基于 TinyML 实现对不同投篮姿势的识别

摘要

TinyML 是近年来非常火热的微型机器学习的技术，它的特点是低功耗、小内存、低成本、小体积，一块芯片即可承载，在各种终端应用广泛。受此启发，在携带 arduino 的手环上利用 arduino 自带的 IMU 读取投篮的三维加速度数据，运用 TinyML 进行推断，从而实现对不同投篮手势的识别，具有投篮姿势矫正等实际的应用场景和意义。

关键词　TinyML　投篮手势　机器学习　arduino

问题重述

Arduino 是 TinyML 中比较常用的开发板，其具有三维加速度传感器 IMU 和摄像头等部件。要求基于 TinyML 技术进行数学建模，使得在实际投篮运动中，通过携带佩有 arduino 的手环，实现对不同投篮姿势的鉴别。

（该问题来自于作者选修的 tinyML 课程）

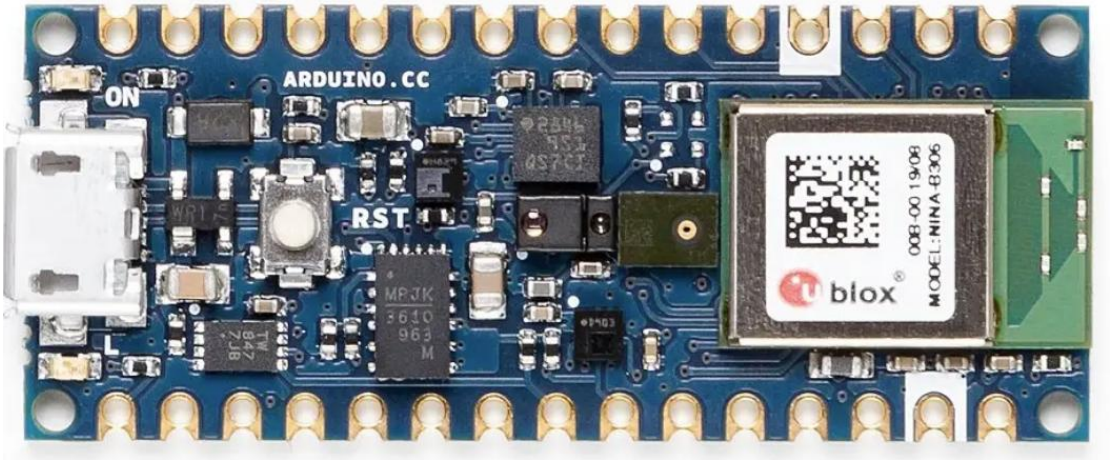

图 1 arduino 开发板

问题分析

此问题实质上是要能利用 TinyML 对不同人的投篮姿势进行识别，由于最终落地形式是佩戴携带 arduino 的手环，所以无法利用图像识别进行检测，于是考

虑利用手环上的 arduino 的 IMU 收集投篮时惯用手的三维加速度数据，并通过事先烧录好的模型识别不同投篮者的该数据，从而进行推断。

具体而言，可以分为数据收集、数据处理、模型建立和训练三个步骤。

模型假设

为了解决这个问题，优化模型应用，我们将作出以下假设，以排除现实因素对模型的微小影响。我们作出的假设都是合理且符合事实的。

1) 假设数据收集时的数据误差在允许范围内
2) 假设收集的投篮数据可以视为实际篮球运动中的投篮数据。

符号说明

符号	含义
IMU	三维加速度传感器

模型建立和求解

数据收集

通过绳子将 arduino 固定在手腕上方，并将 arduino 与电脑连接，烧录上 IMU 数据收集程序，打开串口监视器，随着手腕的挥动，触发数据收集的阈值，IMU 收集的数据在串口监视器上显示。

每个人收集 5 次，共收集 25 组投篮数据。

每次收集的数据长度为 100*3，采用降采样，即将收集频率设置为 25hz，所以每次采集 4s 的时间长度的数据。

图 2 数据收集现场

数据处理

将 100*3 的数据截取成 40*3 的数据，确保整个数据窗口都是有效的投篮移动数据。

将每个人的 5 条数据通过平移 10%以内的方式，衍生成 30 条。

进行数据分类，以 6:2:2 的比例分为训练集、验证集、测试集。

对训练集进行数据增强：增加随机噪声、上下平移一定比例或者一定值。

padding 处理，因为模型的一组输入是 64*3。

最终得到的数据如图（4 个人的 y 轴的数据）。

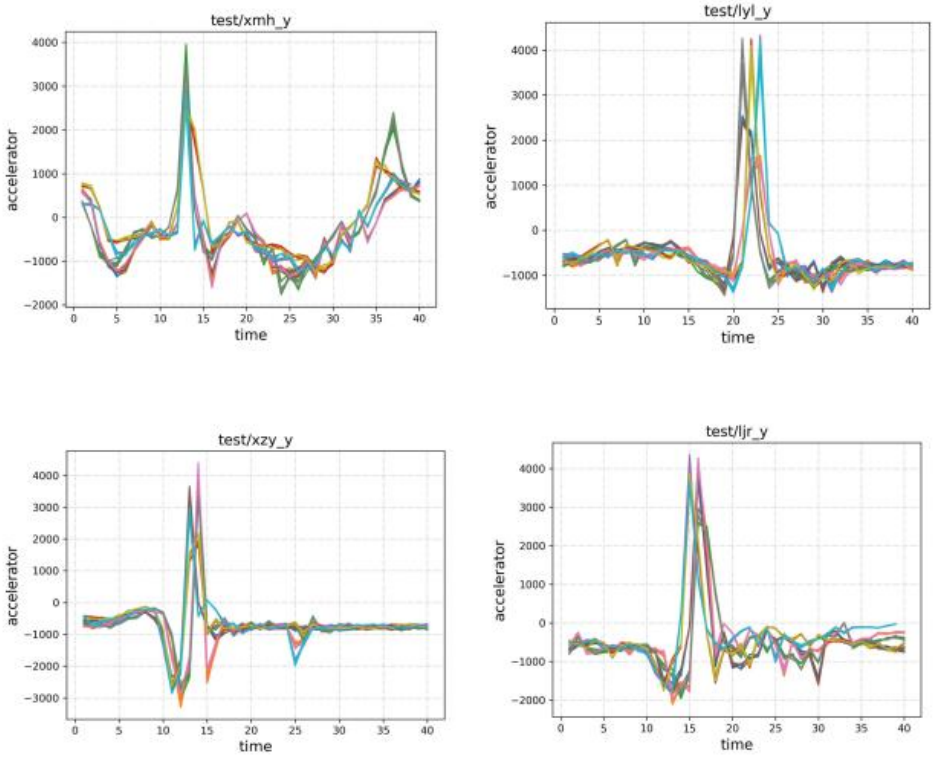

图 3 4 个人的 y 轴数据

模型建立

采用两层卷积核和池化模型，具体如图 7。

模型输入是 1*64*3*1，输出为 1*5，代表 5 个分类的概率。输入不用 40*3，是考虑了实际篮球运动时数据收集的误差。以数据的余弦相似度和欧氏距离两个指标来衡量 5 分类的可行性。

17.21616	15.55458	5.954562	6.457811	1.886986
15.55458	17.31438	6.965492	8.272341	2.094539
5.954562	6.965492	19.31571	10.45784	2.562115
6.457811	8.272341	10.45784	19.10866	9.04941
1.886986	2.094539	2.562115	9.04941	22.32022

图 4 余弦相似度

如上图，第(i,j)组数据为第 i 个人和第 j 个人的 5 组原始数据的余弦相似度之和，如（1，1）即为图 5 的数据之和。由余弦相似度的定义易得其值越大，代表数据间的相似度越高。从而可以看出不同人的数据相似度并不高。

1	0.900096	0.867869	0.659091	0.514298
0.900096	1	0.926352	0.387361	0.276338
0.867869	0.926352	1	0.382815	0.291379
0.659091	0.387361	0.382815	1	0.902482
0.514298	0.276338	0.291379	0.902482	1

图 5 一个人的 5 组数据的余弦相似度

以欧式距离为指标时，数值越小，代表相似度越高，其余同理。

10.00061	12.68726	18.4692	19.34933	21.07067
12.68726	9.934161	18.12334	18.48762	21.08518
18.4692	18.12334	8.687751	16.93225	19.23592
19.34933	18.48762	16.93225	9.49414	17.19282
21.07067	21.08518	19.23592	17.19282	5.614837

图 6 欧氏距离

因为输入的是一组时间序列数据，所以采用图像识别惯用的 Conv2D 和 MaxPool2D 层，可以有效识别相邻数据之间的关系，比如投篮数据中的波峰和波谷。训练时，通过减小过滤器面积，实现对数据特征的更精确的把握。最终确立过滤器大小如下图。

```
model = tf.keras.Sequential([
    tf.keras.layers.Conv2D(
        8, (4, 3),
        padding="same",
        activation="relu",
        input_shape=(seq_length, 3, 1)),  # output_shape=(batch, 64, 3, 8)
    tf.keras.layers.MaxPool2D((3, 3)),  # (batch, 21, 1, 8)
    tf.keras.layers.Dropout(0.1),  # (batch, 21, 1, 8)
    tf.keras.layers.Conv2D(16, (4, 1), padding="same",
                           activation="relu"),  # (batch, 21, 1, 16)
    tf.keras.layers.MaxPool2D((3, 1), padding="same"),  # (batch, 22, 1, 16)
    tf.keras.layers.Dropout(0.1),  # (batch, 7, 1, 16)
    tf.keras.layers.Flatten(),  # (batch, 112)
    tf.keras.layers.Dense(16, activation="relu"),  # (batch, 16)
    tf.keras.layers.Dropout(0.1),  # (batch, 16)
    tf.keras.layers.Dense(5, activation="softmax")  # (batch, 6)
])
```

图 7 模型参数

模型训练

```
model.compile(optimizer="adam",
              loss="sparse_categorical_crossentropy",
              metrics=["accuracy"])
```

图 8 训练方式

采用 adam 的优化方式，以及交叉熵的损失指标。

```
Epoch 91/100
100/100 [==============================] - 0s 4ms/step - loss: 0.2980 - accuracy: 0.9155 - val_loss: 0.0687 - val_accuracy: 1.0000
Epoch 92/100
100/100 [==============================] - 0s 4ms/step - loss: 0.2558 - accuracy: 0.9334 - val_loss: 0.0666 - val_accuracy: 1.0000
Epoch 93/100
100/100 [==============================] - 0s 4ms/step - loss: 0.2612 - accuracy: 0.9271 - val_loss: 0.0681 - val_accuracy: 1.0000
Epoch 94/100
100/100 [==============================] - 0s 4ms/step - loss: 0.2572 - accuracy: 0.9287 - val_loss: 0.0617 - val_accuracy: 1.0000
Epoch 95/100
100/100 [==============================] - 0s 4ms/step - loss: 0.2423 - accuracy: 0.9325 - val_loss: 0.0626 - val_accuracy: 1.0000
Epoch 96/100
100/100 [==============================] - 0s 4ms/step - loss: 0.2280 - accuracy: 0.9364 - val_loss: 0.0596 - val_accuracy: 1.0000
Epoch 97/100
100/100 [==============================] - 0s 4ms/step - loss: 0.2312 - accuracy: 0.9355 - val_loss: 0.0574 - val_accuracy: 1.0000
Epoch 98/100
100/100 [==============================] - 0s 4ms/step - loss: 0.2319 - accuracy: 0.9337 - val_loss: 0.0582 - val_accuracy: 1.0000
Epoch 99/100
100/100 [==============================] - 0s 4ms/step - loss: 0.2323 - accuracy: 0.9358 - val_loss: 0.0601 - val_accuracy: 1.0000
Epoch 100/100
100/100 [==============================] - 0s 4ms/step - loss: 0.2348 - accuracy: 0.9362 - val_loss: 0.0565 - val_accuracy: 1.0000
1/1 [==============================] - 0s 18ms/step - loss: 0.0550 - accuracy: 1.0000
tf.Tensor(
[[12  0  0  0  0]
 [ 0 12  0  0  0]
 [ 0  0 12  0  0]
 [ 0  0  0 12  0]
 [ 0  0  0  0 12]], shape=(5, 5), dtype=int32)
Loss 0.05502120032906532, Accuracy 1.0
```

图 9 训练结果

在 colab 上进行训练，某次稳定的结果如图，可以看出，在最后的 epoch 时，其验证精度已经达到 1.0。在测试集上的推断都是正确无误，精度也到到 1.0。

结果呈现

将该训练完的模型通过xxd转化为C文件，编写测试文件magic_wand_test.cc（附录），在电脑端模拟arduino的实现，即手动给模型输入一个人的数据，推断输出应该是此人的代号，结果如图。

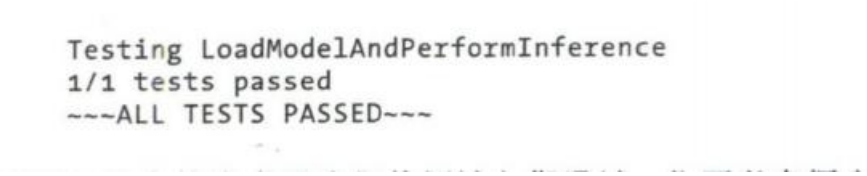

图 10 测试结果

说明测试通过,即模型训练无误。由于设备和场地原因，无法实现实际投篮的检验。

结论

尽管模型训练结果较好，测试也没有问题，但是在实际篮球运动中，不同的位置出手投篮会有不同的三维加速度数据产生，这意味着同一个人在不同位置的投篮很可能无法被识别为同一个姿势。所以 TinyML 在此方面的应用，更可能应着眼于篮球投篮训练中对投篮姿势的矫正，从而提高投篮命中率。显然，篮球罚球训练是一个比较好的应用场景，可以事先收集比较标准的罚球姿势，并进行模型训练，使得在带有 arduino 的手环的佩戴者训练时，若姿势相同或不同，手环可以予以语音提醒，达到矫正姿势的目的。

附录

随交文件说明

train 文件夹存放数据处理和模型训练的 python 文件。
data 文件夹存放收集的投篮数据。
test 文件夹存放测试文件。
evaluation 文件夹存放余弦相似度和欧氏距离具体数据。