europython
Jul 23 - 26 2020 Online

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

Adrian Meyer
Institute Geomatics
University of Applied Sciences
Northwestern Switzerland

Fachhochschule Nordwestschweiz
Hochschule für Architektur, Bau und Geomatik

Team

Institute Geomatics @ FHNW

Adrian Meyer
Data Scientist

Prof. Denis Jordan
Statistics & Mathematics

Prof. Martin Christen
Geoinformation & Computer Graphics

Project Partners

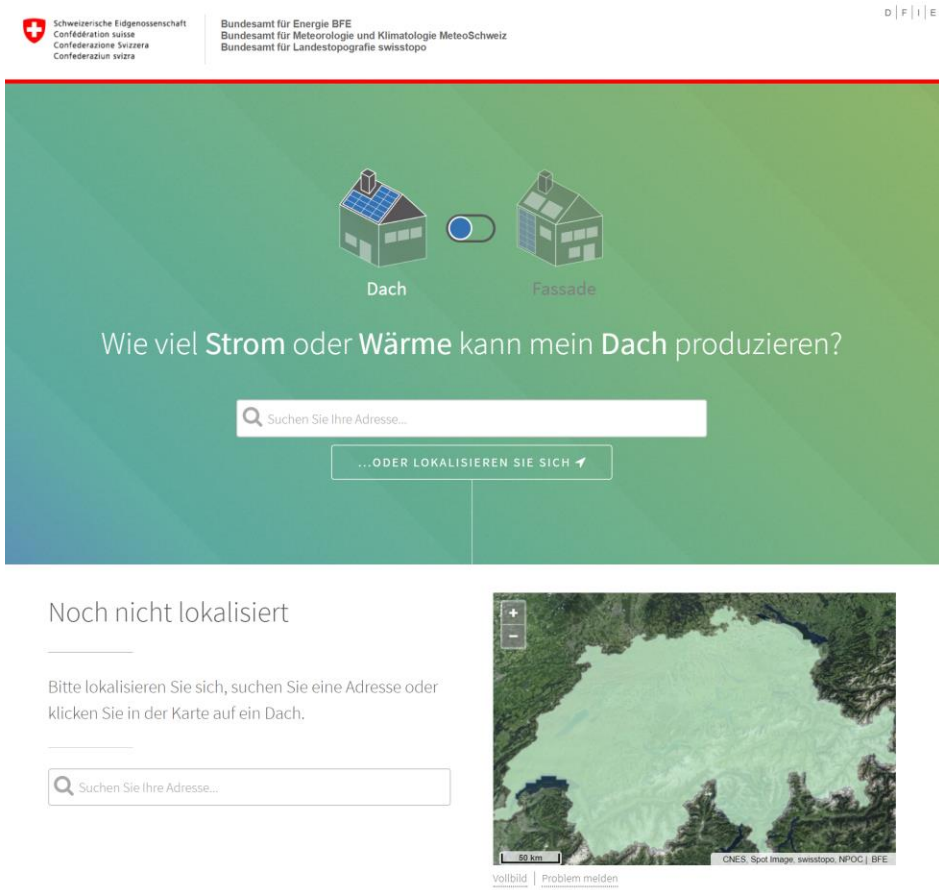
Schweizerische Eidgenossenschaft
Confédération suisse
Confederazione Svizzera
Confederaziun svizra

In collaboration with the cantons

Martin Hertach
Federal Office for Energy (BFE)

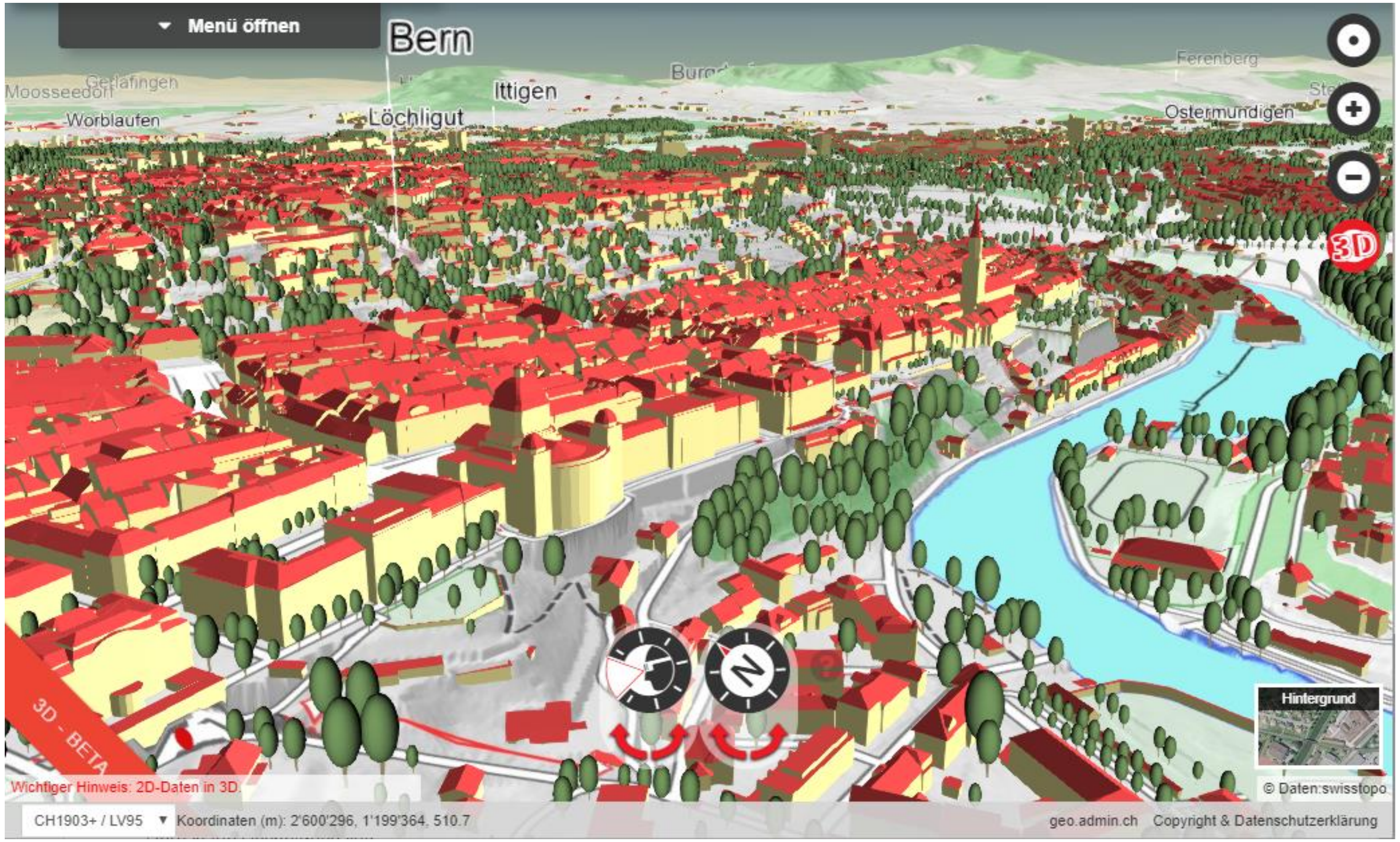
Peter Barmet
Energy Department of Canton Aargau (AG)

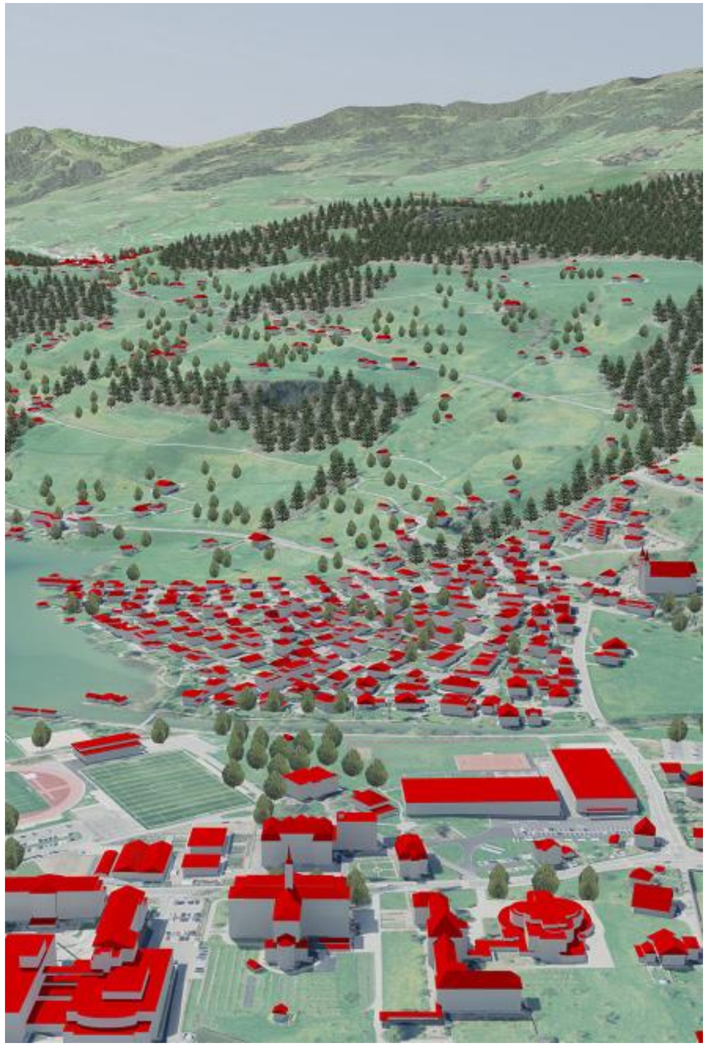
SolAI – Detection of Solar Systems
IGEO/FHNW and Federal Office for Energy (BFE)

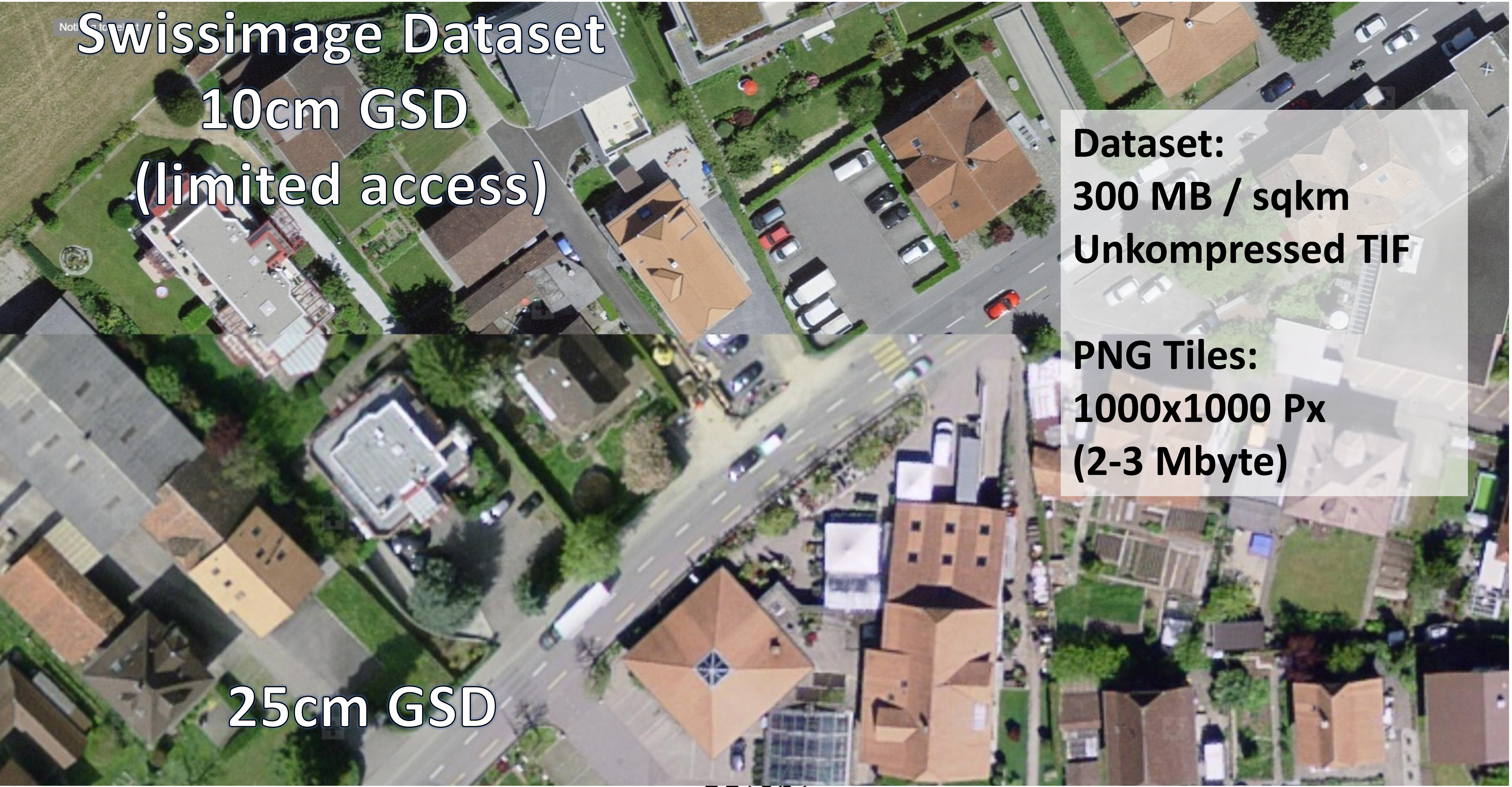


sonnendach.ch

Swiss Buildings 3D Dataset






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

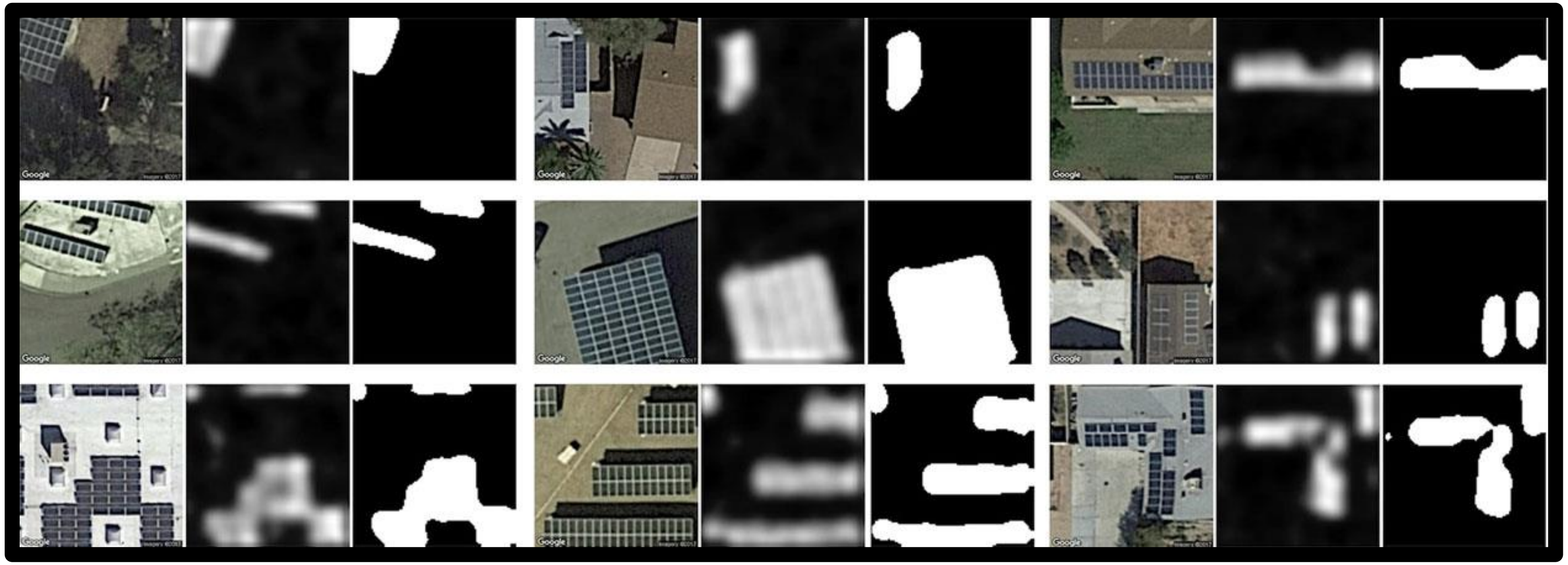
- **We have:**
- Areal imagery (partially in $10cm^2$, partially $25cm^2$ per Pixel)
- Vector data / 3D Data of all roofs in Switzerland
- Roof size and solar potential

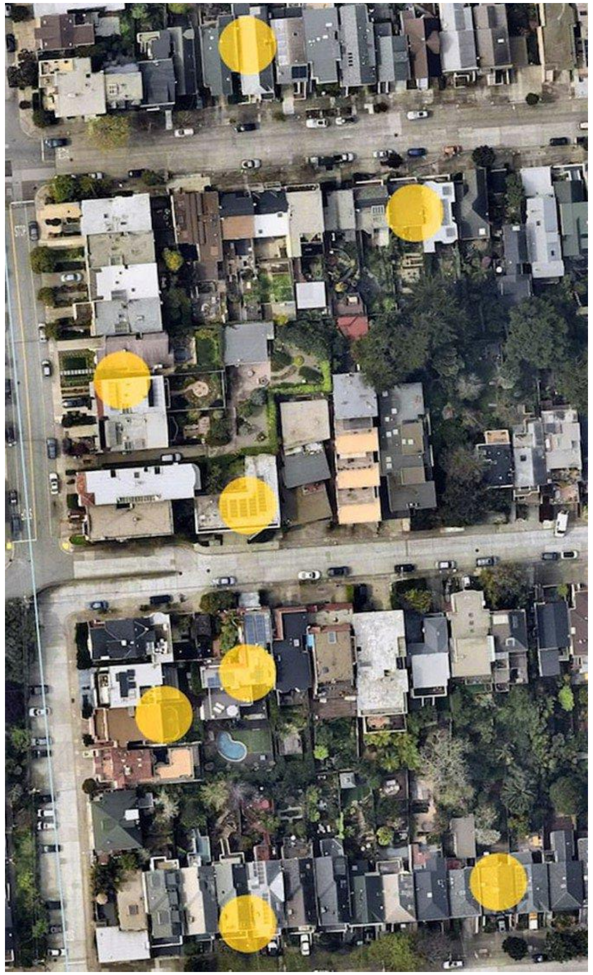
→ There are around 2 million buildings in Switzerland in total

- **We don’t have:**
- Location, Size, and Type of solar panels (PV, Thermal)

Deep Solar (2018)

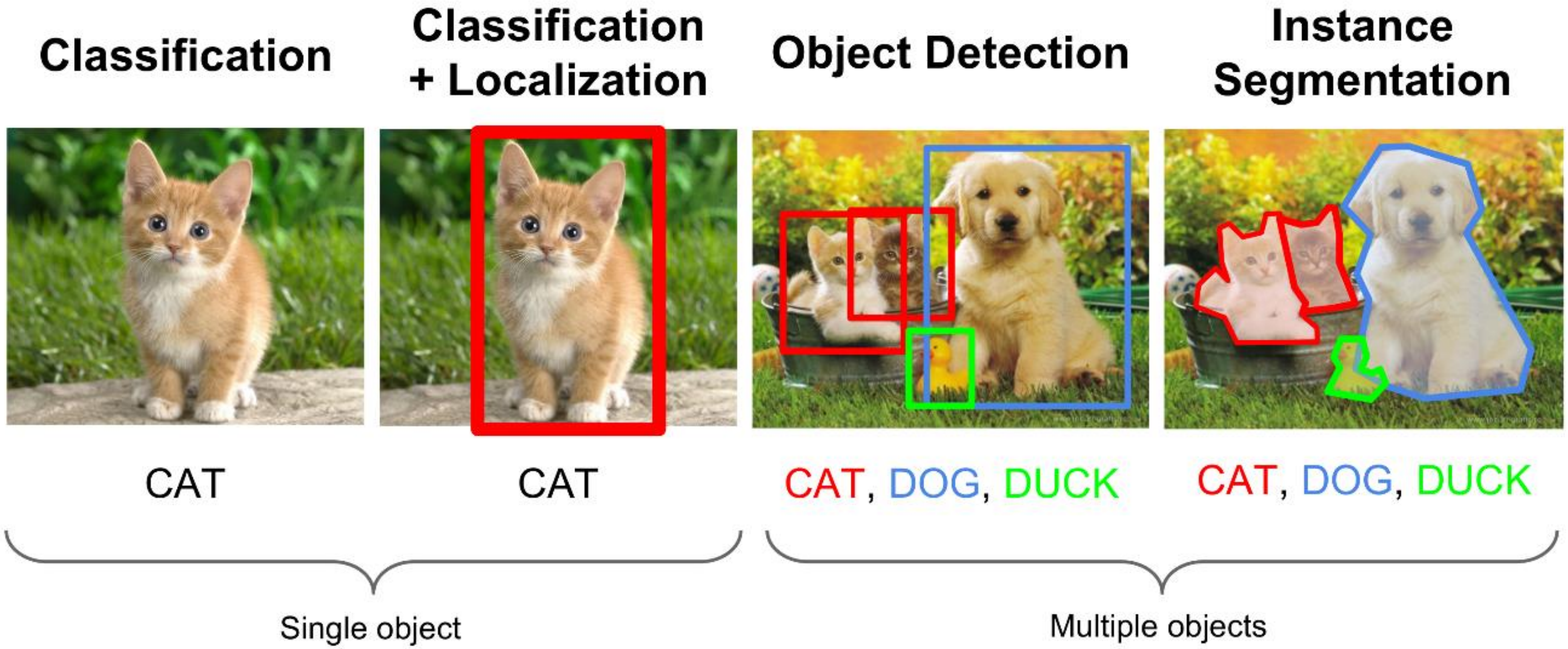
- https://github.com/wangzhecheng/DeepSolar





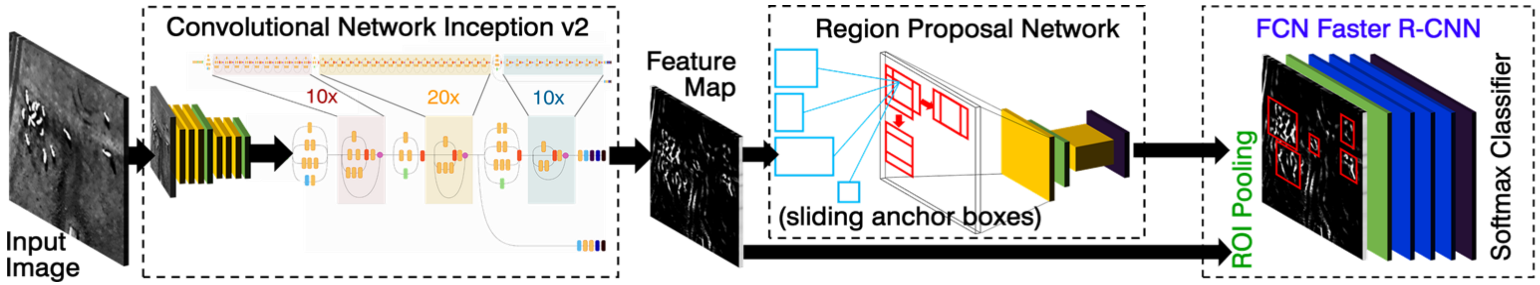
Images: DeepSolar/Stanford

Common Object Detection Tasks



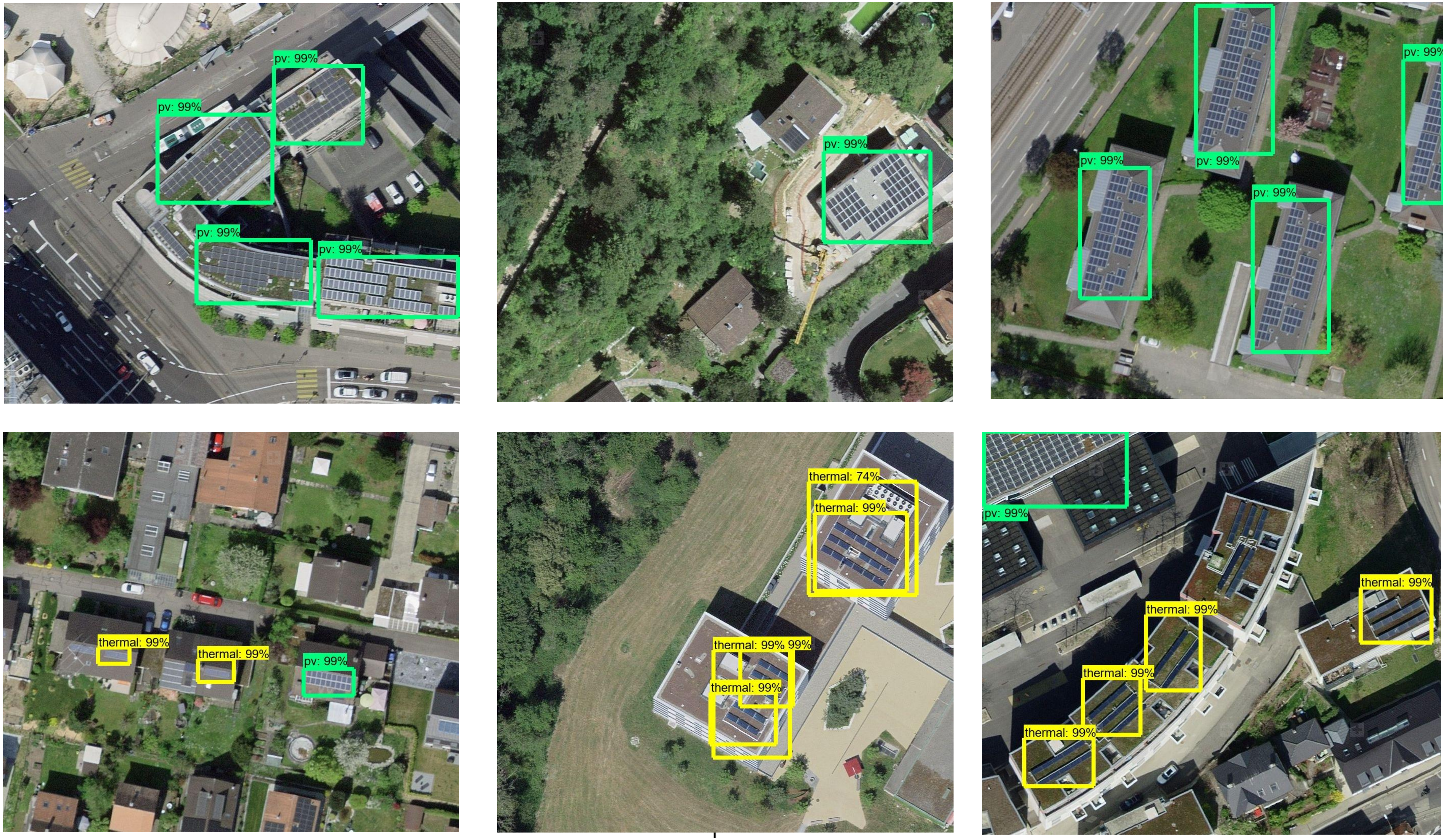
Faster RCNN & TF to Identify Tiles with Panels

Region-based CNN, implemented with Tensorflow



Train your own Solar Detector!

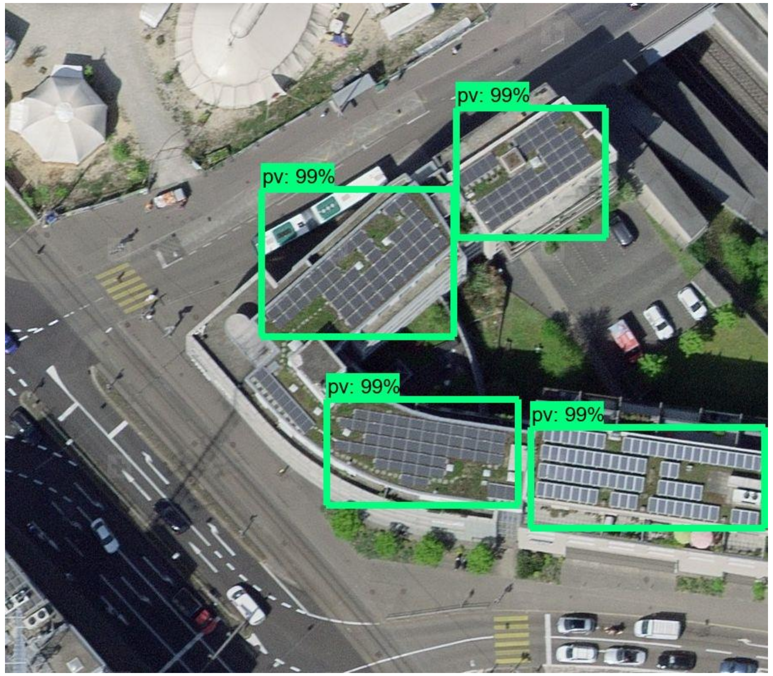
- https://colab.research.google.com/github/FHNW-IVGI/workshop_geopython2019/blob/master/Ex.02_SolarPanels/FasterRCNN_Tutorial_MeyerA.ipynb
- Availabe under: tinyurl.com/solar-detect



pv: 99%
pv: 99%
pv: 99%
pv: 99%
pv: 99%
pv: 99%
pv: 99%
pv: 99%
pv: 99%
pv: 99%
thermal: 99%
thermal: 99%
pv: 99%
thermal: 74%
thermal: 99%
thermal: 99% 99%
thermal: 99%
pv: 99%
thermal: 99%
thermal: 99%
thermal: 99%
thermal: 99%

Pre-Detection

Photovoltaic Systems: 92% mAP
Thermal Systems: 62% (ca. 30% are detected as PV)

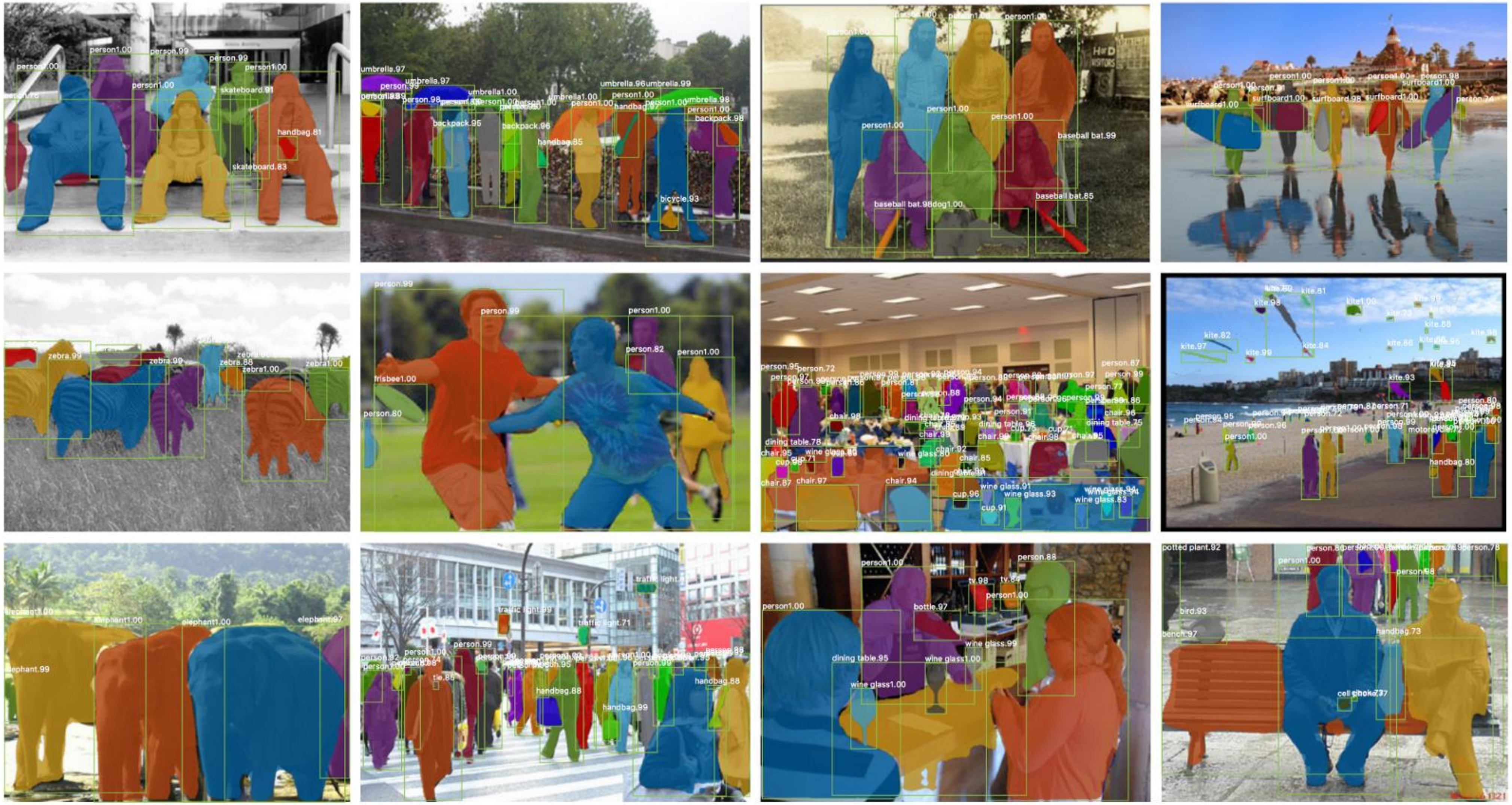


n|w

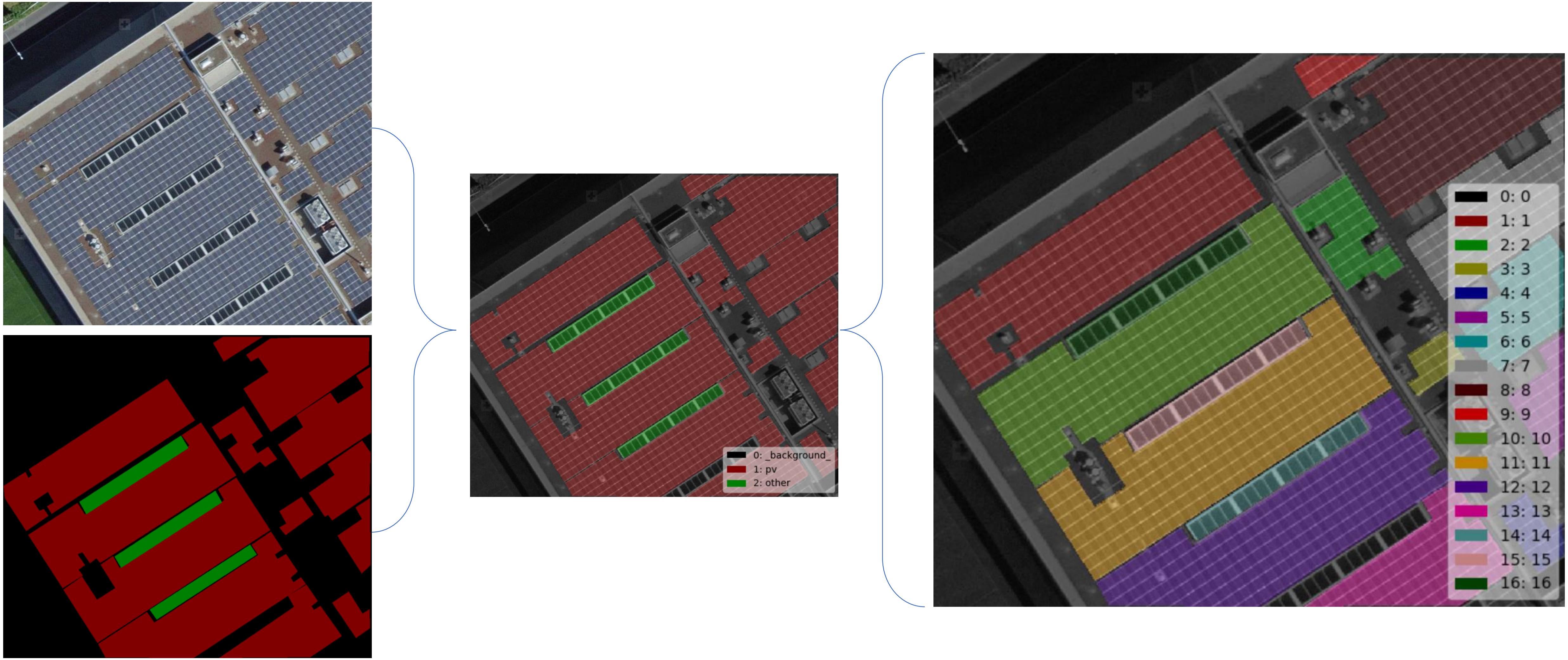
Multilayered Workflow

- 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

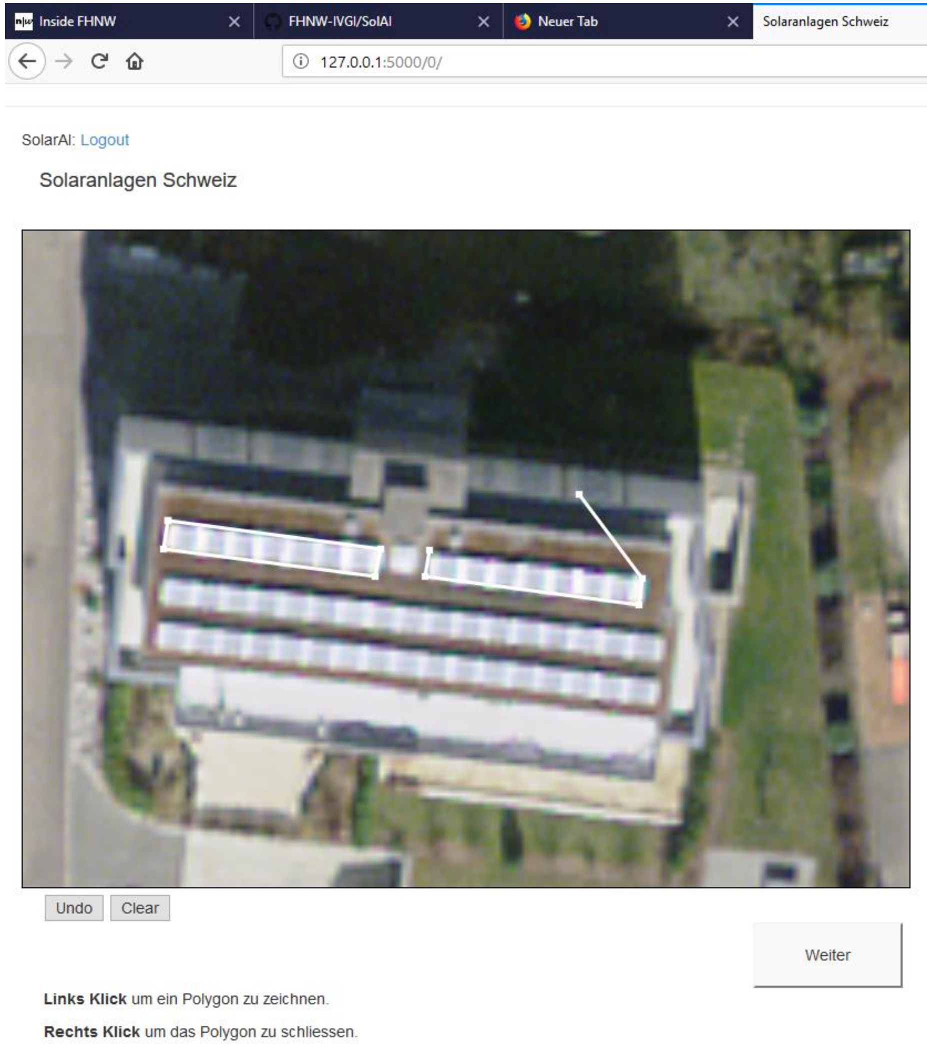
Next Step: Mask RCNN



Instanciation



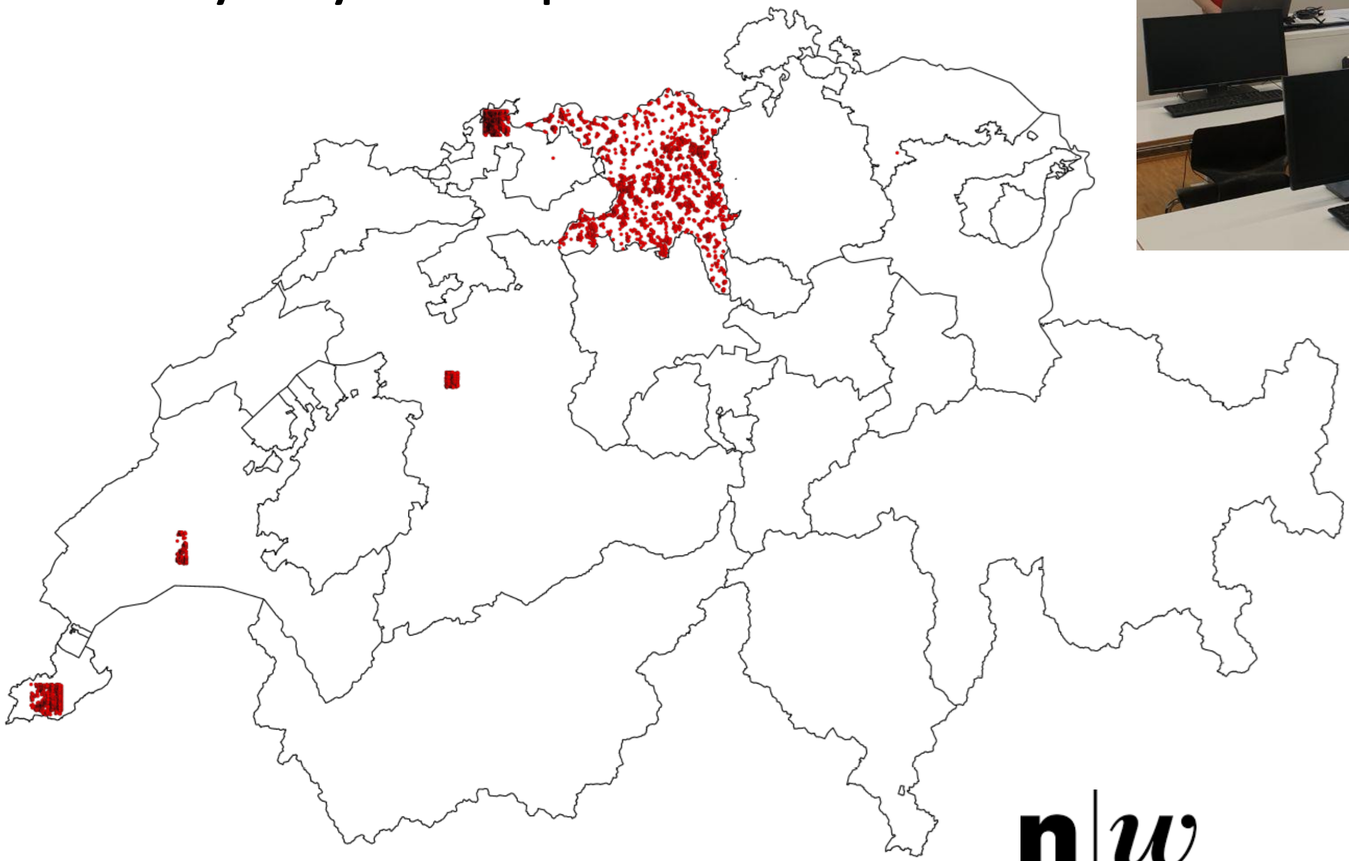
Cloud Contribution Client for Labelling

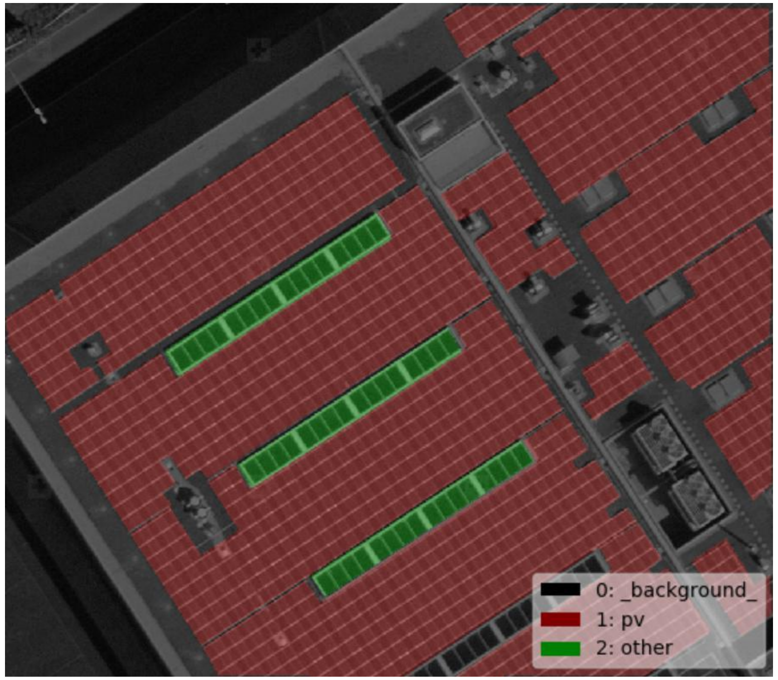


Code-Sprint: Europython 2019

Labelling Workshop

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

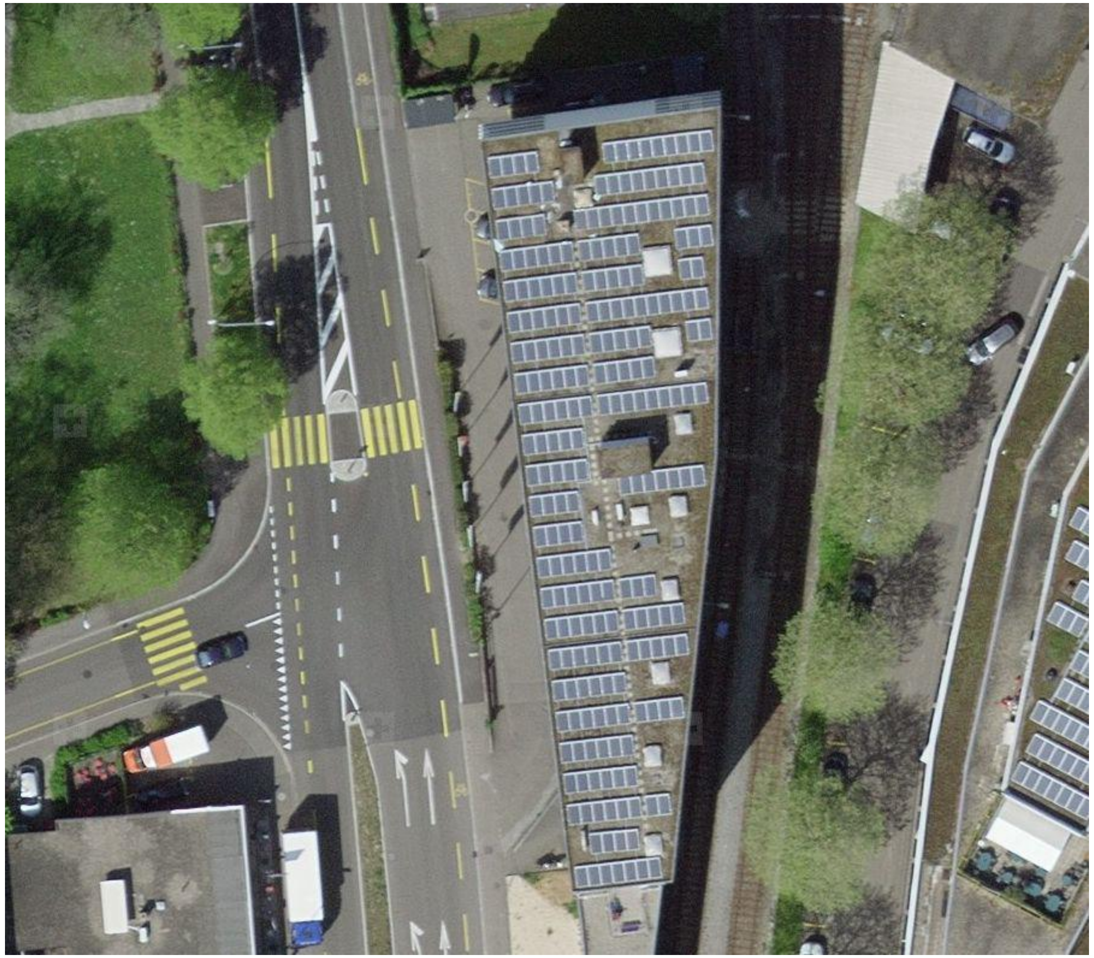


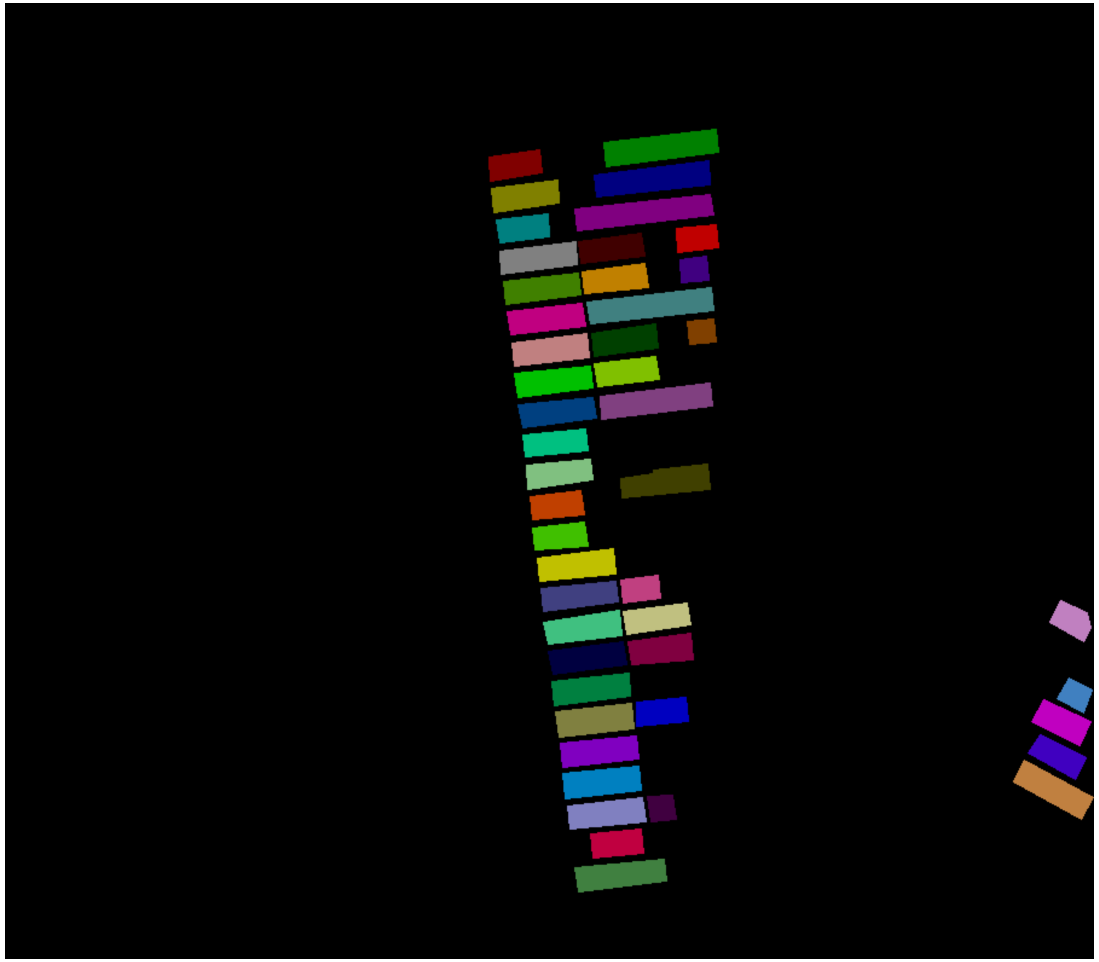


n|w

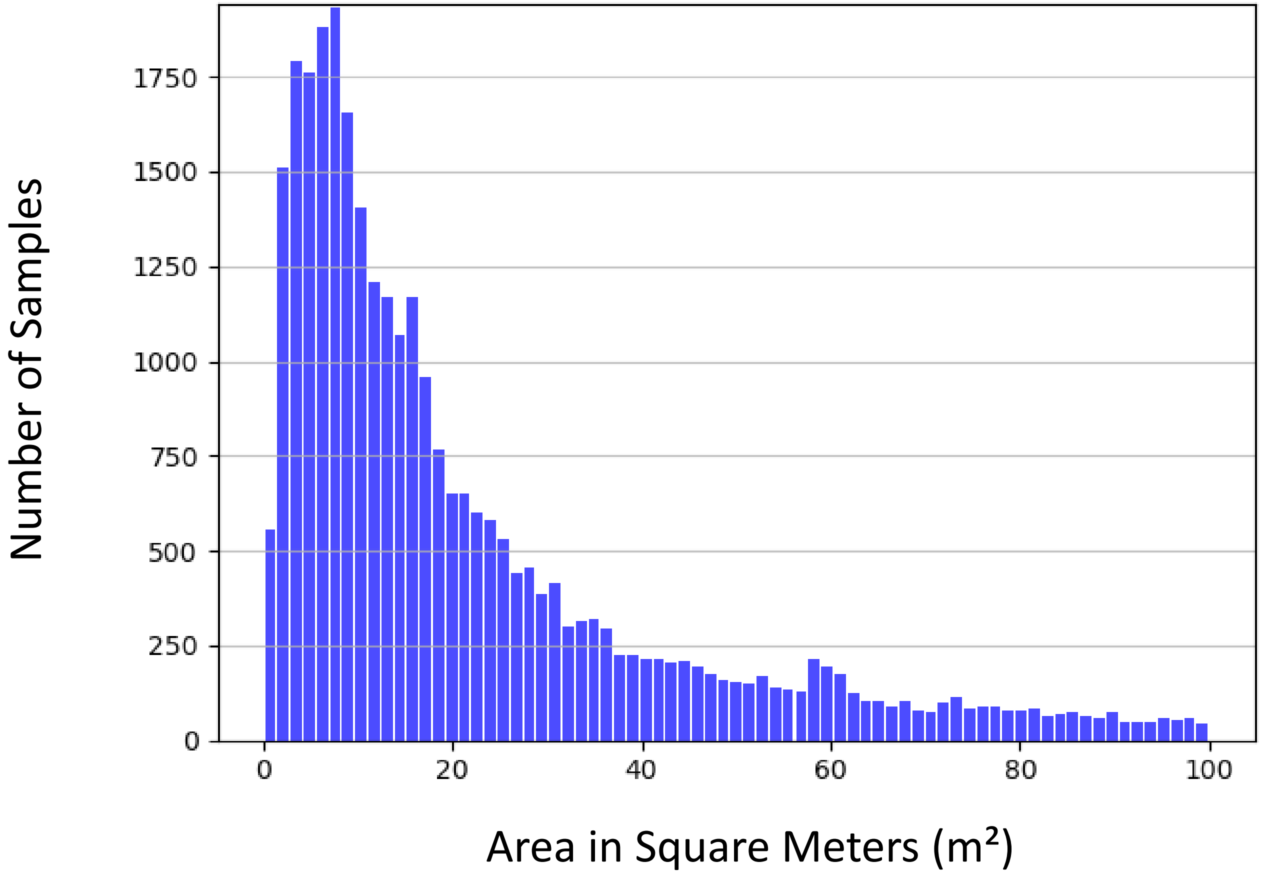
Images 1 & 2 of 7‘839

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_singleread	bn_fw_tr_1C11_kernel_new

Source:
Anto John,
Oct 2018,
IBM Developer Blog

PyTorch

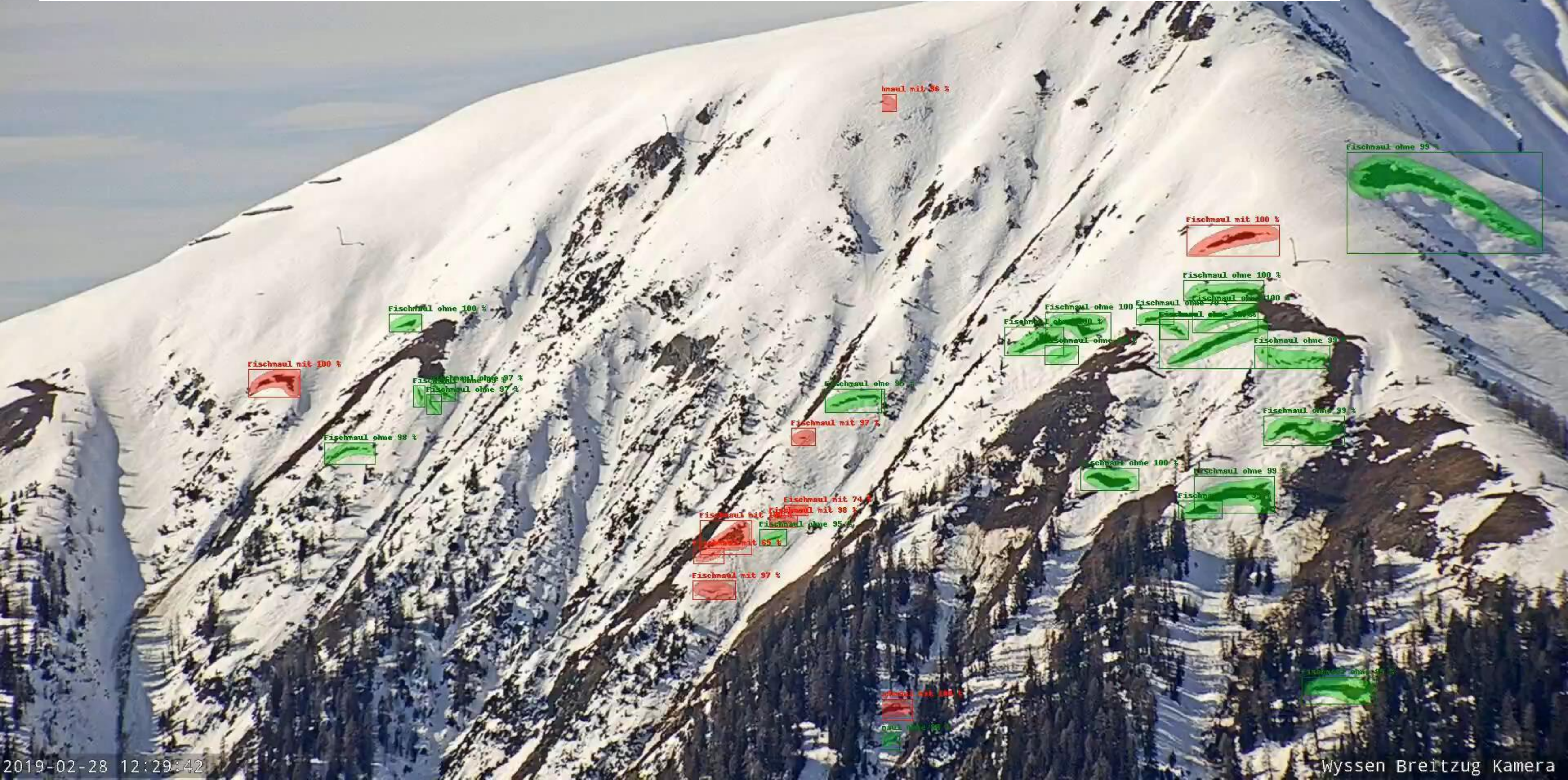
- Open Source Library for Machine Learning (BSD-License)
- Based on Torch Framework (Lua, C++, CUDA), published in 2002
- PyTorch was published in October 2016
- Main Developers are the AI R&D Teams of Facebook
- Development with Python

```python
# 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)
```

Advantages:

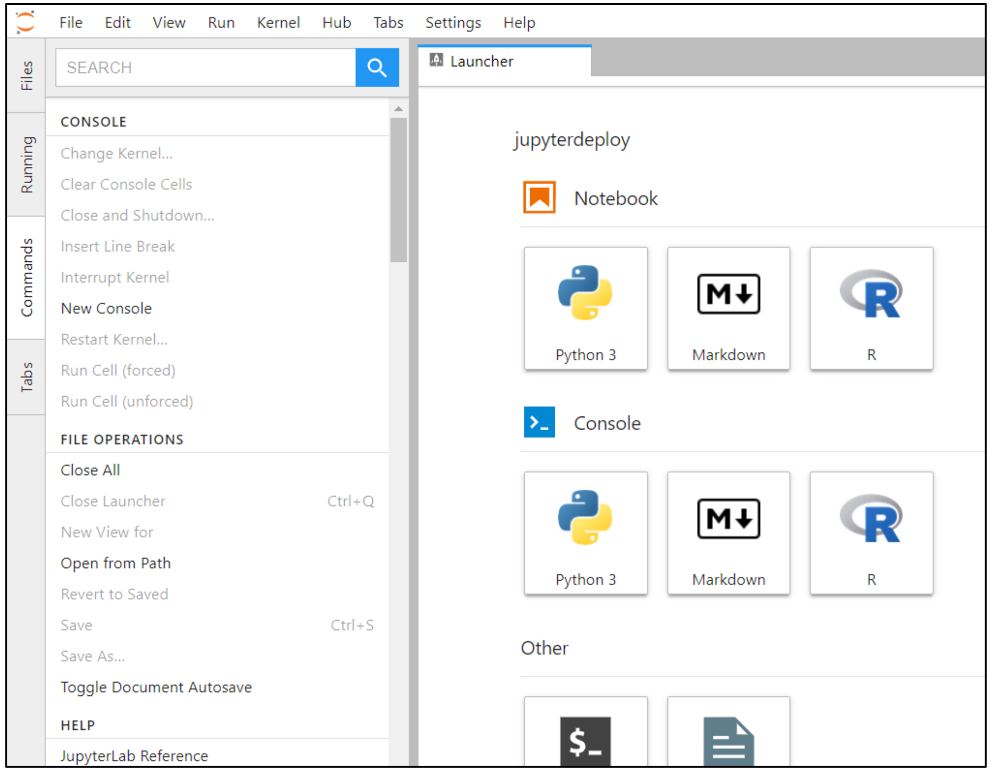
- Pythonic Interface
- GPU Support with nice Interface
  - a.from_numpy/a.numpy torch tensor bridge
  - Pretrained Models available (Torchvision)
  - Multiple Optimizers (SGD/Adam/etc.)

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)

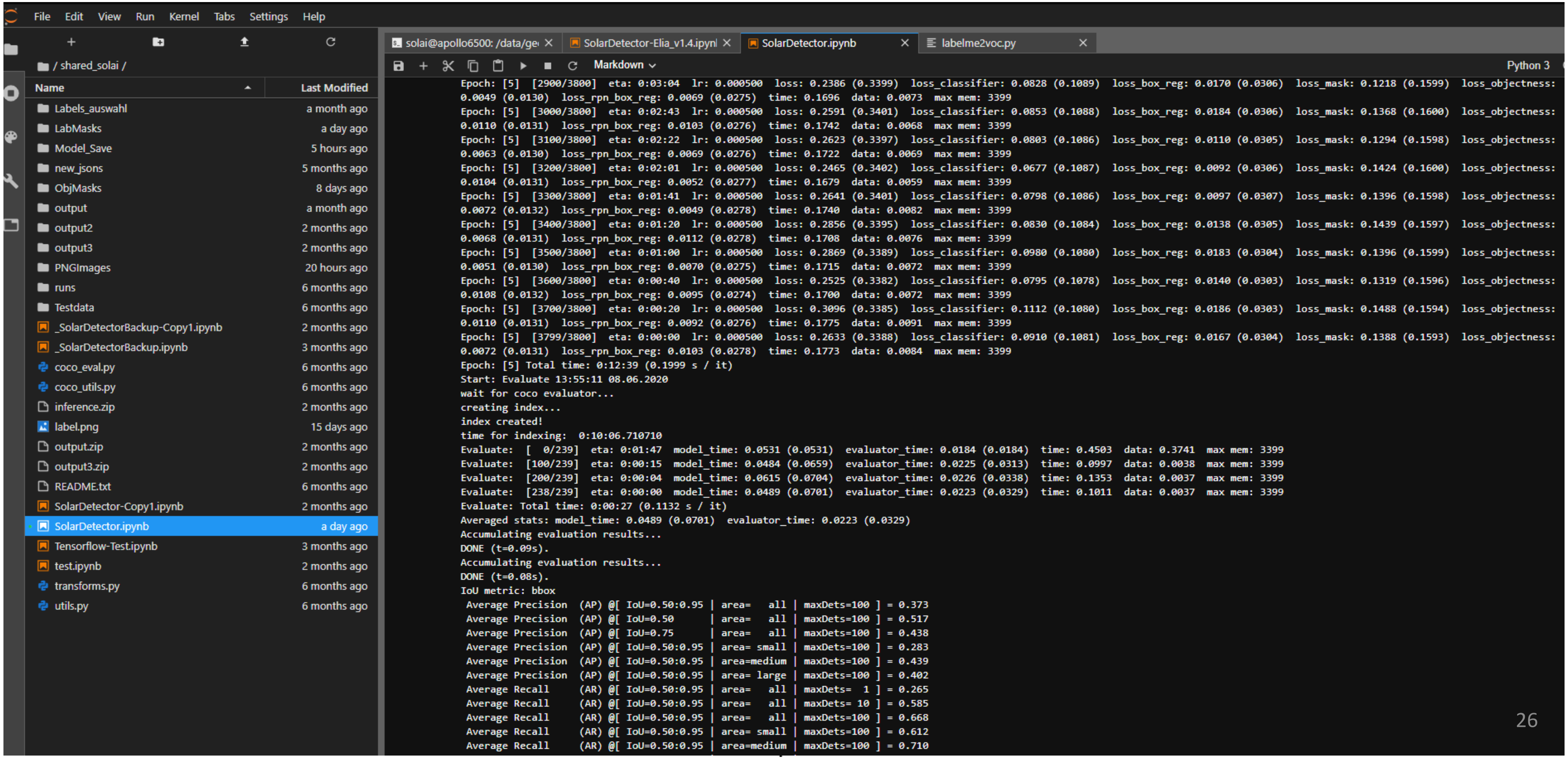


4x

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



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



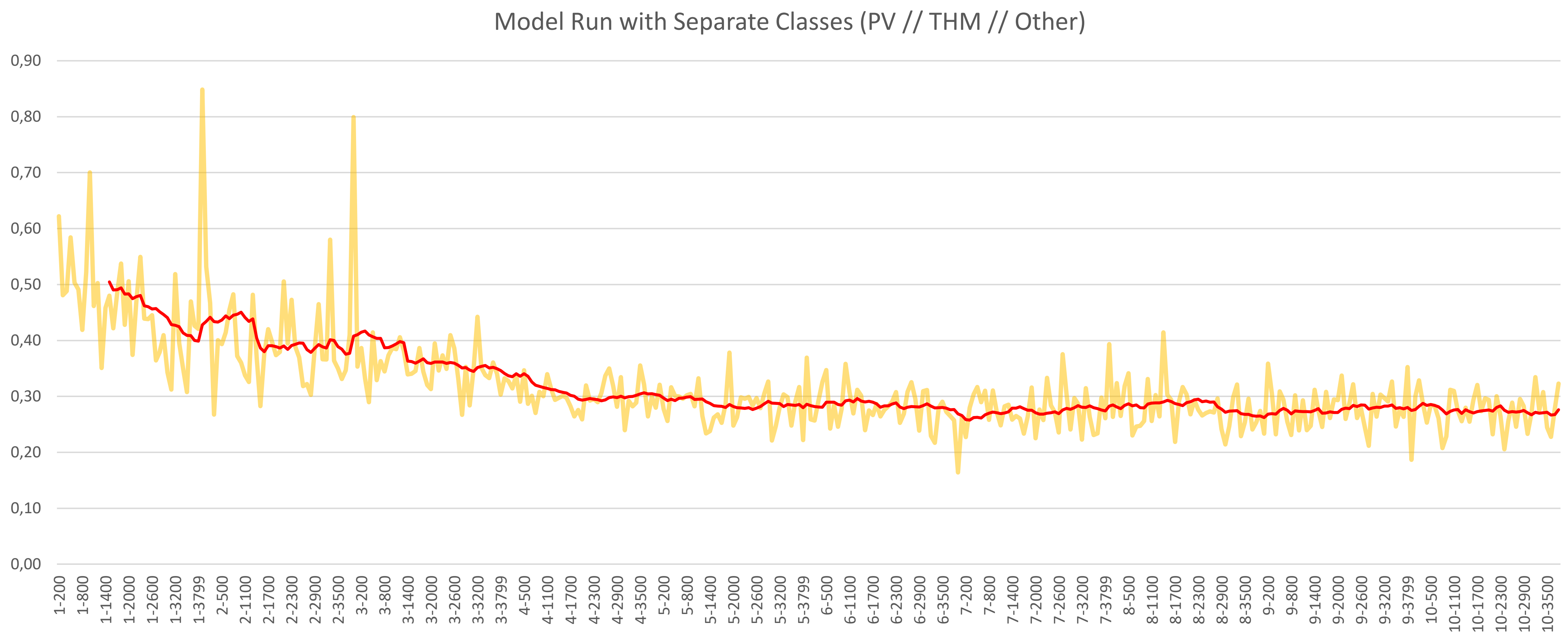

- The tricky part is to support our 4 GPUs. Data Parallelism is implemented using `torch.nn.DataParallel`.

- PyTorch doesn’t really provide many examples for multi-GPU, so it was a little bit try and error.

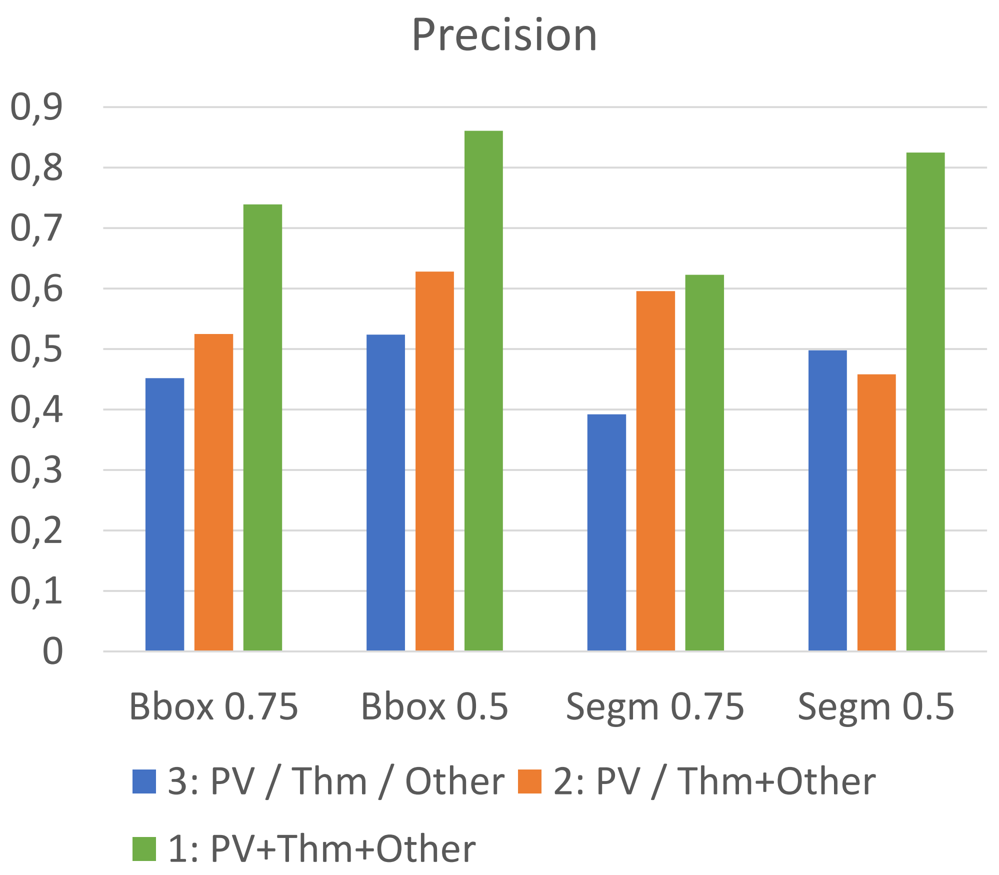
inheriting the class **`torch.utils.data.Dataset`**

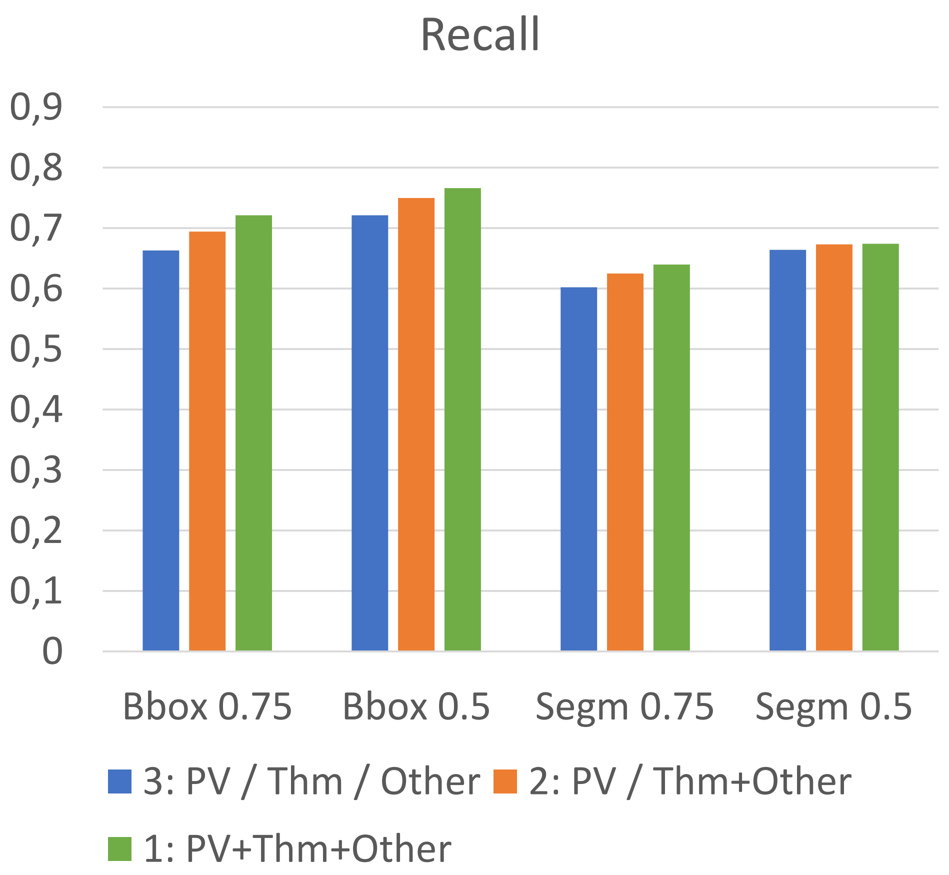
```python
import os
import numpy as np
import torch
import torch.utils.data
from torchvision import transforms, utils, datasets
from PIL import Image, ImageDraw, ImageFont, ImageOps

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

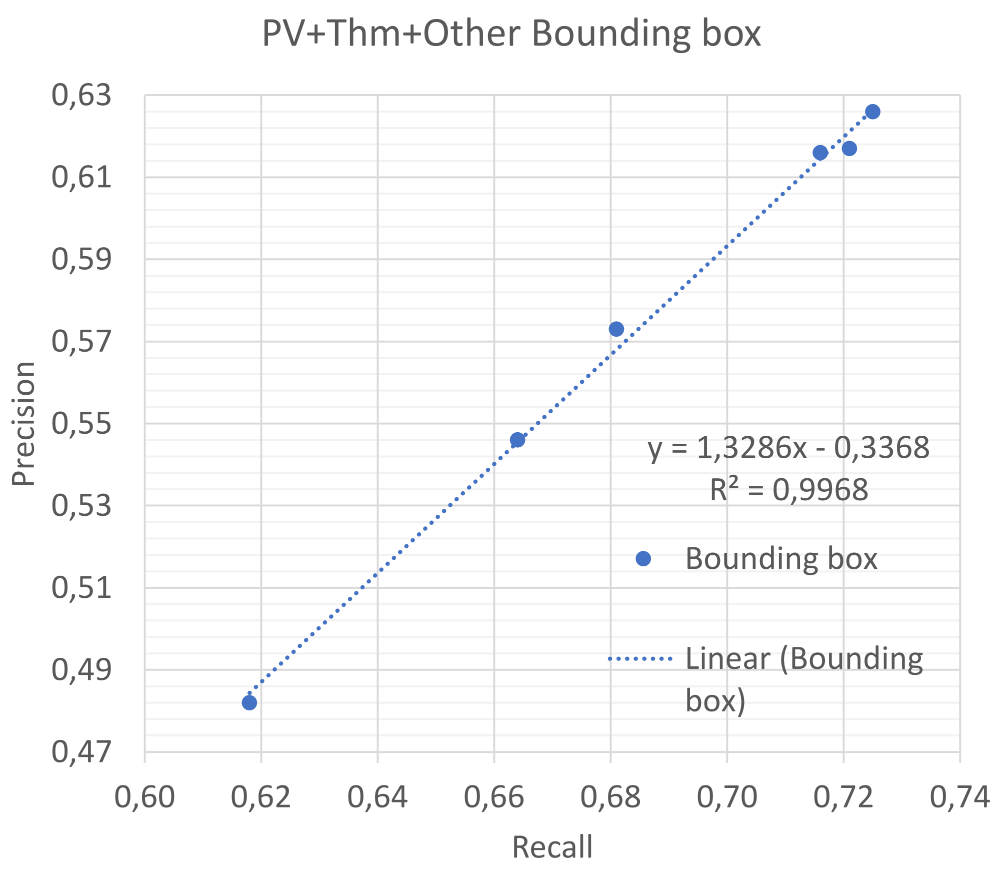


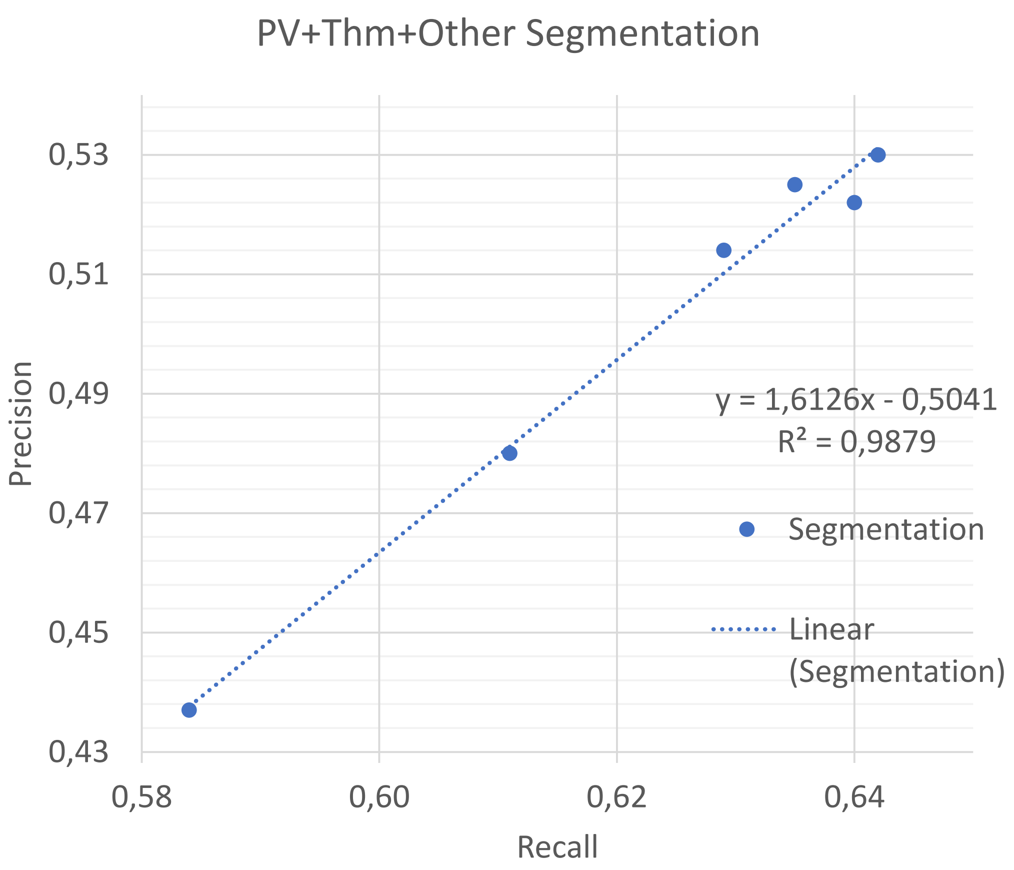
Preliminary Results: Metrics for Class Combination



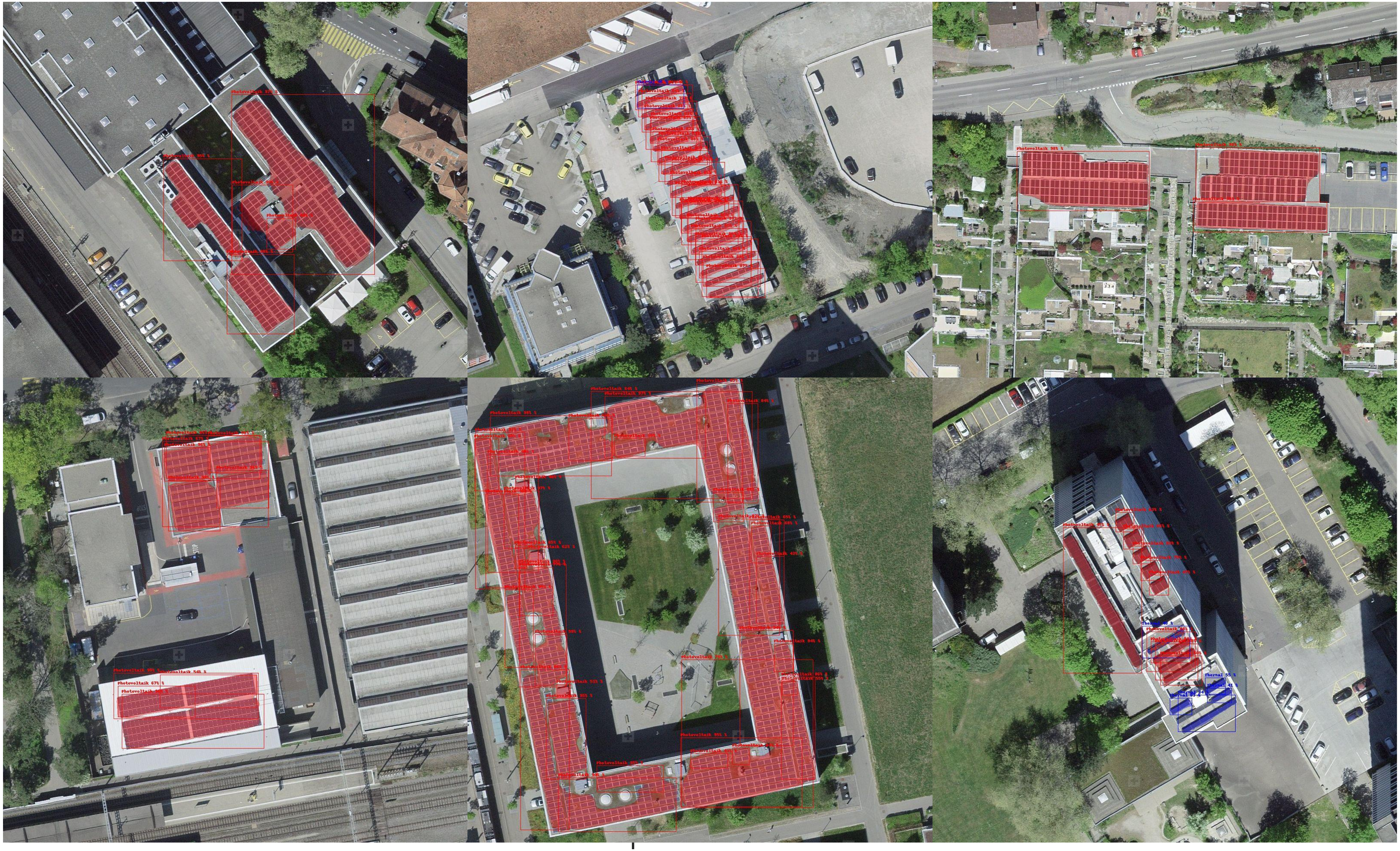


Statistical Linearity

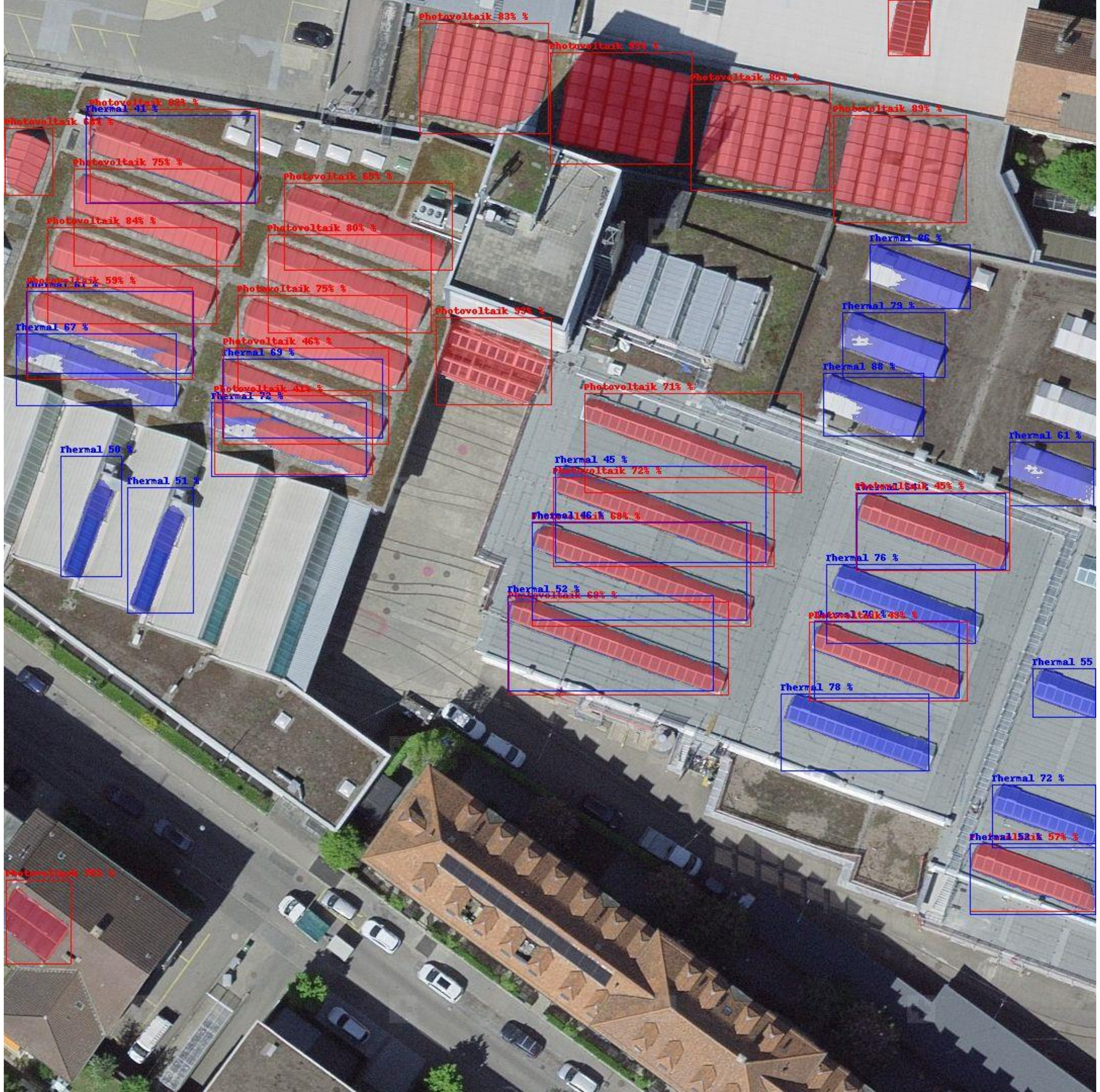


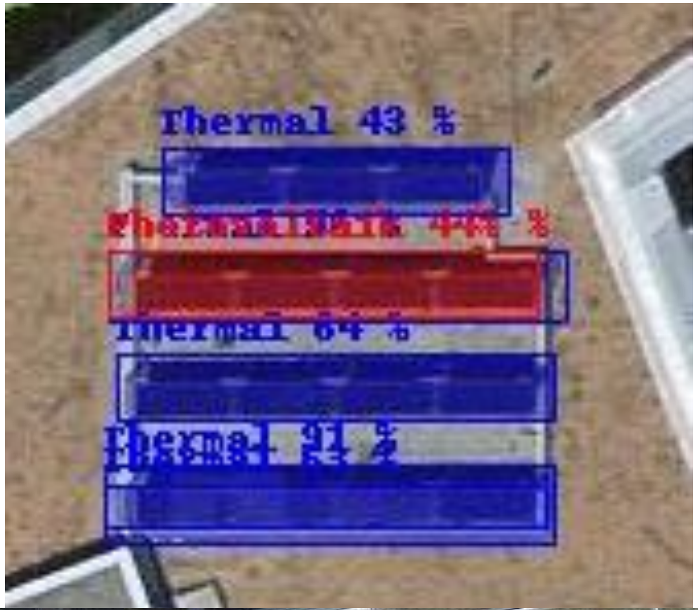


Results



Challenges





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

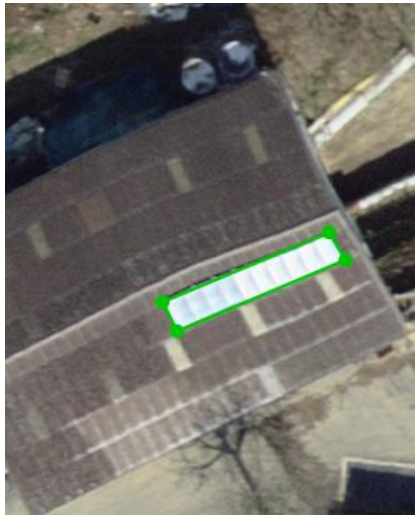





Challenge Labels: Class «Others»

- Difficult

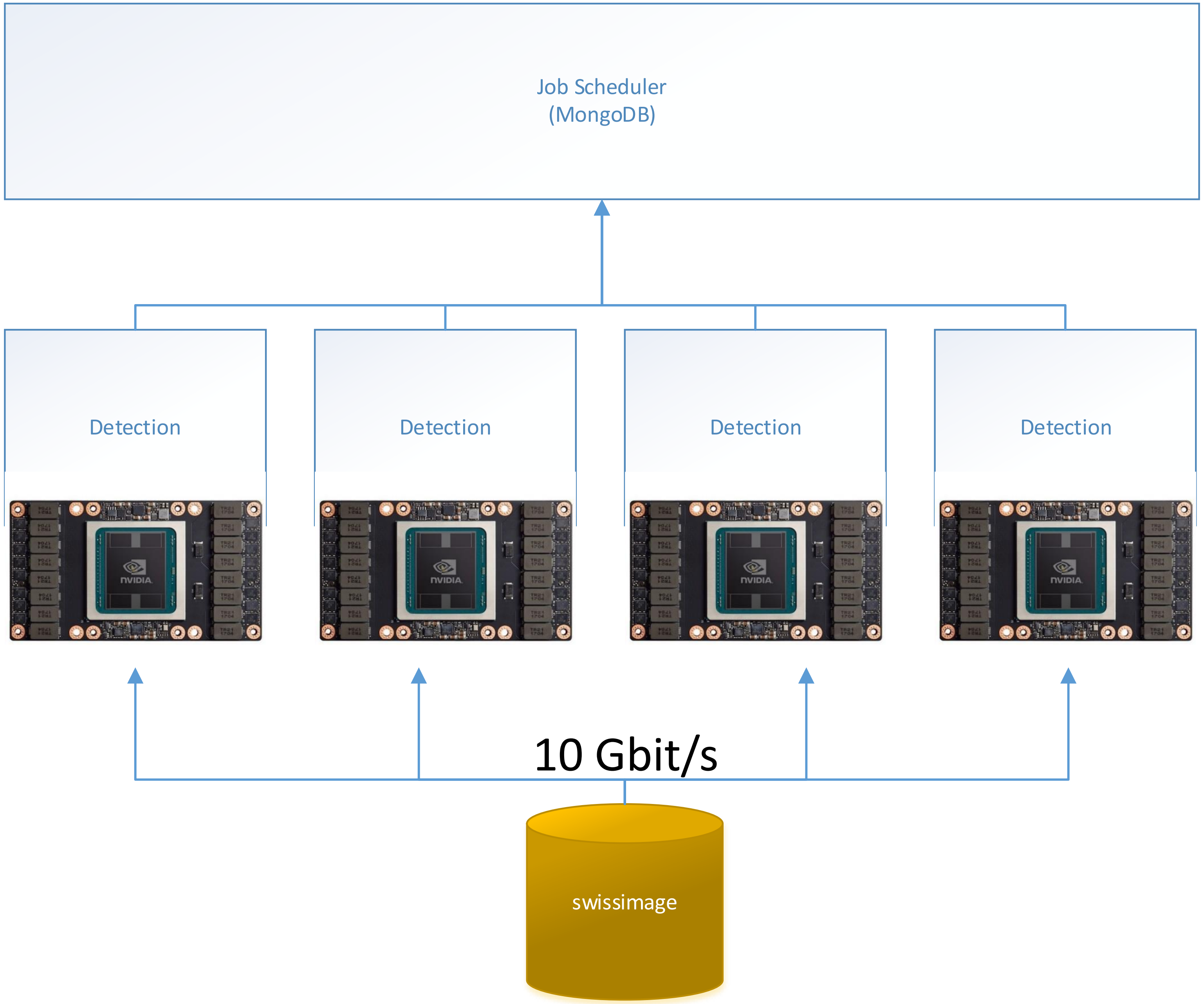



Challenge: Labelling Mistakes

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)



Job Queue

Unlimited* Processes

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

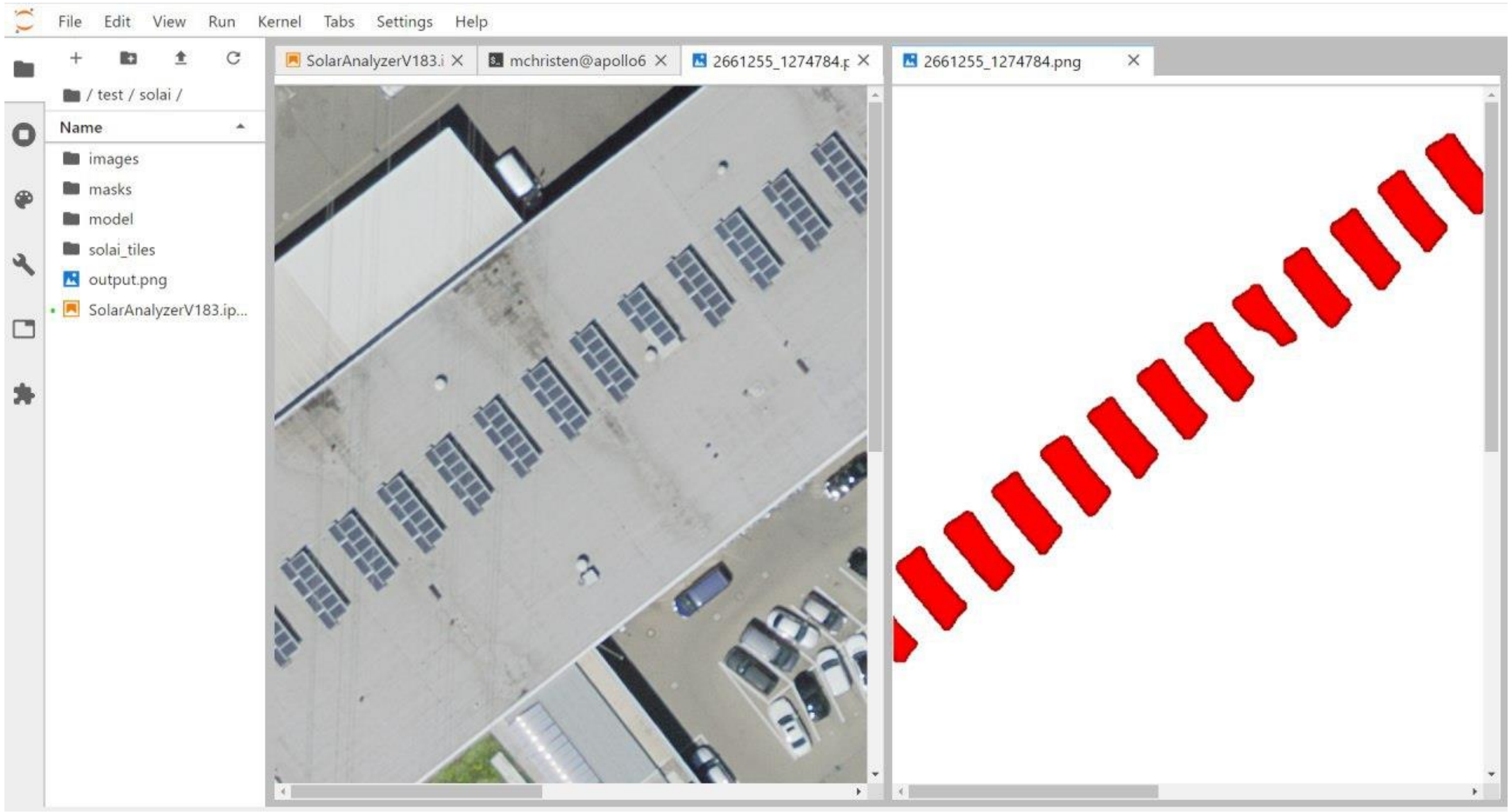
```
  PID USER      PR  NI    VIRT    RES    SHR S  %CPU %MEM     TIME+ COMMAND
67579 mchrist+  20   0 25.701g 4.774g 622564 R 202.0  2.5   7:16.34 python
67436 mchrist+  20   0 22.608g 2.857g 519344 R 196.4  1.5   7:28.21 python
67860 mchrist+  20   0 25.816g 4.811g 622624 R 196.4  2.6   6:51.77 python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

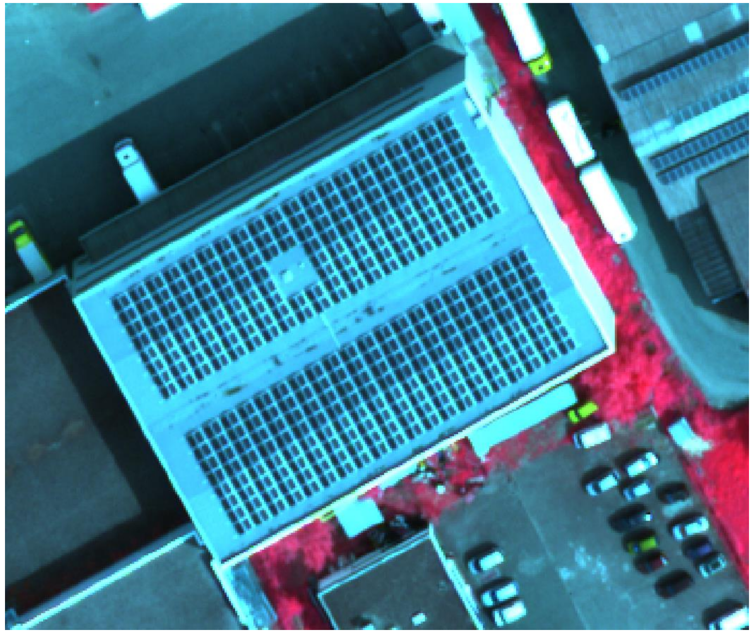
Big Data Inferencing & Classification Workflow

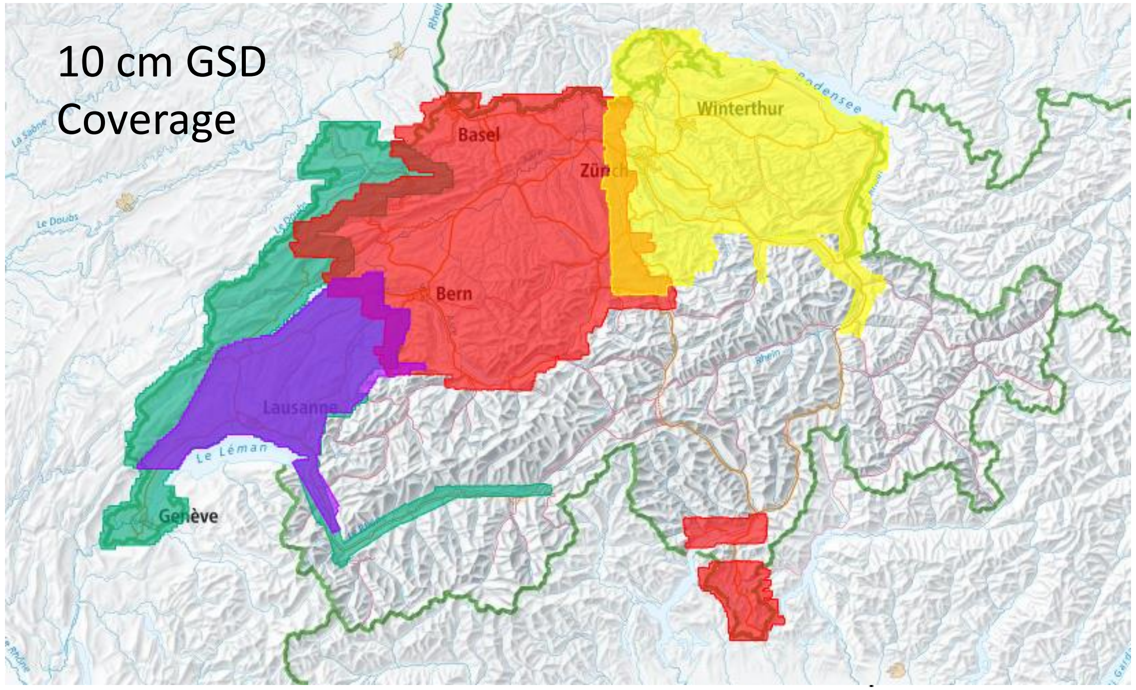


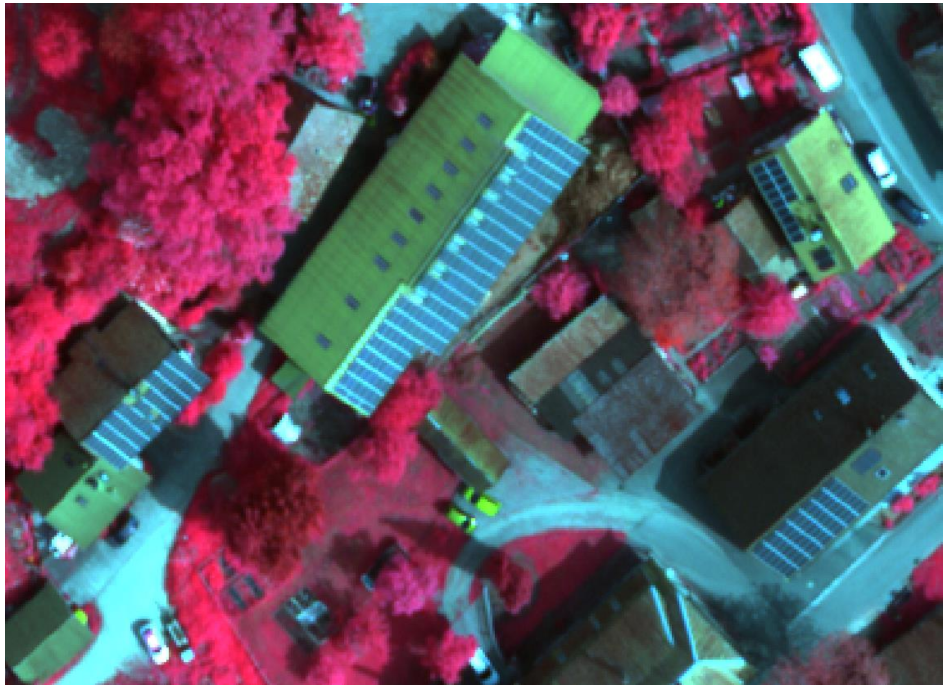
Run multiple models for inferencing, using heuristic measures for edges and segment probability

→ Rasterio / fiona / GDAL

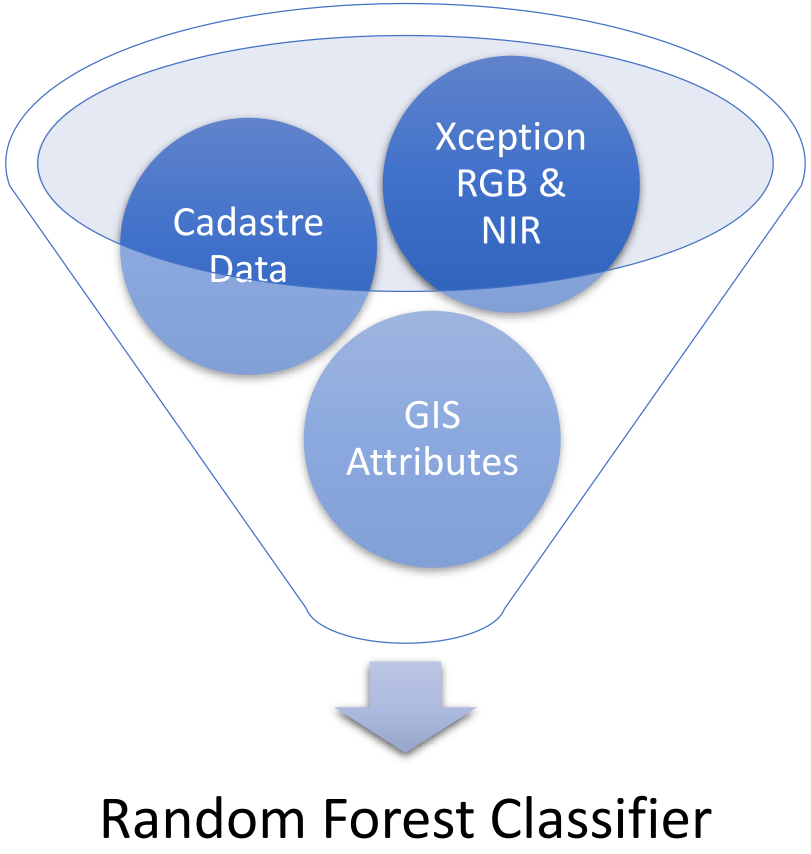
Include Near-Infrared Data

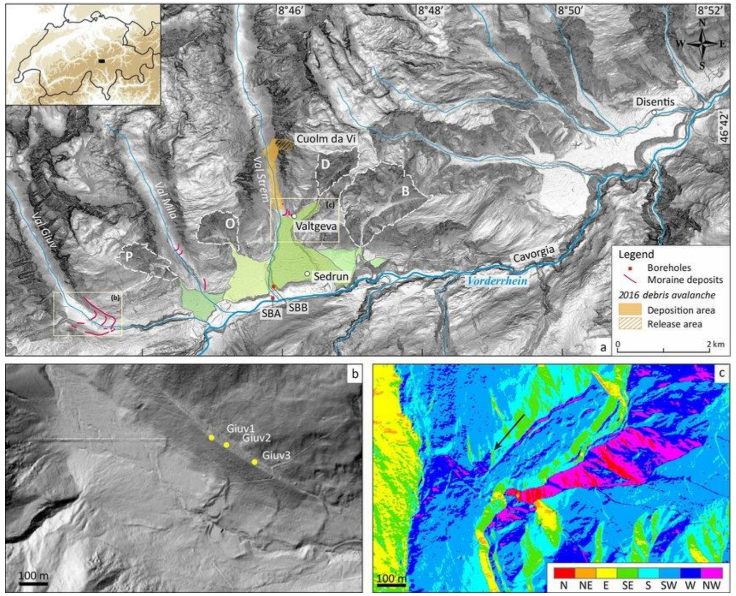






Random Forest & Xception for Post-Classification





Thank you!