**Marathon Match - 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.

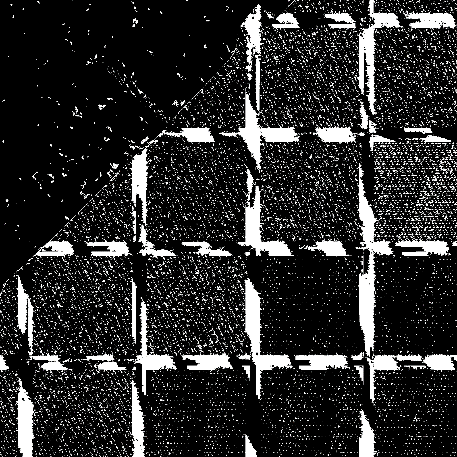
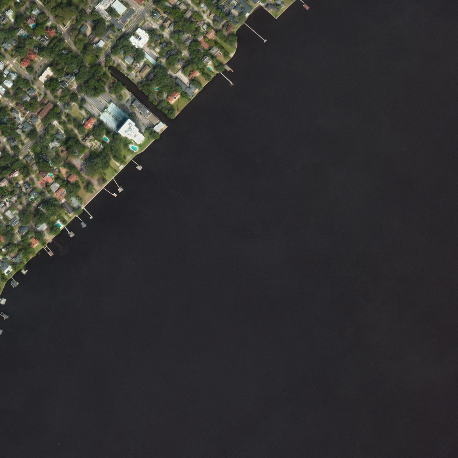
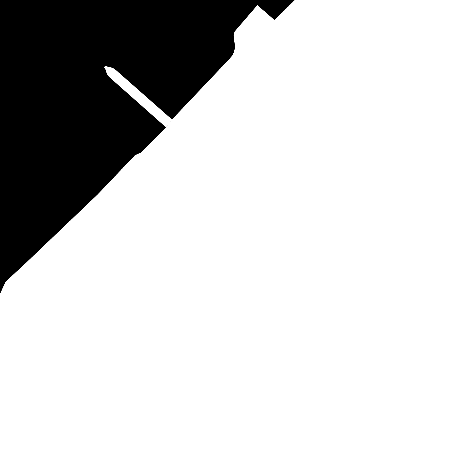
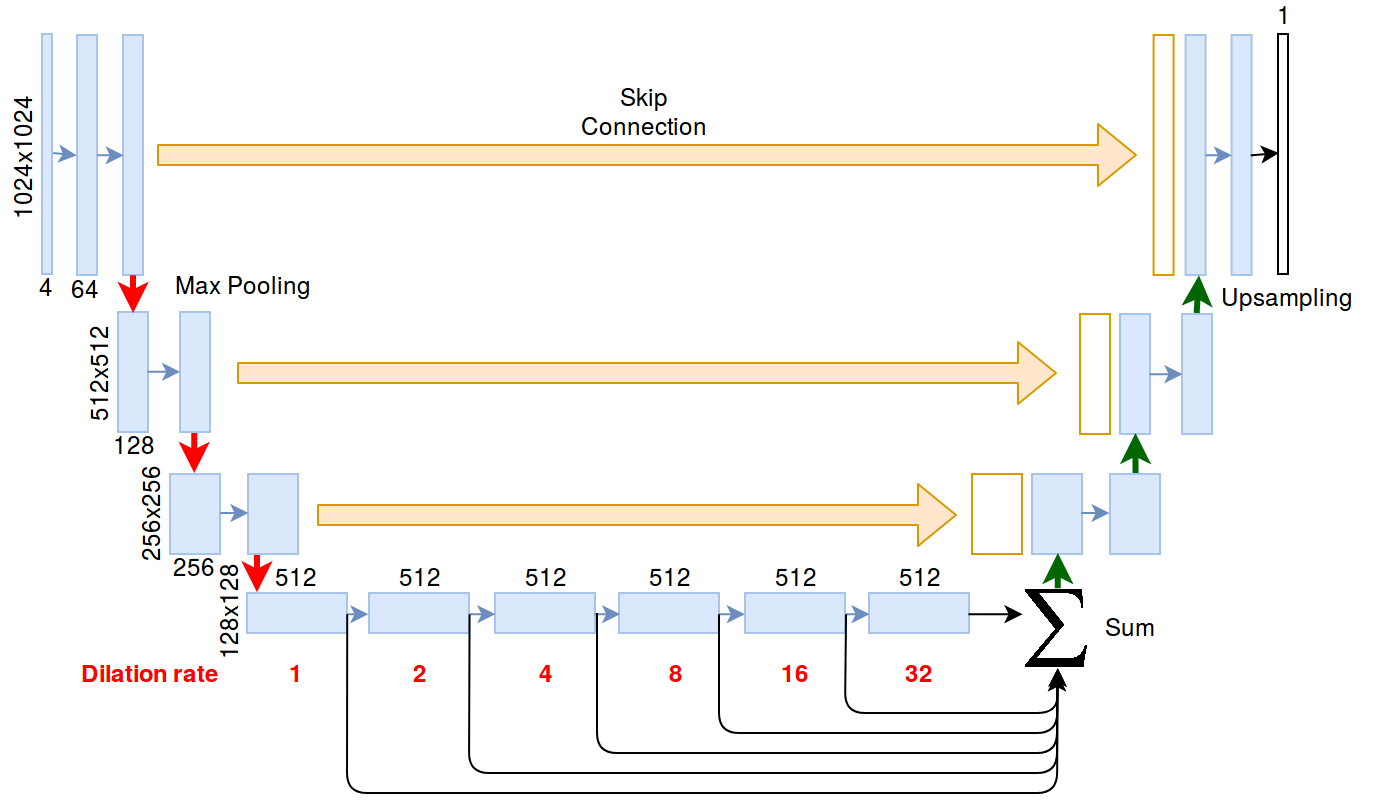
1. **Introduction**

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

* Name: Alina Elena Marcu
* Handle: alina.marcu
* Placement you achieved in the MM: 4
* About you: I have graduated the Faculty of Computer Science and Information Technology of the University “Politehnica” of Bucharest, I have a Master’s Degree in Artificial Intelligence and currently I am a PhD student at the Institute of Mathematics “Simion Stoilow” of the Romanian Academy.
* Why you participated in the MM: I’m studying semantic segmentation of aerial images for my PhD. Most of my training sets are RGB-only – I thought this might be a good opportunity to assess the performance gain of adding a 4th layer (depth) and, of course, see how my solution stacks up against others in an international competition.

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?

* Data preprocessing
  + Normalized DSM – DTM and then clipped the values to -20 (lower bound) and 30 (upper bound).
  + Sliced original 2048x2048 input data in overlapping 1024x1024 patches with a stride of 512.
  + Applied standard data augmentation techniques (random angle rotations and color jittering).
* Water masking
  + Since depth data was very noisy for regions that contained water, we downloaded water polygons from OpenStreetMap for the training dataset and made a water detection model (pictured below, RGB, water label, DSM-DTM)
  + For faster convergence, I only trained the model with patches that contained of at least 100 pixels of water regions
  + I then masked the RGB and DSM data with the detected water regions.
  + Unexpectedly, this also helped remove several other non-house regions, such as several tree patches and noisy warehouse rooftops.
* Trained a modified U-Net with dilated convolutions
  + 3x3 convolutions.
  + Added more dilated convolutions at the middle layers.
  + A detailed figure of the network is shown below:
* Trained Mask R-CNN for instance detection
  + Unfortunately, this hurt detection performance for large buildings with many wings, and was supposed to be used only for splitting smaller house patches
  + There were only a few small houses that benefitted, so I considered the training overhead too significant for the performance gain.
* Trained individual networks based on the building size
  + Since warehouses (and their noisy depth data) were totally different from small houses, I tried training 3 networks – one for warehouses (surface area >= 5000 pixels), one for medium-sized houses (surface area > 500 pixels and < 5000) and one for small houses (surface area <= 500 pixels).
  + Unfortunately, only warehouse detection yielded satisfactory results and then again the training overhead was deemed too high for the performance gain.
  + The small houses that were supposed to be split into instances had a poor f-measure score compared to the all-in-one approach.

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:

* Mask water regions – the depth data proved to be particularly noisy in those regions and hurt training performance
  + Masking water also helped reduce the depth noise on large, flat regions (such as warehouses).
* Iterative training to correct labels
  + Since a healthy amount of labels were noisy (mostly houses occluded by trees), this helped the algorithm learn with better annotations.
* More training data
  + After predicting on the testing set, we added the result to the training set and retrained the neural network.

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 its license type:

* Keras, <https://github.com/keras-team/keras> MIT license
* Tensorflow, <https://github.com/tensorflow/tensorflow/> Apache license
* OpenStreetMap, <https://www.openstreetmap.org/> OdbL license
  + water polygon shapefiles: <http://openstreetmapdata.com/data/water-polygons>
* Overpass API <https://github.com/drolbr/Overpass-API> AGPL license
* U-Net variation <https://github.com/lyakaap/Kaggle-Carvana-3rd-Place-Solution>
* GDAL <http://gdal.org/> , MIT license
* Python3 <https://www.python.org/> , PSFL license
* Shapely <https://github.com/Toblerity/Shapely> BSD 3-clause license
* Scikit-image, <http://scikit-image.org/> BSD 3-clause license
* Mahotas <https://github.com/luispedro/mahotas>, MIT license
* Nvidia-docker, <https://github.com/NVIDIA/nvidia-docker>, BSD 3-clause license
* Docker, <https://www.docker.com>, Apache 2.0 license, BSD 3-clause license

1. **Potential Algorithm Improvements**

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

* Instance detection
  + House clusters are detected as a single instance, this needs to be addressed.
* Multi-scale detection
  + There should be an algorithm that decides whether to join or not a specific building wing based on the area of the detection.
* Remove ‘other’ classes
  + A tree detector would have helped a lot to remove the ‘houses on trees’ problem and limit the search domain.

1. **Algorithm Limitations**

Please specify any potential limitations with the algorithm:

* No instance detection
  + House clusters are detected as single instances.
* Issues detecting
  + Very small instances (usually partially occluded).
  + Very large instances with wings connected with a small number of pixels (e.g., bridges).
* High quality imagery is required for a good detection.

1. **Deployment Guide**

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

1. Install the prerequisites (nvidia-docker)
2. Build image: docker build -t urban3d .
3. Open terminal inside image: docker run --runtime=nvidia -v /data:/data -it test1 bash
4. Test: test.sh /data/train /data/test /data/alina.marcu.txt
5. **Final Verification**

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

1. Install nvidia-docker, with all prerequisites (proper driver etc)
2. Build image: docker build -t urban3d .
3. Open terminal inside image: docker run --runtime=nvidia -v /data:/data -it test1 bash
4. Train: train.sh /data/train /data/test
   * Please note that the testing data is also required for training
5. Test: test.sh /data/train /data/test /data/alina.marcu.txt
   * This produces /data/alina.marcu.txt
6. **Feedback**

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

* Problem Statement
  + The problem was clear – instance-wise house detection
  + The ‘ignored houses’ problem could have been avoided by providing images without back areas.
* Data
  + The labels could have been better – some were (poorly) manually labeled, some didn’t take into account the underlying vegetation.
  + The RGB images were fairly blurred
    - This is probably due to the post-processing for removal of foreign objects, such as cars, but this means interpolated values and the result probably hurt training (compared to a snapshot).
* Contest
  + The progress prizes were a nice touch; however, the last one set the bar too high, given the quality of the data and inherent ambiguity of the task.
* Scoring
  + The algorithm scored with 0 small houses with holes, but there were cases when there was actually a hole in the house (inner garden, for example) and we believe they were scored inappropriately.
  + Additionaly, buildings touching the edge of the image in the ground truth, even with a single pixel, were not scored as a whole – again, this affected detection performance.

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