Geospatial data processing for image automatic analysis

PyParis 2018 - 15/11/18

Raphaël Delhome

Introduction

Oslandia...

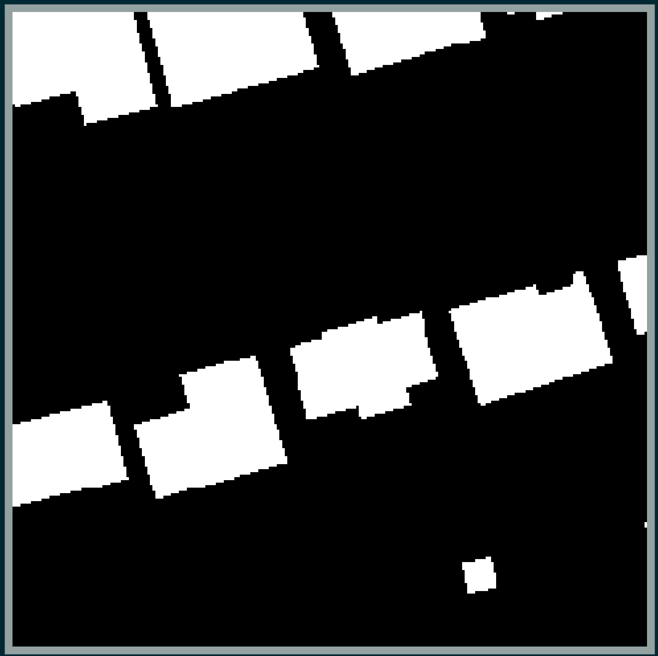
- since 2009
- Open Source specialist
- GIS experts (QGIS contributors)
- Provide geospatial data solutions
- today: 17 teammates

...and I

- At Oslandia for 1.5 year
- Data Scientist
- in charge of R&D actions

Context

- Artificial Intelligence at Oslandia
- Aerial image democratization
- A historic use case: building footprint detection



Deep learning and geospatial data

Image analysis use cases at Oslandia

Tech stack: Linux, Python (Keras, Pillow, ...)

Semantic segmentation

- Street-scene images
- Aerial images
- OpenStreetMap data parsing

(github.com/Oslandia/deeposlandia)

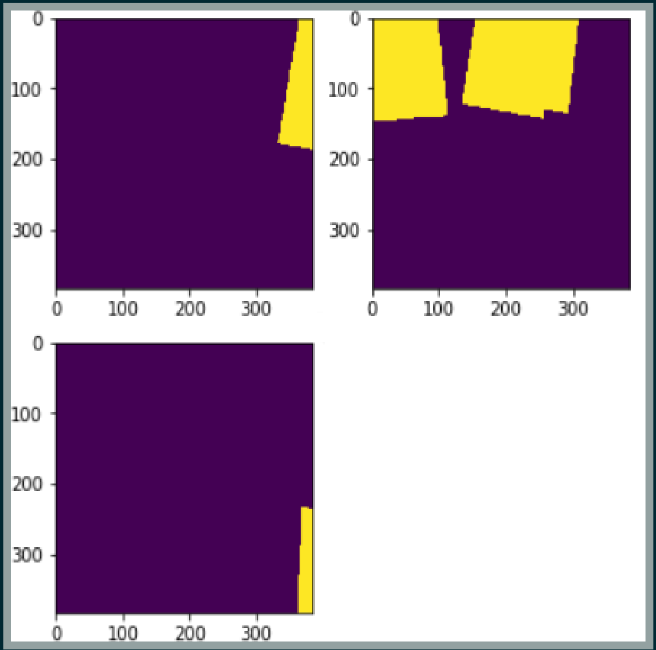
Instance segmentation

- Aerial images

Semantic segmentation

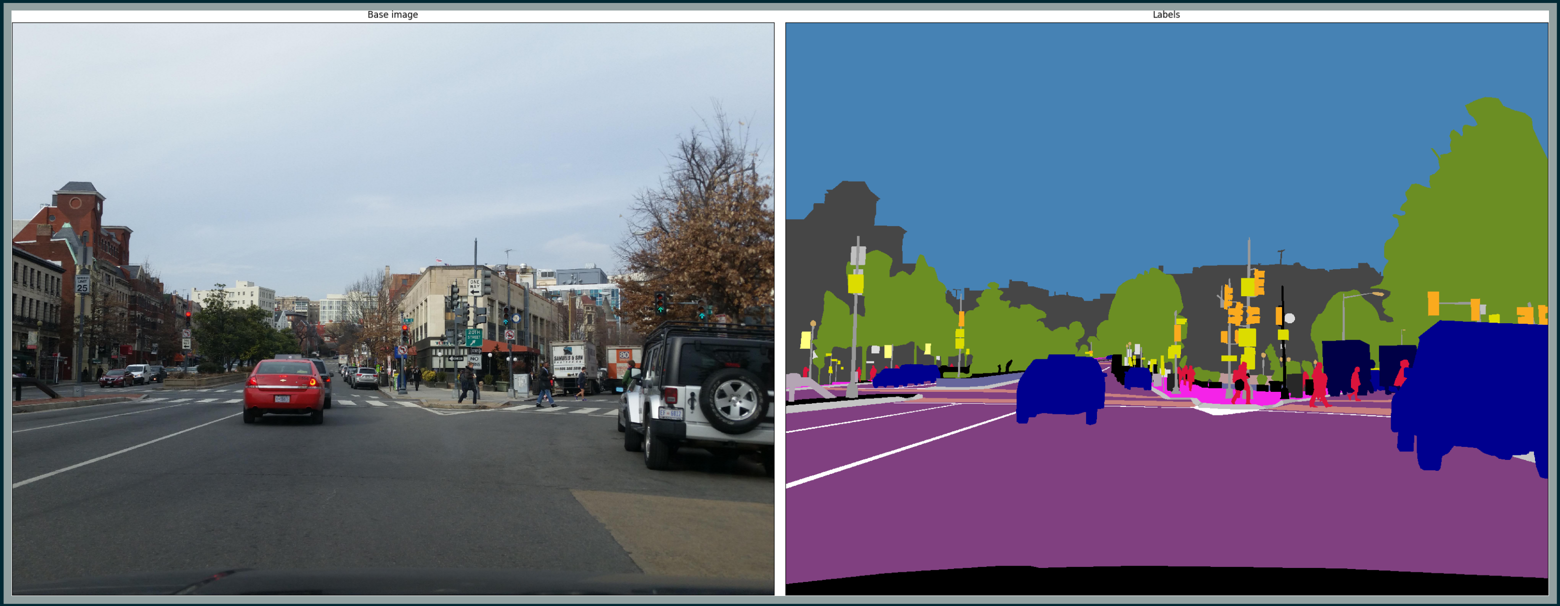
Inputs *N* images (*P* x *P* pixels, *C* channels), *L* labels

Outputs *N* arrays of shape *P* x *P* x *L*



Mapillary dataset

- *.jpg* images and *.png* labels (from 800x600 pixels to 5500x4000 pixels)
- 25000 images (18000 for training, 2000 for validation)



AerialImage (INRIA)

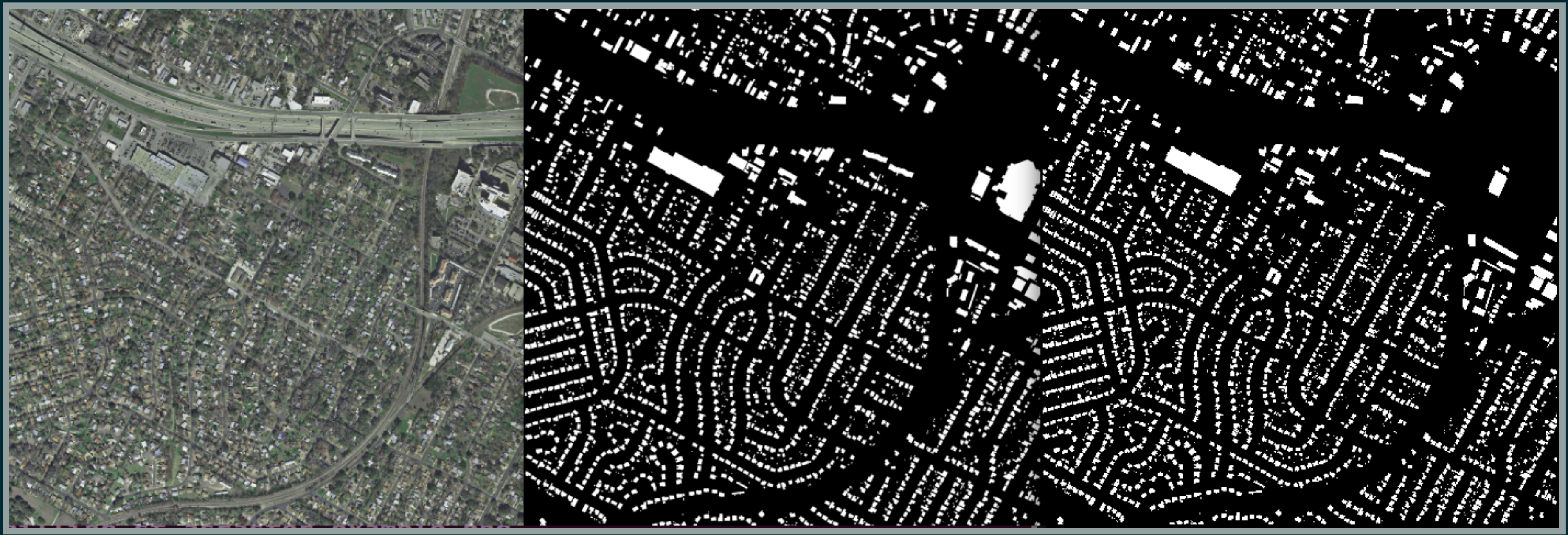
- Georeferenced *.tiff* images (5000 * 5000 pixels)
- 360 images (10 cities of 36 tiles each)
- 50% training, 50% testing

Link with OSM data

- Rebuild labelled images starting from OSM database
- OSM data as Ground-truth *OR* additional input data
- Process:
  - Extract coordinates (GDAL)
  - Query OSM data (Overpass)
  - Store the data in the database (osm2pgsql)
  - Generate raster tile (Mapnik)

(github.com/Oslandia/osm-deep-labels)

Link with OSM data

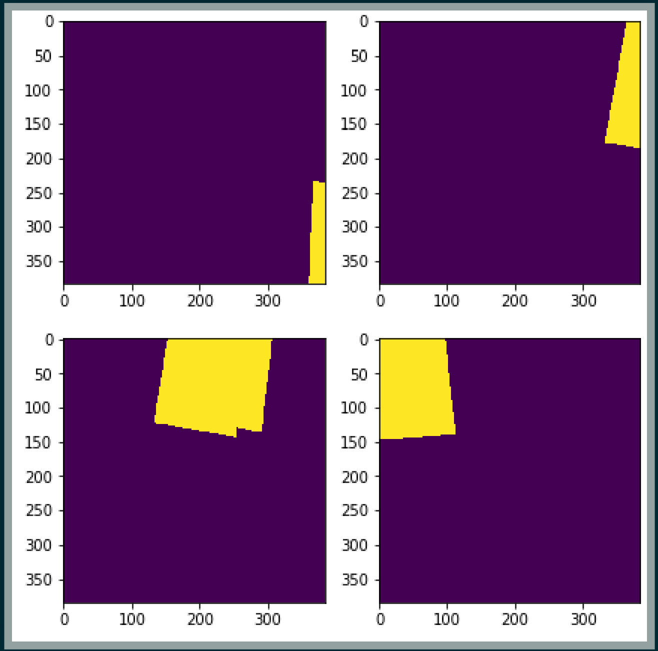


- Left : raw image
- Center : ground-truth label
- Right : OSM raster

Instance segmentation

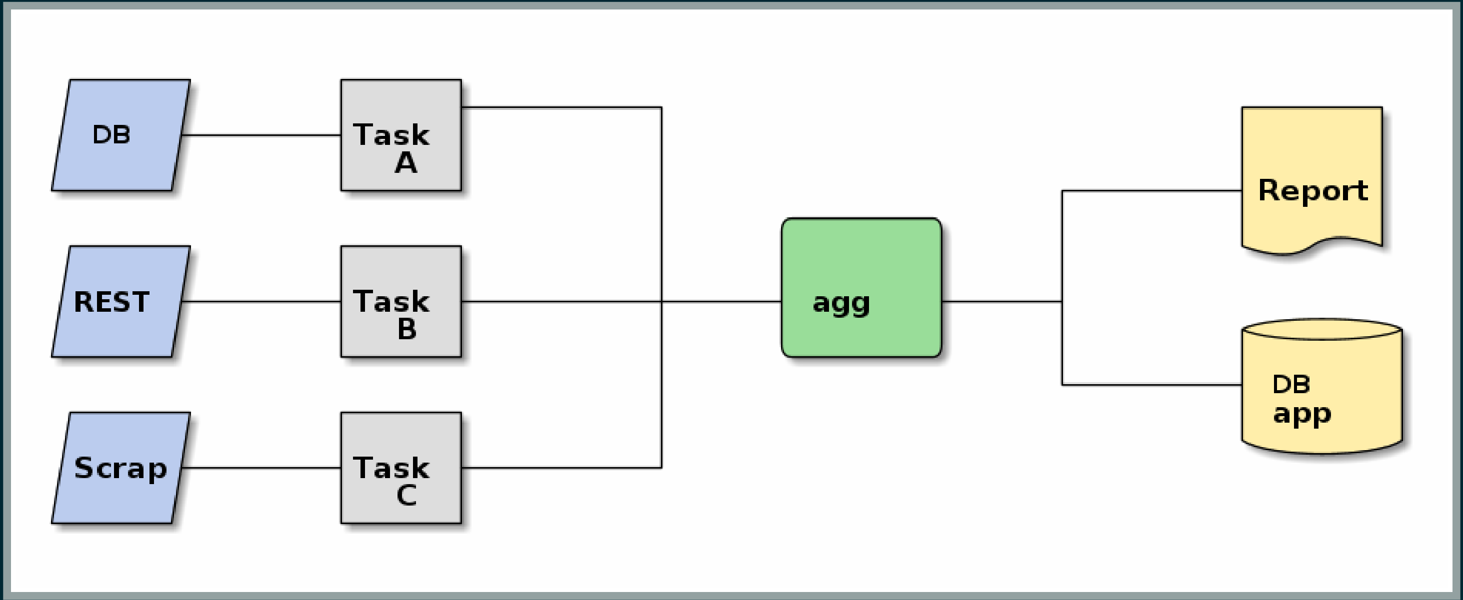
Inputs *N* images (*P* x *P* pixels, *C* channels), *L* labels

Outputs *N* arrays of shape *P* x *P* x *S*, with *S* the instance number (*cf* Mask-RCNN)



Main issue

Design a geospatial data pipeline for IA treatments :
Luigi package (1 operation = 1 pipeline task)



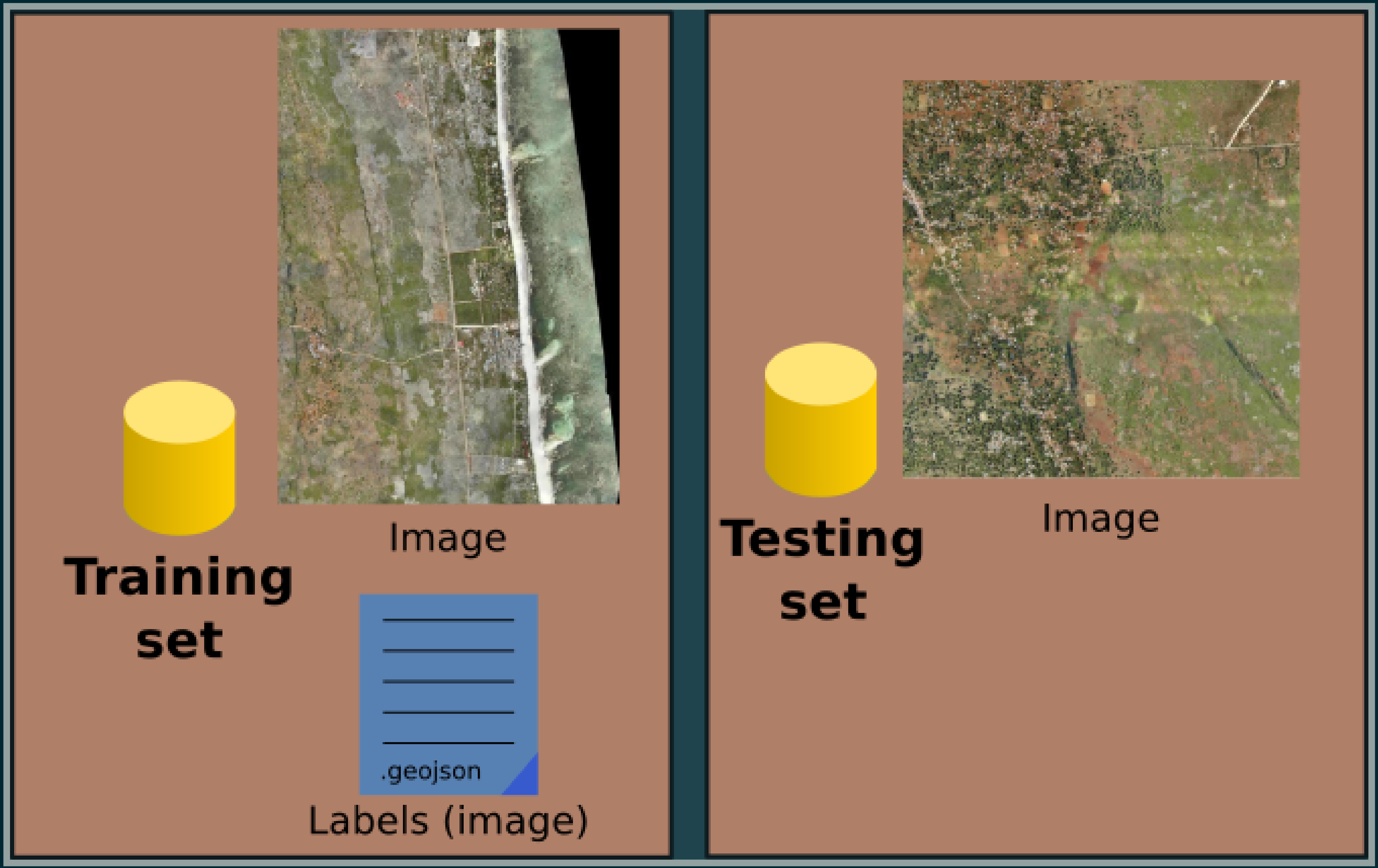
Tanzania challenge as an opportunity to implement it

Pipeline design

Tanzania challenge

- Challenge organized by WeRobotics
- Building instance detection and status discrimination (completed, unfinished, foundation) in Tanzania
- 13 images (from 17k x 42k to 51k x 51k pixels)

Data parsing



Data preprocessing

- Generate tiles: GDAL (integrated in the Python pipeline through `sh`)

```
gdal_translate -srcwin <min-x> <min-y> <tile-width> <tile-heigh
      <input-path> <output-path>
```

- Get geo-features: GDAL

```
from osgeo import gdal
ds = gdal.Open(filename)
# ds.RasterXSize, ds.RasterYSize,
# ds.GetGeoTransform(), ds.GetProjection()
```

Data preprocessing

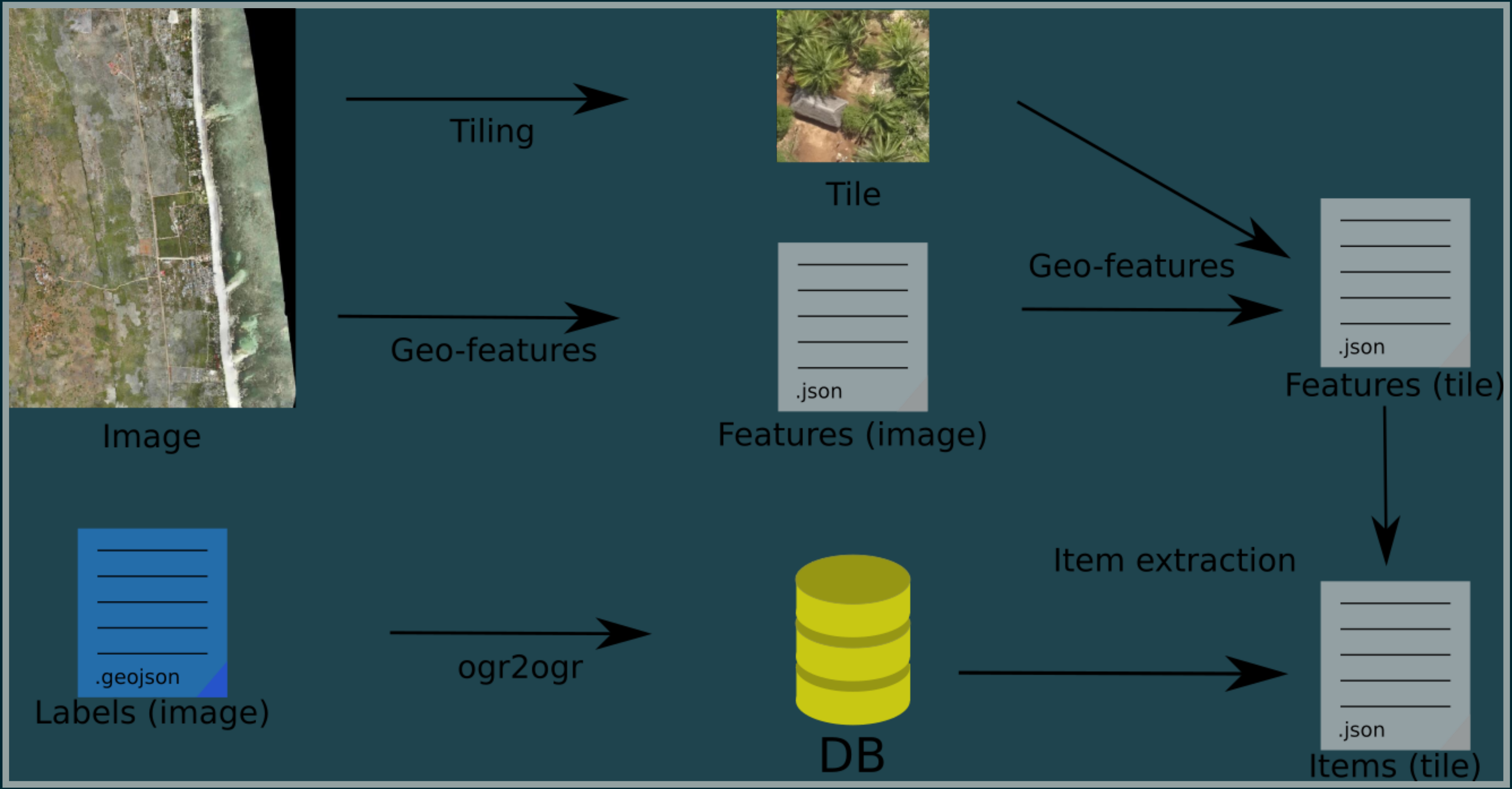
- Store labels to database: `ogr2ogr` (integrated in the Python pipeline through `sh`)

```
ogr2ogr -f PostGreSQL <conn-string> <input-path>
      -t_srs EPSG:<srid> -nln <table-name> -overwrite
```

- Extract tile items: `PostGIS` (and `psycopg2`)

```
WITH bbox AS SELECT(ST_MakeEnvelope(<bbox_coordinates>))
SELECT <building_intersection>
FROM <table> AS t JOIN bbox
ON ST_Intersects(t.geom, bbox.geom)
```

Data preprocessing

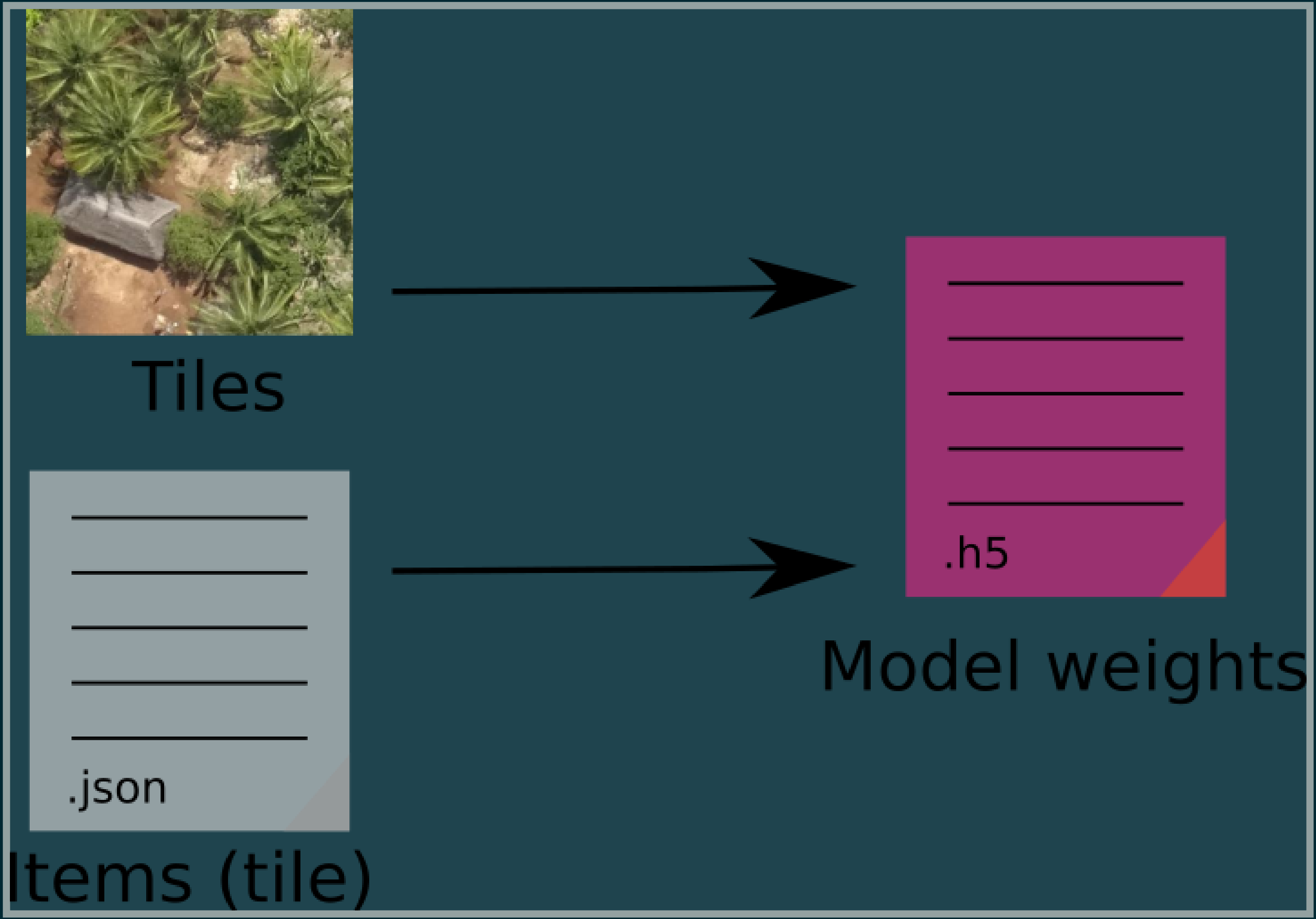


Model training

github.com/matterport/Mask_RCNN

- Instance-specific segmentation on various object types (complete buildings, incomplete buildings, foundations)
- Hyperparameter settings: number of training epochs? Learning rate?
- Hardware criticity: 1 GTX 1070Ti GPU

Model training



Model inference

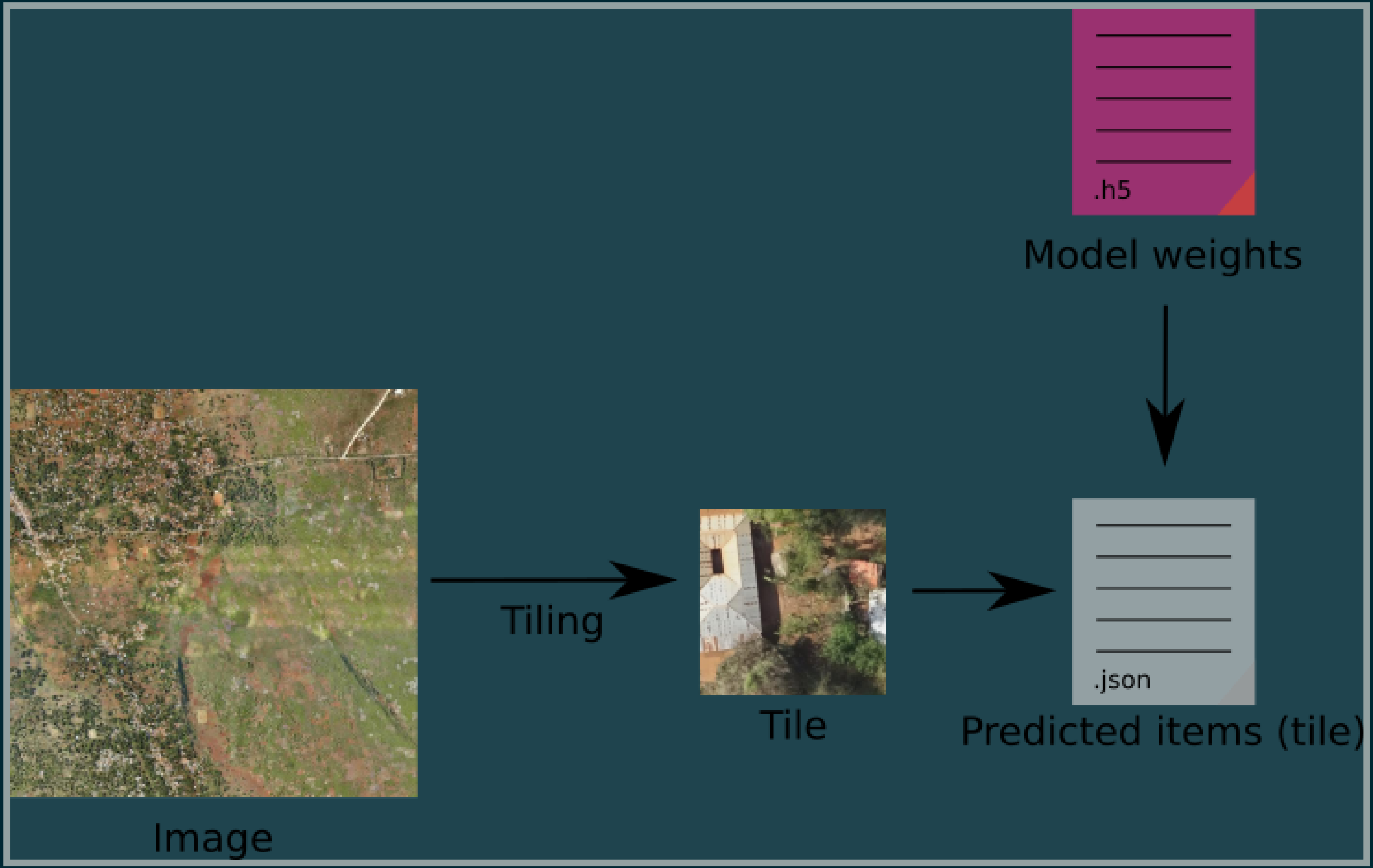
- Generate tiles on test images (*cf* training image processing)
- Prediction through Mask_RCNN package

```
from mrcnn import model as modellib

model = modellib.MaskRCNN(mode="inference",
                          config=<config>,
                          model_dir=<model_path>)
weights_path = model.find_last()
model.load_weights(weights_path, by_name=True)
result = model.detect(<image_data>)
```

Output: N boolean masks, N class_ids, N scores (N being the number of detected instances)

Model inference



Postprocessing

- Post-process detection output
  - Detect polygon contours within boolean masks: `OpenCV`
  - Transform pixels into geographical coordinates
  - Build polygons with `geojson` and `shapely`

```
geom = geojson.Polygon(<list-of-points>)
polygon = shapely.geometry.shape(geom)
```

Output: `.csv` files with building IDs, prediction scores, geometries

Postprocessing

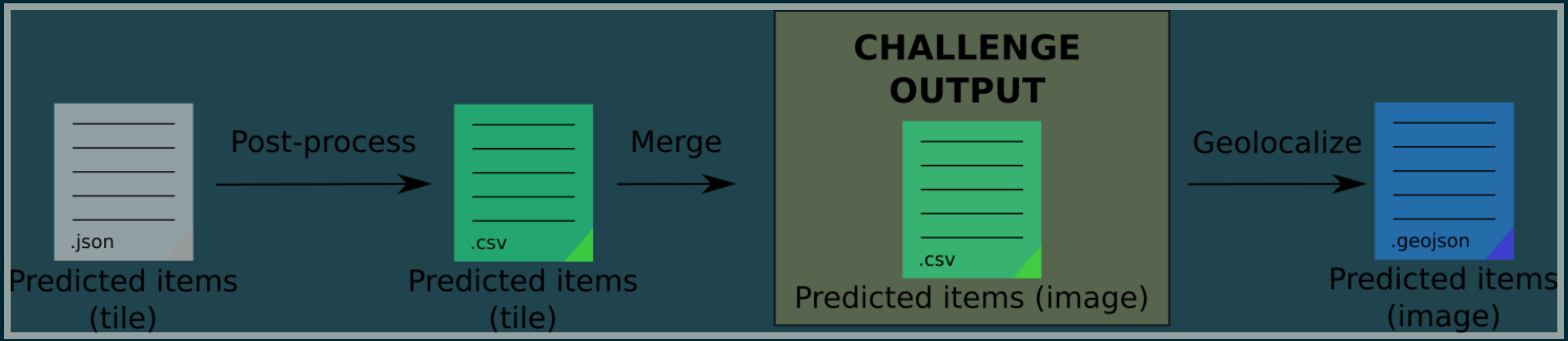
- Merge results: pandas

```
pred = pd.concat([pd.read_csv(filename)
      for filename in <postprocess_folder>])
```

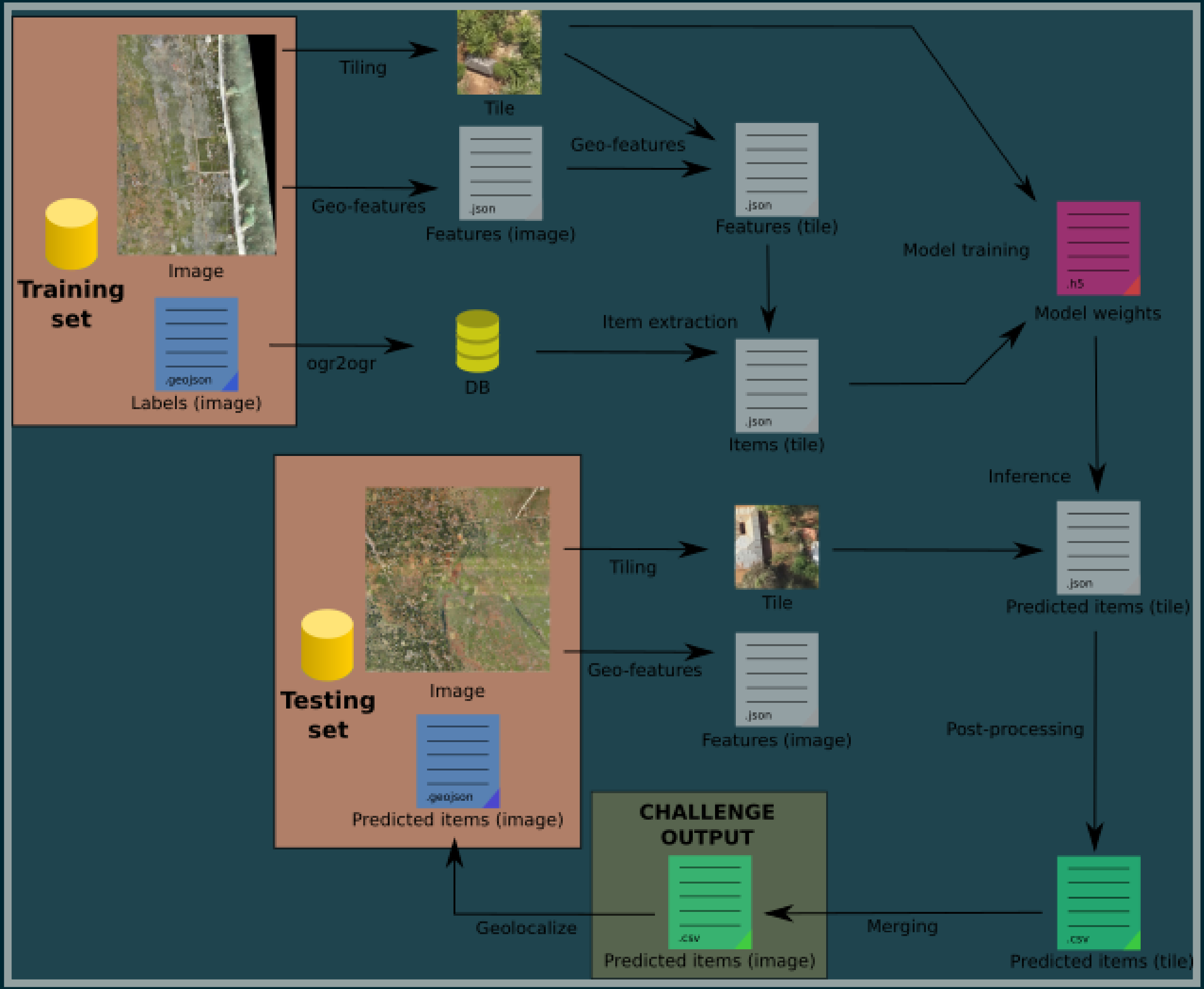
- Geo-localize results : shapely, GeoPandas

```
pred["geom"] = [shapely.wkt.loads(s)
      for s in pred["coords_geo"]]
gdf = gpd.GeoDataFrame(pred, geometry="geom")
gdf.to_file(<outputpath>, driver="GeoJSON")
```

Postprocessing



Put it all together



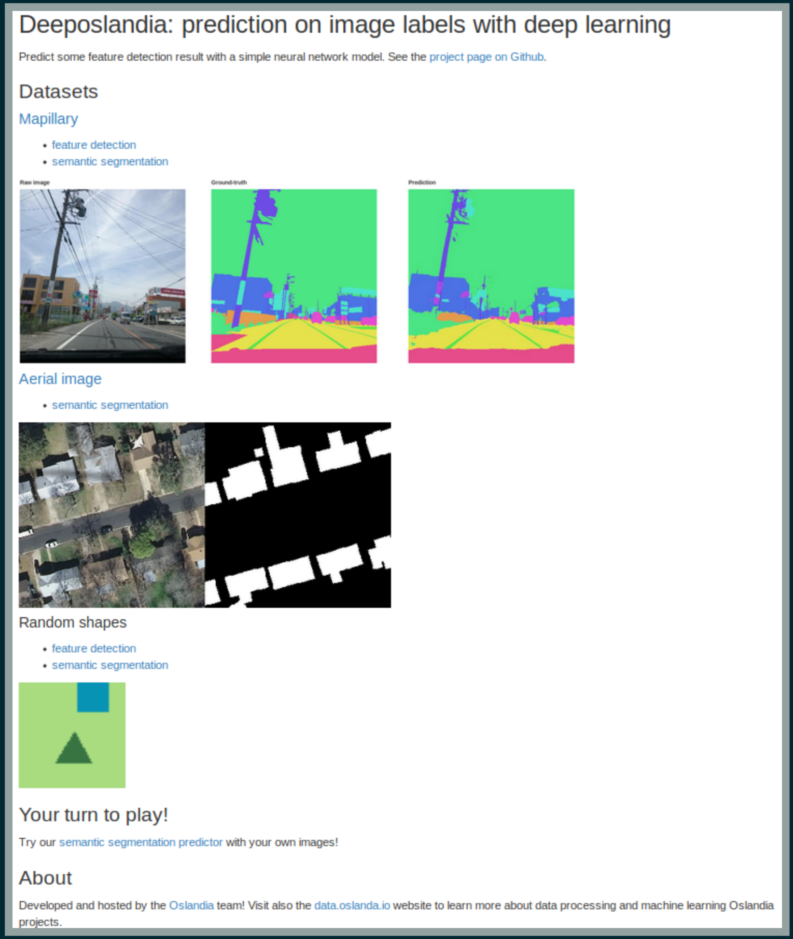
Result visualization

Conclusion

Output and discussion

- Geospatial data pipeline Proof of Concept
- ...However very poor results for now :-(
- Areas for improvement:
  - consider the images without any instance
  - manage identical building on adjacent tiles on Robosat manner
  - ...
- Still on processing! :-)

Bonus track: web app demo



Thank you for your attention!

Find out more:

- (Tanzania challenge code open sourced soon)
- https://oslandia.com/en/blog/
- github.com/Oslandia/deeposlandia
- github.com/Oslandia/osm-deep-labels
- http://data.oslandia.io