基于安卓的超声手势识别

万浩然

2020 年 3 月 5 日

第一章 系统框架

基于超声波的非接触式手势识别系统利用设备自身的扬声器和麦克风发射超声波，该声波在用户手部反射后被麦克风接收，经信号处理后提取出用户手部移动距离、移动速度、手部位置等关键信息。最后通过这些关键位移信息来对手势进行识别。

1.1 系统基本框架

本次实现的系统基本框架如图 1-1 所示，其主要系统模块包括发射通路和接收通路两大部分。发射通路主要负责超声信号的发射，接收通路负责信号的接收、预处理、经过神经网络处理以及手势识别，将最终结果提交给上层应用。

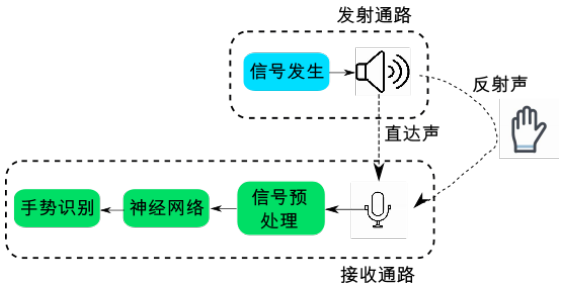

图 1-1: 基于安卓的手势识别基本框架

1.2 发射通路

发射通路包括信号生成，发射信号使用 ZC 序列，是一种相关性非常好的信号，在接收端能够很好的提取相应的距离信息，具体的性质可以参考 [1] 论文，代码实现在 SoundPlayer.java 中。

1.2.1 信号发生

信号发生的主要内容实现在 SoundPlayer.java 中，使用原始长度为 127，u=1 的 ZC 序列通过 FFT 后将其整体搬移到 18.75kHz 的频率部分，再通过 IFFT

得到时域长度为 1024 的探测信号。该信号为循环信号，实现上采用生成 1024 个采样点的信号后循环播放这段信号。本项目中接收端和发射端均使用 48kHz 的采样率。信号发生器的输入包括采样频率和循环播放的缓存大小，同时对上层应用提供基本参数设置、开始发送、停止发送接口。

1.2.2 扬声器

扬声器的调用同样在 SoundPlayer 类中实现，用于发射超声信号。这里通过调用 AudioTrack 类循环写入周期性的探测信号，来实现循环放音的功能。顶部扬声器对应的调用参数是 STREAM_VOICE_CALL。在 ThreadInstantPlay() 中完成放音操作。

1.3 接收通路

接收通路包括麦克风、信号预处理、神经网络以及手势识别几个部分。本节主要介绍前两部分。

1.3.1 麦克风

麦克风用于采集超声信号。这里通过调用 AudioRecord 类的 read 方法，循环读取录音得到的数据存入到预先定义好的录音缓存中。双声道通过设置 channelConfig 为 CHANNEL_IN_STEREO，缓存中得到的数据流会以左、右、左、右的形式排列。对于大部分安卓手机来说，左声道对应的是其底部麦克风，右声道对应的是其顶部麦克风。对于每个声道，每接收到 1024 个点 (约 0.02s) 可以看作一帧数据，将进行数据的预处理。

1.3.2 数据预处理

这里数据预处理是使用对 ZC 序列的传统处理方法进行处理得到每个信道的 Channel Impulse Respose（CIR），具体方法是将发送原信号进行 FFT 取出相应的频率部分，对接收到的信号做 FFT 也取出相应的频率部分，然后将原信号的部分进行共轭变换之后与接收信号的部分进行复数相乘。这一步相当于频域上的相关操作，然后将复数相乘之后的 127 个点补 0 到 1024 个点进行 IFFT 得到该帧的 CIR，这 1024 个点对应的是 1024 个距离上的反射强度，最高峰对应着直达路径，后面的则对应反射路径。每两条路径之间的距离是 0.7cm，1024 个点对应着约 3.6m（来回）的范围，然后做差分来获取移动信息。在 ThreadInstantRecord() 中完成录音以及预处理的操作。

第二章 神经网络

本节介绍此项目中使用的神经网络架构以及使用该架构的一些思路。

2.1 输入信息

根据上一节的介绍，我们通过对每帧的信号处理可以得到每帧信号的CIR，这个 CIR 对应着每条路径上反射信号的强度，我们有两个麦克风，一个扬声器，就能够形成单发两收的场景，通过对两个链路的信息的分析我们能够粗略的定位出手的位置。再结合时间信息，我们能够持续对 CIR 上的某个反射体进行追踪。这是使用建模的想法，此项目中使用深度学习的方法则不同自己对频谱图进行分析，我们将两个通道中的直达路径之后的 256 条路径作为追踪范围，由于每个动作会持续一段时间，我们使用 48 帧作为持续的总时间。那么形成的输入为 $2 \times 256 \times 48$。实际使用中是使用每帧产生后将其前面的 47 帧和它一起作为输入来保证每产生一帧都能产生一个输出。

2.2 类别设计

对于手势识别的类别设计是比较重要的，因为有些手势会比较容易混淆。此项目中考虑八种类别，分别是：UP，UP#，DOWN，DOWN#，GRAB，GRAB#，STATIC，OTHERS。分别对应着上翻，上翻回位，下翻，下翻回位，握拳，握拳回位，静止和其他冗余动作。该目标的困难点在于：

1、上翻回位的动作和下翻的动作模式是非常类似的，同样的下翻回位和上翻的动作也是很类似，唯一的区别可能是两者速度有差别，上翻回位的用户手部速度是慢于下翻动作的，我们认为用户做动作时要比回位动作更加干脆。

2、对于单个向上的模式，例如单次上翻，然后用户停止了手部动作，此时，我们将其判别为上翻还是下翻呢?

目前的方法：

由于原始数据是类似图像的模式，于是我们使用划窗卷积神经网络（CNN）来提取特征，然后使用序列模型来对序列相关的特征进行处理。其中可以使用状态机来对神经网络的结果进行约束以增加实时的效果，不过本项目中并没有使用状态机，同学们可以尝试使用，状态机可以考虑前后序列的状态来调整本次的输出，Transformer 只能考虑序列内的前后关系。

2.3 网络架构

本节介绍我们所使用的神经网络架构。网络总体架构如图 2-1 所示，包括

卷积层和 Transformer Encoder 层，最后加一个全连接层判别输出。

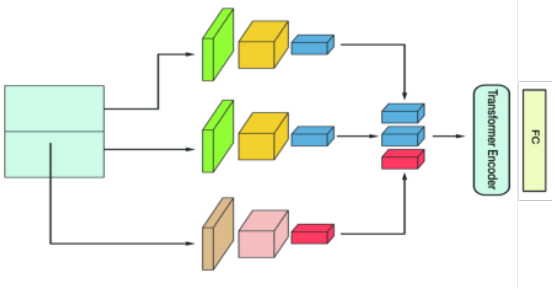

图 2-2: 神经网络架构

2.3.1 卷积层

对于卷积层来说，得到的 CIR 的维度为 $2 \times 256 \times 48$，我们使用滑动窗口的方法产生输入，那么这个窗口的大小是多大呢，太大了容易将两个动作放到同一个窗口里，太小了只能看到部分信息，于是考虑设计多分枝，将前半部分和后半部分的序列分别输入到卷积层，然后将整体输入到第三个分支。我们考虑一个窗口的大小为 32，使用的步长为 8，然后一个序列由三个组成，于是总序列长度就是 48，然后每个窗口的前 16 个时间段是输入到第一个分支，后 16 个时间段输入到第二个分支，整个 32 个时间段输入到第三个分支。

卷积层使用的 kernel size 是 3×3，并且使用了 padding 来补零，一共三层卷积层，使用的 filter 数目分别是 16，32，16，每层后面都有 max pooling 层来选取特征，并使用 batch 归一化进行处理，然后进过激活函数，三个 CNN 的末尾使用的激活函数是 $ReLU(x - \Delta)$。

为了让网络学习到速度的信息，在 CNN 后面加入一个阈值判别，高于这个阈值的数值才会被输入到后续的 Transformer 层，因为速度越快，会让 CIR 的差分数值越大，所以我们通过这个阈值的方法来让输入给 Transformer 的一定是有运动的数值，这个阈值设定的是 10^{-3}。

仅使用 CNN 进行学习能够提取固定窗口内的信息，但是一个动作的完成往往是前后相关联的，这个任务类似于唇语识别等自然语言处理的任务，于是我们在 CNN 之后加入了 Transformer 层来将序列的前后层联系起来。

2.3.2 Transformer 层

Transformer 的概念是在[2]这篇文章中提出的，torch 里的 Transformer 架构

也是依据这篇论文实现的。Transformer 层内部会有一系列的操作这里不进行具体介绍了，只需要了解到 Transformer 能够将序列前后的信息联系起来就好，这里使用了一层 transformer 层，3 步作为一个序列，使用了 8 个 head。

2.3.3 全连接层

在经过了 CNN 和 Transformer 层的处理之后，采用一层全连接层来处理输出的分类。这里的神经网络层数都是经过了缩减来想要在移动端上运行的，大家可以设计自己的神经网络来进行不同任务的完成。具体的网络架构见 net.py 文件。

2.4 模型量化

在训练完模型之后，为了在移动端上有更好的表现，可以对模型进行量化处理以减少参数的总大小，上述神经网络的参数大小为 1.2g，在安卓端上运行一次 inference 需要 8s 的时间，量化操作可以降低这个时间。

第三章 移植到安卓端运行

由于深度学习的推断过程仍需要大量的计算资源，我们可以使用手机手机信息通过网络传输给 PC 让 PC 来完成这个推断的计算。或者我们也能把网络设计的小一点来让其运行在安卓端，实现真正的一体化操作。本节将介绍如何将深度学习模型移植到安卓端。

3.1 保存模型

本项目使用的是 torch 搭建的神经网络，torch 官方推荐的保存模型的方式是仅保留其参数部分，架构部分则是通过代码来写出。为了将其移植到安卓端运行，我们需要将模型转换成 torch_script 代码，然后才能够在 java 端运行。

首先我们需要读取模型，在模型的设备我们设置为‘cpu’，因为手机一般是没有 GPU 的，然后我们根据模型的要求生成一个样例输入来供 torch.jit.trace() 来追踪模型的运算过程，同样也可以检查一下对于这个样例输入我们的模型是否是正常运行。然后我们调用 torch.jit.trace() 来追踪这个模型的计算过程并保存下来，这个保存下来的模型是可在安卓端运行的，详细操作可以参考 toAndroid.py 文件或者 https://pytorch.org/mobile/android/ 。

3.2 安卓端操作

在安卓端我们先前介绍了放音，收音以及一些预处理的方式，现在我们拿到了处理之后的数据并准备作为输入来给我们的神经网络模型。首先在 Gradle 依赖性中我们需要加入 implementation 'org.pytorch:pytorch_android:1.4.0' 和 implementation 'org.pytorch:pytorch_android_torchvision:1.4.0' 这两个依赖，注意这个版本需要和你 PC 上运行的 torch 版本一致，否则会报错。本项目是将模型存储在 SD 卡中，也可以直接保存在安卓工程的 asset 里，不过这会导致每次调试安装起来很慢，通过 torch 在 java 这边的 api 读入我们的模型，具体的 api 文档在 https://pytorch.org/javadoc/ 中。

我们需要将收到的信号转成 tensor 类型来作为输入，原数据我们保存为 double 类型的数组，调用 java api 中的 Tensor.fromBlob 方法来形成 tensor，这个方法需要我们将数据先变成一维的形式然后将一维的数据和数据的 shape 输入到这个方法中来生成。注意这里我们生成的 tensor 的数据类型要和你的模型的数据类型保持一致，例如都是 double 或者都是 float，否则无法运算。具体操作参考安卓工程 MainActivity.java 的 ThreatInference()。

3.3 展示结果

对于推断完成之后我们可以使用一些方式来展示推断结果，本次仅使用 Message Handler 来展示在屏幕上。同学们可以使用状态机之类的进一步提升使用体验。

第四章 讨论

这个项目仍然是一个非常初步的作为参考性质的例子以供大家参考，仍然有很多可以改进的地方，例如网络的规模能否进一步缩小来让移动端的运行更加流畅，以及多次提到的状态机来提升实时系统中的精度，或者使用别的一些手势或者应用来增强使用的趣味性。

参考文献

[1] Ke Sun, Ting Zhao, Wei Wang, and Lei Xie. Vskin: Sensing touch gestures on surfaces of mobile devices using acoustic signals. In *Proceedings of the 24th Annual International Conference on Mobile Computing and Networking*, MobiCom' 18, page 591 – 605, New York, NY, USA, 2018. Association for Computing Machinery.

[2] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need, 2017.