DETECTREE: TREE DETECTION FROM AERIAL IMAGERY IN PYTHON

Martí Bosch

January 23, 2020

TYPES OF URBAN TREE CANOPY DATASETS

URBAN TREE CATALOGS

Format: geo-referenced list of trees

Example: Cantonal inventory of isolated trees (Geneva)

URBAN TREE CATALOGS

PROS

- Often comes with valuable attributes (e.g., dimensions, species, age…)

URBAN TREE CATALOGS

CONS

- **Costly**: manual surveys
- Often restricted to public space

CANOPY HEIGHT MODELS

Format: raster of tree height values

Example: Canopy height model (Montreal)

CANOPY HEIGHT MODELS

PROS

- Can be automatically derived from *raw* LIDAR data, i.e., classified cloud of points

CANOPY HEIGHT MODELS

CONS

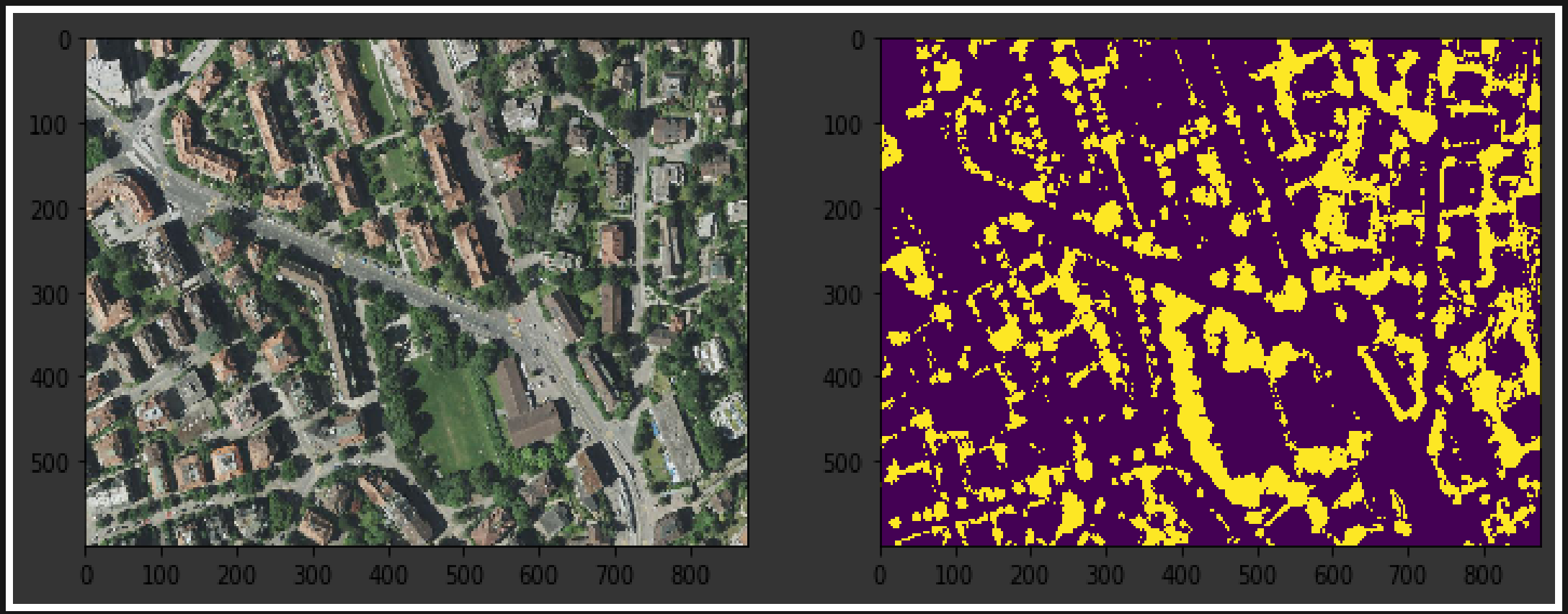
- LIDAR data is **expensive**
- Building canopy height models requires *raw* LIDAR data (surface and terrain models are not enough)
- Few open *raw* LIDAR datasets

WHAT DOES DETECTREE DO?

IDEA IN A NUTSHELL

- **Input**: high resolution aerial imagery
- **Output**: binary raster of tree/non-tree pixels

IDEA IN A NUTSHELL



Example: DetecTree output (right) for Zurich's 2014/15 Orthophoto (left)

IDEA IN A NUTSHELL

SUPERVISED LEARNING

The **training set** $S = \{(x_i, y_i), i = \{1, \ldots, M\}\}$ is a sample of M pixels, each represented by a:

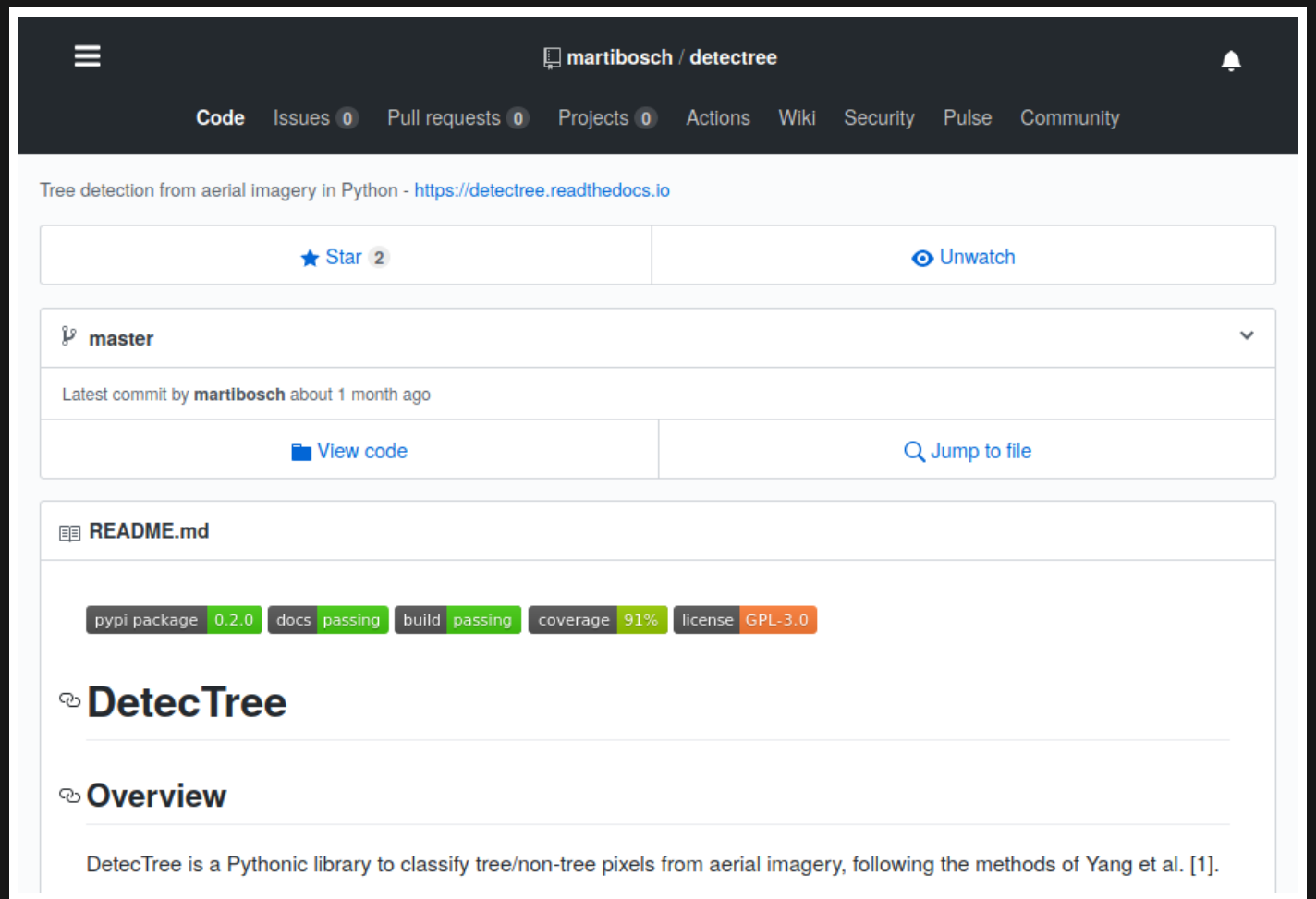
- **27-component feature vector**: $x_i \in \mathbb{R}^{27}$, with information of color, texture and entropy
- **binary response**: $y_i \in \{0, 1\}$, with $y_i = 1$ if pixel i actually corresponds to a tree, $y_i = 0$ otherwise

THE IDEA IS NOT MINE

- Approach proposed by Yang et al. [1] in 2009
- Others have implemented as well: Mapping All of the Trees with Machine Learning

HOWEVER

To my knowledge, DetecTree is the first open source tool to perform such a task



OVERVIEW OF THE COMPUTATIONAL WORKFLOW

STEP 0: SPLIT IMAGE INTO TILES

Input: aerial imagery raster for the area of interest

- The dataset might already come as a mosaic of tiles
- Otherwise, you might use DetecTree's `split_into_tiles` function

STEP 1: TRAIN/TEST SPLIT

We might use random sampling to select, e.g., 1% of the tiles as training data, **however** …

… the training tiles should be **as representative as possible** of the overall dataset

STEP 1: TRAIN/TEST SPLIT

How do we optimize the **representativity** of the training set?

1. For each tile, compute a GIST descriptor [2], i.e., a vector describing key semantics of the tile's scene
2. Apply *k*-means to the GIST descriptors to get *k* clusters of tiles, with *k*= size of the training set
3. For each cluster, select the tile that is closest to the cluster's centroid for training

STEP 1: TRAIN/TEST SPLIT

```
split_df = dtr.TrainingSelector(
    img_dir='path/to/tiles').train_test_split(
        method='cluster-I')
split_df.head()
```

	img_filepath	train
0	data/interim/tiles/1091-322_00.tif	False
1	data/interim/tiles/1091-142_20.tif	False
2	data/interim/tiles/1091-124_11.tif	False
3	data/interim/tiles/1091-144_21.tif	False
4	data/interim/tiles/1091-213_05.tif	False

STEP 1: TRAIN/TEST SPLIT

```
split_df[split_df['train']]
```

	img_filepath	train
28	data/interim/tiles/1091-231_12.tif	True
98	data/interim/tiles/1091-144_06.tif	True
172	data/interim/tiles/1091-231_07.tif	True

In this example we have 225 tiles. A training sample of 1% needs 2.25 tiles, thus DetecTree automatically sets k to the ceil of such number, i.e., $k = 3$

STEP 3: GET THE TRAINING GROUND-TRUTH TREE/NON-TREE MASK

- For each tile of the training set, we need to provide the ground-truth tree/non-tree masks to get the pixel-level responses y_i , $\forall i \in \{1, \ldots, M\}$
- We might use GIMP, Adobe Photoshop or LIDAR data (see the detectree-example repository)

STEP 4: TRAIN A BINARY PIXEL-LEVEL CLASSIFIER

Given the pixel-level responses, DetecTree will compute the pixel-level features x_i , $\forall i \in \{1, \ldots, M\}$ and train a binary classifier of the form:

$$f : \mathbb{R} \rightarrow \{0, 1\} \quad \text{i.e.,} \quad \hat{y}_i = f(x_i)$$

where $\hat{y}_i$ is the tree ($y_i = 1$)/non-tree ($y_i = 0$) prediction for pixel i

STEP 4: TRAIN A BINARY PIXEL-LEVEL CLASSIFIER

```
clf = dtr.ClassifierTrainer().train_classifier(
    split_df=split_df, response_img_dir=response_dir)
```

- `response_dir` is where the response tiles are located
- `clf` is the training classifier, i.e., scikit-learn's `AdaBoostClassifier`

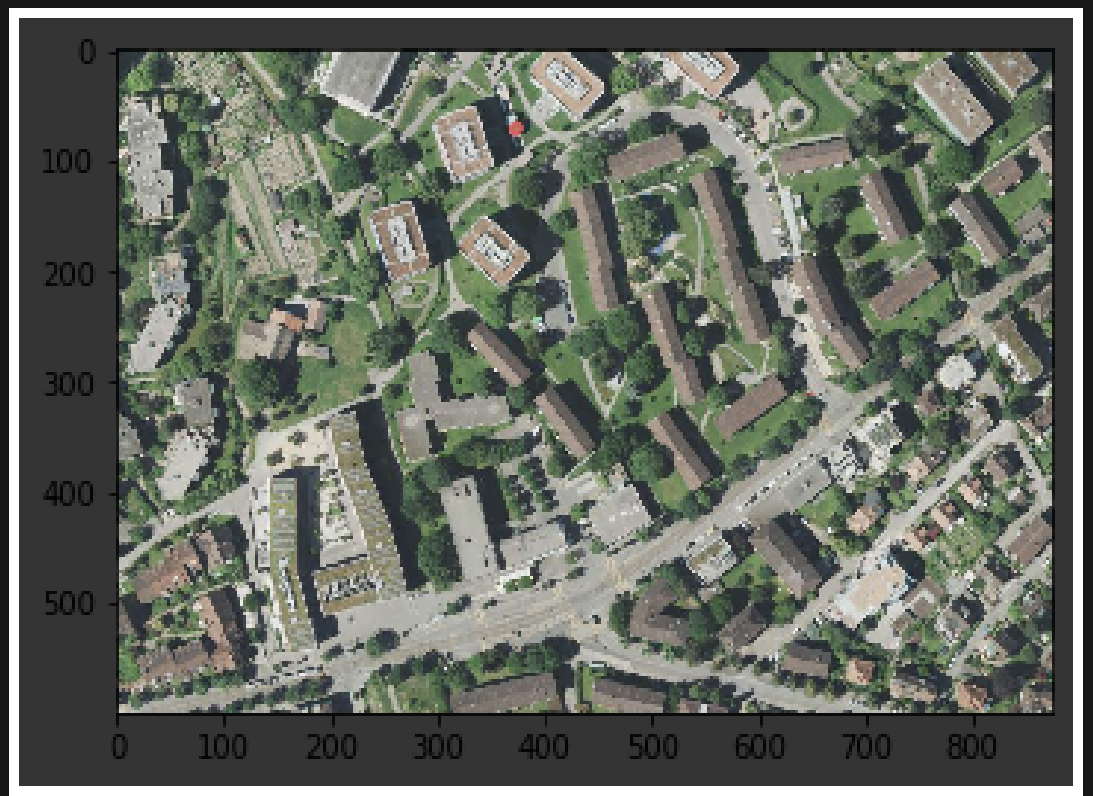
STEP 5: PIXEL-LEVEL CLASSIFICATION

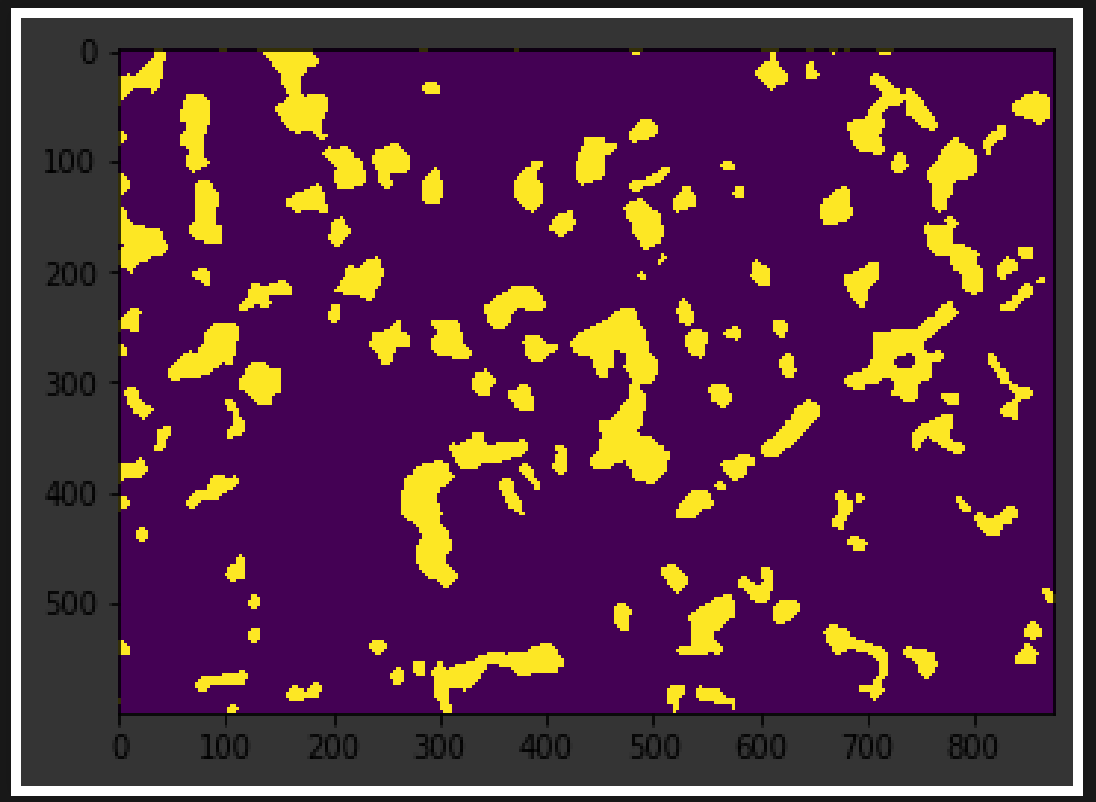
Given the trained classifier `clf`, we might use the `classify_img` method as follows:

```
y = dtr.Classifier().classify_img(
    'path/to/some/tile.tif', clf)
```

STEP 5: PIXEL-LEVEL CLASSIFICATION

Which will give us something like:

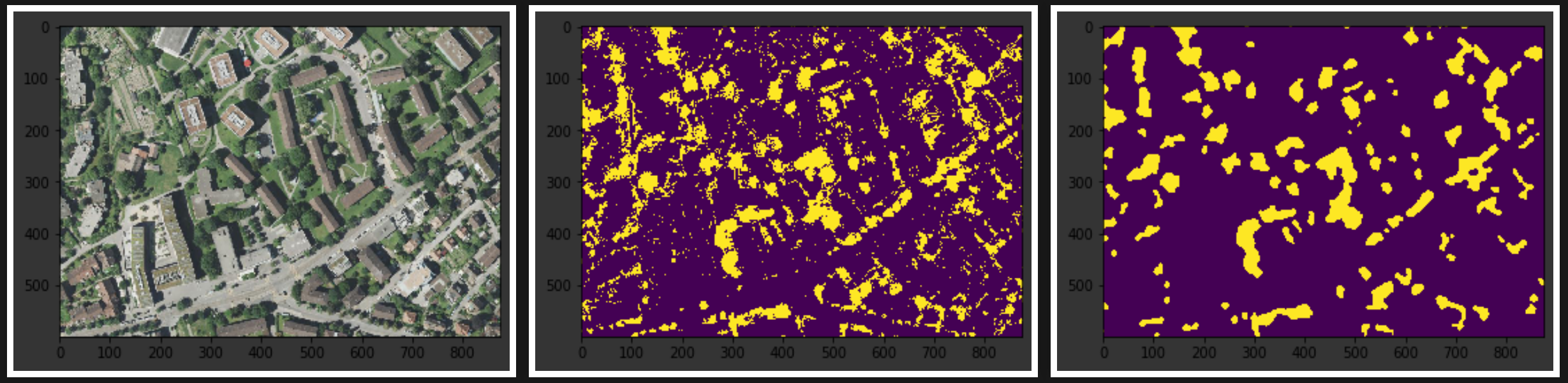




STEP 5: PIXEL-LEVEL CLASSIFICATION

- **Note**: the pixel-level classification predicts each pixel independently, which might yield noisy results, e.g., sparse points on grass fields labeled as trees
- Following the approach of Yang et al. [1], DetecTree refines the pixel-level classification to ensure consistency between adjacent pixels using the graph cuts algorithm of Boykov and Kolmogorov [3].

STEP 5: PIXEL-LEVEL CLASSIFICATION



Original tile (left), pixel-level classification (middle), refined classification (right)

STEP 5: PIXEL-LEVEL CLASSIFICATION

We can use the `classify_imgs` method directly on the train/test split dataframe `split_df` to classify all the tiles at scale with Dask:

```
dtr.Classifier().classify_imgs(
    split_df, 'path/to/output/dir, clf=clf)
```

PROS OF DETECTREE

- Many available datasets of HRO, e.g., NAIP open dataset: Continental USA at the 0.6 to 1m resolution
- Modest memory requirements compared to LIDAR, e.g., Geneva:
  - LIDAR (25 pt/m^2): **310 GB**
  - SWISSIMAGE (1m): **1 GB**

CONS OF DETECTREE

- Only provides *binary pixel-level* classification
- If tree species/dimensions are important, DetecTree is not the best

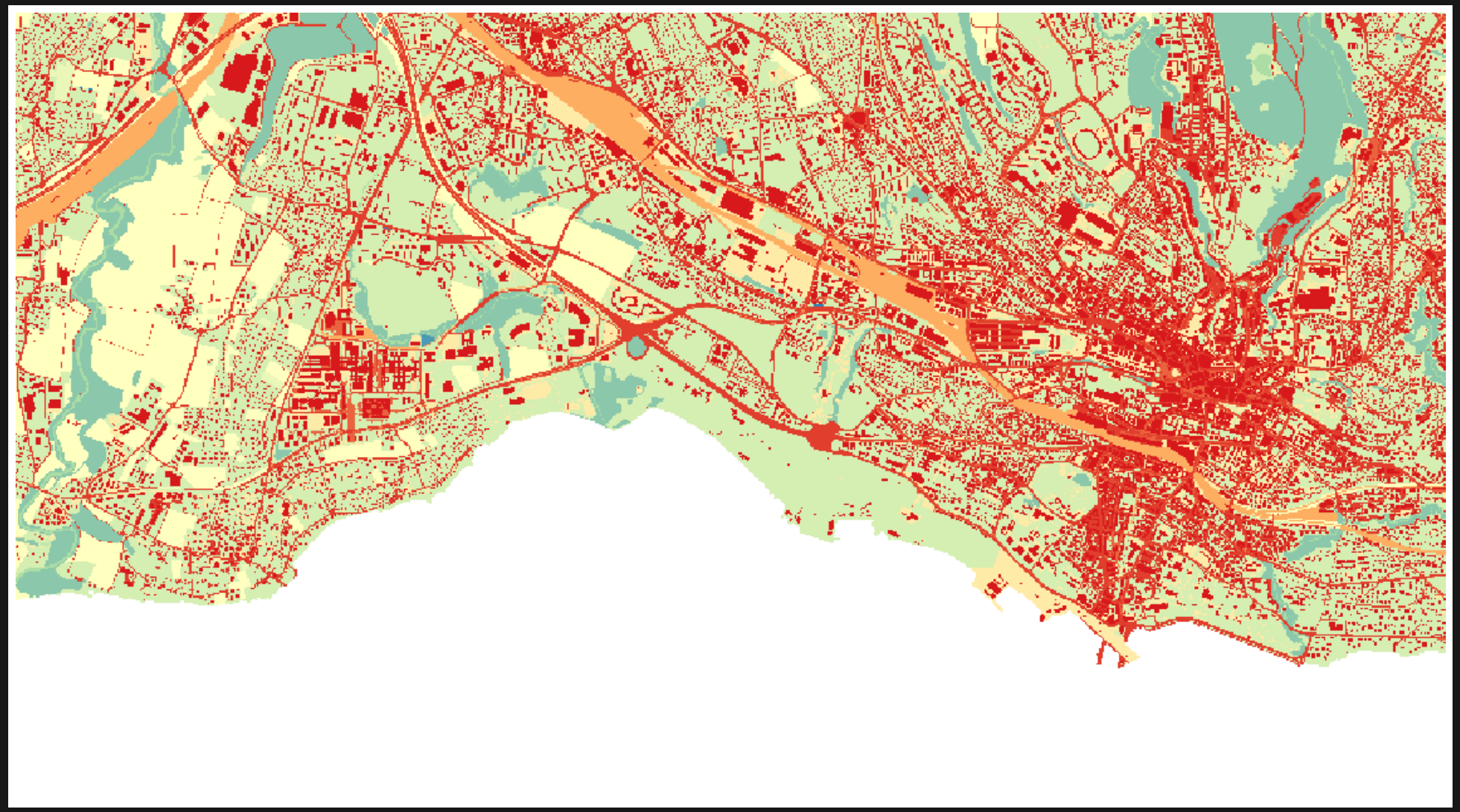
SCOPE OF DETECTREE

- When we are only interested in 2D aspects of trees, e.g., proportion of land cover/spatial distribution
- LIDAR is not available
- LIDAR is available but it is too expensive
- LIDAR is available but you don’t want to process 300GB of data

EXAMPLE APPLICATION: URBAN HEAT ISLANDS IN LAUSANNE

OBJECTIVE

Given a land cover map (10m), predict the spatial distribution of air temperature



APPROACH: INVEST URBAN COOLING MODEL

- For each pixel:

$$T_{air} \sim f(ET, shade, albedo)$$

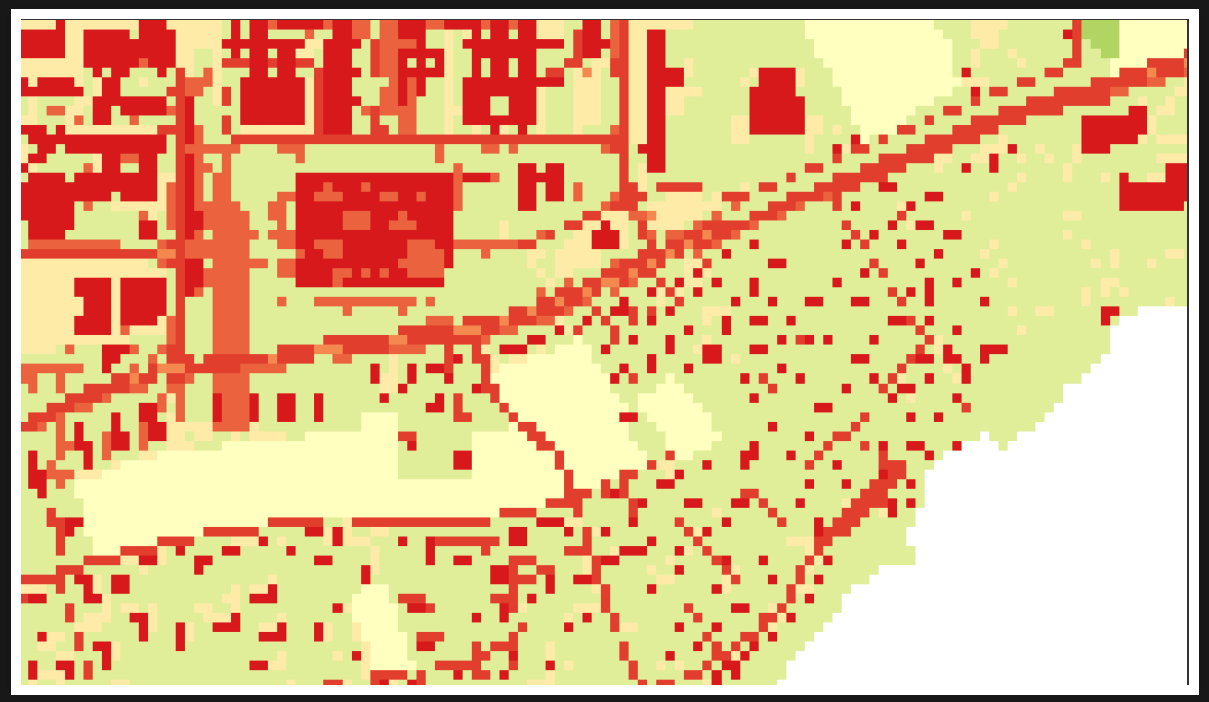
- **However**: pixels of the same land cover, e.g., road, can have very different levels of tree cover (which influences the ET, shade, albedo…)

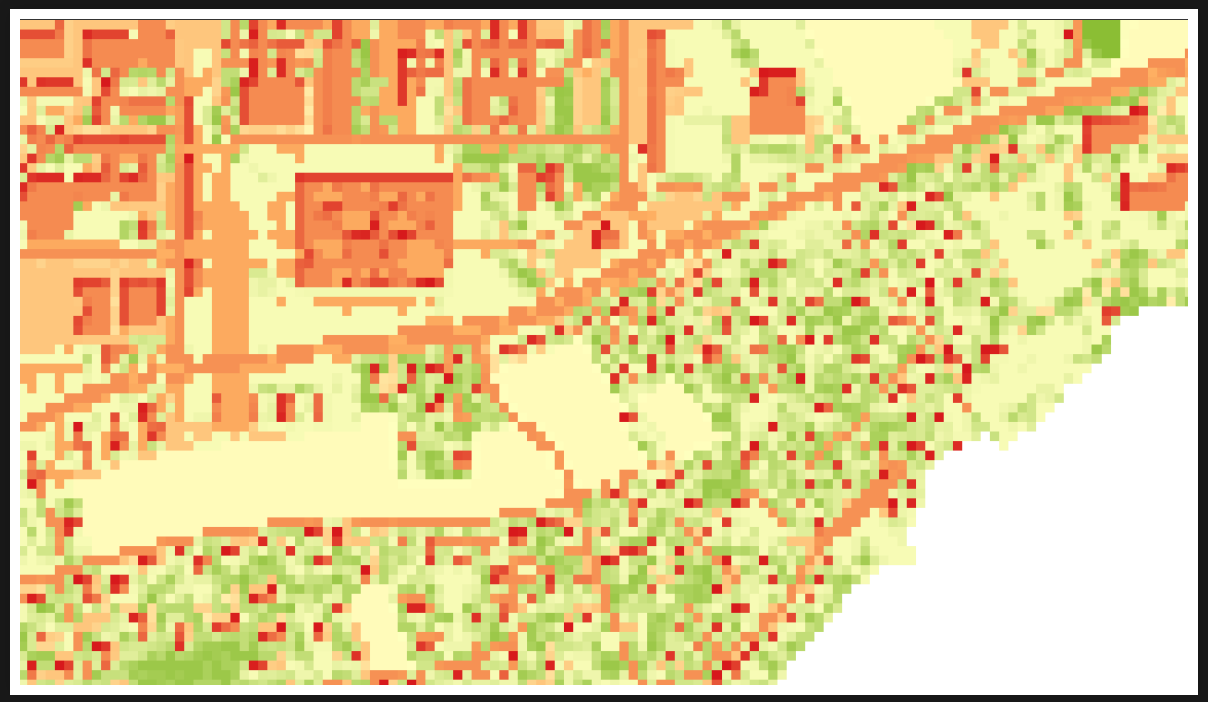
ENTER DETECTREE

- Refine each land cover class into subclasses depending on the level of tree cover
- For example, pixels of “road” land cover are further divided into
  - “road with high tree cover”
  - “road with intermediate tree cover”
  - “road with low tree cover”

ENTER DETECTREE

We go from 25 to 100 land use classes …





… which will likely increase the precision of the model predictions

RESULTS OF THE STUDY COMING SOON

THANK YOU

REFERENCES

1. Yang, L., Wu, X., Praun, E., & Ma, X. (2009). Tree detection from aerial imagery. In Proceedings of the 17th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems (pp. 131-137). ACM.
2. Oliva, A., & Torralba, A. (2001). Modeling the shape of the scene: A holistic representation of the spatial envelope. International journal of computer vision, 42(3), 145-175.
3. Boykov, Y., & Kolmogorov, V. (2004). An experimental comparison of min-cut/max-flow algorithms for energy minimization in vision. IEEE Transactions on Pattern Analysis & Machine Intelligence, (9), 1124-1137.