Homework 4: Surface Reconstruction with MLP

姓名: 方嘉聪 学号: 2200017849

1 整体介绍与结果

1.1 实现效果

参考了 IGR [1] 和 [2], 实现了在所给数据集上的 MLP 和 MLP with Fourier features 的表面重建, 结果见 **Figure** 2:

- 基于简单 MLP 的方法能够重建出与 ground truth 相似的表面形状, 但是在高频细节上存在明显的模糊和缺失, 例如第一行的飞机上下两层的连杆等等, 细节效果比较一般.
- 基于 MLP with Fourier features 的方法能够重建出更为精细的表面形状, 包含更多的高频细节, 如飞机的螺旋桨、导弹等等都得到了较好的重建, 整体效果较接近 ground truth. 较基于简单 MLP 的方法, 该方法在高频细节上有明显的提升.

1.2 项目结构

项目结构如下:

- `REAMDE.md`: 项目说明文件, 包含了项目依赖、运行方法等信息.
- `results/`: 重建结果, 在 `./results/result_naive` 和 `./results/result_fourier` 中分别存储简单 MLP 和 MLP with Fourier features 的重建结果. 以`.obj` 格式存储
- `doc/`: 报告 LaTeX 源文件及重建结果 MeshLab 可视化截图 (`./doc/imgs/`)
- `./`: 源代码, 包括训练和测试 (`train.py` `inference.py` 及相应 sh 脚本), 数据加载 (`dataset.py`), 模型定义 (`model.py`) 等. 两种方法的训练和测试代码集成在同一文件中, 通过命令行参数选择.

2 方法介绍

2.1 网络架构与参数量

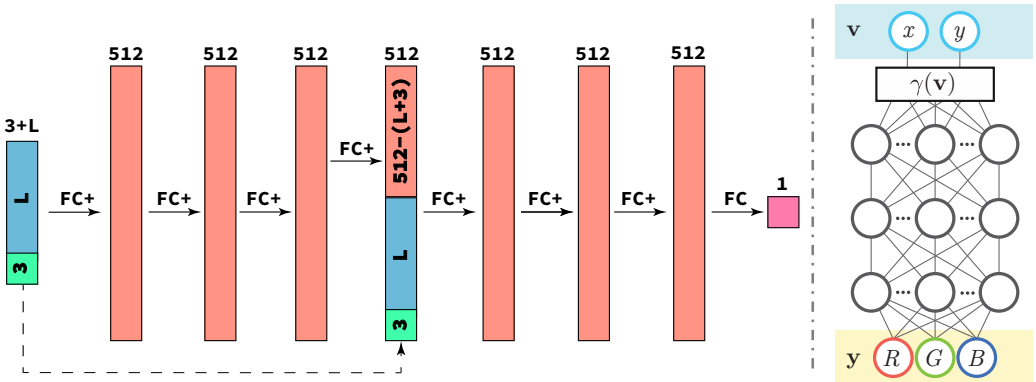

Figure 1: (左) MLP 架构 [1], (右) MLP with Fourier features 架构 [2]。实际使用的 MLP 架构基于左图进行扩大, 为 10 层, 每层 768 个神经元, 在第 5 层进行跳跃连接。MLP with Fourier features 在输入层前增加了对输入坐标的傅里叶特征变换 ($3 \to 128$).

最后编译时间: 2025 年 6 月 1 日

如 **Figure** 1 所示, 实际使用了 10 层的 MLP, 每层包含 768 个神经元, 在第 5 层进行了跳跃连接, 架构参考了 IGR [1] 使用 geometry initialization, 细节见 `model.py`. MLP with Fourier features 的架构使用相同的 MLP, 但是在输入层前增加了对输入坐标的傅里叶特征变换, 将输出的 3 维坐标 (x, y, z) 转换为 128 维的傅里叶特征, 参考了 [2] 实现, 代码见下:

```python
class GaussianFourierFeatureTransform(nn.Module):
    def __init__(self, num_input_channels=3, mapping_size=64, scale=5):
        super().__init__()

        self._num_input_channels = num_input_channels
        self._mapping_size = mapping_size
        self._B = torch.randn((num_input_channels, mapping_size)) * scale
        self.output_dim = 2 * mapping_size

    def forward(self, x):
        """
        x: Point cloud coordinates of shape [B, 3]
        returns: Fourier features of shape [B, 2*mapping_size]
        """
        _, channels = x.shape
        x = x @ self._B.to(x.device)
        x = 2 * np.pi * x
        return torch.cat([torch.sin(x), torch.cos(x)], dim=1)
```

MLP 和 MLP with Fourier features 的参数量/模型文件大小均为 21.3 MB, 实际参数量二者差别不大.

2.2 训练策略与细节

损失函数. 参考作业文档, 记 $\tilde{F}(q) := \mathrm{MLP}(q)$ 为 MLP 对于输入点 q 的预测 SDF, $F(q)$ 为 ground truth SDF, $\nabla_q \tilde{F}(q)$ 和 $\nabla_q F(q)$ 分别为 MLP 和 ground truth 对于输入点 q 的梯度. 损失函数定义为:

$$\mathcal{L} = \lambda_{\text{sdf}} \cdot \mathcal{L}_{\text{sdf}} + \lambda_{\text{grad}} \cdot \mathcal{L}_{\text{grad}} \tag{1}$$

其中

$$\mathcal{L}_{\text{sdf}} = \frac{1}{N} \sum_{i=1}^{N} \left\| \tilde{F}(q_i) - F(q_i) \right\|_2^2, \quad \mathcal{L}_{\text{grad}} = \frac{1}{N} \sum_{i=1}^{N} \left\| \nabla_q \tilde{F}(q_i) - \nabla_q F(q_i) \right\|_2^2. \tag{2}$$

注: 参考 IGR [1] 还尝试了 eikonal norm loss, 在经过测试效果不好, 这里不赘述.

数据集. 对于 MLP 由于输入的维度只有 3 维, MLP 具有连续的特点, 我在实验中直接使用了点云数据进行训练, 测试效果可以恢复出较好的表面形状, 且在空间中未出现明显的 artifacts. 对于 MLP with Fourier features, 由于输入的维度为 128 维, 我使用了点云数据混合采样点, 单次采样的比例为点云 : 采样点 = 2 : 3.

详细配置. 学习率、优化器、训练轮数、超参数的设置等详细设置见 **Table** 1. 两个方法训练过程中均使用了 Adam 优化器, 学习率 scheduler 均使用 `CosineAnnealingLR(eta_min=1e-6)`. 对于 MLP with Fourier features, `fourier_mapping_size = 64`, `fourier_scale = 5.0`.

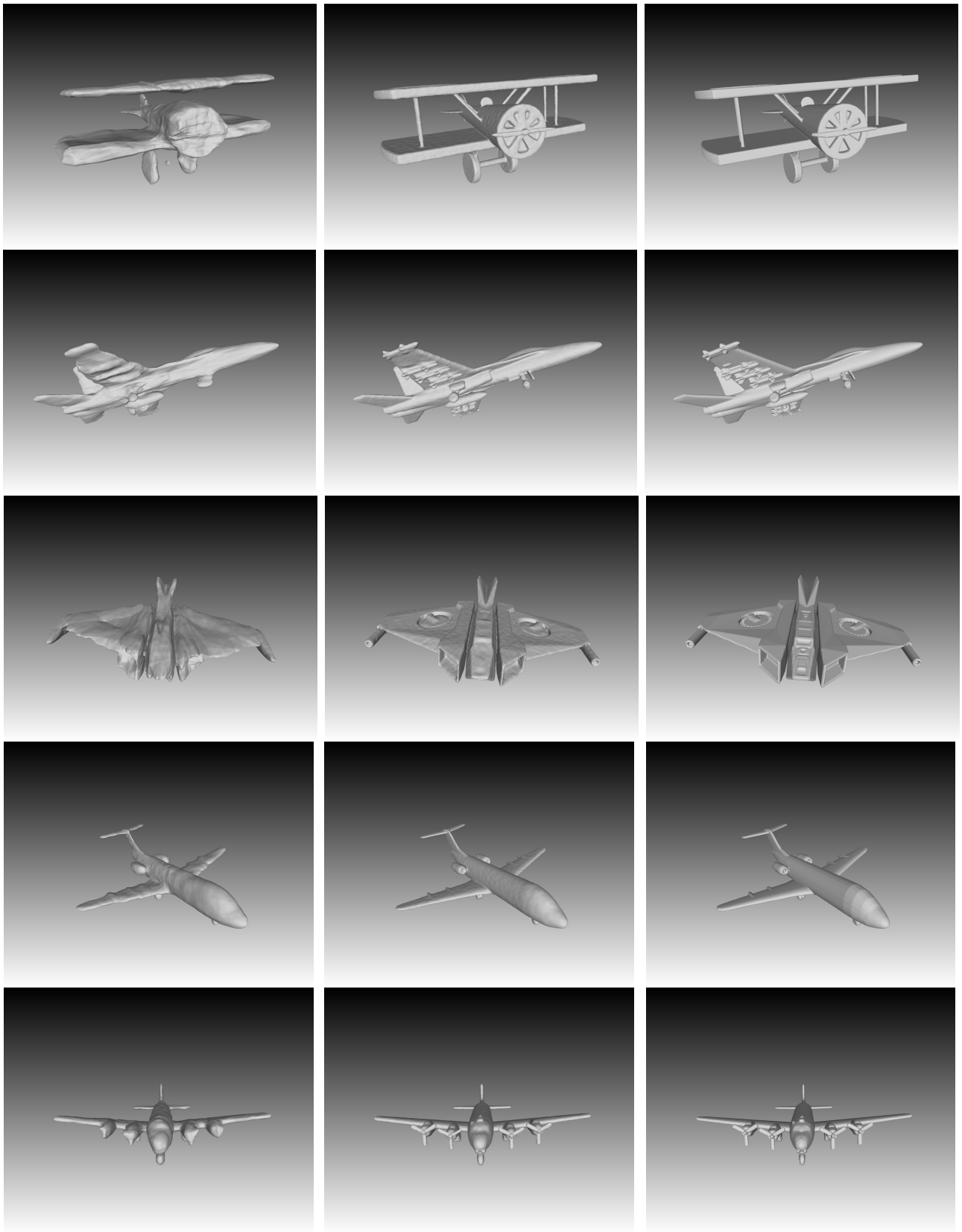

Figure 2: MLP 和 MLP with fourier features 的重建结果。放大查看细节。只有 MLP 生成的结果比较模糊且有较多细节缺失, 而使用 MLP with fourier features 生成的结果包含更多高频细节。

Methods	学习率	优化器	单次采样数	训练轮数	λ_{sdf}	λ_{grad}	训练用时/mins
MLP	10^{-4}	Adam	20000	60000	1.0	0.5	83
MLP w/ Fourier	2×10^{-4}	Adam	25000	100000	2.0	0.5	197

Table 1: 训练超参数和用时对比, 实验在单卡 NVIDIA RTX 4090D 进行.

2.3 测试细节

后处理. 由于 MLP with Fourier features 的输入维度较高, 对于一两个物体经过训练后空间中会残留极少量的无意义碎块, 在测试时, 通过 Marching Cubes 得到 Mesh 后, 只保留了最大的 connected component, 以去除这些碎块. 代码如下:

```
components = saved_mesh.split(only_watertight=False)
if components:
    # Find the largest component by number of faces
    largest_component = max(components, key=lambda component: len(component.faces))
    saved_mesh = largest_component
else:
    print("No components found or mesh is empty after split.")
```

当然有其他后处理方式, 比如设定一个阈值, 只保留大于该阈值的 connected component. 实验中采取了上述方式, 效率比较高. 注意**后处理只在 MLP with Fourier features 上进行**. MLP 由于高频细节较少, 部分物体会断开, 我选择直接使用 Marching Cubes 得到的 Mesh.

Marching Cubes. 直接调用了 `skimage.measure.marching_cubes`, 而后用 trimesh 保存为 `.obj` 格式. 可以通过传入命令行 `--level` 参数来设置 Marching Cubes 的 level.

测试用时. 当 `grid_size` 设置为 256 时, 两个方法的测试用时在 5-6 秒. 当 `grid_size` 设置为 512 时, 两个方法的测试用时在 40-50 秒, 后处理的耗时不大. 最终保存下来的结果均使用了 `grid_size = 512`, 以保证更高的精度.

3 结果分析与讨论

与 ground truth 相比, 只使用 MLP 的效果只能恢复出粗略的表面形状, 在 **Figure** 2 第一行的飞机模型中, 上下两层的连杆、螺旋桨等细节均未能恢复出来, 高频细节存在较多的缺失, 整体效果较差. 在训练过程中发现损失函数快速收敛且 $\mathcal{L}_{\text{sdf}}$ 下降到较小的值, 单纯通过增大训练步数不能够明显改善高频细节的恢复, 可能可以通过进一步增大模型或选择更精细的设计来改善结构的恢复. 限于时间原因没有进一步尝试.

使用 MLP with Fourier features 的方法能够恢复出更为精细的表面形状, 包含更多的高频细节, 较为接近 ground truth, 达到了预期的效果. 由于模型大小的限制, 以及没有设置更大的 Fourier features 的维度, 仍然存在一些更精细的高频细节的缺失, 例如第三行的飞机背部的纹理还存在模糊的情况, 以及最后一行飞机螺旋桨的棱角较为的平滑等等. 但相较于 MLP 的方法, 整体效果有了明显的提升.

参考文献

[1] Amos Gropp, Lior Yariv, Niv Haim, Matan Atzmon, and Yaron Lipman. Implicit geometric regularization for learning shapes, 2020.

[2] Matthew Tancik, Pratul P. Srinivasan, Ben Mildenhall, Sara Fridovich-Keil, Nithin Raghavan, Utkarsh Singhal, Ravi Ramamoorthi, Jonathan T. Barron, and Ren Ng. Fourier features let networks learn high frequency functions in low dimensional domains, 2020.