**Marathon Match - Solution Description**

1. **Introduction**

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

* Name: Selim Seferbekov
* Handle: selim\_sef
* Placement you achieved in the MM:
* About you: During the day I’m a Machine Learning Engineer working on projects aimed to improve maps. As a hobby I participate in a lot of competitions related to Deep Learning (mostly Computer Vision).
* Why you participated in the MM: I participated in all Spacenet challenges starting from Spacenet 3. I would not have an excuse to miss this one.

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?

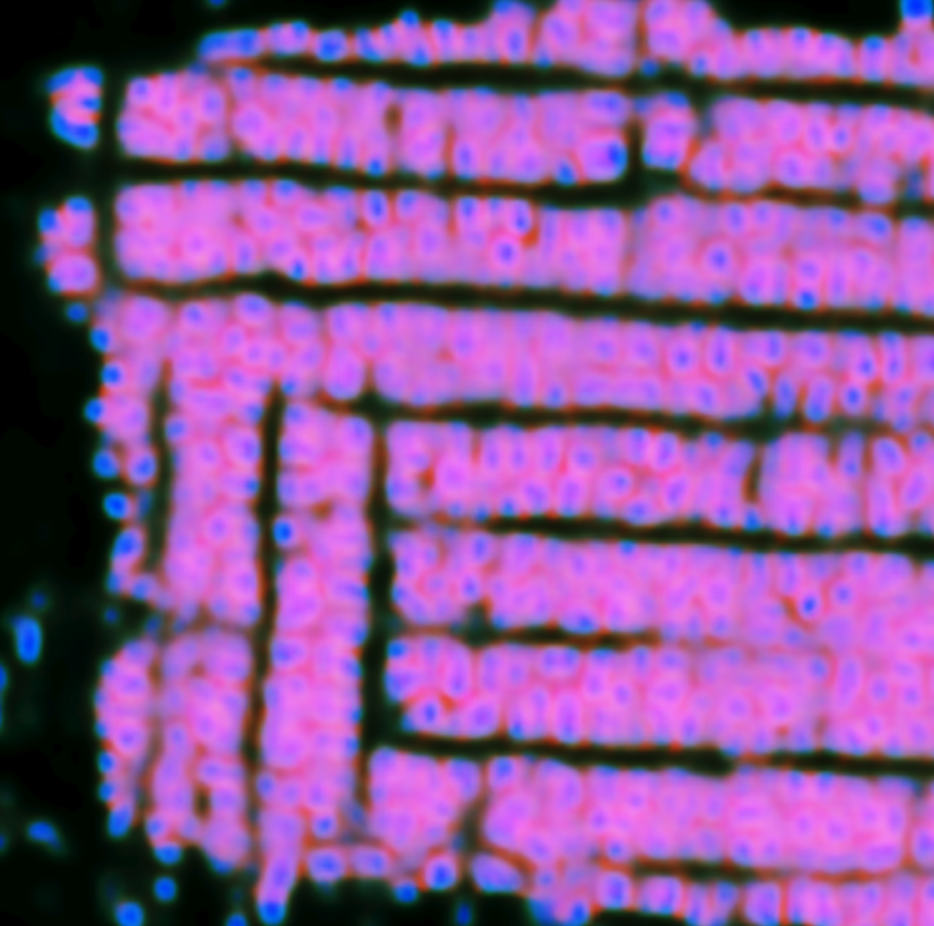
**Instance segmentation**

Certainly F1 change score would rely on better building detections.

I started with a baseline that used a standard approach with three-channel masks and watershed post processing.

Small resolution was an obstacle to that as it was not possible to separate very small buildings. Models usually over segmented these parts with touching borders. Even with resized images

(3072x3072) predicted segmentation masks looked weird.

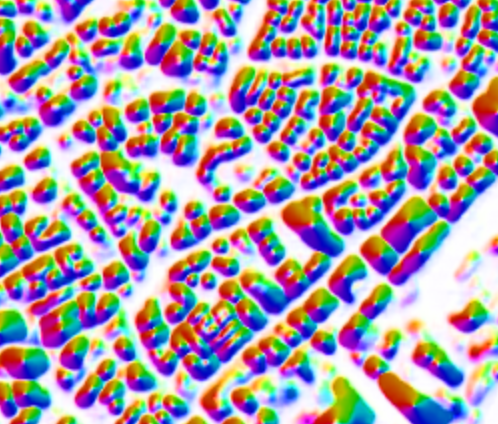


Watershed failed miserably in these places. I decided to try a multitask learning approach similar to Panoptic Deeplab where 3 targets are predicted

* Segmentation mask (three channel mask in my case)
* Center mask (with gaussians)
* Offset from centers (X, Y in pixels)



Offsets were looking nice but I could not find a use for them. Adding this offset head to a model also led to unstable training in mixed precision.



In the end I decided to predict only segmentation masks and gaussian centers.

I used center masks to remove touching borders pixels in centers. And then applied a standard watershed approach from previous challenges which uses 3-channel masks. This resulted in 10% F1 track score improvement for complex images. Also it allowed me to get a good provisional score even with small 2048x2048 images.

**Tracking**

At first I wrote a simple tracker that used KDTrees to find intersecting building instances on different slices, it was much faster than the baseline version.

I tried to use optimization (linear sum assignment) but it was a bit worse than a simple greedy approach.

Validation scores were very low due to a lot of false positive change predictions. After adding thresholds (i.e. leave only buildings that exist on 5+ slices) the score improved significantly.

Also I spotted that sometimes buildings disappear between frames even though they exist on images. I added a logic that fixes this issue and fills missing buildings between frames

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:

* I used EfficientNet B6, B7 as encoders for semantic segmentation
* To have a larger batch size/crops I trained all models with Nvidia Apex in mixed precision.
* As a segmentation loss function I used a combination of soft dice loss + binary cross entropy + focal loss.
* For center detection I used MSE loss as it worked better than binary cross entropy in this case.
* Augmentations: Flips, rotations, color jittering, random sized crops. I also implemented random sized crop around buildings - this leads to much faster convergence and better segmentation quality (because most of the images will have some non zero pixels, otherwise with large images it is not the case).
* Test time augmentations (TTA): horizontal/vertical flips
* For tracking I implemented a custom algorithm that removes most of the false positive predictions (and some true positives of course) and also fills gaps in datacubes.

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:

* Docker, https://www.docker.com (Apache License 2.0)
* Nvidia-docker, https://github.com/NVIDIA/nvidia-docker, ( BSD 3-clause)
* Python 3, https://www.python.org/, ( PSFL (Python Software Foundation License))
* Numpy, http://www.numpy.org/, (BSD)
* Tqdm, https://github.com/noamraph/tqdm, ( The MIT License)
* Anaconda, https://www.continuum.io/Anaconda-Overview,( New BSD License)
* OpenCV, https://opencv.org/ (BSD)
* Pytorch https://pytorch.org/ (BSD)

1. **Potential Algorithm Improvements**

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

* Using Siamese networks with segmentation + centers approach would improve scores significantly. My initial experiments did not bring any success and I decided to drop them. At the very late stage of the competition I found a bug in mask generation code that affected siamese architecture training. I made another experiment and it boosted validation scores by a few percent but I did not have time for a big code change.

1. **Algorithm Limitations**

Please specify any potential limitations with the algorithm:

* Even with center regression watershed doesn’t split very small buildings correctly

1. **Feedback**

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

* Problem Statement - problem statement is very detailed
* Data - the dataset was well designed. Validation correlated well with the provisional leaderboard.
* Contest - the contest is great as all Spacenet challenges are!
* Scoring - scoring metric is a bit tough to optimize. Fortunately we had a very good local scorer and visualizer tool.

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