

# 全国大学生物联网设计竞赛

## 基于深度学习的移动端空中手写数字识别

学校名称: 武汉大学

团队名称: 安卓 AI 小分队

队长: 董小威

队员 1: 肖轩淦

队员 2: 魏森

队员 3: 吴效怡

全国大学生物联网设计竞赛组委会

2019 年 6 月

## 设计作品名称

### 摘要

随着 3D 导航、实景 VR、健康、环境监测等全新需求的出现，传感器也在越变越丰富和越来越专业，比如后来陆续出现的高度气压传感器、温湿度传感器、红外线传感器、计步器、指纹传感器甚至环境辐射传感器等等。现在的传感器都在越变越小，性能越来越强。我们基于手机内置的传感器，设计了一个空中手写数字识别系统，能对数字 0~9 进行有效识别，为用户提供一种更加便捷的交互方式。

我们的系统在实现对空中手写数字进行识别的基础上，已经实现的应用场景有两个，一个是人机验证，一个是紧急求援：

(1) 人机验证。本系统实现了图书馆预约座位场景下的人机验证，用户在预约座位时，点击座位，会弹出图片数字型验证码，要求用户持手机在空中书写数字，然后单击提交按钮。如果用户的输入正确，则用户预约座位成功，否则用户再次输入。本系统的人机验证，相较于传统的验证码，具有新意，且使用便捷。用户不需要面对复杂难以辨认的图片验证码，需要做的仅仅是持手机写下数字。

(2) 紧急求援。当用户在一些场景下，如乘坐“顺风车”时，打开 APP，此时界面为全黑，看上去和没有解锁一样。如果用户遭遇紧急情况，可以触摸屏幕，以较小的幅度书写“110”等紧急求助号码，服务器端识别出为“110”时，会将用户此时的地理位置信息发送给用户的紧急联系人，增加用户获救的可能性。

**关键词：空中手写数字；人机验证；深度学习**



# 目 录

基于深度学习的移动端空中手写数字识别.....	I
摘要.....	I
第一章 设计需求分析.....	1
1.1 设计思想.....	1
1.2 设计需求.....	3
1.2.1 人机验证.....	3
1.2.2 紧急情况.....	7
第二章 特色与创新.....	9
2.1 行之有效.....	9
2.2 操作友好.....	9
2.3 应用广泛.....	9
第三章 功能设计.....	10
3.1 系统功能总架构.....	10
3.2 人机验证.....	12
3.2.1 模块功能描述.....	12
3.2.2 工作流程图.....	13
3.3 紧急求援.....	14
3.3.1 模块功能描述.....	14
3.3.2 工作流程图.....	15
第四章 系统实现.....	16
4.1 数据采集.....	16
4.1.1 数据采集示意图.....	16
4.1.2 安卓手机传感器.....	16
4.1.3 传感器开发.....	18
4.2 数据传输.....	20
4.3 软件架构和开发.....	23
4.3.1 总体架构.....	23
4.3.2 客户端.....	24
4.3.3 服务器端.....	31
4.4 算法实现.....	34
4.4.1 数据采集和预处理.....	34
4.4.2 数字切割算法.....	37
4.4.3 数字可视化算法.....	41
4.4.4 数字识别算法.....	48
参考文献.....	52

# 第一章 设计需求分析

## 1.1 设计思想

人机交互作为人与移动终端之间沟通的桥梁,由最初的命令行交互方式逐步发展到图形用户界面,使许多没有计算机相关专业知识的人群依旧能快速地使用计算机提供的各类资源。而智能手机也成为人们日常最常用的移动终端,全球智能手机用户规模迅速壮大,在2018年全球智能手机用户数量超过30亿,亚太地区的智能手机用户数量超过了全球用户总数的一半<sup>[1]</sup>。当下用户对智能手机的要求不仅停留在打电话、写短信、收发邮件这样基础的功能上,而且对智能手机的智能化提出了更高的期望。

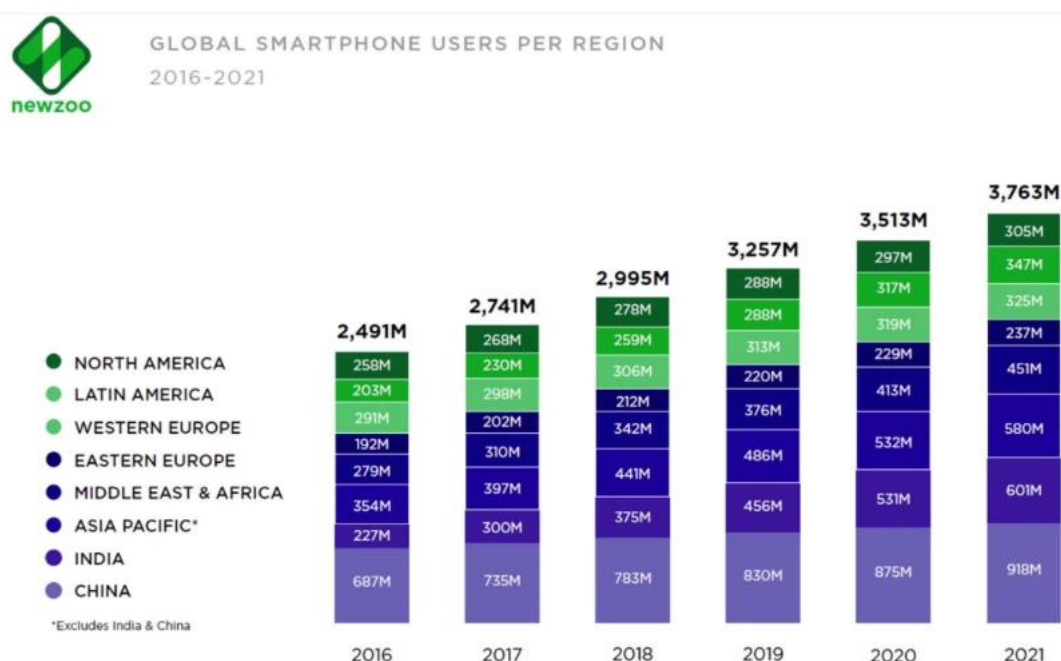


图 1-1 智能手机用户数量

智能手机中传统交互功能还不够高效与便捷,使智能手机显得不足够智能化。例如在与他人对话中,对方突然提及重要的时间节点,你想要把这些内容记录下来,在这之前你需要完成一系列动作,打开手机电源键,滑动屏幕,输入解锁密码,打开记事本工具,调整输入法,最后再进行输入。在正式开始输入前的一系列动作都可能导致时间的浪费,以致于错过将重点内容记录下来的机会。诸如此类的困扰使得用户对

于智能手机的更加人性化的交互功能提出了更高的要求。



图 1-2 智能手机内置传感器<sup>[2]</sup>

更加智能化的人机交互离不开传感器在智能手机中的应用。可以被成为“幕后英雄”的传感器们的作用渗入到使用手机的方方面面，几年前最初的机型可能仅有光线、距离、磁力、加速度等基本的传感器，所实现的功能无非也是根据环境亮度自动调节屏幕亮度以及基本的地图导航等等。而如今随着 3D 导航、实景 VR、健康、环境监测等全新需求的出现，传感器也在越变越丰富和越来越专业，比如后来陆续出现的高度气压传感器、温湿度传感器、红外线传感器、计步器、指纹传感器甚至环境辐射传感器等等。现在的传感器都在越变越小，性能越来越强<sup>[2]</sup>。

传感技术应用在智能手机中使 3D 输入方式，所谓的 3D 输入方式是指借助智能手机在空中手写内容完成信息输入的方式，这样的人机交互方式可以为使用者解除空间和线缆的束缚，在自由的空间完成相应的人机交互。将在空中手写的内容采集处理后，对所写内容进行识别是一项非常困难的任务。不同用户有不同的书写习惯，比如会使用不同的笔画顺序，字体大小和书写速度也不相同，各个国家的人还会使用不同的语言来进行输入等等。这些都对识别工作带来了很大的挑战，很难提出一个算法对所有情况进行一个归一化处理，而且解决此类问题对数据集提出了很高的要求。所以我们的系统对于数字 0~9，利用智能手机内置三轴加速度传感器设计了一个空中手写数字识别系统，为用户提供一种更加便捷的交互方式。

## 1.2 设计需求

### 1.2.1 人机验证

验证码，是全自动区分计算机和人类的公开图灵测试，英文简称为 CAPTCHA，是一种能够区分出用户是计算机还是人的程序。验证码的原理是，通过服务器对用户进行挑战，根据用户的响应，服务器来判断用户是机器还是人。服务器发出的挑战，应该是一个由人才能回答的问题，而不是机器能回答的问题，因而回答出了问题的用户，会被认为是人类。

由于网络更加深入地融合到人们的生产生活中，暴力猜测登陆、垃圾广告、网络爬虫等在网络中泛滥，一方面消耗了大量的服务器资源，另一方面也可能威胁到服务器和用户账号的安全，因而，验证码被作为一种简单有效的技术被广泛使用<sup>[3]</sup>。验证码可以在一定程度上防止恶意暴力破解密码、刷票、抢座、灌水、DOS 攻击等行为，能有效防止某些恶意用户对某些特定用户或网站使用暴力破解等方式进行的不断的非法攻击<sup>[4]</sup>。

目前常见的验证码主要以下几种：

(1) 文本型验证码：文本型验证码通常是以图片的形式呈现，在图片中，出现字母或者数字，用户需要识别出其中的文本，使用键盘进行输入，从而成功验证。这样的验证码通常如下图 1-3 所示，包含 4 个字符或 6 个字符，用户在登录前需要正确输入验证码。



图 1-3 字符型验证码

(2) 点选型验证码：点选型验证码的特点是给出很多张图片，或者一张图片中有几个需要点击的点，要求用户进行点击，如果点击的图片或位置正确，则能通过验证。点击图片的验证码最经典的是 12306 的验证码，例如下图 1-4 中，要求



用户点击所有的红枣；点击图片中某些位置的验证码如下图 1-5 所示，要求用户按顺序点击图中的“掉”、“通”、“关”字符。



图 1-4 图片点选型验证码



图 1-5 文字点选型验证码

(3) 滑动拼图型验证码：滑动拼图型验证码的特点是，图中缺了一块拼图，在图的最左边或者最右边有一块拼图，用户需要拖动滑块，使得拼图恰好落在图中的空缺位置，如果位置准确，则验证通过。一个滑动拼图型验证码的例子如下图 1-6 所示，需要用户按住滑块，向右拖动滑动拼图。



图 1-6 滑动拼图型验证码



(4) 短信验证码: 短信验证码也是一种常见的验证码, 往往是用在比较敏感的操作上, 例如登录操作、密码找回等。下图 1-7 是中国移动登录界面, 用户如果想登录, 需要点击获取验证码, 然后在收到验证码短信后, 正确输入短信中的验证码, 才能登录成功。短信验证码一方面可以证明是人还是机器, 另一方面也可以进行身份认证。

该图展示了中国移动的登录界面。顶部有一个输入框，提示文字为“请输入手机号码”，右侧有一个蓝色的用户图标。下方是另一个输入框，提示文字为“请输入短信验证码”，其右侧有一个圆角矩形按钮，文字为“获取验证码”。在两个输入框下方，有一个宽大的圆角矩形按钮，文字为“登录”。

图 1-7

传统的验证码进行人机验证, 存在一些挑战和问题。对于文本型验证码, 目前已经有基于深度学习的人工智能用于识别破解字符型验证码的方法。字符型验证码本身就是机器产生, 每种验证码都是基于一定的规则的验证码生成程序来生成。由于验证码的原始图片大小固定, 字符出现的位置也相对固定, 使得可以跳过文字检测环节直接对整图作为文字区域来识别验证码内容。虽然文本型验证码进行了一些性能增强, 例如对字符进行了扭曲, 背景有随机干扰点或者干扰线条, 生成的图片的字体更多样化, 多位字符在图片上的位置出现重合, 无法完美切割。但是近年来, 不法分子仍能利用基于深度学习的人工智能技术搭建打码平台, 快速海量识别破解字符型验证码, 为网络黑产分子撞库窃取用户信息作恶提供帮助。在人工智能被广泛应用的现在, 使用 CNN 或者其他技术, 对多位字符验证端到端的识别并不是太困难, 传统的文本字符型验证码已经不足以发挥人机验证的作用了<sup>[5,6]</sup>。

对于点选型验证码, 例如 12306 的验证码, 存在的问题往往是, 人也不能够以很高的概率回答出服务器发来的挑战。例如让选出所有的“红枣”、“网球拍”、“灯笼”甚至很怪异的物体等, 很多时候还会有迷惑性选项, 让用户并不能识别出哪些是要选的物体, 从而登录失败。google 的验证码 reCAPTCHA 也存在类似的问题, 虽然在升级

之后,通常情况下, reCAPTCHA v3 有约 50%的可能,用户只需要点击方框,网页后台就会智能判断当前的用户是人还是机器,但是另外 50%的情况下,用户面对的也是点选型验证码, google 的点选型验证码难度也比较大,而且描述比较模糊,往往是要求“点击所有的大巴”、“点击所有的交通灯”等,用户往往不能确定交通灯是否包括交通灯架等,造成回答错误,需要再次完成服务器发来的挑战。对于点选型验证码中的图片,如果图片分辨率过高,还有使用 google 提供的反向图片搜索功能来对 reCAPTCHA 系统进行破解的方法<sup>[7,8]</sup>。



图 1-8 谷歌 reCAPTCHA 系统

滑动拼图型验证码则存在一些兼容性的问题,有的系统网页端的滑动拼图验证码,在 PC 上正确把拼图滑到对应对应位置是可以通过的,而在手机端打开网页之后,无论怎么滑动,都会出现验证错误的情况,用户无法正常访问服务器上的服务。

2012 年,因为一个谷歌研究团队几乎百分百破解了其文本验证码系统,于是,谷歌将验证码升级到了语音验证码。但是,马里兰大学的研究人员提出的 unCaptcha 攻击方法 unCaptcha 可以轻松地骗过谷歌的 reCaptcha。他们使用来自实时网站的超过 450 次 reCaptcha 挑战来评估 unCaptcha,并证明可以在 5.42 秒内以 85.15% 的准确率进行破解。该方法的原理是,在用户请求语音验证码后,通过 UnCaptcha 下载音频验证码,将音频分成单个数字音频片段,将片段上传到多个其他语音转文本服务,如 Google 的音频转文字 API,然后将这些服务得出的结果转换为数字编码。随后经过一些同音词猜测后,它会决定哪个语音到文本输出最接近准确值,然后将答案上传到 CAPTCHA 字段,完成对语音验证码的破解<sup>[9]</sup>。

对于验证码破解,也有人工进行破解的,称为“打码”,多用于在自动化运行程序

防止被反自动化机制拦截，如游戏自动外挂、爬虫、黄牛外挂等，就是通过将需要验证的图片等数据转发到有人值勤的终端，由其他人类代为识别回答后再把答案转发回验证发送端回答，已经有专门的打码公司提供专门的人员识别服务。“打码”过程中，识别验证码的过程的确是由人来完成的，准确率比较高，门槛比较低，谷歌的 reCaptcha v3 的勾选验证码能够进行防范，但是很多其他的验证码并不能有效防范此类行为。

在综合分析了现有的验证码机制后，我们提出了基于手机空中手写数字的验证码机制，该机制相比现有的机制，更便捷、有效。在登录网站、提交信息时，弹出的图片验证码，用户的输入数据形式是传感器的数据，服务器端对数据进行验证识别，判断用户输入是否正确。

我们的验证码机制主要有两方面的优点：一方面，该验证码机制操作简单，用户不需要面对 12306 那样的复杂的验证码，识别出每张图片的内容，而是简单的数字，用户看清楚要写的数字验证码后，只需要把数字写在空中或者手上，就能够完成验证。另一方面，该机制同时也是有效的，机器难以提交正确的数据。并且，该机制能够提高打码平台的门槛，要求其人员必须配备一部具有相关传感器的手机，在空中手写少量数字是方便轻松的，如果短时间内写大量的数字，对其人员的体力消耗则是极大的，因而该机制在一定程度上也能够应对打码行为。

### 1.2.2 紧急情况

最近几年，多起“顺风车”事件让人揪心。在这些紧急情况中，受害人如果能够及时隐蔽地向外界求助，还是有一定的机会化险为夷的。在紧急情况中，嫌疑人可能会监视乘客，不让其报警，乘客对手机的敏感操作可能会引起嫌疑人的警觉，如何在尽量不引起嫌疑人警觉地情况下向外界传递信息成为我们考虑的重点。

在紧急情况下，我们构想了一个通信系统，使得用户能够在尽量不引起嫌疑人警觉的情况下，向自己的亲戚好友传递信息。以乘车为例，用户在乘车时，先打开我们

的 APP，APP 运行时，会让整个屏幕变为黑色，看上去就像没有打开屏幕一样。如果出现了紧急情况，用户触摸手机屏幕，在装做与嫌疑人沟通的过程中或者递交手机时，以微小的动作挥动手机写数字，如“110”。此时 APP 会将用户的地理位置信息和用户写下的数字发送到 APP 的服务器端，用户的紧急联系人、好友、亲戚等，会收到消息，进而进行相应的处理。并且我们调用了相关地图的 API，使得用户的坐标信息能够转化为实际的地理位置，便于其他人对用户进行高效的定位，如果用户的手机没有开启飞行模式，在进行求助后，其地理位置信息可以每过一段时间，进行一次回传，实现实时的定位。用户在求助完成之后，手机屏幕仍然是黑屏的，就像没有打开解锁进行操作一样，尽可能地不引起嫌疑人的警觉，而能够向外界传递信息，增加获救的机会。

我们的系统之所以采用触摸后空中书写数字的方式，而不是更简单的一键报警等，是因为如果是按音量键等一键报警，可能是会由于用户的误操作造成的，如果用户碰到了一键报警或求助按钮，想取消，必定会有一个取消的页面。在真实情景下，如果在报警或求助后还有一个取消页面，并且屏幕是亮的，可能会引起嫌疑人警觉，用户很容易被发现进行了报警或求助，造成不可挽回的后果。因而，如果是黑屏情况下，触摸一下屏幕，然后持手机在空中书写数字，隐蔽地向外界发出求助信息，比较适合这样的场景，能够在尽量不引起嫌疑人警觉地情况下，增加用户获救的机会。

## 第二章 特色与创新

### 2.1 行之有效

传统的验证码机制，例如页面中的图片字符型验证码，在深度学习进一步走向应用的过程中，已经不能再担当人机验证的角色，因为 CNN 等能够对图片字符型验证码进行高效的识别破解。我们基于空中手写数字识别，实现了基于空中手写数字的验证码，使得恶意用户难以绕过验证码机制，有效地进行人机验证。

### 2.2 操作友好

对于普通的正常用户，在验证码的场景下，只需要辨识数字，然后使用手机在空中书写验证码，对于用户来说是操作友好的，并不需要辨识复杂的验证码中的图片。而在紧急情况下，我们考虑到可能该情况下用户不被允许打开手机，所以我们设定 APP 开启时，屏幕会变为全黑，看上去和黑屏一样，书写数字也不需要做其它操作，只需要触碰一下，然后就像无意义地挥动手机一样，写下如“110”，就能够将自己的位置信息传递给紧急联系人。

### 2.3 应用广泛

我们围绕当前的应用需求，如图书馆座位预约系统受到外挂脚本的影响，乘坐“顺风车”需要有比较可靠的求助方式，已经设计实现图书馆预约中空中手写数字验证码机制，以及紧急情况下的手写数字发出求助信息等。空中手写数字还可以有很多其他方面的应用，如结合身份验证，扩展到空中手写数字做为支付密码的输入，能提高支付的安全性等。

## 第三章 功能设计

### 3.1 系统功能总架构

本系统功能基本架构如图 3-1 所示：

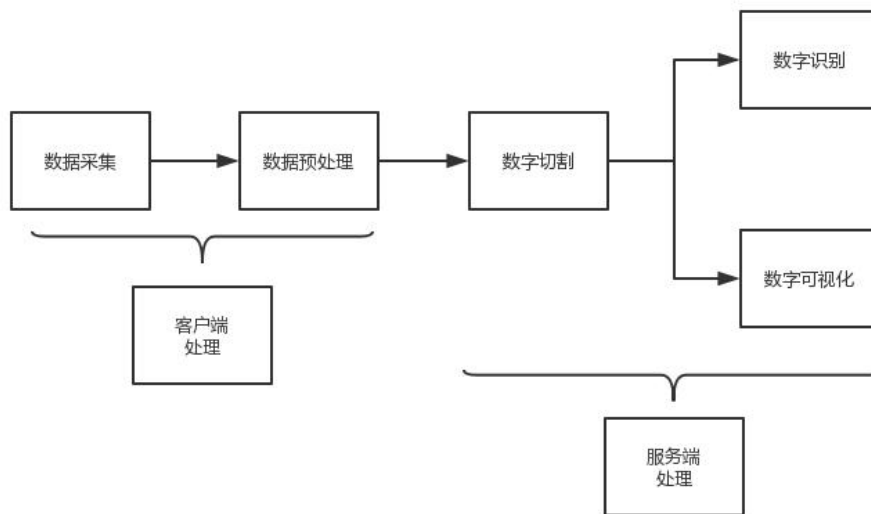


图 3-1 系统功能架构图

识别数字需要一个空中手写数字的数据库作为训练集，所以事先我们已经通过自己开发的数据采集 APP 采集了数据作为训练集。接着我们通过手机上内置三轴加速度传感器，获取用户的输入数据。我们在收集数据时，就采用打标记的方式进行收集，用于判断此段数据是属于静止段还是数据段。然后，我们就使用刚才进行预处理之后的数据实现一个判断一段数据是属于静止段还是数据段的二分类器。在完成数字切割之后，原本连续的数据变成了相互独立的数字段，采用基于深度学习的 CNN-BLSTM 模型对每个数字段进行数字识别。最后，就是实现可视化的部分：对于手机采集到的三轴上的线性加速度，我们可以通过一次积分，将加速度转化为速度，再一次对速度



积分，将速度转化为位移，在一个二维平面上画出每个时刻的位移点，这些点就构成了空中书写数字的轨迹。

本系统应用场景如图 3-2 所示

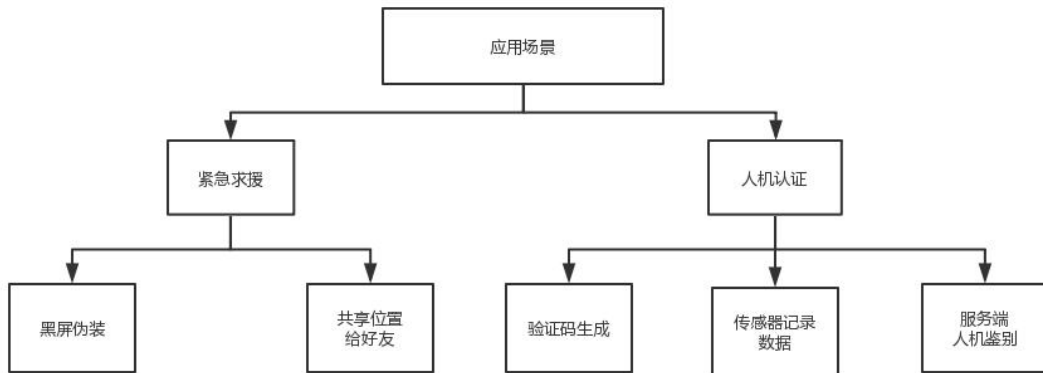


图 3-2 系统应用场景

我们系统有两个应用场景，一是在图书馆预约座位的时候进行人机验证。在图书馆预定位置时，需要防止通过抢座软件恶意抢座位，可以通过我们的验证码机制来进行人机识别。点击任意一个座位图标便会出现验证码，此时可以在空中书写验证码，手机中的传感器会记录下数据，APP 会与服务器端通信，传递刚才采集的数据，由服务器端识别输入，并将结果返回给移动端 APP。

第二个应用场景是在乘坐顺风车遇到危险时进行紧急求援。用户打开 APP 触屏，屏幕会变黑，这样可以不引起企图行凶者的警觉。之后在需要时，乘客可以隐蔽地在空中书写“110”进行求助。当服务器端识别输入数字串为 110 时，客户端会获取当前位置信息并发送给服务器。服务器会将乘客所在的位置再发送给紧急联系人。



## 3.2 人机验证

### 3.2.1 模块功能描述

简单的图片验证码很容易被 OCR 技术破解，基于 CNN 模型，成功识别图片中的字母或者数字已经不是一件复杂的事，传统的图片验证码无法有效的区分人和机器。通过用手机在空中手写验证码，并且将所写的数据进行采集处理传输，服务器根据所写数据，来判断输入验证码的是人还是机器。

相比原传统的图片字符型验证码，我们的验证方式大大降低了被攻击的风险，如果一些恶意用户想跳过验证机制，想要模拟出人的书写方式那么代价是很大的，很可能需要一个高质量的打码平台，使用真正的人来进行输入。而对于正常的普通用户，在进行验证时，只需要用户书写 0~9 的 4 位数字，这种简单的操作对于广大用户来说是非常友好的，既避免了识别复杂验证码带来的困扰，也不用像等待手机短信验证码那样花费较多不必要的时间。

人机验证模块由客户端和服务端构成，客户端可以是网页，也可以是我们制作的 APP。以我们的 APP 中图书馆座位预约为例，在 APP 中预约图书馆座位时，单击座位，会弹出一个图片，内容是 4 位数字的验证码，用户想成功预约座位，需要使用手机在空中正确书写 4 位数字验证码。书写完毕后，格式化的数据会传送到服务端，服务端对数据进行数字切割和识别，最终将数字识别的结果和验证码进行比较，如果一致，用户预约座位成功，页面跳转到成功页面，否则验证码界面进行刷新，用户需要重新书写验证码。通常情况下的人机验证流程图，如接下来的 3.2.2 节中图 3-3 人机验证工作流程图所示。

### 3.2.2 工作流程图

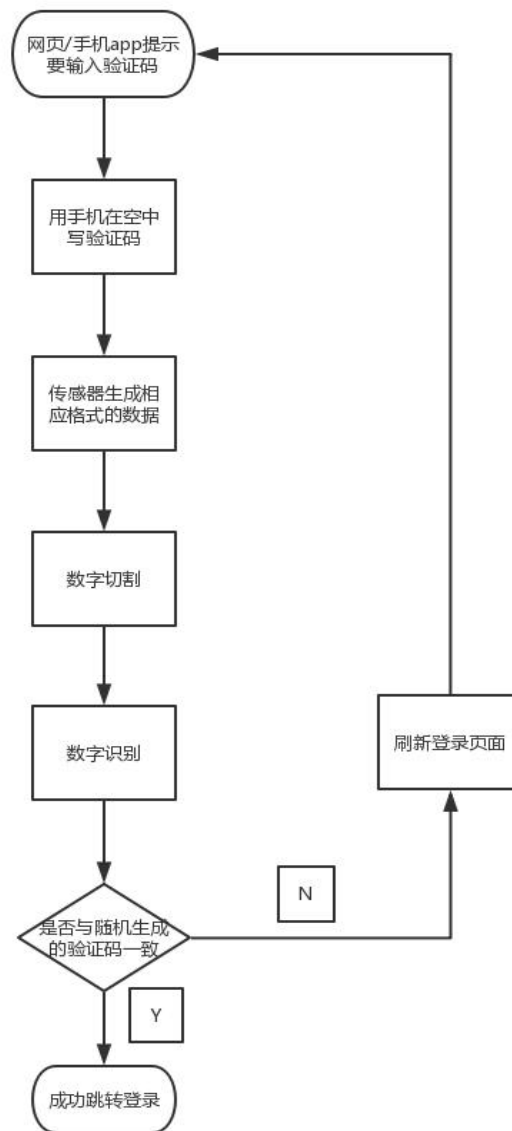


图 3-3 人机验证工作流程图

### 3.3 紧急求援

#### 3.3.1 模块功能描述

近些年来，滴滴打车事故频发，我们不得不重视起这个问题。在紧急情况下，乘车者可能没有机会在嫌疑人监视下解锁手机打字发送求救短信，或者拨打电话向外界求救。我们的系统可以让乘车者快速且隐秘地向外界发送求助信息。首先用户打开APP，屏幕会变黑，这样在紧急情况发生时，用户只需要触摸屏幕，以较小的幅度持手机在空中书写数字，尽量不引起企图行凶者的警觉，在空中手写紧急呼救电话例如“110”，系统服务器端将采集到的数据进行预处理切割，判断输入数字是什么，如果是紧急呼救电话“110”则从数据库中读取好友信息，将乘客此时所在的地理位置通过短信等方式共享给好友。只要有好友看到了这条紧急信息就可以立刻报警，警方也可以快速掌握乘客所在位置，以便及时展开救援行动。

如果系统设计成按一次屏幕或者按一次电源键或音量就共享地理位置，虽然方便但也容易因为乘客误触屏幕或者不小心挤压到电源键或音量键而引发报警机制，引起不必要的麻烦，增加好友的困扰和警方的工作负担。如果设计为一次触屏或触键后共享位置且在用户误操作之后可以取消，必定要设计一个取消求助页面，在真实的紧急情况下，如果企图行凶者发现了这个页面，一方面就会知道用户求助了，另一方面，可能会代用户进行取消。因此，我们的系统采用的设计是先打开APP，屏幕变成黑色，在关键时刻触摸屏幕，使用小动作持手机在空中书写数字。

### 3.3.2 工作流程图

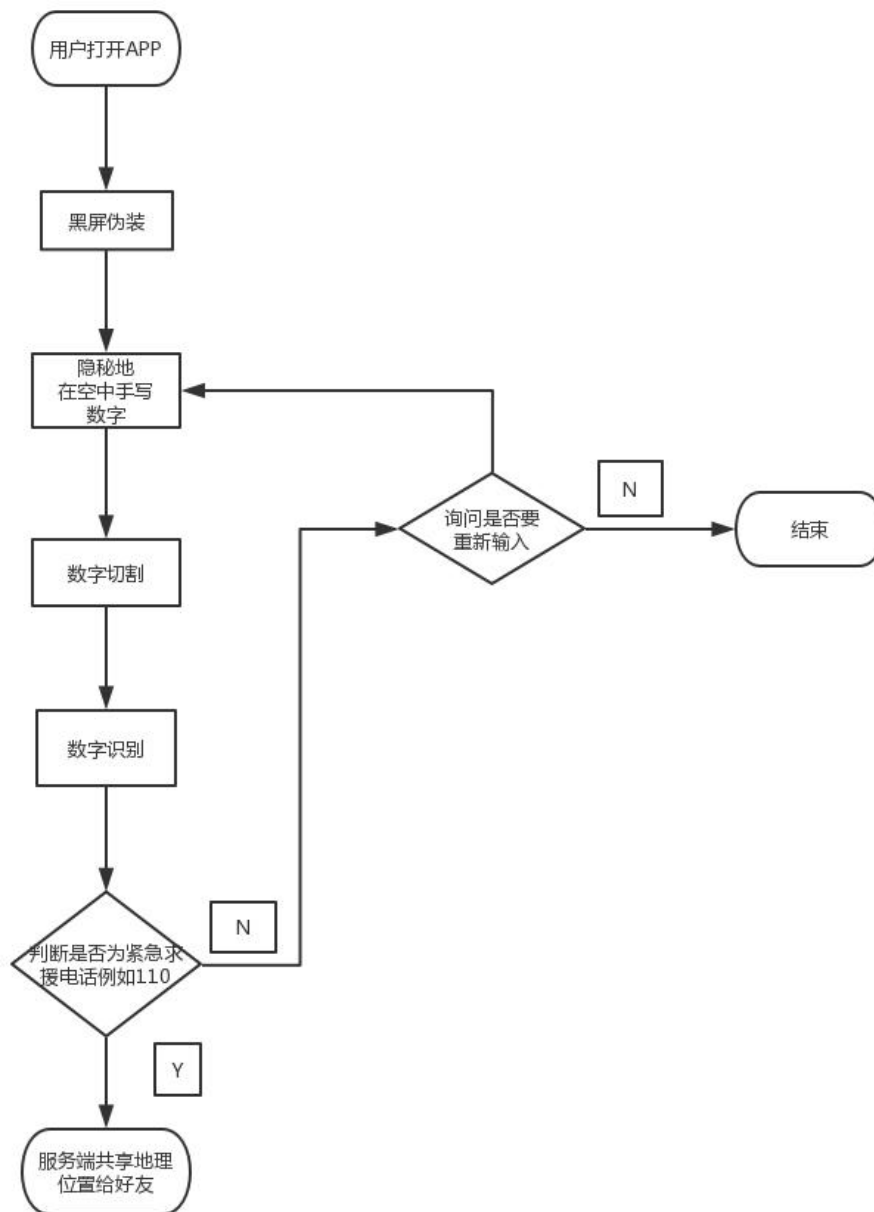


图 3-4 紧急求援工作流程图

## 第四章 系统实现

### 4.1 数据采集

#### 4.1.1 数据采集示意图

当用户使用本系统时，拿着安卓手机在空中写数字，手机记录下用户挥动过程中产生的传感器数据，包括手机的加速度、初始时重力传感器数据等，之后传回服务端，通过网络传送到服务端，在服务端完成验证码识别、密码识别，也可以在识别到用户在紧急情况下写的“110”等数字，将用户的实时地理信息传送到 APP 的紧急联系人端或者直接短信通知给用户的紧急联系人。



图 4-1

#### 4.1.2 安卓手机传感器

我们使用安卓系统的内置传感器来采集数据，安卓在 `android.hardware` 中提供了多种传感器的接口，如果安卓设备的硬件内置了这些传感器，那么我们就可以通过 API 方便地获取手机上这些传感器的数据。比如手机摆放环境外界的磁场、温度和压力、光照强度等等<sup>[10]</sup>。

常用的安卓基础传感器有：

- 加速度计
- 环境温度传感器
- 磁场传感器
- 陀螺仪
- 心率传感器
- 光线传感器
- 近程传感器
- 压力传感器
- 相对湿度传感器

常用的安卓复合传感器有：

- 线性加速器
- 大幅度动作传感器
- 步测器
- 倾斜检测器
- 方向传感器

其中三轴加速度传感器和重力加速度计的三个传感器坐标轴的方向，由设备决定，仅与屏幕的自然方向相关，当设备的屏幕方向更改时，坐标轴不会发生交换。坐标轴示意图如下图 4-2 所示：

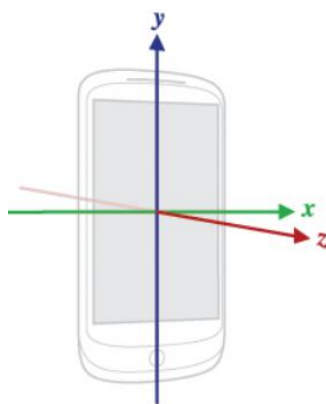


图 4-2 安卓传感器三轴方向<sup>[10]</sup>

线性加速度是线性加速度传感器可报告传感器框架内设备的线性加速度（不包括重力加速度），定义为 `TYPE_LINEAR_ACCELERATION`。从概念上看，输出结果为：加速度计的输出结果减去重力传感器的输出结果，单位为  $\text{m/s}^2$ 。

当设备不动时，所有轴上的读数应接近 0。其返回的三个加速度说明如下：

- 第一个值：代表该设备沿 X 轴的加速度。
- 第二个值：代表该设备沿 Y 轴的加速度。
- 第三个值：代表该设备沿 Z 轴的加速度。

重力加速度计用于感应重力加速度，定义为 `TYPE_GRAVITY`，给我们返回了当前设备的 X、Y、Z 三个坐标轴上的重力加速度。重力加速度的单位是  $\text{m/s}^2$ ，一般为正值，在值在 0 - 9.8 之间：

- 第一个值：代表该设备 X 轴方向上的重力加速度。
- 第二个值：代表该设备 y 轴方向上的重力加速度。
- 第三个值：代表该设备 Z 轴方向上的重力加速度。

#### 4.1.3 传感器开发

在 android 中，传感器开发需要继承自 `SensorEventListener` 类，使用 `getSystemService` 获得一个 `SensorManager` 对象。之后使用 `registerListener` 对传感器进行注册，线性加速度传感器是 `TYPE_LINEAR_ACCELERATION`，重力加速度传感器是 `TYPE_GRAVITY`。可以定义传感器采集数据的频率，如果是 `SENSOR_DELAY_FASTEST` 就是以最快的速度采集数据，其它的还有 `SENSOR_DELAY_GAME`、`SENSOR_DELAY_NORMAL` 和 `SENSOR_DELAY_UI`。

```
sensorManager = (SensorManager) getSystemService(Context.SENSOR_SERVICE);  
protected void onResume() {  
    super.onResume();  
    sensorManager.registerListener(this,
```



```
        sensorManager.getDefaultSensor(Sensor.TYPE_LINEAR_ACCELERATION),SensorManager.SENSOR_DELAY_FASTEST);
    sensorManager.registerListener(this,
        sensorManager.getDefaultSensor(Sensor.TYPE_GRAVITY),
        SensorManager.SENSOR_DELAY_FASTEST);
}
```

当传感器采集到新的数据，也就是数据发生了变化时，会触发 SensorEvent 事件，进入到 onSensorChanged 函数中进行处理。因为我们使用了多个传感器，所以在 onSensorChanged 中，我们需要获得触发 SensorEvent 事件的传感器类型，来判断数据来自哪一个传感器，并将数据记录下来。

```
public void onSensorChanged(SensorEvent event) {
    int type = event.sensor.getType();
    StringBuilder sb;
    float[] values = event.values;
    switch(type) {
        case Sensor.TYPE_LINEAR_ACCELERATION:
            sb = new StringBuilder();
            sb.append("线性加速度传感器:\n");
            sb.append("X:" + values[0] + "\n");
            sb.append("Y:" + values[1] + "\n");
            sb.append("Z:" + values[2] + "\n");
            AccData[0]=values[0];AccData[1]=values[1];AccData[2]=values[2];
            break;
    }
}
```

## 4.2 数据传输

为了实现数据采集、数据识别、数据验证等功能，我们使用基于 TCP-IP 协议，设计了基于服务器-客户端的数据传输系统，并设计了一套通信协议以便进行数据通信，数据传输系统包含客户端模块以及服务器端模块。我们采用模块化的设计方案，便于后续扩展，也方便移植。

我们使用如下格式传输数据，数据头部用于标示数据的类型，且头部中不会出现数据中的字符，以此来保证传输时如果低层协议出现了数据传输错误，也不会对我们的系统产生影响。

数据头部	数据
------	----

以下是我们协议中用到的数据头部以及其后数据含义：

表 4-1 传输协议数据头部

数据头部的值	含义
0xF0	请求服务器时间以计算延迟并同步时间
0xF1	客户端请求开始传输数据
0xF2	客户端请求开始传输数据校验码
0xF3	客户端请求传输的文件大小
0xF4	客户端请求开始传输位置信息
0xFF	结束通信
0xE0	服务端传输当前时间戳
0xE1	服务端通知客户端文件传输成功
0xE3	服务端接受客户端传来的文件大小成功
0xE4	服务端通知客户端文件与校验码校验成功

服务器端通信模块采用 Python 编写，使用了 socketserver 库优化性能。服务器端处理数据代码大致流程（省去了部分细节）如下：

```
class Myserver(socketserver.BaseRequestHandler):
    timeout = 100
    def handle(self):
        conn = self.request
        while True:
```

```

op_code = conn.recv(1)
op_code = int.from_bytes(op_code, byteorder='big')
# 根据 op_code 的值来执行相应的动作
if op_code == 0xF0:
    # 计算延迟
    conn.sendall(bytes(str(int(time.time()*1000)), encoding="utf-8"))
elif op_code == 0xF1:
    # 接受数据
    file_name=recv_file(size)
    res = generate_number_seq(file_name)
    conn.sendall(bytes(str(res), encoding="utf-8")) # 将生成的
序列返回 app
elif op_code == 0xF4: # 接受位置信息
    str_len = int(conn.recv(4))
    location_str = conn.recv(str_len)
    location_str = location_str.decode().split("..")[0]
    res_json = generate_location_info(str(location_str)) # 将坐标
转化为中文具体位置
    send_message(res_json)
elif op_code == 0xF2:
    # 数据传输结束, 传输 hash
elif op_code == 0xF3:
    # 文件大小
    size = int(str(conn.recv(1024), encoding="utf-8"))
    conn.sendall((0xE3).to_bytes(1, byteorder='big')) # 允许客户机开始
传输
    Recording = False
elif op_code == 0xFE:
    # 客户端开始记录, 重制键盘按键记录线程
    Recording = True
elif op_code == 0xFF:
    # 客户端关闭, 结束通信
    self.request.close()
    break

```

客户端通信模块采用 java 编写, 便于安卓 app 使用。客户端初始化连接代码大致流程 (省去了部分细节) 如下:

```
private void network_init() throws Exception{
    //初始化 IP 和端口
    server_address = ConfigActivity.getServerIP();
    server_port = ConfigActivity.getServerPort();
    while (true) {
        //连接服务器端
        try {
            sk = new Socket(server_address, server_port);
            is = sk.getInputStream();
            os = sk.getOutputStream();
        } catch (Exception e) {
            continue;
        }

        while (true) {
            try {
                os = sk.getOutputStream();
                send_time = System.currentTimeMillis();
                os.write((byte)0xF0);
                os.flush();
                map = read_data_from_socket();
                server_time = Long.valueOf(new String(map.get("data"), 0, byte
teArrayToInt(map.get("len"))));
                current_time = System.currentTimeMillis();
                final long delay = (current_time - send_time)/2;
                sleep(500);
            }
        }
    }
}
```

```

    } catch (Exception e){
    }
}
}
}

```

## 4.3 软件架构和开发

### 4.3.1 总体架构

我们以手机 App 作为客户端, 服务器端可选择云服务器或本地 PC 作为临时服务器, 并将所用数据储存在 Mysql 服务器中。

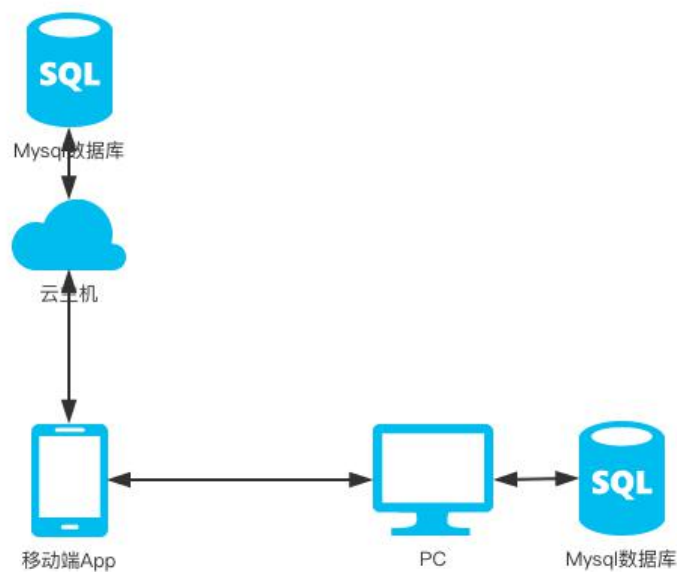


图 4-3 软件总体架构

手机客户端我们设计了两个版本，一是 NumberRecord，用于采集数据、打上标记并进行学习；二是 NumberDistinguish，用于模拟两个简单的应用场景（座位预定、紧急求助），并在后端使用了机器学习对输入数据进行识别。

### 4.3.2 客户端

#### (1) NumberRecord

该 App 作为我们项目第一阶段的客户端，其被用于进行数据采集，以便我们通过机器学习建立模型，从而进行第二阶段应用场景的模拟。

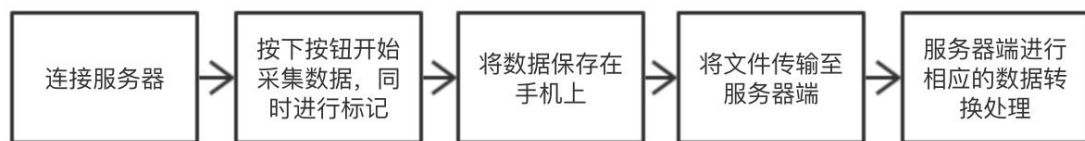


图 4-4 数据采集流程图

UI 布局采用 ConstraintLayout 设计，使用相对布局，无论手机长宽比、分辨率，都不会打乱 UI 的相对结构。主界面 UI 设计如下图 4-5 所示：

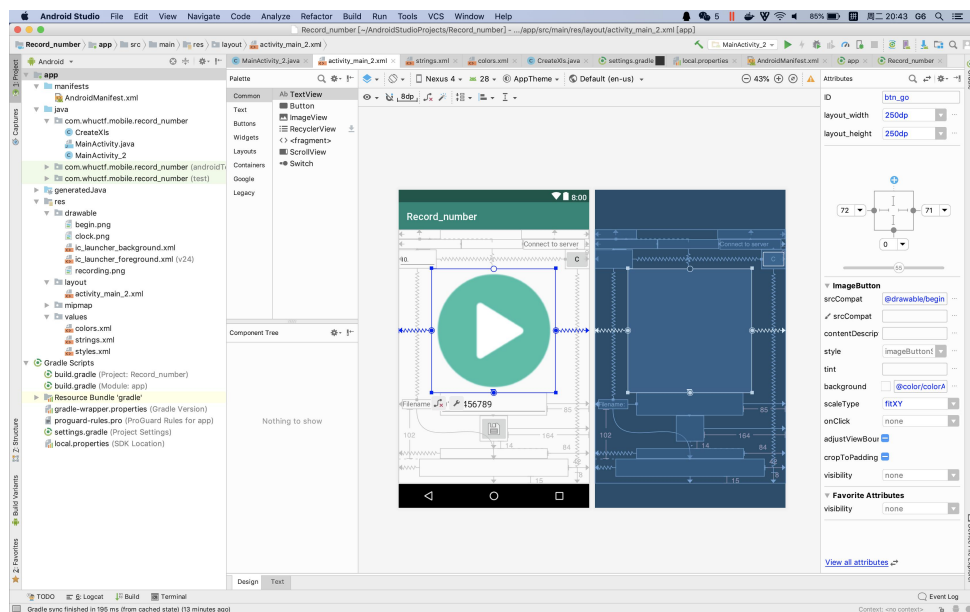


图 4-5 数据采集客户端 UI

在 Andorid 项目中，onCreate 函数是一个 Activity 的初始化函数，我们在其中使用 setContentView 渲染了布局并进行显示，并进行组件的绑定以及网络连接初始化、点击事件的注册。在我们的按钮按下时，会注册传感器监控并记录，同时为了给用户更直观的感受正在记录，按钮图标会发生变化并开始旋转；且为了防止在输入时点

击到其他的按钮，我们在输入时对其他按钮进行的隐藏。按下时的效果如下图 4-6 所示，中间按钮会开始旋转。最后，再通过 WriteXls 写入.xls 文件，并通过另一个用于网络通信的线程发送给服务器端。

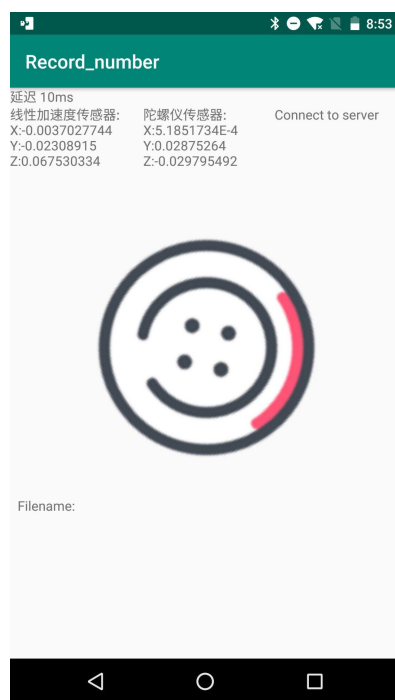


图 4-6 客户端运行时界面

## (2) NumberDistinguish

NumberDistinguish 为我们第二阶段的 App，其中包含两个模拟应用场景（座位预定和紧急求助）。

### A. 座位预定场景

图书馆等地方预定位置时，需要防止通过机器恶意抢座位，需要设计验证码，此时可通过舞动手机进行输入。

主界面中有五个空闲状态的座位，如下图 4-7(a)所示。点击任何一个作为，就会弹出验证码页面，如图 4-7(b)所示。



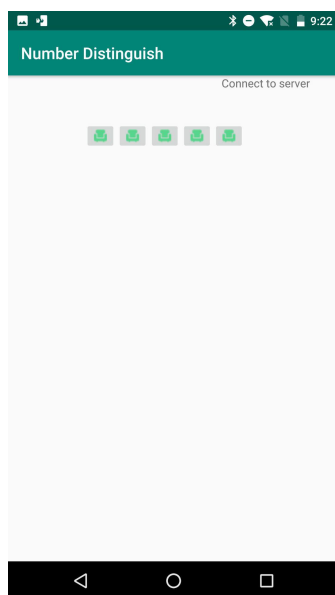


图 4-7(a)

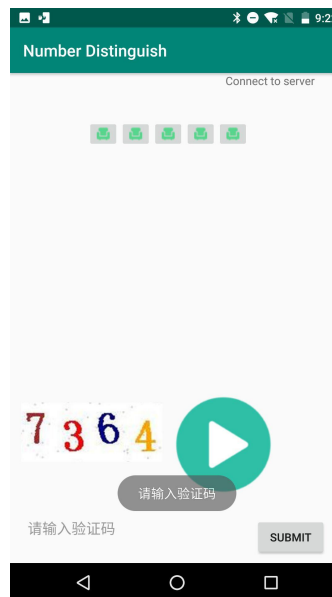


图 4-7(b)

此时，可以通过舞动手机进行输入验证码，app 会与服务器端通信传递数据，由服务器端识别输入，并将结果返回给移动端 App，返回后直接出现在输入框，例如当验证码是“7364”时，用户输入的也是“7364”，如下图 4-8(a)所示。之后点击提交，则用户预定座位成功，效果如图 4-8(b)所示。再次点击已预定的位置即可进行释放操作。

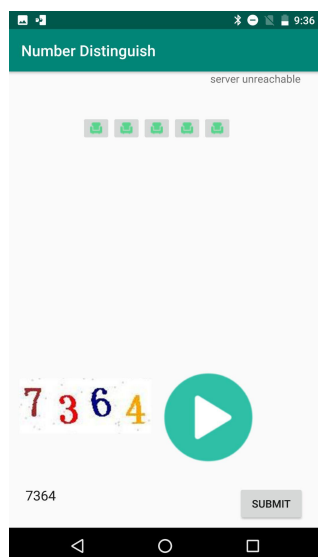


图 4-8(a)

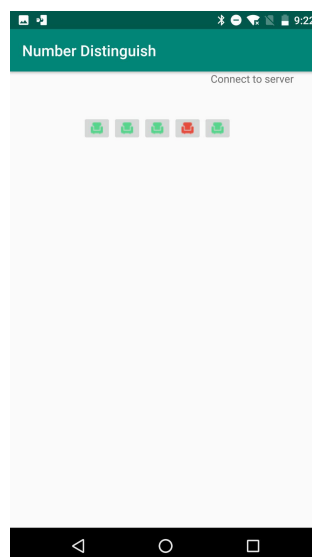


图 4-8(b)

代码实现如下，在开始时注册五个座位的按钮，并为它们注册点击事件，初始化

验证码界面，并将其显示出来：

```
private class SeatButtonListener implements View.OnClickListener {
    public void onClick(View v) {
        btn_id = v.getId();
        if(book_seat_id == btn_id){
            ImageButton imageButton = findViewById(btn_id);
            imageButton.setImageResource(R.drawable.seat_green);    // 将
已选位置标记
            booked = 0;
            book_seat_id = 0;
            Toast.makeText(Activity1.this, "取消成功", Toast.LENGTH_SHOR
T).show();
            return;
        }
        if(booked == 1){
            Toast.makeText(Activity1.this, "只能预定一个位置噢! ", Toast.LE
NGTH_LONG).show();
            return;
        }
        Toast.makeText(Activity1.this, "请输入验证码", Toast.LENGTH_SHOR
T).show();
        // 使验证码界面可见
        constraintLayout_yzm.setVisibility(View.VISIBLE);
        btn_go.setEnabled(true);
        activity1_inputText.setEnabled(true);
        btn_submit.setEnabled(true);
    }
}
```

输入按钮与之前比较相似，不过跳过了指定文件名再保存文件的过程，在按钮松开时，直接保存文件并交付服务器端进行识别，并将结果显示在输入框中，以使用户查看，值得一提的是，Android 的主线程不允许网络通信等阻塞服务，而在子线程并不能直接更改 UI，因此我们使用了 post 函数，通过消息机制使操作系统帮助我们更新 UI：

```
map = read_data_from_socket();
final String number_seq = new String(map.get("data"), 0,
byteArrayToInt(map.get("len")));
```

```

        Log.d("MYLOGDEBUG", number_seq);

        activity1_inputText.post(new Runnable() {
            @Override
            public void run() {
                activity1_inputText.setText(number_seq);
            }
        });

```

输入完成后，按下提交按钮，如果能够返回与验证码匹配的字符串的验证成功座位预定成功，否则验证失败。

```

private class SubmitButtonListener implements View.OnClickListener{
    public void onClick(View v){
        if(btn_id != -1 && post_check(activity1_inputText.getText().toString()) == 1){

            ImageButton imageButton = findViewById(btn_id);
            imageButton.setImageResource(R.drawable.seat_red);// 将已选位置标记
            booked = 1;
            book_seat_id = btn_id;

            // 使验证码界面可见
            constraintLayout_yzm.setVisibility(View.INVISIBLE);
            btn_go.setEnabled(false);
            activity1_inputText.setEnabled(false);
            btn_submit.setEnabled(false);

            Toast.makeText(Activity1.this, "预定成功", Toast.LENGTH_SHORT).show();

        }
        else{
            Toast.makeText(Activity1.this, "验证码错误，请重新输入", Toast.LENGTH_LONG).show();
        }
    }
}

```

## B. 紧急求助场景:

当用户在乘坐顺风车等遇到危险时，不方便拨打电话、发送短信的时候，可使用空中手写数字识别技术，进行隐蔽的输入“110”，从而进行求助。

需求上讲需要将界面做成全屏的无威胁性的界面，为了演示需要，我们留了几个功能性的按钮。记录数字按下的有效范围为左上角切换模式的按钮以外的其他区域，使用户按到屏幕范围内即可开始输入，同时我们还支持模式的切换。效果如下图 4-9(a) 和图 4-9(b)所示。

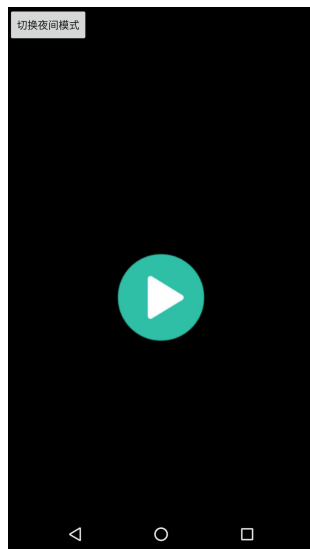


图 4-9(a)

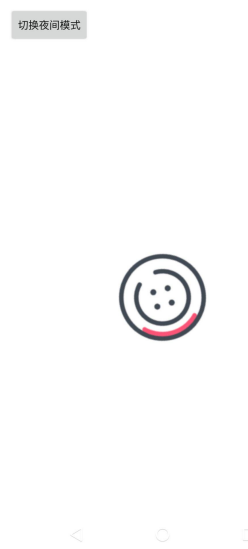


图 4-9(b)

实现如下，进入这个 Activity 是需要全屏显示，我们通过如下代码调用 API 实现：

```
WindowManager.LayoutParams lp = getWindow().getAttributes();
lp.flags |= WindowManager.LayoutParams.FLAG_FULLSCREEN;
getWindow().setAttributes(lp);
getWindow().addFlags(WindowManager.LayoutParams.FLAG_LAYOUT_NO_LIMITS);
```

除此之外，我们还能过进行日间、夜间模式的切换，通过监听切换按钮时间实现，实现代码如下：

```
private class SwitchButtonListen implements View.OnClickListener{
    public void onClick(View v){
        current_mode = (current_mode + 1) % 2;
        if(current_mode == 0){
            //正常模式
            //day();
            constraintLayout.setBackgroundColor(getResources().getColor(R.color.black));
            btn_go.setBackgroundColor(getResources().getColor(R.color.black));
        }
    }
}
```

```

    }
    else if(current_mode == 1){
        //night();//夜间模式
        constraintLayout.setBackgroundColor(getResources().getColor(R.color.white));
        btn_go.setBackgroundColor(getResources().getColor(R.color.white));
    }
}
}

```

当服务器端识别输入数字串为 110 时，客户端会获取当前位置信息并发送给服务器，获取位置信息通过系统 API 实现，后续工作由服务器端向紧急联系人发送短信。

```

public void getLocation() {
    locationManager = (LocationManager) getSystemService(getApplicationContext().LOCATION_SERVICE);
    //下面注释的代码获取的 location 为 null，所以采用 Criteria 的方式。
    Criteria criteria = new Criteria();
    criteria.setAccuracy(Criteria.ACCURACY_COARSE);//低精度，如果设置为高精度，依然获取不了 location。
    criteria.setAltitudeRequired(false);//不要求海拔
    criteria.setBearingRequired(false);//不要求方位
    criteria.setCostAllowed(true);//允许有花费
    criteria.setPowerRequirement(Criteria.POWER_LOW);//低功耗
    //从可用的位置提供者中，匹配以上标准的最佳提供者
    locationProvider = locationManager.getBestProvider(criteria, true);
    if (ActivityCompat.checkSelfPermission(getApplicationContext(),
        Manifest.permission.ACCESS_FINE_LOCATION) != PackageManager.PERMISSION_GRANTED && ActivityCompat.checkSelfPermission(getApplicationContext(),
        Manifest.permission.ACCESS_COARSE_LOCATION) != PackageManager.PERMISSION_GRANTED) {
        Log.d("Location", "onCreate: 没有权限 ");
        return;
    }
    Location location = locationManager.getLastKnownLocation(locationProvider);
    Log.d("Location", "onCreate: " + (location == null) + "..");
    if (location != null) {
        Log.d("Location", "onCreate: location");
        //不为空,显示地理位置经纬度
        showLocation(location);
    }
}

```

```

        updateLocation(location);
    }
}

```

除此之外，我们的 Demo 还支持动态服务器地址端口配置。

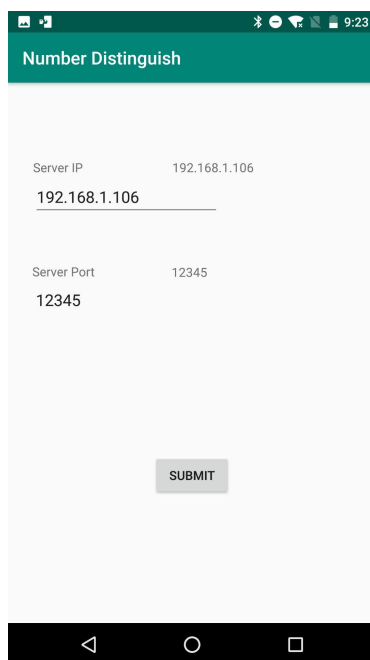


图 4-10 动态服务器端口配置

### 4.3.3 服务器端

服务器端的功能为接受客户端传来的数据并进行处理,再调用识别数字的函数进行数字的识别,最后返回给客户端,并在必要时向用户的紧急联系人发送求助短信。实现这么多功能并保证可靠性主要依托于我们自己设计的一套协议,通信协议的架构在 4.2 节中已经详细说明。

我们期望的短信中包含时间、精确的地点、经纬度等信息,我们在 App 中获取到了经纬度,在服务器端调用百度的 API,将其转换为中文位置信息。

```

def generate_location_info(loc_str):
    print(loc_str)
    ak = ""
    url = "http://api.map.baidu.com/reverse_geocoding/v3/?ak="+ak+"&output=json&coordtype=w

```

```
gs84ll&location="+ loc_str
    # print(url)
    res = requests.get(url)
    # print(res.text)
    return res.json()
```

在使用 twilio 的 API 发送短信给紧急联系人:

```
def send_message(res_json):
    current_time = time.strftime("%Y-%m-%d %H:%M:%S", time.localtime())
    current_loc = res_json['result']['addressComponent']['country'] + res_json['res
ult']['formatted_address'] + res_json['result']['business'] + res_json['result']['addressCom
ponent']['street']

    jindu = res_json['result']['location']['lat']
    weidu = res_json['result']['location']['lng']

    # message = "【Warning】您的朋友于%s 向您发出求救信息，根据他的手
机定位，他目前位于： %s， 经度： %d， 纬度： %d。请您尽快与您的朋友联系，如有
需要请联系公安局协助。" % (current_time, current_loc, jindu, weidu)

    message = "[Warning] Please help me! I'm in Latitude: %f ,Longitude: %f
" % (jindu, weidu)

    account_sid = ""
    auth_token = ""

    client = Client(account_sid, auth_token)

    message_ = client.messages.create(
        to="+8613110454286",
        from_="+12024108057",
        body=message)
```

服务器端输出信息大致效果如下图 4-10 所示，会出现发送的信息。



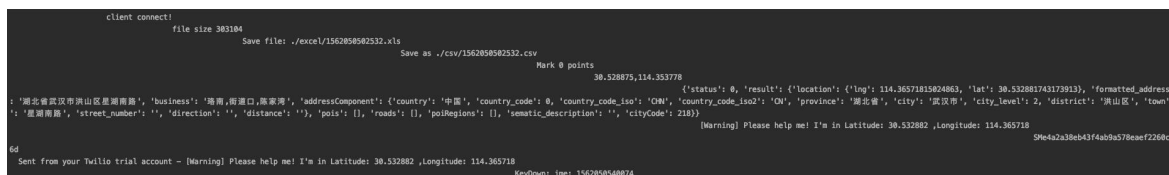


图 4-10 发送消息时服务器端消息输出

紧急联系人收到短信如下图 4-11 所示，会有经纬度信息，若调用百度 API 可以通过经纬度进一步获得地理位置信息。



图 4-11 紧急联系人收到短信

在我们采集数据样本时，我们需要在 PC 上敲击键盘以对数字端与非数字端进行分割，我们通过 getch 来实现，且做了多平台处理，兼容 Windows、Linux、MacOS 平台。实现如下：

```
def Getch():
    def _GetchUnix():
        import sys, tty, termios
        fd = sys.stdin.fileno()
        old_settings = termios.tcgetattr(fd)
        try:
            tty.setraw(sys.stdin.fileno())
            ch = sys.stdin.read(1)
        finally:
```

```
        termios.tcsetattr(fd, termios.TCSADRAIN, old_settings)
    return ch
def _GetchWindows():
    import msvcrt
    return msvcrt.getch()
try:
    impl = _GetchWindows()
except ImportError:
    impl = _GetchUnix()
return impl
```

并在将.xls 文件转换为.csv 文件的过程中，对样本点进行标记：

```
# 上一条记录时间小于按键时间，下一条大于按键时间
if int(df.loc[i, 'Time']) <= int(keydown_time_list[time_list_index][0]) and int(keydown
_time_list[time_list_index][0]) < int(df.loc[i+1, 'Time']):
    df.loc[i, 'Keydown'] = keydown_time_list[time_list_index][1]
```

## 4.4 算法实现

### 4.4.1 数据采集和预处理

要识别用户持手机在空中写的数字是什么，首先需要通过手机上的传感器，尤其是线性加速度传感器，获取用户的输入数据，然后在服务器端对用户的输入数据进行处理，反馈处理结果。如果是输入验证码或密码，用户可以看到自己的输入是否正确。

值得一提的是，识别数字需要一个空中手写数字的数据库，作为训练集。在数据库中，发掘数字间的差异性，和同一个数字的相似性，进而找出最合适的特征及最合适的数字识别算法，对用户输入的数字进行高效准确的识别判断。但是，我们查阅了网上相关资料和论文，并没有发现适合我们的应用的空中手写数字数据库，于是我们利用自己开发的数据采集 APP，收录数据作为训练集。

在我们前期调查的基础上，我们发现用户持手机写数字大致会采用两种方式，一种是抓住手机，在竖直平面内写；另一种是平着握手机，在水平平面内写。无论采用哪一种方式，用户倾向于把数字写在一个平面上。我们考虑了这两种方式，数据库中只有竖直平面内的数据，对于水平面的数据，我们进行标准化之后，将采集到的 X 和 Y 方向上的数据，转到 X 和 Z 方向上，就可以使用原来的训练模型进行数字的识别匹配。例如，用户可以以按以下方式持手机在空中书写数字，一般是单手在竖直方向或者水平方向上写，如果手抖得比较厉害，不适应空中书写，用户可以选择按在另一个手上写数字。

通常情况下的几种持手机书写姿势，如下表 4-2 中所示，有竖直方向写、水平方向写、按手面上写等。





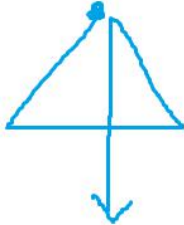
表 4-2 持手机书写姿势

持手机书写姿势	姿势图
竖直方向写	

水平方向写	
按在手上写	

对于数字的写法，我们进行了约定，用户无论使用哪种姿势，需要按照约定书写数字，这样能保证较高的准确率。数字的写法笔画如下表 4-3 中所示。

表 4-3 数字 0-9 笔画图

数字 0	数字 1	数字 2	数字 3	数字 4
				

数字 5	数字 6	数字 7	数字 8	数字 9
				

为了后面能够把连续书写的数字切割成每个数字为一段，数字与数字间有静止段，我们在收集数据时，就采用打标记的方式进行收集。具体地，手机端与电脑端通过 TCP/IP 的 Socket 进行通信，手机端连接上电脑端以后，书写时，每写完一个数字，就敲击一下键盘，代表静止段的开始，在写另一个数字前，再敲一次键盘，代表静止段的结束。数据由数字段和静止段交替构成，它们的分界就是敲击键盘打下的标记，如图 4-12 所示。

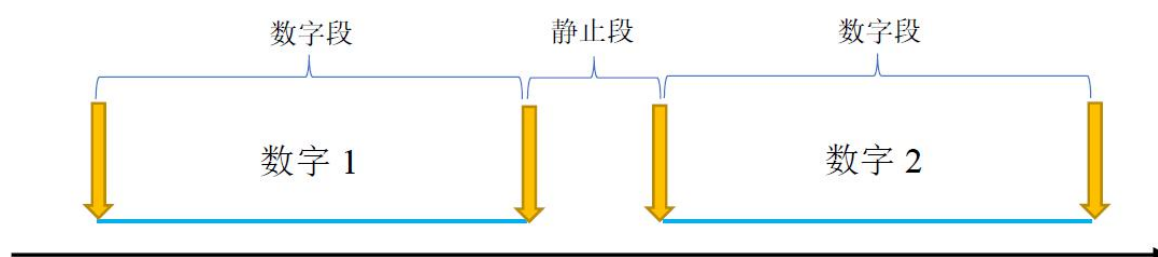


图 4-12 数字段和静止段

#### 4.4.2 数字切割算法

用户手写数字输入应该是一个连续的过程，不可能让用户一边写数字一边敲键盘，或者一次只能输入一个数字。我们的数据库中的数据都是有标记的，标记出了某段数据具体是数字段还是静止段。我们可以利用有标记的数据，来训练一个切割数字的模型，一段连续的用户书写数据作为输入，输出是单独的数字段。

具体地，我们采用打了标记的数据集，实现一个判断一段数据是属于静止段还是数据段的二分类器。常用的二分类器有基于神经网络的、基于提升树的和基于 SVM

的等。在前期实验的基础之上，我们选择的是支持向量机作为二分类器。

支持向量机是一种二分类模型，它的基本模型是定义在特征空间上的间隔最大的线性分类器，间隔最大使得它有别于感知机，支持向量机还包括核技巧，这使它成为实质上的非线性分类器，支持向量机的学习策略就是间隔最大化，可形式化为一个求解凸二次规划的问题，也等价于正则化的合页损失函数的最小值问题。支持向量机的学习算法是求解凸二次规划的最优化算法。

支持向量机学习方法包含构建由简至繁的模型：线性可分支持向量机、线性支持向量机及非线性支持向量机。当训练数据线性可分时，通过硬间隔最大化，学习一个线性的分类器，又称为硬间隔支持向量机；当训练数据近似线性可分时，通过软间隔最大化，也学习一个线性的分类器，即线性支持向量机，又称为软间隔支持向量机；当训练的数据线性不可分时，通过核技巧及软间隔最大化，学习非线性支持向量机，核函数往往取高斯径向基函数。

假设给定一个特征空间上的训练数据集：

$$T = \{(x_1, y_1), (x_2, y_2), \dots, (x_n, y_n)\}$$

其中， $x_i \in X = \mathbf{R}^n$ ， $y_i \in Y = \{+1, -1\}$ ， $i=1, 2, \dots, n$ ， $x_i$  为第  $i$  个特征向量， $y_i$  为  $x_i$  的类标记，假设训练数据集不是线性可分的，线性不可分意味着某些样本点  $(x_i, y_i)$  不能满足线性可分支持向量机的函数间隔大于等于 1 的约束条件。为了解决这个问题，可以对每个样本点引入一个松弛变量  $\xi_i \geq 0$ ，使得函数间隔加上松弛变量大于等于 1，这样，约束条件为：

$$y_i(w \cdot x_i + b) \geq 1 - \xi_i$$

同时，对每个松弛变量  $\xi_i$ ，支付一个代价  $\xi_i$ ，目标函数为：

$$\frac{1}{2} \|w\|^2 + C \sum_{i=1}^N \xi_i$$

这里， $C > 0$  称为惩罚函数，在训练中，我们选择  $C=1$ 。C 值大时对误分类的惩罚增大，C 值小时对误分类的惩罚减小。

线性不可分支持向量机的学习问题为如下凸二次规划问题：

$$\min_{w,b,\xi} \frac{1}{2} \|w\|^2 + C \sum_{i=1}^N \xi_i \quad (1)$$

$$s.t. \quad y_i(w \cdot x_i + b) \geq 1 - \xi_i, i = 1, 2, \dots, N \quad (2)$$

$$\xi_i \geq 0, i = 1, 2, \dots, N \quad (3)$$

原始问题(1)~(3)是一个凸二次规划问题，因而关于(w,b,ξ)的解是存在的，可以证明 w 的解是唯一的，但 b 的解可能不唯一，而是存在于一个区间。

设问题(1)~(3)的解是  $w^*$ ， $b^*$ ，于是可以得到分离超平面  $w^* \cdot x + b^* = 0$  及分类决策函数  $f(x) = \text{sign}(w^* \cdot x + b^*)$ 。比起线性可分支持向量机，线性支持向量机具有更广的适用性。

对于输入的连续数据，进行切割时，我们采用的是类似音频信号处理中的办法，先对数据进行  $n=5$  的移动平均处理，相当于低通滤波，然后进行分段，每 10 个采样点为 1 个窗，窗的标记为 0 表示处于静止段，为 1 则表示处于数字段。我们选取的特征是平均加速度变化率 avg\_Da，平均移动方差 avg\_Rv，平均加速度 avg\_Ac 及平均能量 avg\_En。使用这些特征，配合 SVM 分类器，能取得较好的数字段切割效果。特征四元组 Ft 表示为：

$$Ft = \{\text{avg\_Da}, \text{avg\_Rv}, \text{avg\_Ac}, \text{avg\_En}\}$$

其中，X 方向上时刻 i 的加速度变化率 Da 定义为 X 方向上时刻 i+1 的加速度与时刻 i 的加速度之差：

$$Da_x[i] = ACCx[i+1] - ACCx[i]$$

平均加速度变化率 avg\_Da 定义为窗长为 len 的窗内加速度变化率值取平均：

$$\text{avg\_Da} = \frac{\sum_{t=i}^{t=i+len} (Da_x[i] + Da_y[i] + Da_z[i])}{len}$$

移动方差则是需要对窗内的每个采样点计算一次方差，每次计算取的样点为 10



个，avg\_Rv 是三个方向上的移动方差的均值，计算方式类似上面的平均加速度变化率 avg\_Da。方差定义为：

$$Var = \frac{1}{len(len-1)} (len \sum_{i=1}^{len} x_i^2 - (\sum_{i=1}^{len} x_i)^2)$$

平均加速度 avg\_Ac 则是最直观的特征，是窗内加速度的均值，如果平均加速度很大，用户应该是在书写数字，否则认为用户持手机处于静止状态。

在语音信号处理中，能量等低阶特征往往能很容易地进行语音活动检测，而且准确率并不低，这里借鉴语音处理，采用平均的能量作为一个特征。下图 4-13 中，我们可以看到，能量特征和移动方差特征的确很容易将数字段和静音段区分出来，尤其是能量特征。在数字段中，三轴的能量都比较高，在静音段，也会有一定幅度的能量值，是因为用户手抖已经传感器噪声造成的，静音段的能量与数字段的能量相比，明显小了很多。

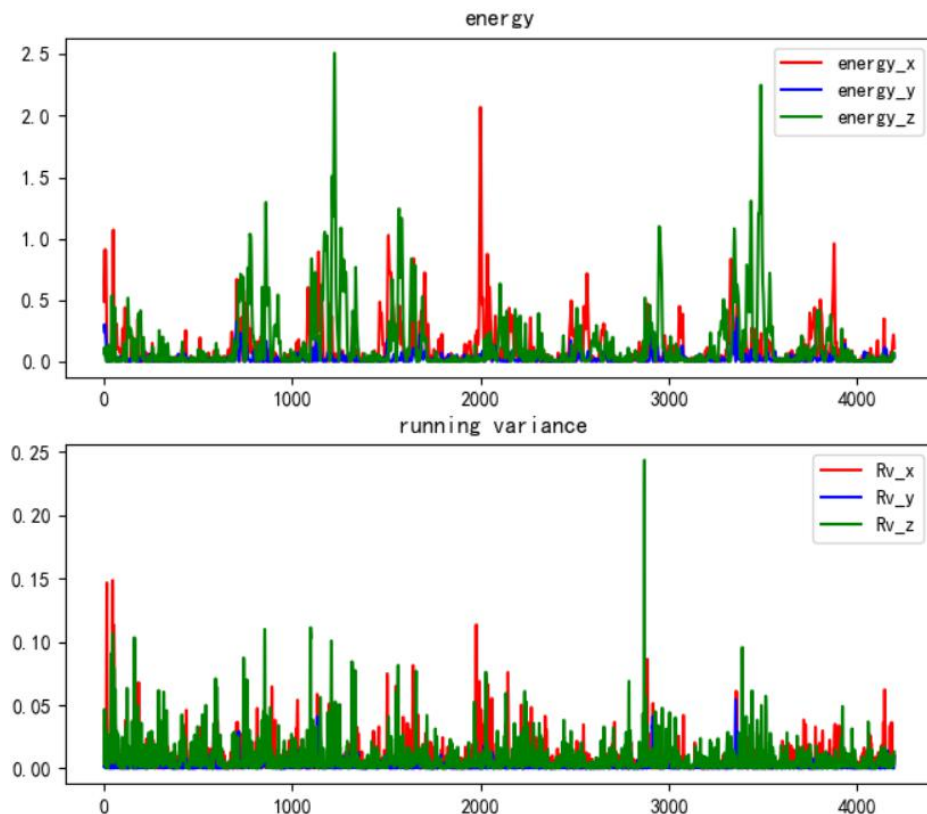


图 4-13 某段数据的能量和移动方差



在训练好切割数字的模型之后,对于新来的数据输入,只需要知道用户写了多少个数字,就能进行有效的切割,把数据分成窗长为 10 的窗,每个窗采用训练好的 SVM 切割模型进行预测,如果预测为 0 说明是静止段,如果预测为 1 说明是数字段。在 SVM 预测完之后,对预测的结果进行分析,如果是写了 10 个数字,取出最长的 10 段连续的静止段,标记为真的静止段,而其它的全部标记为数字段,就能够形成连续的 10 个数字段和静止段,之后可以把数字段依次送到数字识别模块进行数字识别,最终获取到数字识别的结果。在测试过程中,某次测试的结果如下图 4-14 所示,红色的是根据组员数据采集时敲击键盘进行的划分,蓝色的是使用 SVM 进行预测时候输出的结果,可以看到它们基本上是重合的,每个数字段内是连续的,没有出现两个数字段连起来或者错位等情况,说明效果比较好。这里是录了 10 个数字,最后一小段不算是输入,取前 10 段预测标记为 1 的数字段即可。

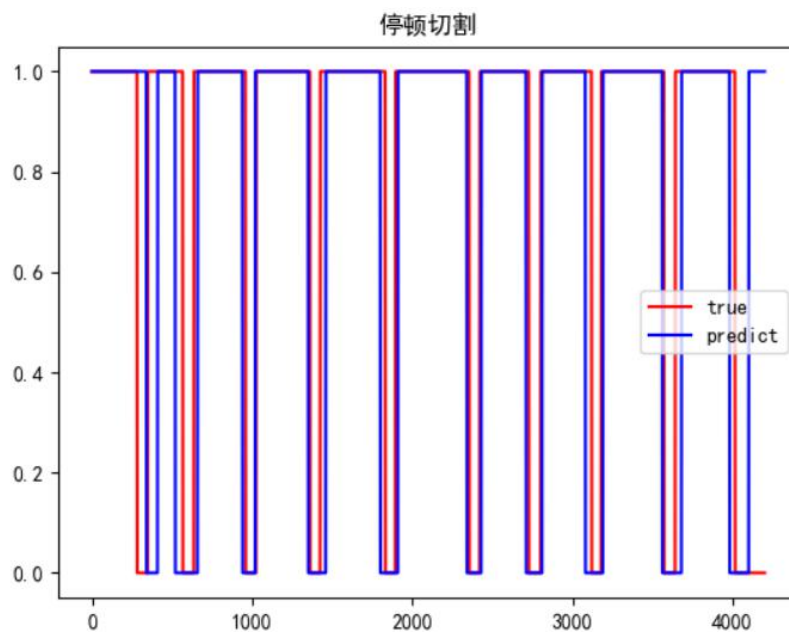


图 4-14 数字切割示意图

#### 4.4.3 数字可视化算法

对于手机的线性加速度传感器采集到的手机三轴上的线性加速度,我们可以通过一次积分,将加速度转化为速度,再一次对速度积分,将速度转化为位移。然后在

个二维平面上展示出每个时刻的位移点，这些点就构成了空中书写数字的轨迹。手机的三轴线性加速度为  $ax, ay, az$ ，速度为  $vx, vy, vz$ ，位移为  $dx, dy, dz$ ，以 X 轴方向上为例，因为是离散的数据，积分采用累加的方法，得到每个时刻的速度；再对速度做累加，得到每个时刻的位移。

$$\begin{aligned} vx_0 &= 0 \\ vx_i &= vx_{i-1} + ax_i (i > 0) \\ dx_0 &= 0 \\ dx_i &= dx_{i-1} + \frac{vx_i}{2} (i > 0) \end{aligned}$$

在实验中，我们尝试过，如果是在竖直平面上书写，使用手机的 X 轴和 Z 轴上的加速度二次积分，在二维平面上画出轨迹，但是我们发现有时候投影平面不是太准确。在这种情况下，我们发现，如果对三维轨迹使用 PCA 进行处理，之后再投影到二维平面上，有的时候会取得不错的效果。因此我们在做数字的轨迹可视化时，采用了 PCA。

PCA 是主成分分析，在机器学习中常用来做数据降维。PCA 是对原始数据进行投影，投影之后，使得投影后数据的方差最大，让数据更分散；同时，也是找出数据里最主要的方面，用数据里最主要的方面来代替原始数据，实现数据降维。

设原始特征有 N 个，为  $X = x_1, x_2, \dots, x_N$ ，进行 PCA 变换之后的特征为  $Z = z_1, z_2, \dots, z_N$ ， $z_i$  是原始特征的线性组合，对于每个原始特征权重是  $w_{ij}$ ，：

$$\begin{aligned} z_i &= \sum_{j=1}^p w_{ij} x_j = w_i^T x \\ w_i^T w_i &= 1 \end{aligned}$$

对于其中的  $z_1$  和  $z_2$ ，有：

$$\begin{aligned} var(z_1) &= \sum_{z_1} (z_1 - \bar{z}_1)^2 & ||w_1||_2 &= 1 \\ var(z_2) &= \sum_{z_2} (z_2 - \bar{z}_2)^2 & ||w_2||_2 &= 1 \\ w_1 \cdot w_2 &= 0 \end{aligned}$$

其中， $z_1$  的方差可以用  $x$  的协方差矩阵来表示：

$$\begin{aligned}
 var(z_1) &= \frac{1}{N} \sum_{z_1} (z_1 - \bar{z}_1)^2 \\
 &= \frac{1}{N} \sum_x (w_1 \cdot x - w_1 \cdot \bar{x})^2 \\
 &= \frac{1}{N} \sum_x (w_1) \cdot (x - \bar{x})^2 \\
 &= \frac{1}{N} \sum_x (w_1)(x - \bar{x})(x - \bar{x})^T w_1 \\
 &= (w_1)^T \frac{1}{N} \sum_x (x - \bar{x})(x - \bar{x})^T w_1 \\
 &= (w_1)^T Cov(x) w_1
 \end{aligned}$$

PCA 要做的，是在约束条件下寻找使得  $z_1$  方向上方差最大的  $w_1$ :

$$\begin{aligned}
 def \quad S &= Cov(x) \\
 w_1^* &= \arg \max_{w_1} (w_1)^T S w_1 \\
 s.t. \quad ||w_1||_2 &= (w_1)^T w_1 = 1
 \end{aligned}$$

使用拉格朗日乘子法求解，求导之后可以得到，实际上要寻找  $w_1$  使得方差最大，就是寻找最大的  $\alpha$ 。也就是说， $w_1$  是协方差矩阵  $S$  的最大的特征值对应的特征向量。

$$\begin{aligned}
 g(w_1) &= (w_1)^T S w_1 - \alpha((w_1)^T w_1 - 1) \\
 S w_1 - \alpha w_1 &= 0 \\
 (w_1)^T S w_1 &= \alpha (w_1)^T w_1 \\
 \implies max \quad \alpha
 \end{aligned}$$

同理，在求解  $w_2$  时，多一个约束条件，即  $w_2$  要与  $w_1$  正交:

$$\begin{aligned}
 w_2^* &= \arg \max_{w_2} (w_2)^T S w_2 \\
 s.t. \quad ||w_2||_2 &= (w_2)^T w_2 = 1 \\
 (w_2)^T w_1 &= 0
 \end{aligned}$$

通过拉格朗日乘子法，可以求导之后进行整理，得到与前面类似的结论， $w_2$  应该是  $x$  的协方差矩阵的第 2 大的特征值对应的特征向量。

$$\begin{aligned}
 g(w_2) &= (w_2)^T S w_2 - \alpha((w_2)^T w_2 - 1) - \beta((w_2)^T w_1 - 0) \\
 \implies S w_2 - \alpha w_2 - \beta w_1 &= 0 \\
 (w_1)^T S w_2 - \alpha(w_1)^T w_2 - \beta(w_1)^T w_1 &= 0 \\
 \implies ((w_1)^T S w_2)^T &= (w_2)^T S^T w_1 \\
 \because S w_1 &= \lambda w_1 \\
 (w_2)^T S w_1 &= \lambda_1 (w_2)^T w_1 = 0 \\
 S w_2 - \alpha w_2 &= 0 \implies S w_2 = \alpha w_2
 \end{aligned}$$

重复上面的方法, 就可以依次得出由相应特征值对应的特征向量构成的主成分  $Z$ 。在应用中, 我们可以根据需求, 提取前  $k$  个主成分。在空中手写数字可视化中, 我们需要提取的是 2 个主成分。

具体做法是, 首先, 通过两次积分得到位移坐标, 是一个三维的列向量, 坐标系为  $x, y, z$ 。点集为  $P$ , 变换之后的点集为  $Q=CP$ , 变换矩阵是  $C$ 。在求取了  $P$  的协方差矩阵之后, 可以求出特征值和特征向量, 特征向量对应于新的直角坐标系  $x', y', z'$ 。其中最小的特征值对应的特征向量为  $z'$ ,  $x'$  和  $y'$  的平面就是投影平面, 从  $z'$  方向往  $x'$  和  $y'$  的平面上看, 数字最正。但是不能直接把  $x'$  和  $y'$  作为可视化的方向, 否则数字可能会倒过来, 原先  $P$  中  $x$  和  $y$  的方向是正确的。所以应该将  $x'$  方向取为与原  $z$  和新  $z'$  同时垂直的方向,  $y'$  同时取为与原  $x'$  和  $z'$  同时垂直的方向, 这样字符的方向也是正确摆放的。这一过程如下图 4-15 所示:

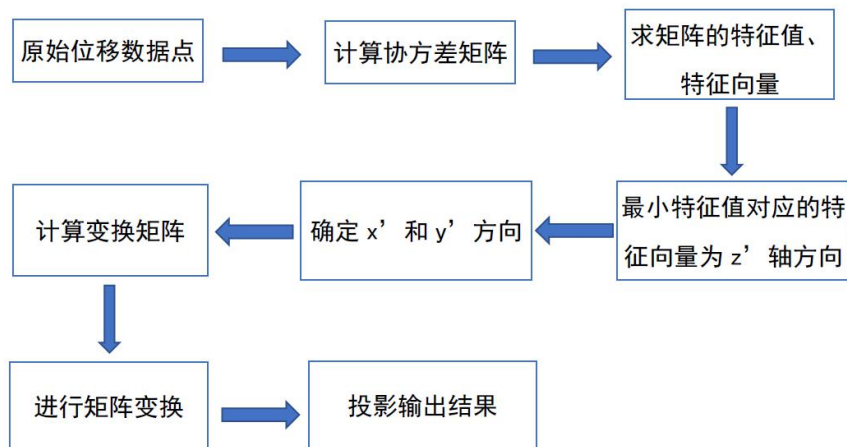
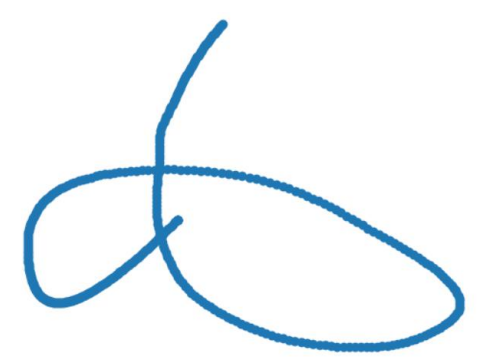
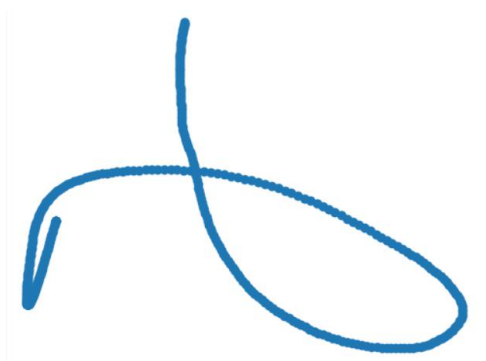
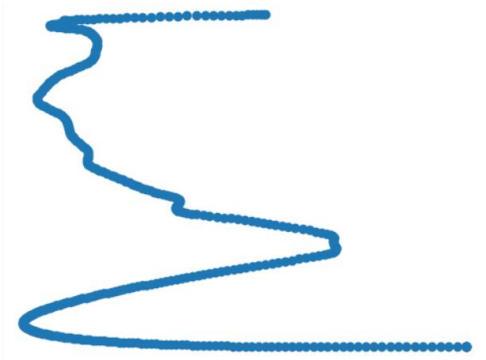
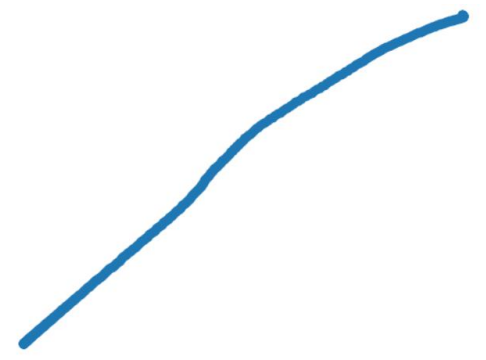
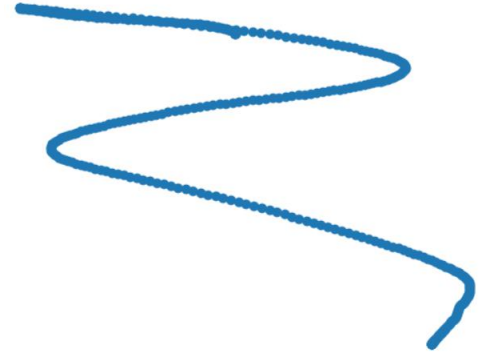

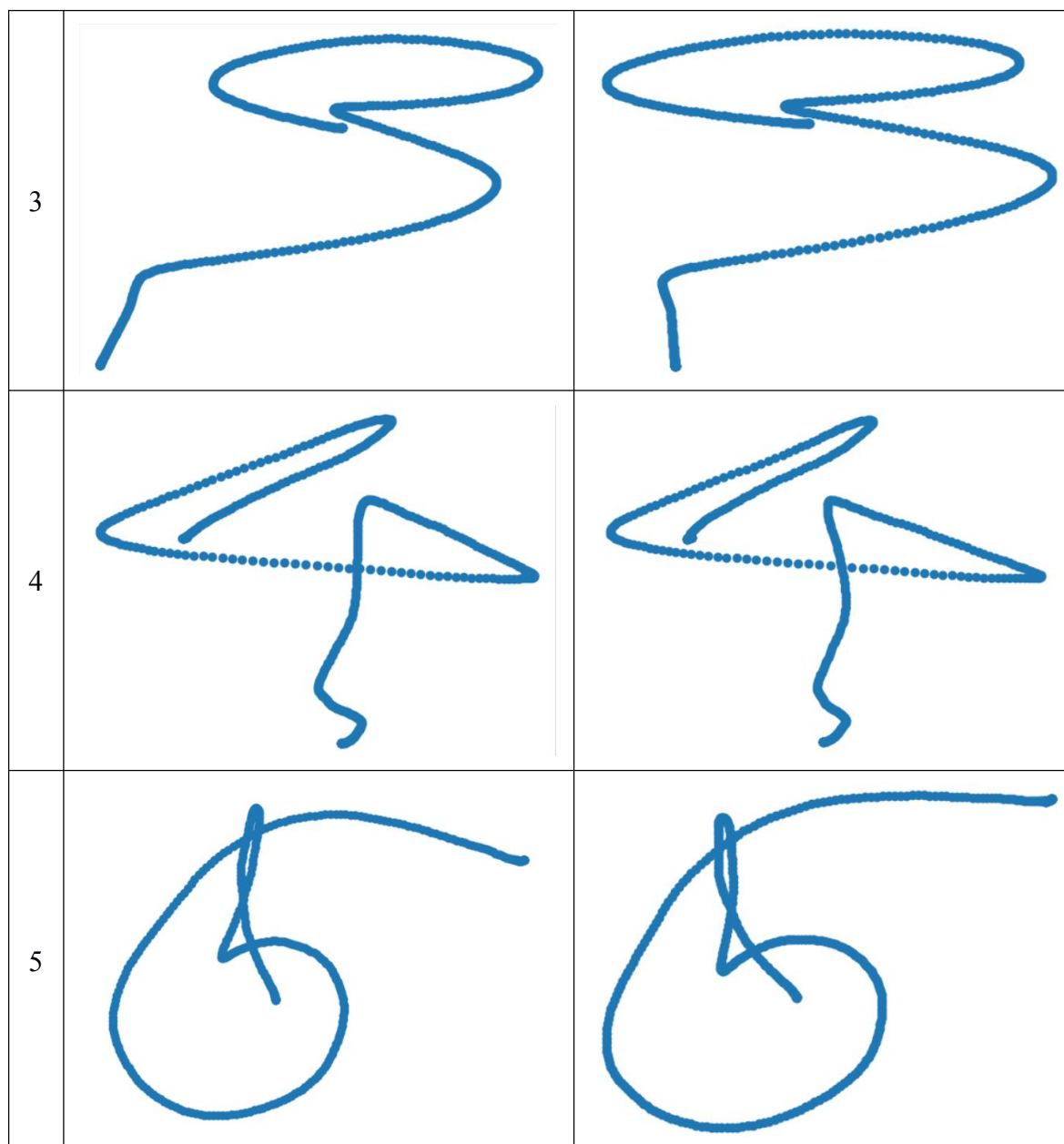


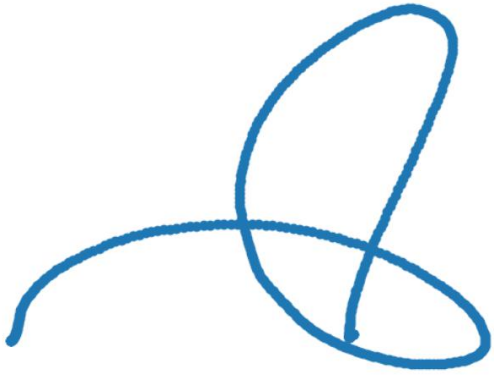
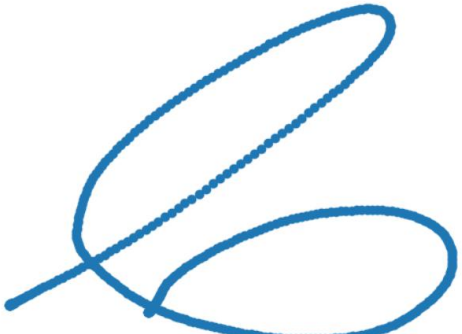

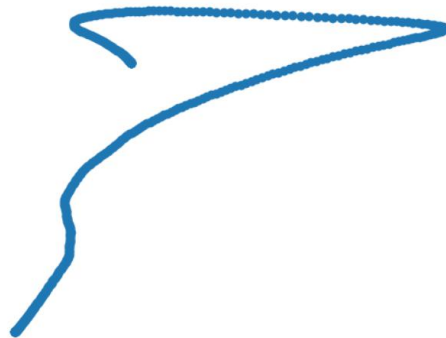
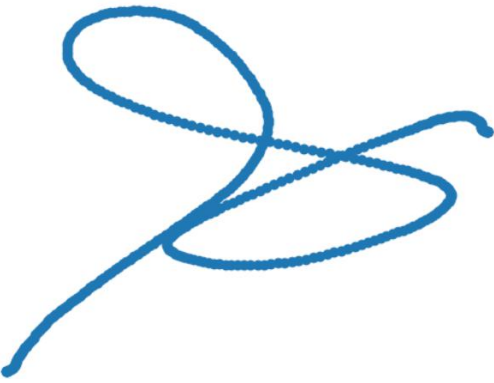
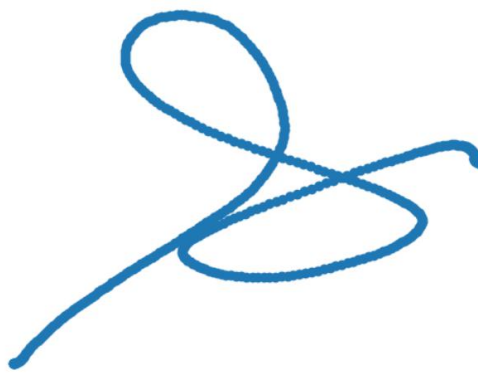


图 4-15 数字可视化算法

使用 PCA 后，一些数字的可视化效果更好例如数字 1 在没有经过 PCA 处理时，是比较乱的轨迹，经过投影之后，轨迹明显能看出来是 1，数字 6 和 9 看上去也显得更正了。少量数字的效果没有明显变化，如下表 4-4 所示，可见我们对轨迹的处理是有效的，也还有进一步提升的空间。

表 4-4 数字可视化效果表

数字	PCA 处理前轨迹	PCA 处理后轨迹
0		
1		
2		



6		
7		
8		
9		



#### 4.4.4 数字识别算法

在完成数字切割之后，用户输入的连续的数据，变成了相互独立的数字段，可以对每个数字段进行数字识别。数字识别算法上，我们采取的是基于深度学习的 CNN-BLSTM 模型。

CNN 卷积神经网络是多层感知机的变种，通过卷积和池化操作，可以提取到局部特征，也能够实现数据的降维，加速训练过程。

对于序列标注问题的处理，以前往往是采用 RNN，而 LSTM 是 RNN 的一个变种，继承了大部分 RNN 模型的特性，同时解决了梯度反向传播过程中的梯度消失问题。BLSTM 是由前向 LSTM 与后向 LSTM 组合而成，利用 LSTM 对时间序列进行建模存在一个问题，无法编码从后到前的信息，而通过 BLSTM 可以更好的捕捉双向的信息，提升模型的性能。

一个 LSTM 单元如下图 4-15 所示，包含输入门(Input Gate)、输出门(Output Gate)和遗忘门(Forget Gate)<sup>[11]</sup>。

- Input Gate, Output Gate, Forget Gate: 这三个 Gate 本质上就是权值，类似电路中用于控制电流的开关。当值为 1，表示开关闭合，流量无损耗流过；当值为 0，表示开关打开，完全阻塞流量；当值介于(0,1)，则表示流量通过的程度。而这种[0,1]的取值，其实就是通过 Sigmoid 函数实现的。
- Cell: Cell 表示当前 Memory Block 的状态，对应于原始 RNN 中的 Hidden Layer 的神经元。
- Activation Function: 一般，对 Input Gate, Output Gate, Forget Gate，使用的 Activation Function 是 sigmoid 函数；对于 Input 和 Cell，Activation Function 使用 tanh 函数。



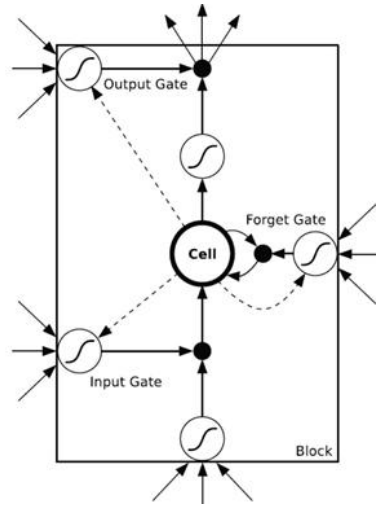


图 4-15 LSTM 单元示意图

在 LSTM 中用到的激活函数有 sigmoid 函数和 tanh 函数，在全连接层中会用到 ReLU 函数，它们分别为：

$$\begin{aligned} \text{sigmoid}(x) &= \frac{1}{1 + e^{-x}} \\ \text{tanh}(x) &= \frac{e^x - e^{-x}}{e^x + e^{-x}} \\ \text{ReLU}(x) &= \begin{cases} x & \text{if } x > 0 \\ 0 & \text{if } x \leq 0 \end{cases} \end{aligned}$$

我们的模型结构如下图 4-16 所示，首先是 CNN 的卷积层，经过一个 dropout 层之后是最大池化层，没有按照常见的 CNN 使用压平层 Flatten，而是把输出送入到 BLSTM 层，BLSTM 比 LSTM 更能捕捉到时序特征，BLSTM 层的输出送到全连接层，之后经过 softmax 送到输出层，损失函数使用交叉熵，训练时使用 Adam 优化，总共大约训练了 100 个 epoch，每个阶段训练完，需要对学习率进行一定的调整，然后再加载权重训练。

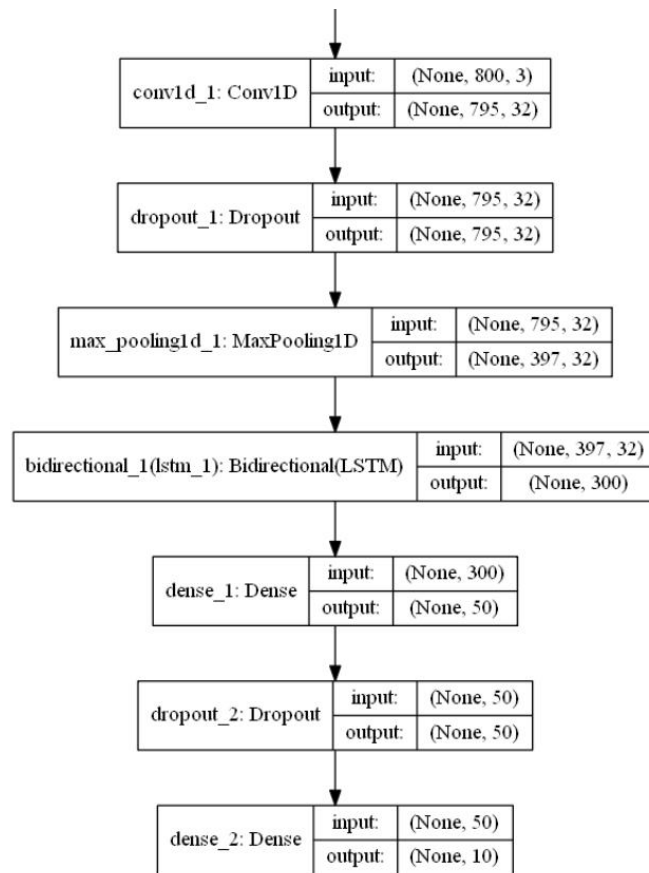


图 4-16 CNN-BLSTM 模型架构图

首先是模型的训练阶段，我们选取采集到的数据库作为训练集，里面有约 3000 个数字，按照 7:3 的比例划分训练样本和测试样本，训练样本中再按照 7:3 的比例划分为训练集和验证集。将输入的数字段的三轴加速度，分别作为时间序列的三个特征，数据长度作为 `time_step` 输入到模型中，当 `loss` 降到比较低及准确率比较高时，停止训练。否则，进行在线训练，使用较低的学习率，结合 Adam 优化，使得模型学习到较好的参数。一次训练过程中的损失如图 4-17(a)所示，准确率如图 4-17(b)所示，可以看到，仅仅迭代了 5 次左右，准确率就接近 90%，说明我们的模型设置比较合理。

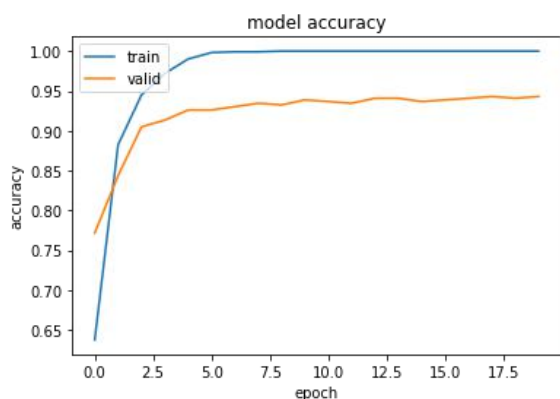


图 4-17(a)

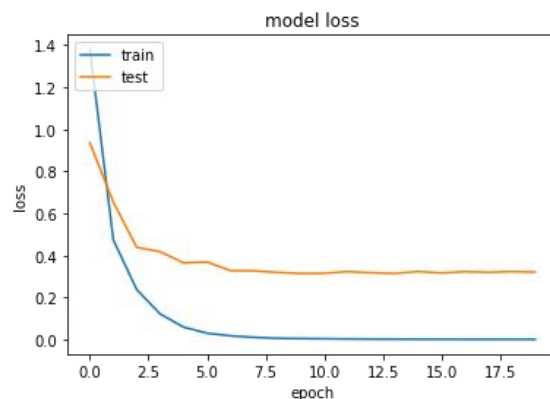


图 4-17(b)

在一次训练结束后，调低学习率，再次训练，直到验证集上的 loss 不再下降，训练集上拟合得较好。在训练阶段结束后，测试阶段，我们使用测试集对模型的准确率进行测试，测试结果如下图 4-19 所示，数字识别准确率为 92.96%，可见我们的模型是非常有效的，取得了非常好的效果，在应用中可以调用模型，应用于各种场景。

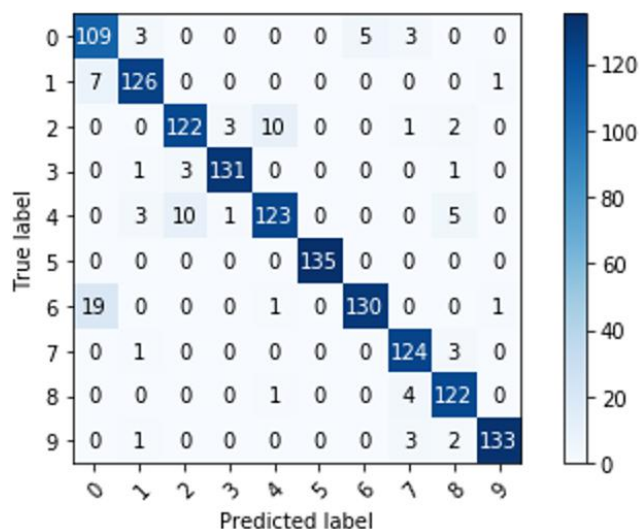


图 4-19 测试集上混淆矩阵

## 参考文献

- [1] DEAN TAKAHASH. Newzoo: Smartphone users will top 3 billion in 2018, hit 3.8 billion by 2021[EB/OL].<https://venturebeat.com/2018/09/11/newzoo-smartphone-users-will-top-3-billion-in-2018-hit-3-8-billion-by-2021/> . 2018,11
- [2] Majumder, Sumit & Deen, M.J.. (2019). Smartphone Sensors for Health Monitoring and Diagnosis[J]. Sensors. 19. 2164.
- [3] 绿盟科技. 浅谈 Web 安全验证码[EB/OL]. <http://blog.nsfocus.net/discussion-web-security-authentication-code/> .2017
- [4] 张立新.多种类型验证码的研究与分析[J].福建电脑,2016,32(10):76-76,125.
- [5] 周正,文亚飞,鲍文平.基于深度学习的人工智能用于识别破解字符型验证码[J].通信技术,2017,50(11):2572-2576
- [6] 何福泉,李伟烽,林培娜,等.验证码的识别技术分析与研究[J].甘肃科技纵横,2019,48(2):1-4,22
- [7] Google. webmaster central blog[EB/OL]. <https://webmasters.googleblog.com>. 2019, 6
- [8] Jake Kenny. Google 的 reCAPTCHA 不再强大, 安全专家从中发现漏洞[EB/OL]. <https://www.kaspersky.com.cn/blog/googles-recaptcha-defeated-by-security-researchers/4144/> . 2016
- [9] 薛命灯. unCaptcha: 准确率 85%的谷歌语音验证码破解工具[EB/OL]. <https://www.infoq.cn/article/2017/11/unCaptcha-85-Speech-verification> . 2017
- [10] Google.sensor-types[EB/OL]. <https://source.android.com/devices/sensors/sensor-types>
- [11] 余文毅 . 当我们在谈论 Deep Learning: RNN 其常见架构[EB/OL]. <https://zhuanlan.zhihu.com/p/27485750>. 2017