III. Model documentation and write-up

Information included in this section may be shared publicly with challenge results. You can respond to these questions in an e-mail or as an attached file. Please number your responses.

1. Who are you (mini-bio) and what do you do professionally?

At daylight I work as a computer vision research engineer. At night – I solve different machine learning competitions for the sake of fun, learning and challenging myself. I've secured some good progress in competitive ML on Kaggle (Top 100), Signate (Top 1) and elsewhere. Also, I'm the author of Albumentations – an open-source image augmentation library that is de-facto standard library for image augmentations in machine learning competitions and pytorch-toolbelt – a library for quick creation of different neural network model architectures.

2. What motivated you to compete in this challenge?

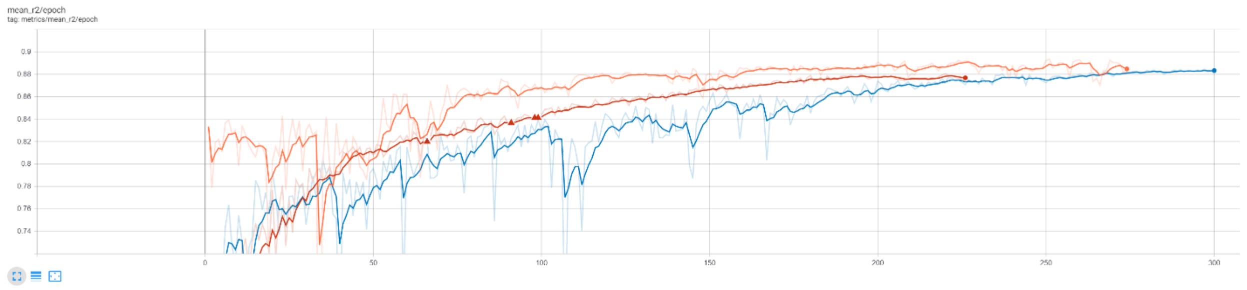
I was curious to test my skills in finding a key to this multi-task learning problem. I had a little experience in solving dense regression problems in satellite imagery and decided to participate to check how learned best-practices from semantic segmentation and object detection would help me in this competition.

3. High level summary of your approach: what did you do and why?

In a nutshell, I've trained a bunch of UNet-like models and averaged their predictions. Sounds simple, yet I used quite heavy encoders (B6 & B7) and custom-made decoders to produce very accurate height map predictions at original resolution. Another crucial part of the solution was extensive data augmentation. I've written custom augmentation operations to augment image in RGB domain, height map, orientation, scale and gsd consistently. I believe this technique improved model generalization capacity a lot.

4. Do you have any useful charts, graphs, or visualizations from the process?

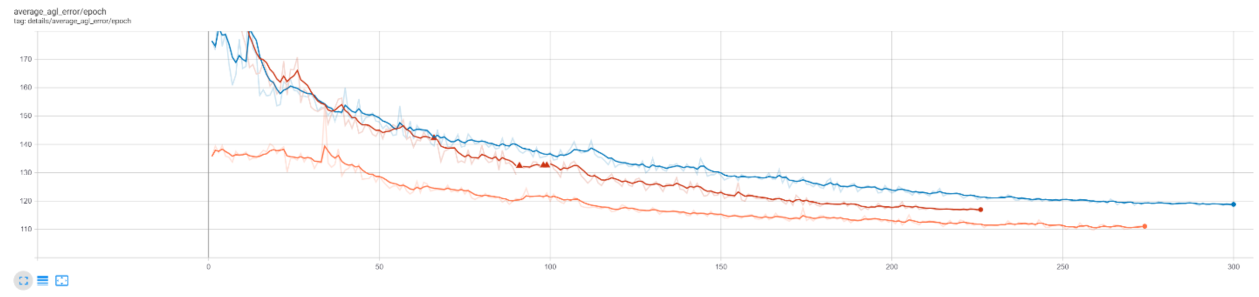
First, I wanted to show mean R2 on the validation for three different folds and models I used in my final solution:



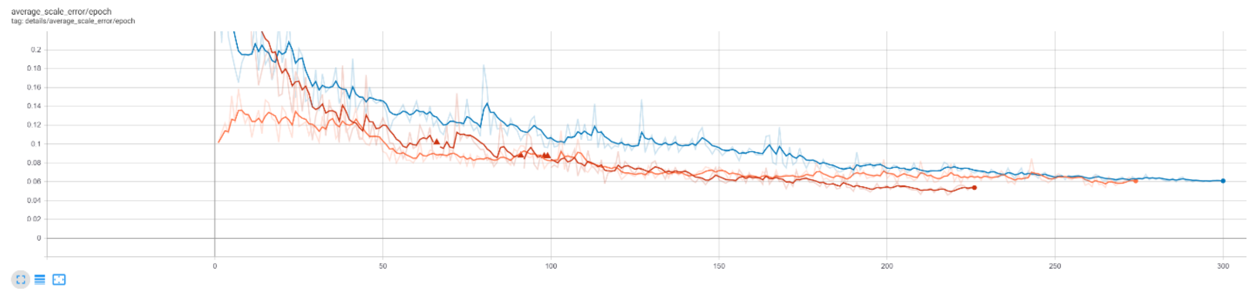
To me, all three modes haven’t reached a plateau and could have been trained even longer.

Loss & average error scores also proves this statement:

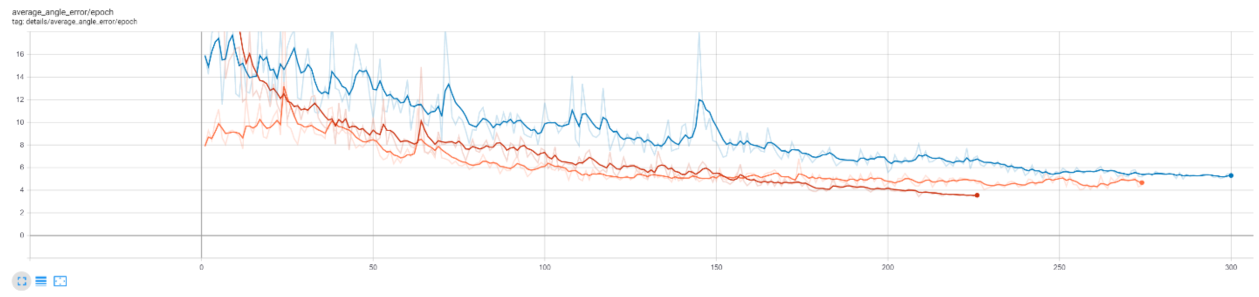
Average AGL prediction error (cm)



Average relative scale prediction error



Average angle error (in degrees)



I think it’s totally possible to reach 0.95 R2 with an additional computational budget with my model architectures.

5. Copy and paste the 3 most impactful parts of your code and explain what each does and how it helped your model.

   First, is the data augmentation technique. Here’s an example of custom-made affine augmentation class (Based on Albumentations library) that augments orientation angle and scale parameters consistently to each other. It helped to increase diversity of image orientation and scale range during the training

```
class GeoShiftScaleRotate(A.ShiftScaleRotate):
    mask_interpolation = cv2.INTER_LINEAR

    @property
    def targets(self):
        return {
            "image": self.apply,
            "mask": self.apply_to_mask,
            "masks": self.apply_to_masks,
            "bboxes": self.apply_to_bboxes,
            "keypoints": self.apply_to_keypoints,
            "gsd": self.apply_to_gsd,
            "vflow_scale": self.apply_to_vflow_scale,
            "vflow_angle": self.apply_to_vflow_angle,
        }

    def apply_to_mask(self, img, angle=0, scale=0, dx=0, dy=0, **params):
        return A.shift_scale_rotate(img, angle, scale, dx, dy,
self.mask_interpolation, self.border_mode, self.mask_value)

    def apply_to_gsd(self, gsd, scale=0, **params):
        """
        Ground sample distance (GSD) in meters per pixel.
        GSD is the average pixel size in meters. Units - meters/pixel
        """

        # When scale is < 1, crop patch is smaller than destination patch, so
number of meters in pixel decreases
        # When scale is > 1, crop patch is larger than destination patch, so
number of meters in pixel increases
        return gsd * scale

    def apply_to_vflow_scale(self, vflow_scale, scale=0, **params):
        return vflow_scale / scale

    def apply_to_vflow_angle(self, vflow_angle, angle=0, **params):
        angle_rads = np.radians(angle)
        vflow_angle = vflow_angle + angle_rads

        if vflow_angle > math.pi * 2:
            vflow_angle -= math.pi * 2
        if vflow_angle < 0:
            vflow_angle += math.pi * 2
        return vflow_angle
```

Second item is proper choice of loss function.

I’ve used Huber loss for height maps prediction (With gamma parameter of 5m for AGL and 10px for Magnitude map) and cosine distance loss for predicting an angle.
I’ve also used combination of cosine distance and mse on orientation predictions. According to my experiments, this combination worked the best in terms of the average angle error.

```yaml
aggregation: sum
losses:
  - loss:
      _target_: geopose.losses.HuberLossWithIgnore
      ignore_value: 65535 # AGL_IGNORE_VALUE
      delta: 5
    prefix: agl_huber
    target_key: TARGET_AGL_MASK
    output_key: OUTPUT_AGL_MASK
    weight: 1.0

  - loss:
      _target_: geopose.losses.HuberLossWithIgnore
      ignore_value: 65535 # AGL_IGNORE_VALUE
      delta: 10
    prefix: mag_huber
    target_key: TARGET_MAGNITUDE_MASK
    output_key: OUTPUT_MAGNITUDE_MASK
    weight: 1.0

  - loss:
      _target_: geopose.losses.CosineSimilarityLoss
    prefix: angle_cos
    target_key: TARGET_VFLOW_DIRECTION
    output_key: OUTPUT_VFLOW_DIRECTION
    weight: 1.0


  - loss:
      _target_: torch.nn.MSELoss
    prefix: angle_mse
    target_key: TARGET_VFLOW_DIRECTION
    output_key: OUTPUT_VFLOW_DIRECTION
    weight: 1.0
```

A third item is D4 TTA and deep ensemble for averaging predictions. Models alone were quite accurate. But TTA gave them extra kick for some extra inference time. It became possible, since during the training I've also applied D4 augmentations to the training sample.

I've implemented D4 augmentations entirely on GPU using parts of the code from pytorch-toolbelt, with proper changes to deaugment height predictions, orientation and scale. According to my measurements, D4 TTA gave boost of ~0.005~0.01 R2 depending on the model.

6. Please provide the machine specs and time you used to run your model.
   - CPU (model): AMD Threadripper 1950x
   - GPU (model or N/A): 2x3090
   - Memory (GB): 96Gb
   - OS: Ubuntu 20.04
   - Train duration: ~48h for each model
   - Inference duration: ~7h

It is worth noting that final solution ensemble (`fp32_0.8914_b7_fold0_b7_rdtsc_fold1_b6_rdtsc_fold9_bo3_d4.yaml`) contains models trained on 3 out of 10 folds and from each fold I top 3 checkpoint based on validation R2 score. In total there are 12 models in the ensemble, whose predictions are averaged to generate a final prediction. Such a large ensemble with D4 TTA requires a significant amount of time for inference.

By using only one checkpoint per fold, one can increase inference time by a factor of three with marginal decrease in the target metric (0.8914 -> 0.8910 R2 score on private LB). Switching from D4 TTA to D2 or completely disabling it would also improve the runtime performance by a factor of 2 and 8 accordingly.

7. Anything we should watch out for or be aware of in using your model (e.g. code quirks, memory requirements, numerical stability issues, etc.)?

   In order to generate predictions for the ensemble of two models it should be enough 11Gb RAM on the GPU. Two other ensembles (See README) contain more models and can't fit into 11Gb. A 16Gb card is a recommended minimum for them.

   During the training, I've also observed some instability when using PyTorch's mixed precision training mode. At random times during the training, loss became NaN. It still not clear to me what exactly caused this issue. After struggling few days with this issue, I switched to training my models in FP32 mode, which seemed to solve the issue.

8. Did you use any tools for data preparation or exploratory data analysis that aren't listed in your code submission?

   I didn't. EDA in the introductory blogpost gave pretty much all the information for quick start. During training I looked at the intermediate visualizations to see how model performs, analyzed most common mistakes and addressed them via adding more augmentations and changing to more noise-robust loss function.

9. How did you evaluate performance of the model other than the provided metric, if at all?

I've implemented R2 metric and was able to track its progress during the training. It was the main metric I used to save N best checkpoints to use at the inference stage. In addition to that, I tracked (for every location and globally averaged):

- average orientation error (in degrees)
- average scale error
- average height error (mse)

These additional metrics provided additional insights on model performance in each geographical location and helped to tune weights of individual components of loss function (agl, scale, orientation)

10. What are some other things you tried that didn't necessarily make it into the final workflow (quick overview)?

I've experimented with different loss functions and their combinations: Wing Loss, pure L1 loss, Log-Cosh loss, but their performance was either sub-optimal or lead to numerical instabilities during the training.

Tried training on half-resolution data, but score was significantly slower (0.88). Training & inference time was much more appealing though.

I did experiments with the orientation prediction head to exclude areas of the input image, with small above the ground values (<1m). The idea was that such areas contain no useful signal to infer orientation from. Usually, we want to look at the places with strong and smooth gradients of the AGL map (Edges of the buildings, trees, fences). Given a short amount of time I had, I ditched this idea, yet I see real value in it.

11. If you were to continue working on this problem for the next year, what methods or techniques might you try in order to build on your work so far? Are there other fields or features you felt would have been very helpful to have?

Next time I will start a bit earlier. Two weeks was indeed a "last call" to hop on. The challenge was indeed complicated one. Also, there were many strong participants on the leaderboard. A few ideas left intact this time:

- Transformers-based model. It has been shown recently, that ViT, SWIN and other approaches can beat classical CNN models in segmentation tasks. Therefore, it would be interesting to have a bite.
- Trimmed losses. Monocular depth estimation benefits greatly on using trimmed losses to exclude influence of strong outliers during the training. I wonder whether this can be applied to geopose estimation as well.
- HR-Net, FPN architectures, NFNets
- GAN-based data augmentation. I feel this is largely underestimated technique, which requires significant effort to master, but can lead to substantial boost of the model generalization.
- Direct optimization of R2 score. Could be interesting to look how better/worse it will be compared to MSE/Huber losses.