**Urban Mapper 3D - Solution Description**

**Overview**

Congrats on winning this marathon match. As part of your final submission and in order to receive payment for this marathon match, please complete the following document.

1. **Introduction**

Tell us a bit about yourself, and why you have decided to participate in the contest.

* Name: Victor Durnov
* Handle: cannab
* Placement you achieved in the MM: 3rd
* About you: I’m independent software developer/freelancer interested in hard algorithmic challenges and machine learning.
* Why you participated in the MM: I prefer to learn on practice and challenges like this one is a very good opportunity to learn something new.

1. **Solution Development**

How did you solve the problem? What approaches did you try and what choices did you make, and why? Also, what alternative approaches did you consider?

* Neural Networks are most suitable tool for such kind of tasks. I’ve treated the problem as image segmentation task (even though it’s instance segmentation task). So, the main tool was Encoder-Decoder Neural Networks (UNet with modified encoders and LinkNet architectures). (UNet paper: <https://arxiv.org/pdf/1505.04597.pdf> LinkNet: <https://arxiv.org/pdf/1707.03718.pdf> )
* Training resources were limited by contest rules and on my side (also trained on AWS), so I’ve used pretrained weights for Encoder part from https://keras.io/applications/ (ResNet50 and VGG16) – this helped to train Neural Networks much faster and improved accuracy.
* Masks quality was not so good, so I’ve fixed them with script (fix\_masks.py) using predictions of early Neural Network version: removed buildings on black, removed buildings without intersections with predictions (mostly buildings inside trees or missing on image). Also removed pixels of neighbor buildings closer than 6px to each other for better separation.
* Post-processing used to separate predicted masks better – 2nd level boosted trees (LightGBM) model trained to predict if mask is False Positive or True Positive and select the best threshold.

1. **Final Approach**

Please provide a bulleted description of your final approach. What ideas/decisions/features have been found to be the most important for your solution performance:

* Three kinds for Neural Network’s architecture used: Unet with pretrained ResNet50 encoder (separate encoder branch for nDSM=DSM-DTM channel, fusioned with main RGB channels in decoder part), Unet with pretrained VGG16 (nDSM channel used together with RGB channels), LinkNet trained from scratch (described in models.py and linknet.py). Unet models trained in 2 versions: original images and processed with CLAHE filter (contrast limited adaptive histogram equalization, opencv). So total 5 different model types.
* Train set splitted to 4 folds and each model type trained 4 times using each fold as validation. So I had out-of-fold (OOF) predictions of entire train set for each Neural Network type.
* As loss function for training used combination of Binary Crossentropy and Dice coefficient. For some models I’ve tried to give more weight for pixels near the building border.
* Models trained on 512\*512 random crops with random horizontal flip augmentation. Prediction made on full image size also with horizontal flip test time augmentation. Predictions of all models just averaged.
* Building’s candidates found using 3 different thresholds: 70 of 255 (main candidate), 130 of 255, 130 of 255 + erosion. Then for each candidate extracted following features to train LightGBM models:
* Area
* Is Contour Convex or Not, Convex area
* Min Area Rectangle’s features: min side, max side, side’s ratio
* Solidity, eccentricity, extent
* Perimeter
* Major and minor axis length
* Mean and std for RGB values, prediction values, nDSM
* Neighbor candidates count in distance of 100, 200, 300, 400 meters
* Median area of neighbors and it’s ratio to candidate area
* 200 LightGBM models trained to detect True Positive candidates and False Positive using random 50% of data for validation and OOF predictions made for whole train set. For each candidate selected threshold/version with max prediction. Then final threshold found using F1 metric.

1. **Open Source Resources, Frameworks and Libraries**

Please specify the name of the open source resource along with a URL to where it’s housed and it’s license type:

* Anaconda as base Python 3 environment, www.anaconda.com
* Tensorflow, www.tensorflow.org Apache License
* Keras, https://keras.io The MIT License
* OpenCV, <https://opencv.org> BSD License
* LightGBM, <https://github.com/Microsoft/LightGBM> The MIT License

1. **Potential Algorithm Improvements**

Please specify any potential improvements that can be made to the algorithm:

* Improve masks quality/accuracy
* Add more Neural Network types to ensemble and train for longer time

1. **Algorithm Limitations**

Please specify any potential limitations with the algorithm:

* It was hard to detect right building shape inside trees (when only part visible) and separate or join big complex buildings. I think better masks quality can help to train better models.

1. **Deployment Guide**

Please provide the exact steps required to build and deploy the code:

Dockerized version prepared as requested. For clean installation python 3 required with libraries (all in anaconda3 default installation): numpy, sklearn + install LightGBM, OpenCV, Tensorflow, Keras

1. **Final Verification**

Please provide instructions that explain how to train the algorithm and have it execute against sample data:

train.sh and test.sh scripts meet required specification.

1. **Feedback**

Please provide feedback on the following - what worked, and what could have been done better or differently?

* Problem Statement – very good
* Data – masks could be better, also additional channels can help (like infrared)
* Contest – very good contest. Liked it
* Scoring – good scoring

**NOTE**: Please save a copy of this template in word format. Please do not submit a .pdf