# 基于无人机的人体动作识别
# 竞赛算法说明书

| | | | |
|---|---|---|---|
| 队伍名称 | 为什么参数调不队 | | |
| 队长姓名 | 宋泱 | 队员姓名 | 刘子康 梅朗潇 |
| 学校 | 哈尔滨工业大学 | | |

# 一、算法背景

随着无人机技术的不断发展，无人机在各个领域的应用也日益广泛，包括但不限于农业、环境监测、安防等。通过无人机搭载的传感器和摄像头，可以实现对人体行为的实时监测和识别，然而如何设计高效的人体行为识别算法，是当前亟待解决的核心问题之一。

我组采用的算法和方案基于目前较为前沿的成果：TE-GCN 和 HDBN，其中前者在赛事数据集上已有较为不错的基准结果；而后者将目前效果较好的模型进行了整合，分为 GCN 和 Transformer 两个大类，其中 GCN 包括 CTR-GCN、TD-GCN、MST-GCN，同时还针对数据做了不同的处理，如同一骨骼两端关节相减得出骨骼坐标，前后帧相减得出关节或骨骼动作，以此发展出较多模态的训练数据集，最后使用融合方法得出最终的置信度文件。我组算法即在这二者基础上进行融合，谋求较为完整的项目结构。

# 二、算法方案

我组采用的改进方案主要包括数据集上的扩增，以及模型维度改进等相关算法。

## 2.1 数据集上的扩增

### 2.1.1 算法背景

在构建深度学习模型过程中，往往需要大量的数据进行训练，以获得较好的效果，而在数据预处理中，往往会出现数据量不足或质量不高的情况，这时常可以通过数据增广对数据进行预处理。

本次国赛数据集训练集共 16724 个样本，分为 155 个动作类别，则每个类别约有 107 个样本，训练量并不大。而考虑到 155 个类别中有较多“对称”动作，例如 sit down（坐下）和 stand up（站起）、put on a coat（穿外套）和 take off a coat（脱外套）、accelerate（加速）和 decelerate（减速）等，经过分析发现这些数据的预测准确率也较低，因此对于这些对称的样本对，将其中一个样本的动作序列进行“倒放”，生成一个新的样本，并给予样本对中另一个样本的 label，以此实现数据增广，以期达到提升数据集含量、质量等目标。

扩充后的数据集样本数为 20637，测试发现在增加一定 epoch 后模型性能得到了提升。

### 2.1.2 算法目的

数据量不足时，模型可能无法充分学习到数据的特征，因此模型选择和超参数优化的选择空间受限；模型没有足够的数据来学习数据的分布，可能无法很好地泛化到新数据；也可能出现个别类别的样本数量过少、类别不平衡问题。

因此对现有数据进行一定程度的变换，生成新的数据样本，从而扩大数据集的规模和样本的多样性，以提高模型的泛化能力，并防止过拟合现象。数据量增加也可以让模型更好地掌握数据的特征分布，从而提高其在未见数据上的表现。

### 2.1.3 具体实现

首先找出所有“对称”动作的 label 并进行分组，逐个遍历数据集样本，如果存在“对称”动作，就截取有效帧（即非空白帧）部分的动作序列，然后使用数组切片操作进行“倒

放”，扩充至 300 帧后添加到数据集中，同时用对称动作的 label 进行标注，最后得到扩增后的数据集，可用于模型训练。

**代码如下：**

```
import numpy as np


# label 对称组
index = [(3, 4), (6, 7), (11, 12), (13, 14), (15, 16), (34, 35), (38, 39), (49, 50), (51, 52), (84, 85),
         (102, 103), (104, 105), (106, 107), (108, 109), (112, 113), (119, 120), (144, 145), (153, 154)]

# 时间帧倒放
def reverse_frame(data_file, label_file):
    dataset = np.load(data_file)
    labels = np.load(label_file)
    length = dataset.shape[0]
    count = 0
    for data, label in zip(dataset, labels):
        for label1, label2 in index:    # 对称 label
            if label == label1:
                sample = np.zeros((3, 300, 17, 2))
                zero_start = find_zero_start(data, axis=1)
                sample[:, :zero_start, :, :] = data[:, zero_start-1::-1, :, :]  # 仅倒放非零段
                dataset = np.append(dataset, [sample], axis=0)
                labels = np.append(labels, label2)
            elif label == label2:
                sample = np.zeros((3, 300, 17, 2))
                zero_start = find_zero_start(data, axis=1)
                sample[:, :zero_start, :, :] = data[:, zero_start-1::-1, :, :]  # 仅倒放非零段
                dataset = np.append(dataset, [sample], axis=0)
                labels = np.append(labels, label1)
        count += 1
        if(count % 1000 == 0 or count == length):
            print(f'已处理： {(count/length)*100:.2f}%')
    return dataset, labels

# 截取非零段
def find_zero_start(arr, axis=1):
    shape = arr.shape
    for i in range(shape[axis]):
        if np.sum(arr[:, i, :, :]) == 0:
            return i
    return shape[axis]


```

```
# if __name__ == '__main__':
#     sets = ['bone', 'joint', 'bone_motion', 'joint_motion'] #所需模态
#     for set in sets:
#         data_file = f"./data/train_{set}.npy"  # 训练集 data 路径
#         label_file = "./data/train_label.npy"  # 训练集 label 路径
#         data_new, label_new = reverse_frame(data_file, label_file)
#         print(data_new.shape)
#
#         # 3D
#         data_new = np.transpose(data_new, (0, 4, 2, 3, 1))
#         np.savez(f"./data_new/train_{set}.npz", x_train=data_new, y_train=label_new)
#
#         # # 2D
#         # data_new = np.transpose(data_new, (0, 2, 4, 3, 1))
#         # np.savez(f"../data_new/train_{set}_2d.npz", x_train=data_new, y_train=label_new)

if __name__ == '__main__':
    sets = ['joint'] #所需模态
    for set in sets:
        data_file = f"./data/train_{set}.npy"  # 训练集 data 路径
        label_file = "./data/train_label.npy"  # 训练集 label 路径
        data_new, label_new = reverse_frame(data_file, label_file)
        print(data_new.shape)

        # 3D
        data_new = np.transpose(data_new, (0, 4, 2, 3, 1))
        np.savez(f"./data_new/train_{set}.npz", x_train=data_new, y_train=label_new)

        # # 2D
        # data_new = np.transpose(data_new, (0, 2, 4, 3, 1))
        # np.savez(f"../data_new/train_{set}_2d.npz", x_train=data_new, y_train=label_new)
```

## 2.2 模型的维度修改

在深入研究各模型后，我组发现部分模型如 MST-GCN 等，其只能适配 2d 数据集，导致精度可能有损失，我组在考虑对比后在模型中修改参数，使其能够在 3d 数据集上正常运行并训练，以寻求利用 3d 维度的优势。

**代码如下：**

```
class Model(nn.Module):
    def __init__(self, num_class=60, num_point=25, num_person=2, graph=None, graph_args=dict(), in_channels=3,
                 drop_out=0, adaptive=True, num_set=3):
        super(Model, self).__init__()

```

```
        if graph is None:
            raise ValueError()
        else:
            Graph = import_class(graph)
            self.graph = Graph(**graph_args)

        A = np.stack([np.eye(num_point)] * num_set, axis=0)
        self.num_class = num_class
        self.num_point = num_point
        self.data_bn = nn.BatchNorm1d(num_person * in_channels * num_point)

        self.l1 = TCN_GCN_unit(in_channels, 64, A, residual=False, adaptive=adaptive)
        self.l2 = TCN_GCN_unit(64, 64, A, adaptive=adaptive)
        self.l3 = TCN_GCN_unit(64, 64, A, adaptive=adaptive)
        self.l4 = TCN_GCN_unit(64, 64, A, adaptive=adaptive)
        self.l5 = TCN_GCN_unit(64, 128, A, stride=2, adaptive=adaptive)
        self.l6 = TCN_GCN_unit(128, 128, A, adaptive=adaptive)
        self.l7 = TCN_GCN_unit(128, 128, A, adaptive=adaptive)
        self.l8 = TCN_GCN_unit(128, 256, A, stride=2, adaptive=adaptive)
        self.l9 = TCN_GCN_unit(256, 256, A, adaptive=adaptive)
        self.l10 = TCN_GCN_unit(256, 256, A, adaptive=adaptive)
        self.fc = nn.Linear(256, num_class)
        nn.init.normal_(self.fc.weight, 0, math.sqrt(2. / num_class))
        bn_init(self.data_bn, 1)
        if drop_out:
            self.drop_out = nn.Dropout(drop_out)
        else:
            self.drop_out = lambda x: x

    def forward(self, x):
        N, C, T, V, M = x.size()
        x = x.permute(0, 4, 3, 1, 2).contiguous().view(N, M * V * C, T)
        x = self.data_bn(x)
        x = x.view(N, M, V, C, T).permute(0, 1, 3, 4, 2).contiguous().view(N * M, C, T, V)
        x = self.l1(x)
        x = self.l2(x)
        x = self.l3(x)
        x = self.l4(x)
        x = self.l5(x)
        x = self.l6(x)
        x = self.l7(x)
        x = self.l8(x)
        x = self.l9(x)
        x = self.l10(x)
```

```
50.
51.         # N*M,C,T,V
52.         c_new = x.size(1)
53.         x = x.view(N, M, c_new, -1)
54.         x = x.mean(3).mean(1)
55.         x = self.drop_out(x)
56.
57.         return self.fc(x)
```

## 三、算法成果

我们在效果较好的模型 CTR-GCN 上进行了对比测试，原数据集在其上的结果约 43%,而我们的数据集扩增方法将其提升至 44.8%，在其他模型或数据模态上也有较好的表现，最终验证集准确率 51%,测试准确率 48.572%。

## 四、创新点

- 对赛事数据集进行数据增广，将“对称”动作反序生成新样本，增加数据量，提高了模型性能和泛化能力；
- 采用集成模型，对众多单模型的预测结果进行融合，降低单模型训练的偶然性
- 将现有模型 TE-GCN 经过 xxx 修改，添加至集成模型，提高了模型综合性能；
- 修改了部分模型代码，以匹配赛事数据集格式，集成了更多模型，提高了综合性能；
-

从数据集的类别，分类具体目标出发，提取出共同点，在数据集上做文章，针对分类准确率较低的类别互相充实训练集，以达到目的。