Best practices for elegant experimentation in data science projects (case study)

Jakub Czakon[1], *Kamil A. Kaczmarek*[1],
Andrzej Pyskir[2], *Piotr Tarasiewicz*[2]
(alphabetical order)

Introduction and Motivation

In the course of the project, data scientists face multiple issues. Difficulties with reproducibility, lack of the ability to prepare experiments quickly and dirty data are just three examples. Data science projects involve a lot of experimentation and quick adoption of new ideas and technologies. Such environment makes it difficult to keep the code clean as well as keep track of small changes that makes new experiment successful.

Here, we use an instance segmentation challenge - called Mapping Challenge - hosted on the crowdAI platform to show:
1) our **best practices when working in data science projects**,
2) **competition results.**

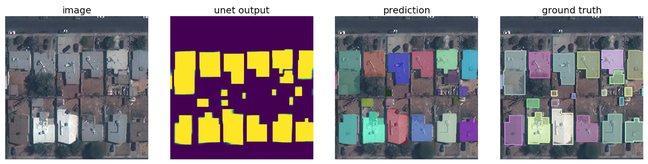
Our best practices involve **usage of the steppy library**, which provides minimal interface for building machine learning pipelines. Besides this, **we organized our work in a transparent and open way, publishing code, tasks and experiments results**.

On the poster, we share our results **regarding pre- and post-processing routines, network architectures and training scheme**. We also present technology stack that we use. It is a blend of well established Python packages (like numpy and sklearn) and our own open source initiatives, that is steppy and steppy-toolkit.

Results

AVERAGE PRECISION **0,943**

AVERAGE RECALL **0,954**



Tech stack

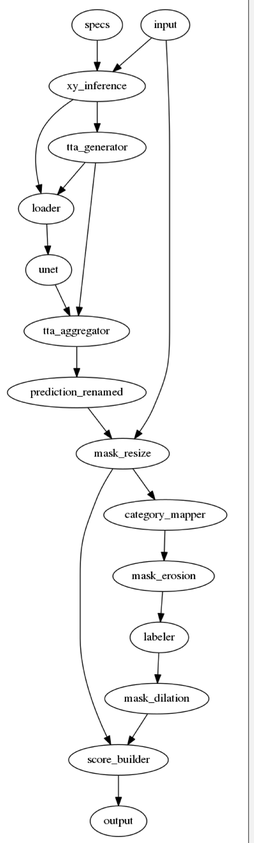
Python stack for data science:

Steppy-toolkit: https://github.com/neptune-ml/steppy-toolkit
Curated set of transformers that make your work with steppy faster and more effective.

Steppy: https://github.com/neptune-ml/steppy
Steppy is a lightweight, open-source, Python 3 library for fast and reproducible experimentation. Steppy's minimal interface does not impose constraints, however, enables clean machine learning pipeline design.

Neptune: https://neptune.ml
Tool for data scientist that facilitates clean process in data science and machine learning projects.

Pipeline diagram



Preprocessing

√ What Worked

- Overlay binary masks for each image is produced.
- Distances to the two closest objects are calculated creating the distance map that is used for weighing.
- Size masks for each image is produced.
- Dropped small masks on the edges.
- We load training and validation data in batches:
  using torch.utils.data.Dataset and torch.utils.data.DataLoader makes it easy and clean.

× What didn't Work

- Ground truth masks are prepared by first eroding them per mask creating non overlapping masks and only after that the distances are calculated.
- Dilated small objects to increase the signal.

Network

√ What Worked

- Unet with Resnet34, Resnet101 and Resnet152 as an encoder where Resnet101 gave us the best results.

× What didn't Work

- Network architecture based on dilated convolutions.

Postprocessing

√ What Worked

- Test time augmentation (tta). Make predictions on image rotations (90-180-270 degrees) and flips (up-down, left-right) and take geometric mean on the predictions.
- Second level model. We tried Light-GBM and Random Forest trained on U-Net outputs and features calculated during postprocessing.

× What didn't Work

- Test time augmentations by using colors.
- Inference on reflection-padded images was not a way to go. What worked better (but not for the very best models) was replication padding where border pixel value was replicated for all the padded regions.

Training scheme

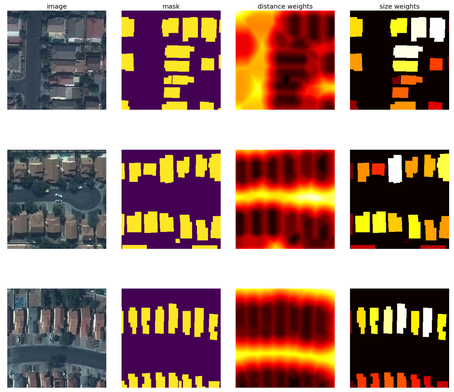
√ What Worked

- Use pretrained models!
- Our multistage training procedure:
  - train on a 50000 examples subset of the dataset with lr=0.0001 and dice_weight=0.5
  - train on a full dataset with lr=0.0001 and dice_weight=0.5
  - train with smaller lr=0.00001 and dice_weight=0.5
  - increase dice weight to dice_weight=5.0 to make results smoother
- Multi-GPU training
- Use very simple augmentations

Loss function

√ What Worked

- Distance weighted cross entropy explained in the famous U-Net paper.
- Using linear combination of soft dice and distance weighted cross entropy.
- Adding component weighted by building size (smaller buildings has greater weight).



Our resources

1. Source code for our solution:
   https://github.com/neptune-ml/open-solution-mapping-challenge
2. Solution write-up:
   https://github.com/neptune-ml/open-solution-mapping-challenge/blob/master/README.md
3. Experiments and results: https://app.neptune.ml/neptune-ml/Mapping-Challenge

References

1. Original U-Net paper: https://arxiv.org/abs/1505.04597
2. Effective Use of Dilated Convolutions for Segmenting Small Object Instances in Remote Sensing Imagery: https://arxiv.org/abs/1709.00179
3. Image augmentation Python package: https://github.com/aleju/imgaugContextual Hourglass Networks for Segmentation and Density Estimation: https://openreview.net/pdf?id=S1F-dpjjM
4. TernausNetV2: Fully Convolutional Network for Instance Segmentation: https://arxiv.org/abs/1806.00844

Acknowledgements

We would like to thank CrowdAI and “Humanity & Inclusion” for organizing great deep learning challenge and Paulina Prachnio♡ for extensive help regarding this poster.

Bonus

1. If you wish to reproduce entire end-to-end solution, please check these instructions: https://github.com/neptune-ml/open-solution-mapping-challenge/blob/master/REPRODUCE_RESULTS.md
2. If you wish, you can grab additional compute credits for it. Check out this resource for more details: https://github.com/neptune-ml/EuroPython2018

europython Edinburgh 23-29 July 2018

contact info: kamil@neptune.ml

authors affiliations:
1. neptune.ml
2. deepsense.ai BIG DATA SCIENCE