掌纹识别项目说明文档

18373263 贺梓淇

代码结构

本代码共有五个文件，它们各自的用途如下：

- `DiffModel.py` 包括了模型定义、数据集定义以及模型的使用相关函数
- `Expander.py` 负责将数据集随机化、起到了扩展数据集的功能
- `Loader.py` 负责了各类数据集的预处理
- `Trainer.py` 负责模型训练
- `use_example.py` 是使用本模型的样例代码

数据预处理

1. 作为源数据集的 BMP600 按照 5 : 1 的比例分割为训练集和测试集
2. 由于给到的数据集很小，为了防止过拟合、增加模型的鲁棒性，对于测试集中的每一个样本，都应用随机化仿射复制了 9 份带有一定缩放和旋转的新样本
3. 为了提取图片中的特征，使用了 `PyTorch` 官方提供的 `resnet50` 对每个样本先进行了一遍预处理，变为了 1000 维的向量

构建模型

本模型的设计为：输入两个预处理后的张量，判断它们是否出自同一个人

主要的实现方法是简单的 `FC` 层加上带有可学习参数的 `BatchNorm` 层交叉而成，输入向量将被直接拼接为 `[BatchNum, 2000]` 的向量输入模型

```python
class DiffModel(nn.Module):
    def __init__(self):
        # initing model
        self.version = "v1"
        super(DiffModel, self).__init__()
        # set layers
        self.input = nn.Linear(2000, 1000)
        self.bn1 = nn.BatchNorm1d(1000)
        self.fc1 = nn.Linear(1000, 400)
        self.bn2 = nn.BatchNorm1d(400)
        self.fc2 = nn.Linear(400, 100)
        self.bn3 = nn.BatchNorm1d(100)
        self.fc3 = nn.Linear(100, 40)
        self.bn4 = nn.BatchNorm1d(40)
        self.output = nn.Linear(40, 1)

    def forward(self, x1: torch.Tensor, x2: torch.Tensor):
        assert(self.version == "v1")

        x = torch.cat([x1, x2], dim=1)
        x = self.input(x)
        x = func.relu(self.fc1(self.bn1(x)))
        x = func.relu(self.fc2(self.bn2(x)))
        x = func.relu(self.fc3(self.bn3(x)))
        return torch.sigmoid(self.output(self.bn4(x)))
```

构建数据集

由于训练集中随机挑选样本对时会出现相同样本数远远小于不同的样本数，这种不平衡性容易使模型快速想输出恒为 `0` 的错误方向收敛。

为了使数据集中的采样是均匀而平衡的，我设计了一个数据集，使得从中采样得到的不同类别的样本数量的比例是可控的。在训练环境下，这个比例是 `1:1`

```python
class DiffModelDataset(torch.utils.data.Dataset):
    def __init__(self, ds, n_p_ratio=1, extend=1):
        """
            ds: size=(n_class, n_sample, len(data)), ds[i] represents all data in class i
            n_p_ratio: n_negative / n_positive, integer
        """
        self.ds = ds
        self.num_classes = len(ds)
        self.n_p_ratio = n_p_ratio
        self.extend = extend
        self.length = len(ds) * len(ds[0]) * (1 + self.n_p_ratio)
        # used to go through and generate data
        data_indexs = []
        for c in range(len(ds)):
            for i in range(len(ds[c])):
                data_indexs.append([c, i])
        self.data_idx = np.array(data_indexs)
        self.data_loaded = 0
        self.waitlist = []

    def __len__(self):
        return self.length * self.extend

    def __getitem__(self, idx):
        with torch.no_grad():
            # class_idx and sample_idx of x1 is specified by idx
            c1, i1 = self.data_idx[(idx % self.length) // (self.n_p_ratio + 1)]
            # class_idx of x2 is specified by idx and random number
            c2 = c1
            if idx % (self.n_p_ratio + 1) != 0:
                y = torch.tensor([0], dtype=torch.float, requires_grad=False)
                n_try = 0
                while c2 == c1 and n_try < MAX_TRY:
                    c2 = np.random.randint(self.num_classes)
            else:
                y = torch.tensor([1], dtype=torch.float, requires_grad=False)
            # sample_idx of x2 is a random number
            i2 = np.random.randint(len(self.ds[c2]))
            # get x1, x2 as tensors
            x1 = torch.tensor(self.ds[c1][i1], dtype=torch.float)
            x2 = torch.tensor(self.ds[c2][i2], dtype=torch.float)
        return x1, x2, y
```

训练以及调参

对模型进行训练以及调参的过程较为漫长，以下仅说明找到的效果较好的结果

ITEM	CHOICE
损失函数	Binary Cross Entropy Loss
优化器	SGD with momentum (0.8)
学习率	0.01 (multiply by 0.4 every 20 steps)

实际使用

在判断一张图片是否处于训练集中的某类样本的过程如下：

1. 将图片经过预处理转换为 `1 * 1000` 的张量
2. 将图片与所有训练集中的样本进行比较（按照不同先后顺序输入模型中）
3. 在类中平均图片与样本的输出结果，取置信度最大的类为输出
4. 如果该解的置信度小于阈值则判断为拒绝

训练结果

目前表现最好的模型为 `models\results\diff_model_v1_11-10_08-13.pkl`

- 如果对置信度的阈值不做限制的情况下，准确率为 `96%`
- 如果置信度的阈值取能让所有非拒绝输出都是正确解的情况下，准确率为 `89%`

详细测试数据见 `models\results\diff_model_v1_11-10_08-13_info.txt`

存在的问题以及可能的改善方法

使用时速度较慢

目前的代码在判断类别时速度较慢，需要数秒才能得到结果

这个问题出现的主要原因在于每次使用时，输入会与训练集中的 5000 个样本逐个比对，双层 `for` 循环并在每个循环间创建张量的时间代价较高，解决方法可能有以下几种：

- 利用随机采样，在每个类别中随机采样计算均值，而非与所有样本比对（这样可能牺牲部分准确率和稳定性）
- 利用并行计算替代 `for` 循环嵌套，将逐个比对替换成基于 `mini_batch` 的比对

###