北京师范大学珠海分校

本科生毕业论文

论文题目： 人机交互-手势识别的实现与应用

学　　院	信息技术学院
专　　业	软件工程
学　　号	1601030056
学 生 姓 名	李绍铭
指导教师姓名	赵志文
指导教师单位	信息技术学院

2020　　年　3　月　1　日

北京师范大学珠海分校学位论文写作声明和使用授权说明

学位论文写作声明

本人郑重声明： 所呈交的学位论文，是本人在导师的指导下，独立进行研究工作所取得的成果。除文中已经注明引用的内容外，本论文不含任何其他个人或集体已经发表或撰写过的作品或成果。对本文的研究做出重要贡献的个人和集体，均已在文中以明确方式标明。本声明的法律结果由本人承担。

论文作者签名: 日期: 年 月 日

学位论文使用授权说明

本人完全了解北京师范大学珠海分校关于收集、保存、使用学位论文的规定，即：按照学校要求提交学位论文的印刷本和电子版本；学校有权保存学位论文的印刷本和电子版，并提供目录检索与阅览服务；学校可以采用影印、缩印、数字化或其它复制手段保存论文；在不以赢利为目的的前提下，学校可以将学位论文编入有关数据库，提供网上服务。（保密论文在解密后遵守此规定）

论文作者签名: 导师签名:

日期: 年 月 日

人机交互---手势识别的实现与应用

摘要

随着计算机的高速发展，如今人机交互在智能生活中成为了必不可少的一部分。尤其是 VR 和 AR 可穿戴设备的越来越普及以及越来越低廉的价格，人机交互轻而易举地走进了平常人们的生活中。语言是人类之间沟通的桥梁，在使用不同语言的人们交流时，人们会下意识地使用肢体语言去补充自己的意思，因为肢体语言是人类发展历史中最原始的沟通方式。所以肢体语言上的人机交互在人与机器的交流上也有非常大的研究价值，也为了使人与计算机的交互变得更加简便。

近几年来卷积神经网络（CNN）在机器视觉领域表现十分出色，与传统的识别方法比起来对于复杂场景拥有更强的适应性。卷积神经网络是通过大量的已经进行标记的数据进行学习后提取图像的特征信息。在许多领域比如文字识别，图像分类等应用场景都拥有着优秀的表现。

本研究旨在设计一个基础的具有通用性的手势识别程序，可以应用在大部分的实例场景，进行一个基础的人机交互操作，旨在简化人机之间的操作，手势识别可以不需要多么复杂的操作，只需要把最简单的操作简化，如换台和调节音量，可以减少找遥控器的步骤，并且这种无接触的操作也可以减少人们的直接接触同一件物品，减少了细菌传染的可能。

本研究通过使用 Python+Keras+opencv 构建 CNN（卷积神经网络）后使用预处理过的数据集训练集进行多种不同参数的模型搭建与训练。并对不同模型训练后得出的数据与数据集测试集进行比对后再进行数据可视化分析操作，获得各模型下的各手势判断的准确度，分析各参数诸如数据规模和训练次数对于最后准确度的影响，最后使用理论较优的参数训练出来的模型对实时的手势进行捕捉以及判断。研究最终将完成一个实时的手势识别程序。旨在实现一个通用的手势识别项目，并将其应用在实例场景中。

关键词：Python；单目视觉；卷积神经网络；人机交互

Human-computer interaction --- the realization and application of gesture recognition

Abstract

With the rapid development of computers, human-computer interaction has become an indispensable part in the ubiquitous intelligent life. Especially with the increasing po pularity of VR and AR wearable devices and lower and lower prices, human-computer interaction has easily entered the daily life of the family. Language is the bridge betwe en human beings. When people using different languages communicate, people sub consciously use body language to supplement their meanings, because body language is the most primitive way of communication in the history of human development. There fore, human-machine interaction in body language also has great research value in hum an-machine communication, in order to make human-computer communication more con venient and simple.

In recent years, Convolutional Neural Networks (CNN) has performed very well in the field of machine vision, and has a stronger adaptability to complex scenes than tr aditional recognition methods. Convolutional neural networks extract feature information of an image after learning from a large amount of labeled data. In many fields such as text recognition, image classification and other application scenarios have excellent p erformance.

The purpose of this research is to design a basic universal gesture recognition prog ram that can be applied to most of the example scenarios to perform a basic human-co mputer interaction operation. It is designed to simplify the operation between human an d machine. For complicated operations, only the simplest operations need to be simplifi ed, such as changing channels and adjusting the volume, which can reduce the steps of finding a remote control, and this non-contact operation can also reduce people's direc t contact with the same item, reducing bacterial infection Possible.

This study uses Python + Keras + opencv to build a CNN (Convolutional Neural Network) and uses a pre-processed dataset training set to build and train models with many different parameters. The data obtained after training of different models are com pared with the test set of the data set, and then the data visualization analysis operatio

n is performed to obtain the accuracy of each gesture judgment under each model, and the analysis of parameters such as data size and training times is accurate for the fina l In the end, a model trained with theoretically optimal parameters is used to capture a nd judge real-time gestures. The research will eventually complete a real-time gesture r ecognition program. The aim is to implement a universal gesture recognition program a nd apply it to the example scenario.

Key Words: Python; monocular vision; convolutional neural network; human -computer interaction

目 录

1 绪论

1.1 课题的研究背景及意义

随着可穿戴设备种类的日益增加和相关技术的发展，人机交互逐渐成为智能生活中的必不可少的部分。人机交互的目标是实现人与机器自然且便捷地沟通交流，因此手势识别研究是智能生活发展的必然趋势。然而由于手本身是个可以进行复杂变形的变形体和视觉上本身的不适定性，同时手势自身就具有的多变性、多样性，所以手势识别的人机交互系统变成了一个多种不同学科交叉共同完成的研究项目。

从古代到现代，人类出现的文明不计其数，各个文明又有着自己独特的语言，就算是有同一语系的国家之间也会有着因自己的文明而影响着出现的不同语言习惯，诸如文字、眼神、肢体、口语等不同的语言表达方式，而肢体语言是不同语言之间的人们交流最简单直接的方式之一，表现出的含义也是最简单直白的，手势则是人类肢体语言中最常用的一部分，不同语言之间交流不通畅的时候往往会伴随着手舞足蹈，而手舞就是使用各种手势来令自己的表达更加清晰。手势交互就是利用了机器视觉领域中的不同技术来识别出不同的手势，再将识别出来的手势赋予不同的命令来对机器进行操作。手势识别的人机交互的使用相较于直接使用鼠标、键盘和遥控器等设备来说要麻烦，但是也会让人们省去找遥控器这一常见的步骤。而近年出现的 HTC vive[i]和 Microsoft HoloLens[ii]的出现让人机交互的方式增加了许多不同的方式，通过 VR 的手势来与 VR 内的场景进行交互让人机交互更加直观和简易，而 VR 设备的不断更新使得人们能够不受时间、地点、背景的约束来进行人机交互。

再者，由于文化背景以及应用环境的不同，手势在不同的文明有着不同的意义，使得手势识别的人机交互研究无法整合到一个完整的框架中。手势识别的项目大多是根据指定的要求实现指定的功能，而不能将其应用到自己的项目中，往往需要为系统设计专用的功能，单独为其造轮子。因此迫切需要对手势识别研究领域中一些常见的难题进行解决，使得项目可以通用于各种项目，不必重复造轮子。

在逐渐普及的智能生活中，手势识别会逐渐在人们的生活中占据重要的地位，如在家中做一个手势自动开灯，手势识别不需要识别非常复杂的手势，只需要碎

片化的出现在人们生活的各个位置，一个简单的手势就可以让原本复杂的操作变得简单。

1.2 国内外研究现状

手势识别的方法主要分为两种：第一种是基于机器视觉技术的方法，第二种是基于传感器设备的方法。

而这些年随着智能生活主题的提起，这就要求不能让用户使用过多的硬件设备，所以研究方向就开始侧重于机器视觉的手势识别方法，要求用户的体验更加倾向于自然和简便地使用人机交互。国内外的许多公司、研究所和高校都将手势的检测、识别、人机交互列为研究的重点项目。目标检测和物体识别主要有两个阶段，第一个阶段是传统方法的分类器方法，第二个是基于人工智能的机器学习方法提取特征和结合分类器的识别方法。比如早期的 HOG 算法，，基于 HOG 算法来提取目标特征，再使用 SVM 分类器来实现目标检测。

在手势检测方面，Lee[iii]等人提出了一种用熵分析的算法，将手势从具有复杂背景的视频流中切分出来，以达到检测的目的，然后再完成手势识别任务，这种算法的识别率在 95%左右，同时只能完成 6 种不同的手势的识别。

基于深度学习的手势识别技术，多列深度神经网络进行图像识别，相较于单个深度神经网络的识别率提高了 30%至 80%。NVIDIA 公司提出一种 3DCNN 的动态手势识别的算法，算法的核心是将两个 CNN 神经网络进行结合，首先是对连续输入的视频流进行逐帧处理，分别是 Low-resolution 和 Hight-resolution 两个网络，获得来自不同空间的特征信息，然后再连接分类器来识别不同的动态手势。

在近年来，深度学习的热度越来越高，而神经网络中的深度卷积神经网络是提取出图像中的局部特征，再将局部的特征通过非线性的变化，最后得到图像的高层特征信息，然后通过这些特征信息来进行目标的检测和识别，CNN 模型是端到端的模型。在学习深度学习的过程中，许多的人都喜欢用 CNN 来进行目标的检测和识别，因为 CNN 在图像识别领域有着非常优秀的表现，在人体姿态识别、人脸识别、视频目标分类这些不同的机器视觉领域，CNN 都有着非常好的效果。

而对于手势识别的人机交互应用来说，微软 Kinect 的面世让手势识别的交互更加深入大众，在使用 Kinect 进行游戏时采集的诸多数据集对于推动手势交

互有着巨大的贡献，这也预示着越来越多的企业、研究所和高校都将手势识别列入人机交互的一个重要项目中。比如 VR 和 AR、无人机等人机交互场景中手势识别都拥有着良好的表现，更加展现了手势识别在人机交互中重要的地位，推动了手势识别技术的发展。

1.3 本论文的研究任务

本篇论文的研究任务是解决 RGB 手势图像在低像素的复杂场景中的手势识别以及指尖定位，CNN 在图像识别领域占有很高的地位。因此本文使用 CNN 卷积神经网络算法框架，利用自行采集和已标注好的手势图像，进行基于神经网络的手势交互技术的研究。主要可以分为三个阶段，数据采集和标注，模型搭建，实时手势识别。使用 Python 以及分析网络上现有的较完善的人工智能框架 Keras、Pytorch、TensorFlow 进行选择适合的框架，主要关注框架的使用难度、运行速度、代码可移植性。神经网络模型的搭建，之后使用预处理过的数据集训练集反复进行模型的训练，修改模型的规模以及训练的次数。

训练完后再使用测试集进行可视化的数据分析，通过分析不同规模以及不同训练次数的模型所得出的各手势的判断准确度，来分析规模以及训练次数与手势判断准确度的关系。最后通过训练出来的模型和使用 OpenCV 对实时的手势进行处理后判断。

2 相关技术分析

2.1 TensorFlow 的简介

TensorFlow 是由 Google Brain 团队开发的第二代的多语言的开源软件库，用于各种感知和语言理解任务的机器学习项目，是一个机器学习的框架，它的底层核心引擎由 C++实现，通过 gRPC 实现网络互访，分布式执行，虽然它的 Python/C++/Java API 共享了大部分执行代码，但是反向传播梯度计算的部分则需要在不同语言实现， 目前只有 Python API 比较丰富地实现了反向传播的部分，所以大部分 TensorFlow 的使用者使用 Python 进行模型训练。

2.2 Keras 的简介

Keras 是一个用 Python 编写的开源神经网络库，能够在 TensorFlow、Microsoft Cognitive Toolkit、Theano 上运行，Keras 旨在快速实现深度神经网络，专注于模块化以及可扩展性，是 ONEIROS 项目的部分研究产物。TensorFlow 的核心库提供了对 Keras 的支持，它提供了更高级更直观的抽象集。Keras 包含了例如层、目标、激活函数、优化器和一系列许多常用神经网络构建块的实现，可以让开发者更轻松的处理图像和文本数据。

除了标准神经网络以外，Keras 还提供了对卷积神经网络和循环神经网络的支持，也对一些常见的实用公共层如 Dropout、归一化和池化层提供支持。

2.3 OpenCV 的简介

OpenCV 是一个跨平台的计算机视觉库，致力于 CPU 密集型的任务，是一个为了推进机器视觉研究，提供一套开源且优化基础库，让开发者不需要自己去实现去自己造轮子的项目。OpenCV 用 C++语言进行边写，主要接口也是 C++但是也有大量对于 Python/Java/MATLAB/OCTAVE 等语言的接口。OpenCV 被广泛应用于增强现实、人脸识别、手势识别、人机交互等诸多计算机视觉领域。

2.4 卷积神经网络

卷积神经网络对于传统的神经网络来说，不只是形式上的改进，也是功能上的改进，卷积神经网络中的每一个卷积核都是一个提取图像特征信息的提取器，在输入图像上进行卷积操作，提取出对应感受野的图像局部特征信息，最后将输入的图像转变成一个储存有特征信息的矩阵或向量。而 CNN 对于二维图像的比例缩放、平移等变形具有高度的不变性的特点，所以 CNN 在图像处理领域上就有邻域高度相关性的数据处理能力，可以提取到有效地特征表达并且能自动学习。

2.5 Adam 学习率优化算法

Adam 是在深度学习模型里用于代替随机梯度下降的一种算法，它实际上是将 Momentum 和 RMSProp 结合在一起，同时综合了 RMSProp 和 AdaGrad 两种算法的最优性能，同时还能提供解决稀疏梯度和噪声问题的优化算法。AdaGrad 算法是调整每个维度上的学习率，调整的依据是每个维度的梯度值大小，这样可以避免同一学习率很难去适应所有维度的问题；而 RMSProp 是一种脱胎于 AdaGrad 的非常高效但是并没有发表的适应性学习率的算法，RMSProp 使用了梯度平方的滑动平均，基于梯度的大小来更新每个权重的学习率，使它逐渐变小或不变，效果与 AdaGrad 方法同样优秀，但是这会导致在前期学习率以一种非常快的速度下降，但是到了后期学习率已经很低了却还是没有找到一个比较完美的解决方案时，到了后期会非常慢甚至有可能找不到有用的解。

Adam 可以看做是 RMSProp 的动量版，是在 RMSProp 上进行改进的，Adam 不仅对累积状态的变量进行指数加权平均，还对于每一个小批量的梯度也进行了指数加权平均。

2.6 关于手势识别项目的计划

2.6.1 RealSense 和 librealsense2.0 计划和环境配置

在本篇论文中，初步选择的硬件是使用 Intel 的 Realsense D435i 摄像头，以及软件方面使用 Python+TensorFlow。

首先在官网下载并安装 5 个 SDK 分别是可视化界面、debug 工具、代码示例、深度质量工具、第三方技术集成（Python、C#、C++等），看到 macOS 仅仅支持 SDK 的部分功能，所以初步决定使用 Win10 的 1903 64 位版本来作为开发环境。然后要进行 CMake 的安装以及编译，在 windows 的系统设置里打开 Win10SDK 的支持，然后下载 CMake 文件并解压，安装 gcc 和 g++，使用命令行进入 cmake 文件目录并 install，用 cmake --version 来查看是否安装成功。第二步进行 librealsense2.0 的安装

在 github 上下载 librealsense2.0 的压缩包并使用命令解压并进入目录，使用以下命令进行 cmake 编译

- `mkdir build`
- `cd build`
- `cmake ../ -DBUILD_PYTHON_BINDINGS:bool=true`

Note: To force compilation with a specific version on a system with both Python 2 and Python 3 installed, add the following flag to CMake command: `-DPYTHON_EXECUTABLE=[full path to the exact python executable]`

- `make -j4`
- `sudo make install`

图 1 Win10 下 librealsense 安装命令

在使用 make -j4 命令时进行了 2 小时的执行过程，-j*是执行过程使用 CPU 核心数，根据 CPU 情况使用 make -j16 使执行过程缩短为 20 分钟。

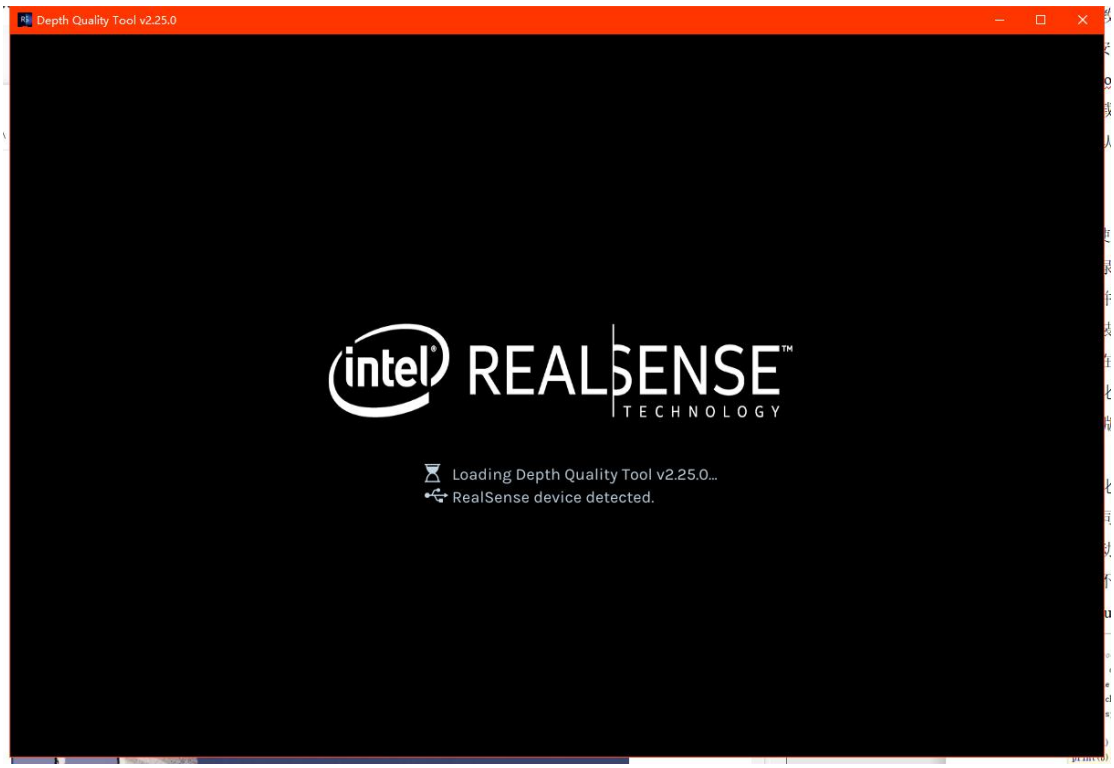


图 2Realsense 界面

由于 Windows 下每次 Python 的编译都需要 cmake 进行编译，所以决定将配置环境改为是 Ubuntu18.04 64 位系统，进行 RealSense D435i 的环境配置。使用 VMware 进行 Ubuntu 的虚拟机安装，在官网查知最新固件为 RealSense 2.0，根据 github 上的教程进行 pyrealsense2.0 和 librealsense 的环境配置：

安装完系统后第一时间安装 pip3 然后开始配置相应的 python 模块。然后在装 matplotlib 模块的过程中下载速度一直处于 2kb/s 的速度，需要将默认下载源更换为国内源，通过修改配置文件添加下图所示代码将 apt 和 pip3 的默认下载源更换为清华的镜像源。

```
[global]
index-url = https://mirrors.aliyun.com/pypi/simple
```

使用命令将 apt-get 进行更新，然后使用 uname -r 命令确认内核版本。

在 github 上拉取 librealsense2.0 源码并解压后确认没有连接到 realsense 摄像头以后进行以下命令操作

```
sudo apt-get install git libssl-dev libusb-1.0-0-dev pkg-config libgtk-3-dev
sudo apt-get install libglfw3-dev libgl1-mesa-dev libglu1-mesa-dev
./scripts/setup_udev_rules.sh
./scripts/patch-realsense-ubuntu-lts.sh
```

图 3Ubuntu 下 librealsense 安装命令

其中第一个命令是构建 librealsense 的二进制文件，并且同时安装受影响的内核模块所需的核心软件包，第二个命令是安装用于 ubuntu18 版本的软件包，第三个命令是运行位于 librealsense 根目录中的 Intel Realsense 权限脚本，第四个命令用于构建和应用模块，并且适用于特定的具有 LTS 内核的 ubuntu18 版本系统，这个命令需要执行约 2 小时。执行过程中报“连接超时”错误， 因为执行过程中有外网资源，所以连接很慢，使用“git config --global url."https://".insteadOf git://”命令将资源通过 git 下载，因为之前将 git 的下载源更换成了国内源，所以同时解决更换为 git 后仍然会出现超时的错误。

```
正克隆到 './ubuntu-xenial-Ubuntu-hwe-4.13.0-45.50_16.04.1'... fatal: unable to access 'https://kernel.ubuntu.com/ubuntu/ubuntu-xenial.git/':
Failed to connect to kernel.ubuntu.com port 443: 连接超时
```

图 4 git 连接超时报错

在执行构建模块之前先使用“sudo apt-get install libssl-dev ”命令来进行 Openssl 库的安装再进行模块的构建。执行完成后插上 realsense 摄像头并执行命令，

无报错则安装驱动成功。

使用“cmake ../ -DBUILD_EXAMPLES=true ”命令进行 cmake 编译 python 源码为 c++运行，其中“-DBUILD_EXAMPLES=true”命令的作用是演示和教程

2.6.2 Pytorch 计划和环境配置

Pytorch 官网上进行 pip+pytorch+cuda 对应版本的一键配置，最新的 CUDA 版本是 10.2，pytorch 只支持 10.1 版本的 CUDA，进行 CUDA 版本的回退，因为 NVIDIA 官网只有 10.0/10.2 版本的 CUDA，选择 10.0 版本的 CUDA 进行安装。

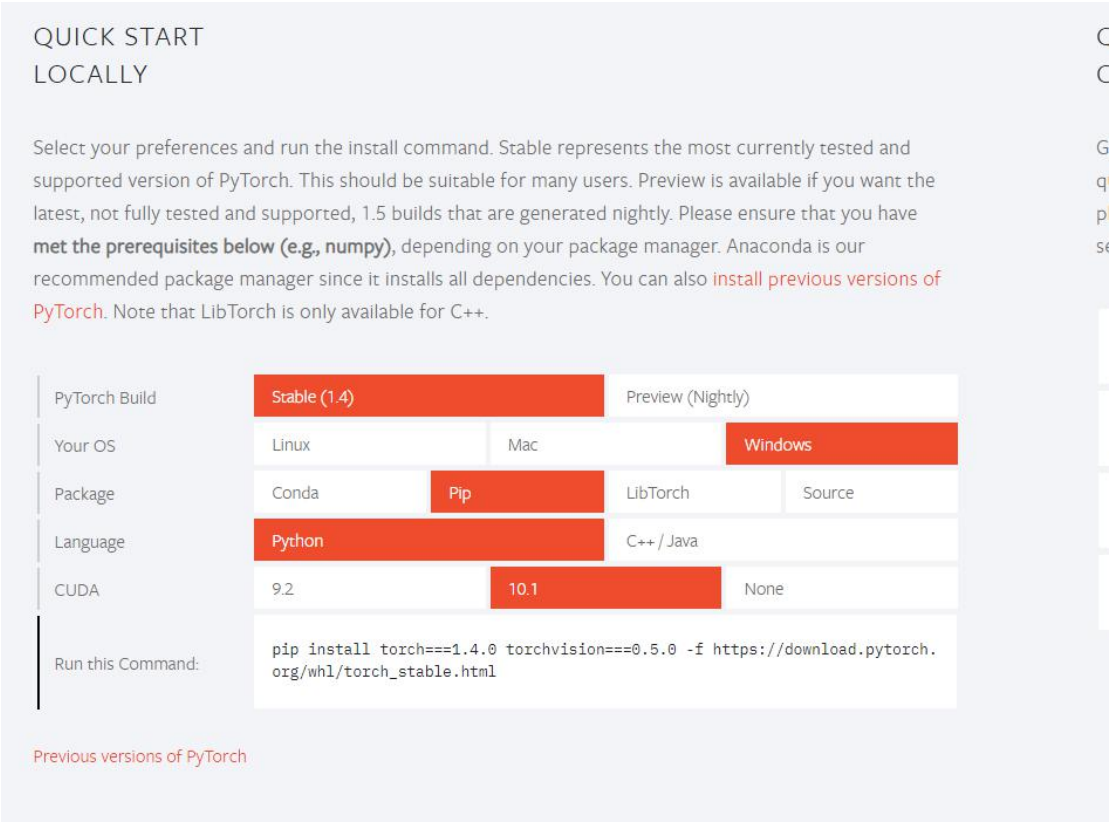


图 5 Pytorch 官网下载安装界面

2.6.3 TensorFlow+Keras 计划和环境配置

在此次环境配置中，初步决定使用的是 Vmware 虚拟机为基础的 Ubuntu18.04 版本的系统，配置好 python3.6 版本后开始 TensorFlow 相关环境的安装与配置。

使用“pip install tensorflow ”命令安装 TensorFlow，并根据官网要求安装 cuDNN SDK，使用的 CUDA 为安装 pytorch 时安装的 10.0 版本。进行编译后显示

CUDA 版本过旧，更新 CUDA 至 10.2。运行代码过程中报以下错误：

```
ImportError: Could not find 'cudart64_100.dll'
```

图 6 CUDA 报错

原因是 CUDA，cuDNN 和 TensorFlow 的版本不一致导致的，最新的 CUDA 是 10.2 但是最新的 cuDNN 仅仅支持 10.0 版本的 CUDA，而 TensorFlow 只支持 1 0.1 以上的 CUDA。将 CUDA 回退到 10.0 版本以后不再报以上错误，但是再次报新的错误，因为 TensorFlow1.13 才支持 CUDA10.0 版本，TensorFlow2.0 版本仅支持 CUDA10.1 以上的版本，而 TensorFlow2.0 更新了许多语法，最终决定使用 tf2. 0 版本进行开发。

经查询资料无果，尝试同时安装 10.0 和 10.2 版本，并把 10.0 版本中的对应 d 在运行代码进行模型训练后发现效率极低，进行运行情况追踪后发现没调用 GPU 进行模型训练，使用调用 GPU 的语句后报错，发现是安装 TensorFlow1.13,版本，默认调用 CUDA10.0，安装对应旧版的 TensorFlow-gpu，发现还是无法调用 GPU 进行训练。再次自查后发现虚拟机无法调用现实显卡，放弃使用虚拟机进行开发，改为使用 Windows10 的 1903 版本进行开发，再次进行环境配置。

由于 win10 系统下的 pip 已经更新到最新版本，所以自动安装 TensorFlow2.0，以上默认安装旧版 TensorFlow 的问题被解决，完成环境配置。

3 CNN 项目相关设计

3.1 基于 CNN 的手势识别方法

一个完整的基于视觉手势识别系统的结构包括：图像采集、预处理、特征提 cc 取和选择、分类器设计以及手势识别。其大致流程如下图：

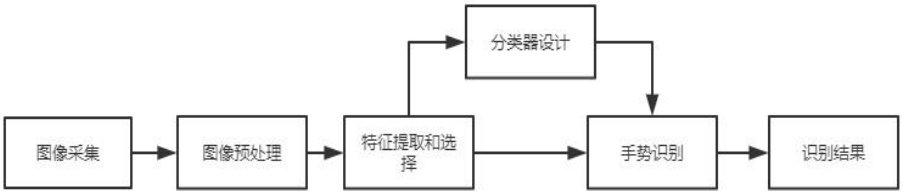


图 7 手势识别系统流程图

传统的手势识别方法是通过边缘检测，提取局部的手势区域，再通过特征提取来利用特征向量来完成手势识别。这种方法的学习能力往往比较差，只能用于简单的场景，对于复杂的场景的适应性较弱。而基于 CNN 神经网络的手势识别方法，较于前者拥有对于复杂场景更强的适应性

3.1.1 手势图像的预处理

在系统的实际应用中，手势图像只是录取的图像中很小的一部分，为了避免手势之外的无关特征对于手势识别会造成的影响，手势识别系统一般在进行手势识别之前会对图像进行预处理，例如手势区域的检测，手势分割以及图像二值化操作。将处理完的图像逐帧输入进神经网络的模型进行识别。

3.1.2 CNN 神经网络模型设计

(1) 输入层是 100*100 大小的图像矩阵，后面链接一个卷积层进行计算，其中有 32 个尺寸为 3*3 的卷积核，步长为 1，经过卷积计算得到 32 张 98*98 的局部特征图片

(2) 卷积层后插入一层 BatchNormalization 和 ReLU 激活函数来将这层的特征值分布拉回标准正态分布，将特征值落在激活函数对于输入较为敏感的区间，使得梯度变大来避免梯度消失，这样也同时可以加快收敛过程。

(3) 之后将上一步得到的输出用 32 个 3*3 的卷积核进行计算，计算方式同第（1）步，最后得到 32 张尺寸为 96*96 的特征图片

(4) 卷积层以后插入一层 BatchNormalization 和 ReLU 激活函数，操作方式同第（2）步

(5) 接着将输出连接一个池化层对特征图片进行采样操作，其中核大小为（2*2），步长设置为 1，经过池化计算得到 32 张尺寸为 48*48 的特征图片

(6) 插入一层 dropout 层防止过拟合，失活系数为 0.5，之后通过全连接层计算后

将每一个特征矩阵转换为一维特征向量，全连接层的大小是 128

(7) 将全连接层的输出输入到一个分类器进行分类，使用 L2 正则优化。

(8) 在最后加入一个 Adam 算法作为模型的优化器，来通过计算梯度的一阶矩估计和二阶矩估计来为不同的参数设计独立的自适应学习率。

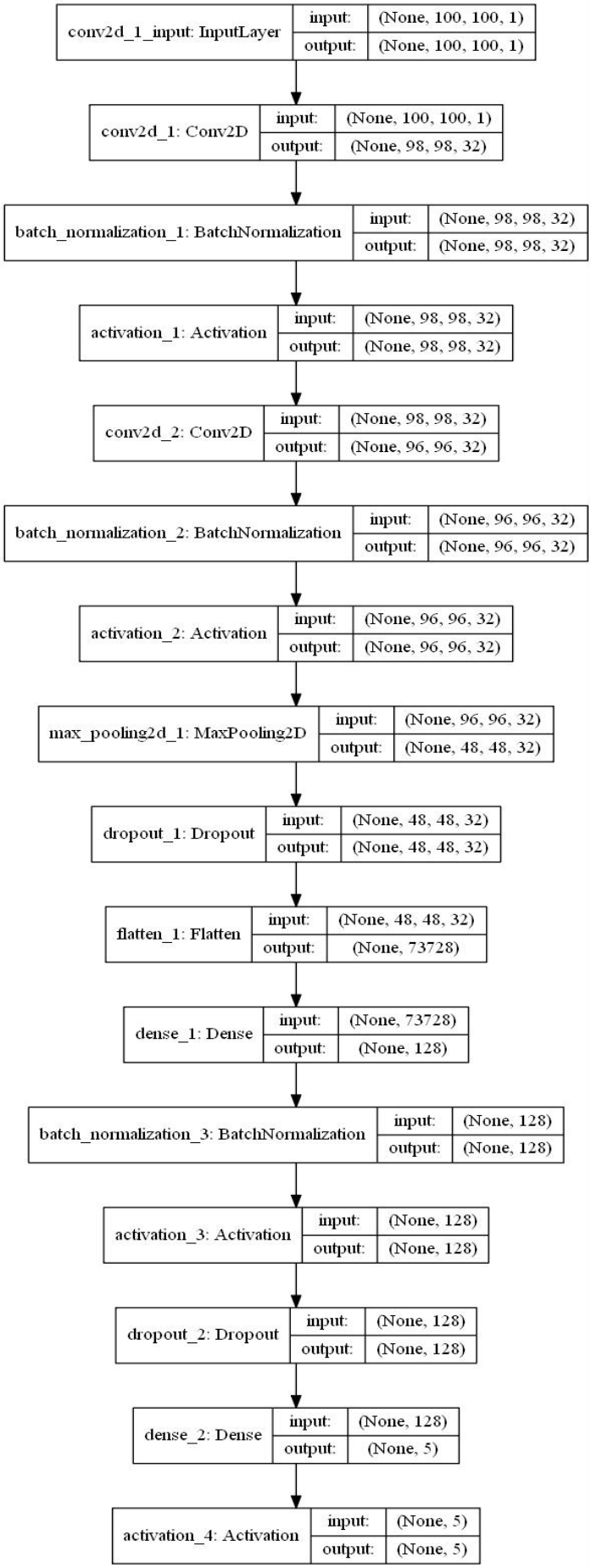

conv2d_1_input: InputLayer
input: (None, 100, 100, 1)
output: (None, 100, 100, 1)
conv2d_1: Conv2D
input: (None, 100, 100, 1)
output: (None, 98, 98, 32)
batch_normalization_1: BatchNormalization
input: (None, 98, 98, 32)
output: (None, 98, 98, 32)
activation_1: Activation
input: (None, 98, 98, 32)
output: (None, 98, 98, 32)
conv2d_2: Conv2D
input: (None, 98, 98, 32)
output: (None, 96, 96, 32)
batch_normalization_2: BatchNormalization
input: (None, 96, 96, 32)
output: (None, 96, 96, 32)
activation_2: Activation
input: (None, 96, 96, 32)
output: (None, 96, 96, 32)
max_pooling2d_1: MaxPooling2D
input: (None, 96, 96, 32)
output: (None, 48, 48, 32)
dropout_1: Dropout
input: (None, 48, 48, 32)
output: (None, 48, 48, 32)
flatten_1: Flatten
input: (None, 48, 48, 32)
output: (None, 73728)
dense_1: Dense
input: (None, 73728)
output: (None, 128)
batch_normalization_3: BatchNormalization
input: (None, 128)
output: (None, 128)
activation_3: Activation
input: (None, 128)
output: (None, 128)
dropout_2: Dropout
input: (None, 128)
output: (None, 128)
dense_2: Dense
input: (None, 128)
output: (None, 5)
activation_4: Activation
input: (None, 5)
output: (None, 5)

图 8 CNN 模型设计

4 Python+Keras+TensofFlow 训练模型

4.1.1 定义和初始化变量

如图 9 所示，初始化批处理数量，训练次数，分类个数、文件夹目录、模型名称等变量为在调用时传入的数据，方便调用时修改。

```
def __init__(self, batch_size, epochs, categories, train_folder, test_folder, model_name, type):
    # 批处理数量
    self.batch_size = batch_size
    # 训练次数
    self.epochs = epochs
    # 分类个数
    self.categories = categories
    # 训练集文件夹
    self.train_folder = train_folder
    # 测试集文件夹
    self.test_folder = test_folder
    # 模型名
    self.model_name = model_name
    # 手势类型
    self.type = type
    # 图片尺寸
    self.shape1 = 100
    self.shape2 = 100
```

图 9 初始化变量

4.1.2 读取数据集并对数据集进行处理

如图 10 所示，定义读取图像的函数“read_train_images”来对储存的图片进行处理，通过 os 模块对文件进行处理，os,listdir 方法读取“folder”目录下的文件列表，使用变量 file 在列表内进行循环读取训练集和测试集数据，将二维矩阵增加一维变成三维矩阵后加入 numpy 的 array 列表中。处理完数据以后将训练集和测试集图片进行归一化处理，并将图像和标签转换为 array 格式后再将标签转换为独热编码。因为读取出来的数据是无序的，转换为独热编码以后将离散特征的取值扩展到了欧式空间，离散特征的某个取值就对应了欧式空间的点，对于离散型的特征使用独热编码会让特征的距离计算更加的合理。

```python
def read_train_images(self, folder):
    """从文件夹中读取图像和标签，放回图像列表和标签列表"""
    img_list = []
    lable_list = []
    for file in os.listdir(folder):
        # print(folder)
        img = Image.open(folder + file)
        img = np.array(img).reshape(self.shape1, self.shape2, 1)
        img_list.append(img)
        lable_list.append(int(file.split('_')[1][0]))
    return img_list, lable_list

def train(self):
    train_img_list, train_lable_list = self.read_train_images(folder=self.train_folder)
    test_img_list, test_lable_list = self.read_train_images(folder=self.test_folder)
    # 测试集图像归一化，并将图像和标签转化为numpy中的array格式
    test_img_list, test_lable_list = np.array(test_img_list).astype('float32') / 255, np.array(test_lable_list)

    # 训练集图像归一化
    train_img_list = np.array(train_img_list).astype('float32') / 255
    train_lable_list = np.array(train_lable_list)

    # 训练集和测试集的标签转化独热编码
    train_lable_list = np_utils.to_categorical(train_lable_list, self.categories)
    test_lable_list = np_utils.to_categorical(test_lable_list, self.categories)
```

图 10 读取数据集并进行处理

4.1.3 基于 CNN 的模型训练

模型建立以后，需要对其进行训练，并且对模型的参数要进行调节，以达到优化的目的。本文采用了自行录制的手势作为数据集，选择了 5 类手势作为训练对象，训练集共 4904 张图片，测试集共 1498 张图片，共训练了 8 个模型，第 1 个模型每次训练选取的样本数为 32，共训练 300 次；第 2 个模型样本选取数为 48，训练 300 次；第 3 个模型每次训练选取样本数为 64，共训练 300 次；第 4 个模型每次选取 128 个样本，共训练 300 次。第 5 到第 8 个模型同样都是训练 500 次，每次训练所选取的样本数分别为 32/48/64/128。训练过程如图 11。

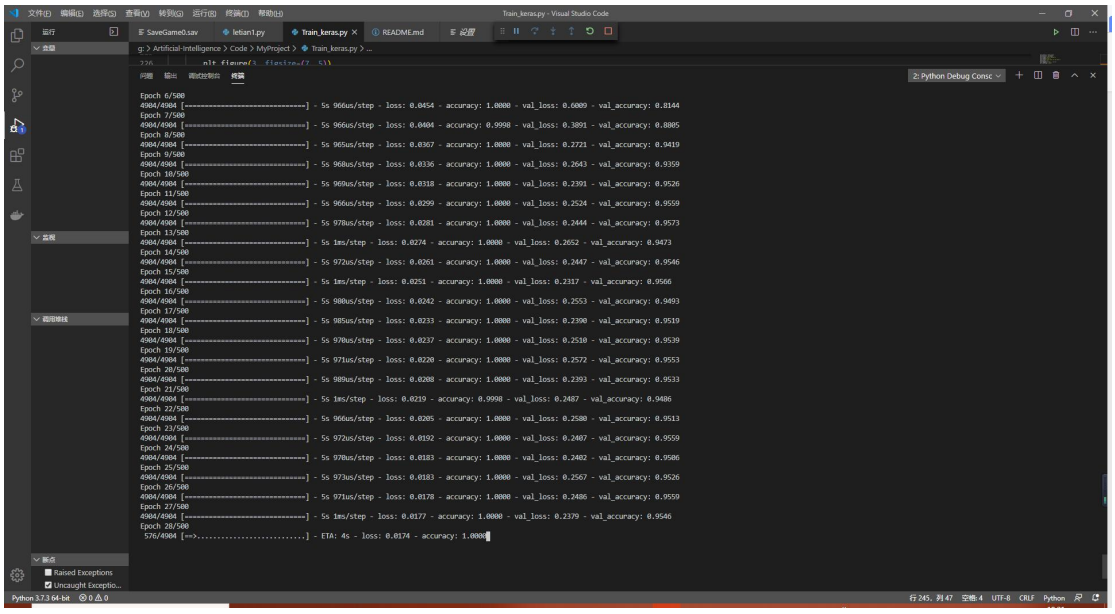


图 11 训练过程

4.1.4 对于不同参数模型的准确度数据分析

通过 Keras 内置的函数来创建误差矩阵，即混淆矩阵（Confusion Matrix）是用于评判模型结果的一个指标，比较适用于分类型的数据模型的模型评估，混淆矩阵拥有 3 级指标，通过样本采集，同时在能够得到真实数据的情况下，哪些数据是真实正确同时预测正确的（True Positive, TP），真实正确但是预测错误的（True Negative, TN），真实错误但是预测正确的（False Positive，FP），真实错误同时预测错误的（False Negative，FN），这样的四个基础指标我们称之为一级指标。在预测性的分类模型中，我们判断一个模型是否准确，在混淆矩阵中就是判断 TP 和 TN 占总数的比例是否大。但是矩阵中的四个指标都是个数，难以去判断模型的优劣，所以混淆矩阵在基础的 4 个一级指标上又拓展了 4 个二级指标分别是准确率（Accuracy）、精确率（Precision）、召回率（Recall 也称灵敏度 Sensitivity）、特异度（Specificity）。准确率是所有判断正确结果在总预测值中的比重，即 TP+TN 在总值中的比重；精确率是真实正确同时预测正确的值在预测正确的值中的比重；召回率是真实正确同时预测正确的值在全部真实正确的值中所占的比重；特异度则是在真实错误且预测错误的值在全部判断正确的值中的比重。通过这 4 个二级指标可以将混淆矩阵中的 4 个一级指标的结果转换为 0-1 之间的比率，并以此来进行标准化的一个模型的衡量，同时在这 4 个二级指标的基础上还可以产生一个三级指标，即 F1 Score，公式为（2*P*R）/（P+R），其中的 P 代表精确率（Precision），R 代表召回率（Recall），F1-Score 综合了精确率和召回率的结果，可以评估一个模型预测是否准确。

表 1 至表 5 训练次数同样为 300 次，每次训练选取的样本数分别为 32/48/64/128 的四个模型计算出来的 4 个二级指标和 1 个三级指标，图 12 则是对应的混淆矩阵：从准确度来分析，模型 1 的 95.33%准确度最高，同时模型 3 的 95.19%准确度相差不大，而模型 2 的 94.39%和模型 4 的 94.59%准确度相对来说就比较低；而 F1-Score 的平均值则是第 3 个模型的 95.20%最高，而第 1 个模型的 95.10%相差不大，而第 2 第 4 个模型的 F1-Score 则都不到 95%。综合模型准确度和 F1-Score 的数据说明在训练 300 次的时候选取的样本为 32 或 64 预测比较准确。

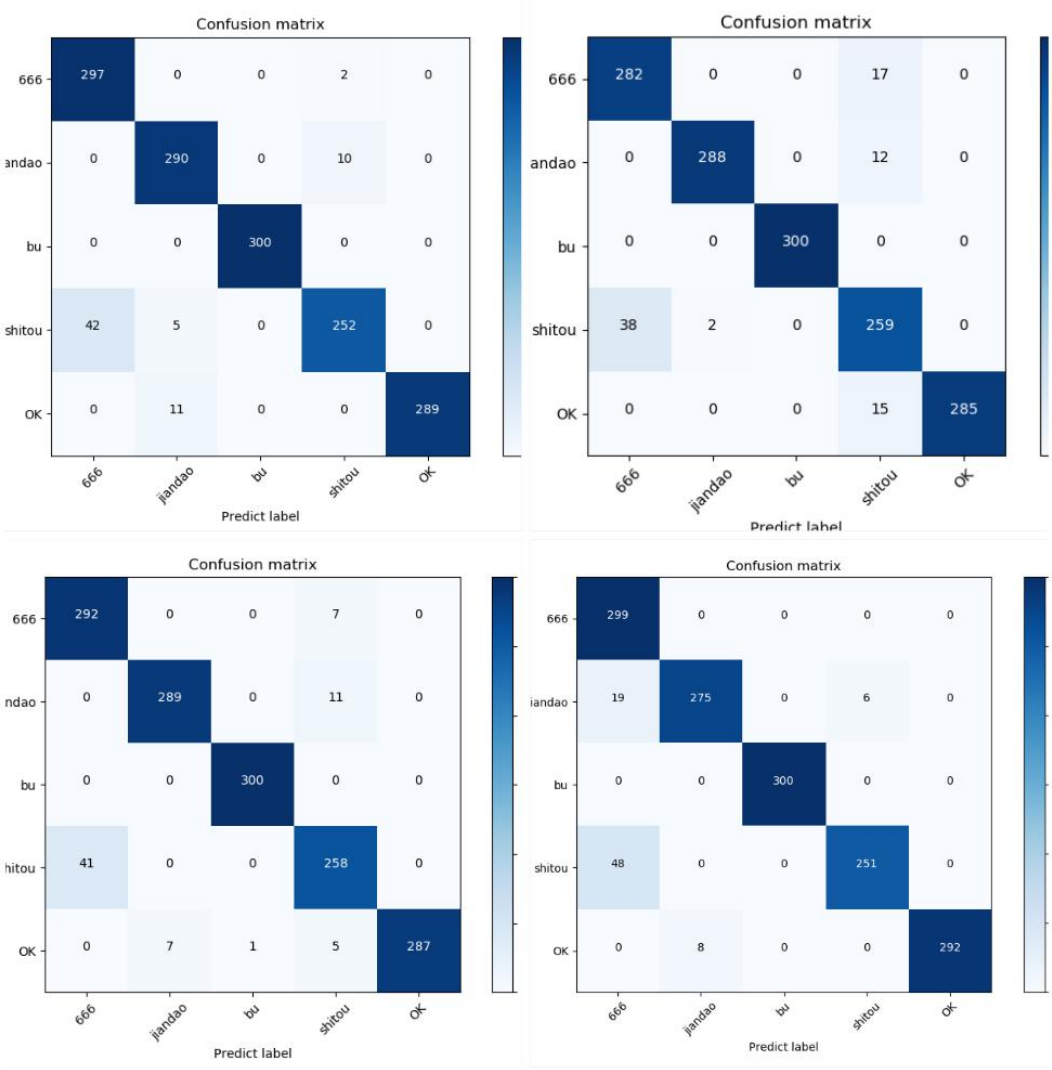


图 12 300 训练次数的四个模型对应的混淆矩阵

手势类型	精确率(Precision)	召回率(Recall)	特异度(Specificity)	F1-Score
六	87.61%	99.33%	96.50%	93.10%
剪刀	94.77%	96.67%	98.66%	95.71%
布	100.00%	100.00%	100.00%	100.00%
石头	97.30%	84.28%	99.42%	90.32%
OK	100%	96.33%	100%	96.33%

表 1 模型 1-32-300 混淆矩阵数据分析

手势类型	精确率(Precision)	召回率(Recall)	特异度(Specificity)	F1-Score
六	90.26%	92.98%	97.50%	91.60%
剪刀	92.61%	95.33%	99.42%	96.46%
布	99.01%	100.00%	99.75%	99.50%
石头	84.79%	87.63%	99.08%	86.18%
OK	100%	95.00%	100%	97.44%

表 2 模型 2-48-300 混淆矩阵数据分析

手势类型	精确率(Precision)	召回率(Recall)	特异度(Specificity)	F1-Score
六	87.69%	97.66%	96.58%	92.41%
剪刀	97.64%	96.33%	99.42%	96.98%
布	99.67%	100.00%	99.92%	99.83%
石头	91.81%	86.29%	98.08%	88.97%
OK	100.00%	95.67%	100.00%	97.79%

表 3 模型 3-64-300 混淆矩阵数据分析

手势类型	精确率(Precision)	召回率(Recall)	特异度(Specificity)	F1-Score
六	81.69%	100.00%	94.41%	89.92%
剪刀	97.17%	91.67%	99.33%	94.34%
布	100.00%	100.00%	100.00%	100.00%
石头	97.67%	83.95%	99.50%	90.29%
OK	100%	97.33%	100.00%	98.65%

表 4 模型 4-128-300 混淆矩阵数据分析

	32 样本	48 样本	64 样本	128 样本

准确度	95.33%	94.39%	95.19%	94.59%
F1-Score 平均值	95.09%	94.24%	95.20%	94.64%

表 5 不同样本数训练 300 次的准确度和 F1-Score 平均值

表 6 至表 10 是模型 5 到模型-8 的数据分析，这四个分别是在样本数为 32/48/64/128 时训练次数为 500 次所得出来的模型，图 13 是它们对应的混淆矩阵。由表可知，从准确度分析，模型 8 的 95.28%是准确度最高的，而模型 567 的准确度则相差不大，都在 94.5%左右，以模型 7 的 94.41%最低；从 F1-Score 的角度分析，模型 8 的 95.32%最高，模型 5 与模型 6 相差不大，而模型 5 的 94.61%则是四个模型中 F1-Score 最低的。由此可知，在训练 500 次时，选取样本数为 128 所得出的模型无论是准确度还是模型的精确率都是最高的，因此在训练 500 次时样本数为 128 较优。

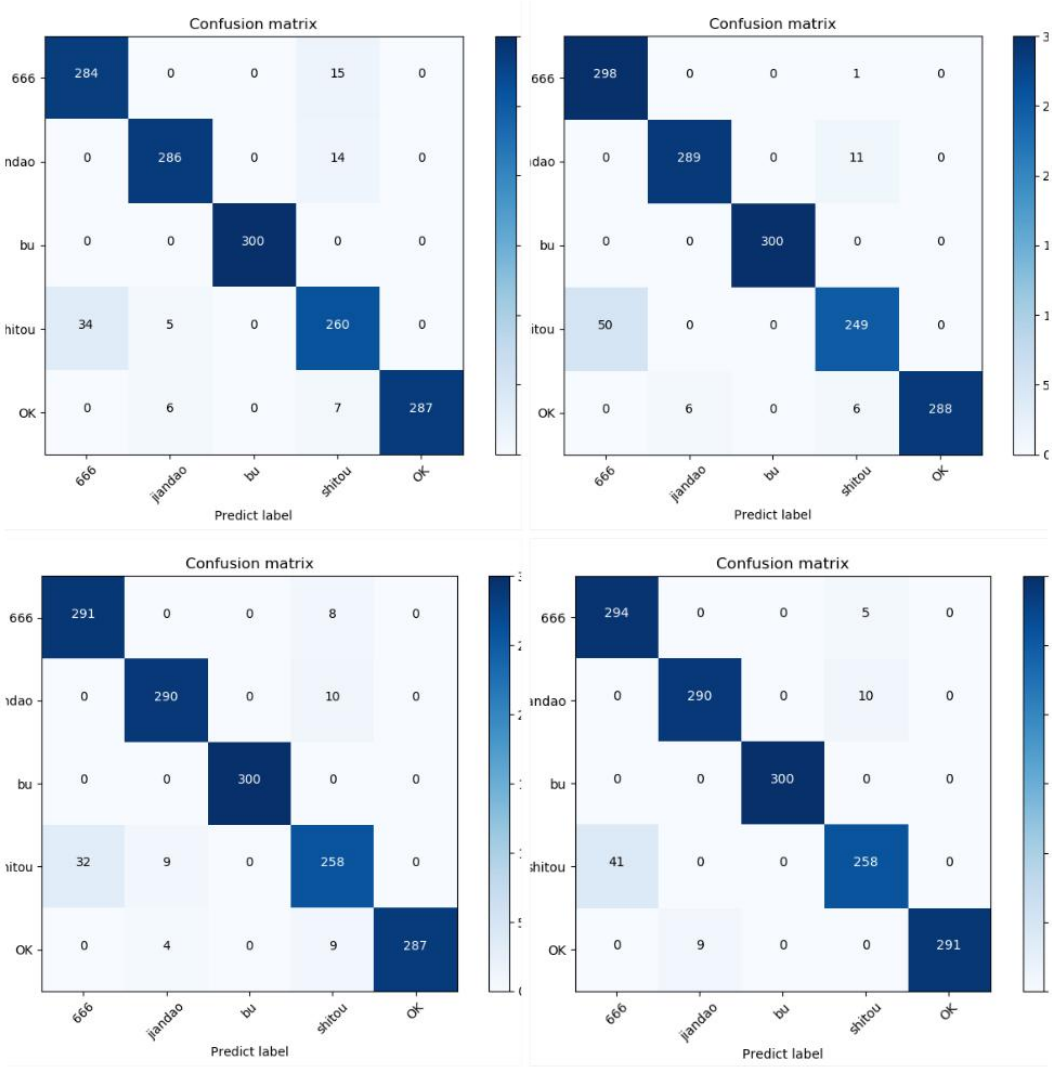


图 13 500 训练次数的四个模型对应的混淆矩阵

手势类型	精确率(Precision)	召回率(Recall)	特异度(Specificity)	F1-Score
六	89.31%	94.98%	97.16%	92.06%
剪刀	96.30%	95.33%	99.08%	95.81%
布	100.00%	100.00%	100.00%	100.00%
石头	87.84%	86.96%	97.00%	87.39%
OK	100.00%	95.67%	100.00%	97.79%

表 6　模型 4-32-500 混淆矩阵数据分析

手势类型	精确率(Precision)	召回率(Recall)	特异度(Specificity)	F1-Score
六	90.26%	92.98%	97.50%	91.60%
剪刀	92.61%	95.33%	99.42%	96.46%
布	99.01%	100.00%	99.75%	99.50%
石头	84.79%	87.63%	99.08%	86.18%
OK	100%	95.00%	100%	97.44%

表 7 模型 5-48-500 混淆矩阵数据分析

手势类型	精确率(Precision)	召回率(Recall)	特异度(Specificity)	F1-Score
六	87.69%	97.66%	96.58%	92.41%
剪刀	97.64%	96.33%	99.42%	96.98%
布	99.67%	100.00%	99.92%	99.83%
石头	91.81%	86.29%	98.08%	88.97%
OK	100.00%	95.67%	100.00%	97.79%

表 8 模型 6-64-500 混淆矩阵数据分析

手势类型	精确率(Precision)	召回率(Recall)	特异度(Specificity)	F1-Score
六	81.69%	100.00%	94.41%	89.92%
剪刀	97.17%	91.67%	99.33%	94.34%
布	100.00%	100.00%	100.00%	100.00%
石头	97.67%	83.95%	99.50%	90.29%
OK	100%	97.33%	100.00%	98.65%

表 9 模型 8-128-500 混淆矩阵数据分析

	32 样本	48 样本	64 样本	128 样本
准确度	94.59%	94.68%	94.41%	95.28%
F1-Score 平均值	94.61%	94.66%	95.18%	95.32%

表 10 不同模型训练 500 次时准确度和 F1-Score 平均值

5 实时手势识别

5.1 初始化

初始化变量和环境参数，如图 14 所示，预留键盘模拟器的接口可以进行手势模拟键盘操作。

```
def __init__(self, train_path, predict_path, gesture, train_model):
    # 初始化参数
    self.blurValue = 5
    self.bgSubThreshold = 36
    self.train_path = train_path
    self.predict_path = predict_path
    self.threshold = 60
    self.gesture = gesture
    self.train_model = train_model
    self.skinkernel = cv2.getStructuringElement(cv2.MORPH_ELLIPSE, (5, 5))
    self.x1 = 380
    self.y1 = 60
    self.x2 = 640
    self.y2 = 350
    self.cap_region_x_begin = 0.5  # 起点/总宽度
    self.cap_region_y_end = 0.8
    self.learningRate = 0

    # 判断是否工作的变量
    self.isBgCaptured = 0  # 布尔类型，背景是否被捕获
    self.triggerSwitch = False  # 如果正确，键盘模拟器将工作
```

图 14 实时识别变量初始化

5.2 视频流捕获和处理

代码如图 22 所示，首先打开摄像头并对获取到的视频流进行逐帧处理，确认摄像头打开后以后先对视频流做镜像反转处理，然后进行双边滤波处理达到保边降噪的处理，进行背景捕获以后，使用初始化过的学习计算前景掩膜，使用特定的结构来侵蚀图像之后使用掩膜移除静态的背景。将处理过的图像进行背景减法，再对图像进行边缘的腐蚀。将原始图像与背景减法和腐蚀后的蒙版做“与”操作。最后进行灰度处理和高斯滤波处理图像，进行阈值分割以后，然后运用cv2.findContours()函数寻找轮廓，通过计算每一块轮廓的面积，进行比较确定最大的轮廓为手掌轮廓，找出最大轮廓以后，使用“cv2.convexHull()”函数得出最大轮廓点集的凸包，再画出最大区域轮廓和凸包的轮廓，如图 15 所示。

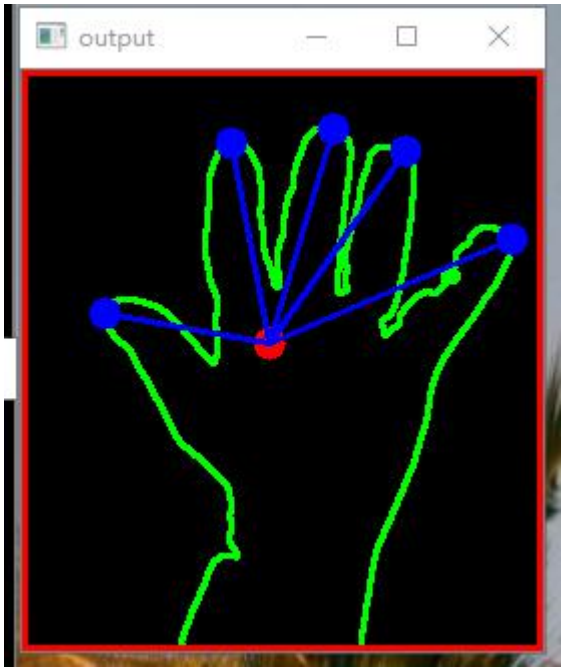


图 15 画出手部轮廓

调用之前训练好的模型来对进行手势预测，使用 Keras 模块的 load_model 函数来调用模块，调用过程如图 16 所示。

```
elif k == ord('p'):  # 预测手势
    predict = True
    while True:
        model_name = input('请输入模型的名字:\n')
        if model_name == 'exit':
            break
        if model_name in os.listdir('./'):
            print('正在加载{}模型'.format(model_name))
            p_model = load_model(model_name)
            break
        else:
            print('模型名字输入错误，请重新输入，或输入exit退出')
```

图 16 调用模型

最后在识别框内的手势会根据训练好的模型进行识别并且显示在识别框的左上角，各手势的实时识别结果如图 17-图 21 所示，在对实时视频输入流进行处理后的每一步都会输出一个对应的视频框以显示处理结果，最后会取手部轮廓并判断指尖数量，从手掌中心连接到指尖。

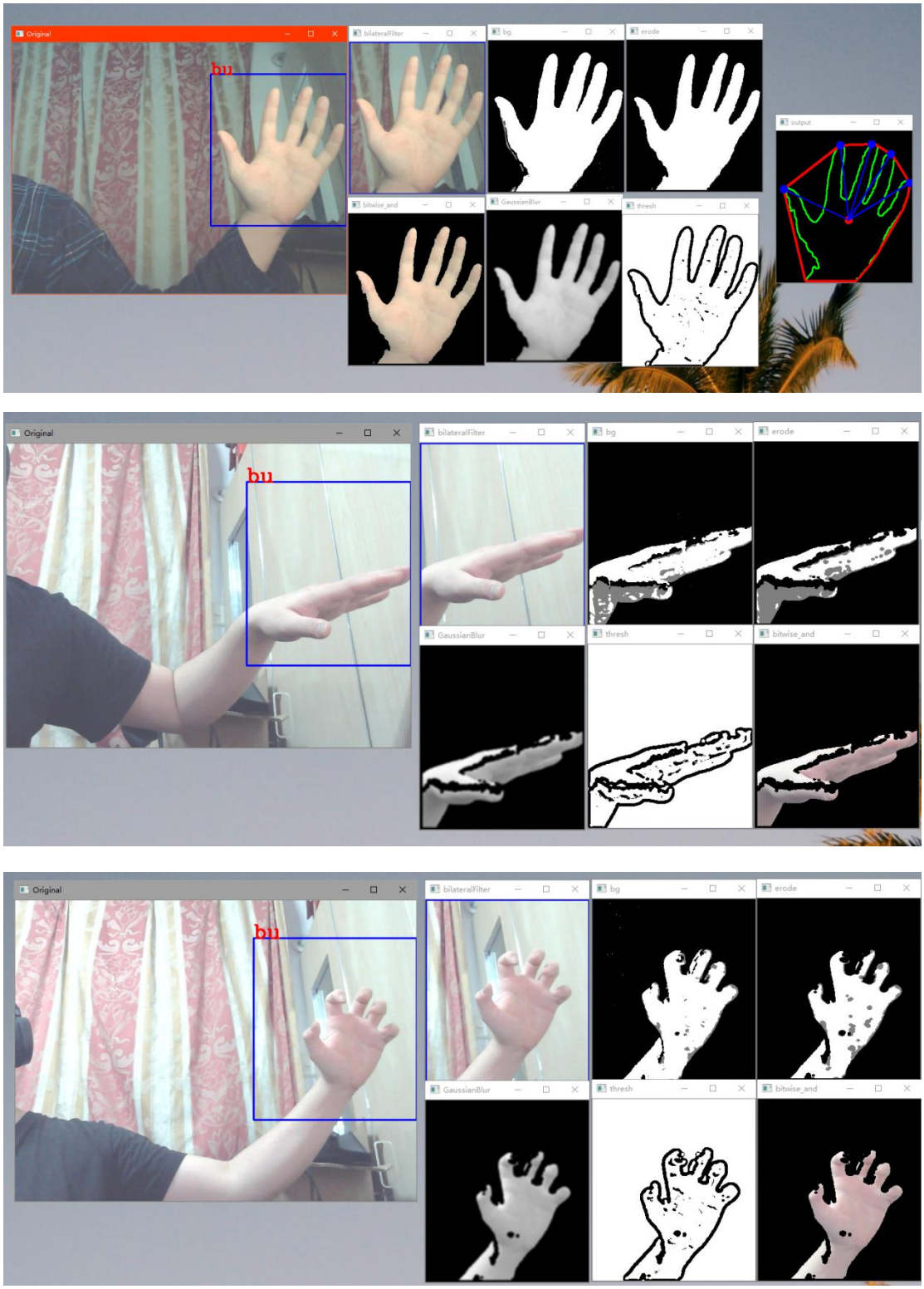
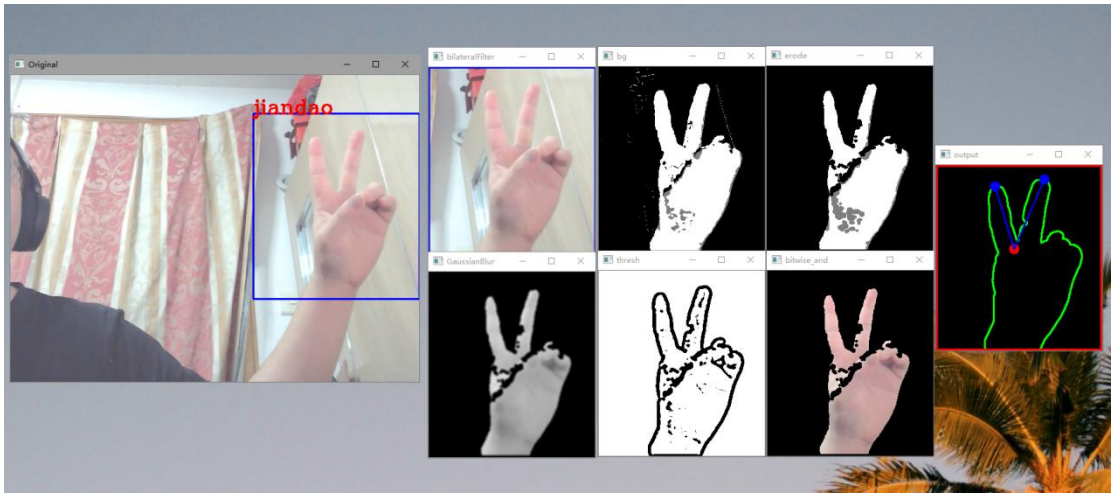
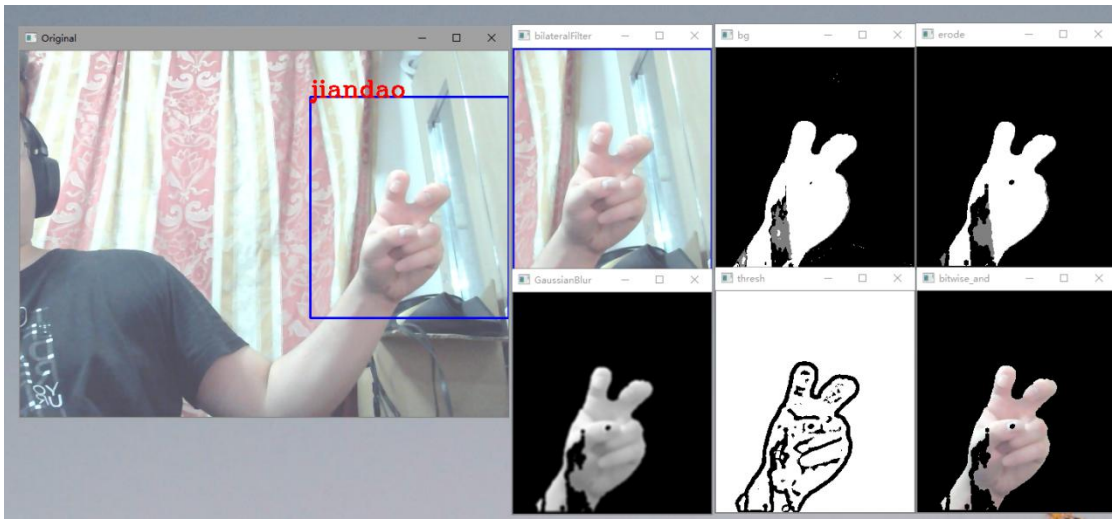
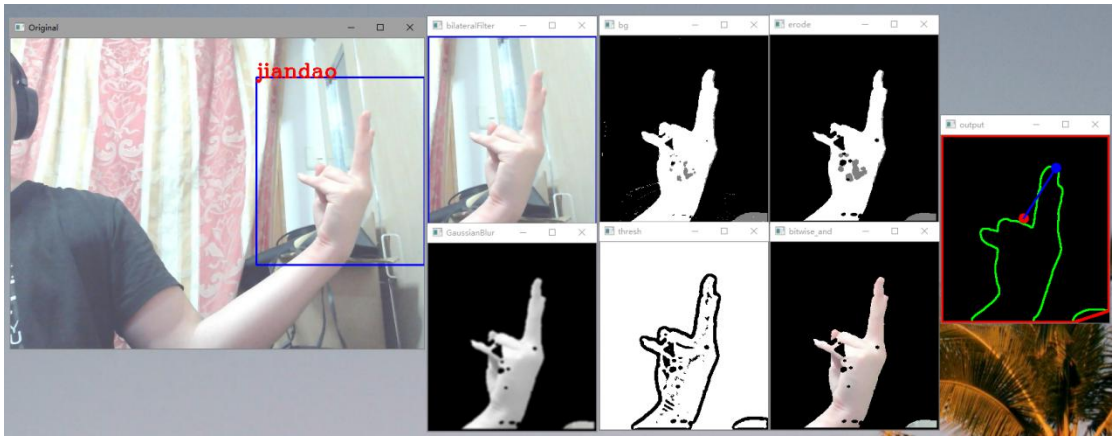


图 17 "布"各角度手势的实时识别


Original
jiandao
bilateralFilter
bg
erode
output
GaussianBlur
thresh
bitwise_and


Original
jiandao
bilateralFilter
bg
erode
GaussianBlur
thresh
bitwise_and


Original
jiandao
bilateralFilter
bg
erode
output
GaussianBlur
thresh
bitwise_and

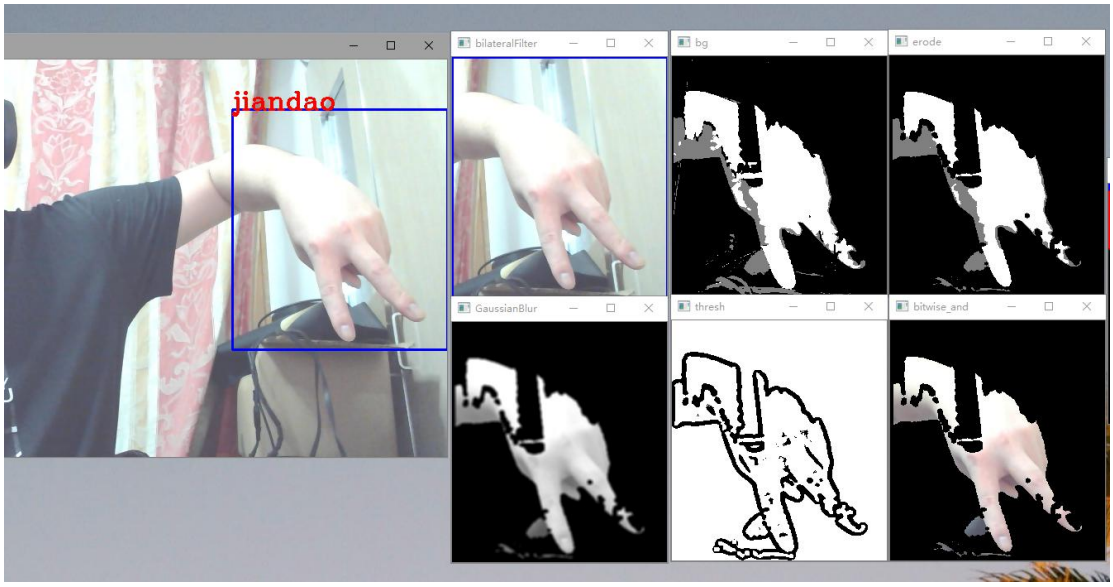
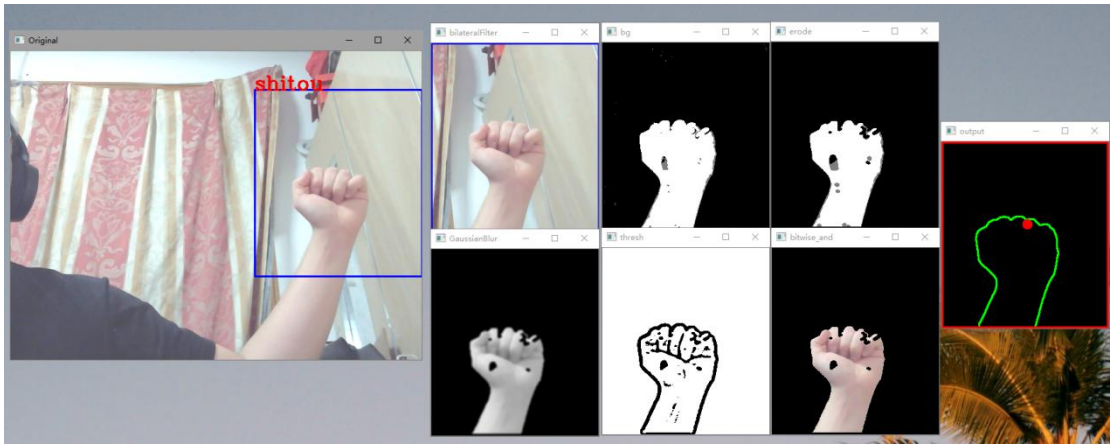
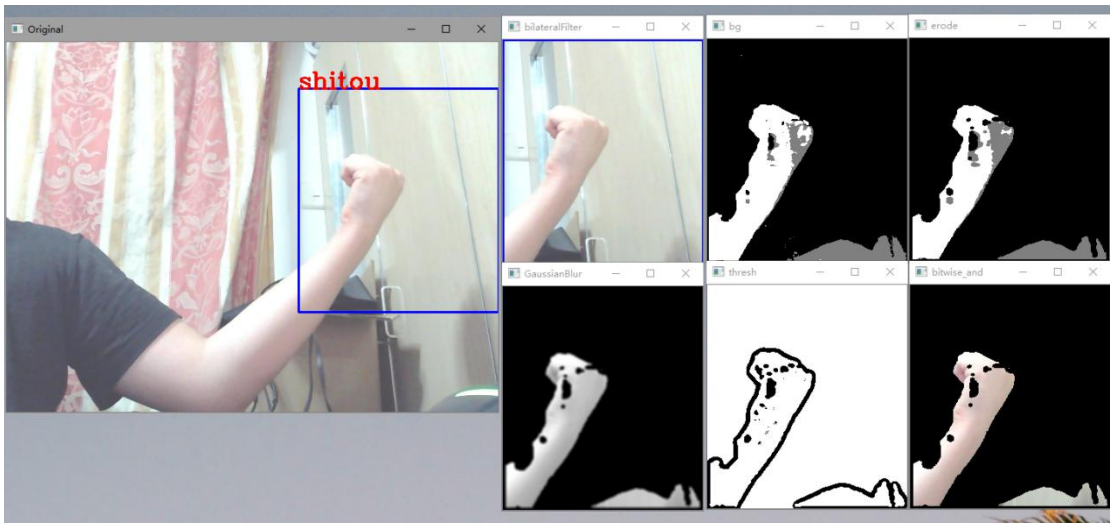


图 18 “剪刀”各角度手势的实时识别





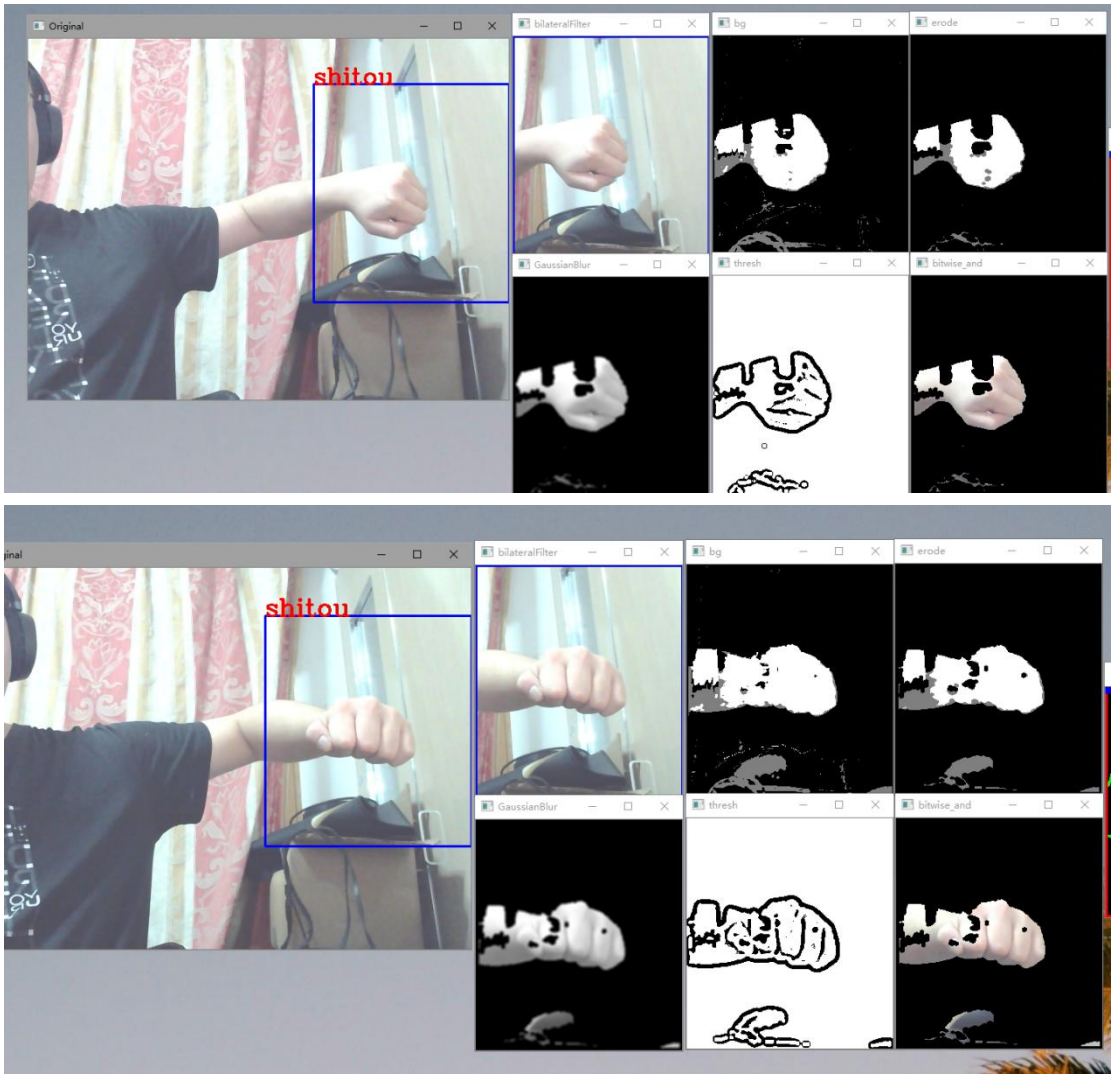
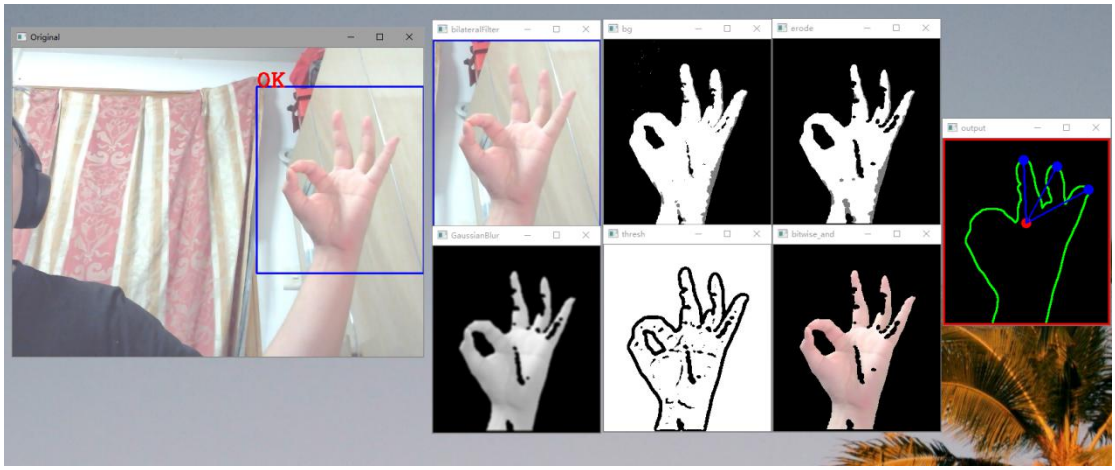


图 19 “石头”各角度手势的实时识别



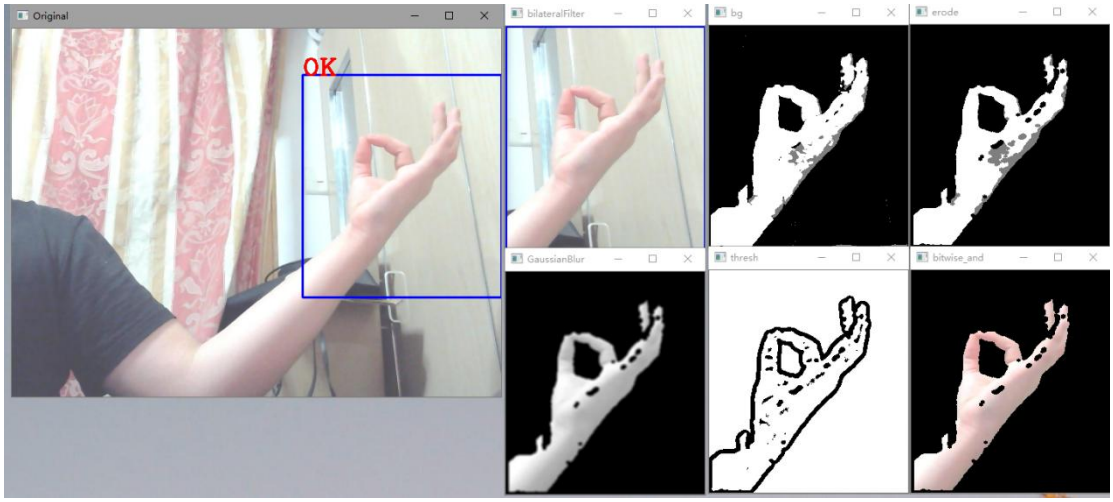
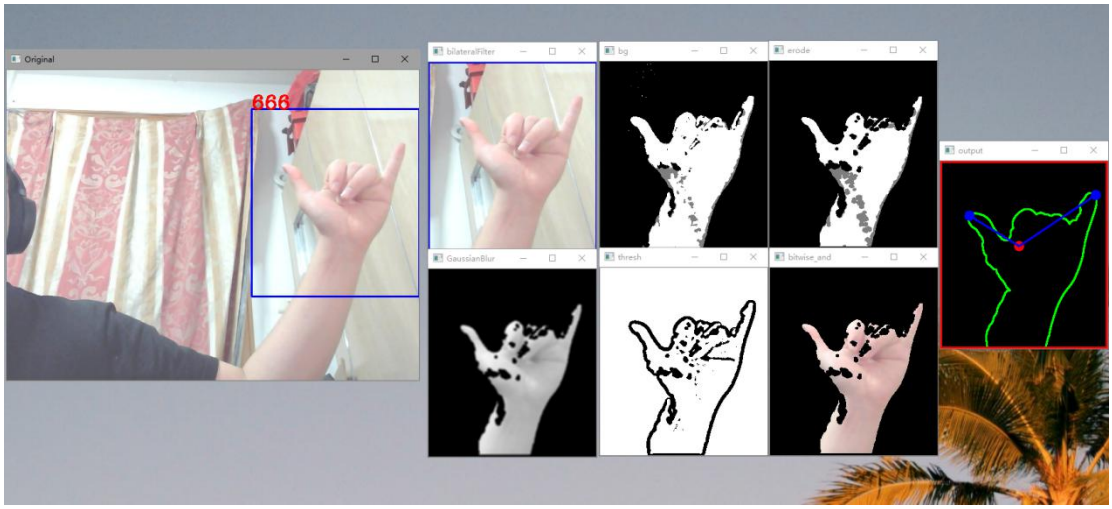
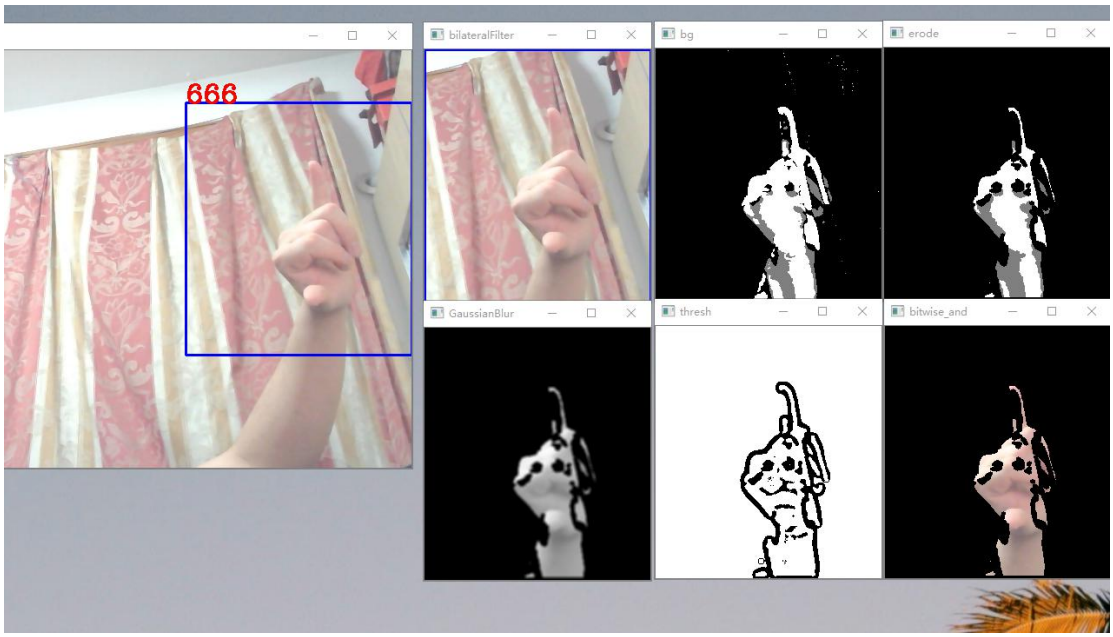


图 20 “OK”各角度手势的实时识别





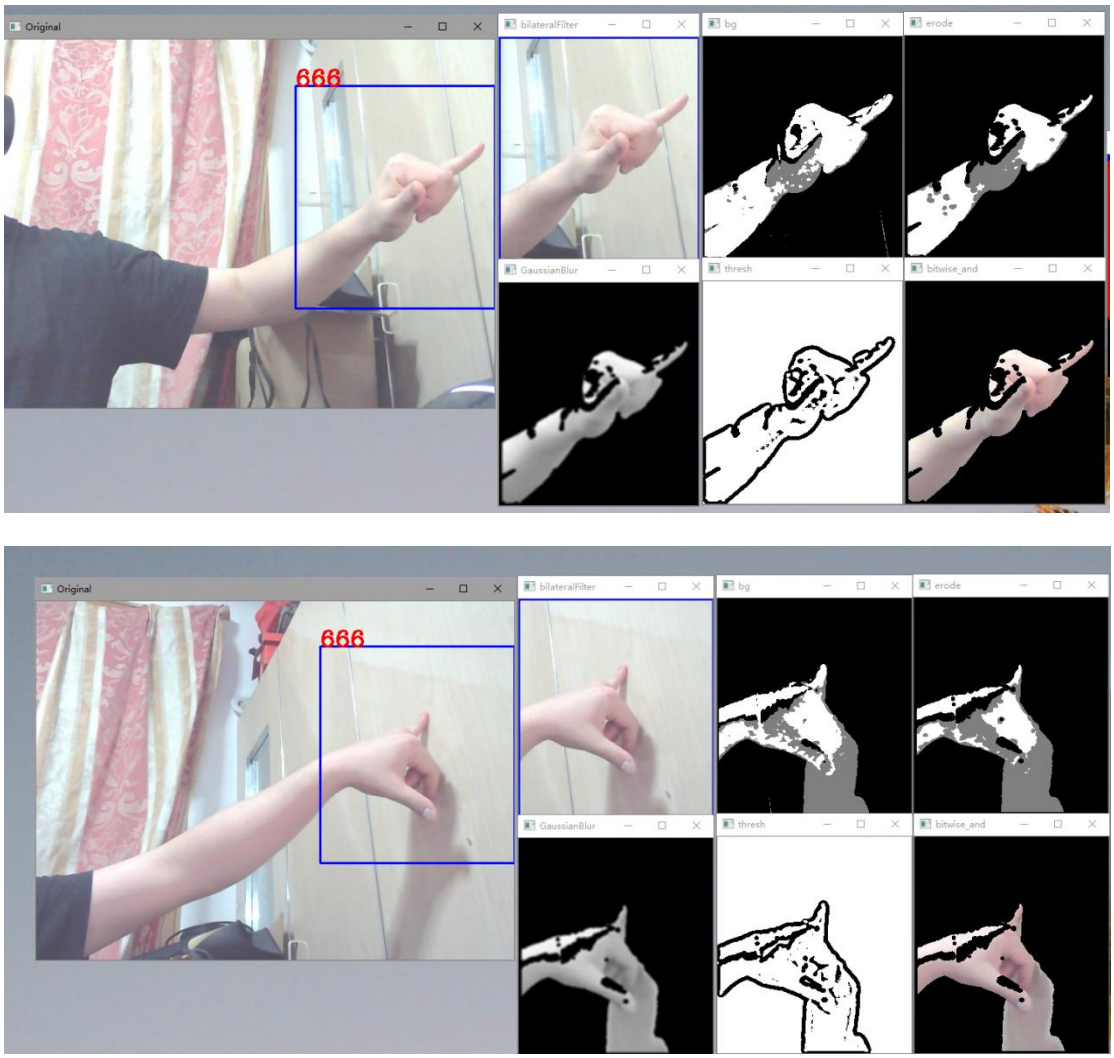


图 21 “六”各角度手势的实时识别

```python
# 背景减法创建及初始化
bgModel = cv2.createBackgroundSubtractorMOG2(0, self.bgSubThreshold)

while camera.isOpened():  # 摄像头已打开
    # 读取视频帧b
    ret, frame = camera.read()
    # 镜像转换
    frame = cv2.flip(frame, 1)

    cv2.imshow('Original', frame)
    # 双边滤波
    frame = cv2.bilateralFilter(frame, 5, 50,100)

    # 绘制矩形，第一个为左上角坐标(x,y)，第二个为右下角坐标
    # rec = cv2.rectangle(frame, (220, 50), (450, 300), (255, 0, 0), 2)
    rec = cv2.rectangle(frame, (self.x1, self.y1), (self.x2, self.y2), (255, 0, 0), 2)


    if self.isBgCaptured == 1:  # isBgCaptured == 1 表示已经捕获背景
        fgmask = bgModel.apply(frame, learningRate=self.learningRate) #计算前景掩膜
        kernel = np.ones((3, 3), np.uint8)
        fgmask = cv2.erode(fgmask, kernel, iterations=1) #使用特定的结构元素来侵蚀图像。
        img = cv2.bitwise_and(frame, frame, mask=fgmask) #使用掩膜移除静态背景
        # img = img[0:int(cap_region_y_end * frame.shape[0]),int(cap_region_x_begin * frame.shape[1]):frame.shape[1]]  # 剪切右上角矩形框区域
        # cv2.imshow('mask', img)


        # 定义roi区域，第一个为y的取值，第2个为x的取值
        # frame = frame[50:300, 220:450]
        frame = img[self.y1:self.y2, self.x1:self.x2]
        cv2.imshow('bilateralFilter', frame)
        # 背景减法运动检测
        bg = bgModel.apply(frame, learningRate=0)
        # 显示背景减法的窗口
        cv2.imshow('bg', bg)
        # 图像边缘处理--腐蚀
        fgmask = cv2.erode(bg, self.skinkernel, iterations=1)
        # 显示边缘处理后的图像
        cv2.imshow('erode', fgmask)
        # 将原始图像与背景减法+腐蚀处理后的蒙版做"与"操作
        bitwise_and = cv2.bitwise_and(frame, frame, mask=fgmask)
        # 显示与操作后的图像
        cv2.imshow('bitwise_and', bitwise_and)
        # 灰度处理
        gray = cv2.cvtColor(bitwise_and, cv2.COLOR_BGR2GRAY)
        # 高斯滤波
        blur = cv2.GaussianBlur(gray, (self.blurValue, self.blurValue), 2)
        cv2.imshow('GaussianBlur', blur)

        # 使用自适应阈值分割(adaptiveThreshold)
        thresh = cv2.adaptiveThreshold(blur, 255, cv2.ADAPTIVE_THRESH_GAUSSIAN_C, cv2.THRESH_BINARY, 11, 2)
        cv2.imshow('thresh', thresh)

        Ges = cv2.resize(thresh, (100, 100))
        # 图像的阈值处理(采用ostu)
        # _, thresh = cv2.threshold(blur, 0, 255, cv2.THRESH_BINARY+cv2.THRESH_OTSU)
        # cv2.imshow('threshold1', thresh)
```

图 22 逐帧视频处理

6 总结与展望

随着各种可穿戴设备的兴起以及普及，手势交互成为令得智能生活普及必不可少的一部分，这让如何提高手势的检测和识别的准确度和效率成为研究者们的重点课题。以往的手势识别仅仅是通过计算机视觉领域的技术来进行图像上的处理来判断手势，而随着人工智能-深度学习的发展热潮的掀起，尤其是卷积神经网络在机器视觉领域的优秀表现，这无疑是将机器视觉的发展往前推了一大步。本项目是研究基于卷积神经网络的对手势的检测、识别、以及手掌和指尖的定位算法。主要在研究 Intel 的 Realsense 深度实感摄像头和 Pytorch、Keras、TensorFlow 这三个深度学习框架上的区别以及适用性，对于三个框架的体量、可迁移性、代码复用性以及上手的难易度进行了一个区分。Pytorch 比较适合数学基础比较好的人，比较适合做研究，而 keras 更适合做一个程序员的项目。最终选择的是

Keras+TensorFlow 做后台的组合，仅仅是下载了 TensorFlow 的 python 模块作为运行 Keras 的基础，比起完整地使用 TensorFlow 来说更为轻量，而 Keras 在 TensorFlow2.0 也被加入其中成为 TensorFlow 的一个内置模块则更是降低了其学习难度，而作为对于图像的处理 OpenCV 模块的表现非常优秀，许多内置的函数令开发人员在处理图像的时候不再需要去单独实现一个函数，只需要调用即可。

但是本文存在的不足仍然非常多，接下来就是对于下一步工作的展望以及思考：

1. 对于手势，我的研究仅仅是建立在一个很简单的室内背景的使用场景，通过单目摄像头的采集来标注识别了五种手势，仅仅能满足一些简单的人机交互场景，而数据集也只是通过自己的采集来建立了自己的数据集，并不能达到通过大数据的形式来训练神经网络的目的，所以希望能够获得更多的数据集，并且从单目摄像头改为双目摄像头来进行手势识别。
2. 仅仅是使用了 CNN 的神经网络实现了手势检测、识别，在手势识别的准确度和检测速度上仍然不太喜人，希望以后能使用多级联的网络实现任务，来提高检测精度和速度。
3. 算法上在学习手势识别的过程中学习到了许多新的算法，比如 YOLO、SSD-Gesture、Faster R-CNN，其中 YOLO 算法已经更新到 YOLOv3 版本，希望以后能够学会并使用这些算法来提升自己。
4. 希望以后能够构建出对手势检测、识别、指尖定位构建识别速度更快识别精度更高，鲁棒性更好地人机交互的系统。

参考文献

[i] HTC VIVE [Online],Available:https://www.vive.com/cn/
[ii] Microsoft Hololens [Online],Available: http://www.microsoft.com/microsoft-hololens/en-u
[iii] Lee J., Lee Y.,et al Hand region extraction and gesture recognition from video stream with complex background entropy analysis [J]. ConfProc IEEE Eng Med Biol Soc,2004, 2(2):1513-1516

致　谢

在本次论文的完成过程中，感谢赵志文老师的指导和帮助，在论文选题、研究设计和写作阶段，赵志文老师给予了指导和无私的帮助，对本人的论文提出宝贵的修改意见，在此表示最诚挚的感谢。同时也感谢整个课题组的同学，在研究课题的时候提供意见和帮助，感谢他们愿意无私的帮助。在论文的最后，祝愿赵志文老师和课题组的同学们身体健康，事业有成。