

7th place solution write-up

1. Provide a brief abstract of your final approach (one paragraph or less), highlighting the most important or interesting ideas and impactful decisions made.

I developed a solution that reached 7th place in the private leaderboard with a score of 0.8518 which is 0.05 improvement over the baseline solution [1]. First, I implemented training with automatic mixed precision in order to speed up training and facilitate experiments with the large architectures. Second, I implemented 7 popular decoder architectures and conducted extensive preliminary research of different combinations of encoders and decoders. For the most promising combinations I ran long training for at least 200 epochs to study best possible scores and training dynamics. Results of the long runs are provided in the table below (even for the architectures which were not included in the final solution). Third, I implemented an ensemble using weighted average for *height* and *scale* target and circular average for *angle* target. These approaches helped to substantially improve baseline solution. I believe that my solution and research results will be useful for further development of this task.


2. Explain your technical approach in detail. Discuss what you did and why. Discuss what worked well and not so well in your final solution. It is not necessary to provide background material on the challenge formulation unless it helps explain your solution.

General approach

My final solution is an ensemble of predictions from 5 architectures each of which is an average of predictions from 5 checkpoints trained on 5-fold splits of training data stratified by the city (25 checkpoints total). I also used simple post-processing described below. Architectures are based on U-Net [2] decoder and the following encoders: ResNet34, Inception v4, SE-ResNeXt50, RegNetX064, and ResNeXt50. First 4 were initialized with Imagenet pretrained weights while the last one uses SWSL [3] weights. The best single architecture is U-Net/ResNeXt50 (SWSL weights). I extensively researched other decoders but did not see an improvement. All models were trained using all 4 cities jointly. Augmentation was used for all models with the same parameters as in the baseline model. Adam optimizer was used for all models.

From the beginning my interests were concentrated in the field of architecture and ensembling. I implemented 7 popular decoder architectures and ran short preliminary experiments to identify the most promising encoder-decoder combinations. Preliminary experiments were trained on full data for about 20 epochs while recording validation scores after each epoch. By comparing validation score dynamics I chose the most promising models and ran long training. Details about all models which were trained long enough are represented in Table 1. Unfortunately none of the new decoders improved the result of the U-Net decoder. Also they did not contribute to the final ensemble. But the scores were relatively high and close to the baseline.

Improving training time



In order to speed up the training process I implemented automatic mixed precision training using standard tools from Pytorch library (*torch.cuda.amp*). As expected this leads to about 2x speed up and 2x increase in GPU memory capacity facilitating usage of larger encoders and decoders.

Resolution

I run several experiments using full image resolution (2048 x 2048). I did not obtain any improvements. Large image size makes the training process less stable, requiring adjustment of hyperparameters accordingly.

Hyperparameters

In order to explore the effect of large batch size I implemented gradient accumulation. This approach allows usage of batches of arbitrary effective size. I was looking in this direction because in some tasks large batches can improve generalization. I experimented with hyperparameters from a wide range of values. Learning rate: from 1e-5 to 1e-3; batch size: from 1 to 128. Eventually large batch sizes did not bring any substantial improvements. Learning rates higher than 1e-4 often lead to unstable training.

Ensembling

For ensembling I used weighted average for *height* and *scale* targets, and circular mean (*scipy.stats.circmean*) for *angle* target. Ensembling was performed on raw predictions (fp32). Then casting and compression were applied. Examples of gradual improvements are represented below (all scores are from the private leaderboard without post-processing):

- 0.8079 - U-Net/ResNet34 baseline, single fold, 500 epochs
- 0.8294 - U-Net/ResNet34 baseline, mean of 5 folds, 500 epochs each fold
- 0.8336 – Ensemble of 2 architectures (5 folds each):
 - U-Net/ResNet34 baseline, mean of 5 folds, 500 epochs each fold;
 - U-Net/Inception-v4, mean of 5 folds, 290 epochs each fold
- 0.8367 – Ensemble of 3 architectures (5 folds each):
 - U-Net/ResNet34 baseline, mean of 5 folds, 500 epochs each fold;
 - U-Net/Inception-v4, mean of 5 folds, 290 epochs each fold;
 - U-Net/SE-ResNeXt50, mean of 5 folds, 120 epochs each fold

Ensembling experiments led to interesting conclusions. First, we can see that the largest improvement comes from ensembling the folds of the same architecture. Contributions from the architectures with different encoders are much smaller. Second, we may expect good contributions from the architectures with different decoders instead of U-Net (due to seemingly less correlated predictions). But this is not the case. E.g. contribution to the ensemble from FPN/ResNeXt50 was negligible (0.0005). Possibly this is related to the fact that the score of this architecture itself was relatively low: 0.8090.

Post-processing

I found an interesting bias and corresponding post-processing via multiplication of *height* by the following city-specific coefficients: ARG x 1.20, JAX x 1.05, OMA x 1.10. These coefficients worked equally well for public and private subsets of the leaderboard test set; and also for any single model and for my final ensemble of 5 models leading to about 0.01 improvement. Given that all coefficients are greater than 1.0 we can tell that models consistently predict slightly lower *height* values. Possibly this effect is caused by the model's reaction to ARG examples seen during training which are substantially different from another 3 cities. Also it may be related to the slightly different distributions of the cities or viewpoints in the training set and test set. Given that convergence takes a long time I did not manage to accommodate this post-processing in the model itself, but I suspect that playing with training distribution or applying appropriate finetuning may result in similar improvement as described post-processing.

Meta features

Another marginal improvement (0.001) came from the separate XGBoost model trained to predict the *scale* target based on *gsd* and *city* features from metadata. Actually my ensemble already gave accurate predictions for the *scale* target but I was curious to explore effects of metadata.

It's also worth mentioning that I tried to train a similar separate XGBoost model for angle components (*xdir*, *ydir*) based on *gsd* and *city* features. This model did not outperform *angle* prediction from the baseline model, but still gave relatively good *angle* accuracy. These facts lead us to the conclusion that *gsd* and *city* are good predictors. And while the *city* may not be a good choice because we may want to generalize to the unseen cities, *gsd* may be useful.

I also made an attempt to use meta features for *angle* target training directly in *EncoderRegressionHead* (*xydir_head*). I had not enough time to run long training but no substantial improvement expected according to initial epochs. So this approach was not used in the final solution.

Experiment results

Table 1. Architectures and corresponding results. In the column "Time per epoch" *fp32* means full precision, and *amp* means automatic mixed precision. The best score for each architecture is bold. The best overall score is bold underlined. Architectures with asterisk are included in the final solution.

#	Decoder	Encoder	Hyperparameters	Time per epoch (T4 GPU, minutes)	Epoch	Private LB (mean of 5 folds)
1*	Unet	ResNet34 (resnet34)	Batch: 6 LR: 1e-4	16.5 (fp32)	200	0.8224
					300	0.8260
					400	0.8286

					500	0.8294
2	Unet	EfficientNet-B0 (efficientnet-b0)	Batch: 4 LR: 1e-4	19 (fp32)	200	0.7929
					400	0.7938
3*	Unet	Inception-v4 (inceptionv4)	Batch: 6 LR: 1e-4	20 (amp)	90	0.8154
					170	0.8200
					290	0.8247
					400	0.8195
4*	Unet	SE-ResNeXt50 (se_resnext50_32x4d)	Batch: 5 LR: 1e-4	20 (amp)	120	0.8263
					180	0.8213
					240	0.8221
5*	Unet	RegNetX064 (timm-regnetx_064)	Batch: 5 LR: 1e-4	29 (amp)	80	0.8234
					150	0.8202
6*	Unet	ResNeXt50 (SWSL weights) (resnext50_32x4d)	Batch: 6 LR: 1e-4	18 (amp)	100	0.8273
					170	0.8306
					230	0.8333
7	DeepLabV3 [4]	ResNet34 (resnet34)	Batch: 6 LR: 1e-4	25 (amp)	140	0.8010
					190	0.8067
					240	0.8099
					350	0.8139
8	FPN [5]	ResNeXt50 (resnext50_32x4d)	Batch: 6 LR: 1e-4	14 (amp)	140	0.8090
					330	0.8089

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

1) Automatic mixed precision speeds up the training and increases GPU capacity

```

class TrainEpoch(Epoch):
    # ...
    def batch_update(self, x, y):
        self.optimizer.zero_grad()
        with torch.cuda.amp.autocast(enabled=amp_enabled):
            xydir_pred, agl_pred, mag_pred, scale_pred = self.model.forward(x.float())
        # ...
        self.scaler.scale(loss_combined).backward()
        self.scaler.step(self.optimizer)
        self.scaler.update()

```

2) Gradient accumulation allows to use batches of arbitrary effective size

```

class Epoch:
    # ...
    def run(self, dataloader):
        # ...
        for batch_id, itr_data in enumerate(iterator):
            # ...
            if self.stage_name != 'valid':
                # ...
                do_step = ((batch_id+1) % self.accumulation_steps == 0)
                results = self.batch_update(image, y, do_step=do_step)

class TrainEpoch(Epoch):
    # ...
    def batch_update(self, x, y, do_step):
        # ...
        if do_step:
            self.optimizer.step()

```

3) Attempt to use meta features (*gsd* and *city*) directly in *angle* head (*xydir_head*) (was not used in the final solution):

```
class EncoderRegressionHead(nn.Module):

    def __init__(self):
        super().__init__()
        self.pool = nn.AdaptiveAvgPool2d(1)
        self.flatten = Flatten()
        self.dropout = nn.Dropout(p=0.5, inplace=True)
        self.cat = torch.cat
        # 514 instead of 512
        self.linear = nn.Linear(514, 2, bias=True)

    def forward(self, encoder_features, meta):
        x = self.pool(encoder_features)
        x = self.flatten(x)
        x = self.dropout(x)
        # concat 'gsd' and 'city' meta features
        x = self.cat((x, meta), 1)
        x = self.linear(x)
        return x
```

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

I used only provided metric and its components for different targets i.e. R^2 and RMSE.

5. Please provide the machine specs and time you used to run your model.

- CPU (model): virtual cloud CPU (4 cores, 2.3 GHz)
- GPU (model or N/A): nVidia T4 GPU, 16 GB
- Memory (GB): 28 GB RAM
- OS: Ubuntu 18.04
- Train duration: 1900 hours (for the whole ensemble)
- Inference duration: 6 hours (for the whole ensemble)

6. What are some other things you tried that didn't necessarily make it into the final solution?

After training one of my models for 400 epochs on full data I tried to finetune it on the city-specific subsets of the data with and without augmentation. None of these experiments leads to substantial improvement.

Also I implemented TTA (test-time augmentation) as 3 rotations multiple of 90 degrees. The score from the average of all TTA predictions were slightly lower than without TTA.

7. 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? Is there other data or metadata that you felt would have been very helpful to have? Would you frame the problem differently?

I would focus my further efforts on researching possibilities for loss function, influence of shadows, and augmentation. I think augmentation has a great potential to further increase generalisation and mitigate possible biases.

8. Provide any references cited in your discussion above.

[1] G. Christie, K. Foster, S. Hagstrom, G. D. Hager, and M. Z. Brown, "Single View Geocentric Pose in the Wild," in CVPRW, 2021. [<https://arxiv.org/abs/2105.08229>]

[2] Olaf Ronneberger, Philipp Fischer, Thomas Brox "U-Net: Convolutional Networks for Biomedical Image Segmentation" [<https://arxiv.org/abs/1505.04597>]

[3] I. Zeki Yalniz, Hervé Jégou, Kan Chen, Manohar Paluri, Dhruv Mahajan "Billion-scale semi-supervised learning for image classification" [<https://arxiv.org/abs/1905.00546>]

[4] Liang-Chieh Chen, George Papandreou, Florian Schroff, Hartwig Adam "Rethinking Atrous Convolution for Semantic Image Segmentation" [<https://arxiv.org/abs/1706.05587>]

[5] Alexander Kirillov, Ross Girshick, Kaiming He, Piotr Dollár "Panoptic Feature Pyramid Networks" [<https://arxiv.org/abs/1901.02446>]