**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: Alexander Buslaev
* Handle: albu
* Placement you achieved in the MM: 1
* About you: I have 5 years experience in classical computer vision and worked in a number of companies in this field, especially in UAV. About a year ago I started to use deep learning for various tasks in image processing - detection, segmentation, labeling, regression.
* Why you participated in the MM: It was interesting task and interesting metric

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?

* Firstly, I used my code from previous competitions to train classifier on RGB images. Architecture was linknet with resnet34 encoder as default choice.
* After I added dsm/dtm data to input of classifier because this information is important. There are different ways of adding this channel to architecture. I decided the simplest one – just throw out first layer weights and retrain them from scratch. It worked fine so I did not investigated a lot into this. In addition, I heat up weight by training everything 5 epochs without dsm/dtm layer. It should lead to better convergence.
* Then I made smart post processing to distinct different instances of objects. It was following: generate binary maps by thresholding probability map on 0.3 and 0.7. First map was space for division and second map was seeds for watershed algorithm. After running watershed we have areas for each house from first map and instances from second map.
* Lastly I invested some time to optimize neural network architecture. There are many alternatives, but I tried only few because there were not much time for this. I tried: resnet{18,34,50}, inception-resnet-v2, inception-v3 encoders with linknet-like and unet-like decoders. The best architecture for me was resnet34 with unet-like decoder. I always use pre-trained networks for faster convergence.
* During final re-training model performed better then on my setup, here is why:

Initially I tried many configurations and converged to current only in the end of competition. I have four gtx 1080 GPU each of which has 8 GB video memory. Last submission was trained in this setup: 4 GPU, 2 images in one GPU (around 5.5-6gb).

To emulate many GPU training on one GPU everyone uses trick called iter\_size. It stands for accumulating gradients from sequential steps and make optimizer step using accumulated gradients, so effective batch size gets bigger.

I did it too. So setup became 3 images per GPU (around 8gb), iter\_size 3 (effective batch size 9). It is hard to train model that consumes all the memory, it throws OUT OF MEMORY errors sometimes because of bugs in pytorch and cudnn. After competition end I re-trained everything in this setup (it consumed much more time) and on local validation it looked about the same (I did not have chance to test it against provisional data), so I submitted this setup for re-training. It works fine and does not throw OOM errors because p2.xlarge have 12 GB of memory.

Bigger effective batch size is always better because we make better steps to local minima.

Bigger per- GPU batch size is also always better for architectures that use BatchNorm layers (as resnet34 that stands for encoder in my architecture). BatchNorm layer computes statistics (mean, stddev) on every batch and use them on validation stage. All DL frameworks use this statistics only from first GPU, but with iter\_size and one GPU, we have three images in batch and all statistics from every iter. Statistics in your re-training are just more precise then mine.

To emulate behavior of four GPU better we need to do batch size 2 and iter\_size 4 (effective 8) and don't update BatchNorm statistics on last 3 iterations of every effective batch (what is not that straightforward).

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:

* So finally I came to this pipeline:
* Split data to 5 folds randomly
* Use resnet34 as encoder and unet-like decoder (conv-relu-upsample-conv-relu) with skip connection from every layer of network. Loss function: binary\_cross\_entropy + (1 – dice\_coeff). First 40 epochs components had the same weight, but last 2 epochs I changed components weights to 1.5 and 0.5.Optimizer – Adam with default params
* Heat up weights training 5 epochs without dsm/dtm layer.
* Train 25 epochs with lr 1e-4
* Train 10 epochs with lr 1e-5
* Train 2 epochs with loss, which have different weights for components. It should prevent hard overfitting, but I’m not sure
* Merge folds by mean
* Finally run watershed algorithm on probability maps.

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:

* tqdm (<https://pypi.python.org/pypi/tqdm>), MPLv2, MIT
* numpy (<https://pypi.python.org/pypi/numpy>), BSD
* opencv-python (<https://pypi.python.org/pypi/opencv-python>), MIT
* matplotlib (<https://pypi.python.org/pypi/matplotlib>), BSD
* scipy (<https://pypi.python.org/pypi/scipy>), BSD
* scikit-image (<https://pypi.python.org/pypi/scikit-image>), Modified BSD
* scikit-learn (<https://pypi.python.org/pypi/scikit-learn>), BSD
* tensorboardX (<https://pypi.python.org/pypi/tensorboardX>), MIT
* pytorch (<http://pytorch.org/>), BSD
* torchvision (<https://pypi.python.org/pypi/torchvision>), BSD
* GDAL (<https://anaconda.org/conda-forge/gdal>), MIT

1. **Potential Algorithm Improvements**

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

* Consider trying other ways of initializing weights for network with dsm/dtm layer
* Try different thresholds for watershed
* Train bigger batch size (maybe reducing crop size)

1. **Algorithm Limitations**

Please specify any potential limitations with the algorithm:

* It should not generalize to new kinds of data (big difference in weather conditions, zoom, etc); it is limitation for all machine learning algorithms.

1. **Deployment Guide**

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

1. Please use steps from Dockerfile. If you use clean system – you also need to install nvidia driver, cuda 8, cudnn 6.
2. **Final Verification**

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

1. It’s mostly described in “final verification” document (<https://docs.google.com/document/d/16We8eHYM58Cm4dYGF34Nb07Xe02Dcqtci8OWJBuo8Yg/edit>). If you run training, then for testing on new trained models you can use script “test\_retrain.sh” instead of “test.sh”
2. **Feedback**

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

* Problem Statement - OK
* Data – data was not accurately labeled
* Contest - OK
* Scoring – it could be faster using more or better AWS.

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