241026

项目代码解释

数据处理:

json2json.py(将一组坐标化成单个拆开并保存为新的json，确保每个特征就是对应一个参数)

scale_bbox.py(对连续性参数进行规范化到一定大小[-1,1]，并将结果保存为json文件)

dxf_process_DeepDXF.py

将（scale_bbox.py，json2json.py，dataset/json2vec.py,dataset/DeepDXF_json_dataset.py）融合在一份代码中，直接从原始的dxf文件->h5文件

dxf_process_CGMN.py，直接从原始的dxf文件->”json“文件

dxf_process.py集成了dxf_process_DeepDXF.py和dxf_process_CGMN.py

DeepDXF:

DeepDXF_dataset.py

DeepDXF_embedding.py

DeepDXF_loss.py

transformer_encoder.py

DeepDXF_train.py

CGMN:

CGMN_cfg_train.py

CGMN_dataset.py

model/layers

CGMN_cfg_config.py

CGMN_utils.py

simgnn_utils.py

相似度：

sim.py

SimCGMN.py

SimDeepDXF.py

训练文件：

train.py

DeepDXF构建

part 0:原始数据到JSON

考虑10种实体类

HATCH去除pattern_name，TEXT去除text，MTEXT 去除text，DIMENSION去除dim_type，LEADER去除annotation_type，INSERT去除name

如果坐标是三维的，去除z轴坐标

HATCH 特征:

{'pattern_name': 'ANSI31', 'solid_fill': 0, 'associative': 0, 'boundary_paths': 8}

{'pattern_name': 'ANSI31', 'solid_fill': 0, 'associative': 1, 'boundary_paths': 4}

pattern_name: 填充图案名称

solid_fill: 是否为实体填充， 0 表示非实体填充,即使用图案填充

associative: 是否为关联填充，0 表示非关联填充,1 表示关联填充。关联填充意味着填充边界与其他实体（如线、圆等）相关联。当这些边界实体发生变化时，填充会自动更新以匹配新的边界。

'boundary_paths'：边界路径数量指的是定义填充区域的封闭轮廓的数量

TEXT 特征:

{'text': 'HALF ETCH ON BOTTOM', 'insert_point': Vec3(396.0818019561805, 139.6943445264609, 0.0), 'height': 0.08, 'rotation': 0}

text: 文本内容

insert_point: 插入点坐标

height: 文字高度

rotation: 旋转角度

MTEXT 特征:

{'text': 'FULL METAL', 'insert_point': Vec3(404.1361941913078, 142.9183805374713, 0.0), 'char_height': 0.06666666, 'width': 0.6219441976340474, 'rotation': 0}

text: 多行文本内容

insert_point: 插入点坐标

char_height: 字符高度

width: 文本框宽度

LWPOLYLINE 特征:

{'closed': True, 'points': [(395.819853154926, 139.6625137883657, 0.0, 0.0, 0.0), (395.9806541764283, 139.6625137883657, 0.0, 0.0, 0.0), (395.9806541764283, 139.8233148098681, 0.0, 0.0, 0.0), (395.819853154926, 139.8233148098681, 0.0, 0.0, 0.0)], 'count': 4}

closed: 是否闭合

points: 多段线的点列表

count: 点的数量

ARC 特征:

{'center': Vec3(398.8221825223083, 143.4364333052549, 0.0), 'radius': 0.11875, 'start_angle': 180.0, 'end_angle': 90.0}

center: 圆心坐标

radius: 半径

start_angle: 起始角度

end_angle: 结束角度

LINE 特征:

{'start_point': Vec3(398.9971825223084, 143.5551833052549, 0.0), 'end_point': Vec3(398.5440575223073, 143.5551833052551, 0.0)}

start_point: 线段的起点坐标

end_point: 线段的终点坐标

CIRCLE 特征:

{'center': Vec3(398.8221825223083, 142.9364333052549, 0.0), 'radius': 0.11875}

center: 圆心坐标

radius: 半径

DIMENSION 特征:

{'defpoint': Vec3(400.3284325223083, 146.1966954184052, 0.0), 'text_midpoint': Vec3(399.7003075223083, 146.1966954184052, 0.0), 'dim_type': 0}

defpoint: 定义点坐标

text_midpoint: 文本中点坐标

dim_type: 标注类型

LEADER 特征:

{'vertices': [(398.2190951127764, 141.8598343414721, 0.0), (396.7597151407759, 140.0412732757189, 0.0), (395.7713643418502, 140.0412732757189, 0.0)], 'annotation_type': 3}

vertices: 引线的顶点列表

annotation_type: 注释类型

INSERT 特征:

{'name': '*U121', 'insert_point': Vec3(384.5059270744619, 143.5915590496204, 0.0), 'scale': (1, 1, 1), 'rotation': 0}

name: 插入块的名称

insert_point: 插入点的坐标

scale: X、Y、Z方向的缩放比例

rotation: 旋转角度

将一组坐标化成单个拆开并保存为新的json，确保每个特征就是对应一个参数,这样得到如下38种参数：

solid_fill:

associative:

boundary_paths：

text_insert_point_x,text_insert_point_y

height:

text_rotation:

mtext_insert_point_x,mtext_insert_point_y

char_height:

width:

closed:

points_x,points_y

count

arc_center_x,arc_center_y

arc_radius

start_angle

end_angle

start_point_x,start_point_y

end_point_x,end_point_y

circle_center_x,circle_center_y

circle_radius

defpoint_x,defpoint_y

text_midpoint_x,text_midpoint_y

vertices_x,vertices_y

insert_insert_point_x,insert_insert_point_y

scale_x,scale_y

insert_rotation

对连续性参数进行规范化到一定大小[-1,1]

坐标参数（如 `start_point`、`end_point`、`center`、`insert_point`、`defpoint`、`text_midpoint`、`points`、`vertices`）：

- 对于每个坐标的 x 和 y 分量，使用 `normalize_coordinate` 函数将其归一化到 [-1, 1] 范围。
- 公式：`normalized_value = 2 * (value - min_value) / (max_value - min_value) - 1`。

长度或尺寸参数（如 `radius`、`height`、`char_height`、`width`）：

- 使用 `normalize_length` 函数，根据模型的最大尺寸 `max_dim` 进行归一化。
- 公式：`normalized_length = value * (2 / max_dim)`。

角度参数（如 `rotation`、`start_angle`、`end_angle`）：

- 角度参数表示方向，不进行缩放，保持原始值。

非坐标和非尺寸参数（如 `solid_fill`、`associative`、`boundary_paths`、`dim_type`、`annotation_type`）：

- 这些参数表示分类、布尔或整数值，不需要缩放，保持原始值。

名称和文本参数（如 `pattern_name`、`text`、`name`）：

- 名称和文本不进行处理，直接保留原始值。

Part 1：从JSON到DXFSequence:

将JSON数据解析为DXFSequence对象

- 定义DXFSequence类,类似于CADSequence,用于表示一个DXF文件中的实体序列。
- 实现DXFSequence.from_dict(json_data)方法,将JSON数据解析为DXFSequence对象。
- 解析过程中,根据实体类型创建相应的实体对象(如Line、Circle等),并将其添加到DXFSequence中。

DXFSequence的规范化

实现DXFSequence.normalize()方法,对部分连续参数(如坐标、长度、角度等)量化为[0, 255]的整数，对于角度值，如text_rotation，insert_rotation，Arc类中的start_angle和end_angle，先将其值除以360进行归一化，然后再量化到[0, 255]范围，对原本就是离散型的参数，范围截断在[0,255],最终所有参数存储为8位整数，

对于每个实体,调用其normalize()方法。

Padding至固定长度以及限制最大命令数量

设置最大实体数量Nc(512),筛选并丢弃超过Nc个实体的DXF文件。

对于实体数量小于Nc的DXF文件,在序列末尾添加空实体(如'EOS'),直到序列长度达到Nc。

其中‘EOS’类型是DXF文件中本身不存在的，我引入‘EOS’类型的目的是为了在DXF文件的实体不足512个时，用‘EOS’类型填充为512个.

每个实体考虑256种参数，每个参数处理为[0,255]，该实体若涉及该参数，这参数值为-1.

连续参数的量化

对于每个实体的连续参数(如坐标、长度、角度等),使用以下公式将其量化为[0, 255]的整数:

```
value = ((value + 1.0) / 2 * 256).round().clip(min=0, max=255).astype(np.int)
```

将量化后的参数存储为8位整数，这样每个实体的连续参数离散化为256个级别

```
Datasets:
  - dxf_vec
    Shape: (1, 512, 39)
    Dtype: int16
    All elements (3D):
    Sample 0:
      Row 0: [  4  -1  -1  -1  76   0 131 128  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1]
      Row 1: [  4  -1  -1  -1  41  16 131 128  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1]
      Row 2: [  4  -1  -1  -1 117  47 130 128  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1]
      Row 3: [  0  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1]
      Row 4: [  0  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1]
      Row 5: [  0  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1]
      Row 6: [  0  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1]
      Row 7: [  1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1]
      Row 8: [  1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1]
      Row 9: [  0  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1]
      Row 10: [  0  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1]
      Row 11: [  0  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1]
      Row 12: [  0  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1]
      Row 13: [  1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1]
      Row 14: [  1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1]
      Row 15: [  0  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1]
      Row 16: [  0  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1]
      Row 17: [  0  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1]
      Row 18: [  0  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1]
      Row 19: [  1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1]
      Row 20: [  1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1]
      Row 21: [  0  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1]
      Row 22: [  0  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1]
      Row 23: [  0  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1]
      Row 24: [  0  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1]
      Row 25: [  2  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1]
```

DXFSequence到向量的转换

实现DXFSequence.to_vector()方法,将DXFSequence转换为固定长度的向量表示。

向量的每一行对应一个实体,第一列为实体,后面的所有列为所有实体的集体参数，对于没有使用的参数,用特殊值(-1)填充

将向量保存为h5文件

将向量保存为h5文件

- 将向量表示保存为h5文件,便于后续的快速加载。

Part 2: 规范化向量到Transformer输入嵌入的转换

数据加载:

定义DXFDataset类,用于加载预处理后的规范化向量数据

实现DXFDataset.*getitem*()方法,从h5文件中读取一个DXF文件的向量表示。

实体类型和参数的分离

将向量中的第一列(实体类型)和后面的列(实体参数)分离成两个张量

```
entity_type = dxf_vec[:, 0]
entity_params = dxf_vec[:, 1:]
```

构建实体嵌入

定义DXFEmbedding类,用于将实体类型和参数转换为嵌入向量。

将每个实体嵌入 $e(C_i)$ 视为三个部分的总和：**实体类型嵌入** e^i_{cmd} 、**实体参数嵌入** e^i_{param} 和 **位置嵌入** e^i_{pos}

实体类型嵌入（e^i_{cmd}）

定义嵌入矩阵 W_{cmd} (shape: [d_{model} , 11]),用于将实体类型映射为 d_{model} (通常取256)维嵌入向量。

将实体类型 t_i 转换为11维one-hot向量 δ^c_i 。

计算实体类型嵌入: $e^i_{cmd} = \mathrm{W}_{cmd}\delta^c_i$ 。

实体参数嵌入（e^i_{param}）

定义两个嵌入矩阵 W^a_{param} (shape: [d_{model} , 64*38])和 W^b_{param} (shape: [64, 257]),用于将实体参数映射为 d_{model} 维嵌入向量。

将每个参数转换为257维one-hot向量,并将所有参数的one-hot向量堆叠为矩阵 δ^p_i (shape: [257, 38])。

计算实体参数嵌入: $e^i_{param} = \mathrm{W}^a_{param} \cdot \mathrm{flat}(\mathrm{W}^b_{param} \cdot \delta^p_i)$ 。

位置嵌入（e^i_{pos}）

使用基于查找表的可学习位置编码

定义位置嵌入矩阵 W_{pos} (shape: [d_{model} , Nc]),用于将实体位置映射为 d_{model} 维嵌入向量。

将实体位置i转换为 Nc 维one-hot向量 δ_i 。

计算位置嵌入: $e^i_{pos} = \mathrm{W}_{pos}\delta_i$

合并嵌入

- 最后，将 e^i_{cmd} , e^i_{param} e和 e^i_{pos} 三者相加，得到最终的实体嵌入： $\boldsymbol{e}(C_i) = e^i_{cmd} + e^i_{param} + e^i_{pos}$

输入Transformer编码器

- 将DXF文件的实体嵌入序列 $e(C_1)$, ..., $e(C_{N_c})$ 输入Transformer编码器。

序列长度的降维

将8个实体视为一个新实体将512个降维为64个

目标：将输入transformer的 N_c 从512变为64

方式：按8个单元取最大

---

part3:从embding到latent

Encoder处理:

- src通过TransformerEncoder,包含4个TransformerEncoderLayer

- 每层包含:
  - 多头自注意力机制
  - 前馈神经网络
  - 层归一化和残差连接
- 输出memory (S,N,d_model)

潜在向量生成:

- 使用padding_mask提取有效token
- 对有效token进行平均池化: z = (memory * padding_mask).sum(dim=0, keepdim=True) / padding_mask.sum(dim=0, keepdim=True)
- 得到z (1,N,d_model)

Bottleneck处理:

- z通过一个线性层和tanh激活函数: z = tanh(linear(z))
- 最终得到潜在向量z (1,N,dim_z)

part4:从latent Vector到对比损失

生成Latent Vector:

- DXF模型的构造序列通过Transformer编码器处理，得到初始的潜在向量z。
- z经过Bottleneck层(线性层)处理。
- 然后通过投影层(Projection Layer)得到最终的潜在向量。

对比学习增强:

- 对同一个潜在向量z应用两次dropout，得到两个增强版本proj_z1和proj_z2。
- 这两个版本作为正样本对。

对比损失计算: 仿照ContrastCAD提供的两种对比损失计算方法：

a. InfoNCE损失:

- 将proj_z1和proj_z2归一化。
- 计算批次内所有样本对的相似度矩阵。
- 使用温度参数(temperature)缩放相似度。
- 将同一DXF模型的两个增强版本视为正样本对，其他为负样本。
- 使用交叉熵损失计算对比损失。

b. SimCLR损失:

- 将proj_z1和proj_z2归一化并取平均。
- 计算批次内样本之间的相似度。
- 应用温度缩放。
- 使用掩码处理自身相似度。
- 计算交叉熵损失，将同一DXF模型的两个增强版本视为正样本。

损失函数

- 仅仅使用对比损失更新相关参数，最终训练后的效果达到使相似的DXF文件输入，得到相似的最终的潜在空间向量

CGMN和DeepDXF融合

邻接矩阵的建立方式：

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

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

原始数据->训练阶段

新增一个dxf_process.py文件,实现从原始DXF文件到几何编码矩阵、邻接表以及实体类型和参数json的转换，以便调用main.py文件，自动返回处理好的几何编码矩阵、邻接表以及实体类型和参数json，两个模型的数据处理都是从原始的dxf开始->中间文件->加载入模型开始训练

几何编码矩阵、邻接表:dxf->提供到所需的位置->训练

实体类型和参数json:将原始dxf->h5->训练，

train.py

两个模型会独立训练,但共享优化器进行联合优化。每个模型只关注自己的损失。总的损失是两个模型损失的加权和,权重可以通过cgmn_weight和dxf_weight参数调整。

实现了CGMN和DeepDXF模型联合训练，得到一份公共的pth文件

相似性计算

input：DXF文件1的json文件和h5文件，DXF文件2的json文件和h5文件

对于CGMN的模型的相似度计算方式是模型加载入指定两个json文件，然后经过训练好的CGMN模型，对输出的特征矩阵进行相似度计算，得到两个图之间的几何特征的相似度分数。

对于DeepDXF的模型的相似度计算方式是模型指定两个指定的h5文件，经过训练好的DeepDXF模型，得到两个Latent Vector，对这个向量进行相似度计算，得到两个图之间的几何特征的相似度分数。

总的相似性分数是CGMN的模型的几何特征的相似度分数和DeepDXF的模型的几何特征的相似度分数加权作为两份DXF文件的相似度

实验结果

DeepDXF

```
Epoch 10/10, Batch 0/1, Loss: 4.097306251525879
Epoch 10/10, Average Loss: 4.097306251525879
Model saved successfully
```

CGMN

```
/opt/conda/lib/python3.7/site-packages/sklearn/utils/validation.py:72: DataConversi
onWarning: A column-vector y was passed when a 1d array was expected. Please change
 the shape of y to (n_samples, ), for example using ravel().
  return f(**kwargs)
Final Testing: the cosine AUC score is: 0.8601466049382716
Final testing: the lr AUC score is: 0.7875363468146973
Best_valid_auc =  0.8285322359396433
Best_test_auc =  0.8601466049382716
END
```

```
Final Testing: the cosine AUC score is: 0.9805194805194805
Final testing: the lr AUC score is: 0.9375
Best_valid_auc =  1.0
Best_test_auc =  0.9805194805194805
END
```

CGMN_DeepDXF

样本太少，过拟合

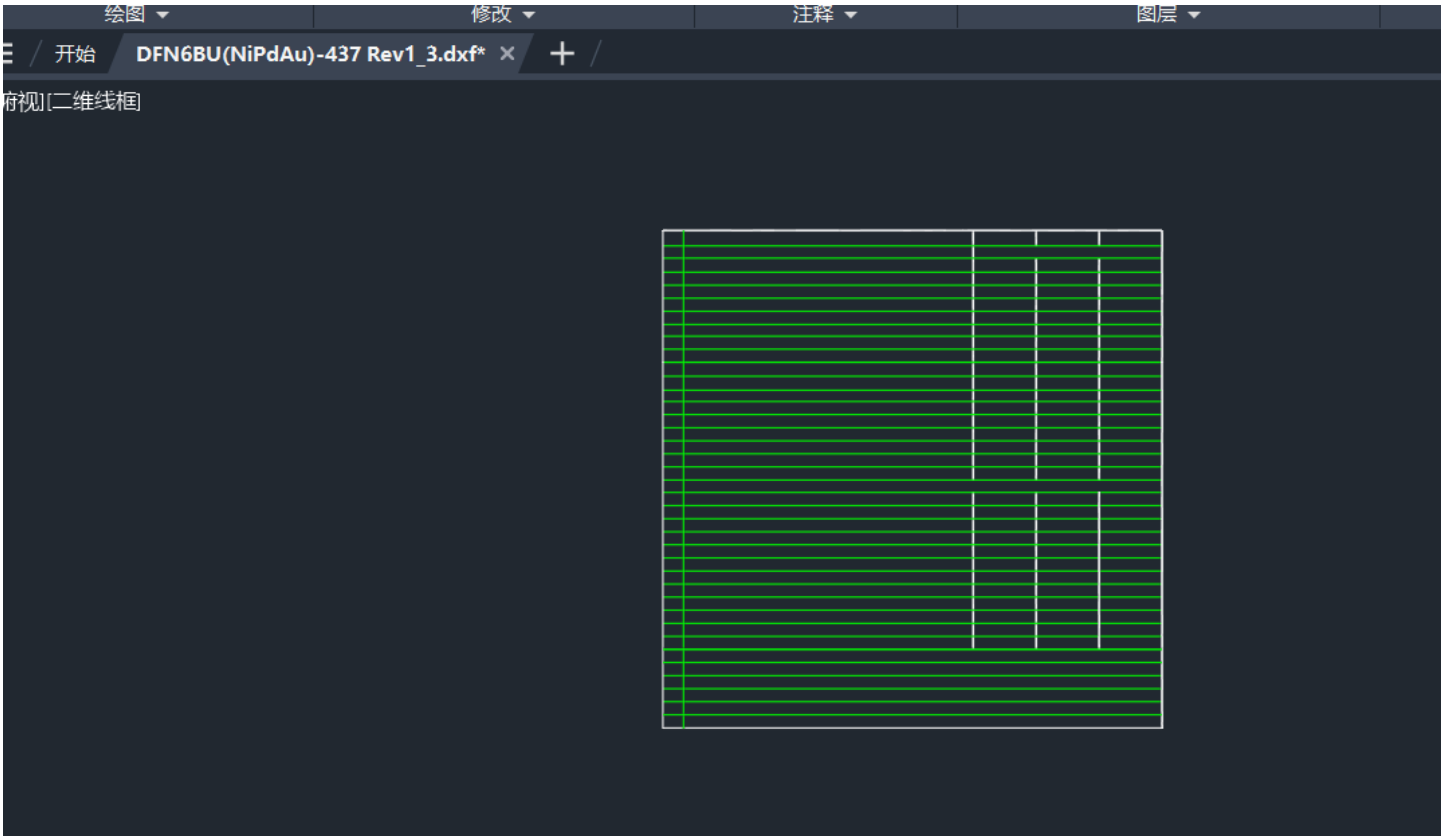
```
Computed AUC score: 0.9909
Validation Score: 0.9909
Validation score decreased (0.990909 --> 0.990909).  Saving model ...

Training completed successfully!
Best validation score: 0.9909
Model saved to: ./checkpoints/combined_model.pth
```

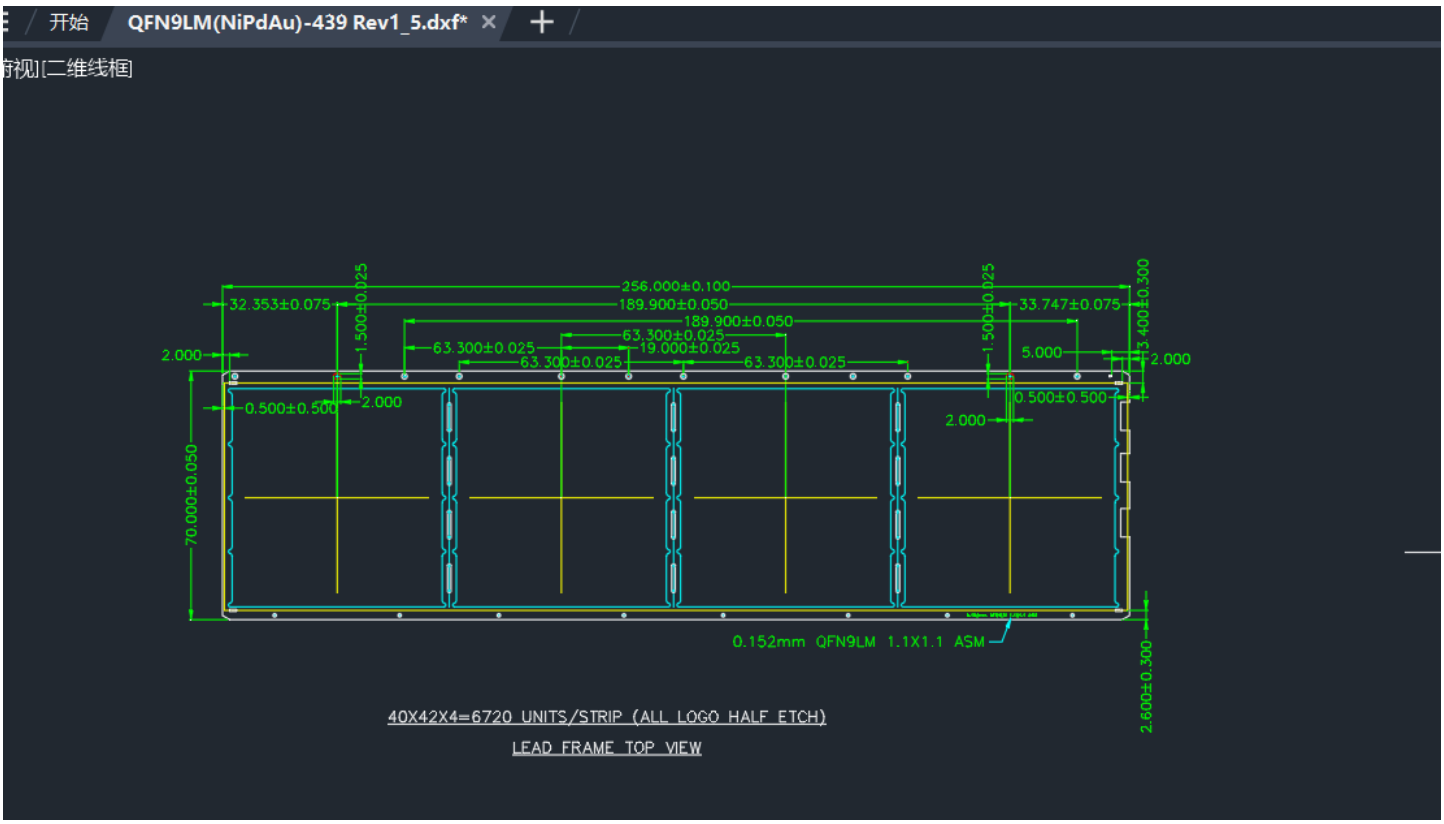
相似结果

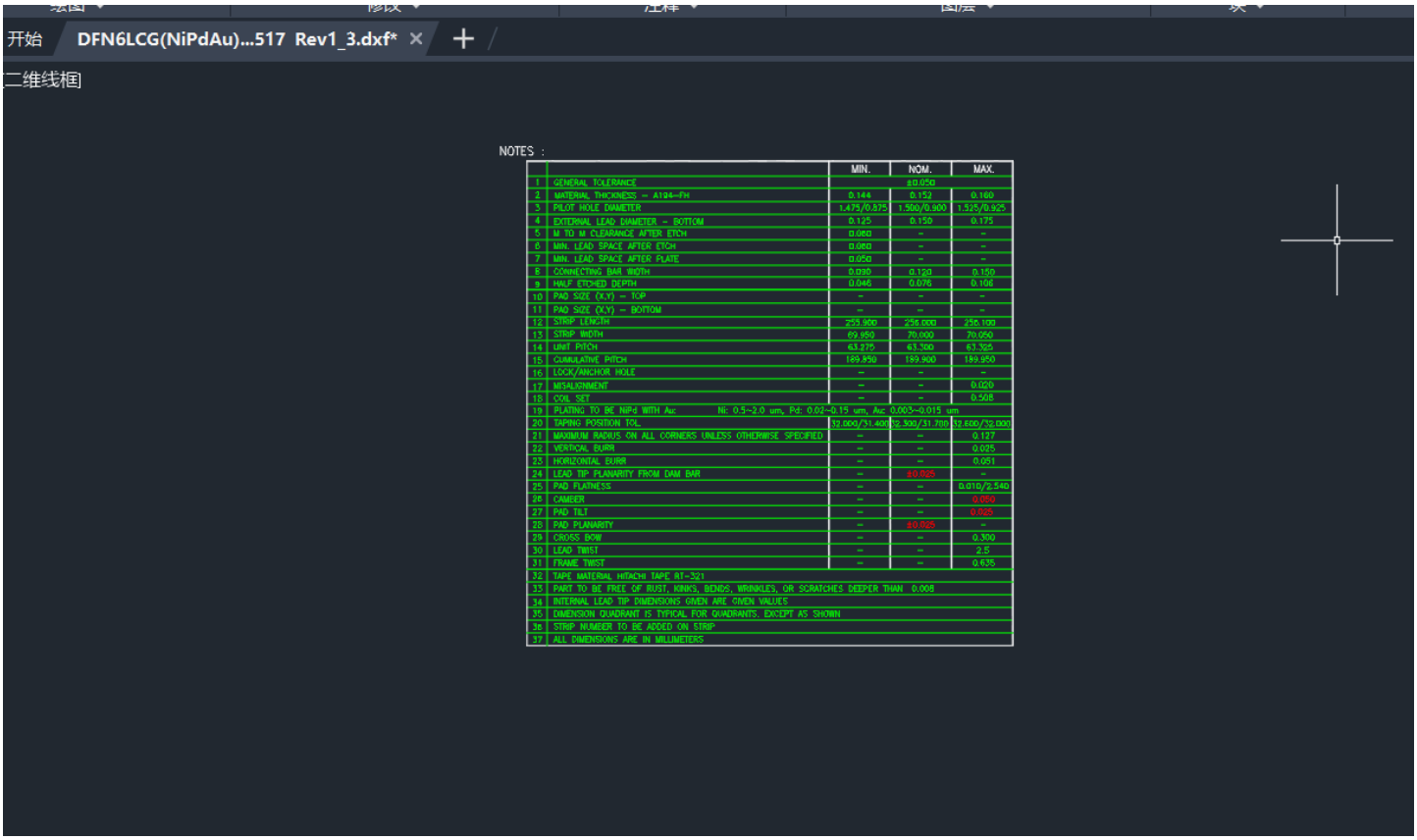
```
Similarity Scores:
/mnt/share/DeepDXF_CGMN/encode/data/test/437_1_2.json /mnt/share/DeepDXF_CGMN/encode/data/test/437_1_2.h5
/mnt/share/DeepDXF_CGMN/encode/data/test/437_1_3.json /mnt/share/DeepDXF_CGMN/encode/data/test/437_1_3.h5
Total Similarity: 0.5980
CGMN Similarity: 0.6353
DeepDXF Similarity: 0.5606
```

```
Similarity Scores:
/mnt/share/DeepDXF_CGMN/encode/data/test/437_1_2.json /mnt/share/DeepDXF_CGMN/encode/data/test/437_1_2.h5
/mnt/share/DeepDXF_CGMN/encode/data/test/439_1_5.json /mnt/share/DeepDXF_CGMN/encode/data/test/439_1_5.h5
Total Similarity: 0.6804
CGMN Similarity: 0.6671
DeepDXF Similarity: 0.6938
```

```
Similarity Scores:
/mnt/share/DeepDXF_CGMN/encode/data/test/437_1_2.json /mnt/share/DeepDXF_CGMN/encode/data/test/437_1_2.h5
/mnt/share/DeepDXF_CGMN/encode/data/test/437_1_2.json /mnt/share/DeepDXF_CGMN/encode/data/test/437_1_2.h5
Total Similarity: 1.0000
CGMN Similarity: 1.0000
DeepDXF Similarity: 1.0000
```


绘图
修改
注释
图层
开始
DFN6BU(NiPdAu)-437 Rev1_3.dxf*
府视][二维线框]


开始
QFN9LM(NiPdAu)-439 Rev1_5.dxf*
府视][二维线框]
256.000±0.100
189.900±0.050
189.900±0.050
63.300±0.025
63.300±0.025
19.000±0.025
63.300±0.025
63.300±0.025
32.353±0.075
33.747±0.075
1.500±0.025
1.500±0.025
3.400±0.300
2.000
2.000
5.000
2.000
0.500±0.500
0.500±0.500
2.000
70.000±0.050
0.152mm QFN9LM 1.1X1.1 ASM
2.600±0.300
40X42X4=6720 UNITS/STRIP (ALL LOGO HALF ETCH)
LEAD FRAME TOP VIEW



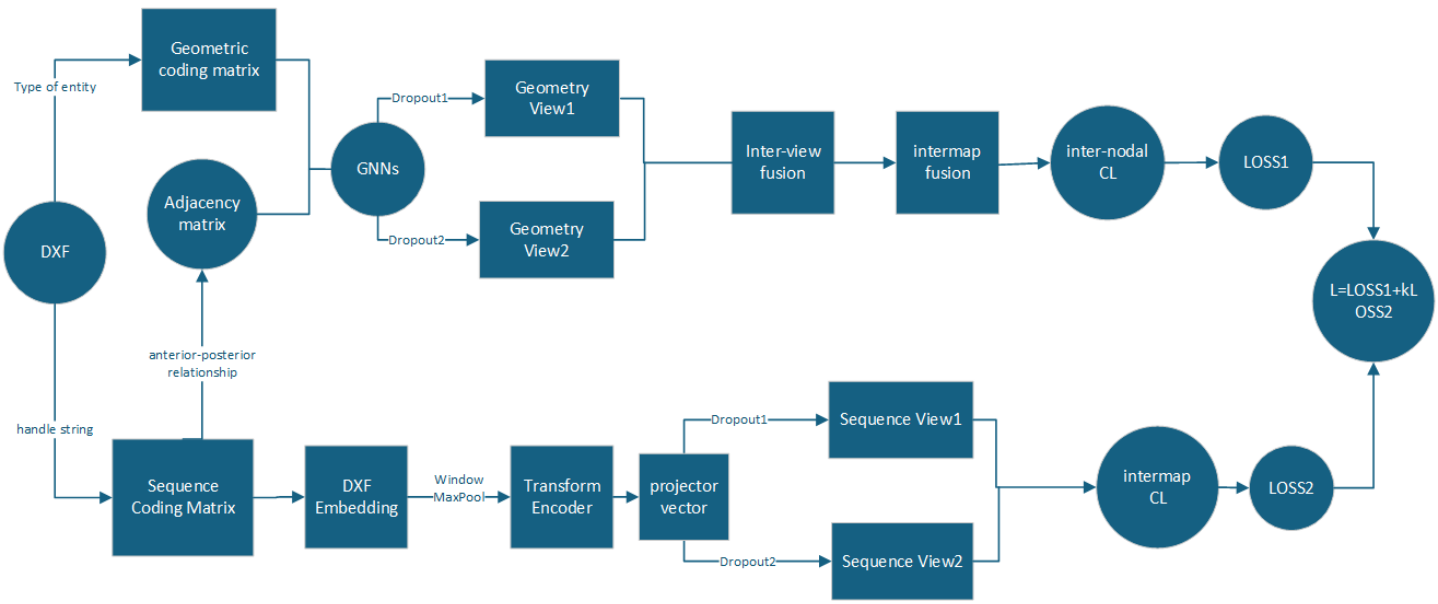
		MIN.	NOM.	MAX.
1	GENERAL TOLERANCE		±0.050	
2	MATERIAL THICKNESS - A194-FH	0.144	0.152	0.160
3	PILOT HOLE DIAMETER	1.475/0.875	1.500/0.900	1.525/0.925
4	EXTERNAL LEAD DIAMETER - BOTTOM	0.125	0.150	0.175
5	M TO M CLEARANCE AFTER ETCH	0.080	-	-
6	MIN. LEAD SPACE AFTER ETCH	0.080	-	-
7	MIN. LEAD SPACE AFTER PLATE	0.050	-	-
8	CONNECTING BAR WIDTH	0.090	0.120	0.150
9	HALF ETCHED DEPTH	0.046	0.076	0.106
10	PAD SIZE (X,Y) - TOP	-	-	-
11	PAD SIZE (X,Y) - BOTTOM	-	-	-
12	STRIP LENGTH	255.900	256.000	256.100
13	STRIP WIDTH	69.950	70.000	70.050
14	UNIT PITCH	63.275	63.500	63.325
15	CUMULATIVE PITCH	189.950	189.900	189.950
16	LOCK/ANCHOR HOLE	-	-	
17	MISALIGNMENT	-	-	0.020
18	COIL SET	-	-	0.508
19	PLATING TO BE NiPd WITH Au	Ni: 0.5~2.0 um, Pd: 0.02~0.15 um, Au: 0.003~0.015 um		
20	TAPING POSITION TOL	12.000/31.400	12.300/31.700	12.600/32.000
21	MAXIMUM RADIUS ON ALL CORNERS UNLESS OTHERWISE SPECIFIED	-	-	0.127
22	VERTICAL BURR	-	-	0.025
23	HORIZONTAL BURR	-	-	0.051
24	LEAD TIP PLANARITY FROM DAM BAR	-	±0.025	-
25	PAD FLATNESS	-	-	0.010/2.540
26	CAMEER	-	-	0.050
27	PAD TILT	-	-	0.025
28	PAD PLANARITY	-	±0.025	-
29	CROSS BOW	-	-	0.300
30	LEAD TWIST	-	-	2.5
31	FRAME TWIST	-	-	0.635
32	TAPE MATERIAL HITACHI TAPE AT-321			
33	PART TO BE FREE OF RUST, KINKS, BENDS, WRINKLES, OR SCRATCHES DEEPER THAN 0.008			
34	INTERNAL LEAD TIP DIMENSIONS GIVEN ARE GIVEN VALUES			
35	DIMENSION QUADRANT IS TYPICAL FOR QUADRANTS, EXCEPT AS SHOWN			
36	STRIP NUMBER TO BE ADDED ON STRIP			
37	ALL DIMENSIONS ARE IN MILLIMETERS			

讨论

CGMN样本需不需随机加强?

网络结构

CGMN_DeepDXF



DeepCAD（2021 ICCV CCF-A）

:[2105.09492] DeepCAD：计算机辅助设计模型的深度生成网络 --- [2105.09492] DeepCAD: A Deep Generative Network for Computer-Aided Design Models (arxiv.org)

https://arxiv.org/html/2404.01645?_immersive_translate_auto_translate=1

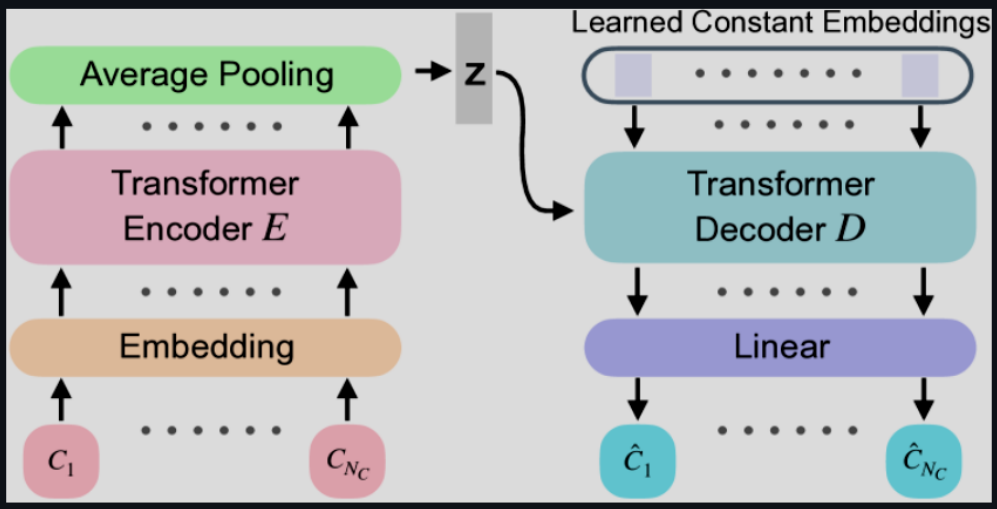


Figure 3: **Our network architecture.** The input CAD model, represented as a command sequence $M = \{C_i\}_{i=1}^{N_c}$ is first projected to an embedding space and then fed to the encoder E resulting in a latent vector z. The decoder D takes learned constant embeddings as input, and also attends to the latent vector z. It then outputs the predicted command sequence $\hat{M} = \{\hat{C}_i\}_{i=1}^{N_c}$..

图 3：**我们的网络架构。**输入 CAD 模型，表示为命令序列 $M = \{C_i\}_{i=1}^{N_c}$ 首先投影到嵌入空间，然后馈送到编码器 E 产生潜在向量 z。解码器 D 将学习到的常量嵌入作为输入，并且还关注潜在向量 z。然后输出预测的命令序列 $\hat{M} = \{\hat{C}_i\}_{i=1}^{N_c}$..

ContrastCAD（2024）

ContrastCAD: Contrastive Learning-based Representation Learning for Computer-Aided Design Models (arxiv.org)，202404 IEEE ACCESS

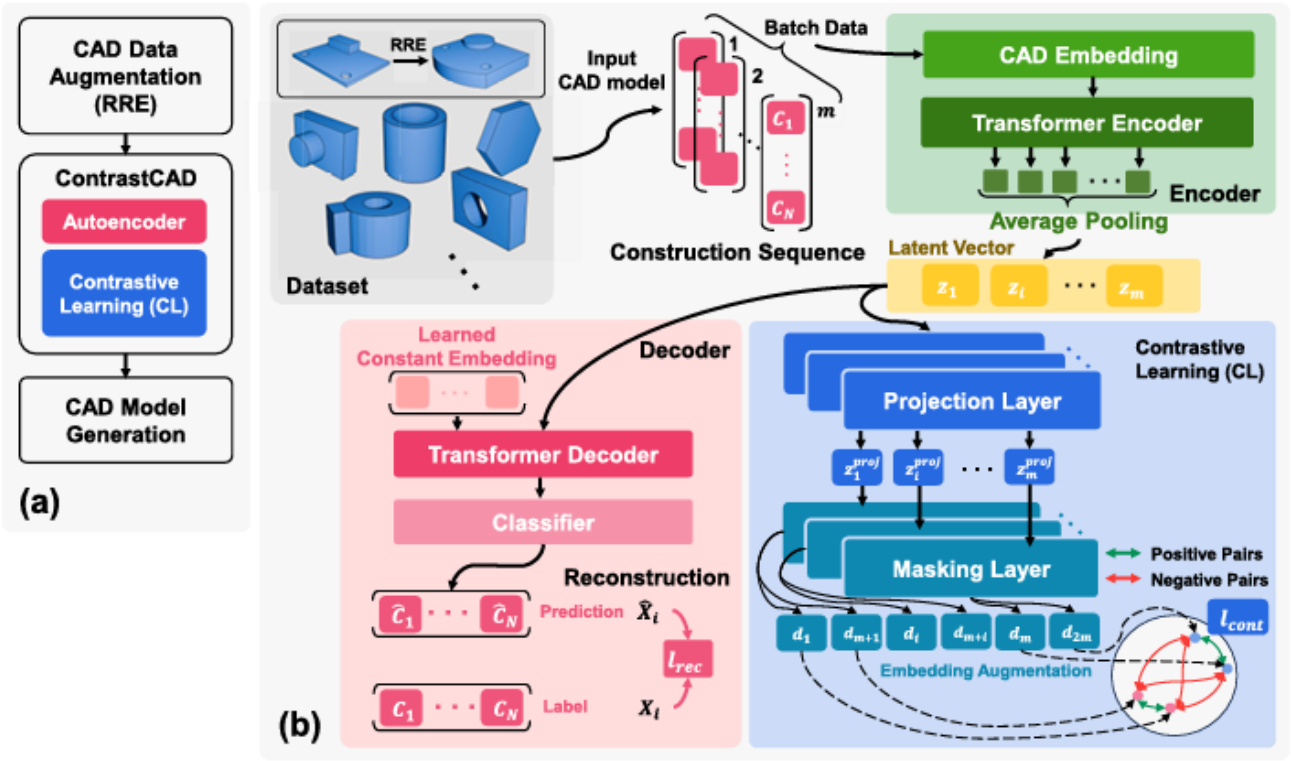
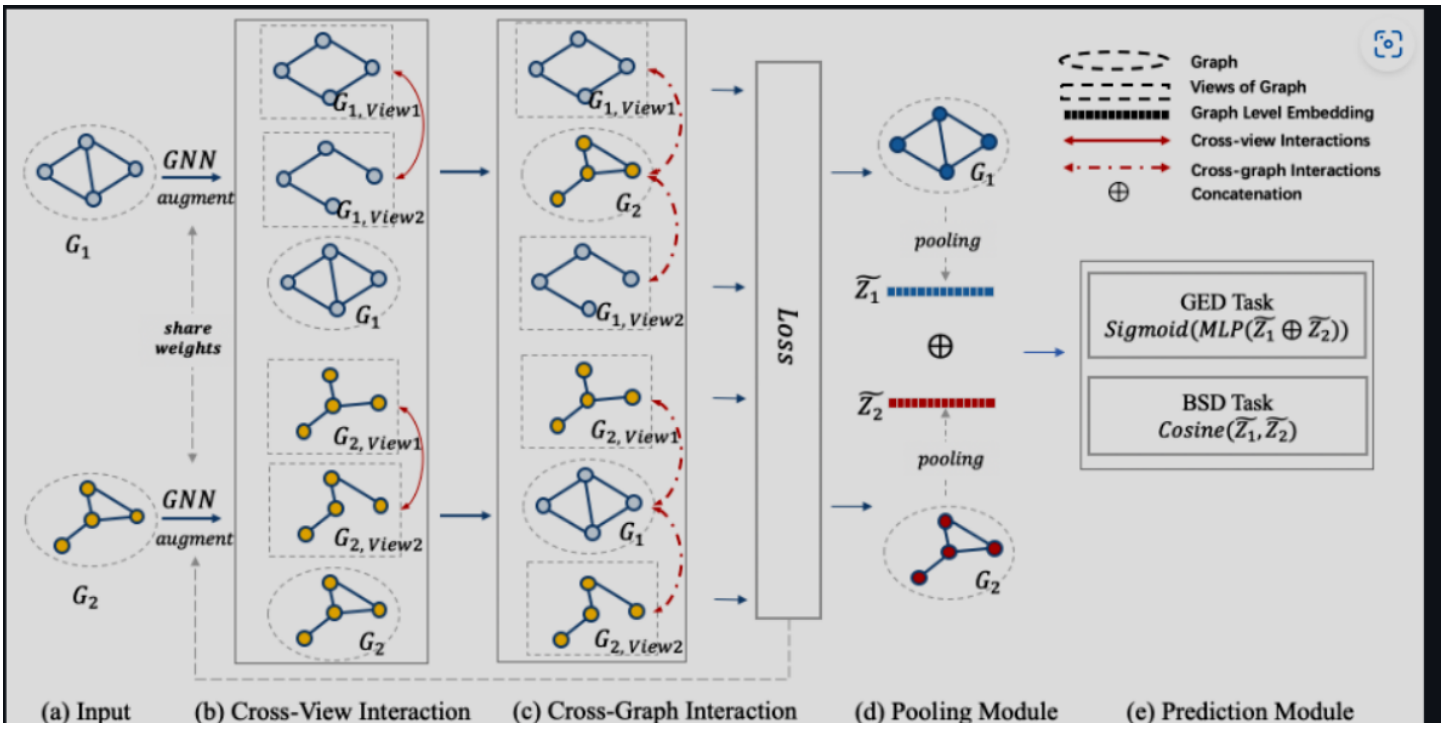


FIGURE 3. (a) Overview of the proposed CAD model learning and generation method and (b) proposed ContrastCAD model based on contrastive learning.

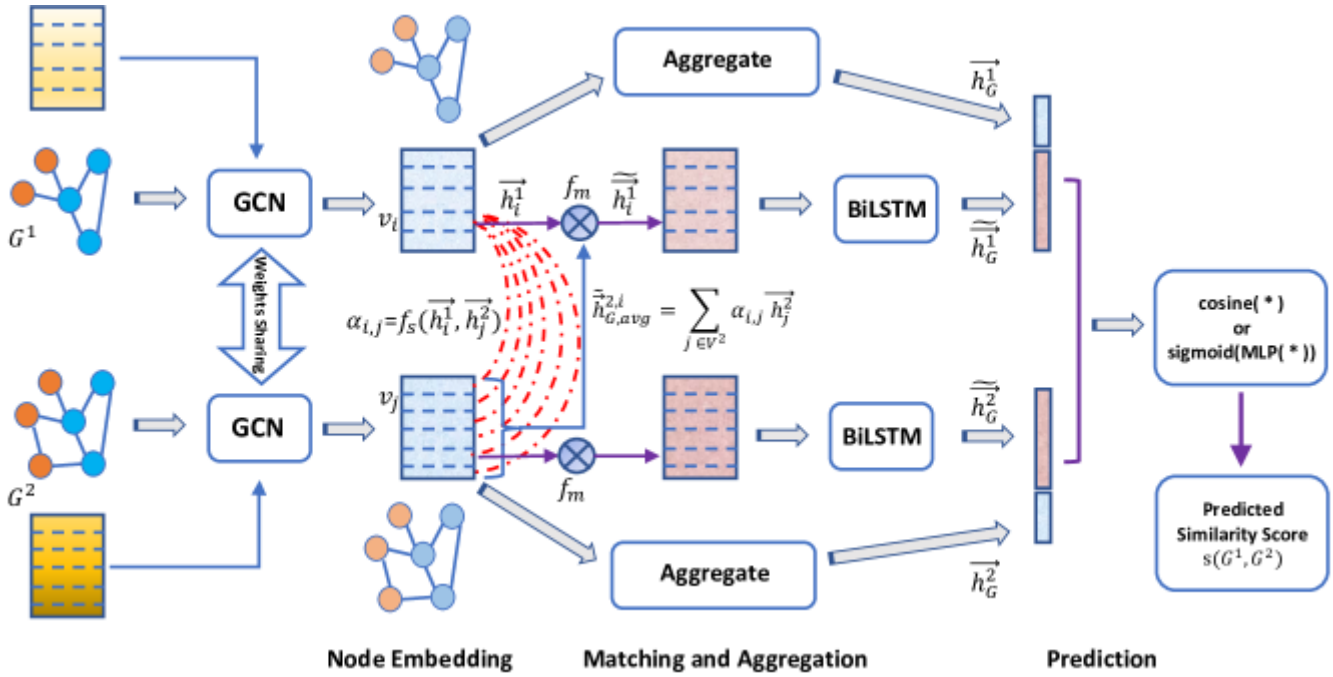
CGMN（CCF-A 2022）

https://ar5iv.labs.arxiv.org/html/2205.15083?_immersive_translate_auto_translate=1



MGMN（CCF-B 2021）

https://ar5iv.labs.arxiv.org/html/2007.04395?_immersive_translate_auto_translate=1



后期想法

实验

优化

transformer

transformer的层数

底层int代替浮点数的计算，节省内存（2？）

DuoAttention

transformer的通用框架

对比损失的方法

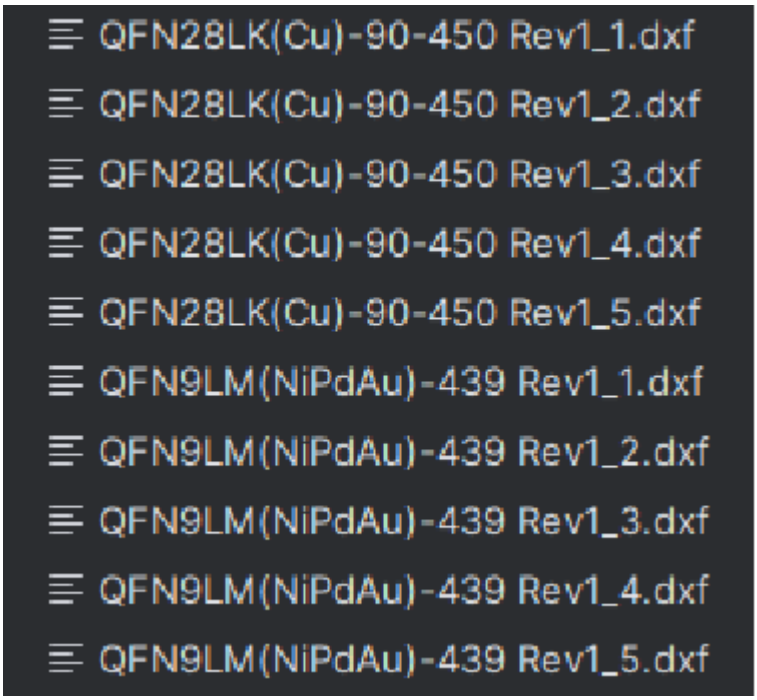
更先进的对比学习技术,如MoCo或BYOL

汇报结果

数据

1.对新的数据和老的数据进行汇总，整理，看看有没有新的实体，实体的数量分布情况，每个实体长什么样，及其属性是怎么提取的（用代码看看）

2.对数据进行分离，得到我们需要的数据，如果代码不能实现的话，用AutoCAD手动



	1月1日	1月2日	1月3日	1月4日	1月5日	2月1日
1月1日	1	0.995	0.422	0.995	0.45	0.99
1月2日		1	0.449	0.999	0.479	0.98
1月3日			1	0.429	0.985	0.41
1月4日				1	0.475	0.9
1月5日					1	0.44
2月1日						
2月2日						
2月3日						
2月4日						
2月5日						

3.模型训练出来看看效果

4.对Dimention标注数字也提取为特征，不能直接用ezdxf，需要二次转化

5.把属性提取代码给cl,让他CGMN也按照我这种提取方式，并有所改进（增加更多属性，包括二次的转化）

6.文本的内容后序继续优化，把文字部分提取出来（后面单独比较也是可以的，数组？）

7.相关的transformer的优化后序再来

8.重新设计一张芯片图看看

9对DeepDXF直接设置为2048试一试

模型

CGMN更加精细的处理

做法

对CGMN模型中，分区考虑，把一张图按照上下左右的顺序分为若干区域，在CGMN需要的json文件中，原本一行表示一个dxf文件，现在一行表示一个区域的某一类实体，节点就是具体的实体，建边方式为每个实体与每个实体是否是否有重叠作为建立边的方式，特征矩阵就是具体每个实体的位置等属性，最终CGMN期望训练出的结果能够区分不同区域，不同实体类的特征，我们将一份DXF文件得到的所有区域拼接在一起作为特征向量来比较相似度

	NUM	Attribute				
LINE		(a, b)		(c, d)		0
CIRCLE		(a, b)		radius	0	0
ARC					start_angle	end_angle
TEXT		(a, b)		height	width	0
MTEXT						0

area 1						area 2					
NUM	Attribute					NUM	Attribute				
	(a, b)		(c, d)		0		(a, b)		(c, d)		0
	(a, b)		radius	0	0		(a, b)		radius	0	0
				start_angle	end_angle					start_angle	end_angle
	(a, b)		height	width	0		(a, b)		height	width	0
					0						0

优点

1.在CGMN模块中具体到实体的编码

2.和CGMN模型的数据较为一致，节点数量可变，但有限制最大值

改进

1.数据的处理方式，可以按我这种方式处理，顺便让他验证下结果对不对，有没有问题，这样就把坐标数据，弧度数据等全部转化为整数，刚好和题目对应（好好研究下芯片的设计）

2.dxf_CGMN_process.py需要处理，从dxf->不同类，不同区域的json，而不是所有dxf->一份json