运行环境

Docker 软件环境:　FROM anibali/pytorch:cuda-9.0

训练/测试数据：　只使用了官方提供的人脸特征文件

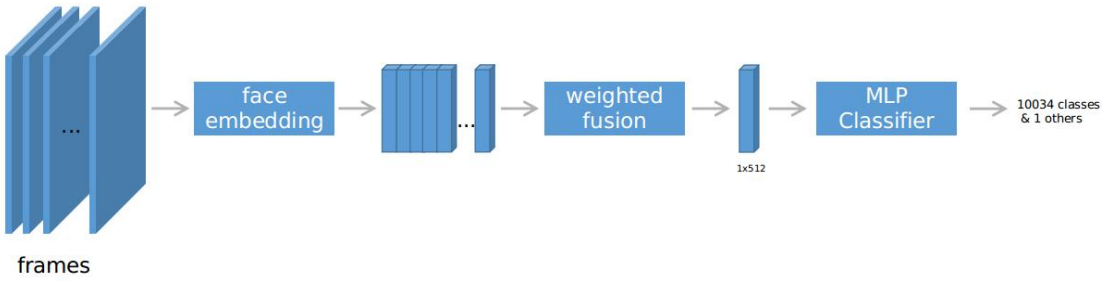
单个模型训练时间：约 30 分钟（训练+验证集，GTX1080）

单个模型前向传播时间：20 秒　（测试集）

测试总耗时：约 30 分钟　（测试集，5 类模型，使用全部训练权重，共 31 次前传，包括数据读写）

算法流程

算法的总体流程是使用官方提供的人脸特征向量，利用人脸特征向量的几何特性，以质量评价指标和检测得分为权重，加权融合一个视频中所有的人脸特征，得到整个视频的人脸识别特征向量。接着对融合后视频的特征向量进行分类，分类为 10035 个类别（10034 个人物类和 1 个其他类），用分类得分大小顺序作为相应类别的检索结果。



视频人脸特征向量融合

我们假定，对一个视频中的所有人脸向量，可以按以下方式融合成整个视频的特征向量：
令融合前每帧人脸特征向量 X_1，X_2，X_3 ... X_n, 融合后特征向量 X_{all}，人脸特征向量对应的质量评价指标为 Q_i ，对应的检测得分为 D_i，权重为 a_i

$$X_{all} = \sum_{1}^{n} \alpha_i Q_i D_i X_i$$

经过对数据集的分析和实验测试，我们选择了如下方式计算 a_i，当质量评价小于 0 时，该特征对应权重 a 为 0，当质量评价在 0-20 之间，a=0.2/sum(a_i)，当质量评价在 20-30 之间，a=0.3/sum(a_i)，当质量评价在 30-40 之间，a=0.6/sum(a_i)，当质量评价指标大于 40，a=1/sum(a_i)，源码如下图所示：

```
for video_ind, video_name in enumerate(face_feats_dict_val):
    all_weight = 0
    weighted_feats = [0]*512
    face_feats = face_feats_dict_val[video_name]
    for ind, face_feat in enumerate(face_feats):
        [frame_str, bbox, det_score, quality_score, feat] = face_feat
        [x1, y1, x2, y2] = bbox
        if quality_score<0:
            quality_score*=0.0
        elif quality_score<20:
            quality_score*=0.2
        elif quality_score<30:
            quality_score*=0.3
        elif quality_score<40:
            quality_score*=0.6
        else:
            pass
        weight = quality_score*det_score
        all_weight += weight
        weighted_feats += feat*weight
    weighted_feats = np.array(weighted_feats, dtype='float32')/all_weight
    new_val_dict[video_name.decode(encoding="utf-8")] = weighted_feats

del face_feats_dict_val
gc.collect()
```

MLP 分类器

我们选用了多层感知机作为分类器，最终提交结果我们选用了五种不同的全连接模型，源码分别如下：

```
class swish(nn.Module):
    def __init__(self):
        super(swish, self).__init__()
        self.sigmoid = nn.Sigmoid()
    def forward(self, x):
        return x*self.sigmoid(x)

#2-layer-sigmoid
model_conv = nn.Sequential(nn.Linear(in_features=512, out_features=1024, bias=True), nn.Sigmoid(), nn.BatchNorm1d(1024),
                                    nn.Dropout(0.25),  nn.Linear(in_features=1024, out_features=10035, bias=True))

#3-layer-sigmoid
model_conv = nn.Sequential(nn.Linear(in_features=512, out_features=512, bias=True), nn.Sigmoid(), nn.BatchNorm1d(512), nn.Dropout(0.2),
                           nn.Linear(in_features=512, out_features=1024, bias=True), nn.Sigmoid(), nn.BatchNorm1d(1024), nn.Dropout(0.5),
                           nn.Linear(in_features=1024, out_features=10035, bias=True))

#2-layer-swish
model_conv = nn.Sequential(nn.Linear(in_features=512, out_features=1024, bias=True), swish(), nn.BatchNorm1d(1024),
                                    nn.Dropout(0.25),  nn.Linear(in_features=1024, out_features=10035, bias=True))

#3-layer-swish
model_conv = nn.Sequential(nn.Linear(in_features=512, out_features=768, bias=False), nn.BatchNorm1d(768), swish(), nn.Dropout(0.2),
                           nn.Linear(in_features=768, out_features=768, bias=False), nn.BatchNorm1d(768), swish(), nn.Dropout(0.2),
                           nn.Linear(in_features=768, out_features=10035, bias=True)
                          )

#3-layer-swish-new
model_conv = nn.Sequential(nn.Linear(in_features=512, out_features=1024, bias=True), swish(), nn.Dropout(0.5),
                          nn.Linear(in_features=1024, out_features=1024, bias=True), swish(), nn.Dropout(0.5),
                          nn.Linear(in_features=1024, out_features=10035, bias=True))
```

选择其中一个主力模型为例：

模型接受融合后的视频 512 维特征向量作为输入，使用 swish 激活函数（即 f(x)=x·sigmoid(βx)，我们只使用固定β=1 的 swish 激活做了实验，论文链接 https://arxiv.org/abs/1710.05941），使用 0.5 的 dropout 概率，堆积三层全连接，不采用 BN（实验发现使用常规的 BN-激活会影响效果，不采用 BN 或者 BN 在激活函数之后则无影响），输出 10035 维向量，第 0 维为其他类的概率，1-10034 维是对应类别的概率值。

最后用测试集所有视频对 10034 类（不包括第 0 维其他类）的分类得分从大到小的顺序排序，取前 100 个视频作为相应类别的检索结果。

模型训练

预处理

1. 合并训练集和验证集，把验证集噪声样本标注为第 0 类（其他类）。加入验证集后，把 baseline 模型排行榜得分从 0.8509 提升至 0.8878。

2. 只筛选 det_score >= 0.8 and quality_score>=30 的人脸特征向量用于训练。

3. 使用 StratifiedKFold 分层划分数据集为 5fold

训练策略

1. 对 5fold 的每一个 fold，不使用噪声样本（其他类样本），只使用属于 10034 类人物的样本，使用 focal loss，视为 10034 个分类任务进行预训练，得到预训练模型。之后模型在此基础上加上噪声数据 finetune（此步骤能加快模型收敛，但不能显著提分）

2. 训练时，对每个视频每次遍历时，25%的概率随机抽取其中一个特征向量用于训练，65%的概率抽取其中三个特征向量，加权融合后用于训练，10%的概率随机抽取五个特征向量加权融合。加权方式与算法测试时加权方法类似，使用 det score 和 quality score 的乘积加权，代码如下图：

```
class faceDataset_train(Dataset):
    def __init__(self, name_list, feature_list, train_class_dict={}, is_train=True):
        self.name_list = name_list
        self.feature_list = feature_list
        self.train_class_dict = train_class_dict
        self.is_train = is_train

    def __len__(self):
        return len(self.name_list)

    def __getitem__(self, idx):
        name = self.name_list[idx]
        feats = self.feature_list[idx]
        det_scores = det_score_lists[idx]
        quality_scores = quality_score_lists[idx]

        rand1 = random.randint(0,len(feats)-1)
        rand2 = random.randint(0,len(feats)-1)
        rand3 = random.randint(0,len(feats)-1)
        rand4 = random.randint(0,len(feats)-1)
        rand5 = random.randint(0,len(feats)-1)
        feats1 = feats[rand1]
        feats2 = feats[rand2]
        feats3 = feats[rand3]
        feats4 = feats[rand4]
        feats5 = feats[rand5]
        score1 = quality_scores[rand1]*det_scores[rand1]
        score2 = quality_scores[rand2]*det_scores[rand2]
        score3 = quality_scores[rand3]*det_scores[rand3]
        score4 = quality_scores[rand4]*det_scores[rand4]
        score5 = quality_scores[rand5]*det_scores[rand5]
        feats_3 = (feats1*score1 + feats2*score2 + feats3*score3) / (score1 +score2 + score3)
        feats_5 = (feats1*score1 + feats2*score2 + feats3*score3 + feats4*score4 + feats5*score5) / (score1 + score
        choise = random.random()
        if choise<0.25: # random choose raw data
            feats = feats1
        elif choise<0.9:
            feats = feats_3
        else:
            feats = feats_5

        if self.is_train:
            label = self.train_class_dict[name]
        else:
            label = 0
        label = get_label(label) #one_hot_label
        return feats,label
```

3. 加上验证集及噪声数据进行训练。先使用 0.5×Focal Loss + 0.5×Softmax Loss，使用小学习率 warm up 5-25 个 epoch。接着只使用 Focal Loss 训练到 150 个 epoch，再改用 0.98×Focal Loss + 0.02×Softmax Loss 继续训练。在 200 个 epoch 左右交叉验证得到最佳得分，保存模型。在训练时学习率每 50 个 epoch 下降为一半。默认优化器 Adam，初始学习率 1.5e-4，warm up 学习率 2e-5。

模型融合

单个 fold 模型排行榜得分约 0.88+

单个模型的 5fold 融合，排行榜得分约 0.8955

按同样的训练策略，使用不同的 seed，训练了如算法流程部分所述的五个模型，每个模型均 5fold 交叉验证得到 5 个验证集最高得分的权重。

最终得分最高的结果融合了上述 5 个 MLP 模型，得分 0.8983。

（注：理论上应使用 5 个模型 x5fold=25 个权重，但由于比赛时间限制，没有完全完成训练计划，最终提交时使用了部分更早版本的权重进行了融合作为替代，否则得分仍会进一步提高）

2019.6.17

seefun