

大连理工大学

专业方向课程设计大作业

---

# 跨域手势识别的研究

---

姓名: 金 田

班级: 网安 1901

学号: 201992268

2022 年 9 月 2 日



大连理工大学

DALIAN UNIVERSITY OF TECHNOLOGY

# 目录

<b>1</b>	<b>课题背景</b>	<b>2</b>
1.1	传统模式和发展现状	2
1.2	弊端及创新点	2
<b>2</b>	<b>环境部署</b>	<b>3</b>
2.1	工具	3
2.1.1	Anaconda	3
2.1.2	Emacs	3
2.1.3	LaTeX	6
2.2	环境	7
2.2.1	pytorch	7
2.2.2	tensorflow(keras)	8
2.3	数据集	8
2.3.1	采集设备	9
2.3.2	采集过程	10
<b>3</b>	<b>实验操作</b>	<b>12</b>
3.1	环境版本	12
3.2	硬件支持	12
3.3	代码实现与调整	13
<b>4</b>	<b>总结与心得</b>	<b>16</b>
4.1	收获	16
4.2	遇到的问题	16
4.3	对发展前景的展望	16

# 1 课题背景

## 1.1 传统模式和发展现状

在智能技术领域，人们持续致力于探索与智能设备更简便更精准的交互方式。在如今人机交互的方式越来越多样化的趋势下，人体姿态和手势的识别成为一个重要发展方向。

手势是人类非语言类型中通俗易懂的一种表达形式，也可以作为人机交互的一种方式。在早期的智能家居模式下，已经可以实现用简单手势来操作一些电器和办公设备。在现有解决方案下已经有很多手势识别方式，其应用场景也各有不同，下面列举了三种典型的手势识别方法。

第一种是借助专门的传感器设备来获取信息，并建立相应的手势模型来进行手势识别。这种方式需要用户佩戴专门的设备进行数据提取，通过分析图像、声音等信息，借助已有的分类模型来分辨其行为。然而佩戴设备无疑增大了设备的使用成本，极大地降低了方便性。

第二种是基于视频的手势识别，即利用摄像机来采集手动作的视频，然后再根据视频数据提取手势特征完成手势识别。这种方法也有其局限性，容易受到光照、角度、分辨率等因素的影响，降低准确度，另外，视频数据的不确定性有时会涉及隐私问题，引起不必要的纠纷。

第三种是基于无线射频的手势识别，也是最接近于目前主流技术的手势识别方法，其过程主要是在特定设备上获取手势动作的信号，对数据进行预处理后对手势进行分类。目前该方向的负面作用较小，有利于持续的研究发展。

随着无线通信技术的快速发展，WIFI 信号也为手势识别的数据来源带来了一种可能。WIFI 信号在空间传播的过程中会由于用户的手势动作而产生多径传播的叠加效应，可以借助 WIFI 信号的波动情况识别出用户的手势动作。在利用 WIFI 信号进行手势识别时，对用户本身没有额外的要求，既不需要佩戴各种智能设备，也不会受到温度、光照条件的影响。目前基于 WIFI 的智能设备已经在实际生活和办公场景中广泛使用，这为 WIFI 环境下的手势识别信息研究提供了基础，通过 WIFI 信号来进行手势识别已经成为无线感知的热门技术之一。

## 1.2 弊端及创新点

在早期的基于无线感知技术的手势识别是从 WIFI 信号中提取物理特征并进行处理映射到手势动作，这些从 WIFI 信号直接提取到的特征包括了大量的环境因素，即会干扰正常手势引起信号波动的数据信息，这些信息会混杂在 WIFI 信号提取的信息中难以提取出来，一方面给纯粹的手势识别处理带来困难，另一方面映射的手势动作只局限于当时所处的实验环境，包括所处空间的结构、用户做手势的位置和方向，都会对结果产生较大影响。所以一旦设备所处的环境改变，就会面临原本功能失效的结果。在业界中，把与手势特征无关的因素（位置、信号强弱、空间分辨）成为“域”。各个科研团队持续研究的重点和难点是如何尽可能地消除环境给手势识别带来的影响，也就是让手势识别模型尽可能在不同的域之间保持稳定的识别准确率。

因此在手势识别的研究中，最大的创新点在于从原始的与域相关的信号中提取仅反映手势本身与域无关的特征。近年来也有不少团队研究了一些成果，2017 年，Chen 等人通过 online-independent SVM (OISVM) 算法，SVM 的参数，识别出环境变化后的动作。2018 年，Yang 等人提出的 FALAR 系统实现了自适应位置无关的活动识别，该系统利用基于核密度估计的运动提取方法和一种从粗到细的搜索策略来进行快速识别，无需考虑周围的环境。2019 年，也是手势识别领域中具有飞跃性的一个阶段，清华大学团队设计了基于 WIFI 的跨域手势识别系统 Widar，在较低信号水平上导出估计手势的速度曲线并形成 BVP 文件，这些数据表示手势的独特动力学特性，与环境并无关联，只需训练一次即可准确识别不同环境中的手势动作。该手势识别系统也作为本次课程报告的研究重点。

## 2 环境部署

在这一部分，将列举本次实际操作用到的一些工具的使用情况、环境配置以及数据集的详细信息。

### 2.1 工具

#### 2.1.1 Anaconda

随着深度学习算法的发展，衍生出了许多具有稳定架构的深度学习框架。现有的深度学习框架能够大幅度降低初学者的学习成本，通过实践发现，这些深度学习框架往往对编程语言和依赖包的版本有所要求。在大多数情况下，在某一环境下的深度学习框架不能在其他训练要求下生效，因此在执行不同的学习任务时，需要频繁地进行版本切换。

Anaconda 工具的出现就很好地解决了这个问题，Anaconda 是一个提供 Python 开发所需的工具包，包括 Python / IPython / CPython / Conda 等 180 个依赖项，同时支持 Linux，Mac，Windows 三大平台。有了 Anaconda，依赖包的安装会简便许多。然而使用此工具的核心在于：Anaconda 能够创建虚拟环境，可以针对不同的学习任务配置相应的包的版本。比如在一个学习任务中要求编程语言 python 3.7，而另一个学习任务中要求 python 的版本为 3.5，并且版本之间相互不兼容。在传统做法上，我们通常会修改现有的环境版本，这样做不仅过程十分繁琐，还可能因考虑不周出现依赖包残余，影响下一步的学习任务。有了 Anaconda，只需要创建两个虚拟环境，编程语言和依赖包设置两个任务要求的版本，并且两个虚拟环境不存在冲突的情况，更有利于多学习任务的执行。下图展示了 Anaconda 安装之后的界面，表面上与 cmd 控制面板一致，同时会多一些 conda 的控制命令。

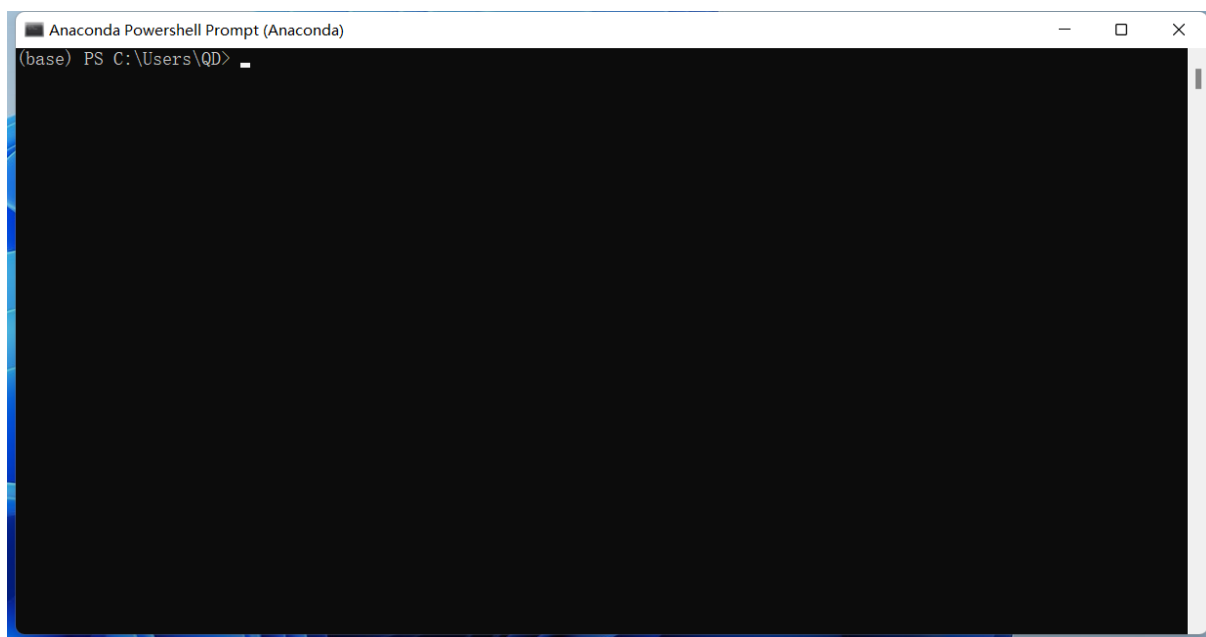


图 1: Anaconda 界面

#### 2.1.2 Emacs

Emacs 是小学期课程任务的重点，也是本次实践操作最常用的书写、编码工具。Emacs 本质上和 vim 的定位一样，是一个古老而功能强大的专业编辑器。在 Emacs 的编辑器系列分支中，其中最主流的一支是 GNU Emacs，本次使用的 Emacs 也是指 GNU Emacs。

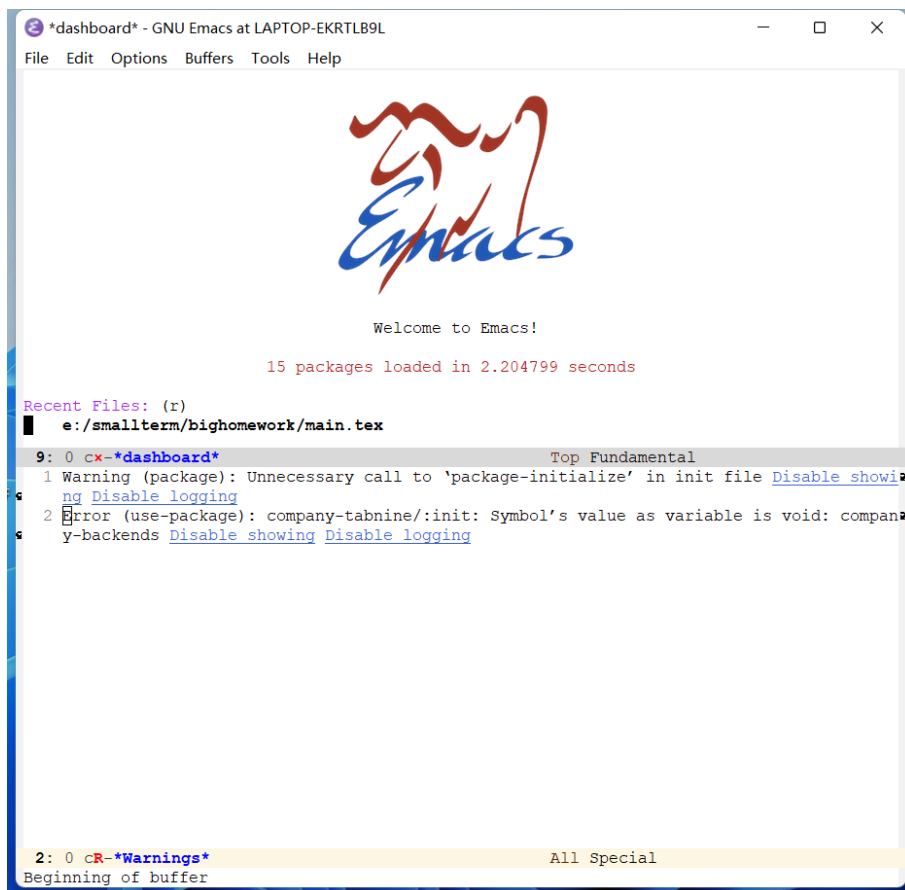


图 2: GNU Emacs 初始界面

Emacs 能够在当前大多数操作系统上运行，包括类 Unix 系统（GNU/Linux、各种 BSD、Solaris、AIX、IRIX、Mac OS X 等等）、MS-DOS、Microsoft Windows 以及 OpenVMS 等，还有移动 Android 平台以及 iOS。Emacs 依靠键盘的快捷键可以实现对文本、文件、代码的任意操作，图中整合了 Emacs 的常用操作，其中绝大多数的指令已经在实际操作中使用到。

操作描述	快捷键	命令名
输入命令	M-x	execute-extended-command
退出程序	C-x C-c	save-buffers-kill-terminal
放弃当前输入	C-g	keyboard-quit
光标向上一行 (方向键上)	C-p	previous-line
光标向下一行 (方向键下)	C-n	next-line
光标向左一个字符 (方向键左)	C-b	backward-char
光标向右一个字符 (方向键右)	C-f	forward-char
光标向左移动一个词	M-b	backward-word
光标向右移动一个词	M-f	forward-word
光标移至行首	C-a	move-beginning-of-line
光标移至行尾	C-e	move-end-of-line
光标移动到一行缩进的开头	M-m	back-to-indentation
光标移至句首	M-a	backward-sentence
光标移至句尾	M-e	forward-sentence
光标移至文件开头	M-<	beginning-of-buffer
光标移至文件结尾	M->	end-of-buffer
光标移动到窗口的中间、最上、最下	M-r	move-to-window-line-top-bottom
删除光标右侧字符	C-d	delete-char
移除光标右侧词	M-d	kill-word
移除光标左侧词	M-	backward-kill-word
移除右侧直到句子结尾	M-k	kill-sentence
移除右侧直到行尾	C-k	kill-line
设置标记以选择区域	C-SPC	set-mark-command
复制区域	M-w	kill-region-save
移除区域	C-w	kill-region
插入已移除文本	C-y	yank
插入历史移除文本	M-y	yank-pop
撤回	C-/ 或 C-_ 或 C-x u	undo
跳转到上一标记	C-x C-SPC 或 C-u C-SPC	pop-global-mark
跳转到行号	M-g M-g	goto-line
重键	C-u	universal-argument
向下一页	C-v	scroll-up-command
向上一页	M-v	scroll-down-command
移动页面使得光标在中央/最上方/最下方	C-l	recenter-top-bottom
向后搜索	C-s	isearch-forward
向前搜索	C-r	isearch-backward
交换前后字符	C-t	transpose-chars
交换前后词	M-t	transpose-words
交换前后两行	C-x C-t	transpose-lines
在下方新建一行	C-o	open-line
删除连续空行为一个空行	C-x C-o	delete-blank-lines
将后面的词变为小写	M-l	downcase-word
将后面的词变为大写	M-u	upcase-word
将后面的词变为首字母大写	M-c	capitalize-word
简要描述快捷键功能	C-h c	describe-key-briefly
描述快捷键功能	C-h k	describe-key
描述函数功能	C-h f	describe-function
描述变量	C-h v	describe-variable
列出含某一关键词的命令	C-h a	apropos-command
列出含某一关键词的符号的文档	C-h d	apropos-documentation
帮助的帮助	C-h ?	help-for-help

编辑于 2022-08-08 19:50

图 3: Emacs 快捷键

下图展示了在学习 Emacs 的一些使用场景。

操作描述	快捷键	命令名
下拉菜单栏	<f10>	menu-bar-open
互动菜单栏	M-`	tmm-menubar
打开文件	C-x C-f	find-file
保存文件	C-x C-s	save-buffer
打开并只读文件	C-x C-r	find-file-read-only
打开另一相近文件	C-x C-v	find-alternate-file
只读模式	C-x C-q	read-only-mode
切换到 Buffer	C-x b	switch-to-buffer
列出 Buffer	C-x C-b	list-buffers
关闭 Buffer	C-x k	kill-buffer
鼠标列出 Buffer	C-mouse-1	mouse-buffer-menu
上下分割出 Window	C-x 2	split-window-below
左右分割出 Window	C-x 3	split-window-right
关闭当前 Window	C-x 0	delete-window
只保留当前 Window	C-x 1	delete-other-windows
切换到另一 Window	C-x o	other-window
在另一 Window 中打开文件	C-x 4 f	find-file-other-window
在另一 Window 中切换 Buffer	C-x 4 b	switch-to-buffer-other-window
在另一 Window 中打开目录	C-x 4 d	dired-other-window
创建新的 Frame	C-x 5 2	make-frame-command
在另一 Frame 中打开文件	C-x 5 f	find-file-other-frame
让另一 Window 向下翻页	C-M-v	scroll-other-window
让另一 Window 向上翻页	C-M-S-v	scroll-other-window-down

编辑于 2022-08-08 19:50

图 4: Emacs 快捷键续

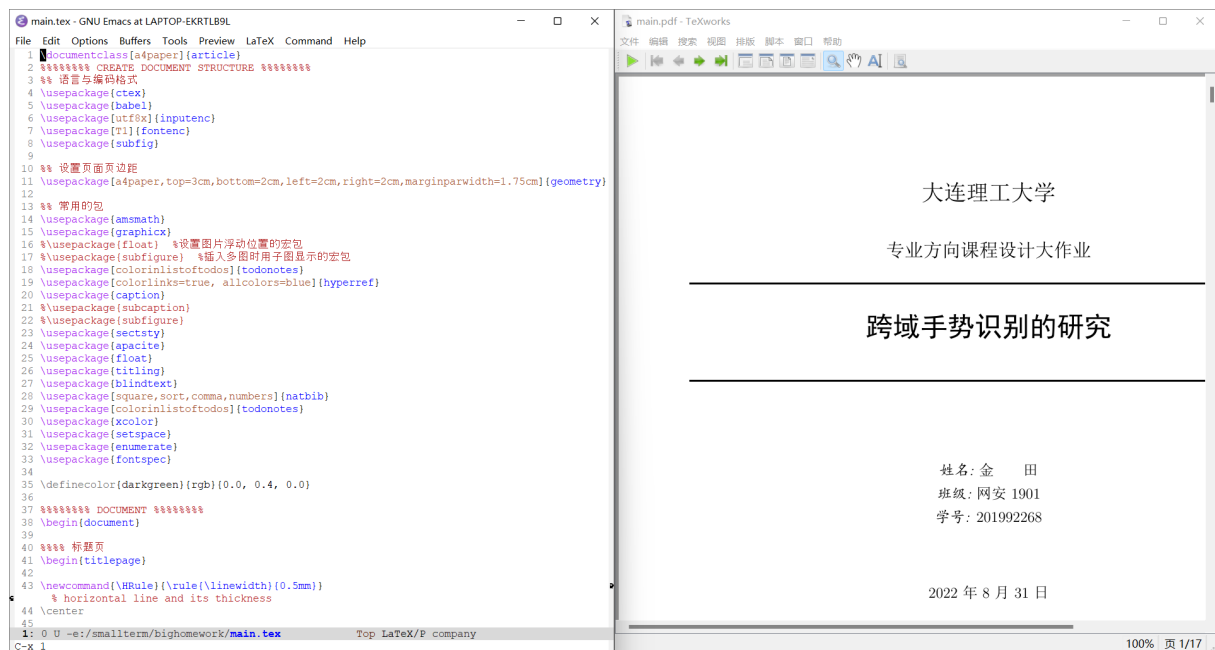
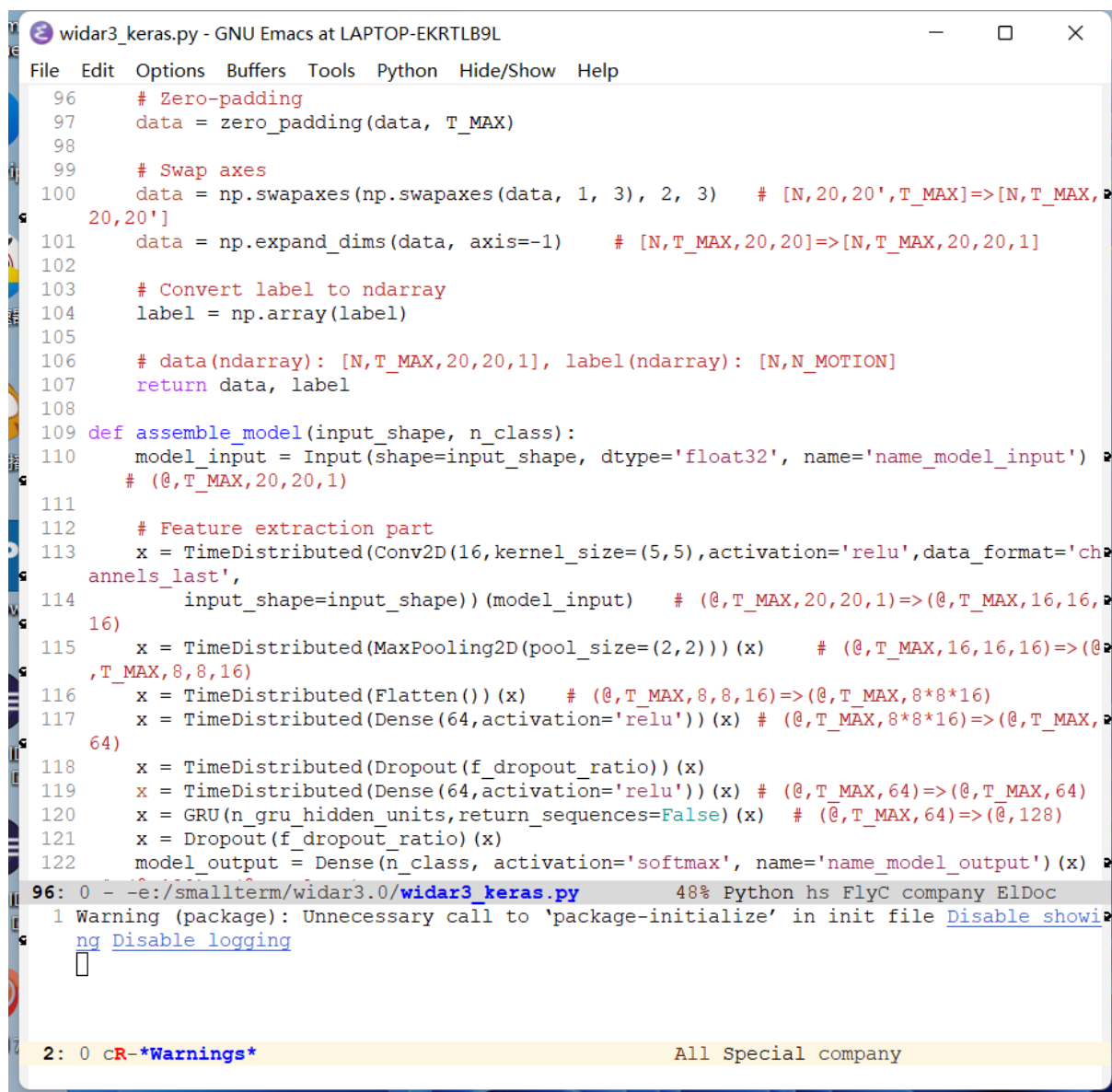


图 5: Emacs Latex 的桌面布置



```
96 # Zero-padding
97 data = zero_padding(data, T_MAX)
98
99 # Swap axes
100 data = np.swapaxes(np.swapaxes(data, 1, 3), 2, 3) # [N,20,20',T_MAX]=>[N,T_MAX,
20,20']
101 data = np.expand_dims(data, axis=-1) # [N,T_MAX,20,20]=>[N,T_MAX,20,20,1]
102
103 # Convert label to ndarray
104 label = np.array(label)
105
106 # data(ndarray): [N,T_MAX,20,20,1], label(ndarray): [N,N_MOTION]
107 return data, label
108
109 def assemble_model(input_shape, n_class):
110     model_input = Input(shape=input_shape, dtype='float32', name='name_model_input')
111     # (N,T_MAX,20,20,1)
112     # Feature extraction part
113     x = TimeDistributed(Conv2D(16,kernel_size=(5,5),activation='relu',data_format='ch
annels_last',
114     input_shape=input_shape))(model_input) # (N,T_MAX,20,20,1)=>(N,T_MAX,16,16,
16)
115     x = TimeDistributed(MaxPooling2D(pool_size=(2,2)))(x) # (N,T_MAX,16,16,16)=>(N,
,T_MAX,8,8,16)
116     x = TimeDistributed(Flatten())(x) # (N,T_MAX,8,8,16)=>(N,T_MAX,8*8*16)
117     x = TimeDistributed(Dense(64,activation='relu'))(x) # (N,T_MAX,8*8*16)=>(N,T_MAX,
64)
118     x = TimeDistributed(Dropout(f_dropout_ratio))(x)
119     x = TimeDistributed(Dense(64,activation='relu'))(x) # (N,T_MAX,64)=>(N,T_MAX,64)
120     x = GRU(n_gru_hidden_units,return_sequences=False)(x) # (N,T_MAX,64)=>(N,128)
121     x = Dropout(f_dropout_ratio)(x)
122     model_output = Dense(n_class, activation='softmax', name='name_model_output')(x)
96: 0 - -e:/smallterm/widar3.0/widar3_keras.py 48% Python hs FlyC company ElDoc
1 Warning (package): Unnecessary call to 'package-initialize' in init file Disable showi
ng Disable logging
2: 0 cR-*Warnings* All Special company
```

图 6: Emacs python 代码编写

Emacs 本身能实现的功能已经十分丰富，加上各个专业人员和编辑爱好者开发的插件，使 Emacs 强大的扩展性得以体现，这一古老的专业编辑器直到现在还焕发活力、经久不衰。在官方文档和其他参考资料中提到的功能优化、功能增强、界面美化、编程开发类的插件已达上千种，所以可以根据自己的喜好及功能需要选取合适的插件增加使用体验。

在本课程中，本人安装了一些必要的插件如下：

- use-package，一个管理其它插件的插件，这样在安装新插件的时候，可以借助 use-package 功能下的 Lisp 语言实现自动安装。
- column-number-mode，在编辑器上显示列号，这是后续编写代码的必备插件，在深度学习模型代码报错时，通过列号能够快速找到代码的错误点，从而进行修改。
- flycheck，一个在编程模式下的代码语法检查工具，有了 use-package，可以直接用 hook 模式设置。
- \* dashboard，一个新的欢迎界面，可以列出最近打开的项目、最近打开的文件等等。按下 p 或 r 就可



以快速跳转到相应小结里。还可以列出来标记过的书签、org-mode (Emacs 自带的一个强大的笔记系统) 日程、自定义控件等。

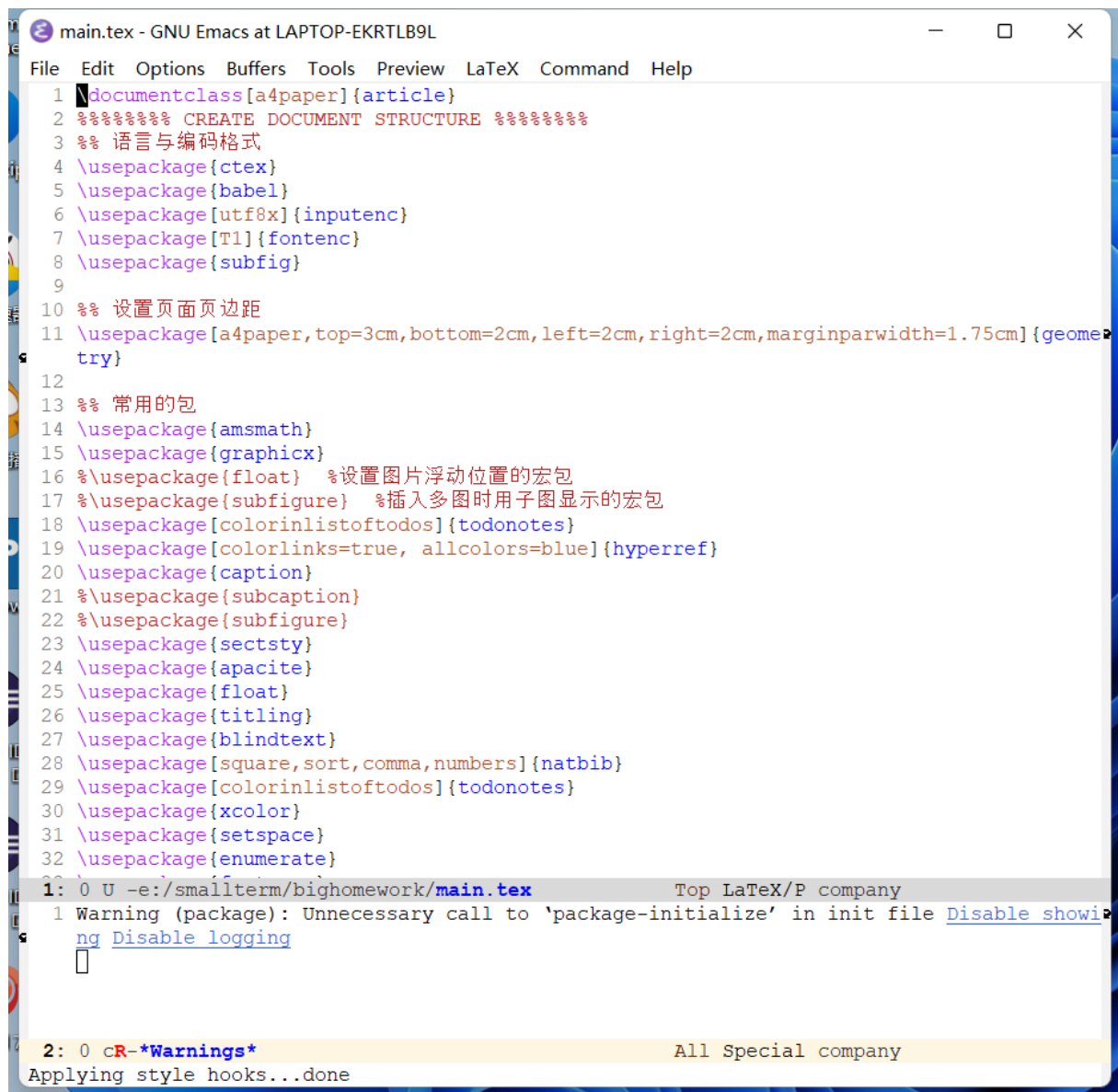
- \* `company-tabnine`, 代码补全工具, 是一个带有 AI 性质的插件, 自动补全代码对编程语言初学者非常友好。

### 2.1.3 LaTeX

本次大作业报告使用 LaTeX 进行排版, LaTeX 的前身是 TeX, TeX 是由 Donald Knuth 创造的基于底层编程语言的电子排版系统。使用 TeX 可以对版面做精细的操作, 生成精美的文档。TeX 提供版面编辑的底层命令, 但是使用时操作复杂, 工作效率不高。TeX 允许用这些底层命令定义出可以执行复杂操作的扩展命令。LaTeX 是由 Leslie Lamport 开发的 TeX 扩展命令集合。LaTeX 通过整合常用的版面设置操作, 降低排版的工作量和难度。LaTeX 强大的排版功能, 特别是对科技文档的支持, 已经使它成为撰写科研论文的事实上的标准。并且在使用 LaTeX 进行排版时, 工作者的专业性得以体现。

在安装配置上, 我选取了 TeX Live 2022 作为启动和运行 TeX 文档生成系统的工具, 并且 TeX Live 自身带有 TeXworks 编辑器, 选取 XeLaTeX 进行编译后 (由于报告采用中文编写, 用 XeLaTeX 避免出现乱码问题), 可以点击按钮直接生成 pdf 文件。但是为了加深对 Emacs 熟练度, 还是采用 Emacs 编写 LaTeX 文档。在 `use-package` 的插件基础上, 利用 “`M-x customize-group tex-command`” 命令打开 LaTeX 配置界面, 修改 TeX Command List、Value Menu->name、LaTeX-command-style 字段就可以使用 C-c C-c 编译, C-c C-p C-p 预览。另外, 在 Emacs 上编写 LaTeX 文档也具有语法高亮的功能, 使用体验上强于 TeXworks。

下图展示了用 LaTeX 编写文档的界面。



```
main.tex - GNU Emacs at LAPTOP-EKRTL9L
File Edit Options Buffers Tools Preview LaTeX Command Help
1 \documentclass[a4paper]{article}
2 %%%%%%%%% CREATE DOCUMENT STRUCTURE %%%%%%%%%
3 %% 语言与编码格式
4 \usepackage{ctex}
5 \usepackage{babel}
6 \usepackage[utf8x]{inputenc}
7 \usepackage[T1]{fontenc}
8 \usepackage{subfig}
9
10 %% 设置页面页边距
11 \usepackage[a4paper,top=3cm,bottom=2cm,left=2cm,right=2cm,marginparwidth=1.75cm]{geometry}
12
13 %% 常用的包
14 \usepackage{amsmath}
15 \usepackage{graphicx}
16 %\usepackage{float} %设置图片浮动位置的宏包
17 %\usepackage{subfigure} %插入多图时用子图显示的宏包
18 \usepackage[colorinlistoftodos]{todonotes}
19 \usepackage[colorlinks=true, allcolors=blue]{hyperref}
20 \usepackage{caption}
21 %\usepackage{subcaption}
22 %\usepackage{subfigure}
23 \usepackage{sectsty}
24 \usepackage{apacite}
25 \usepackage{float}
26 \usepackage{titling}
27 \usepackage{blindtext}
28 \usepackage[square,sort,comma,numbers]{natbib}
29 \usepackage[colorinlistoftodos]{todonotes}
30 \usepackage{xcolor}
31 \usepackage{setspace}
32 \usepackage{enumerate}
33
1: 0 U -e:/smallterm/bighomework/main.tex Top LaTeX/P company
1 Warning (package): Unnecessary call to 'package-initialize' in init file Disable showing Disable logging
□
2: 0 cR-*Warnings* All Special company
Applying style hooks...done
```

图 7: Emacs LaTeX 界面

## 2.2 环境

在本节中，会详细描述在手势识别领域创新点的产生过程和实验环境，它是理解跨域实现原理的基础。

### 2.2.1 pytorch

pytorch 是一个基于 torch 的 python 开源机器学习库，用于自然语言处理等应用程序。作为一个很著名的支持 GPU 加速和自动求导的深度学习框架，在最近几年收到学术界的热捧，主要是因为其动态图机制符合思维逻辑，方便调试，适合于需要将想法迅速实现的研究者。pytorch 运行速度虽然受动态图机制的影响没有 tensorflow 快，但是在 GPU 上的运行速度也是相当不错的。pytorch 的安装也很简便，在 Anaconda 的基础上，创建符合 CPU 和显卡 CUDA 的版本的虚拟环境，利用官网生成的 cmd 命令安装相关依赖包即可。

在本次大作业中，对 pytorch 进行了一个简单的了解，并试验了一些 torch 相关的函数，运行了简单的训练代码。在手势识别的研究中，实际使用的是 tensorflow 框架。

### 2.2.2 tensorflow(keras)

tensorflow (keras) 是大作业的重点训练使用的框架, tensorflow 是一个采用数据流图 (data flow graphs), 用于数值计算的开源软件库。tensorflow 最初由 Google 大脑小组 (隶属于 Google 机器智能研究机构) 的研究员和工程师们开发出来, 用于机器学习和深度神经网络方面的研究, 但这个系统的通用性使其也可广泛用于其他计算领域。它是谷歌基于 DistBelief 进行研发的第二代人工智能学习系统。2015 年 11 月 9 日, Google 发布人工智能系统 tensorflow 并宣布开源。tensorflow 的命名来源于本身的原理, tensor(张量) 意味着 N 维数组, flow(流) 意味着基于数据流图的计算。tensorflow 运行过程就是张量从图的一端流动到另一端的计算过程。张量从图中流过的直观图像是其取名为“tensorflow”的原因。tensorflow 的关键点是: “Data Flow Graphs”, 表示 tensorflow 是一种基于图的计算框架, 其中节点 (Nodes) 在图中表示数学操作, 线 (Edges) 则表示在节点间相互联系的多维数据数组, 即张量 (tensor), 这种基于流的架构让 tensorflow 具有非常高的灵活性, 该灵活性也让 tensorflow 框架可以在多个平台上进行计算, 例如: 台式计算机、服务器、移动设备等。

keras 是基于 tensorflow 和 theano (可以理解为 tensorflow 与 keras 的中间框架) 的深度学习库, 是由纯 python 编写而成的高层神经网络 API, 也仅支持 python 开发。它是为了支持快速实践而对 tensorflow 或者 theano 的再次封装, 让使用者可以不用关注过多的底层细节, 能够把想法快速转换为结果。keras 默认的后端为 tensorflow, 在这里可以看出 keras 和 tensorflow 是一个包含的关系。如果用到 theano 可以自行更改。tensorflow 和 theano 都可以使用 GPU 进行硬件加速, 往往可以比 CPU 运算快很多倍。因此如果显卡支持 cuda 的话, 会利用 cuda 加速模型训练。

## 2.3 数据集

大作业研究的主要内容是在 2019 年清华大学科研团队推出的 Widar3.0 数据集。在以往的模型训练上, 普遍都缺乏对跨域的自适应工作, 因此当出现新的数据域时, 无论是数据收集还是模型重新训练, 都需要额外的训练工作, 带来较大的开销, 从而限制了其实用性。Widar3.0 的核心思想是在较低信号水平上推导和估计手势的速度分布, 这代表了手势的独特动力学特征并且与域无关。Widar3.0 利用卷积神经网络与循环神经网络分别挖掘输入特征 BVP 在空间维度与时间维度的特性, 对 6 种常见的人机交互手势 (推拉、横扫、拍手、滑动、画圆、画之字) 进行判别。另外, 还开发了一种只需要一次性训练但可以适应不同数据域的万能模型。在不同域间的识别准确率也能达到 92.4%。因此 Widar3.0 的亮点不在于训练模型, 而在于对手势相关和手势无关数据的提取。

该数据集包总大小达 325GB, 包括了 2018 2019 年的若干次实验数据, 其中 BVP 生成模块含有 CSI 原始数据、DFS 配置文件及转化算法和 BVP 数据及转化算法; 手势识别模块实现了用于手势识别的深度学习神经网络 (DNN)。Widar3.0 处理的是生成的 BVP 数据, 对每个 BVP 以及整个系列进行归一化操作, 以去除实例和人员的不相关变化。然后将归一化的 BVP 系列输入到时空 DNN 中, 该 DNN 具有两个主要功能。首先 DNN 利用卷积层从每个 BVP 中 提取高级空间特征。然后采用循环层对 BVP 之间的相互特征进行 时间建模。最后, DNN 的输出用于指示用户执行的手势类型。Widar3.0 原则上实现了跨域手势识别的通用模型, 只需要对该深度神经网络进行一次训练, 就可以在域间保持较高的准确率。

另外, 有必要对数据集中的三个重要数据类型作出定义。

- CSI (Channel State Information): 信道状态信息, CSI 是信号在传输过程中利用 OFDM 技术得到的能够表示物理层细粒度的信号特征。在无线通信领域, CSI 指的是通信链路的信道属性。它描述了信号在每条传输路径上的衰弱因子, 即信道增益矩阵  $H$  中每个元素的值, 如信号散射 (Scattering), 环境衰弱 (fading, multipath fading or shadowing fading), 距离衰减 (power decay of distance) 等信息。CSI 可以使通信系统适应当前的信道条件, 在多天系统为高可靠性高速率的通信提供了保障。CSI 信

号的幅度信息可以量化多径效应后的信号功率衰减。因此，通过分析人体手势动作对信号造成的幅度和相位影响可以有效识别人类动作。

- **DFS (Doppler Frequency Shift):** 多普勒频移, 指的是多普勒效应造成的发射源和接收源的频率之差。人的所有身体部位都有独特的速度分布, 可以用作活动指标, 在人反射信号的所有参数中, DFS 体现了速度分布的大部分信息。当一个人做一个手势时, 他的身体部位 (例如, 两只手、两只手臂和躯干) 以不同的速度移动。因此, 这些身体部位反射的信号会经历各种 DFS, 这些 DFS 在接收器处叠加并形成相应的 DFS 轮廓。通过计算同一 Wi-Fi NIC 上两个天线的 CSI 的共轭乘法, 并滤除带外噪声和准静态偏移, 可以去除随机偏移, 并且只保留具有非零 DFS 的显着多径分量。进一步应用短期傅立叶变换会产生时间和多普勒频域上的功率分布。具体来说, DFS 配置文件  $D$  是一个维度为  $F \times M$  的矩阵, 其中  $F$  是频域中的采样点数,  $M$  是收发器链路数。基于来自多个链接的 DFS 配置文件, 最终可以形成 BVP 数据。
- **BVP (Body-coordinate Velocity Profile):** 身体坐标速度分布文件, 它描述了不同速度上的功率分布, 其中身体部位参与了手势运动。我们的观察是, 无论在哪个域中做出手势, 每种类型的手势在身体坐标系中都有其独特的速度分布。因此, 这些身体部位反射的信号会经历各种 DFS, 这些 DFS 在接收器处叠加并形成相应的 DFS 轮廓。虽然 DFS 配置文件包含手势信息, 但它也高度特定于域。相比之下, 人的身体坐标系中物体速度的功率分布仅与手势的特征有关。因此, 为了消除域的影响, BVP 是从 DFS 配置文件中派生出来的。

### 2.3.1 采集设备

Widar3.0 由一个发射器和三个以上的接收器组成, 远端接收器的角度大于  $180^\circ$ 。所有收发器都是现成的迷你桌面 (物体尺寸  $170\text{mm} \times 170\text{mm}$ ), 配备 Intel 5300 无线网卡。Linux-CSI-Tool 安装在设备上以记录 CSI 测量结果。设备设置为在监控模式下工作, 在  $5.825\text{GHz}$  的 165 频道上, 可以将干扰无线电减少到最小, 因为干扰确实会对收集的 CSI 测量值造成严重影响。发射器激活一根天线并以每秒 1,000 个数据包的速率广播 Wi-Fi 数据包。接收器激活所有三个放置在一条线上的天线。并在 MATLAB 和 Keras 中实现了 Widar3.0。

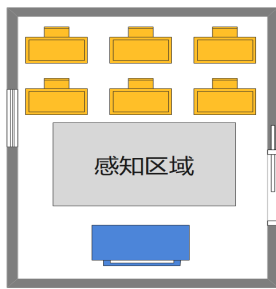


图 8: Room1

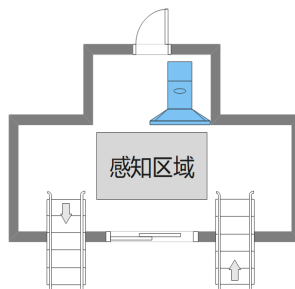


图 9: Room2

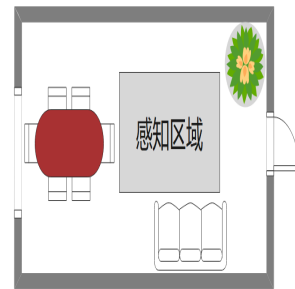


图 10: Room3

在空间环境上, 选取了三个门内环境进行广泛的手势识别实验, 一个配有桌椅的空教室、一个宽敞的大厅和一个配有沙发和桌子等家具的办公室。配有桌椅的空教室处于一个常规的长方体空间, 存在的智能设备功能广泛应用在教学中, 其影响信号的因素在于多个桌椅带来的信号反射不规律性; 宽敞的大厅呈“凸”字形, 其结构虽然不如长方体空间带来的信号间接, 但是其拥有的对称性也是有迹可循; 配有沙发和桌子等家具的办公室相对于配有桌椅的空教室, 其空间大小有所减小, 家具数量少、体积大, 也是智能设备最常存在的情况。这三个门内环境对于手势识别的评估很具有代表性。

### 2.3.2 采集过程

手势数据的采集流程和模型形成的步骤可以概括为下图所示。

模型形成流程

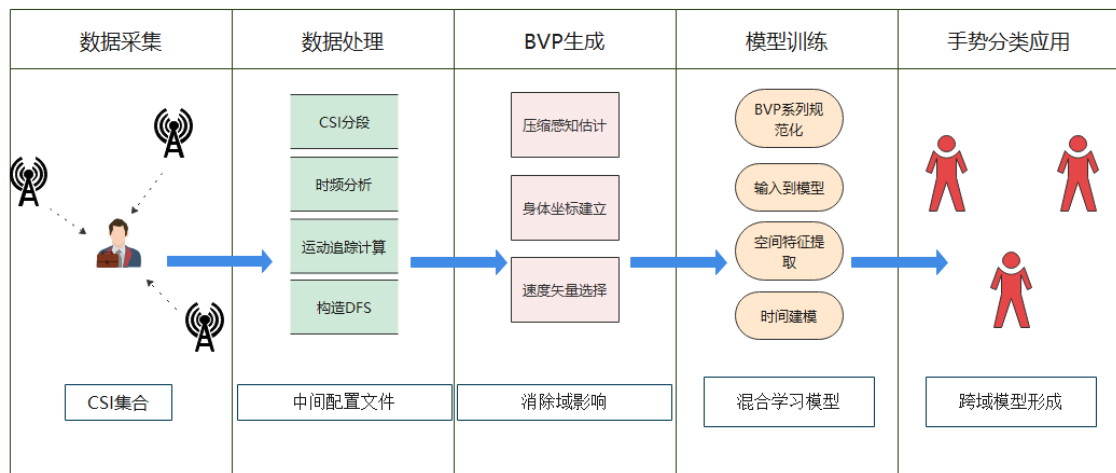


图 11: 模型形成过程

在已存在 WIFI 设备的前提下，首先要进行手势数据的提取，在监控区域周围部署多条无线链路。在接收器处采集用户在监控区域内失真的无线信号，并记录和预处理其 CSI 测量值。CSI 描述了数据包到达时间  $t$  时室内环境中的多径效应，预处理的主要目的是消除幅度噪声和相位偏移。

手势识别系统接收到净化后的 CSI 系列后，将 CSI 系列划分小段，用于分段生成 BVP。在时频分析步骤中，要充分考虑路径数和相应的复衰减和传播延迟。通过用相应的 DFS 表示多径信号的相位，CSI 可以转换为频谱形式，进一步产生时间和多普勒频域上的分布。之后，通过运动跟踪方法计算人的方向和位置信息，这里提出了身体坐标的概念，因为每种类型的手势在身体坐标系中都有其独特的速度分布，这也是导出速度分量的信号功率分布的前提，最后得到了人的方向、位置信息以及 DFS 配置文件，这些数据相当于 BVP 生成的中间文件。

最关键的一步也是 BVP 数据文件的生成，当用户与设备进行交互时，会靠近它并进行交互手势来识别和响应。人的先前运动为估计位置和方向提供了机会，这些位置和方向是人在轨迹末端的位置和移动方向。由于基于 Wi-Fi 的被动跟踪已经被广泛研究，Widar3.0 利用了现有复杂的被动跟踪系统，例如 LiFS、IndoTrack 和 Widar2.0，来获取目标的位置和方向。然而，Widar3.0 与这些被动跟踪方法的不同之处在于估计 BVP 而非主躯干速度，因此进一步扩展了基于 Wi-Fi 的传感范围。一个 BVP 被量化为维度为  $N \times N$  的离散矩阵，其中  $N$  是沿身体坐标的每个轴分解的速度分量的可能值的数量。建立局部身体坐标，其原点人的位置，正  $x$  轴与人的方向对齐。我们将在第 4.4 节讨论估计一个人的位置和方向的方法。目前，假设该人的全球位置和方向是可用的。然后将已知的无线收发器的全球位置转换为局部身体坐标。人体周围的速度分量将其信号功率贡献给对应的频率分量，由于在计算 DFS 配置文件之前过滤掉了具有零 DFS 的

静态分量，因此只保留人的反射信号。当人靠近 WIFI 链路时，只有一次反射的信号具有明显的幅度，因此根据链路信息和速度分量可以对 DFS 与 BVP 之间的关系建立模型。由于单个手势的 DFS 文件变量达到数百个之多，而物理链路只能提供有限数量的约束，所以要利用反射多径信号的数量有限性以及 DFS 剖面带来的投影重叠，可以从有限的链路的 DFS 文件中正确恢复 BVP。

在模型训练上，Widar3.0 设计了一个 DNN 学习模型来挖掘 BVP 系列的时间和空间特征，该深度神经网络包含了用于空间特征提取的卷积神经网络（CNN）和用于时间建模的递归神经网络（DNN），这样的混合深度学习模型以前一步的 BVP 数据作为输入，实现准确的跨域手势识别。首先将 BVP 系列进行规范化处理，设置手势的标准时间长度，缩放 BVP 中所有速度分量的倍数，假设每个身体部位移动的总距离保持固定，将 BVP 系列进行时间缩放，之后将序列重新采样到原始 BVP 序列的采样率。这样经过规范化处理的输出数据仅与手势相关，将其输入到深度学习模型。

卷积神经网络提供了一种提取空间特征和压缩数据的实用技术，它可以用于处理高度稀疏但保留空间局部性的单个 BVP，BVP 类似于图像序列，单个 BVP 描述了在足够短的时间间隔内物理速度上的功率分布，连续的 BVP 系列说明了分布如何对应于某种动作的变化。输入的 BVP 系列作为维度为  $N \times N \times T$  的张量，其中  $T$  是 BVP 快照的数量，对于第  $t$  个采样的 BVP，矩阵被馈送到 CNN。在 CNN 中首先将 16 个 2-D 滤波器应用于  $V..t$  以获得速度域中的局部模式，从而形成输出  $v(1)..t$ ，然后将最大池化应用于  $v(1)..t$  对特征进行采样，输出表示为  $v(2)..t$ 。两个以 ReLU 作为激活函数的 64 单元密集层用于进一步提取更高级别的特征。在两个密集层之间添加了一个额外的 dropout 层以减少过度拟合。最终输出  $v..t$  表征第  $t$  个采样 BVP，并且输出序列用作后续循环层的输入以进行时间建模。

除了每个 BVP 中的局部空间特征外，BVP 系列还包含手势的时间动态。递归神经网络（RNN）的吸引力在于它们可以对复杂的序列时间动态进行建模。有不同类型的 RNN 单元，例如 SimpleRNN、长短期记忆（LSTM）和门控循环单元（GRU）。与原始 RNN 相比，LSTM 和 GRU 更适用于学习长期依赖关系，GRU 在序列建模上实现了与 LSTM 相当的性能，但涉及的参数较少，并且更容易用更少的数据进行训练。Widar3.0 采用单层 GRU 来建模时间关系，来自 CNN 的输入  $v..t$   $t=1 \dots T$  输出被馈送到 GRU，并生成一个 128 维的向量。此外，为正则化添加了一个 dropout 层，并使用一个带有交叉熵损失的 softmax 分类器进行类别预测。经过了上述过程的模型训练，Widar3.0 的整体准确率可达 92.7%，在 Room1 中收集的 90% 和 10% 的数据分别用于训练和测试。Widar3.0 对所有手势均实现了超过 85% 的高精度。我们还额外添加了“未知”类的手势进行实验。除上述 6 种手势外，志愿者需要进行任意手势。整体准确率下降到 90.1%，Widar3.0 可以区分未知类，准确率达到 87.1%。来自未知类别的手势在一定程度上类似于预定义的手势，会使准确率没有下降得太多；收集到的未知手势数量仍然有限，未来如果引入额外得过滤机制或修改，结果可以进一步改善。

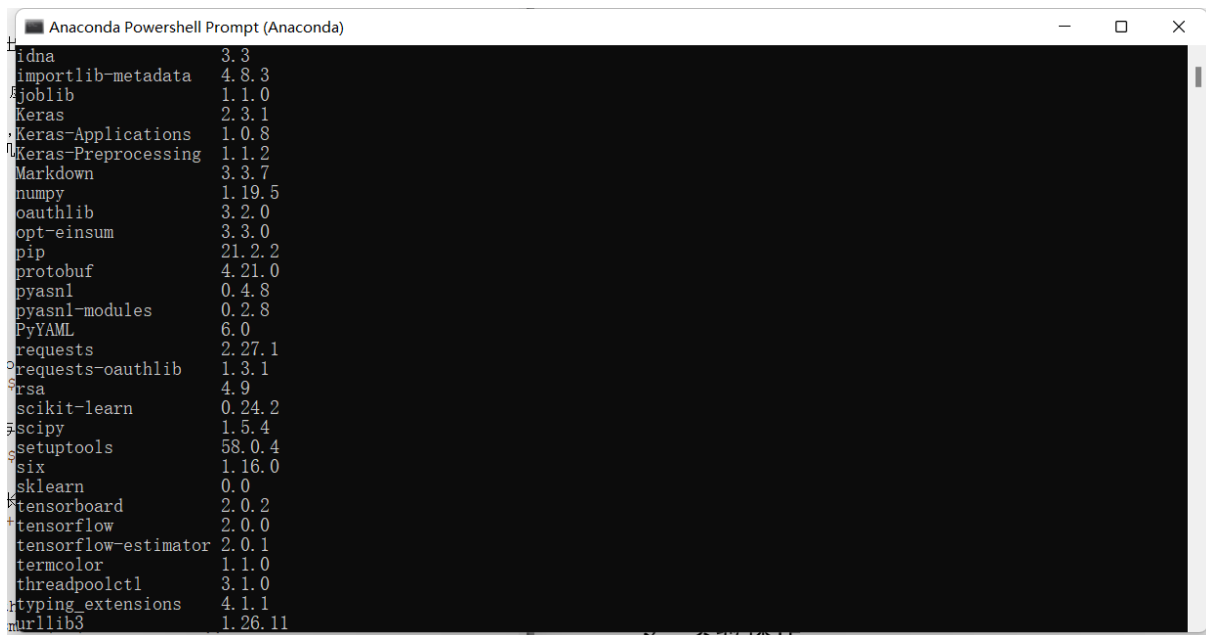
### 3 实验操作

在本章节中，重点对模型训练部分的实践做出概述。

#### 3.1 环境版本

深度学习环境以 Anaconda 作为依赖包资源和虚拟环境创建工具，Anaconda 的版本并无硬性要求，所以安装了最新版本的 Anaconda 是可以投入使用的。

使用的深度学习框架为 tensorflow 和 keras，其中 keras 相当于 tensorflow 的一个高阶应用程序接口，所以安装时尤其注重了版本之间的对应关系，后面的代码实现会体现 tensorflow 的简便性。在本机环境中 tensorflow 版本为 2.0.0，对应了 keras 的版本为 2.3.1，同时也对 python 的版本做出了范围要求，在创建虚拟环境时给出参数选取了 python 为 3.6，实际安装版本为 3.6.13，以上的搭建保证了代码正常运行的基础条件。其他附带的依赖包如下图所示。



```
Anaconda Powershell Prompt (Anaconda)
conda 3.3
importlib-metadata 4.8.3
joblib 1.1.0
Keras 2.3.1
Keras-Applications 1.0.8
Keras-Preprocessing 1.1.2
Markdown 3.3.7
numpy 1.19.5
oauthlib 3.2.0
opt-einsum 3.3.0
pip 21.2.2
protobuf 4.21.0
pyasn1 0.4.8
pyasn1-modules 0.2.8
PyYAML 6.0
requests 2.27.1
requests-oauthlib 1.3.1
rsa 4.9
scikit-learn 0.24.2
scipy 1.5.4
setuptools 58.0.4
six 1.16.0
sklearn 0.0
tensorboard 2.0.2
tensorflow 2.0.0
tensorflow-estimator 2.0.1
termcolor 1.1.0
threadpoolctl 3.1.0
typing_extensions 4.1.1
urllib3 1.26.11
```

图 12: 该虚拟环境下需要安装的依赖包

#### 3.2 硬件支持

在模型部署时，为了把握训练速度和精度，需要重点关注硬件设备。由于实验条件的限制，只能利用笔记本电脑的配置进行深度学习。

- CPU: Intel(R) Core(TM) i7-9750H CPU @, 在 2019 年第二季度推出, 拥有 6 核心 12 线程, 主频 2.60GHz, 自带英特尔® 超核芯显卡 630, 可以作为辅助 GPU 为训练加速处理。
- GPU: NVIDIA GeForce RTX 2060, CUDA 核心数量为 2176, 基础频率 1.47GHz, 加速频率 1.65GHz, 配有 12GB GDDR6 显存。

以上配置可以保证本次手势识别深度学习的训练速度和准确度。



### 3.3 代码实现与调整

在提供的深度神经网络（DNN）模型代码中，运用了大量受版本限制的库和函数，在实践过程中可以判断中，本机使用版本与源代码实现版本不同且不兼容，进行后的调整后导入包如下图所示。

```
from __future__ import print_function

import os, sys
import numpy as np
import scipy.io as scio
import tensorflow as tf
import keras
from keras.layers import Input, GRU, Dense, Flatten, Dropout, Conv2D, Conv3D, MaxPooling2D,
    MaxPooling3D, TimeDistributed
from keras.models import Model, load_model
import keras.backend as K
from sklearn.metrics import confusion_matrix
from keras.backend.tensorflow_backend import set_session
from sklearn.model_selection import train_test_split
```

图 13: import 导入情况

其中 os, sys 是系统模块，涉及到文件读取的操作需要用到；numpy 主要是对数组进行转维和一些计算；scipy 依赖于 numpy，目的是进行便捷且快速的 N 维数组操作。

之后引入了 tensorflow 和 keras 模块，为了之后的张量计算和时间建模，同时还要引入 Input、GRU（门控循环单元）、Dense 等包。keras.models 模块与模型建立有关，backend 是 keras 默认支持的后端，在运行中会出现“Using tensorflow backend”字样。sklearn 库用来进行自动参数调整和神经模型调参，sklearn.model\_selection 中可实现的功能有构造数据的交叉验证迭代器以及拆分训练集与测试集。

在训练前还需要设置一些参数如下图所示。

```
# Parameters
use_existing_model = False
fraction_for_test = 0.1
data_dir = 'BVP/'
ALL_MOTION = [1, 2, 3, 4, 5, 6]
N_MOTION = len(ALL_MOTION)
T_MAX = 0
n_epochs = 30
f_dropout_ratio = 0.5
n_gru_hidden_units = 128
n_batch_size = 4 #32
f_learning_rate = 0.001
```

图 14: 预先设置的 Parameters

上述参数指定的一些内容为为：不利用已有的训练模型，设置了训练集与测试集的比例，规定了训练精度要求，指出了训练次数和每次训练样本数以及一些其他的单元数据。

对于 Widar3.0 实验提供的参数，为了与本机配置环境兼容，做出了调整。首先，在之前的背景知识了解下明确该模型处理的是 BVP 数据，所以将 data\_dir 由先前的 Data 调整为 BVP；由于在训练时发生了显存不足的错误，所以调整了 n\_batch\_size 从 32 下降到 4，这样可以防止程序途中崩溃。



训练步骤开始，首先提供了硬件选择的代码。

```
if len(sys.argv) < 2:
    print('Please specify GPU ...')
    exit(0)
if (sys.argv[1] == '1' or sys.argv[1] == '0'):
    os.environ["CUDA_VISIBLE_DEVICES"] = sys.argv[1]
    config = tf.compat.v1.ConfigProto()
    # config.gpu_options.per_process_gpu_memory_fraction = 0.4
    config.gpu_options.allow_growth = True
    tf.compat.v1.keras.backend.set_session(tf.compat.v1.Session(config=config))
    tf.random.set_seed(1)
else:
    print('Wrong GPU number, 0 or 1 supported!')
    exit(0)
```

图 15: 硬件选择

从代码中的 `sys.argv` 可以看出，它给予用户硬件选择的机会，在运行 `python` 程序时需要给出选择的 GPU，并作为 `os.environ["CUDA_VISIBLE_DEVICES"]` 的参数。`config = tf.compat.v1.ConfigProto()` 配置了运算方式，这里进行了版本代码的调整。由于 `tensorflow` 默认会跑满整个内存，且之前的实践过程出现了多次的内存错误，调整 GPU 动态使用并不能解决问题，所以硬性地指定了 GPU 的使用比率，通过观察硬件使用情况得出结论：在使用 40% 的 GPU 时效果较好。

基于 `keras` 的特征提取部分代码及损失函数如下图所示。

```
def assemble_model(input_shape, n_class):
    model_input = Input(shape=input_shape, dtype='float32', name='name_model_input') # (0,T_MAX,20,20,1)

    # Feature extraction part
    x = TimeDistributed(Conv2D(16, kernel_size=(5,5), activation='relu', data_format='channels_last',
        input_shape=input_shape))(model_input) # (0,T_MAX,20,20,1)=>(0,T_MAX,16,16,16)
    x = TimeDistributed(MaxPooling2D(pool_size=(2,2)))(x) # (0,T_MAX,16,16,16)=>(0,T_MAX,8,8,16)
    x = TimeDistributed(Flatten())(x) # (0,T_MAX,8,8,16)=>(0,T_MAX,8*8*16)
    x = TimeDistributed(Dense(64, activation='relu'))(x) # (0,T_MAX,8*8*16)=>(0,T_MAX,64)
    x = TimeDistributed(Dropout(f_dropout_ratio))(x)
    x = TimeDistributed(Dense(64, activation='relu'))(x) # (0,T_MAX,64)=>(0,T_MAX,64)
    x = GRU(n_gru_hidden_units, return_sequences=False)(x) # (0,T_MAX,64)=>(0,128)
    x = Dropout(f_dropout_ratio)(x)
    model_output = Dense(n_class, activation='softmax', name='name_model_output')(x) # (0,128)=>(0,n_class)

    # Model compiling
    model = Model(inputs=model_input, outputs=model_output)
    model.compile(optimizer=keras.optimizers.RMSprop(lr=f_learning_rate),
        loss='categorical_crossentropy',
        metrics=['accuracy'])

    return model
```

图 16: 特征提取参数和损失函数

该特征提取部分也是 `keras` 框架相对于 `pytorch` 最大的一个优点，`keras` 将张量的计算等操作集成到了几个 API 中，调用现成的函数即可完成张量的计算及 `TimeDistributed` 层的包装，相同的功能如果用在 `pytorch` 中，则需要自行将输入数据转换成张量的形式。

在损失函数方面，采用了 `keras` 的 `RMSprop` 优化器解决自适应调整学习率急速下降的问题，`Loss` 损失函数选取了 `categorical_crossentropy`，为了应对多类模式的标签。

另外，代码中也内置了一些如 `load_data`、`normalize_data`、`zero_padding` 等对数据集的导入、标准化和分量选择等操作。训练之后生成 `h5` 模型文件并保存在当前目录下。

```
Anaconda Powershell Prompt (Anaconda)
30703/30703 [=====] - 75s 2ms/step - loss: 0.5272 - accuracy: 0.8084 - val_loss: 0.6335 - val_a
ccuracy: 0.7732
Epoch 27/30
30703/30703 [=====] - 72s 2ms/step - loss: 0.5150 - accuracy: 0.8119 - val_loss: 0.5800 - val_a
ccuracy: 0.7890
Epoch 28/30
30703/30703 [=====] - 72s 2ms/step - loss: 0.5114 - accuracy: 0.8107 - val_loss: 0.5778 - val_a
ccuracy: 0.7761
Epoch 29/30
30703/30703 [=====] - 70s 2ms/step - loss: 0.5021 - accuracy: 0.8172 - val_loss: 0.6188 - val_a
ccuracy: 0.7752
Epoch 30/30
30703/30703 [=====] - 72s 2ms/step - loss: 0.4955 - accuracy: 0.8181 - val_loss: 0.6137 - val_a
ccuracy: 0.7752
Saving trained model...
Testing...
[[517 26 35 48 20 8]
 [ 16 497 50 16 45 23]
 [ 16 17 576 6 15 24]
 [ 50 16 24 459 65 9]
 [ 7 32 19 51 468 44]
 [ 4 12 44 18 35 479]]
[[0.79 0.04 0.05 0.07 0.03 0.01]
 [0.02 0.77 0.08 0.02 0.07 0.04]
 [0.02 0.03 0.88 0.01 0.02 0.04]
 [0.08 0.03 0.04 0.74 0.1 0.01]
 [0.01 0.05 0.03 0.08 0.75 0.07]
 [0.01 0.02 0.07 0.03 0.06 0.81]]
0.7902927987338433
(tfcpu) PS E:\smallterm\Widar3.0>
```

图 17: 训练结果

上图展现了其中一次的深度学习输出结果，事实上 tensorflow 的 `random.set_seed(1)` 的设置，导致该深度神经网络使用增益或权重的值每一次都会有所变化，因此每次训练结果的数值会有差别。在这一次的训练结果中，训练学习率达到了 81.81%，在随后的测试集中，给出了测试矩阵及测试结果，虽然测试准确率仅为 79.03%，较之前的学习准确率有所下降，但是在当时的研究现状上，跨域识别中准确率未曾有如此大的突破，可以证明 BVP 模块的思想是跨域识别中重要的贡献。

## 4 总结与心得

### 4.1 收获

在为期四周的小学期中，我阅读参考了一些关于无线感知和手势识别甚至人的其他点位识别的研究方式的文献，并与 Widar3.0 所论述的观点进行了简要的对比分析。并且在已有的环境基础之上，第一次亲自搭建了深度学习的环境，并且运用了功能强大的 tensorflow keras 框架，对深度学习的内容有了初步的认识。

在工具使用上，我学习了 Emacs 这一强大的专业编辑器，Emacs 贯穿了小学期工作的始终，在使用体验上和上学期 FreeBSD 系统上的 vim 编辑器有类似的感受，两者都摆脱了对鼠标的依赖，虽然 Emacs 的快捷键组合复杂难记，但是在实际的使用过程中常用的一些操作渐渐了然于心。

Emacs 不仅用于代码的编写和调整，还用于本报告的 LaTeX 编写，在这次小学期我第一次用 LaTeX 进行文档排版，LaTeX 也是需要一定的代码数量，所以初次使用时会感到难以上手，在掌握了段落、编号、图片插入等基本操作后，文档的编写变得十分容易，并且可以直接编译成 pdf 文件，生成的文档整体简洁美观，我感受到了 LaTeX 排版文档的优势。

### 4.2 遇到的问题

小学期的一切领域和工具都是全新的，所以在完成工作中也是遇到了不少问题。

首先是环境安装的问题，Emacs 本身是一个很好的编辑器，但是也需要一些插件的支持，所以配置文件的编写会遇到问题，Emacs 启动时指定了配置文件的路线，所以配置文件的路径和命名是固定的。我多次创建了错误路径和错误命名的文件，导致即使 Lisp 代码安装的插件格式正确，在重启 Emacs 时也不会进行自动配置。在修改为正确的配置文件和匹配的镜像后，插件能够正确安装。

虽然查阅过资料得知 keras、tensorflow、cuda、cudnn、python 之间存在版本对应关系，但是组合的选取也比较困难，在开始的两周里尝试了若干种版本的组合，因为 python 的库函数和版本之间也有较强的依赖关系，经常遇到的情况是在花费大量时间安装了框架和依赖包后，发现使用的函数在现有的包中已不再适用，所以被迫重新选取所有包，不断地尝试、失败后才确定了一组合适的深度学习环境。

深度学习环境不兼容，Widar3.0 所使用的训练环境和本机有所差距，所以要特别注重对硬件资源的把控。在运行程序时会跑满内存，导致很难进行其他的任何操作，然而即使这样，还是经常会出现内存、显存不足的情况，所以在每次运行程序时，需要手动设置虚拟内存的大小和分配方式。

还有 Emacs 和 LaTeX 工具的使用问题，相对于代码来说，工具的使用问题比较好处理，并且最后都能顺利解决。

### 4.3 对发展前景的展望

小学期中的跨域手势识别研究代表了 2019 年的成果，到如今已经过去了三年的时间，通过对该领域的洞察也发现有更多巧妙的手段能实现跨域的手势识别。随着无线感知技术的快速发展，越来越多的智能设备出现在市面上，手势识别这一人机交互方式被普遍认知是最简单、最轻松的。我们希望在不久的将来，手势识别能够像现在的人脸识别一样准确，并且相关的智能设备产品真正投入到每个百姓的家中使用。