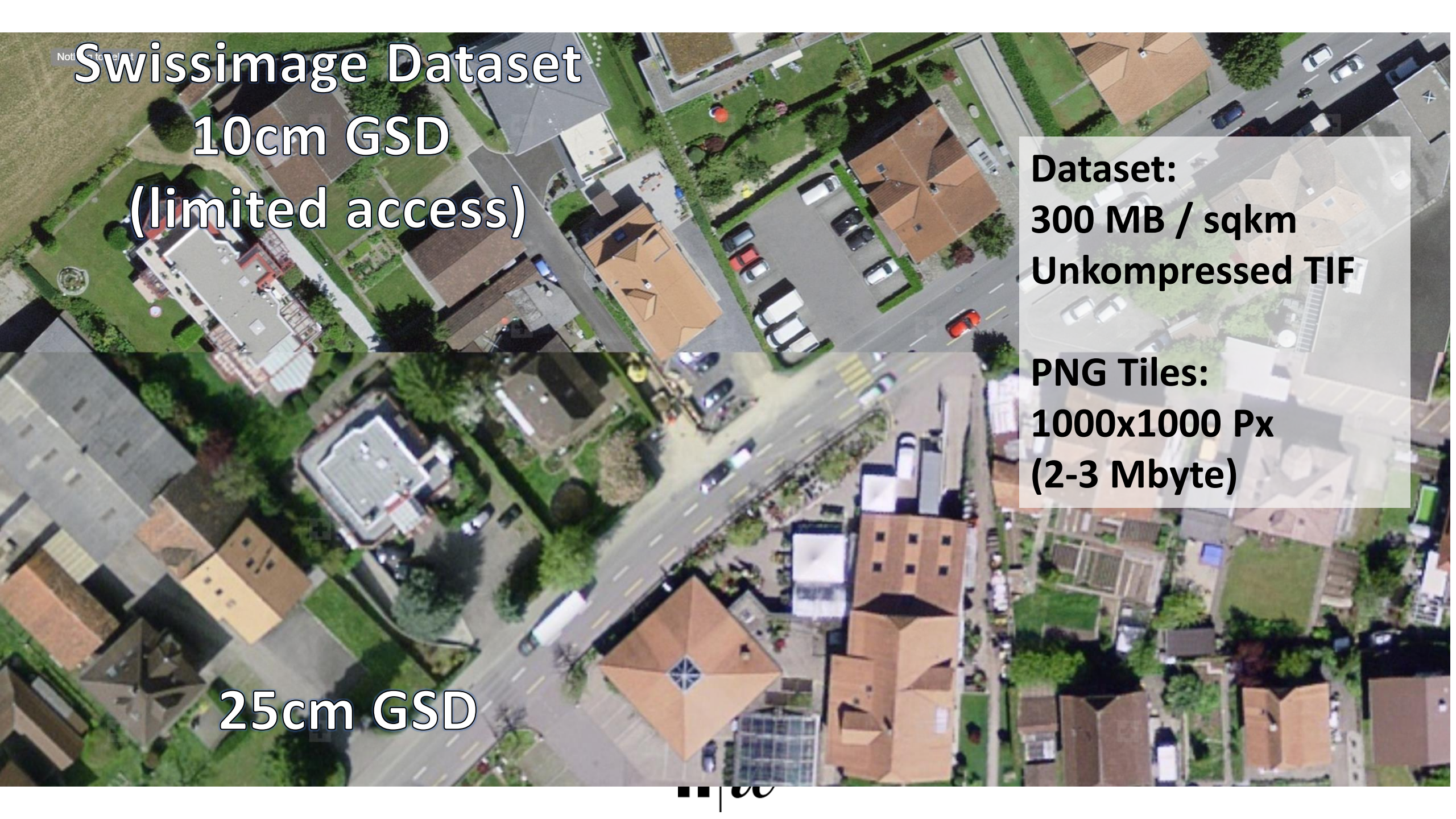
Swissimage Dataset

10cm GSD
(limited access)

Dataset:
300 MB / sqkm
Unkompressed TIF

PNG Tiles:
1000x1000 Px
(2-3 Mbyte)

25cm GSD



Summary of Input Data

- **We have:**

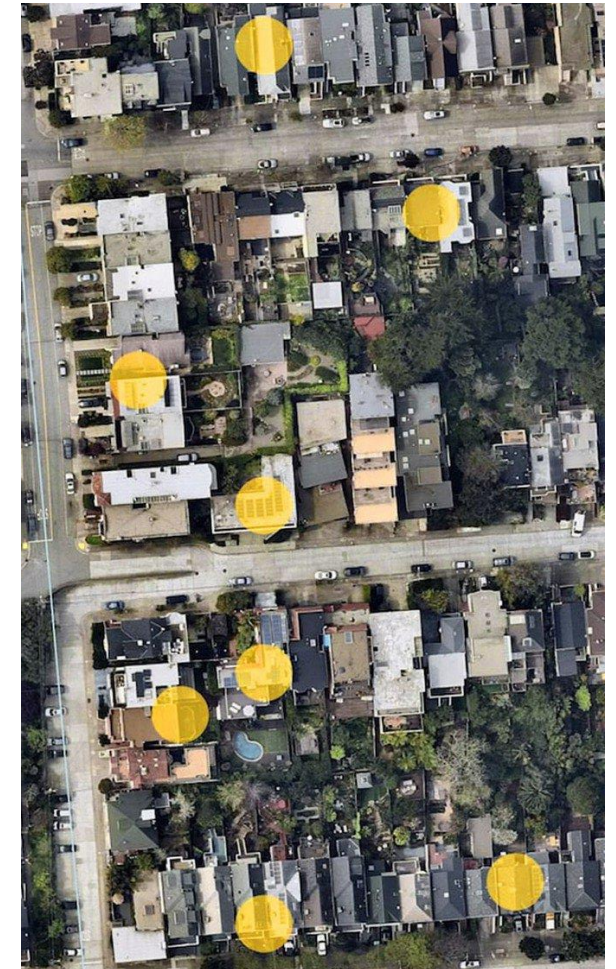
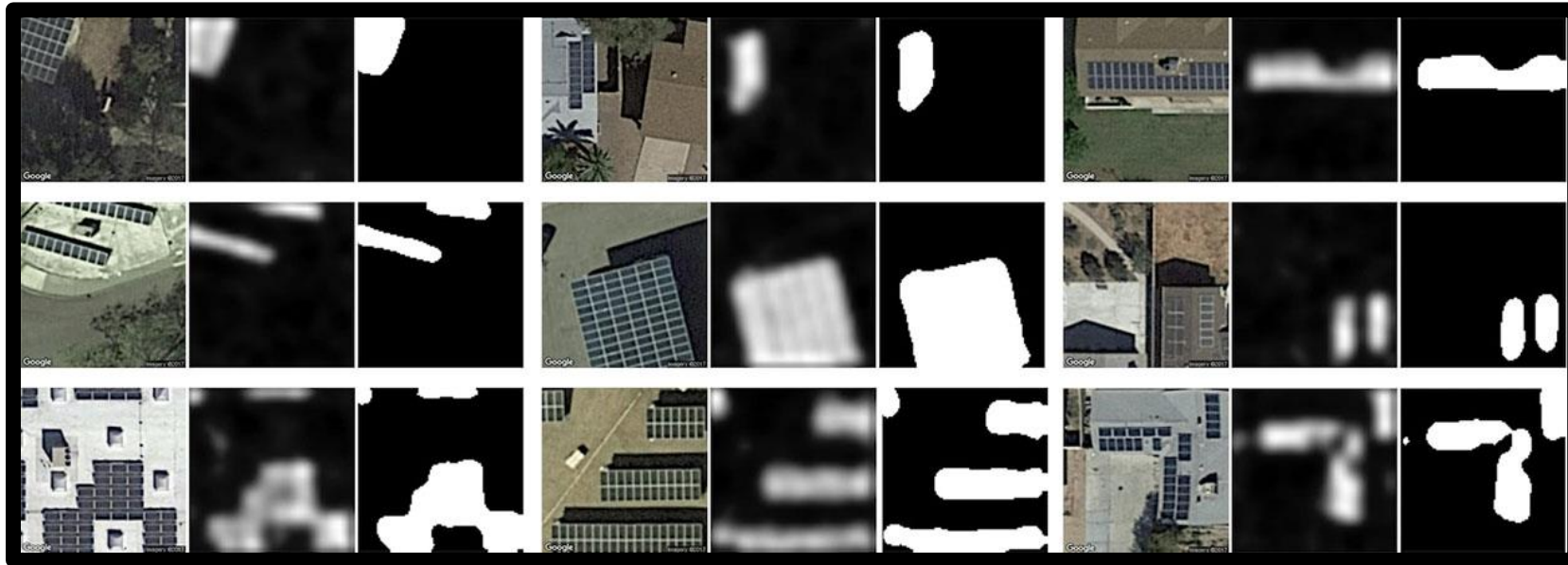
- Areal imagery (partially in 10cm², partially 25cm² per Pixel)
- Vector data / 3D Data of all roofs in Switzerland
- Roof size and solar potential
 - There are around 2 million buildings in Switzerland in total

- **We don't have:**

- Location, Size, and Type of solar panels (PV, Thermal)

Deep Solar (2018)

- <https://github.com/wangzhecheng/DeepSolar>



Images: DeepSolar/Stanford

Common Object Detection Tasks

Classification



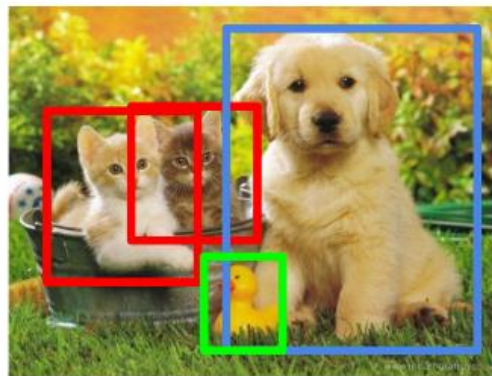
CAT

**Classification
+ Localization**



CAT

Object Detection



CAT, DOG, DUCK

**Instance
Segmentation**



CAT, DOG, DUCK

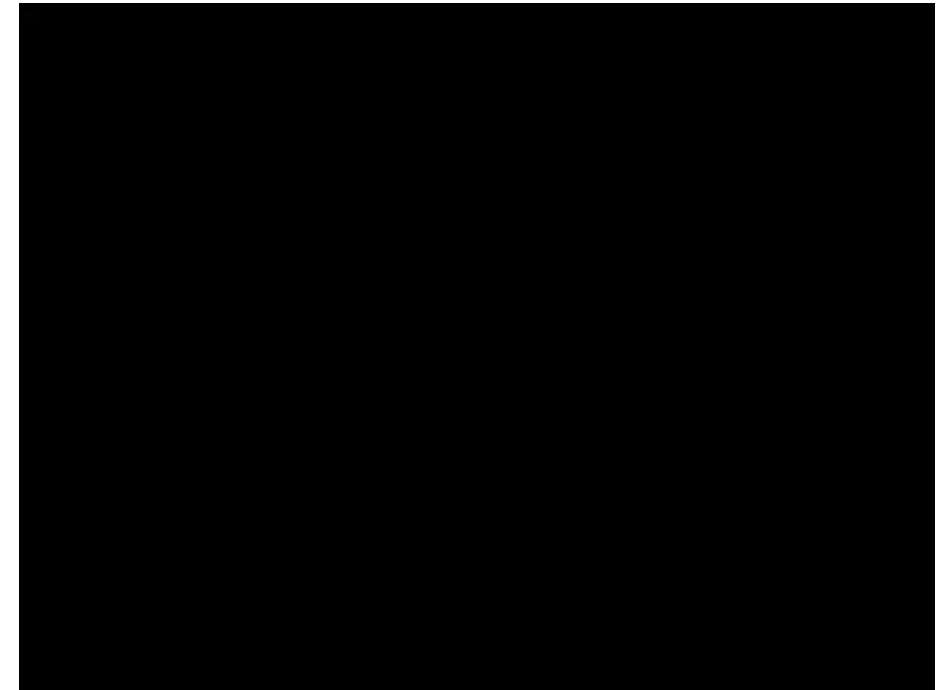
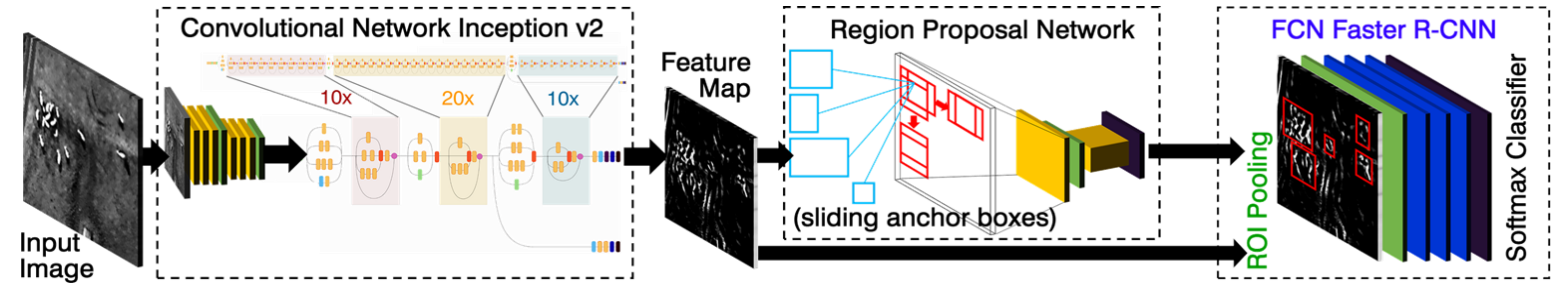
Single object

Multiple objects

Faster RCNN & TF to Identify Tiles with Panels



Region-based CNN, implemented with Tensorflow

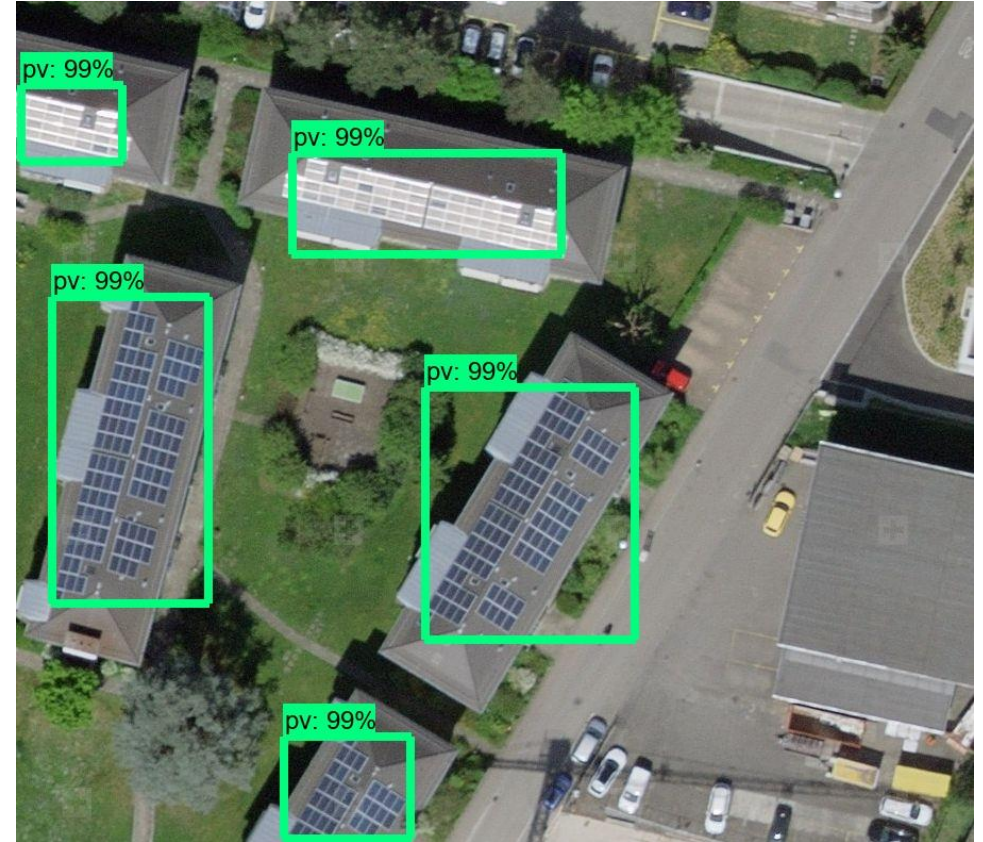


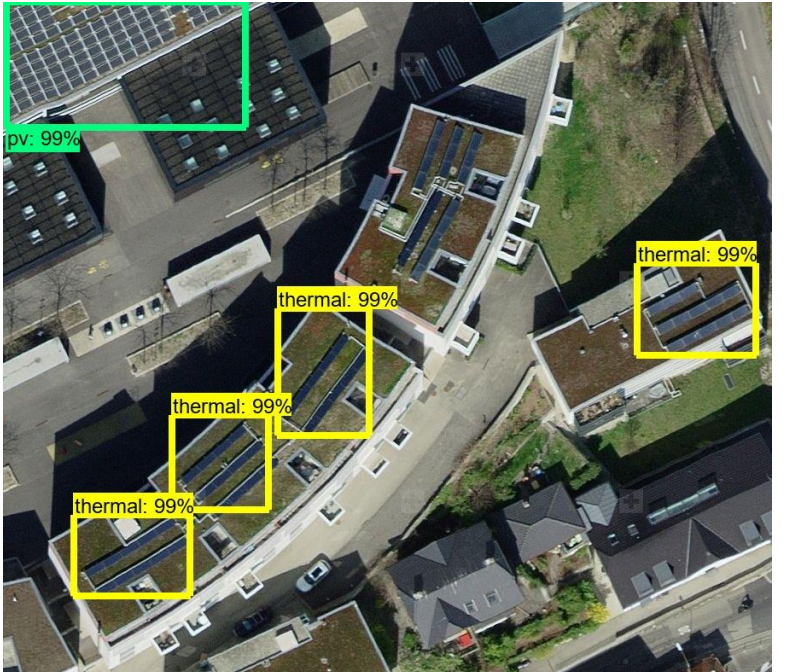
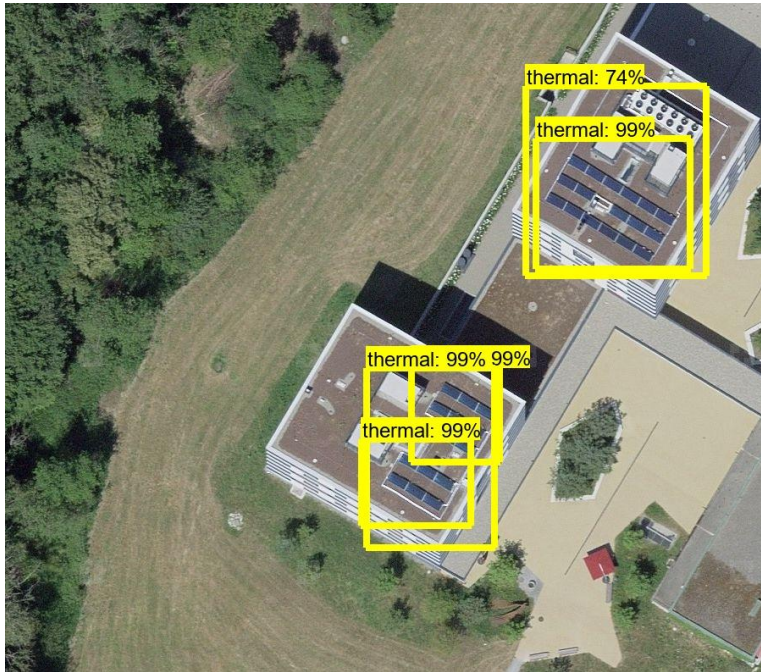
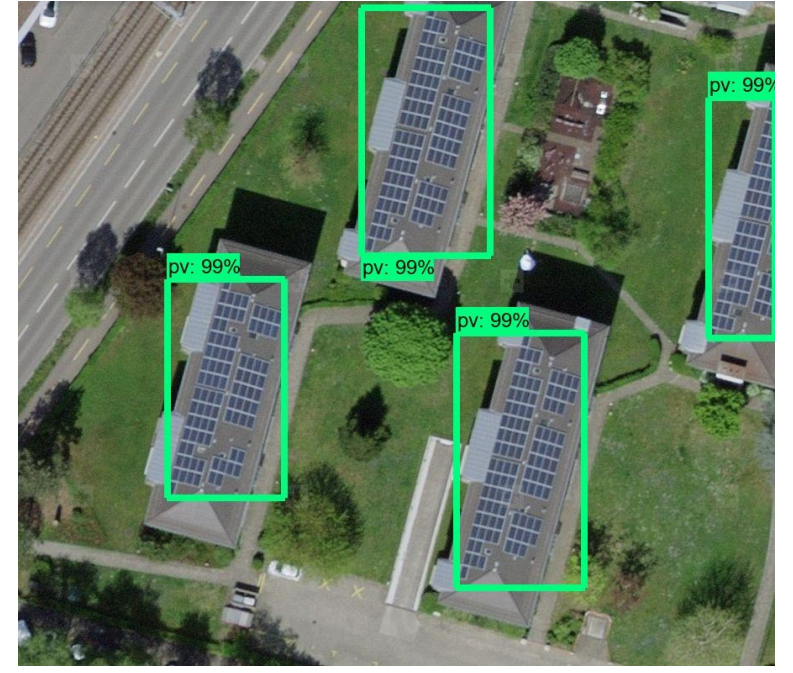
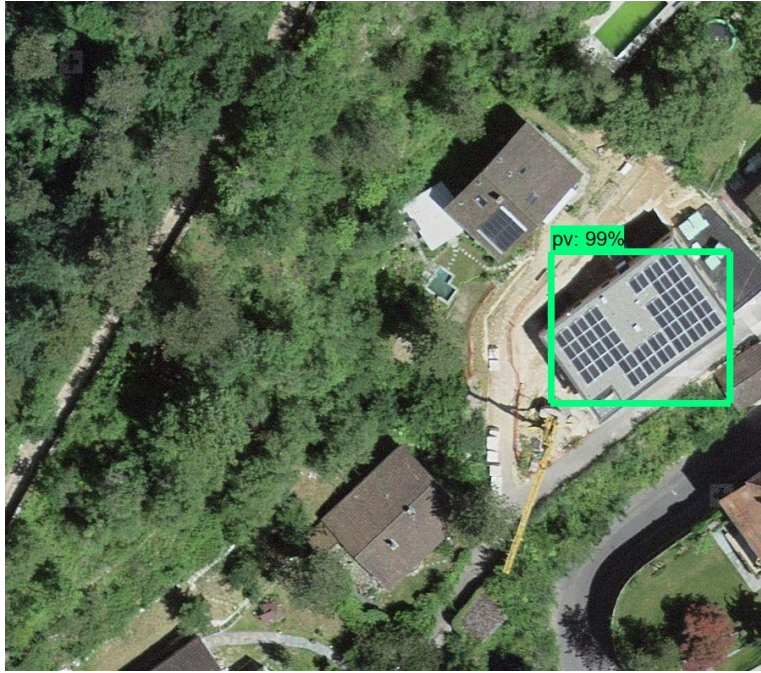
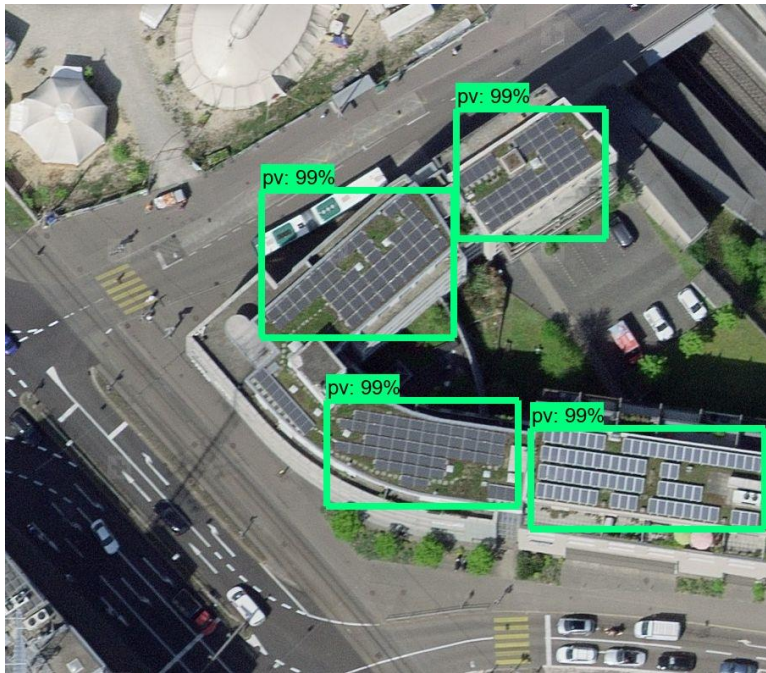
Train your own Solar Detector!

- https://colab.research.google.com/github/FHNW-IVGI/workshop_geopython2019/blob/master/Ex.02_SolarPanels/FasterRCNN_Tutorial_MeyerA.ipynb

- Available under:

tinyurl.com/solar-detect

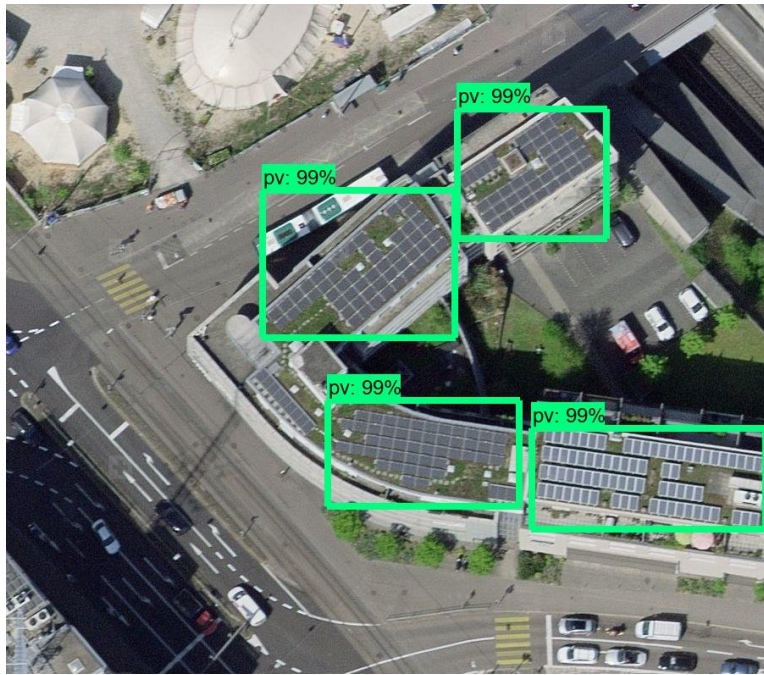




Pre-Detection

Photovoltaic Systems: 92% mAP

Thermal Systems: 62% (ca. 30% are detected as PV)



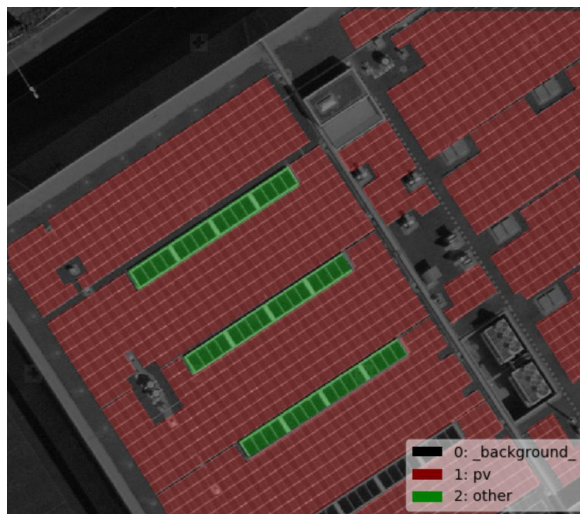
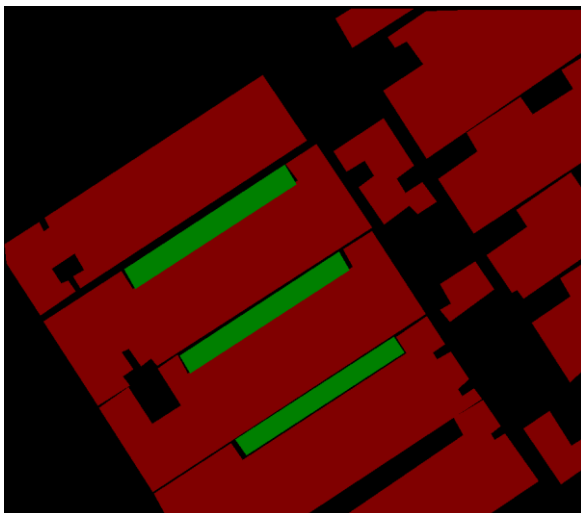
Multilayered Workflow

- Split *Swissimage* Dataset into 1000x1000 px tiles
- Using Faster RCNN to identify tiles with Solar Panels
- Letting trained professional experts specify the geometry of a few thousand solar systems using Cloud Contribution Client
- Train Mask RCNN to find geometry in single class paradigm
- Run Inferencing on multi GPU HPC over the total area of Switzerland
- Read/Write on NoSQL Databases
- Train Xception + Random Forest to decide on class type of panel
- GDAL Geoconversion to vector with joined attributes

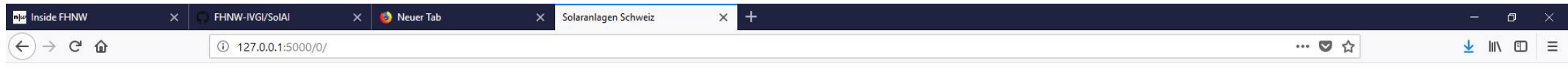
Next Step: Mask RCNN



Instanciación



Cloud Contribution Client for Labelling



Weiter

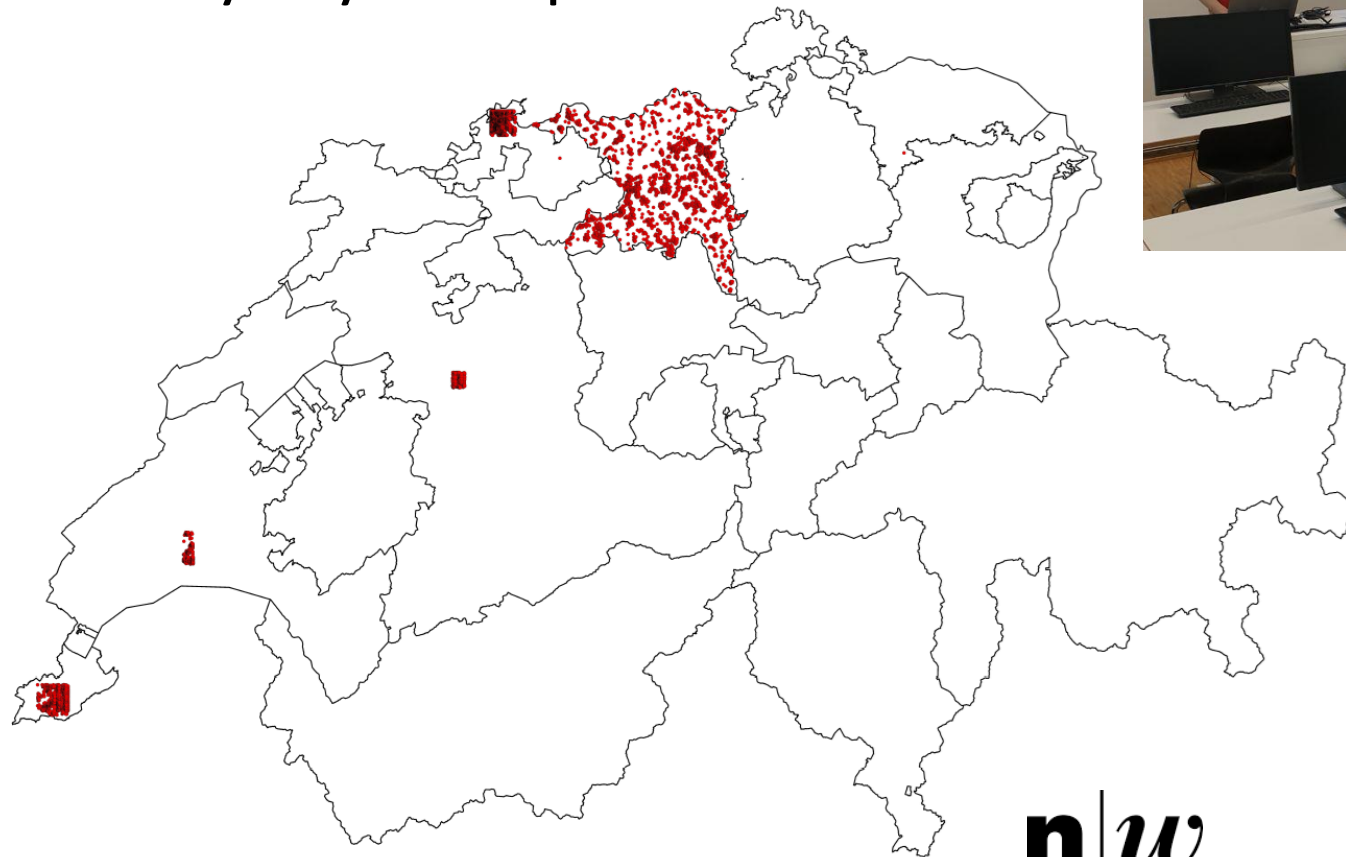
Links Klick um ein Polygon zu zeichnen.
Rechts Klick um das Polygon zu schliessen.

Code-Sprint: Europython 2019

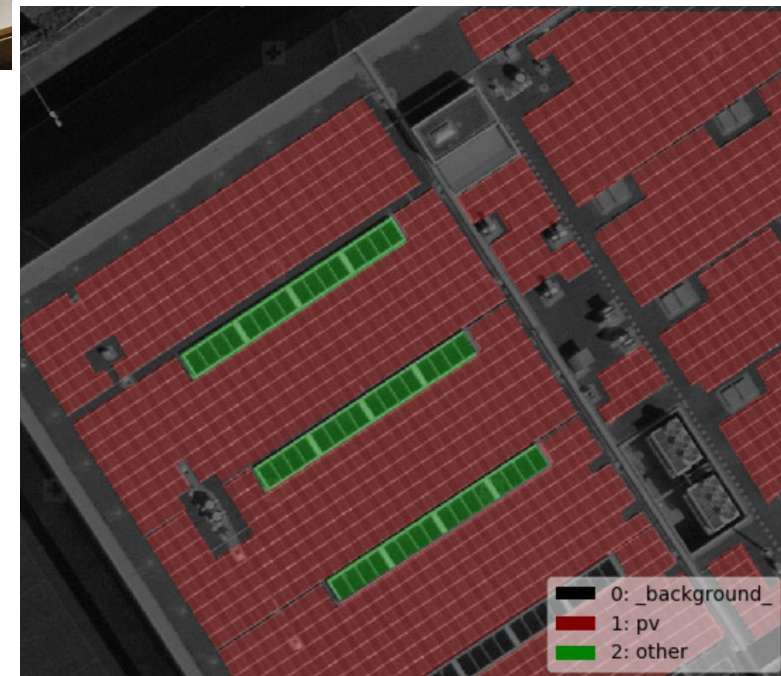


Labelling Workshop

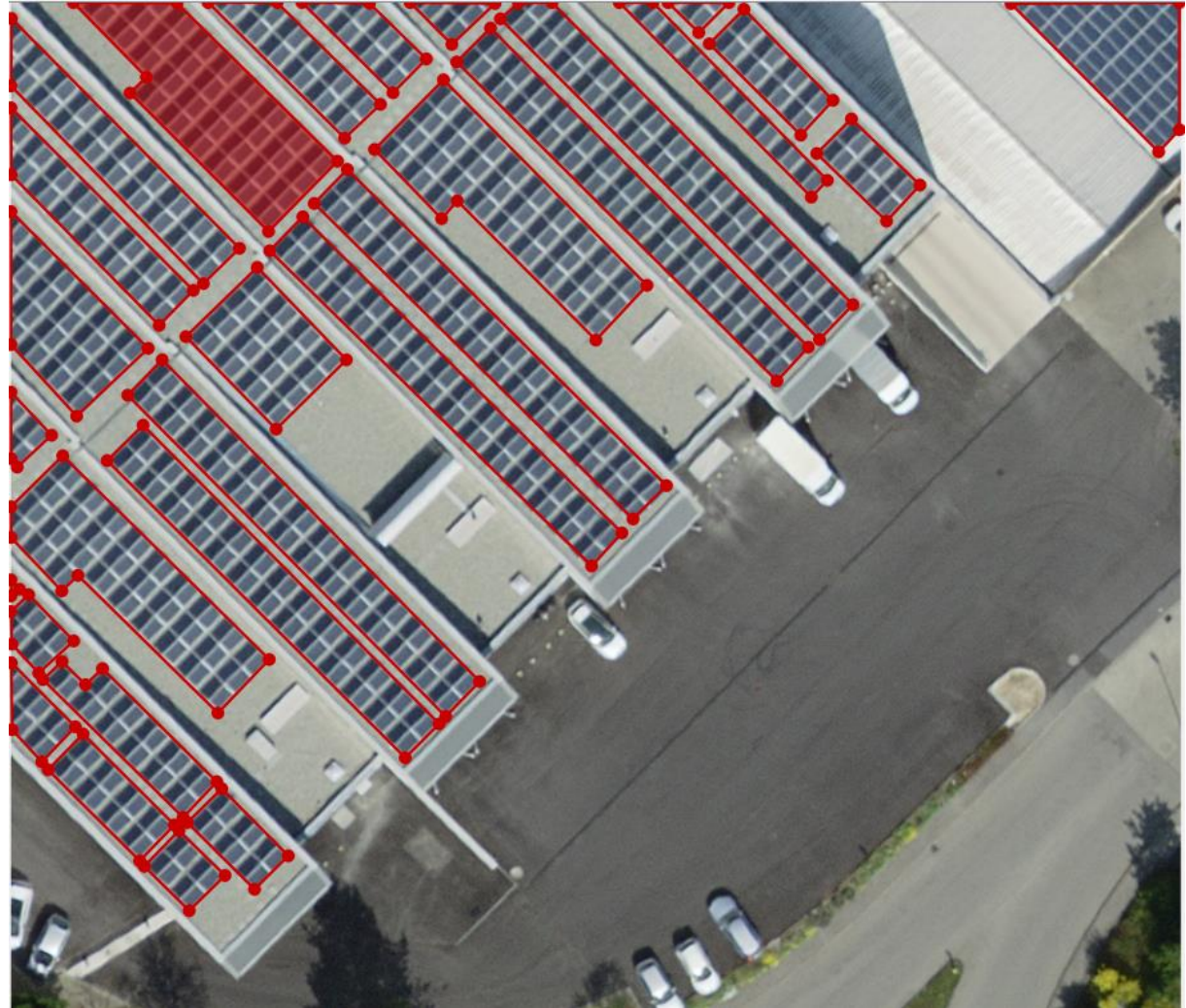
- 7'839 Image Tiles
- 31'401 Polygons (22K PV)
- 5 Days by 10 Experts



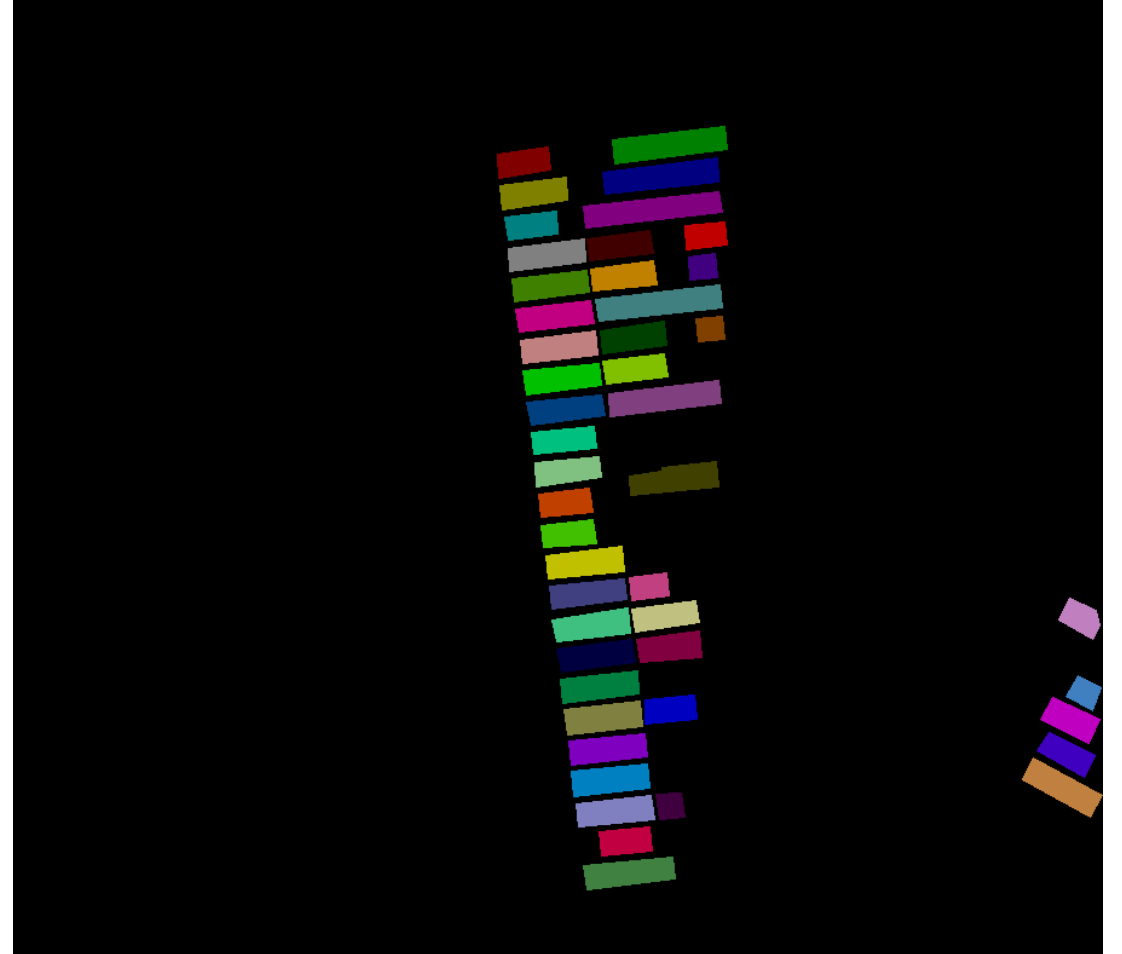
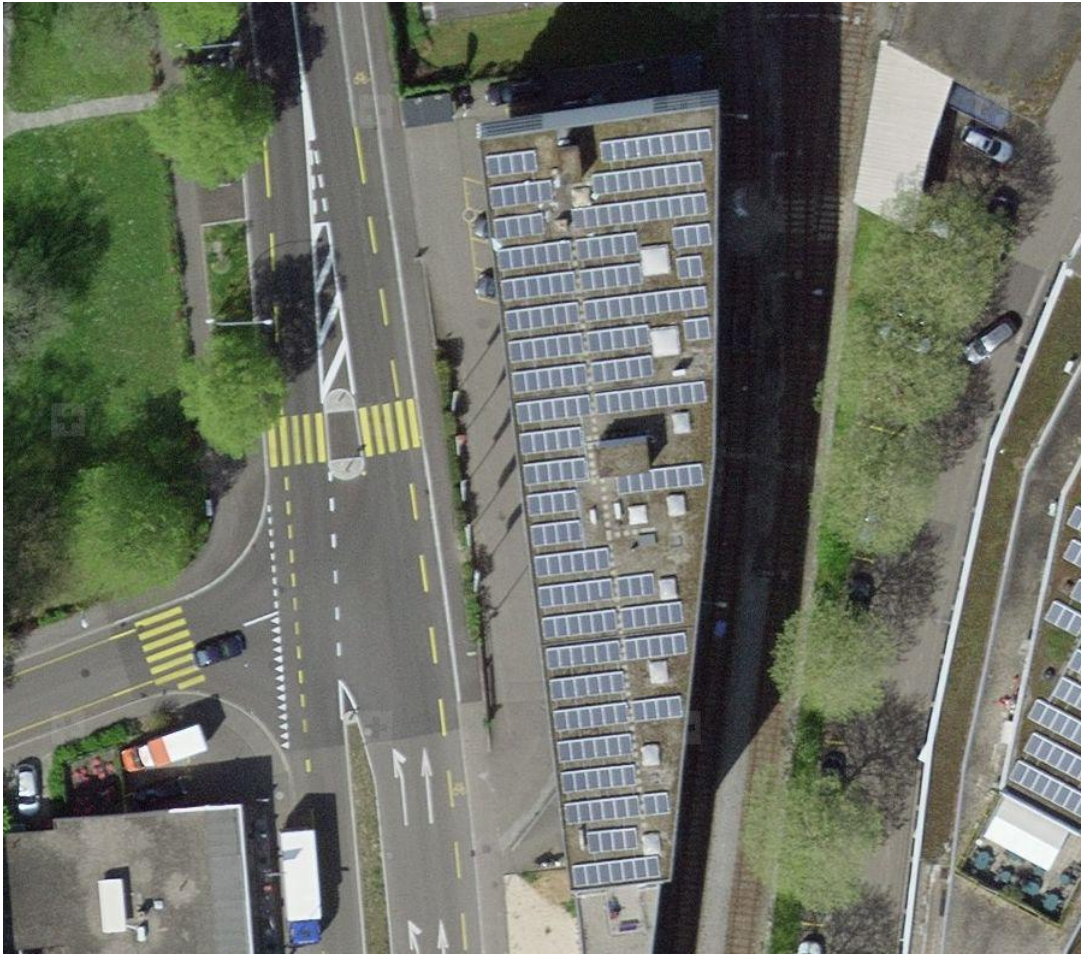
n|w



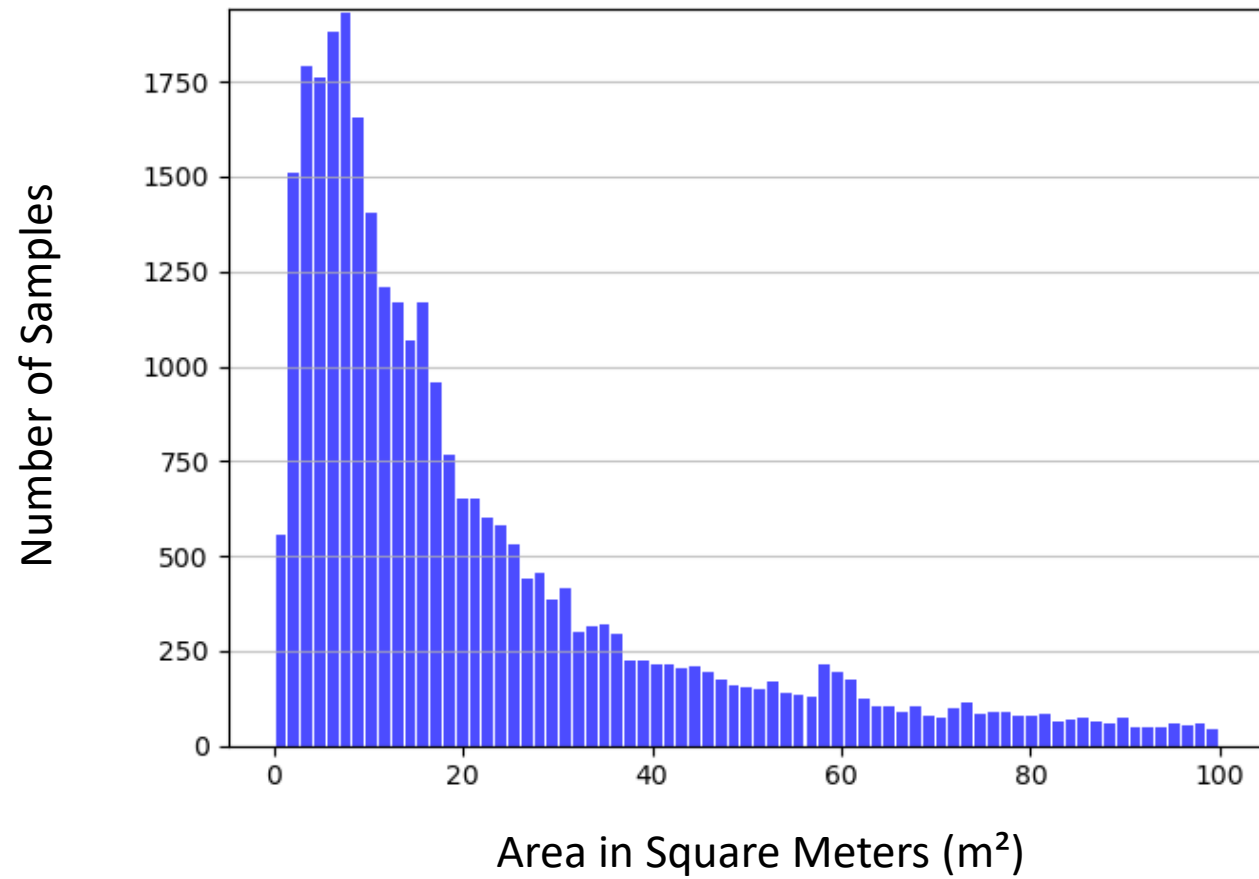
Images 1 & 2 of 7'839



Generating Masks as PNGs



Polygon Size Distribution of Solar Installations



Framework Selection

	PyTorch	TensorFlow
Total Time	160 ms	197 ms
Preprocessing	48 ms	22 ms
Forward prop	31 ms	55 ms
Backward prop	33 ms	120 ms (parallel reduce)
All reduce	48 ms (reduce+broadcast)	
Images	Raw image	TFrecord
kernel	bn_fw_tr_1C11_single_read	bn_fw_tr_1C11_kernel_new

Source:
Anto John,
Oct 2018,
IBM Developer Blog

PyTorch

- Open Source Library for Machine Learning (BSD-License)
- Based on Torch Framework (Lua, C++, CUDA), published in 2002
- PyTorch was published in October 2016
- Main Developers are the AI R&D Teams of Facebook
- Development with Python

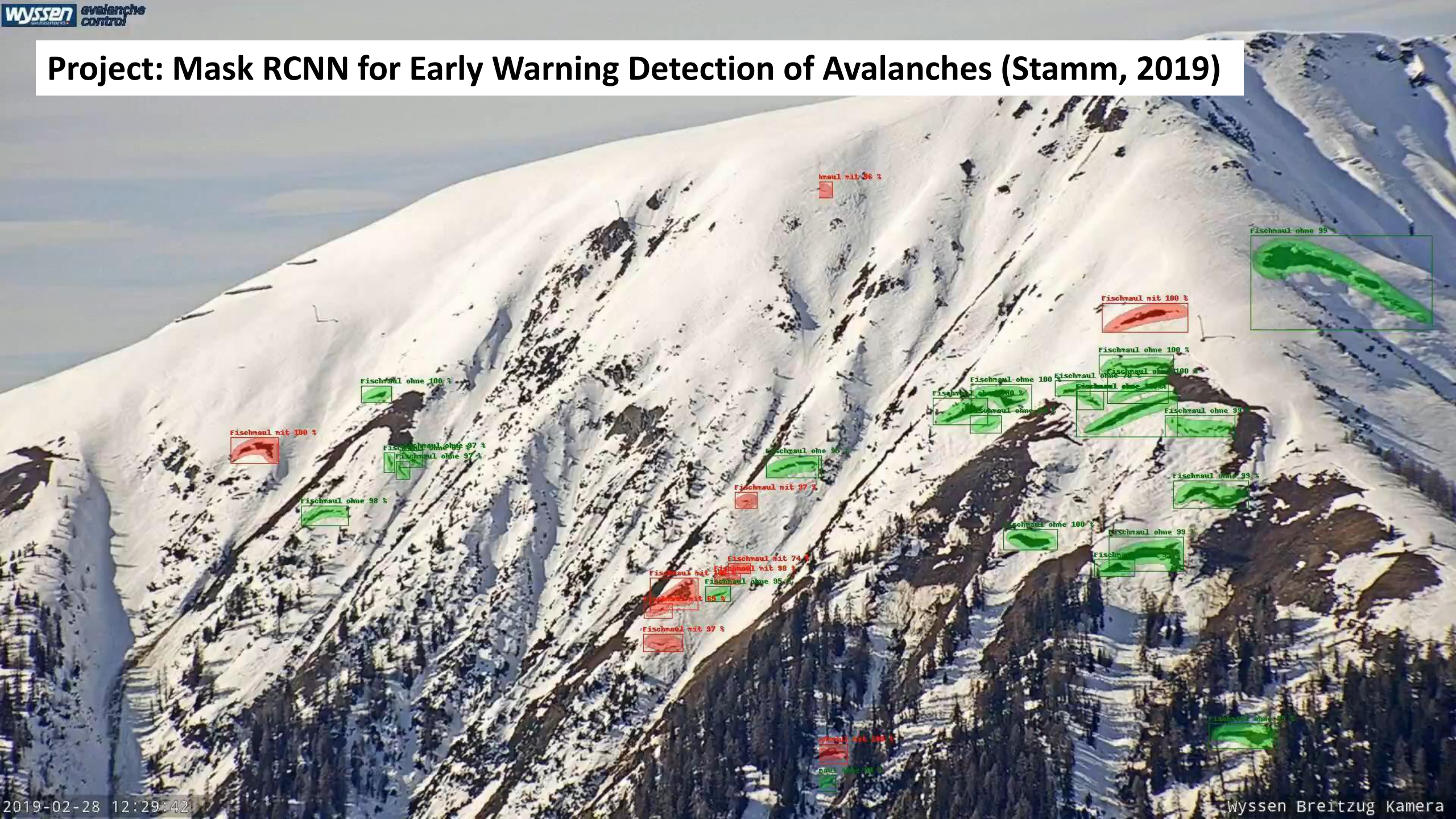
Advantages:

- Pythonic Interface
- GPU Support with nice Interface
 - `a.from_numpy/a.numpy` torch tensor bridge
 - Pretrained Models available (Torchvision)
 - Multiple Optimizers (SGD/Adam/etc.)

```
# construct an optimizer
params = [p for p in model.parameters() if p.requires_grad]
optimizer = torch.optim.SGD(params, lr=0.005,
                             momentum=0.9, weight_decay=0.0005)

# and a Learning rate scheduler which decreases the Learning rate by
# 10x every 3 epochs
lr_scheduler = torch.optim.lr_scheduler.StepLR(optimizer,
                                                step_size=3,
                                                gamma=0.1)
```

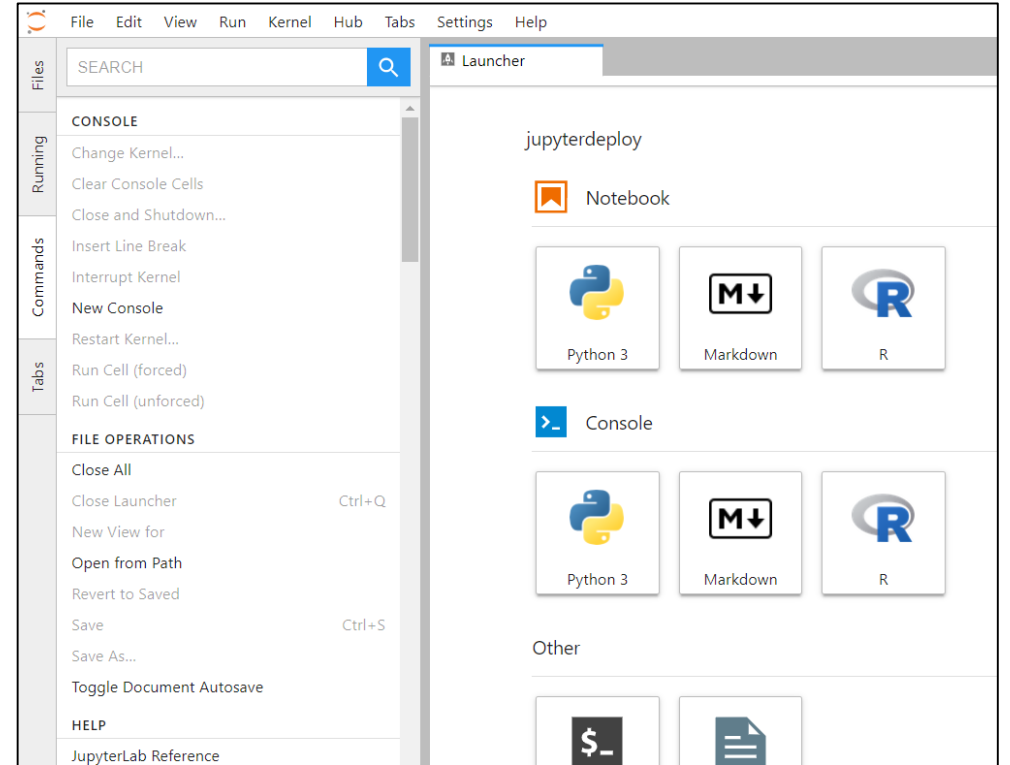

Project: Mask RCNN for Early Warning Detection of Avalanches (Stamm, 2019)



Hardware: HPE Apollo 6500



- 48 cores
- **192 GB RAM**
- Attached to **120 TB HD** (~1 GB/s)



4x



Nvidia Tesla V100 SXM2

- 21 Billion transistors
- 5120 CUDA-cores
- 900 GB/s Mem-Bandwidth
- 12nm
- 300W

- **JupyterHub** using:
- Python 3.7 – Kernel
- Python 3.6 – Kernel
- R-Kernel
- Custom Kernels

Running Pytorch in Jupyterlab



The screenshot displays the JupyterLab interface. On the left, a file browser shows the directory structure of a workspace, including folders like 'Labels_auswahl', 'LabMasks', and 'output', and files like 'coco_eval.py' and 'utils.py'. The main area is a terminal window showing the output of a PyTorch training script. The logs include training progress for multiple epochs, with metrics such as loss, accuracy, and time. The final output shows averaged statistics and IoU metrics for different areas and maxDets.

```
Epoch: [5] [2900/3800] eta: 0:03:04 lr: 0.000500 loss: 0.2386 (0.3399) loss_classifier: 0.0828 (0.1089) loss_box_reg: 0.0170 (0.0306) loss_mask: 0.1218 (0.1599) loss_objectness: 0.0049 (0.0130) loss_rpn_box_reg: 0.0069 (0.0275) time: 0.1696 data: 0.0073 max mem: 3399
Epoch: [5] [3000/3800] eta: 0:02:43 lr: 0.000500 loss: 0.2591 (0.3401) loss_classifier: 0.0853 (0.1088) loss_box_reg: 0.0184 (0.0306) loss_mask: 0.1368 (0.1600) loss_objectness: 0.0110 (0.0131) loss_rpn_box_reg: 0.0103 (0.0276) time: 0.1742 data: 0.0068 max mem: 3399
Epoch: [5] [3100/3800] eta: 0:02:22 lr: 0.000500 loss: 0.2623 (0.3397) loss_classifier: 0.0803 (0.1086) loss_box_reg: 0.0110 (0.0305) loss_mask: 0.1294 (0.1598) loss_objectness: 0.0063 (0.0130) loss_rpn_box_reg: 0.0069 (0.0276) time: 0.1722 data: 0.0069 max mem: 3399
Epoch: [5] [3200/3800] eta: 0:02:01 lr: 0.000500 loss: 0.2465 (0.3402) loss_classifier: 0.0677 (0.1087) loss_box_reg: 0.0092 (0.0306) loss_mask: 0.1424 (0.1600) loss_objectness: 0.0104 (0.0131) loss_rpn_box_reg: 0.0052 (0.0277) time: 0.1679 data: 0.0059 max mem: 3399
Epoch: [5] [3300/3800] eta: 0:01:41 lr: 0.000500 loss: 0.2641 (0.3401) loss_classifier: 0.0798 (0.1086) loss_box_reg: 0.0097 (0.0307) loss_mask: 0.1396 (0.1598) loss_objectness: 0.0072 (0.0132) loss_rpn_box_reg: 0.0049 (0.0278) time: 0.1740 data: 0.0082 max mem: 3399
Epoch: [5] [3400/3800] eta: 0:01:20 lr: 0.000500 loss: 0.2856 (0.3395) loss_classifier: 0.0830 (0.1084) loss_box_reg: 0.0138 (0.0305) loss_mask: 0.1439 (0.1597) loss_objectness: 0.0068 (0.0131) loss_rpn_box_reg: 0.0112 (0.0278) time: 0.1708 data: 0.0076 max mem: 3399
Epoch: [5] [3500/3800] eta: 0:01:00 lr: 0.000500 loss: 0.2869 (0.3389) loss_classifier: 0.0980 (0.1080) loss_box_reg: 0.0183 (0.0304) loss_mask: 0.1396 (0.1599) loss_objectness: 0.0051 (0.0130) loss_rpn_box_reg: 0.0070 (0.0275) time: 0.1715 data: 0.0072 max mem: 3399
Epoch: [5] [3600/3800] eta: 0:00:40 lr: 0.000500 loss: 0.2525 (0.3382) loss_classifier: 0.0795 (0.1078) loss_box_reg: 0.0140 (0.0303) loss_mask: 0.1319 (0.1596) loss_objectness: 0.0108 (0.0132) loss_rpn_box_reg: 0.0095 (0.0274) time: 0.1700 data: 0.0072 max mem: 3399
Epoch: [5] [3700/3800] eta: 0:00:20 lr: 0.000500 loss: 0.3096 (0.3385) loss_classifier: 0.1112 (0.1080) loss_box_reg: 0.0186 (0.0303) loss_mask: 0.1488 (0.1594) loss_objectness: 0.0110 (0.0131) loss_rpn_box_reg: 0.0092 (0.0276) time: 0.1775 data: 0.0091 max mem: 3399
Epoch: [5] [3799/3800] eta: 0:00:00 lr: 0.000500 loss: 0.2633 (0.3388) loss_classifier: 0.0910 (0.1081) loss_box_reg: 0.0167 (0.0304) loss_mask: 0.1388 (0.1593) loss_objectness: 0.0072 (0.0131) loss_rpn_box_reg: 0.0103 (0.0278) time: 0.1773 data: 0.0084 max mem: 3399
Epoch: [5] Total time: 0:12:39 (0.1999 s / it)
Start: Evaluate 13:55:11 08.06.2020
wait for coco evaluator...
creating index...
index created!
time for indexing: 0:10:06.710710
Evaluate: [ 0/239] eta: 0:01:47 model_time: 0.0531 (0.0531) evaluator_time: 0.0184 (0.0184) time: 0.4503 data: 0.3741 max mem: 3399
Evaluate: [100/239] eta: 0:00:15 model_time: 0.0484 (0.0659) evaluator_time: 0.0225 (0.0313) time: 0.0997 data: 0.0038 max mem: 3399
Evaluate: [200/239] eta: 0:00:04 model_time: 0.0615 (0.0704) evaluator_time: 0.0226 (0.0338) time: 0.1353 data: 0.0037 max mem: 3399
Evaluate: [238/239] eta: 0:00:00 model_time: 0.0489 (0.0701) evaluator_time: 0.0223 (0.0329) time: 0.1011 data: 0.0037 max mem: 3399
Evaluate: Total time: 0:00:27 (0.1132 s / it)
Averaged stats: model_time: 0.0489 (0.0701) evaluator_time: 0.0223 (0.0329)
Accumulating evaluation results...
DONE (t=0.09s).
Accumulating evaluation results...
DONE (t=0.08s).
IoU metric: bbox
Average Precision (AP) @[ IoU=0.50:0.95 | area= all | maxDets=100 ] = 0.373
Average Precision (AP) @[ IoU=0.50 | area= all | maxDets=100 ] = 0.517
Average Precision (AP) @[ IoU=0.75 | area= all | maxDets=100 ] = 0.438
Average Precision (AP) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = 0.283
Average Precision (AP) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = 0.439
Average Precision (AP) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = 0.402
Average Recall (AR) @[ IoU=0.50:0.95 | area= all | maxDets= 1 ] = 0.265
Average Recall (AR) @[ IoU=0.50:0.95 | area= all | maxDets= 10 ] = 0.585
Average Recall (AR) @[ IoU=0.50:0.95 | area= all | maxDets=100 ] = 0.668
Average Recall (AR) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = 0.612
Average Recall (AR) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = 0.710
```


Extending Models & Multi GPU Support

- In PyTorch we can add new custom datasets for object detection and instance segmentation by inheriting the class `torch.utils.data.Dataset`
- PyTorch provides an example for that: https://pytorch.org/tutorials/intermediate/torchvision_tutorial.html



- The tricky part is to support our 4 GPUs. Data Parallelism is implemented using `torch.nn.DataParallel`.
- PyTorch doesn't really provide many examples for multi-GPU, so it was a little bit try and error.

inheriting the class `torch.utils.data.Dataset`



```
import os
import numpy as np
import torch
import torch.utils.data
from torchvision import transforms, utils, datasets
from PIL import Image, ImageDraw, ImageFont, ImageOps

class SolarDataset(torch.utils.data.Dataset):
    def __init__(self, root, transforms=None):
        self.root = root
        self.transforms = transforms

        self.imgs = list(sorted(os.listdir(os.path.join(root, "PNGImages"))))
        self.masks = list(sorted(os.listdir(os.path.join(root, "ObjMasks"))))
        self.labs = list(sorted(os.listdir(os.path.join(root, "LabMasks"))))

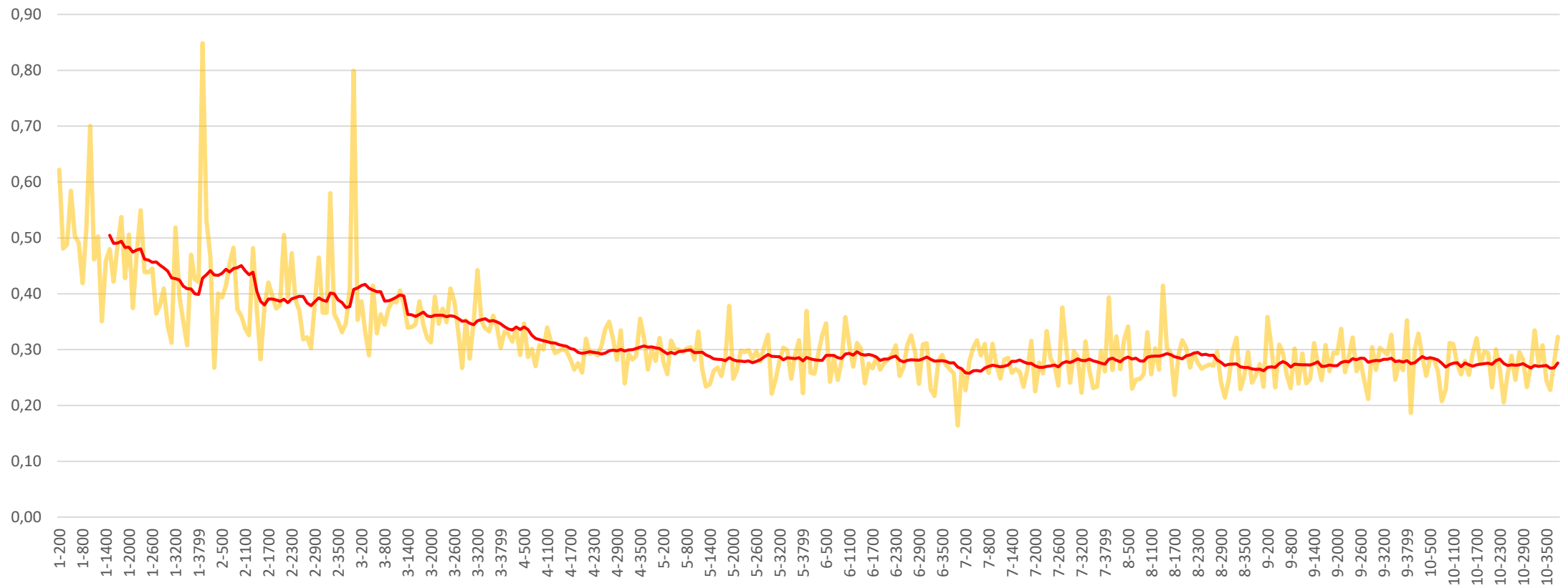
    def __getitem__(self, idx):
        img_path = os.path.join(self.root, "PNGImages", self.imgs[idx])
        mask_path = os.path.join(self.root, "ObjMasks", self.masks[idx])
        labels_path = os.path.join(self.root, "LabMasks", self.labs[idx])

        img = Image.open(img_path).convert("RGB")
```

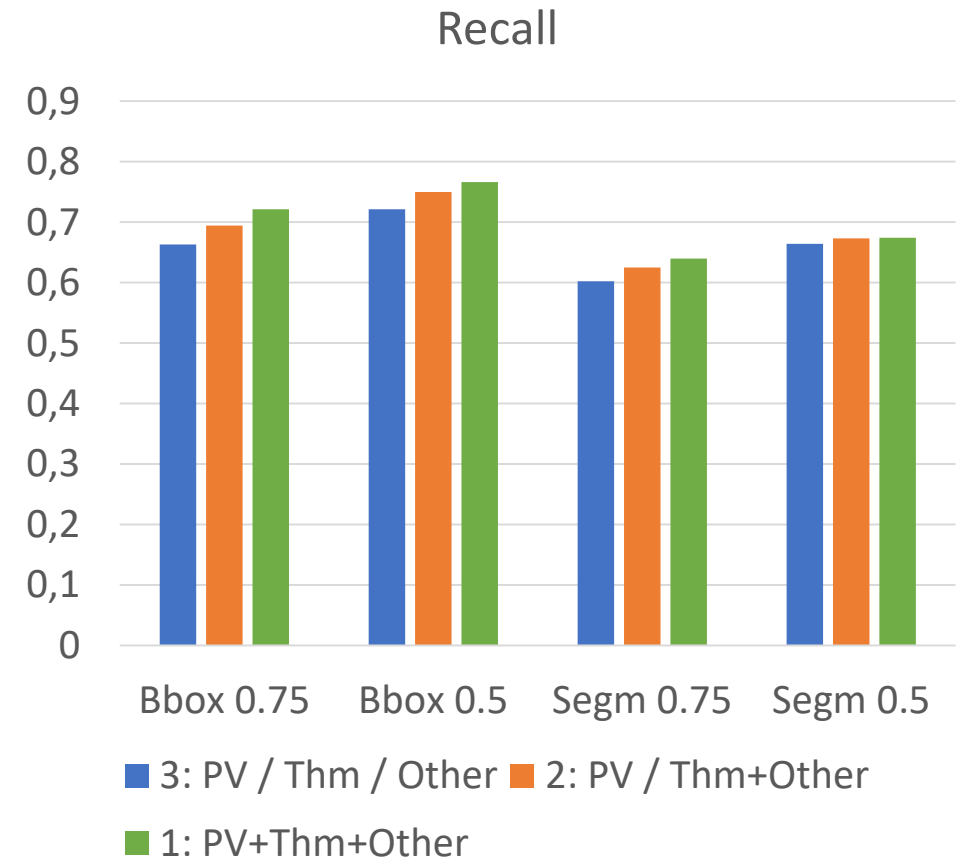
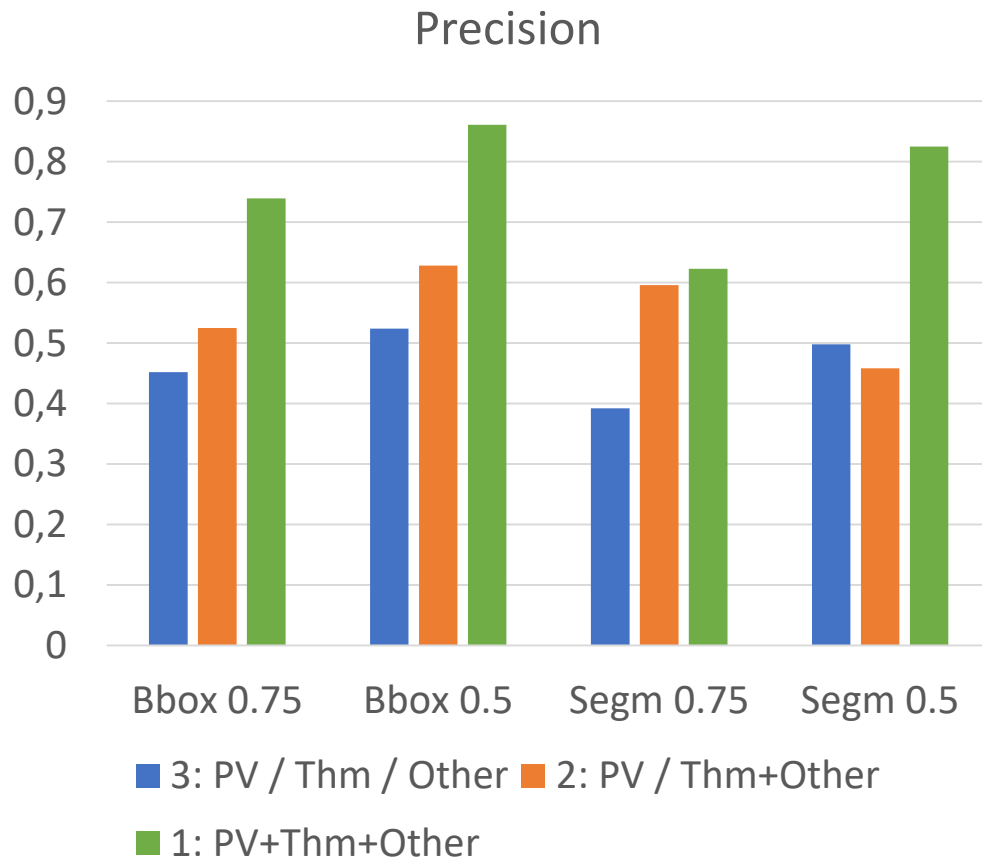
...

Loss Graph Needs ± 6 Epochs

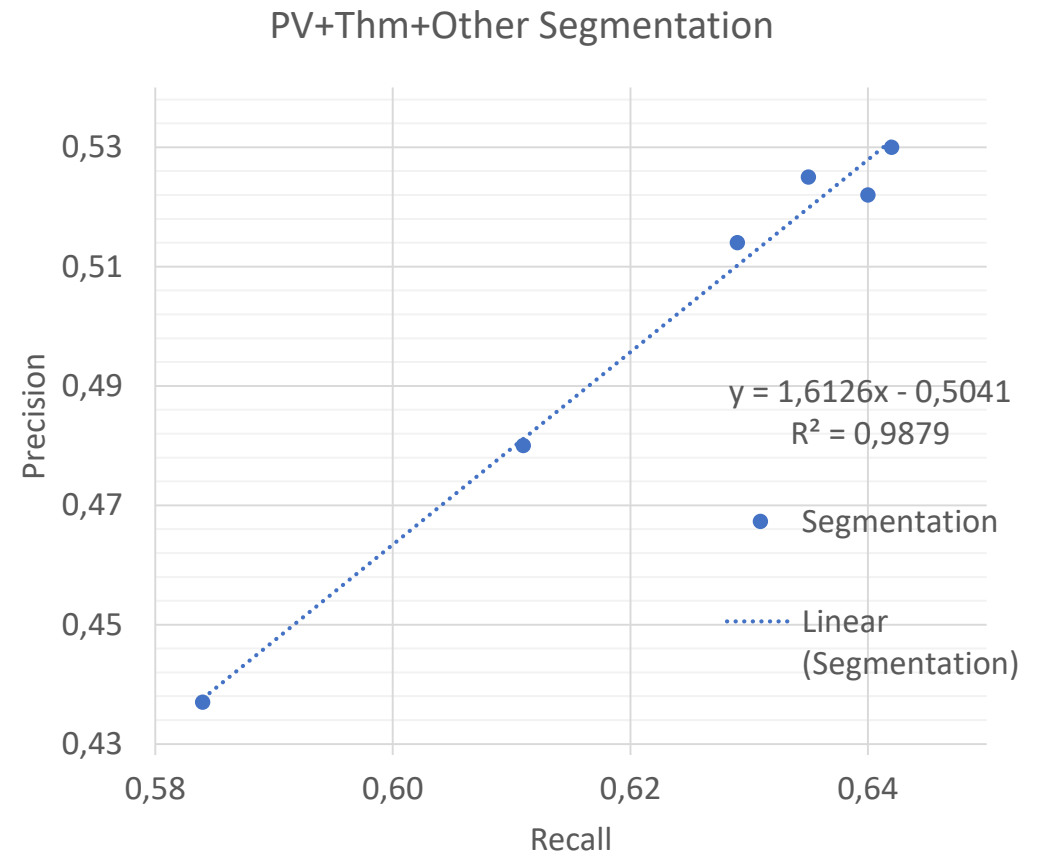
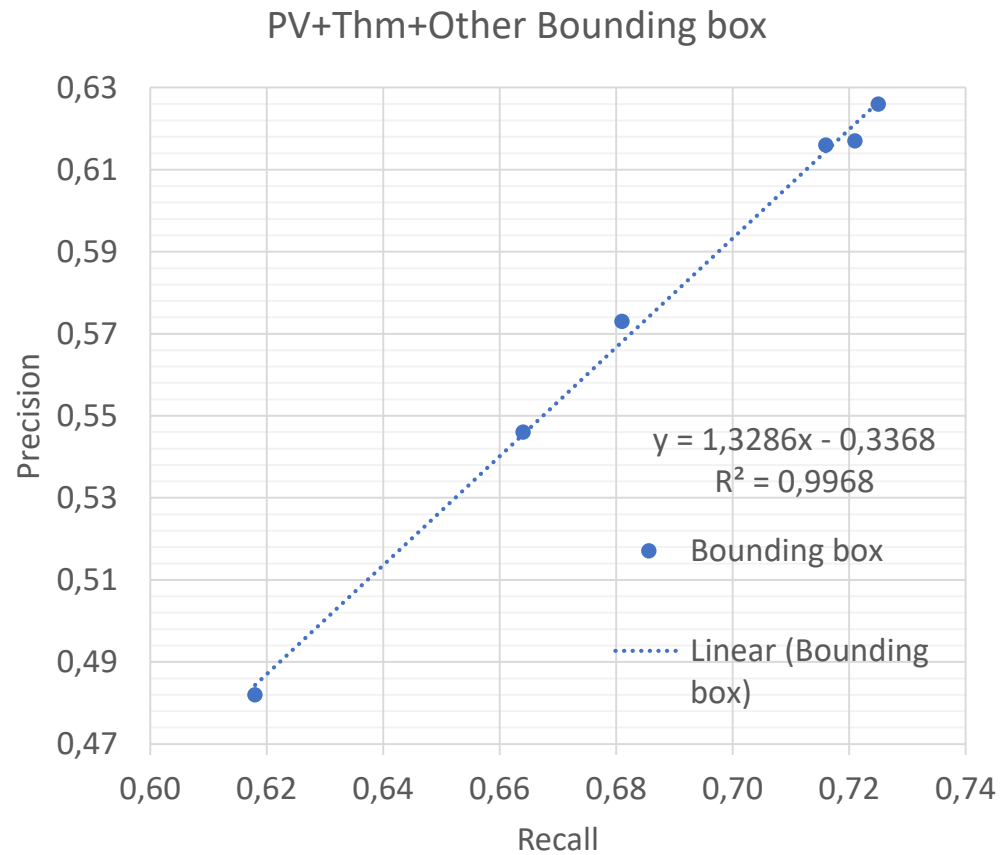
Model Run with Separate Classes (PV // THM // Other)



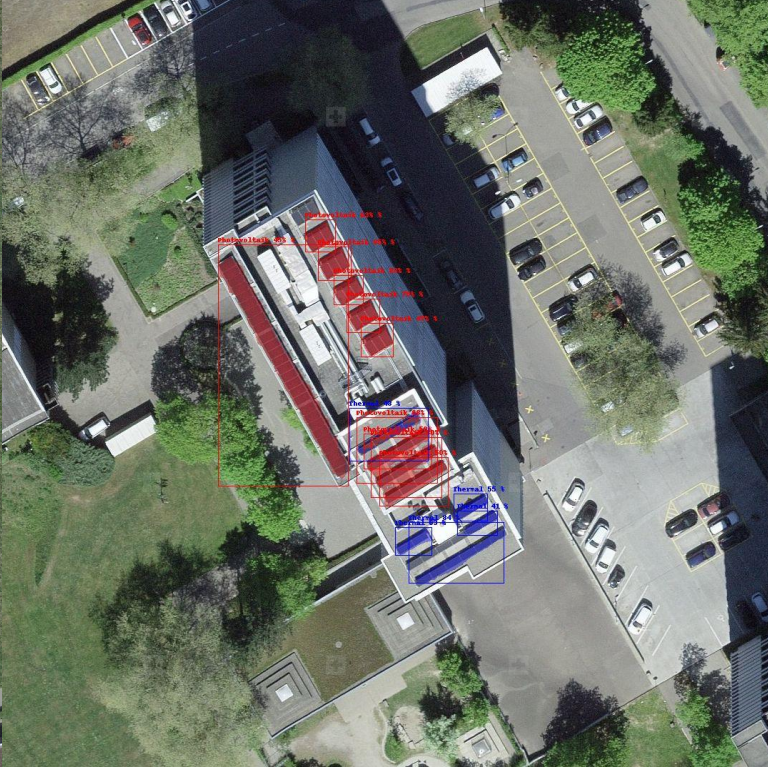
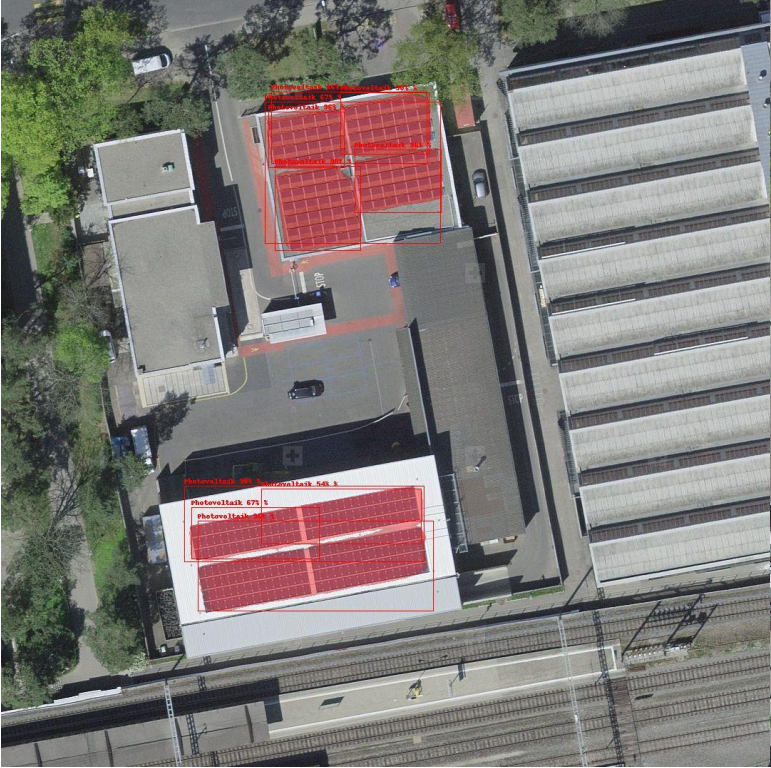
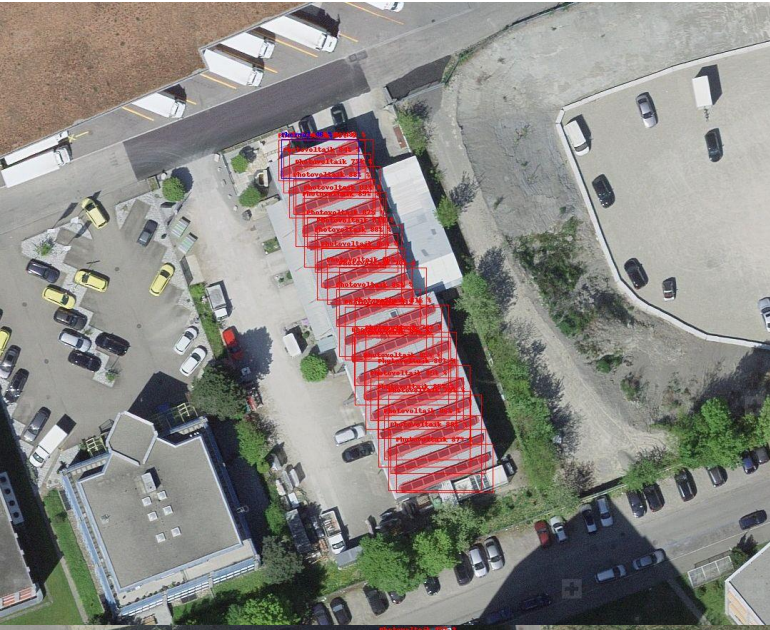
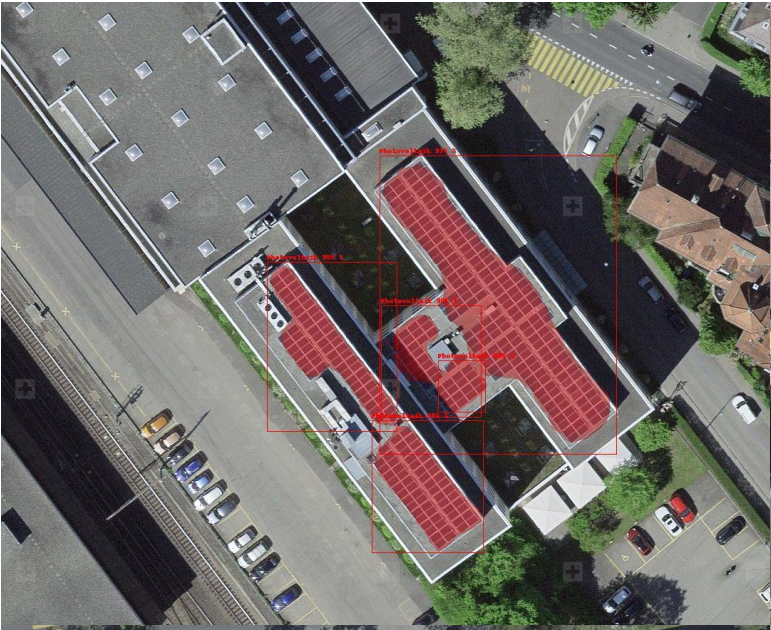
Preliminary Results: Metrics for Class Combination



Statistical Linearity



Results



Challenge: Small Panels?

- Single Class Paradigm (PV+Thm+Other) including no modules smaller than 3qm did not increase Precision or Recall
- Cleaning up the Labels is more important

Challenge Labels: Class «Others»

- These elements probably are photovoltaic panels but display somewhat difficult characteristics



Challenge Labels: Class «Other»

- These are most likely NOT solar panels



Challenge Labels: Class «Others»

- Difficult



Challenge: Labelling Mistakes



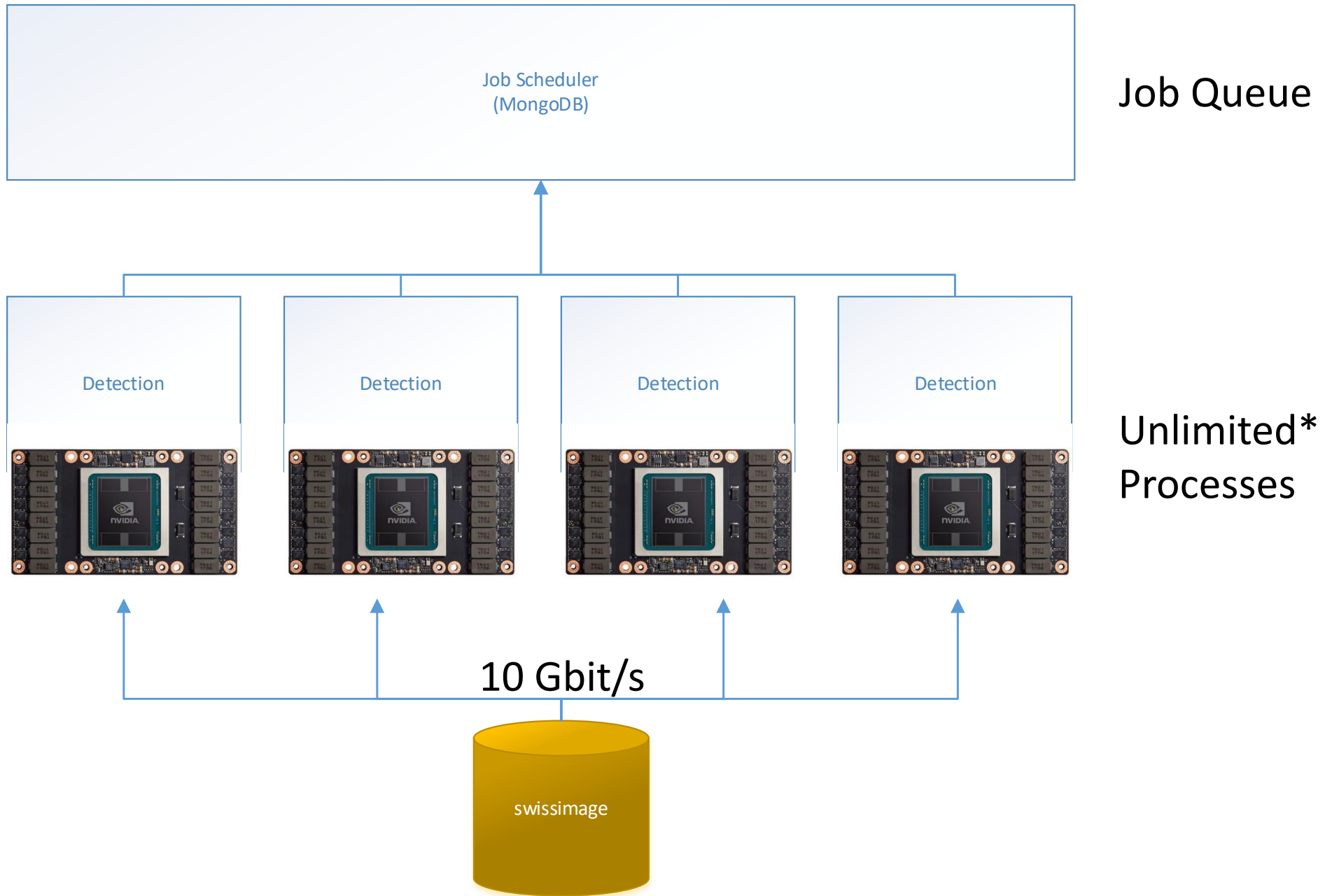
n | *w*



Computational Load for a Single Run over Complete Switzerland

- 4 Million Images with 2-3 Mbyte
- Inferencing & IO Operation Duration per Image:

CPU (44 Cores):	2.1 Seconds (100 Days)
1 GPU:	1.6 Seconds
4 GPUs:	1.0 Seconds (46 Days)



Optimization

- **Currently a Country Level Inferencing Run Takes ± 10 Days**
- Still Potential with Model Optimizations
- Inferencing Times for Tensorflow Possibly Faster but Requires TF Records
- Increase the Load on GPUs
- Hybrid CPUs/GPUs on Multiple HPCs

More Potential: Optimization for higher GPU Load

PID	USER	PR	NI	VIRT	RES	SHR	S	%CPU	%MEM	TIME+	COMMAND
67579	mchrist+	20	0	25.701g	4.774g	622564	R	202.0	2.5	7:16.34	python
67436	mchrist+	20	0	22.608g	2.857g	519344	R	196.4	1.5	7:28.21	python
67860	mchrist+	20	0	25.816g	4.811g	622624	R	196.4	2.6	6:51.77	python
67714	mchrist+	20	0	25.752g	4.810g	623620	R	183.8	2.6	6:55.27	python

```

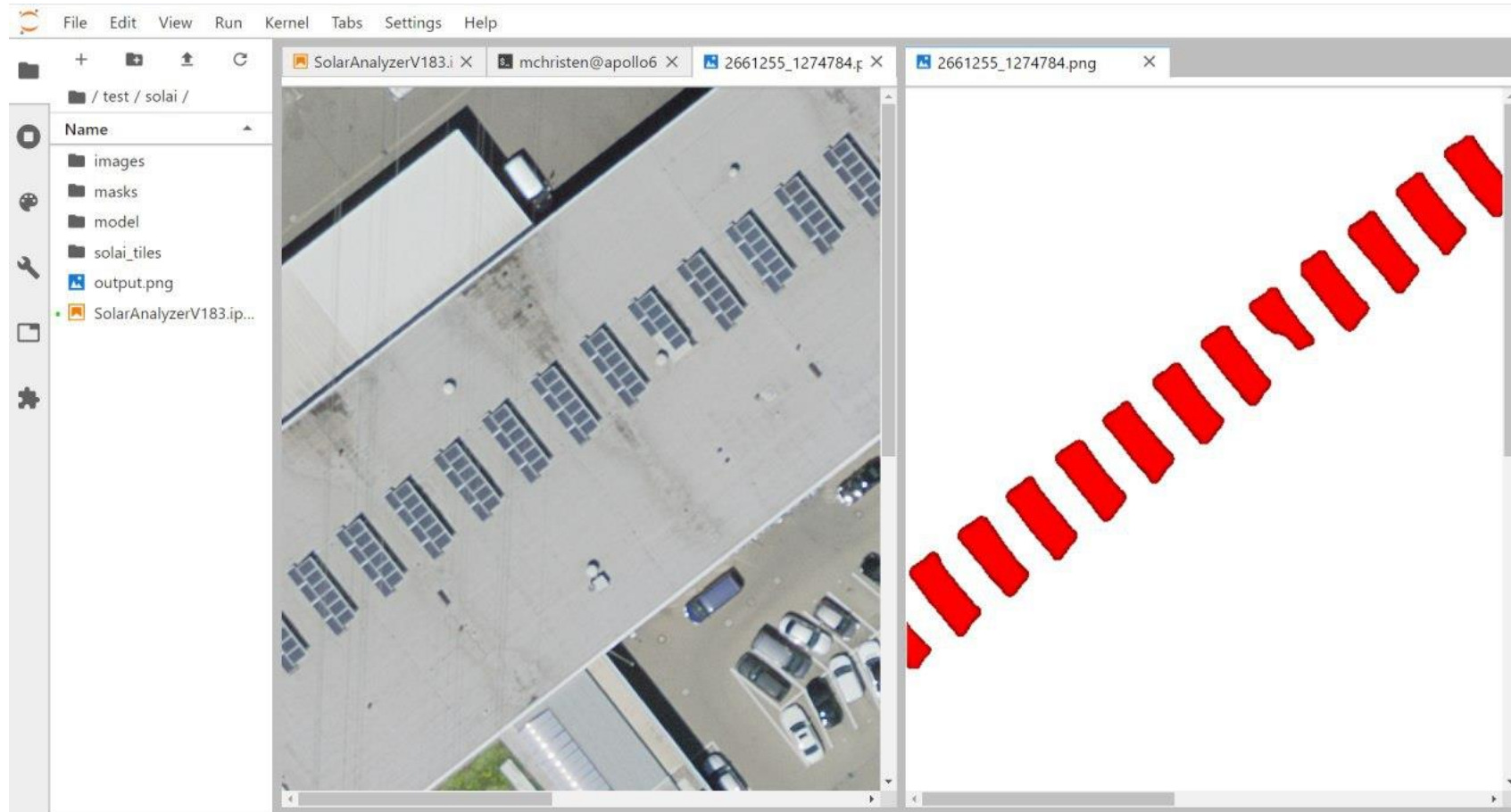
+-----+
| NVIDIA-SMI 418.87.00      Driver Version: 418.87.00      CUDA Version: 10.1      |
+-----+
| GPU   Name                Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf    Pwr:Usage/Cap|      Memory-Usage | GPU-Util  Compute M. |
+-----+-----+
|    0   Tesla V100-SXM2...    Off      | 00000000:15:00.0 Off  |             0         |
| N/A   35C    P0     55W / 300W | 5706MiB / 32480MiB |      0%      Default  |
+-----+-----+
|    1   Tesla V100-SXM2...    Off      | 00000000:16:00.0 Off  |             0         |
| N/A   35C    P0     58W / 300W | 2044MiB / 32480MiB |     13%      Default  |
+-----+-----+
|    2   Tesla V100-SXM2...    Off      | 00000000:3A:00.0 Off  |             0         |
| N/A   34C    P0     55W / 300W | 28316MiB / 32480MiB |     19%      Default  |
+-----+-----+
|    3   Tesla V100-SXM2...    Off      | 00000000:3B:00.0 Off  |             0         |
| N/A   38C    P0     67W / 300W | 6775MiB / 32480MiB |      0%      Default  |
+-----+-----+

```


Outlook

- Pytorch currently just supports ResNet50
 - We want to try out ResNet101 or ResNet+Inception v2 but at the moment we would need Tensorflow for it
 - Trying different sets of optimizers
 - Adapt Learning Rate dynamically
- Some more manual labelling
- Post-Classification Strategies

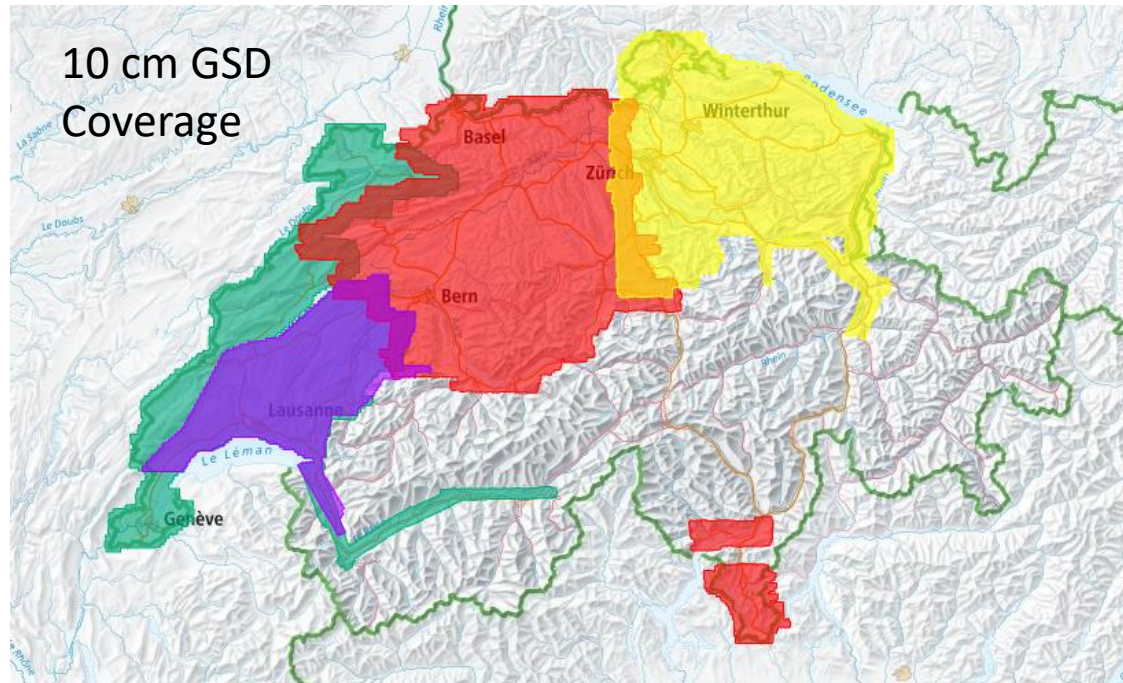
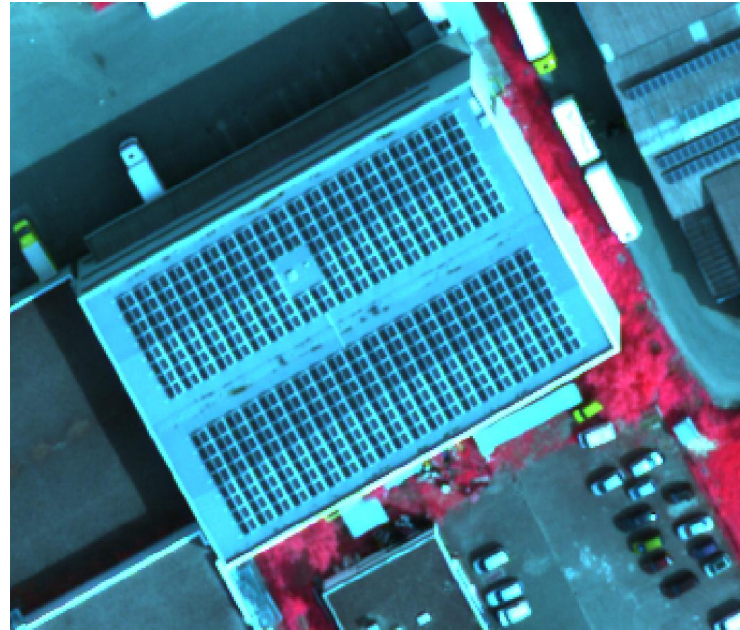
Big Data Inferencing & Classification Workflow



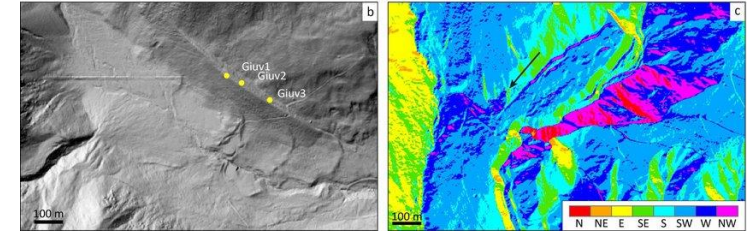
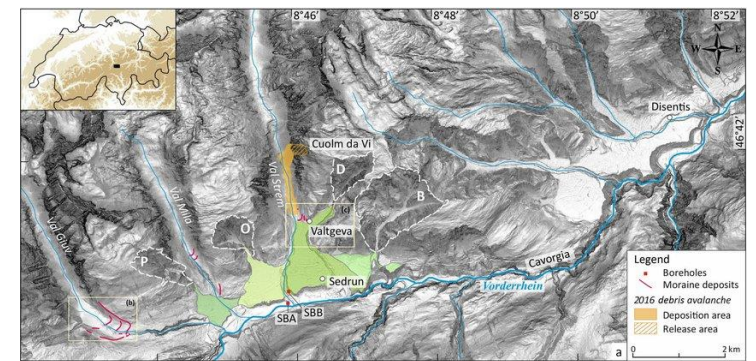
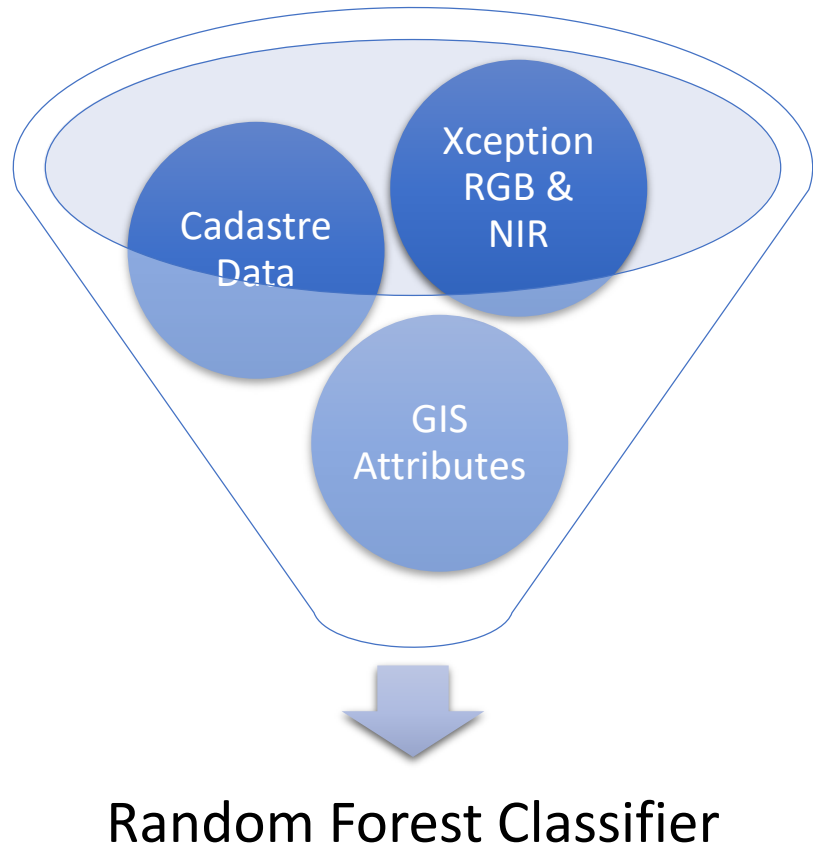
Run multiple models for inferencing, using heuristic measures for edges and segment probability

→ Rasterio / fiona / GDAL
n|w

Include Near-Infrared Data



Random Forest & Xception for Post-Classification



Thank you!

