Report

Instant-NGP

I decided to use the INGP model as it is one of the latest developments and introduces good possibilities for SDF exploration. INGP is a very accurate model, which is also faster than other proposed methods. And also INGP is an improved version of nglod (we have better stats for memory consumption here as we don't save all of the feature vectors in memory). So obviously, it works faster than nglod. (And also this model was considered state of the art when it first made its appearance, and it is still widely used for the variety of tasks)

At first I tried implementing the model from original repository with ingp from nvlabs (https://github.com/NVlabs/instant-ngp), but as the setup there was very difficult, I switched to using solution which implements SDF more specifically and is also easier to setup (https://github.com/ashawkey/torch-ngp/tree/main).

There were some problems with gcc and g++ versions, even with installation of the latest ones, because in some places c++14 was used and it wasn't compatible with the existing version of torch, so it had to be changed for c++17.

As for the training of the model, I mainly used standard parameters that were predefined in the init state of the chosen SDF model. There was no hyperparameter tuning in general, the only things that I changed were the number of epochs and for the last two objects I changed the learning rate scheduler to a scheduler that reduces learning rate on plateau.(In the torch-ngp/sdf/utils.py module) It helped a little bit, but not much.

For the first 4 meshes I trained the model for 20 epochs. The last one was trained for 25 epochs.

Also, for the last two meshes I used amp mixed precision training just to try it out and see whether it will give better results (make runtime faster)

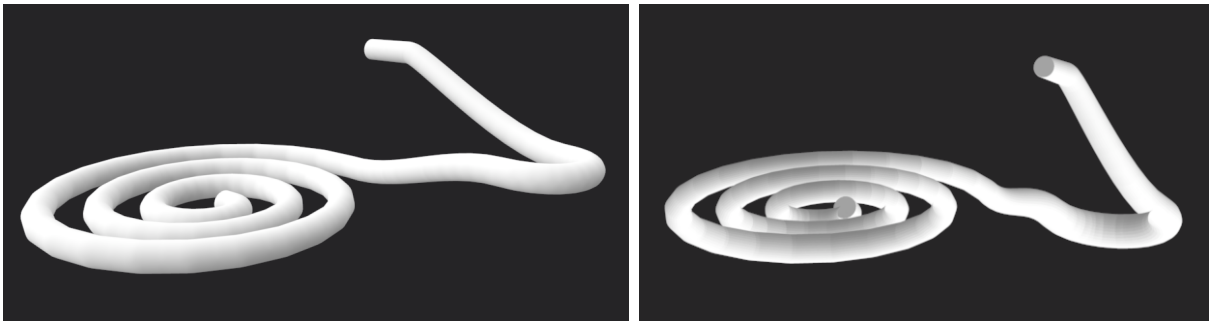
General pipeline of training was following:

1. Repo was cloned to local folder (to be more specific: I used google colab)
2. I needed to install gcc and g++ for compiling
3. Setup for the repo code (building extensions, etc)
4. Training with, by calling the mesh_sdf module with needed arguments (my mesh, that I plan to train the model on)
5. Testing (testing the model on the mesh, that I have previously have trained it on)

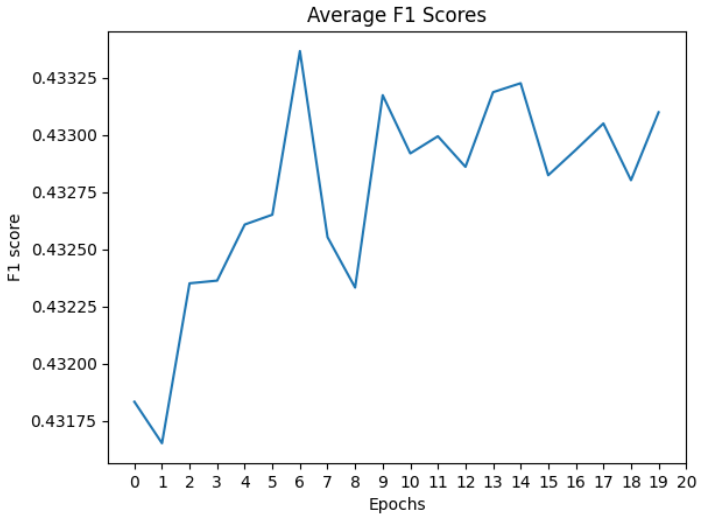
Results of the INGP:

(In the submitted files there are also trimesh points visualizations on the plots)

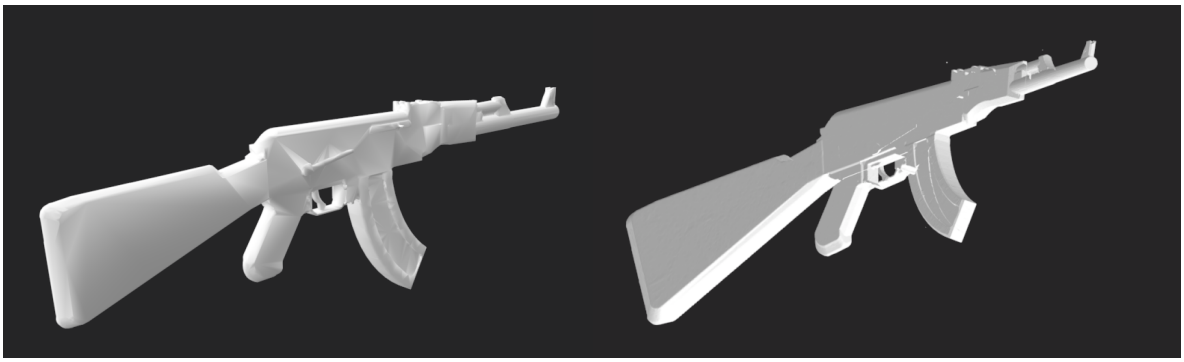
Object2



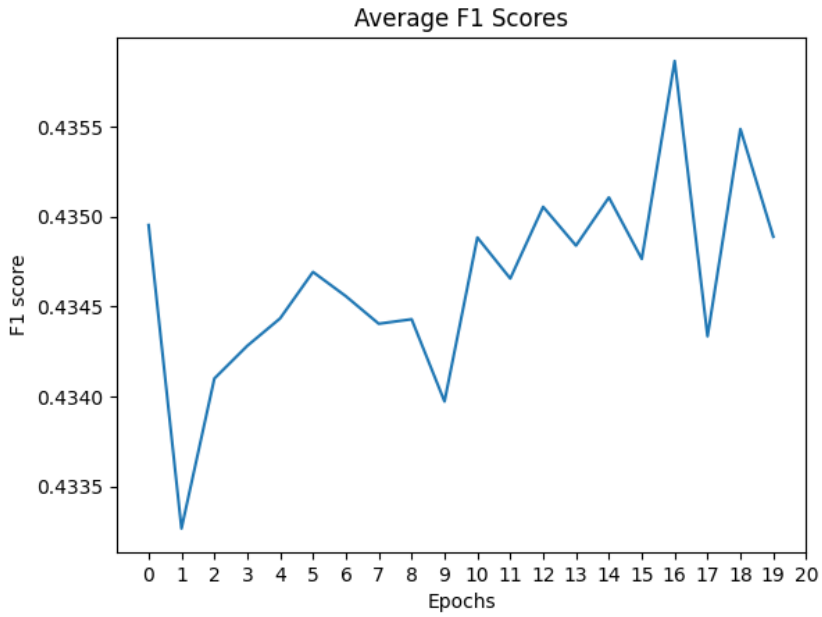
(Left - original, right - result of ingp SDF network)



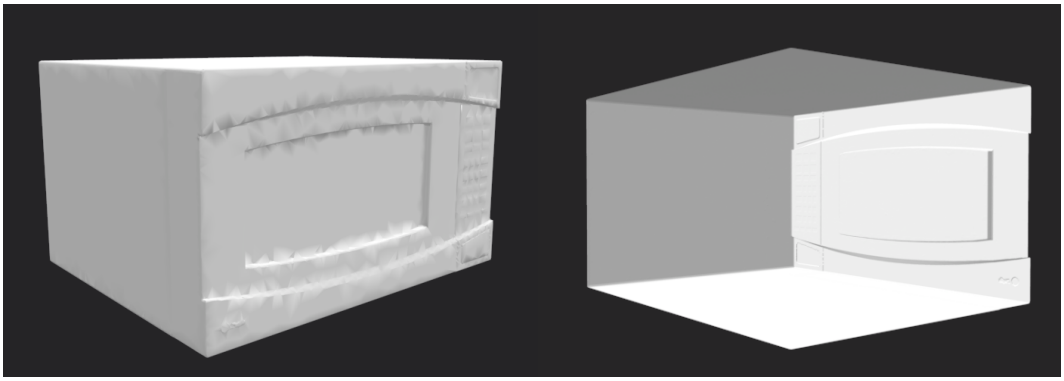
Object31



(Left - original, right - result of ingp SDF network)

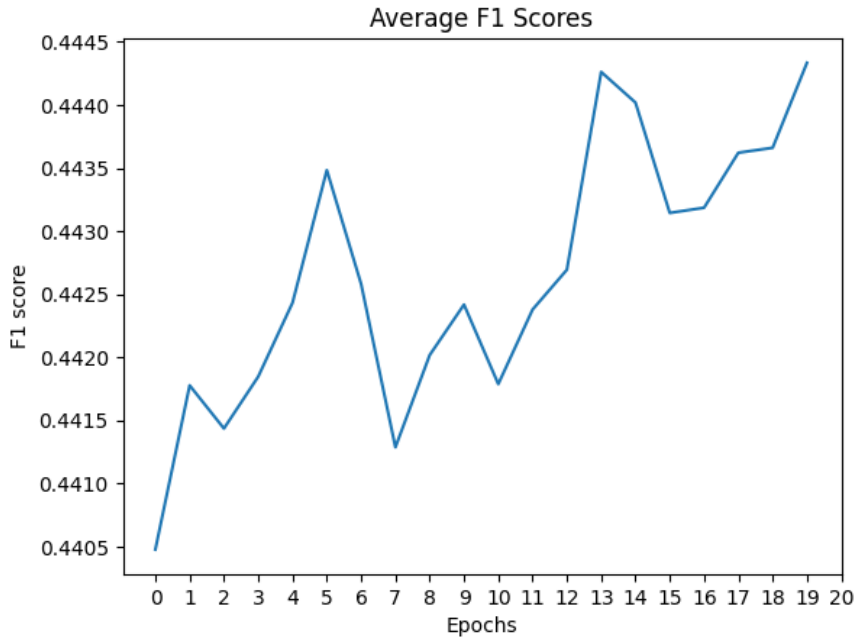


Object44

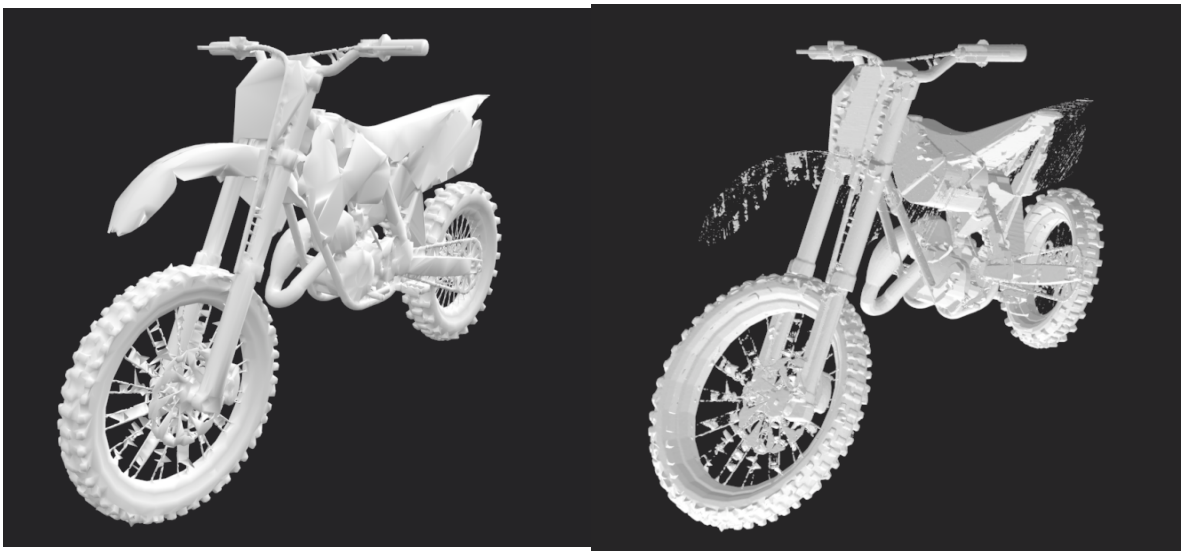


(Left - original, right - result of ingp SDF network)

(Screenshot on the right is taken from the different side of the object, as on my machine the walls are looking half-transparent on the render)



Object 0



(Left - original, right - result of ingp SDF network)

To calculate the F1-score during training I modified the training loop, to calculate the F1-score on each step of each epoch and store them so these scores can later be used in the analysis and visualizations.

```python
def train_step(self, data):
    # assert batch_size == 1
    X = data["points"][0] # [B, 3]
    y = data["sdfs"][0] # [B]

    pred = self.model(X)
    loss = self.criterion(pred, y)

    # y_sign = torch.sign(y)
    # pred_sign = torch.sign(pred)

    y_sign = np.sign(y.cpu().detach().numpy())
    pred_sign = np.sign(pred.cpu().detach().numpy())

    f1 = f1_score(y_sign, pred_sign, average='macro')

    return pred, y, loss, f1
```

```
++> Average F1 score: 0.44433359019899993.
++> Average f1 for all passed epochs: [0.4404756857206156, 0.4417765058144088, 0.4414348888667045,
0.44184646223134266, 0.4424318852076423, 0.4434830483531697, 0.44258073281201854,
0.44128635997076165, 0.4420172246607164, 0.4424172200275023, 0.4417873061444207, 0.4423779177100067,
0.4426933473706863, 0.44426157483544476, 0.4440187891002654, 0.4431439678538431, 0.4431850149867696,
0.44362112536770465, 0.4436600992209012, 0.44433359019899993]
++> Evaluate epoch 20 Finished.
==> Saving mesh to obj_44/validation/ngp_20.ply
==> Finished saving mesh.
[INFO] New best result: 0.013442119213018185 --> 0.013159582932965916
==> Saving mesh to obj_44/results/output.ply
==> Finished saving mesh.
[INFO] Trainer: ngp | 2024-01-05_18-56-09 | cuda:0 | fp16 | obj_44
[INFO] #parameters: 12245936
[INFO] Loading best checkpoint ...
[INFO] loaded model.
==> Saving mesh to obj_44/results/output.ply
[INFO] Trainer: ngp | 2024-01-05_18-56-48 | cuda:0 | fp16 | obj_44
[INFO] #parameters: 12245936
[INFO] Loading best checkpoint ...
[INFO] loaded model.
==> Saving mesh to obj_44/results/output.ply
==> Finished saving mesh.
```

Model weights less than 1 mb.

Ng-lod

This was my other choice, because overall this model is also good (and it was considered a step up from its predecessors - it is not a large network and we are using a grid here)

I tried implementing this method from https://github.com/nv-tlabs/nglod, but after I finished the excruciating setup process, I discovered that there are no built-in options to test the model, so I decided to leave it for now.

Possible improvements

I suspect that there is maybe something wrong with calculation of F-score, because in the logs of the models (they can be viewed in the notebooks or in separate files in the corresponding folders) we can see that loss drops and we don't have an overfit situation happening.

Maybe increasing the number of epochs can also help.