241116

Joint Training

模型独立训练的代码

GF模型

GF_main.py

model/layers/DenseGGNN.py

model/layers/DenseGraphMatching.py

model/GF_dataset.py

utils/GF_utils.py

utils/GF_early_stopping.py

config/GF_config.py

dataset/dxf_process_GF.py

DeepDXF模型

DeepDXF_train.py

model/DeepDXF_dataset.py

model/DeepDXF_embedding.py

model/DeepDXF_loss.py

model/transformer_encoder.py

utils/DeepDXF_early_stopping.py

utils/DeepDXF_utils.py

config/DeepDXF_config.py

dataset/dxf_process_GF.py

Joint train code

Joint_config.py

dataset/dxf_process.py

model/joint_dataset.py

model/joint_loss.py

utils/joint_early_stopping.py

utils/dynamic_weight.py

joint_main.py

joint_train.py

关键点

GF模型的几何编码矩阵的建立方式：

对每个实体建立一个边框，统计重叠数量，考虑10类实体，一张图就是10个节点，每个节点算一个实体类，每个节点维度：1（该实体类，剩余不重叠的数量）+10（该实体类和其它实体类重叠的数量，自身类与自身类重叠数量为0）组成，即该节点对于所有其它实体的重叠数量和该实体类不重叠的实体数量

对于几何编码矩阵的每一行，将自身类对应的自身类放在第一列，后续列为固定顺序的其它实体类，对第一列的所有行进行列归一化（利用起不同实体类，数量不同），对后续列的每一行进行行归一化”，

DeepDXF模型的序列编码矩阵构建方式：

DXF文件的实体句柄的值从小到大表示该实体添加入DXF文件的顺序，反映了设计意图和因果关系，某些实体的存在可能是其他实体存在的前提条件，我们将每个实体以及所有实体的参数作为序列编码矩阵的一行”，

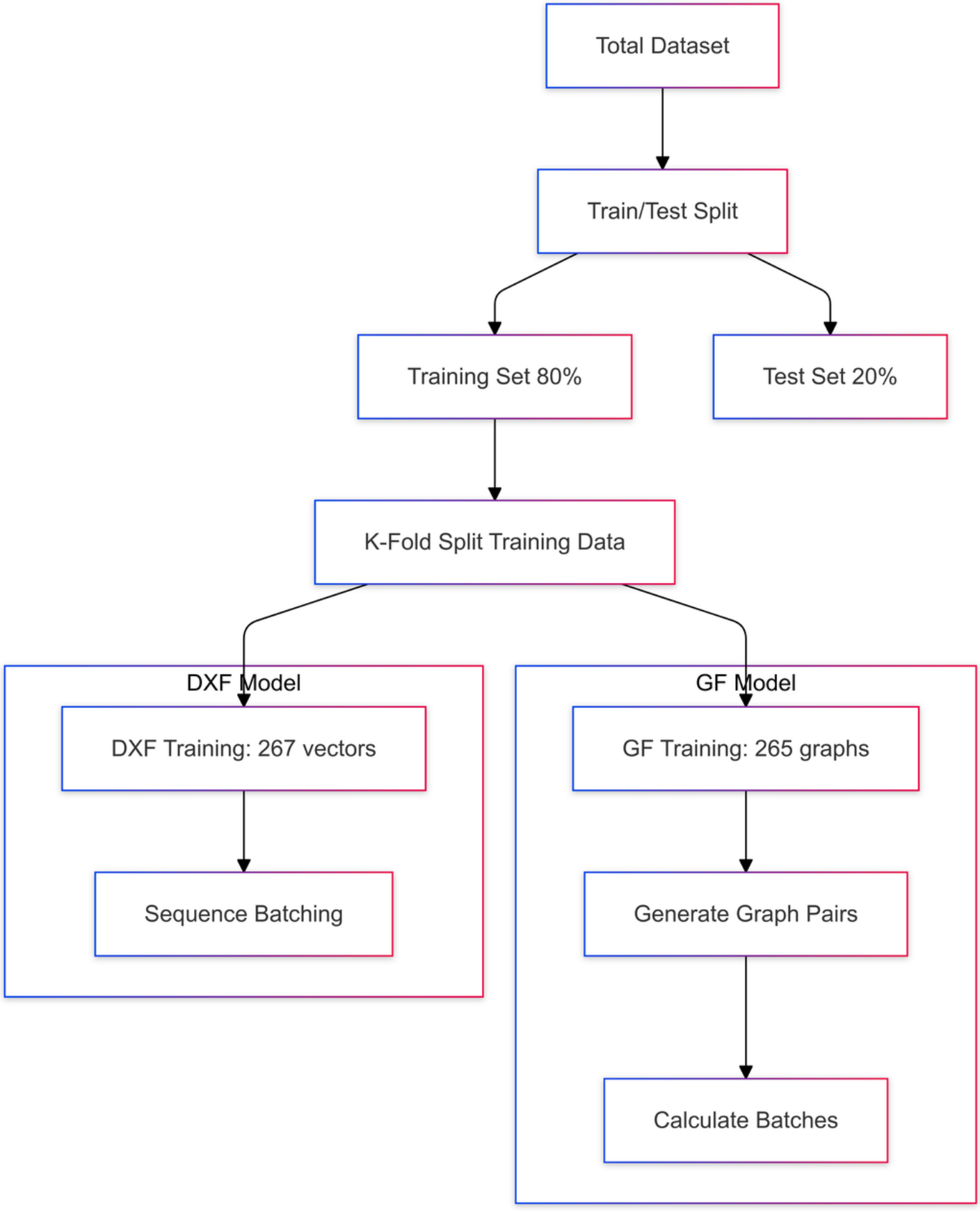
GF模型的邻接表的建立方式

在一份DXF文件中，所有实体按照句柄值从小到大的顺序排序，如果一个实体前是另一种实体，或者一个实体后是另一种实体，则在这两个实体类之间建立一条边。

数据集的划分

加载入GF模型和DeepDXF模型的数据集是不同的，我已经完成了联合训练的数据的预处理，当运行dxf_process.py，dxf_process.py分别调用dxf_process_CGMN.py和dxf_process_DeepDXF.py处理数据,处理完成后，GF模型和DeepDXF模型所需的数据就分别
在/mnt/share/DeepDXF_CGMN/encode/data/GF
和/mnt/share/DeepDXF_CGMN/encode/data/DeepDXF目录下。

加载入两个模型的数据各自都划分20%作为最后的测试数据，各自剩下80%用于各自的k折交叉验证的训练和验证



配置文件（config/joint_config.py）

对于GF模型和DeepDXF模型特有的配置可以使用导入的形式，从GF_config.py和DeepDXF_config.py导入，公共部分的配置则重新编写。

epochs均默认设为50，GF_batch_size=45,DeepDXF_batch_size=5

提供默认配置。

损失计算（实现动态权重分配机制）

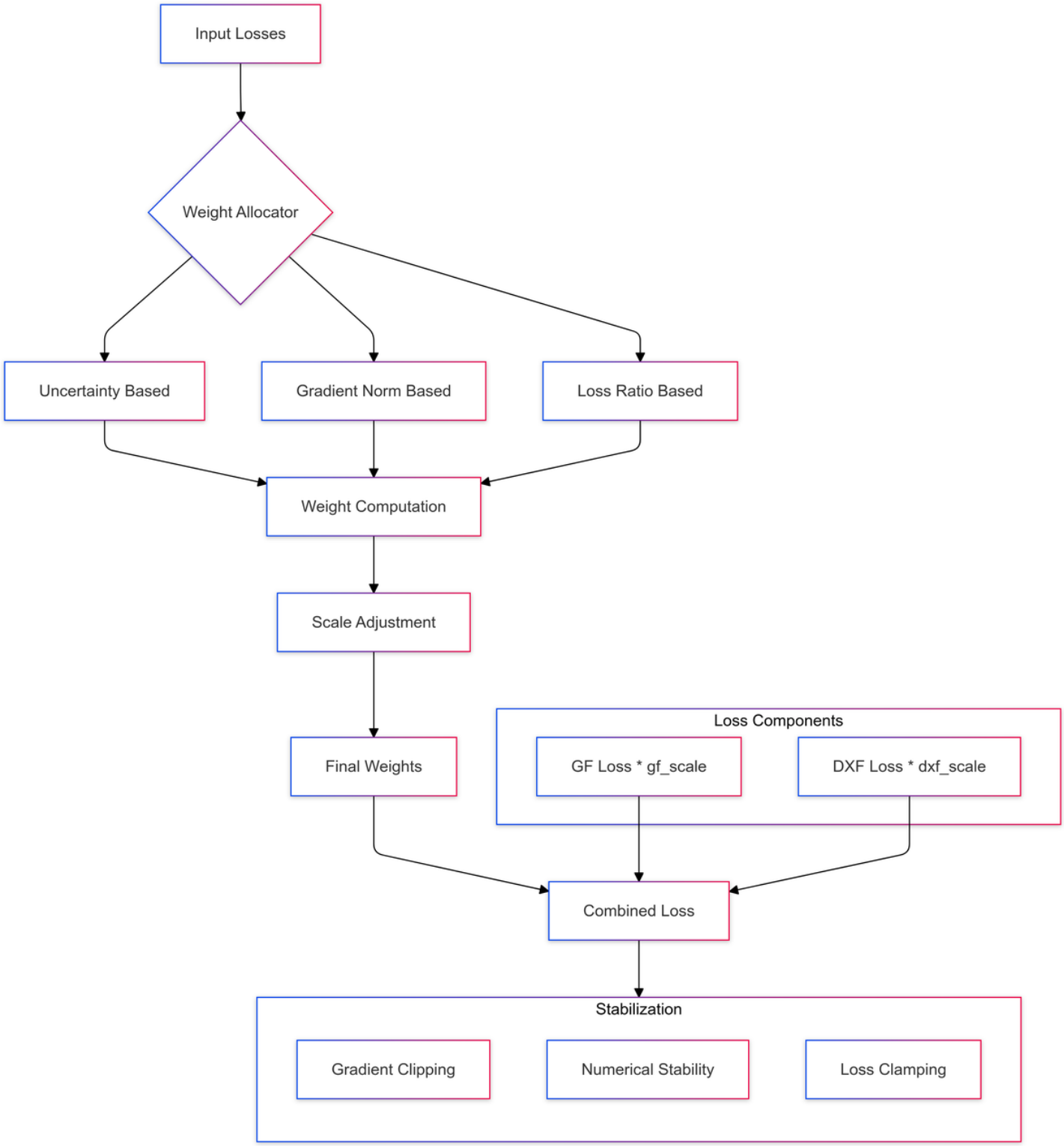
联合训练的损失为两个模型的组合损失（model/joint_loss.py）

直接导入两个模型的损失计算代码（model/layers/DenseGraphMatching.py和model/DeepDXF_loss.py）

组合损失的权重分配代码dynamic_weight.py

动态权重分配策略：

- 基于不确定度的权重计算
- 基于梯度范数的动态调整
- 损失比例自适应调整



早停机制（early_stopping.py）

联合训练的训练，验证，测试损失是基于两者的组合损失，而不是单个模型的损失。

工具类代码

直接导入GF_utils.py和DeepDXF_utils.py中已经编写好的相关函数

数据加载器（model/joint_dataset.py）

在joint_dataset.py直接导入model/DeepDXF_dataset.py和model/GF_dataset.py中已经编写好的相关函数，并补充联合训练所需要的新的函数

联合训练代码（joint_train.py）

参考两个模型单独训练时的训练代码GF_train.py和DeepDXF_train.py

GF模型和DeepDXF模型使用同一个优化器联合训练

主代码（joint_main.py）

运行joint_main.py就是开始联合训练GF和DeepDXF模型

使用梯度累积

梯度累积的原理是将多个batch的梯度累积起来，再进行一次参数更新。因此需要将每个batch的loss除以accumulation_steps，以保证等效batch size下的梯度幅度一致。但这个缩放只应该影响反向传播，不应该影响loss的记录和展示。

梯度累积的作用

1. **实现更大批量训练效果**：在不增加显存占用的情况下，模拟更大批量大小的训练效果，提升模型收敛稳定性。
2. **优化显存使用**：适用于内存或显存有限的设备，支持训练更大模型或使用更大输入数据。
3. **控制梯度波动**：通过更大的有效批量来减少每次参数更新中的梯度波动，使得模型的训练过程更加平稳。

梯度累积的优缺点

- **优点**：使得小批量训练更平稳，收敛效果更好，并且降低了显存使用需求。
- **缺点**：训练过程较慢，因为在实际参数更新前需要多次前向和反向传播，而且每次累积的步数增加了总的训练迭代次数。

批次数量同步迭代

每个epoch中取较小的两个模型较小的批次数量，保证每个批次的总损失都是两个模型的在当前批次的组合损失，通过同时迭代两个数据加载器，确保两个模型一起处理批次。

默认值DeepDXF_batch_size=5,GF_batch_size=45(因为DeepDXF模型在每个batch内的损失矩阵里面考虑了5*2（5*2-2）+5*2=90对图之间的对比损失，而GF模型每个采样的图对计算两个对比损失（因为对比损失是在同一个图的不同视图之间计算的），故默认GF_batch_size=45，实现在数量级上每一个batch中两个模型损失计算的图对相同)

`gf_loader` 的批次数量

```
def generate_pairs(self, graphs, batch_size):
    """为每个batch随机采样固定数量的图对
    Args:
        graphs: 图列表
```

```
        batch_size: 每个batch包含的图对数量(45)
    """
    n = len(graphs)
    print('Total graphs: {}'.format(n))

    # 计算batch数量: 图的数量//5
    num_batches = n // 5
```

- `gf_loader` 的批次大小使用默认值45

DeepDXF模型的数据加载：

DeepDXF模型的训练和验证也需要通过一个数据加载器（`dxf_loader`）来进行。`dxf_loader` 的创建是通过 `JointDataLoader` 类中的 `get_fold_loaders` 或 `get_test_loaders` 方法完成的。

  - 这里的批次计算稍有不同：DeepDXF模型的批量大小（`dxf_batch_size`）是基于训练集的样本数与 `gf_batch_size` 相匹配的。具体来说，`dxf_batch_size` 是通过以下方式计算的：

```
dxf_batch_size = int(len(fold_train_indices) / len(gf_train_loader))
```

这里的 `fold_train_indices` 是训练数据的索引，`gf_train_loader` 是基于 GF 数据集构建的数据加载器，`len(gf_train_loader)` 代表 GF 训练集的批次数，因此 DeepDXF 的批大小是通过使其与 GF 的批次数量相匹配来设置的。

CGMN代码的国产化

原因

他提供的模型，如果是每个样本独立成类，训练是没有意义的（他会把两个相同的图处理为初始的正样本对）

然后训练他的模型，他提供的AUC指标和逻辑分类准确率的指标针对的是分类问题的（首先同一类不能只有一个样本，否则正样本对就是自己和自己）

CGMN的标签作用

1. **标签y的作用仅限于初始构建正负样本对**

   正样本对是同一类中不同图，负样本对是不同类间的图

2. 视图内对比学习:

- 标签没有参与：视图内对比(feature_p1和feature_p2间)完全依靠自监督

3. 视图间对比学习:

- 同样未使用标签：跨视图对比也是自监督的，

数据处理的修改:

- 保持CGMN模型的数据处理方式
- 特征编码的离散化为[-1,255]之间的整数，除了DIMENSION中的数字部分（*1000）
- 移除所有与AUC指标和逻辑分类准确率相关的代码
- 对比学习的核心部分保持不变，只是修改了图对的生成机制

一个图就是一个节点级别表示的矩阵，先进行同一个图不同视图之间的信息融合（利用节点与节点之间的相似度加权），再进行不同图的不同视图之间的信息融合（利用节点与节点之间的相似度加权），对比损失是在同一个图不同视图之间计算的，最后把两个图的不同视图得到的对比损失取平均

新的图对生成机制

原始代码是为每个图生成一个正样本对（随机取同一类中的不同图），和一个负样本对（随机取不同类）

在每个epochs中，也就是某一折中，一次性生成所有的可能对，然后按batch_size划分

固定batch_size=128（每个batch处理128对图）->num_batchs=n = len(graphs)//（2*batch_size）=45（因为2个图组成1个图对）

n=16035，batchs=90,batch_size=128(64对)

图的均匀分配： 我们将图分配到每个 `batch` 时，确保每个 `batch` 包含的图是不同的，并且每个 `epoch` 的所有图都会被使用。

图对的生成： 在每个 `batch` 中，我们从 `batch` 内部的图中（128个图中）随机选取图对（形成64对），保证每个图对涉及的图是不同的，同时在该 `batch` 中不重复使用图。

图对的数量： 每个 `batch` 会生成64 个图对，这些图对由 分配到该batch的128个不同的图组合而成。

数据集划分方式:

移除原CGMN项目中的训练/测试/验证集划分方式

新的划分方式:

20%数据作为最终测试集

剩余80%数据进行10折交叉验证

早停指标

使用对比学习损失作为早停指标

训练的可视化

在命令行和本地保留训练日志，打印训练的必要信息，显示训练的进度条等

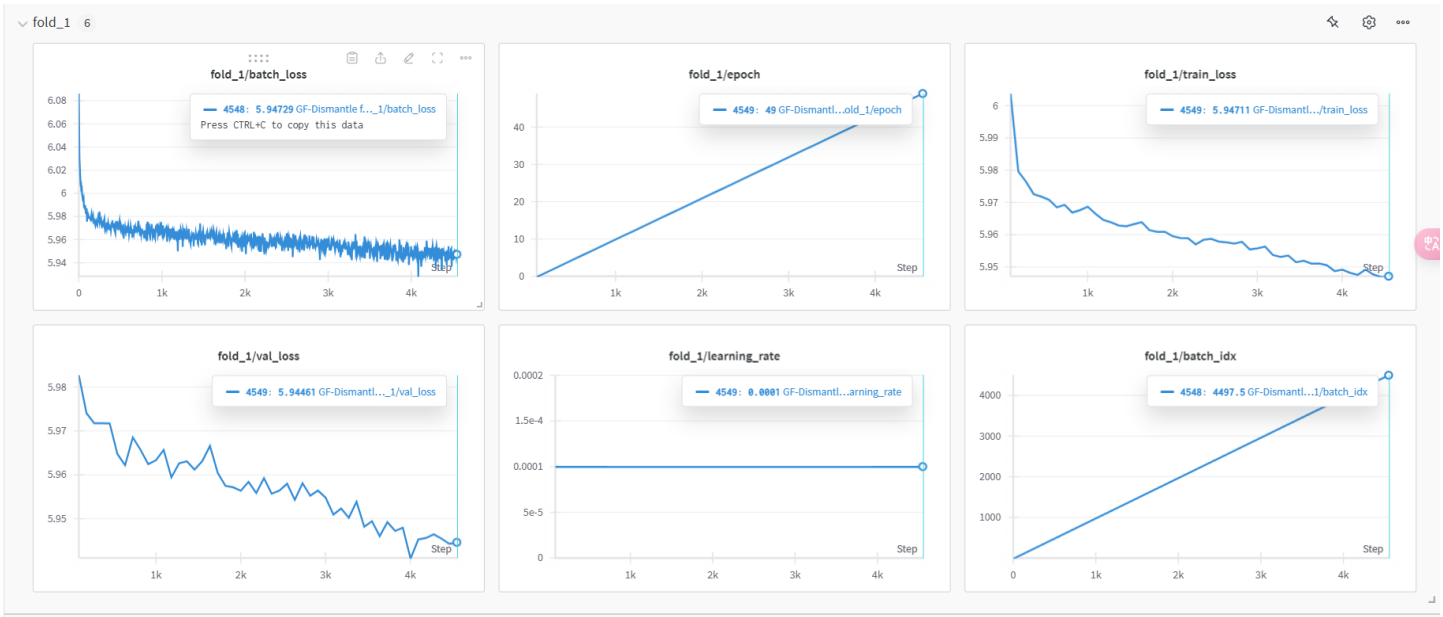
使用wandb云端可视化（仅仅修改配置和训练文件）

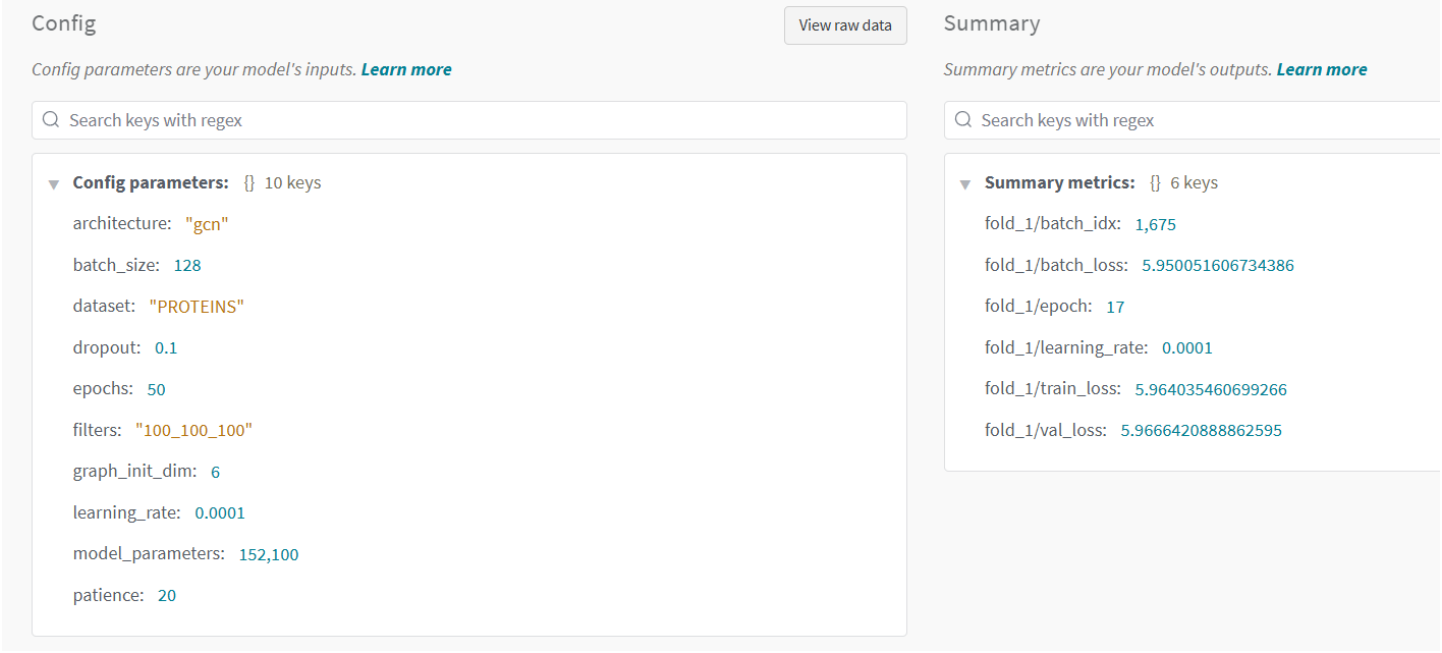
331

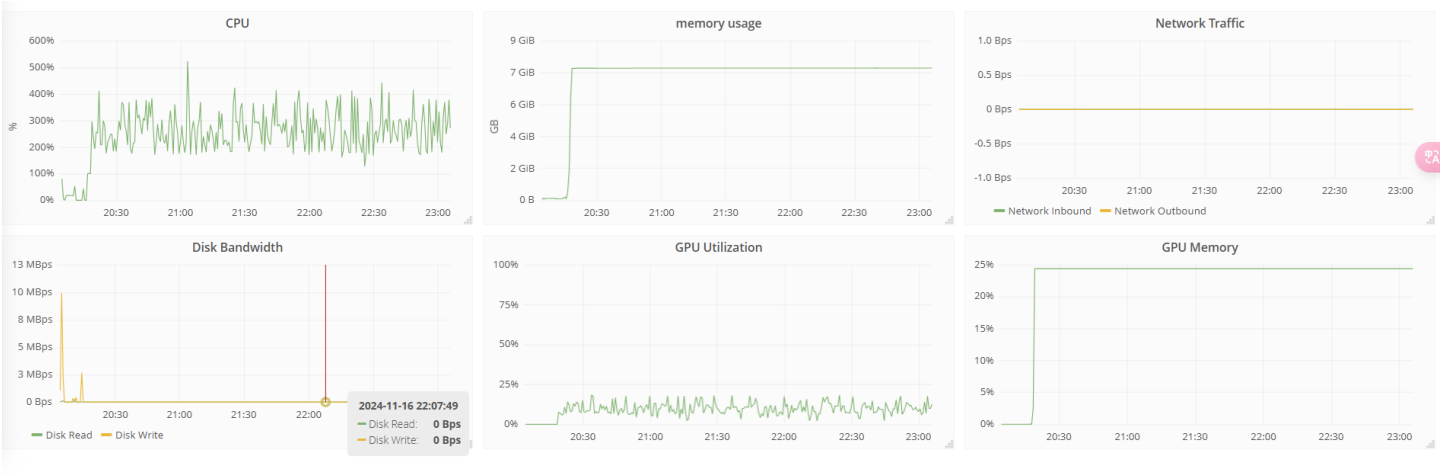
https://wandb.ai/102201525-fuzhou-university/GF/runs/68mm2ck9?nw=nwuser102201525

16035

3h，3*2080Ti







操作流程

```
pip install -i https://mirrors.aliyun.com/pypi/simple/ networkx
torch_geometric
在/opt/conda/lib/python3.7/site-packages/torch_geometric/profile
#from torch.profiler import ProfilerActivity, profile
cd /mnt/share/DeepDXF_CGMN/encode
```

DeepDXF代码完善

对比学习的损失计算方式的修改（完善InfoNCE和SimCLR等）

333个图

充分利用batch内的样本，整个batch中共有N个正样本对，2N×(2N-2)个负样本对

假设 `batch_size = 5`：

- **总样本数：** `2 * 5 = 10`。
- **正样本对数量：** `5`。
- **每个样本的负样本数量：** `10 - 2 = 8`。
- **总的负样本对数量：** `10 * 8 = 80`。

正样本是同一个图经过dropout后的两个不同的视图，负样本是来源不同图的视图

使用矩阵方式进行对比损失的计算，矩阵上的每一个数，代表不同图对之间的相似度

一个图经过投影后得到图级表示的向量，一批N个图，矩阵是2N*2N，第一列都表示正样本对，后续列表示负样本对，比如（1，1）表示1号图的第一个视图与1号图的第二个视图之间的相似度（正负样本对之间）,(1,2)表示1号图的第一号视图与2号图的第一个视图之间的相似度，然后基于这个矩阵，进行该批次的对比损失计算，矩阵运算训练很快，原始论文batch_size是设置为1024，epochs=1000

正则化和温度参数调节正负样本的区分度

使用混合精度训练

提高计算速度：FP16的计算速度显著快于FP32，尤其在GPU上可以更有效地利用硬件资源。由于FP16运算速度更快且能提升并行计算的效率，因此在大型模型和数据集上，训练速度有显著提高。

减少显存占用：FP16占用的显存是FP32的一半，因此可以在有限的显存中放入更大的批量（batch size），提高数据吞吐量，减少显存压力。

缩放梯度： 使用 `scaler.scale(loss).backward()` 来计算梯度，这时梯度会被放大（缩放因子是 `scale`）。

反向缩放： 在进行优化之前，调用 `scaler.unscale_()` 来取消梯度的缩放，恢复到原始的尺度。

梯度裁剪： 可以在取消缩放之后进行梯度裁剪。

优化器步骤： 使用 `scaler.step(optimizer)` 和 `scaler.update()` 来更新参数和调整缩放因子。

数据划分

先划分20%作为测试数据，剩下80%进行k折交叉验证，并使用早停机制（DeepDXF_early_stopping.py）

相似度计算

python SimDeepDXF.py --model_path checkpoints/best_model.pth --file1 data/test/QFN21LA(Cu) -508 Rev1_5.h5 --file2 data/test/QFN21LA(Cu) -508 Rev1_3.h5 --method cosine

实体以及参数的完善

增加SOLID，SPLINE

```
SOLID 特征:
{'points': [Vec3(-45.9049250061654, 5.532250833443754, 0.0),
Vec3(-46.2620678633059, 3.389393690603469, 0.0), Vec3(-45.54778214902774,
3.389393690603469, 0.0), Vec3(-45.54778214902774, 3.389393690603469, 0.0)],
'is_3points': False}
{'points': [Vec3(52.95649949488415, 4.999999999962369, 0.0),
Vec3(53.31364235202466, 2.857142857122312, 0.0), Vec3(52.59935663774365,
2.857142857122312, 0.0), Vec3(52.59935663774365, 2.857142857122312, 0.0)],
'is_3points': False}
包含所有顶点坐标（3点或4点）
判断是否为3点SOLID

SPLINE 特征:
{'degree': 3, 'control_points': [(53.58695631152727, -57.80744777776078, 0.0),
(55.00243766886592, -61.19458633995589, 0.0), (52.26374405508522,
-61.6180566631773, 0.0), (53.55487426490435, -65.3318275429682, 0.0),
(53.56358557980434, -65.35688452587658, 0.0), (53.57248034288137,
-65.3820912919183, 0.0), (53.5815584765644, -65.40744777775751, 0.0)],
'knots': [0.0073395497773584, 0.0073395497773584, 0.0073395497773584,
0.0073395497773584, 1.0, 1.0, 1.0, 1.006697525706382, 1.006697525706382,
1.006697525706382, 1.006697525706382]}
样条曲线的度数
控制点列表
节点向量
```

不考虑ELLIPSE，几乎没出现

训练的可视化

在命令行和本地保留训练日志

使用wandb云端可视化（仅仅修改配置和训练文件）

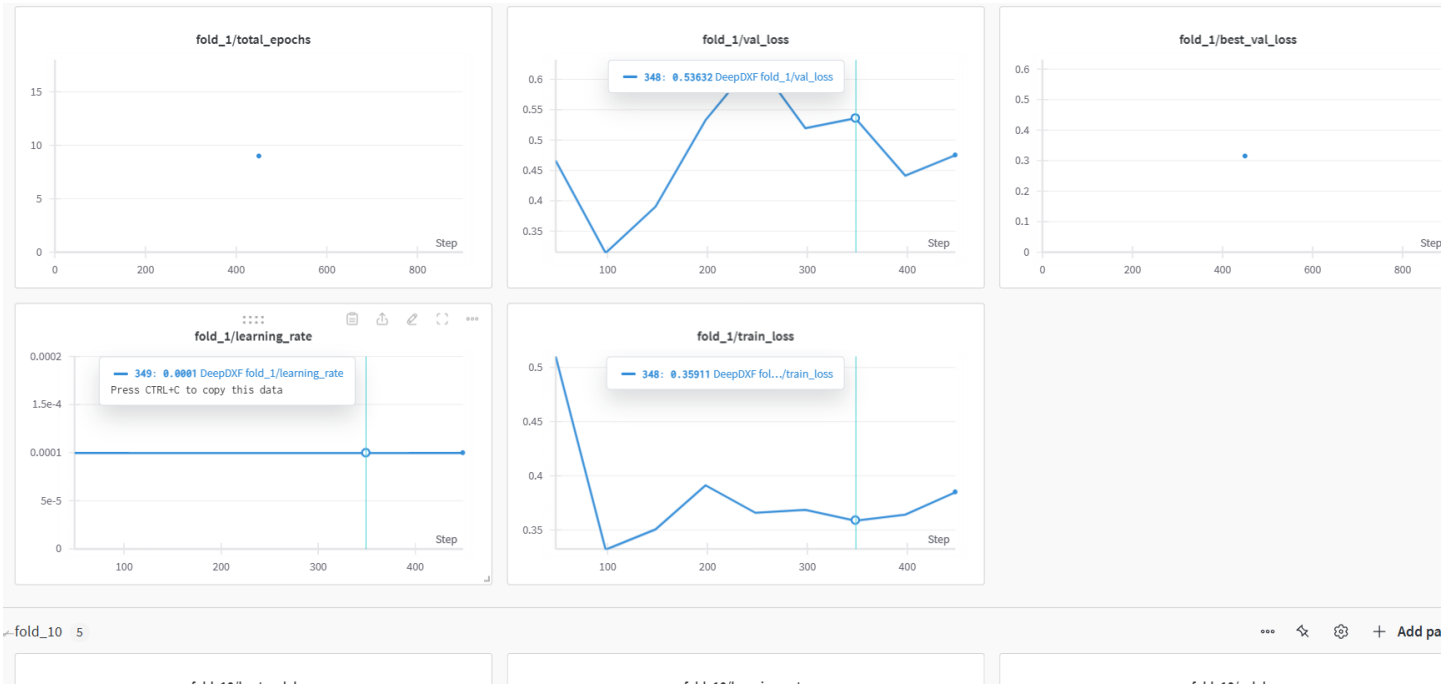
操作流程

```
apt-get update
pip install --upgrade pip -i https://mirrors.aliyun.com/pypi/simple/
pip install -i https://mirrors.aliyun.com/pypi/simple/ wandb
pip install wandb -i https://pypi.tuna.tsinghua.edu.cn/simple
```

```
wandb login
export WANDB_ENTITY="102201525-fuzhou-university"
export WANDB_PROJECT="DeepDXF";是项目名，而不是具体运行的名称
wandb status
python DeepDXF_train.py
```

https://wandb.ai/102201525-fuzhou-university/DeepDXF?nw=nwuser102201525

不正常



待办

DeepDXF参数修改，重新确定

DeepDXF模型训练

GF模型训练出来

联合训练的重新修改