报告和代码整体简要说明

本次深度学习我选择任务 B，使用的框架是 jittor，基于 pygmtools 版本 0.3.3。后续的数据集搭建、训练、测试都是在这个版本上进行的。

本次报告将会从前期准备（主要包括开源工具、论文、开源代码的学习），深度学习框架实际编写（主要包括任务 B 的全部实现流程），实验和数据（主要包括调参、消融实验）三个方面进行介绍。

代码中的 `main.py` 是完整的训练流程，欲测试代码的完整性，可以运行 `python main.py --debug`, 将会进行一个 epoch 的训练，并保存训练的日志。默认参数设置在 `utils/parser.py` 文件中，其余关键文件全部在根目录下。

除此之外，在线例子的复现结果以 `jupyter notebook` 的形式保存在了 `tutorial` 文件夹下，谨供助教按需查阅。

1 框架搭建前期准备

在开始项目前，先阅读 pygmtools 的官方文档，了解整个项目整体的框架和需要使用的数据集、相关代码接口等等，以便对整个项目框架搭建有个基本的规划。

python 开发环境配置这里不再赘述。

1.1 文档学习和图匹配算法选取

任务要从了解什么是图匹配和方法选取开始，pygmtools 的文档给出了较为详尽的图匹配应用和图匹配流程。整体上，首先要从数据集中提取图的结点、边的特征，这一部分可以利用数据集中已经提供的信息；之后计算亲和度矩阵，并用图匹配求解器来解决问题，这一部分需要学习使用 pygmtools 中的工具。

本次任务我使用 PCA-GM 算法进行图匹配，整个神经网络会内嵌一个 PCA-GM 神经网络，并把它当做一个黑盒来处理。对于每对需要匹配的图片，PCA-GM 方法会先用特征提取器（例如 VGG16）得到带有特征的图片对，之后通过数个隐层，在隐层中计算图卷积和交叉卷积，然后计算亲和度，并通过 Sinkhorn 函数得到双随机矩阵，最终得到图匹配结果，用于后续的损失函数计算，损失函数使用排列损失即可。

1.2 开源代码复现

在了解了整个项目的关键点后，对开源代码进行复现，了解现有的数据集和代码 API，以便后续开发。

在线例子中提供了一个基于预训练模型对两张图片进行匹配的流程展示。开源代码的复现不需要做任何的改动，由于使用的是预训练模型的结果，也不需要 gpu 资源，直接在本地配置环境并运行即可。最关键的是通过复现流程理解 GMNet 需要传入什么样的参数，数据集中哪些信息是已经有了的，哪些信息是需要通过 pygmtools 的工具得到的，

willow 数据集只能提供图片和关键点信息，这些数据是不足以支持我们直接使用 PCA-GM 的。在线例子使用了 `delaunay_triangulation` 的方法，将图片中的关键点进行三角剖分，把图片划分成一系列的三角形区域，以便后续的图匹配网络调用。之后，在线例子提供了各种用于图匹配的神经网络算法，这些算法所使用的神经网络接口略有差异，需要我们对数据集中提供的关键点做一定的计算，之后再传入要使用的 GraphMatching

网络，以提供一个端到端的训练流程。在线例子最后提供了一个很规范的 GMNet，我的网络则是在此基础上修改（在线例子不支持带 batch 的数据，下文会详述）。至此，我已经对接下来的开发工作有了必要的准备。

2 完整深度学习框架搭建

这是任务 B 的主要部分，我将按照实际完成作业的顺序，逐步介绍我对开源代码的理解和使用，遇到的 bug 及修复、处理或避开的方法。

2.1 数据集预处理

基础的 WillowObject 数据集中只提供了图片和关键点信息，并不完全适配图匹配任务。且其数据集中五个类别的图片数量不均衡，也会对匹配精度造成一定的影响。图匹配还需要在各个类别内成对拿出用于匹配的图片，并枚举同一类内用于训练的所有图片的组合。所幸，所有这些功能都已经在 pygmtools 中实现了。

为了能够使用 pygmtools 中 benchmark 提供的接口，在现有的 WillowObject 的基础上还需要一些预处理。这一预处理工作只需要新建一个 `pygmtools.dataset.WillowObject` 对象，调用其 `process` 函数即可。这个预处理会成一个 json 文件，包括训练和测试集的划分，为了保证训练集的平衡，每一类选出 20 张图片用于训练，其余的图片则留做测试，测试集各类图片是不平衡的。关键的信息会保存在 `data-(256,256).json` 中，用于后续的数据集加载。

2.2 图匹配数据集实现

我实现的数据集 `GraphPair` 继承自 jittor 的 `Dataset` 类，其中会内嵌一个 pygmtools 的 `Benchmark` 对象。在数据集初始化时，除了配置好各种参数外，还会调用 `load_data_list` 函数，通过调用内嵌的 `Benchmark` 对象的 `get_id_combination` 函数，得到所有的图片组合的列表。由于要匹配所有的类，该列表内保存了 5 个类各自的图片组合的列表，在将各个图片对进行类别标记后，再把列表展开，逐个 `append` 到数据集的 `data` 属性中，就完成了数据集的初始化。

数据集的 `getitem` 方法则比较关键。对于传入的 index，直接通过数据集初始化时生成的 `data` 列表提取相应的图片对名称，然后调用内嵌的 `Benchmark` 对象的 `get_data` 函数，得到相应的图片、关键点信息等数据。注意，在生成训练集时，需要将 `shuffle` 参数设置为 `True`，这样才能保证关键点之间的对应关系具有随机性，否则得到的匹配矩阵的 groundtruth 对角线元素会全部为 1，这样虽然仍然可以正常训练，但是模型会被引导着向预测单位矩阵的方向进行，就失去训练的意义了。此外，得到的图片是无法直接使用的，需要先进行 `permute` 操作，再归一化到 `[0,1]` 区间。通过 `delaunay_triangulation` 方法处理关键点信息。这里由于 jittor 的 `dataset` 并不支持 scipy 的 `coo_matrix`，所以我把以稀疏矩阵的形式保存的 groundtruth 通过 `toarray` 方法转化成 `numpy` 的 `ndarray`。在进行了上述所有的处理之后，把需要的参数全部返回，以供后续的训练即可。

此外，为了加快数据集加载的速度，避免每次 `getitem` 都要重新读取图片、关键点等信息，并进行计算，我还实现了 `load_data` 方法，如果调用此方法，在数据集初始化时，就会一次性地进行上述所有的数据处理，并把他们存储到 `data` 列表中。实测这一方法在 cpu 环境运行时可以实现一次性加载，但是在 cuda 环境中运行时，会超过内存限制。考虑到这一部分数据集处理并不是整个模型训练开销的瓶颈所在（开销主要在训练上），所以我后续没有再使用这一方法。

2.3 网络搭建和训练

本次作业训练使用的网络是根据 pygmtools 在线样例中的网络进行修改得到的。由于在线例子只用了一对图片，等价于 batchsize 是 1，其网络结构的实现在提取节点信息时只支持 batchsize 等于 1 的数据，这一部分我将其修改成对于整个 batch 的数据，循环遍历提取节点信息，将信息整合到 `ndarray` 中，这样得到的节点信息的第一个维度仍然等于 batchsize，以保证整个训练流程的一贯性。

整个图匹配网络内置一个 CNN，这个 CNN 输出得到局部和全局的特征后，再通过上采样得到 4D 的 featuremap，形为 $(batchsize * 1024 * 256 * 256)$，接下来关键在于用 keypoint 的位置来从这些 map 里面提取信息。keypoint 本身是浮点数，需要将其进行 round 得到整数, 之后特征提取的方法如图所示1。

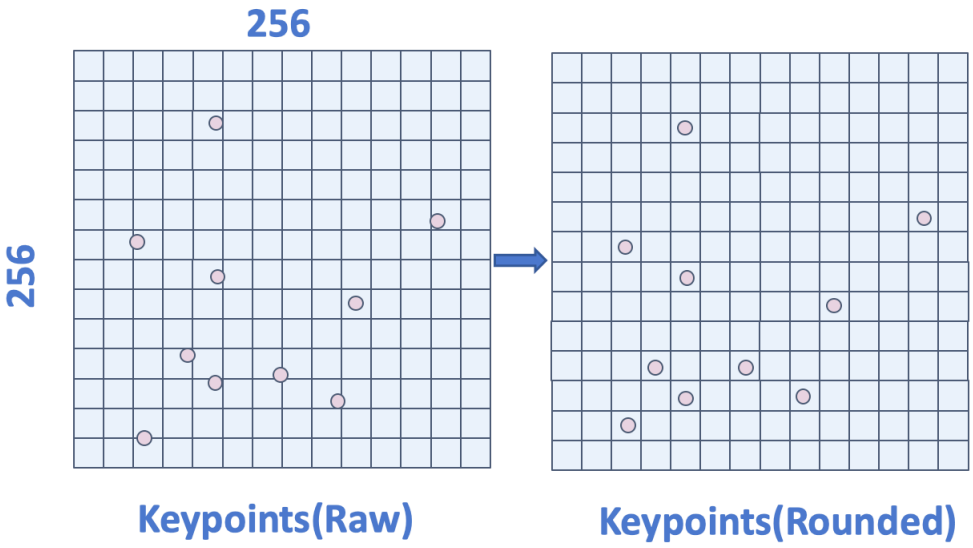

图 1: 使用 keypoint 提取 feature 信息

上采样之后的 featuremap 后两个维度正是 resize 之后的图像大小，助教曾经在群里指出，需要交换在线例子中使用的 roundedkeypoints 下标。这一交换至关重要。在我实际进行训练的过程中，在交换之前，无论怎么调参和改进，评估环节得到的 precision、recall 等等参数都在 0.5 到 0.7 之间，我一度认为这就是标准的训练结果。后来经助教指出，在交换之后，才能把各个评估指标提升到 0.8 以上。这一交换的原因是，keypoint 标记了一张原始图片在 x、y 两个方向上的关键点位置信息，如果 xy 轴坐标弄反了，实际上提取到的根本就不是特征点的信息，而是没有意义的信息，这样引导模型训练的方向会发生错误，无论多少轮的训练，都没办法提高精度。

在训练 loss 的选取上，直接使用 pygmtools 提供的 `permutation_loss`，这一 loss 也是 PCA-GM 这一图匹配算法所使用的，pygmtools 提供的工具已经支持 batch 的数据，这里直接将模型的预测结果和 groundtruth 做 loss 即可。在 evaluate 时，则需要把模型的预测结果通过 `hungarian` 函数转换成可解释的预测结果，以便后续调用 benckmark 的 `eval` 函数进行评估。

2.4 测试和评估

我们要使用 benchmark 的 `eval` 函数来进行评估，但是 eval 函数并不需要传入 groundtruth。就能帮我们完成 evaluate 的过程。其方法是（在 0.3.3 版本的 pygmtools 中），当我们最开始调用 getdata 拿取数据的

时候，如果是测试集，则会在根据当前的进程 pid 生成缓存路径，并把 groundtruch 以 numpy 数组的形式存储下来，这样 evaluate 时直接在本地读取即可。

如果我们不知道 pygmtools 这样的实现原理，有可能会出现错误，如果大家进行 evaluate 时，发现第一次测试的 evaluate 的结果比较高，但是从第二次开始 precision 和 recall 等等参数都稳定在 0.1 左右，无论怎么训练都是这个数值，应该是因为在 getdata 时启用了 shuffle，这样的问题在于 label 只会缓存一次，后续的测试都是用第一次测试 getdata 所拿到的 random 之后的 label 进行的，但是后续的 groundtruth 仍然在进行 shuffle，得到的结果几乎不可能与第一次一样，造成 evaluate 失败。我实现的解决的方法有两个，一种是直接在生成数据集时候就一次性的把所有 data 全 load 进来，之后不再调用 getdata 函数，另一种则比较简便，测试集 getdata 时候不 shuffle 即可，这里比较推荐后面一种。

```
1    self.benchmark.get_data(self.data[index][0], shuffle=self.shuffle)
```

```
1    if shuffle:
2        random.shuffle(obj_dict['kpts'])
```

```
1    if self.sets == 'test':
2        for pair in id_combination:
3            id_pair = (ids[pair[0]], ids[pair[1]])
4            gt_path = os.path.join(self.gt_cache_path, str(id_pair) + '.npy')
5            if not os.path.exists(gt_path):
6                np.save(gt_path, perm_mat_dict[pair])
```

3 实验结果和总结