User: kbrodt

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?

Currently I'm doing research at the University of Montréal in Computer Graphics, using Machine Learning for problem solving: posing 3D characters via gesture drawings. I got my Master in Mathematics at the Novosibirsk State University, Russia. Lately, I fell in love with machine learning, so I was enrolled in Yandex School of Data Analysis and Computer Science Center. This led me to develop and teach the first open Deep Learning course in russian. Machine Learning is my passion and I often take part in competitions.

2. What motivated you to compete in this challenge?

I like competitions and to solve new problems.

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

The method is based on the solution provided by the authors, but in a simpler and straightforward way: using Unet architecture with various encoders (efficientnet-b{6,7} and senet154). The model has only one above ground level (AGL) head and two heads in the bottleneck for scale and angle. The model takes a random 512x512 crop of an aerial image, the city's one hot encoding and ground sample distance (GSD) as input. Then the model outputs the AGL, vector flow scale and angle. The model is trained with mean squared error (MSE) loss function for all targets (AGL, scale, angle) using AdamW optimizer with 1e-4 learning rate. Cosine annealing scheduler with period 25 is used. To reduce the memory consumption and to speedup the training process the model is trained in mixed precision regime with batch size 8. At inference time the model takes a full size 2048x2048 aerial image and outputs a full size AGL.

First, the model is pretrained with heavy augmentations (like flips, rotations, color jittering, scaling, height augmentations etc.) for 525 epochs and then finetuned another 1025 epochs without any augmentations.

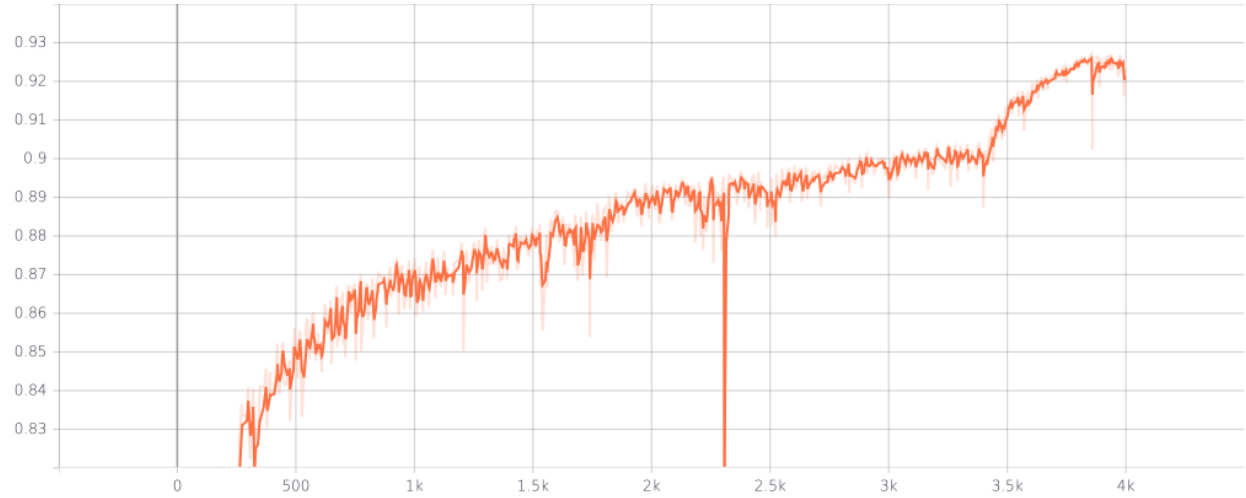
Remark: It turns out that augmentations are damaging for model performance. Model is trained faster without any augmentations and has better performance according to validation.

Note: A single model projected to be 4th place with 0.86 R2 coefficient of determination.

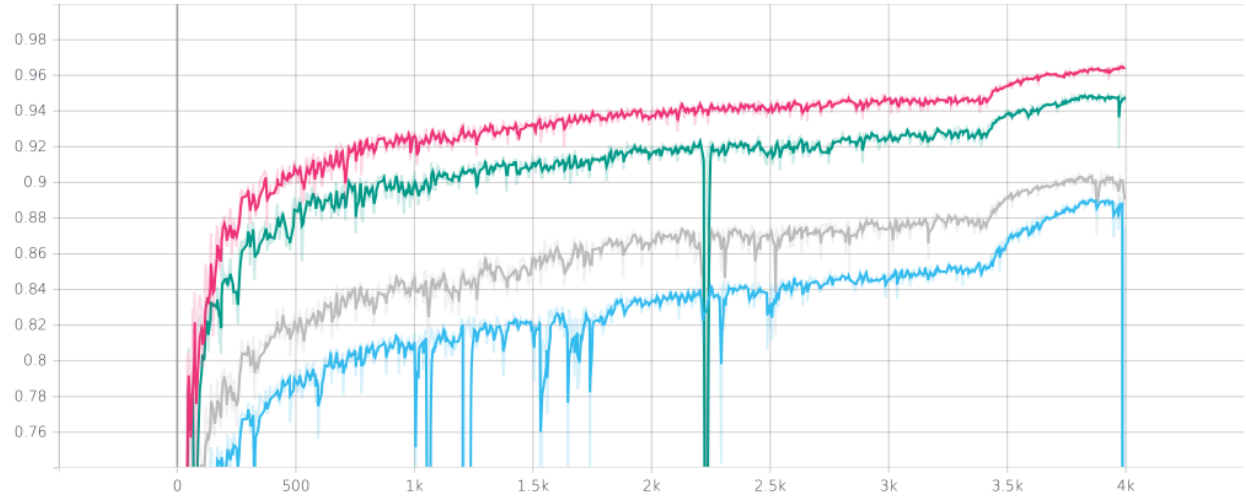
4. Do you have any useful charts, graphs, or visualizations from the process?

Typical learning curves.

Competition metric



Metrics per city (pink ATL, green JAX, grey OMA and cyan ARG)



We can see that metrics are improving dramatically after turning off the augmentations.

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

- `ml_utils.py` line 163-166

```
if (not self.is_test) and (not self.is_val):
            data = self.crop_fn(image=image, mask=agl)
            image = data["image"]
            agl = data["mask"]
```

Crops images to speed up augmentations

- `ml_utils.py` line 153-157

```
with self.env.begin(write=False, buffers=True) as txn:
                image = pickle.loads(txn.get(rgb_path.stem.encode()))
                agl = pickle.loads(
                    txn.get(rgb_path.stem.replace("RGB", "AGL").encode())
                )
```

Read images from database to speedup training process

- `ml_utils.py` line 992-998, line 1011

```
def worker_init_fn(worker_id):
        seed = (
            np.random.get_state()[1][0]
            + torch.distributed.get_world_size() * worker_id
            + args.local_rank
        )
        np.random.seed(seed)
...
Dataloader(
    …
    worker_init_fn=worker_init_fn,
    …
)
```

Fixes bug in pytorch 1.8 with numpy seeding. (https://github.com/pytorch/pytorch/pull/56488). Otherwise there is no augmentation like in the author's code. So they didn't notice the fact that augmentations harm model performance.

6. Please provide the machine specs and time you used to run your model.
   - CPU (model): Intel(R) Xeon(R) Gold 6148 CPU @ 2.40GHz
   - GPU (model or N/A): Nvidia Tesla V100 32GB
   - Memory (GB): 755Gb
   - OS: CentOS 7
   - Train duration: 1 week
   - Inference duration: 20 minutes

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.)?
   To preprocess data and save images into `lmdb` database you need 200Gb of RAM. To train all models you need about 100Gb RAM, 4 GPUs with 32Gb VRAM and about 1 week.

8. Did you use any tools for data preparation or exploratory data analysis that aren't listed in your code submission?
   `lmdb` database for images to speedup reading from disk

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

10. What are some other things you tried that didn't necessarily make it into the final workflow (quick overview)?
    Things didn't work:
    - Train the model with MSE loss on logarithmic AGL
    - Finetune per city
    - Heavy augmentations

- Predict VFLOW and find xy axes via pseudo-inverse (like for scale)
- TGV denoising

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?
    - Use extra target of normal fields obtained from AGL
    - Explore more about TGV denoising in ensembling