知识产权声明

纠错与交流

如果发现本文有错漏，或者有更好的想法，烦请联系我，邮箱：happyjacket@qq.com，谢谢！

基于视觉和 ResNet 的手势识别及其应用

[**摘　要**] 本文设计了一个适用于日常场景下的手势识别算法框架，把手势识别的应用流程分割为五个相对独立的模块，包括一个视频输入模块，一个结合帧差法和 kmeans 的对光照不敏感的手部候选区域发现模块，一个使用 ResNet 深度卷积神经网络的手势识别模块，一个使用基于滑动窗口的投票算法对静态手势序列去噪音、编码的动作队列模块，一个使用观察者模式实现的应用模块。本文提出的算法基于运动信息来发现手部，对光照、颜色不敏感，适合在实际的生活环境中使用。在检测时，能将手部的候选区域的数量大大地缩减，使得程序在低端配置的设备上也能达到实时的运行速度。不同于传统的识别方法，本文采用 ResNet 深度卷积神经网络，无需分割出前景或人工提取特征，即可准确识别。自行收集了一个静态手势数据集，训练得到的网络在测试集上的准确率达到 97.05%。所识别的对象不仅仅是静态手势，还有手势的运动方向、手部的位置，这样相当于把静态手势的表达能力扩大到了 5 倍，即上下左右移动以及静止。基于上述的手部检测算法，本文还提出了一个快速、可以自动标记、只需要少量人力的收集手势数据的方法。在应用方面，本文实现了两个实际的示例程序，一个音乐播放器控制器和一个 PPT 播放控制器。

[**关键词**] 计算机视觉；运动目标检测；图像识别；手势识别

Hand Gesture Recognition and Application Based on Vision and ResNet

[Abstract] This paper designs an algorithm framework for hand gesture recognition, which can be used in daily life. The process is divided into five modules, including a video input module, a hand gesture detection module combined frame difference method and kmeans algorithm, a module to recognize hand gestures, a queue that uses a sliding-window-based voting algorithm to denoise and encode the hand gesture sequence, an application module. The proposed detection algorithm is based on moving information, which means that lights can't have much effect on the detection results. The number of candidate hand regions is greatly reduced, so that the program can run real-time even with a low-end configuration laptop. As for hand gesture recognition, traditional methods need to do foreground segmentation and artificial feature extraction. However, thanks to the powerful deep convolutional neural network, this paper uses a 20 layers ResNet model to recognize hand gesture classes directly. Testing with a self-collected hand gesture dataset, the trained model achieve an accuracy of 97.05%. The proposed method can recognize not only static hand gestures, but also the moving direction and the current position of the hand. Binding with moving state, the expression ability of static gestures is extended by five times. A moving state can be moving-up, moving-down, moving-left, moving-right and no-moving. Based on the detection algorithm mentioned above, we also design a fast, automatic-marking hand gesture data collecting method, requiring merely a little effort from human beings. As for application, two helpful examples are designed, a PPT slide controller and a music player controller.

[Keywords] Computer Vision; Moving-Object Detection; Image Recognition; Hand Gesture Recognition

目录

第一章 引言

1.1 选题背景与意义

随着 AR、VR 技术的日益成熟，人们对新一代的交互技术有了更多的期盼，手势就是其中的一种。手势控制很符合人类的交互习惯，好的手势识别可以让使用者享受到“招之即来，挥之即去”的魔法体验。现在，通过 Kinect、Leap Motion 等昂贵的设备，手势识别可以达到在日常娱乐中使用的程度，只不过硬件成本较高。而基于视觉的手势识别，只需要一台普通的计算机和一个摄像头即可。

视觉的方法涉及手部在视频序列中的的定位、手势的识别。对于手部的定位问题，有两种思路，一是根据一个初始位置进行目标跟踪，一是在每一帧都重新检测目标的位置。在手部运动的场景下，由于存在大面积的人体部位颜色与手部相同，且手部的颜色特征、形态特征等变化很大，实践发现跟踪算法很容易跟丢手部。所以本文决定采用检测的方法，但是运行速度慢是目标检测的一个很大的问题。当前已经有了很好的方法，如 Faster R-CNN[1]、YOLO[2]，不过它们的实时性需要高性能计算机的支持。本文通过运动信息极大地缩减了手部的候选区域的数量，从而大大地加快了检测的速度。

而对于静态手势的识别，归属于图片识别领域，也已经有很多好的模型了，如 AlexNet[3]、GoogLeNet[4]、VGG16[5] 等。手势识别的主要难点在于人体手部的形变程度很大，角度多变，手指、手掌之间的遮盖会产生阴影从而对总体的颜色特征产生影响，手部运动的同时人体有很多其它部位也在运动，在生活场景中背景往往比较复杂且可能含有与手部颜色相近的部分，这些问题对手部的跟踪和识别都造成了很大的困扰。识别手势的传统做法一般是先将背景与前景进行分离，人工构造一些关于手势的特征，如手部轮廓的 Hu 矩、手指的个数之类的特征，然后训练一个分类器来对手势进行识别。但是由于手部的形态、角度等都是多变的，很难人工构造出一个好的特征来识别手势。近年来，随着深度学习的普及，由于深度神经网络有自编码的特性，不需要人工构造特征，且准确率很可观，有不少研究者开始使用深度学习的方法来识别手势，如 2016 年 CVPR 会议上的 Deep Hand 一文[6]。本文采用了目前在图像识别领域最先进的深度卷积神经网络模型之一的 ResNet[7]，训练得到准确率高、泛化性好的分类模型。

另一方面，关于静态手势图片的数据集，目前静态手势识别领域还没有一个公认的合理的数据集。不少研究者自行收集的数据集都是经过特殊处理的，其背景很简单，与手部颜色区别很大，这造成了有很多论文使用人体肤色模型就可以得出很好的结果。

但是收集数据又是一个很耗时耗力的任务，特别是深度学习的方法需要大量的数据。本文基于提出的检测算法，稍微修改一下就得到了一个快速收集、可以自动批量标记手势图片的数据收集方法。本文利用这个方法轻松在 9 个不同的室内背景下收集了 28000 张图片作为数据集。

1.2 论文结构

本文的正文共分为五章，第一章是引言，介绍本文的设计背景、主要难点、提出的算法的概要、主要相关工作的现状；第二章阐述解决问题的思路和算法设计的细节；第三章展示数据集的设置、模型的结构与训练过程；第四章展示结果，分析训练得到的模型的性能、算法的实时性、整个方法的可移植性、局限性；第五章总结本文的工作、反思存在的缺陷、展望未来可以继续深入的角度；最后是参考文献、致谢。

第二章 问题分析与算法设计

本文的思路是先在全局的图像中检测有运动的区域，获取可能含有手部的局部候选区域，然后对其进行识别，筛选出真正的手部区域。由于检测和识别这两个过程都不是百分之百准确的，所以需要将识别的结果放进一个队列中，对其进行噪音去除。另外，本文扩展了静态手势，将其与手部运动方向绑定起来，所以还需要对此进行识别，这个队列称之为动作队列。最后将决策结果输出给应用模块，使其响应各种手势和手部运动状态，从而作出预设的功能反应。本文将整个流程分割为五个模块，各个部分是独立的，责任单一而明确，依靠约定的接口结合在一起，流水线总结如图2-1所示：

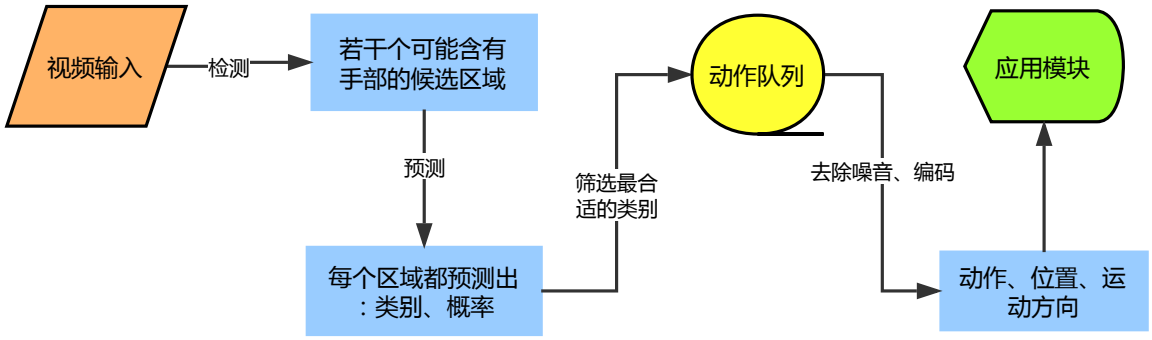


图 2-1 程序逻辑流程

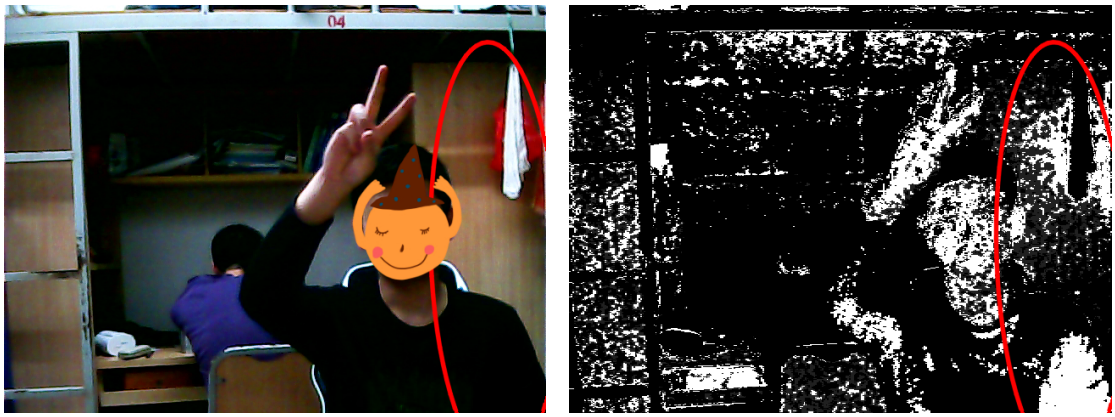
2.1 生成手部的候选区域

2.1.1 跟踪与检测的讨论

在识别手部之前需要先在全局的图片中找到手部的位置，这是一个在视频中定位运动目标的问题，有两种思路，一是从一个初始的位置出发，使用跟踪算法进行目标跟踪；二是在每一帧都重新进行目标的检测。目标跟踪算法的速度一般很快，不过易受光照、颜色、物体互相遮挡、形态变化的影响，从而容易跟丢目标。物体检测算法的速度一般很慢，较难达到实时的标准，不过定位目标的准确度一般较高。

跟踪算法按照所依据的特征，可以简单分为按照颜色特征、按照运动信息、按照通过机器学习模型学习到的特征。camshift[8] 等依赖于颜色的跟踪算法，在手势识别的场景下的效果不好，实践发现当背景的颜色跟手部比较相近时，camshift 几乎失效了，如图2-2所示。即使背景颜色与肤色差别很大，由于人脸的存在，当被跟踪的手部靠近脸部

时，camshift 也很容易把跟踪的焦点定在脸部，从而失去应该跟踪的目标。除了 camshift 这种跟踪算法之外，其它类型的跟踪算法，如 OpenCV 中提供的 MIL，BOOSTING，MEDIANFLOW，TLD，KCF，经过本人的实践验证，其中大多数算法的实时性不够好，在手部运动的场景中准确度也不够好，容易跟丢目标。



(a) 原始帧　　(b) 反向投影图：亮度代表前景概率

图 2-2 camshift 失效示例

YOLO 把 object 的 detection 问题转化为 regression 问题，使用深度神经网络拟合的效果很好。虽然该论文声称这个算法是可以做到实时的，但实测的结果是，这个算法要求计算机有很高的配置（比如 CPU 为 Intel i7，显卡为 NVIDIA GTX1080），目前不适合用来做普通的应用。

Faster R-CNN 的检测效果也很好，但实时性很大程度上依赖于设备配置，而且训练过程复杂，目前不是优先考虑的方案。

有不少论文直接通过肤色模型来提取手部，使用一个固定的肤色范围，比如论文[9][10]，其实验背景是接近纯色的，或者复杂一些但颜色与人体肤色区别很大，这些局限使其较难在日常生活中的光照环境和背景环境中应用。

2.1.2 结合运动信息做手部检测

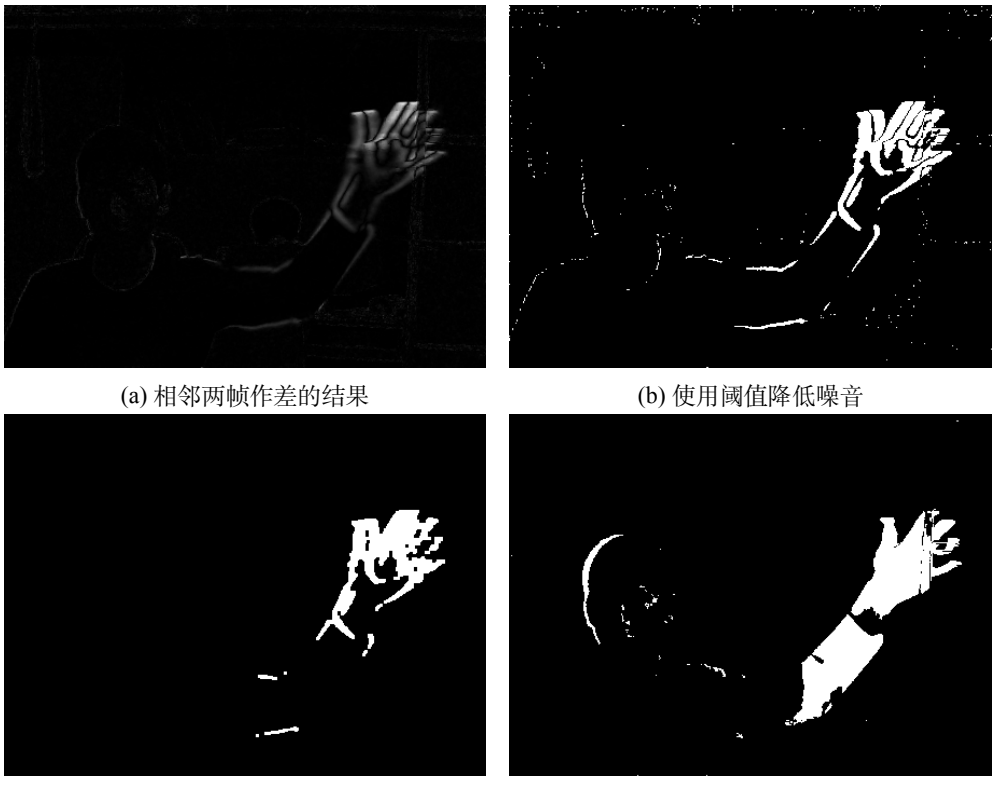
目标检测算法之所以速度慢是因为候选区域太多了，如 Faster R-CNN 一般每一帧是生成 1000-2000 个候选区域，从中进行筛选，最终合成一个个包含目标的区域。而在视频序列中，由于摄像头一般是不动的，即背景也是不动的，前景是运动的物体，可以考虑通过提取运动信息，从而大大缩小候选区域的数量，进而加快检测算法的速度。

使用运动信息可以抵抗颜色和光照的干扰，主要想法是，认为运动的部分是前景，否则为背景，将背景减去即为前景，而判断是否运动的标准是像素值变化的强烈程度。常见的方法有帧差法，高斯混合模型，平均背景法[11]。平均背景法需要在初始时用一

段时间收集静态的不含运动目标的前景，推断出背景中各个点的像素值在当前光照条件下的变化范围，以此来分辨运动部分和背景部分。高斯混合模型（Gaussian Mixture Model，GMM）建立了多个高斯概率密度函数，根据历史帧的信息，来精确量化前景、背景的分布。OpenCV 3.1.0 版本基于 GMM 实现了一个很好的背景减去子 BackgroundSubtractorMOG，此外还有 BackgroundSubtractorMOG2 以及 BackgroundSubtractorGMG，三者的效果都是很棒的，MOG 的效果示例如图2-3d。

本文为了减少处理时间，采用了效果略差但很快速的方法：帧差法。将相邻两帧灰度化之后作绝对差，就可以得出全局大致的运动信息。值得注意的是作绝对差，而不能作简单的差值，因为想要知道的是颜色变化的剧烈程度，而不是前后帧相同位置像素值的大小之分。

遗憾的是这种方法会遇到两个大的问题。第一个问题是，由于光的波粒二象性，光照不是绝对恒定的，以及摄像头的采样误差等原因，即使是没有运动的背景区域，在相邻两帧相减后也会得到很多噪点，如图2-3a。本文采用简单的 threshold 方法去除大部分背景噪音，阈值设为 24，效果如图2-3b所示。然后再用形态学运算去除多余的噪音，具体为一次开操作和一次闭操作，kernel 大小为 3*3，最终的效果如图2-3c。



(a) 相邻两帧作差的结果　(b) 使用阈值降低噪音

(c) 形态学处理后的结果　(d) MOG 的效果

图 2-3 视频的运动信息

第二个问题是，帧差法得到的结果相当于是两帧运动区域边缘的重影，运动物体的

自我重叠部分由于相减而失去，只剩下运动物体的边缘，但是运动物体的边缘又经常是不完整且模糊的，如图2-3c所示，这意味着很难直接从这些信息中直接获取特征从而识别出手部。不过它为 detection 提供了很重要的信息，即在白点密集的区域很有可能有物体在运动，因为经过处理后的白点意味着该点所在的位置的颜色变化强烈。

2.1.3 提取运动区域

原始视频数据经过2.1.2中介绍的算法处理后，得到的结果是一个单通道图像序列，每一帧包含着一些离散的白点。由于人体的运动不是单一的手部运动，不能要求整个人僵直在摄像头前，只有手掌在做动作，所以整张图片中应该是含有若干个运动区域的。要想检测出手部区域，需要先从整张图片中将所有的运动区域提取出来。

本文采用 kmeans 算法，把这些离散的白点的坐标作为 kmeans 的输入，聚成 K 类，以每一类的簇中心为中心，以一个合适的边长圈出一个足够大的矩形区域，比如 60*60，就认为这是提取到的区域。其中，本文把 K 设置为 3，因为在手势识别的场景下，一般最多有 3 个部位在显著运动，即手部、头部、肘部。这样做的效果是很不错的，确实可以把不同的区域提取出来，效果如图2-4所示。这些提取出来的有运动的区域将作为手部的候选区域，需要放进下一个模块进行识别与区分。

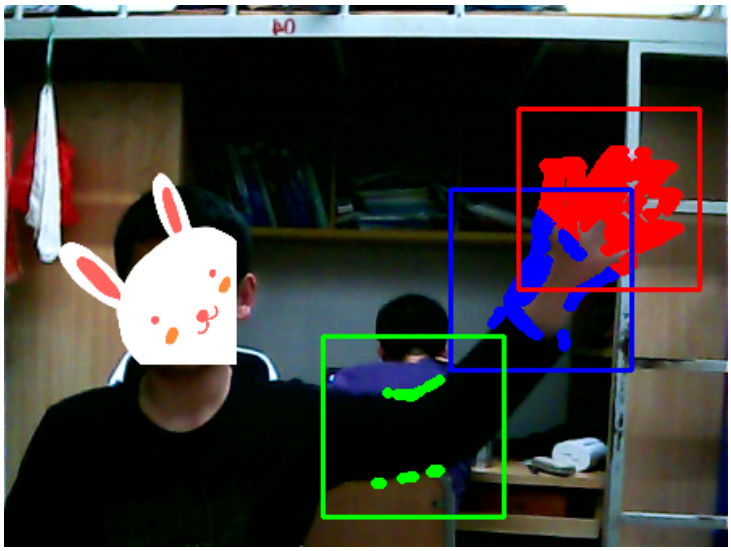


图 2-4 聚类运动区域的效果

2.2 手势识别

经过2.1.3的处理，可以获取到若干个手部的候选区域，接下来需要对各个区域进行识别，筛选出手部。这些候选区域主要有五类：

- 手部；
- 头部；
- 肘部（得考虑是否被衣服遮盖两种情况）；
- 躯干（一般是被衣服遮盖的）；
- 投射到墙上的影子。

本文将除手部外的其它类别都认为是噪声，将其作为一个大的类别，与各种手势类别一起，用一个多分类器进行识别。传统的识别方法是，先分离前景和背景，然后提取 Hu 矩、HOG、LBP 等特征，最后训练一个分类器。实时分割前景是一个很难的问题，再者，人手的自由度很大，角度多变，很难人为构造一种好的特征来识别不同的手势。基于模板匹配的方法、基于点模型的方法也是类似的困境。本文采用了简单直接的方法，不需要人为提取特征，而是基于大量的手势图片，训练一个深度卷积神经网络来进行识别。具体的数据集设置、模型结构与训练过程，请参见本文的第三章。

2.3 生成结果

由2.1.3可知，前面的步骤得到了若干个候选区域，接下来由2.2对每个区域单独进行识别，各自得出一个类别及其分类概率。最后需要解决的问题是，如何从若干个分类结果中决策出一个最终的结果。本文的做法类似于投票算法，统计每一类手势的识别概率总和，认为总和最大的一个类别是当前正在做的手势，并将这个总和的平均值作为该结果的概率。检测、识别的结果还应该包括当前手部的定位。本文对于手部位置的定义是其所在跟踪框的中心坐标，假设根据上述的方法预测出动作类别为 A，且总共有 K 个区域的类别都预测为 A，它们的位置依次为 (x_1, y_1), $(x_2, y_2), \ldots, (x_K, y_K)$，分类的概率依次为 p_1, $p_2, \ldots, p_K$，那么本文认为应该预测当前手部的位置为：

$$(x_{predict},\ y_{predict}) = (\sum_{i=1}^{K} x_i * \frac{p_i}{\sum_{j=1}^{K} p_j},\ \sum_{i=1}^{K} y_i * \frac{p_i}{\sum_{j=1}^{K} p_j}) \tag{2.1}$$

，即用分类概率加权平均各个分类正确的区域的位置。

2.4 动作队列

由于 detection 和 prediction 都不是 100% 准确的，所以2.3输出的单帧识别结果是有噪音的，比如在做手势 0 的时候，可能会输出这样的序列：0,0,2,0,0,0,5,2,0,0，因此需要一个机制来对抗这些噪音。本文的做法分为两步，第一步是设置一个阈值，比如 0.80，

先把置信度（即分类概率）低的预测结果丢弃，值得注意的是，这样的做法不能完全去除噪音。如图2-5所示，可以看出，随着阈值的增大，误判的 sample 越来越少，不过减少的幅度也在变小。当阈值超过 0.70 时，zero 这个类别预测对的样本数突然减少了很多，但预测错的样本数则没有明显的变化，所以阈值不宜超过 0.70，本文选择了 0.70 作为分类的概率阈值。

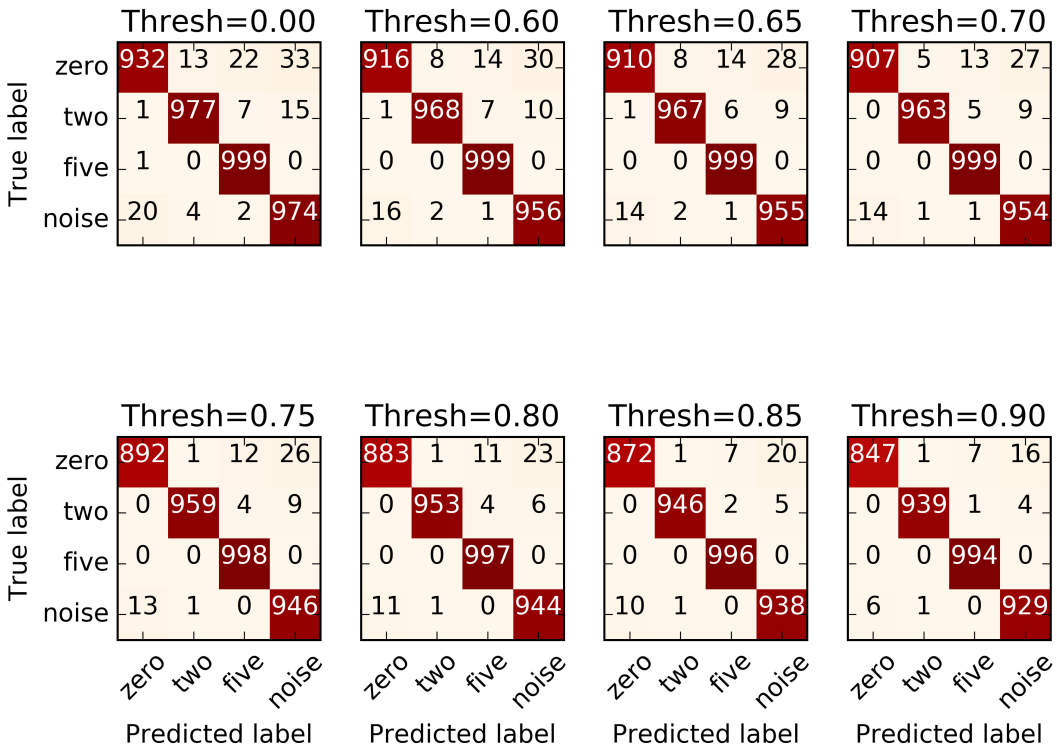


图 2-5 分类概率阈值对 confuse matrix 的影响

第二步需要从有些许噪音的序列中做决策，判断当前是哪个动作。一种简单的方法就是计数，比如连续的 10 帧都预测为 0，才做出手势 0 相对应的动作。但是这种方法要求噪音数量极少，不然就会出现很久都识别不出手势的情况。改进这种做法，本文是通过设置一个滑动窗口，宽度为 N，再设定一个阈值 T，然后对滑动窗口中的动作类别求众数，若众数的个数大于或等于阈值，则认为该众数所代表的类别是当前正在做的动作，把这个动作类别传递给应用模块，并同时清空整个滑动窗口，清空窗口主要是为了让下一个动作的分类不受前一个动作的数据的影响。对于阈值 T 的设置，本文预设 $T = \lceil (N+1) * 0.5 \rceil$，这样的数值既可以允许适当的噪音，也可以保证最多只有一个类别被选出来。例如，当 N 选择为 8 时，T 等于 5，最多允许出现 37.5% 噪声，而很明显在这个滑动窗口里不可能同时出现两个数量都大于等于 5 的不同类别。

接下来需要解决的问题是，如何基于一个滑动窗口的信息，以及一个通过众数方法预测出来的动作，来判断手的移动方向，本文的算法如算法2.1所示。

算法 2.1 预测滑动窗口中手的移动方向

$A \leftarrow$ 滑动窗口投票决定的动作类别
$T\_delta \leftarrow$ 判断是否有往某一方向移动的位置偏离阈值
$start \leftarrow$ 滑动窗口中第一个类别为 A 的元素
$end \leftarrow$ 滑动窗口中最后一个类别为 A 的元素
$X\_delta = end.x - start.x$
$Y\_delta = end.y - start.y$
$delta = max(abs(X\_delta),\ abs(Y\_delta))$
if $delta >= T\_delta$ **then**
    if $delta = abs(X\_delta)$ **then**
        if $X\_delta < 0$ **then**
            return MOVING_LEFT
        else:
            return MOVING_RIGHT
        end if
    else:
        if $Y\_delta < 0$ **then**
            return MOVING_UP
        else:
            return MOVING_DOWN
        end if
    end if
else
    return MOVING_NONE
end if

2.5 应用设计

应用模块实质上是一个响应动作队列产生的动作的程序而已，比如当手势 five 向左移动时就执行某个功能。这一部分使用了设计模式中的观察者模式来实现，因为如果不使用事件驱动的模型来编程，那么应用模块必须轮询动作队列是否有新的动作产生，这是很没必要的 CPU 资源浪费。具体的编程方法是在动作队列类里设立一个回调函数表（即一个二维函数指针数组，行代表动作种类，列代表动作的移动类型如上下左右、不动或随机移动），然后应用模块在初始时向动作队列模块注册回调函数，最后每当动作队列有动作发生时，比如手势 0 向左移动，那么就会根据回调函数表定位到相应的回调函数，直接调用就可以实现控制了。

作为示例，本文实现了一个控制 PPT 的程序，它的功能是控制 PPT 左右翻页、进入/退出全屏、鼠标随人手的移动而移动，具体设置如表2.1所示，控制的原理是模拟系

统键盘和鼠标消息事件。在 Linux 系统下，一切都是文件，包括键盘、鼠标这些设备，所以模拟这些事件，可以直接读写相应的设备文件，不过自己写简单的程序去操作这些设备文件的效率是很低的，所以选择了开源的 pyautogui 库来实现，而且这个库还是跨平台的。

表 2.1 手势动作控制 PPT 播放的含义

类别	向左移动	向右移动	向上移动	向下移动	随机移动
zero	-	-	-	-	-
two	-	-	-	-	是
five	向左翻页	向右翻页	进入全屏	退出全屏	-
noise	-	-	-	-	-

其实这一部分可以实现任意控制程序，只需要满足接口规范即可，如图2-6所示是一个网易云音乐播放器的控制类。

```python
# -*- coding: utf-8 -*-
import pyautogui

class MusicPlayController:
    """控制网易云播放器，需要根据自己设置的快捷键来写控制代码"""
    def __init__(self, image_size):
        pyautogui.FAILSAFE = False  # 关闭pyautogui的安全模式

    def last_song(self):
        """播放上一首歌曲"""
        pyautogui.hotkey('ctrl', 'shift', 'left')

    def next_song(self):
        """播放下一首歌曲"""
        pyautogui.hotkey('ctrl', 'shift', 'right')

    def toggle_mini(self):
        """切换到mini模式/从mini模式恢复"""
        pyautogui.hotkey('ctrl', 'shift', 'N')

    def toggle_song_word(self):
        """显示/关闭歌词"""
        pyautogui.hotkey('ctrl', 'shift', 'D')

    def like_song(self):
        """给当前正在播放的歌曲加红心"""
        pyautogui.hotkey('ctrl', 'shift', 'l')

```

图 2-6 网易云音乐播放控制器

还有一点值得注意的是，由2.3预测出来的手部位置只是手部在摄像头视野中的位置，要想实现人机交互，还应该将这个位置映射到机器屏幕上相对应的位置。本文的做法是，将图片的坐标按比例映射到计算机屏幕的坐标。设摄像头拍摄到的图片大小为

$width_{camera} * height_{camera}$，计算机的屏幕大小是 $width_{computer} * height_{computer}$，当前预测在摄像头视野中的手部位置为 (x, y)，那么预测当前手部在屏幕上的位置为：

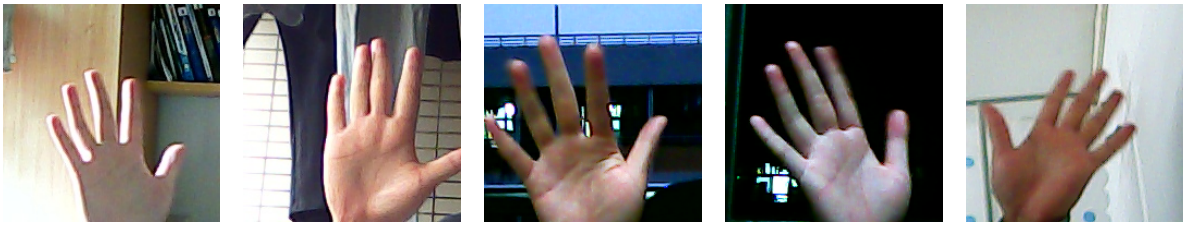
$$(x',\ y') = (x * \frac{width_{computer}}{width_{camera}},\ y * \frac{height_{computer}}{height_{camera}}) \tag{2.2}$$

第三章 数据与模型

3.1 数据集的设置

本文预设的动作有数字 0、2、5，以及噪声类，总共 4 个类别，不区分左右手。噪声的种类主要收集了肘部、手臂、头部、躯干，如图3-2所示。训练集和测试集的环境分别如图3-1所示，具体设置是，在五个室内环境中收集训练数据，各 1000 张，总共是 $1000*4*5=20000$ 张；在两个室内环境中收集验证集数据，各个环境下每个类别 500 张，总共是 $500*2*4=4000$ 张；在两个室内环境中收集测试集数据，各个环境下每个类别 500 张，总共是 $500*2*4=4000$ 张。从图3-1可以看出，训练集、验证集、测试集三者所处的环境是没有交集的，这样才能保证最终结果的泛化性好。

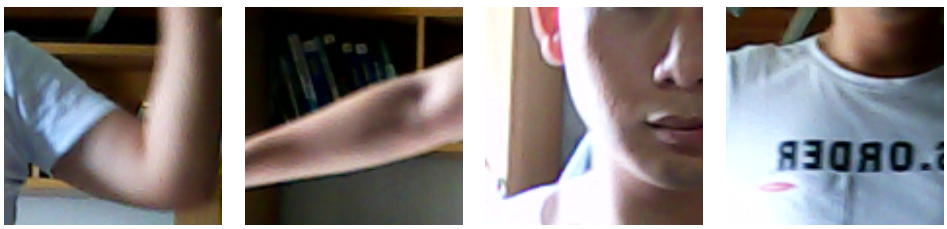
本文还采用了数据预处理和数据增广处理，前者将图片的像素值归一化，后者把图片随机水平翻转、旋转随机的角度（正负不超过 25 度）。



(a) train env1 (b) train env2 (c) train env3 (d) train env4 (e) train env5

(f) valid env1 (g) valid env2 (h) test env1 (i) test env2

图 3-1 数据集示例



(a) 肘部 (b) 手臂 (c) 头部 (d) 躯干

图 3-2 噪声数据示例

3.2 分类模型

本文采用的深度卷积神经网络模型是 ResNet 论文里在 cifar10[12] 数据集上的实验方案，输入层是分辨率为 32*32 的小图片，第二层是一个使用 3*3 卷积核、filter 数量为 16 的卷积层做初步采样处理，紧接着是 3 个 residual block，最后做一次 batch normalization，通过 ReLu 激活层进入一个激励函数为 softmax 的全连接层输出结果。把每个 residual block 参数化，方便在下面图示出来，三个参数分别为：L，F 和 input size。其中，F 表示一个 residual block 内部 layer 数量的基数（layer 数量为 $2*L$）；F 表示每个 layer 的 filter 数量（同个 residual block 内的所有层的 filter 数量是一样的）；input size 表示 layer 的输入的大小。本文选择 L=3，所以整个网络总共有 $(2*L)*3+2=20$ 层。每个 residual block 的 input size 值随其上一个 residual block 最后一层在输出时的 downsample 操作减半，依次为 32、16、8；而 filter 数量则逐个倍增，依次为 16、32、64，整个网络的结构如图3-3所示。

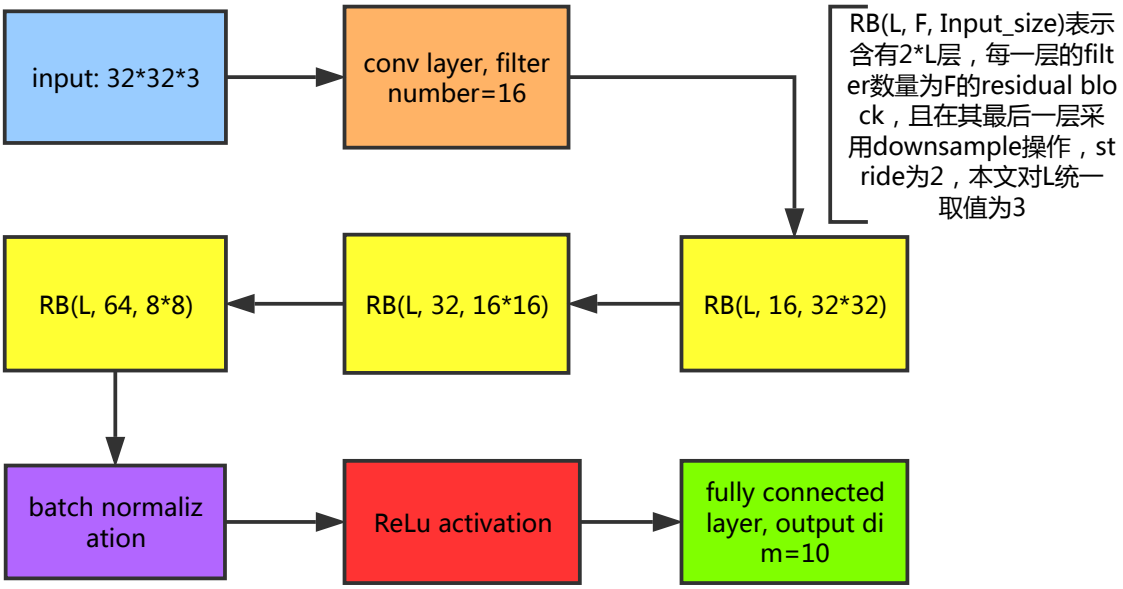


图 3-3 深度卷积神经网络模型

3.3 训练过程

本文采用 SGD（Stochastic gradient descent）方法进行训练，batch size 为 128，损失函数选择为 cross entropy。许多实践表明，learning_rate 随着训练的进行而逐渐减小可以抑制震荡，加快模型的收敛。本文动态调整 learning_rate，其调整方法如式（3.1），设置 $initial\_lr=0.1$，$decay\_rate=0.1$，$decay\_steps=32000$；至于网络权重的更新，设

置 weight_decay 为 0.0001，动量为 0.9。以上这些设置与 ResNet 论文不一样的一点是，从式（3.1）可以看出学习率是逐步下降的，而原文的设置是分别在第 32000、48000 次迭代时除以 10，最后在第 64000 次迭代时停止，原文的这些设置是由模型在验证集上的表现决定的，但本文由于计算资源有限，暂时没法深入探索这么多次迭代。

$$decayed\_lr = initial\_lr * decay\_rate^{(global\_steps/decay\_steps)} \tag{3.1}$$

3.4 快捷收集数据的方法

从2.1.3中可以得知，仅仅依靠帧差法和 kmeans，就可以得到若干个运动区域。本人认为，要想让程序自动标记手势数据，可以固定自己身体的其它部位，只控制手部运动做手势，再设置 kmeans 的簇数量为 1，这样子截取到的区域就是手部了，每次只收集同一个类别的手势，用程序统一批量命名，通过图片文件名来区分不同种类、不同背景环境，这样子相当于自动标记了。后期只需要人为去除掉少量模糊的、非手部的图片即可。而这样的操作，相对于收集一个视频序列，然后逐帧人为标记出一个矩形区域以及类别，应该算是很简单快速的。

同理，收集噪声数据其实没必要刻意分类逐帧去收集、标记。使用上述的方法，不过需要将 kmeans 的簇数量设置正常一些，比如 3，然后在摄像头前面自由运动，这样子将会捕捉到各种各样的部位，将其中与手掌有交集的图片移除即可得到噪声图片。

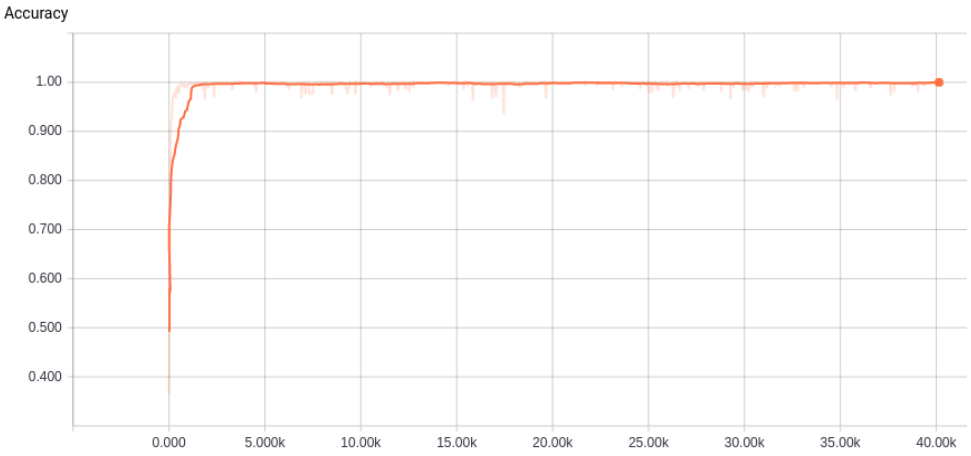
第四章 结果与分析

4.1 模型准确率

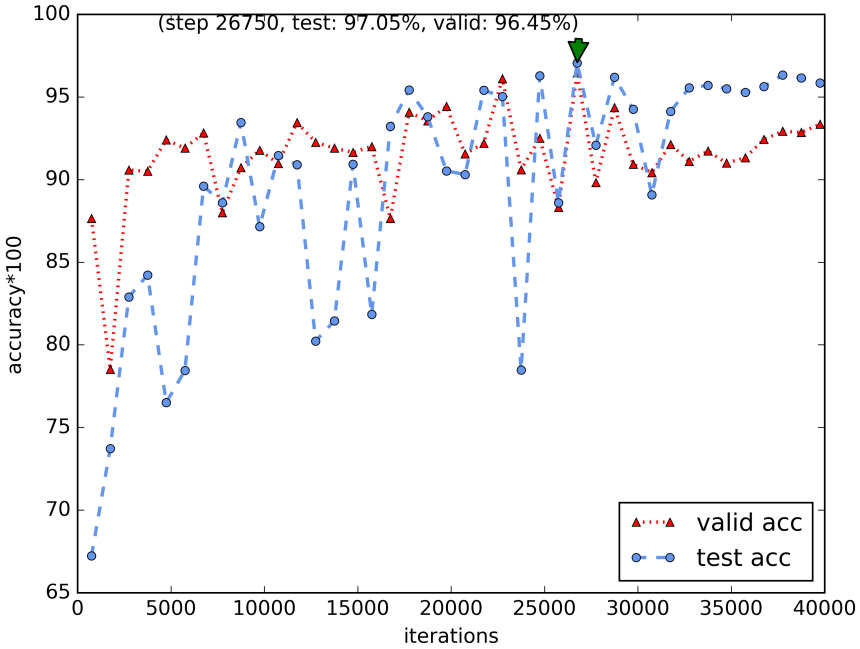
本文使用3.2所述的模型，在 training step 到达 26750 时性能最好，在训练集上的准确率达到 100.00%，而在验证集、测试集上的准确率分别为 96.45、97.05%，训练过程如图4-1a（图中的虚线是实际的变化曲线，实线是 tensorflow 的可视化工具 tensorboard 平滑化后的曲线，横轴是训练的 step 数，纵轴代表准确率）。从图4-1b可以看出，虽然模型的准确率随着训练的迭代进行有较大的波动，不过总体上波动程度是越来越小的，而且模型在验证集、测试集上的准确率也越来越接近，这说明模型确实是有在收敛的，而当前的结果算是轻微的 overfiting。由机器学习的常识知道，通过向训练集增加与已有 sample 不重复的 sample，可以让其 overfiting 程度减轻，使得模型在测试集上的准确率逼近在训练集上的表现。所以如果需要进一步增强模型的性能，可以考虑加大数据集的规模，一方面增加手势的类别，另一方面增加训练集里背景环境的种类。

分析如图2-5所示的 confuse matrix 可以发现，当前的模型很容易分辨出 five、two 和 noise 这三个类别，最差的是 zero，一半以上的误判都归咎于 zero 这个类别，后期可以适当增加一些 zero 类别的 sample，降低其 overfiting 的程度。

在测试集上的优越，并不代表在实际使用的过程中能达到相同的效果，因为摄像头的采样速率与人体运动的速率不一致，很容易对焦不准，如图4-2所示，拍到的是一片模糊的图片，此时的分类结果是不稳定的，这样的图像对于人类视觉系统来说也是很大的挑战。再者，在陌生的环境中，虽然从上述结果可以看出，这个模型的泛化性是比较好的，不过也不能保证在任意情况下都如此。



(a) 训练集的准确率



(b) 验证集、测试集的准确率

图 4-1 训练过程的变化

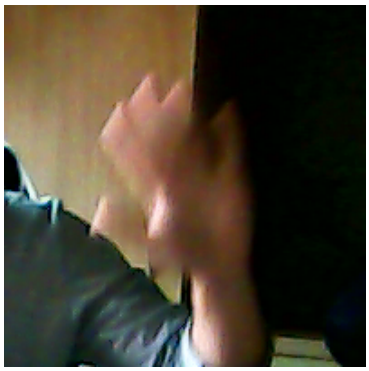


图 4-2 摄像头实时对焦的问题

4.2 实时性

本文使用的编程环境是，操作系统为 Ubuntu 16.04，编程语言为 Python 2.7，笔记本的 CPU 为 Intel 最低端的 CPU 系列赛扬 4，其主频为 1.6GHz，显卡是 Nvidia GeForce 720M，只能做纯 CPU 计算（Compute Capability 仅为 2.1，达不到 tensorflow 1.0 对 GPU 的最低标准，即 3.0）。经过 1820 帧的连续测试，除开摄像头读取图片的时间，运行时间是 122 秒，其中用来预测的时间是 111.5 秒。除开预测部分，平均处理时间约为 5.77ms，173 fps 左右，这是跟踪模块、动作队列、应用模块总的处理速度。也就是说，性能的瓶颈在于预测模块，导致总体的帧率只达到 14.92 fps 左右。本人认为，对于一个放在日常环境下使用的算法，用最好或最坏的硬件水平来测试其性能，都是不公平的。好的方面是，在这样的硬件环境里运行，意味着得到一个接近下限的结果。

4.3 可移植性

本文的算法使用 Python 语言实现，这个编程语言是跨平台的，使用到的其它模块也是跨平台的。不过本文的方法含有几个经验参数，这使得在另外的机器上使用时，可能需要重新调试出新的参数值。比如，滑动窗口的大小跟算法的运行速度是有关系的，假设使用者做动作、对程序的控制效果的反应速率固定，那么如果计算机运算速率越快的话，滑动窗口应该越长，不然会出现人跟不上计算机从而“失控”的情况。

4.4 局限性

经过认真反思，本文至少有以下四个局限：

- 假设了摄像头固定的条件，所以本文的方法只适用于摄像头不动的场景；
- 手部的定位是根据预测的结果加权平均得到的，不够精确，不适合做精细的控制操作，如绘画等；
- 跟踪框的大小是固定的，无法像 camshift 等方法一样自动调节大小；
- 只能跟踪单只手，虽然可以做到不必区分左右手，但不支持双手、多人同时进行操控。

第五章 总结与展望

5.1 工作总结

手势识别及其应用在算法层面包含两部分的工作，第一是在视频序列中的每一帧对手部的定位，第二是对手部姿态的识别。关于第一步，跟踪的方法在人体手部运动的场景下很容易跟丢目标，本文使用了检测的方法，并且利用了运动信息，大大缩减了检测手部的候选区域的数量。目前主流的方法如 Faster R-CNN 对于每一帧一般生成 1000-2000 个候选区域，而本文将其缩减为 K 个，这个 K 值一般是很小的，视运动场景的复杂程度而定。本文在室内环境里实践，发现 K 选取为 3 已经足够准确捕捉到手部了，候选区域的数量一下子缩小了三个数量级，这样可以很大程度加快检测手部的速度。

关于第二步的识别，本文采用了 ResNet 深度卷积神经网络来识别手势，同时收集了一个在多种背景环境下的手势数据集。使用神经网络模型，可以避开人为提取特征的困难，让模型可以较好地适应不同光照条件、不同的背景，这使得基于视觉的手势识别走进生活环境成为可能。

除此之外，本文基于提出的检测算法，还提出一个快速收集手势数据的方法，这可以较为轻松地满足深度学习所需的数据量。同时也设计了一个简单的去除噪音的滑动窗口机制，一个判别手部运动方向的算法，将静态手势的表达能力扩大到五倍。在应用方面，本文将整个流程分割为五个相对独立的模块，模块之间的职责明确而单一，还设计了两个实际的应用示例。不过本文的做法也有其局限性，具体请参见4.4的讨论。

5.2 研究展望

未来可以考虑扩展所支持的手势为动态手势，因为静态手势的表达能力很有限，交互也不够自然，即使加上五个运动状态也是如此。本文的做法可以对手部进行定位，扩展到识别动态手势的时候可以利用这一部分的算法，直接专注于识别部分做研究。

参考文献:

[1] REN S, HE K, GIRSHICK R, et al. Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks[J][J]. IEEE Transactions on Pattern Analysis and Machine Intelligence, 2016, PP(99) : 1 – 1.

[2] REDMON J, DIVVALA S, GIRSHICK R, et al. You Only Look Once: Unified, Real-Time Object Detection[J][J]. 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2016, 00 : 779 – 788.

[3] KRIZHEVSKY A, SUTSKEVER I, HINTON G E. ImageNet Classification with Deep Convolutional Neural Networks[C] // NIPS'12 : Proceedings of the 25th International Conference on Neural Information Processing Systems[M]. USA : Curran Associates Inc., 2012 : 1097 – 1105.

[4] SZEGEDY C, LIU W, JIA Y, et al. Going deeper with convolutions[J][J]. 2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2015, 00 : 1 – 9.

[5] SIMONYAN K, ZISSERMAN A. Very Deep Convolutional Networks for Large-Scale Image Recognition[J][J]. CoRR, 2014, abs/1409.1556.

[6] SINHA A, CHOI C, RAMANI K. DeepHand: Robust Hand Pose Estimation by Completing a Matrix Imputed with Deep Features[C] // 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR). 2016 : 4150 – 4158.

[7] Xiangyu Zhang KAIMING HE S R, SUN J. Deep Residual Learning for Image Recognition[J][J]. 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2016, 00 : 770 – 778.

[8] BRADSKI G R. Computer Vision Face Tracking For Use in a Perceptual User Interface[J][J]. j-INTEL-TECH-J, 1998(Q2) : 15.

[9] 张军; 张孔; 杨正瓴. 基于计算机视觉的多特征手势识别 [J][J]. 计算机应用与软件, 2016, 第 33 卷第 6 期 : 151 – 154+189.

[10] 刘宇航; 顾营迎; 高瞻宇; 徐振邦; 刘宏伟; 吴清文. 基于实时手势识别与跟踪的人机交互实现 [J][J]. 科学技术与工程, 2016, 第 16 卷第 24 期 : 71 – 78+92.

[11] YI Z, LIANGZHONG F. Moving object detection based on running average background and temporal difference[A][C] // 2010 IEEE International Conference on Intelligent Systems and Knowledge Engineering[C]. 2010 : 270 – 272.

[12] KRIZHEVSKY A. Learning multiple layers of features from tiny images[R][R]. 2009.

致谢

感谢我的父母这么多年来对我的关爱，以及从小培养我乐于思考的习惯，我才有机会进入中山大学计算机系学习，最后设计这样一个让自己很兴奋的程序。

感谢我的导师郑伟诗老师，老师在百忙之中抽出了许多时间与我面谈，帮我解决遇到的困难，同时在思路上给我很多启发。在我想把整个设计的各个方面都思考明确才动手实现的时候，老师一语惊醒梦中人，告诫我应该先把整条路走通，快速做一个最简单的版本出来，然后再来把每一个模块优化好，于是我把整个设计的逻辑拆分成五个独立的模块，才有了后面对每个模块逐个击破。很难想象，如果没有老师的提醒，我会陷入感觉各个方面都很难，想一齐解决却又无从下手，花费了大量的时间，最后却什么都没做成的困境。

感谢我的好朋友许泳哲和李万华，在此之前，我对计算机视觉是一窍不通的，在与他们的交流中，我的视野开阔了很多。泳哲近期在做一个人脸图像处理的创业项目，我经常向他请教计算机视觉的知识和算法，受益良多。万华目前是在研究 hand pose 的识别，我们的方向有很多相似的地方，我在思路阻塞时经常向他请教。让我印象最深的一次是，当我苦恼如何区分头部和手部的运动时，本来我已经打算用 HOG 特征和 SVM 训练一个头部的分类器了，他建议我直接把头部当做一种特殊的手势，放进深度卷积神经网络中进行分类。在此之前，我下意识地认为头部和数字 0 的手势很相似，只是 scale 不同而已，两者应该很难用同一个低分辨率模型分辨出来，就忽略了。最终深度卷积神经网络的结果让我惊喜不已，这个想法最后发展成了多分类器的做法。如果没有万华的建议，我会走更多的弯路。

最后还要感谢 OpenCV、tensorflow、tflearn、numpy 等等相关开源软件的贡献者，他们开发出来的这些软件很优秀，大大方便了计算机视觉处理、机器学习的实现，把人们的精力从基本操作中解放出来，可以更专注于想法的实现。