浙江工业大学

本科毕业设计论文

题目：基于深度学习的员工行为分析系统

作者姓名 杨晨铭

指导教师 赵云波教授

专业班级 自动化 1501

学　　院 信息工程学院

提交日期 2019 年 6 月 10 日

Dissertation Submitted to Zhejiang University of Technology for the Degree of Bachelor

Deep learning-based employee behavior analysis system

Student: Chenming Yang

Advisor: Professor Yunbo Zhao

College of Information Engineering

Zhejiang University of Technology

June 2019

浙江工业大学

本科生毕业设计(论文、创作)诚信承诺书

本人慎重承诺和声明：

1. 本人在毕业设计（论文、创作）撰写过程中，严格遵守学校有关规定，恪守学术规范，所呈交的毕业设计（论文、创作）是在指导教师指导下独立完成的；

2. 毕业设计（论文、创作）中无抄袭、剽窃或不正当引用他人学术观点、思想和学术成果，无虚构、篡改试验结果、统计资料、伪造数据和运算程序等情况；

3. 若有违反学术纪律的行为，本人愿意承担一切责任，并接受学校按有关规定给予的处理。

学生（签名）：

年　　月　　日

浙 江 工 业 大 学

本科生毕业设计（论文、创作）任务书

专 业 自动化 班 级 自动化 1501 学生姓名/学号 杨晨铭/201503080126

一、设计（论文、创作）题目：

基于深度学习的员工行为分析系统

二、主要任务与目标：

1. 阅读相关文献，了解本领域研究现状；

2. 实现行人检测、分析，学会搭建卷积神经网络；

3. 编写美观的人机界面进行展示；

4. 撰写毕业论文。

三、主要内容与基本要求：

工厂或店铺中存在员工监管困难的问题，员工的不当操作会造成经济损失甚至引起安全事故。随着计算机视觉的发展，以及监控摄像头的广泛布置，利用深度学习算法对监控画面中的员工进行检测、分析，在特定行为发生时进行记录和提醒，减少监管人员的精力消耗，提升企业自动化管理的水平。

本课题旨在研究人体行为识别方法，利用深度学习算法提升企业自动化管理水平。

四、计划进度：

- 2019 开学前 收集相关资料文献，学习相关知识，完成外文翻译、文献综述；熟悉课题，做好开题准备 - 第 1-3 周 完成开题报告，参加开题交流 - 第 4-8 周 完成行为识别算法，接受中期检查 - 第 9-14 周 完成可展示界面，撰写毕业论文 - 第 15 周 修改毕业论文，参加毕业答辩，提交相关文档资料

五、主要参考文献：

[1] LeCun Y, Bengio Y, Hinton G, “deep learning review,” nature, 2015, 521(7553): 436.. [2] Redmon J, Divvala S, Girshick R, et al, “You Only Look Once: Unified, Real-Time Object Detection,”Proceedings of the IEEE conference on computer vision and pattern recognition. 2016: 779-788. [3] Zheng L, Yang Y, Hauptmann A G，“Person Re-identification: Past, Present and Future,” arXiv:1610.02984, 2016.

任务书下发日期 2018 年 12 月 25 日
设计（论文、创作）工作自 2018 年 12 月 25 日 至 2019 年 6 月 4 日

设计（论文、创作）指导教师 赵云波

系主任（专业负责人） _____

主管院长 _____

基于深度学习的员工行为分析系统

摘　要

员工行为分析是利用摄像头拍摄的监控视频，对员工行为进行检测分析，着重于发现员工偷懒的行为和动作。近些年随着计算机视觉的发展，以及监控摄像头的广泛布置，利用监控视频对员工进行行为分析是一种很好的查勤考评手段。但是传统的人工分析工作量大、时间覆盖率低、监控的难度大，很难长久的坚持下去。

而随着深度学习的日益成熟，利用深度学习算法对监控画面中的员工进行检测、分析，在特定行为发生时进行记录和提醒，可以减少监管人员的精力消耗，降低管理的成本，提升企业和商店的自动化管理的水平。本文对视频中特定人员（员工）的行为进行分析，实现了检测员工的特定行为的功能。论文的主要工作如下：

1、简述课题研究背景和研究现状，根据实际情况提出需求，确定了选择 yolo(you only look once)网络框架进行行人检测、mgn(multiple granularities network)网络框架进行行人重识别、残差网络进行动作识别的方案。

2、搭建了神经网络并实现了在测试集上 95%的正确率。

3、编写了视频检测程序，实现了在视频中检测并保存特定人员玩手机动作的功能。

本文预期目标为在检测程序中实现对监控视频的检测，标记偷懒员工并保存偷懒画面。通过毕设期间的努力达成了该目标，但仍有不足之处：网络对同屏小范围多人的情况检测精度下降；对穿着和身材接近店员的行人有误判的情况发生。这些都应该是在未来进行改善的点。

关键词： 深度学习，卷积网络，行人检测，行人重识别，动作识别

Deep learning-based employee behavior analysis system

ABSTRACT

Employee behavior analysis is the use of surveillance video captured by the camera to detect and analyze employee behavior, focusing on the behavior and actions of employees lazy. In recent years, with the development of computer vision and the extensive arrangement of surveillance cameras, the use of surveillance video to analyze employee behavior is a good means of checking and checking. However, the traditional manual analysis has a large workload, low time coverage, and difficulty in monitoring. It is difficult to persist for a long time.

With the deep maturity of deep learning, the use of deep learning algorithms to detect and analyze employees in the monitoring screen, and record and remind when certain behaviors occur, can reduce the energy consumption of supervisors, reduce the cost of management, and enhance the enterprise and the level of automated management of the store. This article analyzes the behavior of specific people (employees) in the video and implements the ability to detect specific behaviors of employees. The main work of the thesis is as follows:

1. Briefly describe the research background and research status of the subject, and put forward the requirements according to the actual situation. The scheme of selecting the yolo(you only look once) network framework for pedestrian detection, the mgn(multiple granularities network) network framework for pedestrian recognition, and the residual network for motion recognition is determined.

2. Built neural networks and achieved 95% corrects rate on the test set.

3. A video detection program has been written to realize the function of detecting and saving the movement of a specific person in the video.

The goal of this paper is to achieve detection of surveillance video in the detection process, mark lazy employees and save lazy pictures. Through the efforts during the establishment period, this goal was achieved, but there are still some shortcomings: the detection accuracy of the network for a small number of people with the same screen is reduced; the misjudgment of pedestrians who are close to the store staff and the body is wrong. These should be the point of improvement in the future..

Key Words: deep learning, convolutional network, pedestrian detection, pedestrian

recognition, motion recognition

目　　录

第1章 绪 论

1.1 课题研究背景及意义

在工厂中，如果有员工偷懒，势必会对工厂效率有所影响；而在商店中，店主聘用的员工偷懒不光会影响店铺效益，同时还会给过往的顾客留下不好的印象，这一点在餐饮店中尤为严重，店员的偷懒将会让顾客的体验大打折扣。同时员工因偷懒分散了注意力也有可能导致店内物品丢失的情况发生。如果能够利用店铺中布置的摄像头进行员工行为检测，并将结果与员工的薪资挂钩，那么员工偷懒的情况将会大大减少。

目前在工厂和商店中安装自动化视频监控设备已经是一种普遍的行为，但如今很多监控设备大多只是起监视的作用，利用监控设备存储的信息进行控制是有难度的。传统方法是通过人工观看来进行检测，费时费力，而且准确率和效率不高，因此就需要一种自动化检测的方法。随着深度学习的发展，深度学习在视频或图片检测上的应用越来越普遍[13]，通过深度学习的方法对视频内容进行检测，其检测的精度越来越高，速度也越来越快，因此选择深度学习来检测监控视频，实现员工行为分析，替代人工抽查检测。可以有效提高检测效率，增大检测覆盖率。

随着该方法的逐渐成熟，将进一步增强管理层对基层员工的监管力度，让监管成为一个非随机的、长时效的行为，对提升企业或者店铺的运行效率，提高员工对顾客的服务质量有着积极性的作用。同时还可以减少人工检测的依赖性，减少人员因素对检测结果的影响。

1.2 国内外研究现状综述

在论文[1]中，利用手机传感器传回的数据进行人体行为识别，对传感器返回的数据进行处理后送入深度学习网络进行学习，模型的准确率有很大提升。使用手机传感器传回的数据清晰，且记录行为的总体时长长，便于网络的学习分析。但是缺点也很明显，数据的采集离不开手机传感器，一旦测试人员不适用手机进行行动，就无法检测出行为；结合课题背景，为员工配置传感器监控他们的行为是过于理想化的想法，员工可能会不配合检测，检测到的数据往往也不够真实。

论文[3]中，通过课堂监控视频，运用差分法提取目标，然后使用 vgg-16 网络进行检测，取得了一定的准确率，但是差分法提取目标速度慢，检测精度不高，懂事 vgg-16 网络的检测准确率也不高，但是检测思想值得学习。

论文[2], [4]中，使用监控摄像头对行人进行异常行为检测，通过将监控视频分成近景和远景视频两部分，分别针对动作和整体运动两部分进行识别检测，并最终实现异常行为识别。方法非常成熟，但是只是专注进行行为识别，并没有做到对行人身份的识别；而本课题要求的是对员工的行为识别，不仅仅是要进行行为识别，识别出员工同样是重要的步骤。

在参考了国内外相关文献后，确定了本课题需要通过视频信息进行员工行为分析：摄像头采集监控视频相比用传感器等进行信息采集要更加方便，成本更低；同时目前的卷积网络对视频信息的检测精度非常高，而且检测速度也能够满足实际使用。因此选择把监控视频作为信息载体进行员工行为分析。

同时，通过对中外文献的阅读，发现有关行为识别的文献中几乎没有对行人进行分类检测的过程，而本课题需求对行人进行分类，只有对符合要求的行人才会识别他们的动作。因此，在对视频中行人进行行人检测和动作识别操作的过程中，应加入一个行人重识别的流程，用来从行人中筛选出员工，防止对无关人员进行检测，影响最终的功能。

1.3 主要研究内容

本文主要是在视频中使用深度学习的方法对员工的行为进行分析，目的是找出视频中员工玩手机的偷懒的行为。本课题研究的成果用来提升工厂或者商店利用监控设备考勤员工的精度，降低监控考勤的成本，提升自动化管理水平。本文主要结构如下：

第 1 章是绪论部分，主要介绍了课题的研究背景和意义，国内外研究方向和研究现状的综述，以及本文的主要研究内容。

第 2 章介绍并比较了目前普遍使用的网络的性能优劣，根据课题需求选取网络。

第 3 章介绍了网络模型的结构搭建及、算法原理，并介绍了网络训练时的参数设置和训练结果。

第 4 章介绍了功能实现框架结构，总结了功能实现的流程细节，给出了最终效果。

第 5 章对全文进行了总结，总结了主要工作贡献以及不足之处，并且对未来进行了一定的展望。

1.4 本章小结

本章首先介绍了课题的研究背景和意义，之后对于行为识别的国内外研究方向和研究现状进行了综述，提出来本课题的研究思路，最后对本文的主要研究内容进行概括。

第 2 章　需求描述及网络的介绍与选取

2.1　需求描述

本毕设目标是能通过监控视频来进行员工的行为分析，检测出员工玩手机的画面。然而监控视频往往像素不高，而且监控视频时间较长，这就需要网络处理的速度要快，且网络本身素质足够好，不然在实际运用中无法正确检测出员工的行为。

要对视频中的员工进行行为分析，需要通过行人检测、行人重识别、动作检测三个主要步骤。每一个步骤都需要选择合适的网络来实现步骤的功能：行人检测需要网络能够将视频中的行人检测出来，并正确定位；行人重识别需要能够根据预先条件（预设的员工图像信息）来将员工从检测出的行人中分别开来，如果网络本身准确性不高，遇到摄像头的低画质图片将会有更低的准确率；动作识别需要利用高精度的卷积网络检测行人图像，分析其动作是否是预设动作，这就需要使用一个优秀的卷积网络。通过分析，产生了以下三个主要需求：

1、要准确且有较快速度对视频中行人进行定位，需要选择什么网络？

2、因为监控视频图像不够清晰，利用重识别进行店员与行人的分别是有难度的，需要一个够精确的重识别网络。

3、动作识别是针对图片中行人动作进行识别的，因此需要一个性能优秀的、足够深的且比较成熟的卷积网络来当作动作识别的网络进行训练。

以下三个小结将针对这三个需求进行网络选择。

2.2　行人检测网络选取

要分辨出视频中的人，就要对视频进行行人检测。行人检测就是利用计算机视觉来判断图像或者视频中是否存在行人，并对可能的行人进行定位。行人检测是实现本毕设课题功能的第一步，只有先在图片中检测出行人，才能对行人进行区分，并进行后续的操作。通过神经网络的检测，会对视频中的每个行人圈定一个范围，也就是 anchor box，通过行人检测，得到行人的坐标。得到了行人的坐标之后，可以截取相应坐标的行人图像当作之后网络的输入，因此要选择一个效率高，准确性优秀的网络架构是十分重要的。

对此，根据 Redmon, Joseph, Divvala, Santosh, Girshick, Ross 等(2015) [9]文中提出的

网络模型，抛弃了传统的分类器进行目标检测，而是使用直接输入整张图片，直接在输出端回归出边界框(bbox)的位置和所属的类别，该网络就是 yolo(you only look once)网络。

Yolo 网络通过将图画分成许多个网格，每个网络预测多个 bbox，每个网格返回自身负责的所有 bbox 的预测中心坐标、bbox 大小和 bbox 类别置信度，通过损失函数比较选出最优的 bbox 作为物体的检测框。Yolo 网络的速度非常快，而且精度也不错。但对于同屏多目标检测还是有些乏力，精度下降明显。

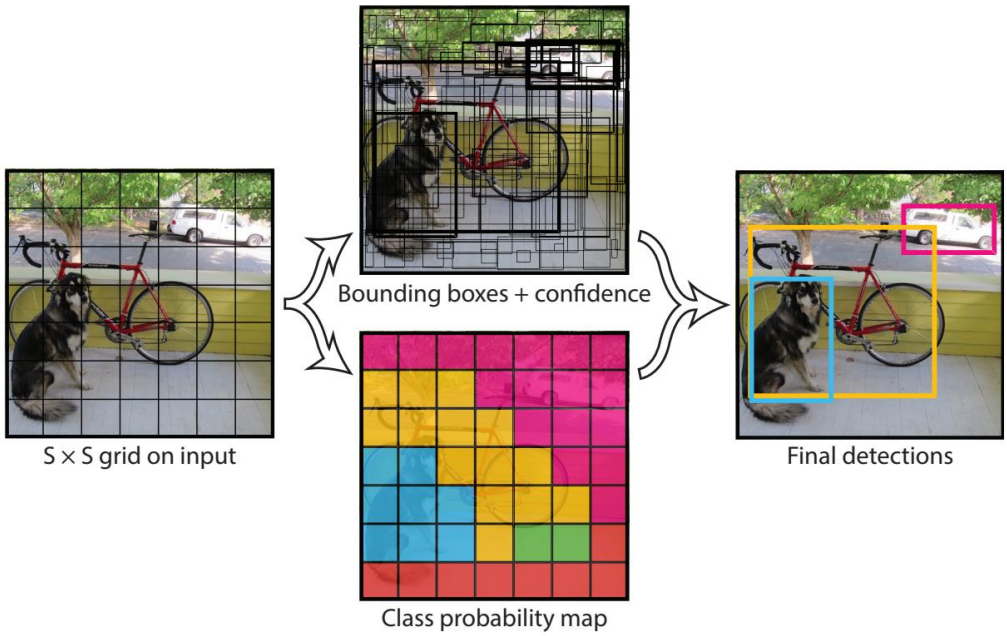

图 2-1　yolo 网络检测物体的过程[9]

之后 Redmon J, Farhadi A 等(2017)[10]中对 yolo 网络就行升级，变成了 yolo-v2。yolo-v2 相较于前一代有了很大的提升，主要在物体定位的精度和更高的速度。在检测网络中放弃了全连接层，引入了 faster RCNN 中对 anchor box 的思想改进了网络结构，其输出层使用卷积层来代替全连接层，通过卷积层来预测 bbox。相比较 yolo-v1，v2 的识别种类、精度、速度和定位准确性都有所提升

V2 在所有卷积层后全部使用 batch normalization（批量归一化，可以显著提高收敛性）优化网络性能的同时规范网络，减少过拟合的发生。删掉了一个池化层使特征分辨率更大一点,在保证准确率几乎不变的情况下使网络的召回率上升， 同时使用了残差网络中恒等映射模块，将浅度特征和深度特征图相连，使模型有了细粒度特征，增加了模

型的性能。

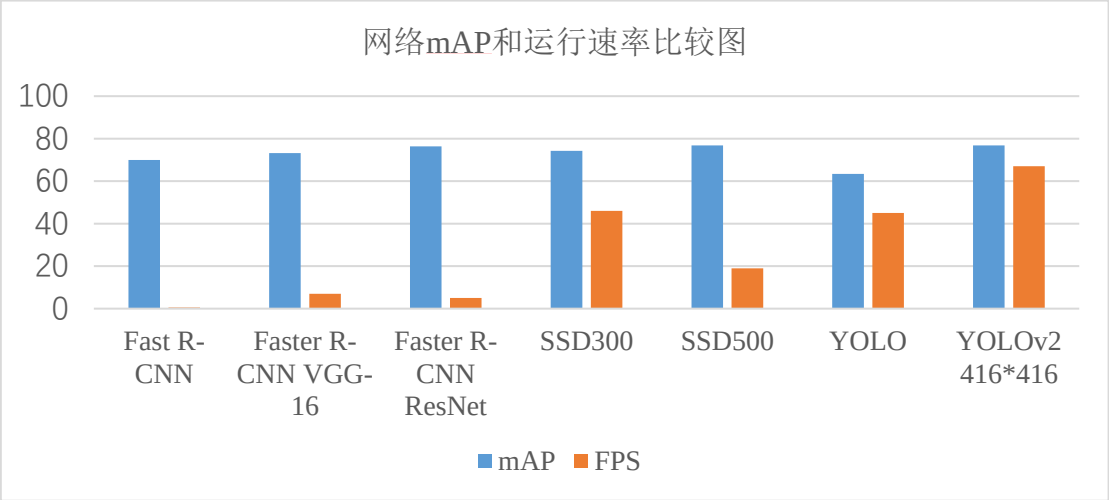

图 2-2　常见网络在 VOC2007 数据集中精度和处理速度的比较

从图 2-2 可以看出，yolo-v2 网络在对视频中检测并定位行人的准确率和速度都是非常优秀的，符合需求。因此在本次设计中使用 yolo-v2 网络框架来进行行人检测。

2.3　行人重识别网络选取及分析

要能分辨出不同的人，在一群人中找到需要的那个人，就需要进行行人重识别。行人重识别，就是利用计算机视觉技术判断图像或者视频序列中是否存在特定行人的技术，被认为是一个图像检索的子问题。给定一个监控行人图像，检索跨设备下的该行人图像。旨在弥补目前固定的摄像头的视觉局限，并可与行人检测/行人跟踪技术相结合，可广泛应用于智能视频监控、智能安保等领域[7]。

在本次毕设中，我侧重于使用行人重识别技术来区分店员和行人，店员就是我选取的那个“特殊行人”。通过行人重识别方法，给与适当的阈值，将店员和行人区分开来。但是因为监控视频清晰度不高，店员的图像比较模糊，行人重识别的准确率会受到一定的影响，因此需要选择一个自身素质够高的行人重识别网络。

云从科技的 mgn(multiple granularities network)网络在行人重识别上获取重大突破，同时在 Market-1501, CUHK03, DukeMTMC-reID 三个数据集上刷新了世界纪录，我决定参考他们论文里的方法进行行人重识别的实现。

在 Wang G, Yuan Y, Chen X 等(2018)[11]中，提出使用融合特征来进行行人重识别，

即将全局特征和局部特征相融合。同时设计了一个多分支的端到端的深度网络来降低训练的难度，提高鲁棒性。损失函数使用的 softmax 和 triplet。

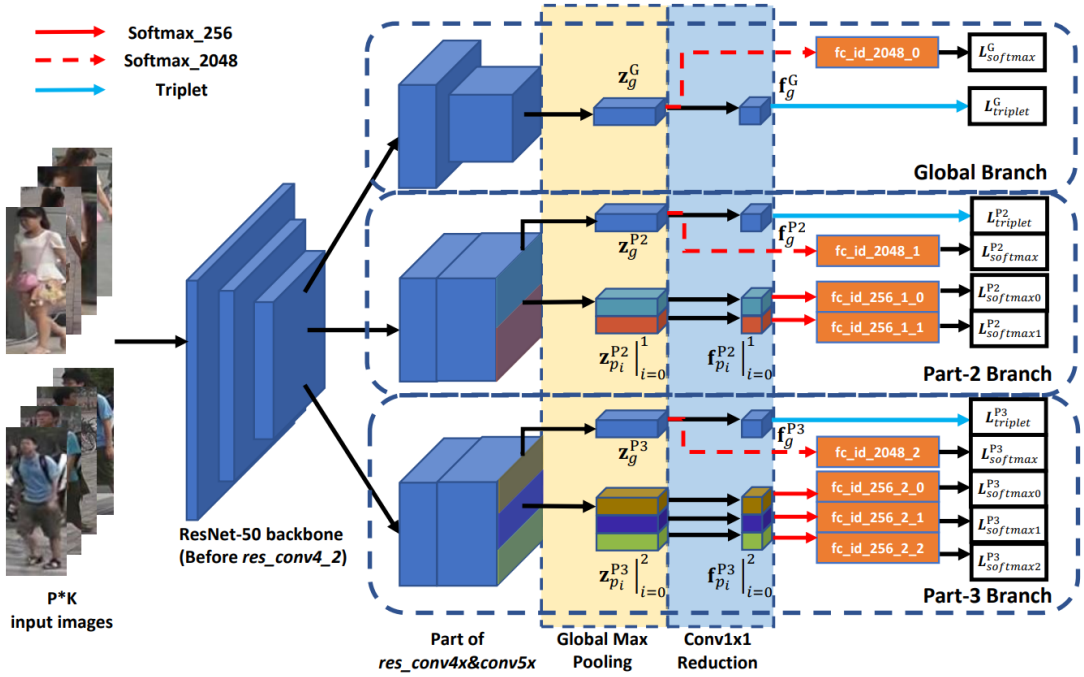

图 2-3　多粒度网络(MGN)结构[12]

此网络前三层使用 Resnet50，用来提取图像的基础特征，高语义特征则设计了三个分支：第一个分支负责整张图片的全局特征提取；第二个分支将图像分成上下两个粒度，提取其中每个粒度的语义特征；第三个分支则将图像分成上中下三个粒度进行语义特征提取。这样特征信息是多层次的，有利于提高精准度。

只有第一个分支在全局最大池化前进行一次降维，后两个分支没有降维操作，因为要分成几个粒度进行学习，这样有助于学到更能反映细节的特征；全局池化后还要进行一次降维（最大化池化）操作，将 2048 维的特征降维成 256 维的特征，这样可以方便特征计算。测试时候使用的是 8 个 256 维特征串联成的 2048 维特征来表示一个图像。

在三个分支中，对未降维的全局特征和粒度特征进行全连接的操作，之后使用 softmax loss 进行约束；对降维的全局特征使用 triplet loss 进行约束。粒度特征不能使用 triplet loss，因为有些粒度本身可能只是背景，使用 triplet loss 在粒度为纯背景时会让网络向背景图的类别进行学习，导致准确度下降。

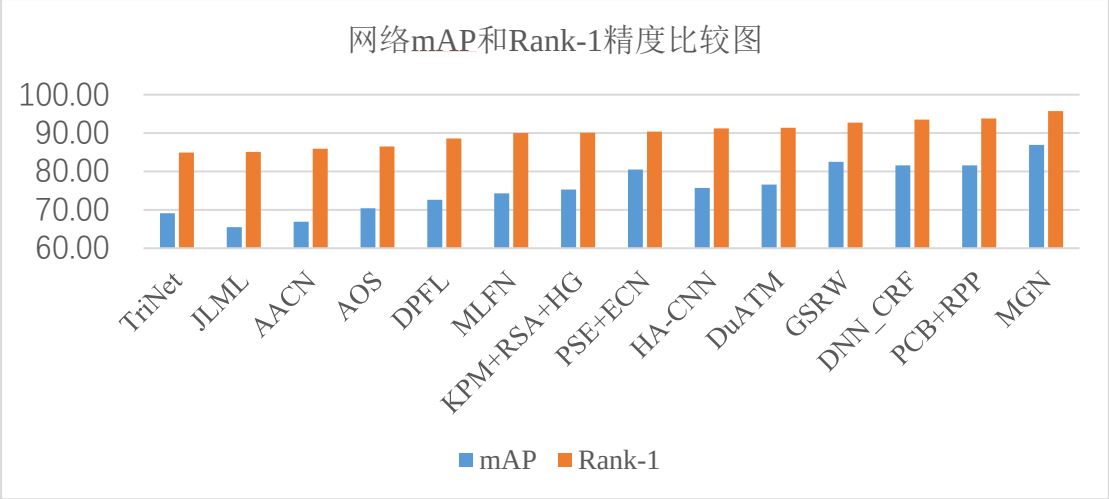

图 2-4　常见重识别网络在 market1501 数据集中对单查询的精度对比

由图 2-4，mgn 网络融合了不同粒度的特征，在行人重识别方面精度很高，mAP 和 Rank-1 精度超过了目前常用的其他网络。因此在本次设计中使用 mgn 网络框架进行行人重识别。

2.4　动作识别网络选取及分析

要检测出员工特定的行为，就要识别出员工的动作，就要有一个动作识别网络进行动作识别。因为本次毕设是针对视频进行员工的行为分析，行为分析目前有几个选择：1、使用姿态识别识别骨骼点；2、使用 3d 卷积网络进行时空特征的提取进行行为分析；3、使用卷积网络进行特定动作的识别。

对于姿态识别，网络输出为身体各部位骨骼点的角度，但是有骨骼点的角度并不能推测出人的动作行为，因为人的动作存在相似性，拿起手机玩和端起水杯在姿态识别里看起来是差不多的，无法通过骨骼点准确定位行为动作；且目前姿态识别网络对骨骼点识别精度也不是非常高，同时还有因摄像头机位变换导致骨骼点角度失效的情况发生，因此姿态识别不适合毕设要求。

对于 3d 卷积网络，它是通过一个 3d 卷积核同时对视频中连续的几帧图像进行学习，也就提取了视频中的时空特征，对于检测持续一段时间的行为或需要一段时间完成的行为准确率高。但是 3d 卷积网络精度受视频质量影响较大，监控视频往往是抽帧存储的，帧与帧之间不是具有时间连续性的，3d 卷积出的结果精度会下降；而且要检测玩手机动作的话不需要时空特征也可以检测出来，考虑到 3d 卷积网络对视频质量有着较

高的要求，因此没有选择 3d 卷积网络进行行为识别。

因此最终选择使用卷积网络进行行为识别。通过对卷积网络进行特定动作的训练，使网络对所要检测的动作敏感，因此需要选取一个性能优秀的 cnn 网络来进行动作识别从而做到对员工的行为分析。残差神经网络[12]就很好的符合要求。

残差网络模型很好地解决了随着深度的增加，网络的训练会变得越来越困难的问题，因此残差网络在深度很深的情况下也可以进行训练，网络性能远超传统的网络模型[6]。残差网络之所以叫做残差网络，是因为它定义了一种残差函数，这里的残差函数借用了高速网络(Highway Network)[17]的想法（高速网络比起普通网络，多了两个关于输入的带参的非线性变换$T(x)$和$C(x)$，分别称作“变换门”和“携带门”。变换门负责控制变换的强度，携带门复测空时原输入信号的保留强度，形式为：

$$y = F(x) * T(x) + x * C(x) \tag{2-4-1}$$

这种结构增加了恢复原始输入的可能，比起原来的结构更加灵活。将高速网络中的变换门和携带门都设定为恒等映射，因此有：

$$y = F(x) + x \tag{2-4-2}$$

此网络需要的函数为右端的残差项，因此叫做残差函数。

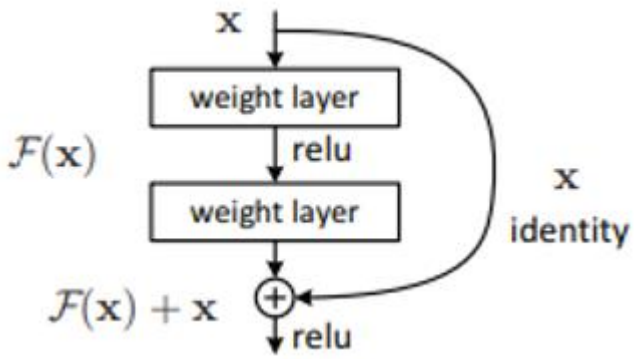

图 2-5　残差学习模块[12]

如图 2-5，残差学习模块有两个分支，一个是左侧的残差函数，一个是右侧的对输入的恒等映射（利用快速链接提取出输入的恒等映射）。这两个分支相加后经过 ReLU 激活函数，最后形成了整个残差学习模块。多个学习模块叠加的网络结构就被称作残差网络。

残差网络性能优秀，同时非常成熟，比起 Alex-Net[14], VGG-Nets[15], NIN[16], 运用残差网络进行动作识别精度可以得到保证。因此选择残差网络进行动作识别。

2.5 损失函数介绍

我们就所选网络中使用到的几个损失函数进行介绍。

2.5.1 softmax 损失函数

Softmax[19]是将神经网络得到的多个值进行归一化处理，使得到的值在[0,1]之间，同时进行反向传播[5]。从而让得到的结果可看做概率，某一类别概率越大，说明归类为某一类别的可能性是越大的。在一个神经网络后面添加 softmax，此时，数据真实的标签就是样本真实的分布，而经过 softmax 得到的结果就是预测的结果，在这种情况下就使用交叉熵函数作为损失函数：

$$L = \sum -\hat{y}_i \, ln \, y_i \tag{2-5-1}$$

其中y_i是神经元的输出，同时也是预测结果，$\hat{y}_i$是第i个类别的真实值，$\hat{y}_i$只能取0或者1，在softmax中取以e为底的对数，方便运算。具体实现过程如下图：

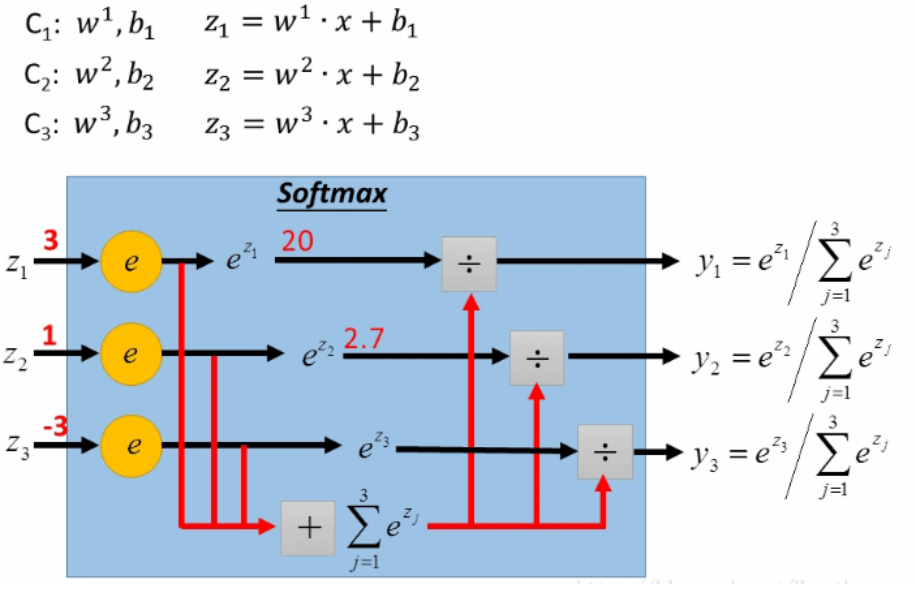

图 2-6 softmax 计算过程

由上图可知，对同时存在的多个类别，softmax 可以定义为：

$$J=-\frac{1}{m}\left[\sum_{i=1}^{m}\sum_{j=1}^{k}l\{y^{(i)}=j\}\,ln\frac{e^{y_j^{(i)}}}{\sum_{l=1}^{k}e^{y_l^{(i)}}}\right] \tag{2-5-2}$$

其中，$l\{y^{(i)}=j\}$ 是一个示性函数，只有当大括号中的值为真时才取值为 1，否则为 0，该函数可以看作公式 2-5-1 中的$\hat{y}_i$。

2.5.2 yolo 检测网络损失函数

$$L_{\text{coord}}=\lambda_{coord}\sum_{i=0}^{S^2}\sum_{j=0}^{B}l_{ij}^{obj}\{[(x_i+\hat{x}_i)+(y_i+\hat{y}_i)]$$

$$+[(\sqrt{w_i}-\sqrt{\hat{w}_i})^2+(\sqrt{h_i}-\sqrt{\hat{h}_i})^2]\} \tag{2-5-3}$$

$$L_{\text{con}}=\sum_{i=0}^{S^2}\sum_{j=0}^{B}l_{ij}^{obj}(C_i-\hat{C}_i)^2+\lambda_{noobj}\sum_{i=0}^{S^2}\sum_{j=0}^{B}l_{ij}^{noobj}(C_i-\hat{C}_i)^2 \tag{2-5-4}$$

$$L_{\text{class}}=\sum_{i=0}^{S^2}l_i^{obj}\sum_{c\in class}(p_i(c)-\hat{p}_i(c))^2 \tag{2-5-5}$$

$$L=L_{coord}+L_{con}+L_{class} \tag{2-5-6}$$

检测网络模型的损失函数不是单一的，而是分为了三个 Coordinate Loss（坐标损失函数），Class Loss（类别损失函数）和 Confidence Loss（置信度损失函数）如上述公式，三种损失函数分别标记为$L_{\text{coord}}, L_{\text{con}}, L_{\text{class}}$，这样分开是有原因的：损失函数的目的是为了让坐标、类别、置信度三个方面都能有很好的平衡，要做的这一点，若只取用平方误差损失函数会背离这个目的，效果难以保证，因为：

1、定位误差和分类误差维度不同，两者权重相同明显是不够合理的；

2、假设使用平方误差损失函数，如果出现一个网格中是没有物体的情况，那么该网格中的 bbox 的置信度就会立刻变为 0，这种变化过于激烈，很容易造成网络不稳定甚至发散。

因此使用三种不同的损失函数共同作用，以提升性能。接下来分别看看每个公式中部分功能。

首先是公式 2-5-3，这是一个坐标预测（损失函数）公式，λ_{coord}为坐标损失权重 ，该权重可以更改，训练时设置为 1.0；x,y为模型检测出的物体的左上角坐标，$\hat{x},\hat{y}$为物体实际的左上角表，w,h为模型检测出的物体的宽和高，$\hat{w},\hat{h}$为物体实际的宽和高。w,h和$\hat{w},\hat{h}$取平方根的原因是为了突出不同大小 boxx 对大小预测的敏感性，通过这个方法，较小的 bbox 可以比较大的 bbox 对大小偏移更加敏感，有利于网络的训练。公式中的l_{ij}^{obj}是判断低 i 个 bbox 中是否负责这个 object（通过计算 bbox 与真实 box 的 IoU 来判

断，值最大的 bbox 负责指定 object）。这一步的目的同样是为了加快训练速度，可以更快的学习到正确位置。

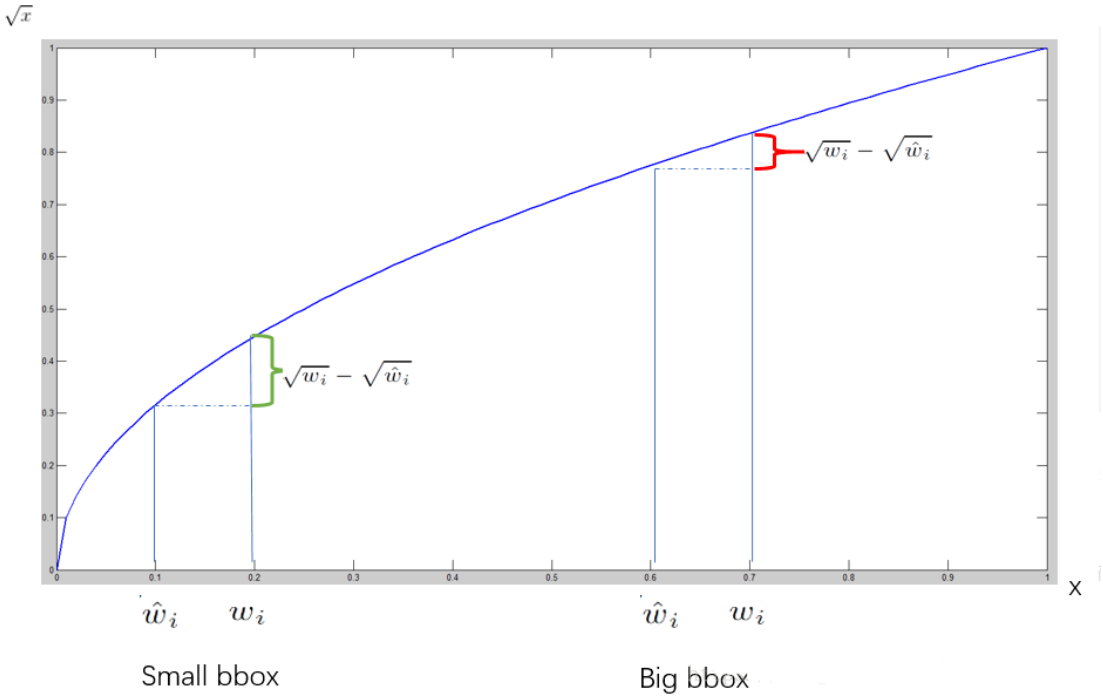

图 2-7　不同大小 bbox 对宽高变化的敏感性

对于公式 2-5-4 与 2-5-5，情况和 2-5-3 类似。2-5-4 中λ_{noobj}是无目标置信度权重，为了减少输出突变的影响，这个值一般大于 1，训练时设置为 5.0。$\sum_{i=0}^{S^2}\sum_{j=0}^{B} l_{ij}^{obj}(C_i - \hat{C}_i)^2$是对含有 object 的 bbox 的置信度预测（损失函数），$\lambda_{noobj}\sum_{i=0}^{S^2}\sum_{j=0}^{B} l_{ij}^{noobj}(C_i - \hat{C}_i)^2$是对不含 object 的 bbox 的置信度预测（损失函数）。2-5-4 与 2-5-5 中同样有 bbox 判断机制，目的相同，都是为了加快训练速度。

2.5.3 triplet-loss 损失函数

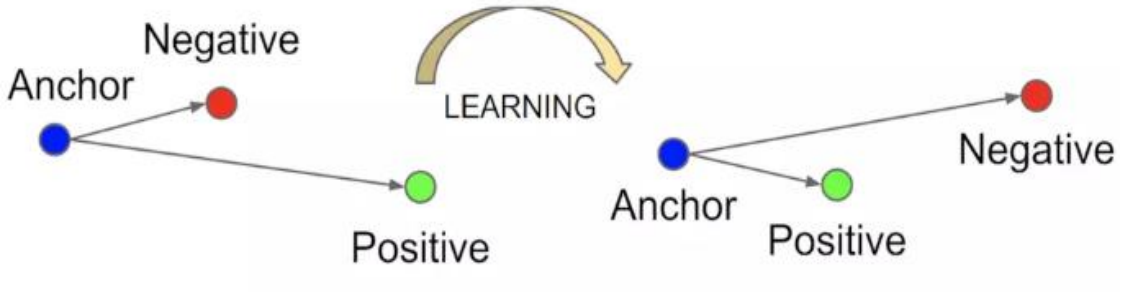

图 2-8 triplet-loss 目的[20]

triplet 是一个三元组，包含一个 anchor 样本，一般是从训练集中随机抽取用来当“标准”，另随机取两个样本，其中一个为和 anchor 同一类的样本，记做 positive；而另外一个则是和 anchor 不一类的样本，记做 negative，这样的三个样本构成了一个三元组。而在这三元组基础上的 triplet-loss[20]，就是通过三元组中三个元素的特征表达来进行优化网络，对于三元组中的样本的特征表达，我们可以记做 x_a、x_p、x_n，triplet-loss 要使 x_a、x_p 之间的距离尽可能的小，而 x_a 与 x_n 之间的距离尽可能的大，且需要使两者有一个预计的差值 margin：

$$L = max(d(a,p) + margin - d(a,n), 0) \tag{2-5-7}$$

具体公式如 2-5-7，实际训练时候往往会有三种情况：1、easy triplets，2、hard triplets，3、semi-hard triplets。easy triplets 就是一开始就是$d(a,p) + margin < d(a,n)$,不需要进行优化就能满足要求；hard triplets 则是$d(a,n) < d(a,p)$，即 anchor 与 positive 的距离很远；最有一个 semi-hard triplets 就是介于上述两者的情况。

2.6 本章小结

本章根据实际情况的分析，提出了实现员工行为分析过程中三个主要步骤所对应的三个需求，这三个问题进行分析，最终得出结论，确定了选择 yolo 网络框架进行行人检测；选择 mgn 网络框架进行行人重识别；选择残差网络进行动作识别的方案。并介绍了所选网络及网络运用的损失函数。

第 3 章　网络模型搭建及训练

3.1　yolo-v2 网络搭建及训练

3.1.1　yolo 分类网络结构

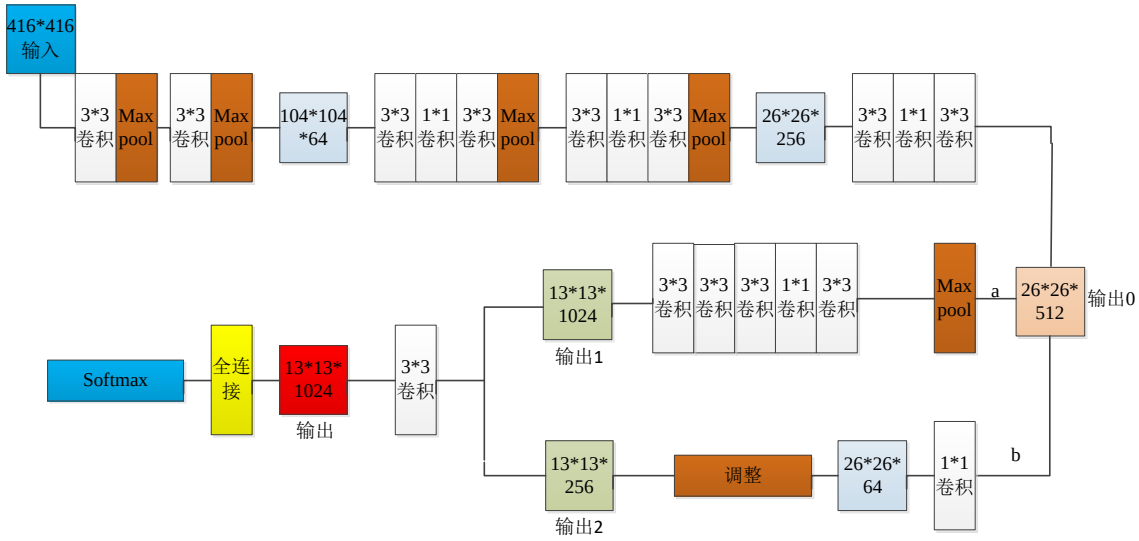

图 3-1　yolo 分类网络结构

由图可知，网络输入是 416*416*3 的数据，在 stage 网络中，先经过两层 3*3 卷积层和两层 2*2 池化层进行降维，将其变为 104*104*64 的数据，然后经过一个 3*3,1*1,3*3 三层卷积组加上一个 2*2 池化层提取特征并降维为 52*52*128 的数据，然后再经过一个 3*3,1*1,3*3 三层卷积组加上一个 2*2 池化层提取特征并降维为 26*26*256 的数据，之后再经过一个 3*3,1*1,3*3 三层卷积组得到 26*26*512 的输出 0。

此处产生一个分支：

a 分支：输出 0 经过一个 2*2 池化层变为 13*13*512 的输出，再经过一个 3*3,1*1,3*3,1*1,3*3 的 5 层卷积结构输出一个 13*13*1024 的输出，再经过两层 3*3 的卷积得到 13*13*1024 的输出 1。

b 分支：输出 0 讲过一个 1*1 卷积层并经过调整成一个 13*13*256 的输出 2.

输出 1(13*13*1024)与输出 2(13*13*256)通过一个 3*3 卷积成一个 13*13*1024 的输出，此输出将浅层特征与深层特征融合，使网络具有多粒度的特性。之后经过一个 1000

神经元的全连接层，最后由 softmax 损失函数进行约束。

网络的每个卷积层层都使用 batch normalization，都使用 ReLU 激活函数。

3.1.2 yolo 检测网络结构

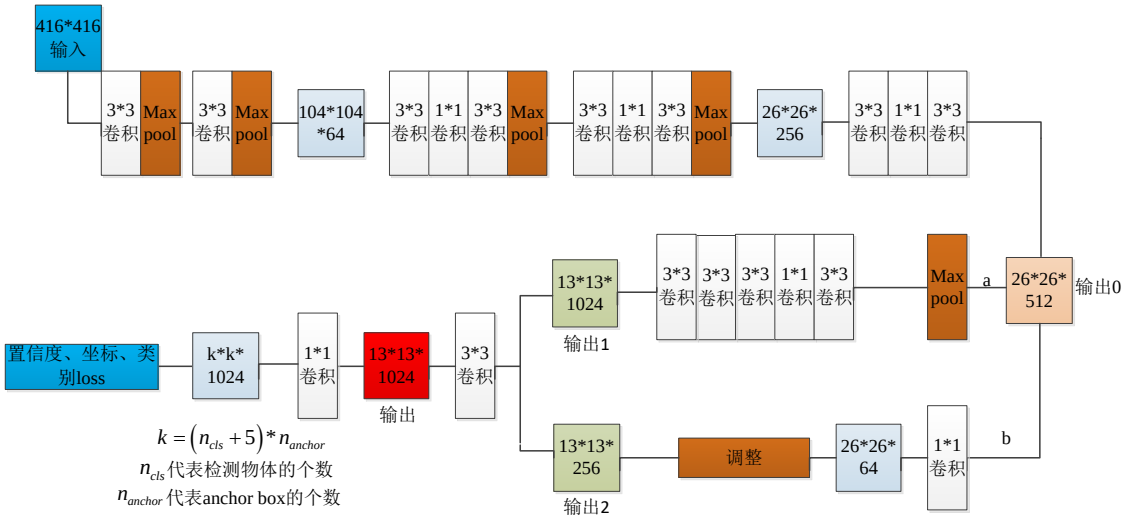

图 3-2 yolo 分类网络结构

由图可知，网络输入是 416*416*3 的数据，在 stage 网络中，先经过两层 3*3 卷积层和两层 2*2 池化层进行降维，将其变为 104*104*64 的数据，然后经过一个 3*3,1*1,3*3 三层卷积组加上一个 2*2 池化层提取特征并降维为 52*52*128 的数据，然后再经过一个 3*3,1*1,3*3 三层卷积组加上一个 2*2 池化层提取特征并降维为 26*26*256 的数据，之后再经过一个 3*3,1*1,3*3 三层卷积组得到 26*26*512 的输出 0。

此处产生一个分支：

a 分支：输出 0 经过一个 2*2 池化层变为 13*13*512 的输出，再经过一个 3*3,1*1,3*3,1*1,3*3 的 5 层卷积结构输出一个 13*13*1024 的输出，再经过两层 3*3 的卷积得到 13*13*1024 的输出 1。

b 分支：输出 0 讲过一个 1*1 卷积层并经过调整成一个 13*13*256 的输出 2.

输出 1(13*13*1024)与输出 2(13*13*256)通过一个 3*3 卷积成一个 13*13*1024 的输出，此输出将浅层特征与深层特征融合，使网络具有多粒度的特性。

网络的每个卷积层层都使用 batch normalization，都使用 ReLU 激活函数。

输出，不经过全连接层，而是经过一个有 k 个 1*1 卷积核的卷积层，k 的取值与检测物体的个数有关：

$$k = (n_{cls} + 5) * n_{anchor} \tag{3-1-1}$$

其中n_{cls}代表检测物体的个数，n_{anchor}代表 anchor box 的个数。

数据通过卷积层后经过 Coordinate Loss（坐标损失函数），Class Loss（类别损失函数）和 Confidence Loss（置信度损失函数）进行约束。

3.1.3 数据集介绍

ImageNet 数据集：ImageNet 数据集是一个用于视觉对象识别研究的大型可视化数据集，有超过 1400 万的图像，包含 2 万多个类别。其中常用的是 ISLVRC 2012 比赛用的子数据集，这个数据集训练集有 128 万多带标签的图像，验证集有 5 万张带标签的图片，测试集有 10 万张不带标签的图片。图片标签不光有类别，还有坐标信息。ImageNet 数据集是用来训练 yolo 分类网络的训练集。

VOC 数据集：VOC 数据集是用来训练 yolo 检测网络的训练集，VOC 2007 数据集来检测的部分一共有 9000 多张图片，且每张图片都有 xml 格式的标注，标注了图片中各个 object 的信息（比如 bbox 坐标，object 检测难度及类别等），标注的物体包括人、动物、交通工具、家具等 20 个类别。

3.1.3 yolo 网络训练

由上述数据集介绍可知，ImageNet 是一个非常大的数据集，自己训练的话每个 epoch 的 batch 会比较大，由于电脑性能限制训练时间会很长，而且最后训练训练精度也会不如官方给的成熟的网络权重。因此 yolo 分类网络使用的就是官方的权重，官方权重经过长时间的训练，训练参数非常成熟，性能也是比较优异；

Yolo 检测网络的检测过程，首先是将图片进行处理，处理为 416*416 大小的图片，然后将图片划分为多个 32*32 的网格，每个网格预测五个大小的 bbox，每个网格都会输出预测的所有 bbox 的坐标、bbox 预测大小以及分类置信度。之后对所有网格的输出进行抑制，之后对网格输出进行处理，输出检测结果。而检测网络的训练则是和三个损失函数紧密联系的。通过损失函数的优化，减少无物体的网格对网络的影响，同时均衡小框和大框单次修正的量（利用开平方根后其对大小变化敏感性不同实现）。

对于检测网络，训练所使用的 VOC 数据集比较小，而且并没有现成的符合要求的训练好的权重，因此我使用 VOC 数据集自己训练。由于我们是训练检测网络，因此我们需要的 xml 标签文件是在 annotation 文件夹中，对应的图片是在 JPEGImages 文件夹

中，命名从 000001.jpg 开始。

首先运行 voc_label.py 文件生成 label（需要把全部 20 类都生成），这一步是从 xml 文件中读取每个 object 的四个顶点坐标和分类信息，label 文件生成之后再开始进行训练。虽然说 VOC 数据集有 20 个类别，包括飞机、自行车、鸟、船、车、狗等，但是因为本次对 yolo 检测网络的训练目的是让网络能进行行人检测，因此训练时候 class 参数只选择 person，忽略其他 19 个选项。训练的图像格式设置为 416*416，同时在训练的时候对图像随机（0.5 概率）进行 HSVAdjust（利用 HSV 颜色空间来调整图像，先将灰度归一化，之后将 rgb 色域转换为 hsv 色域，再对转化后的灰度进行 0-1 化调整，让灰度保持在[0.0,1.0]的范围内，之后再从 hsv 色域转换到 rgb 色域，再对灰度进行还原）、VerticalFlip（图像垂直翻转）、Crop（图像随机裁剪）和 Resize（调整大小）。训练时候设置学习率为 0.001，weight decay 设置为 0.0005，momentum 设置为 0.9，训练 160 个 epoch。

3.2 行人重识别网络搭建及训练

3.2.1 mgn 网络结构及改进

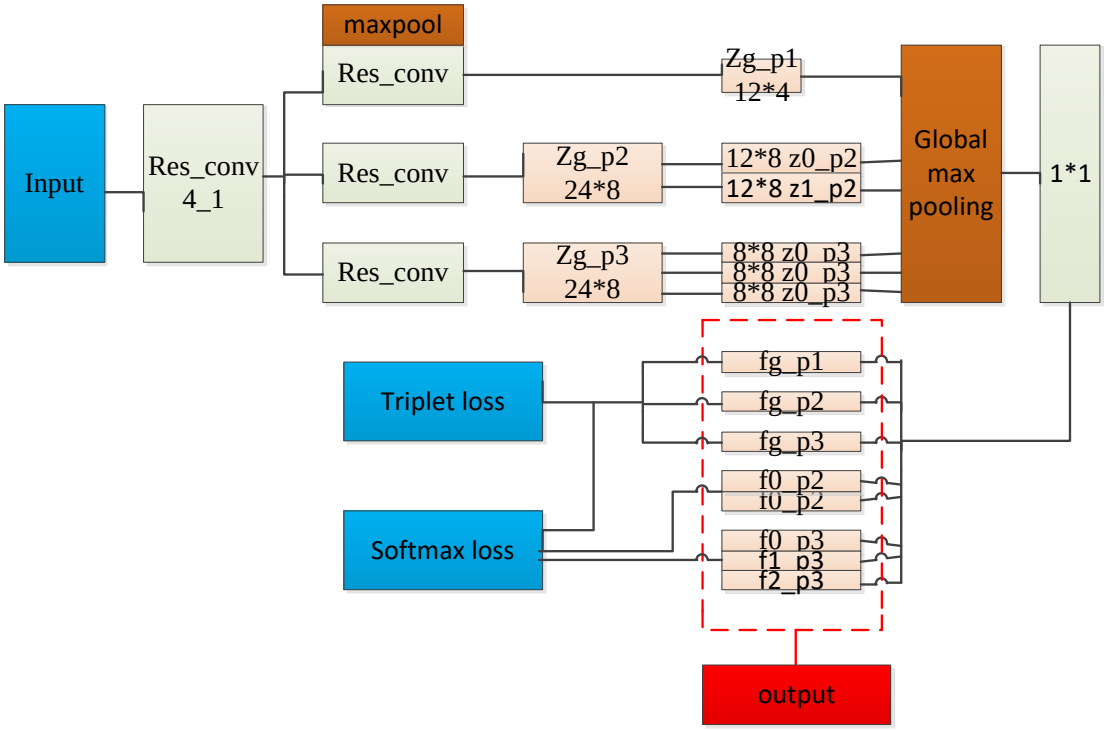

图 3-3 mgn 网络结构

Mgn 是包含一个用于提取全局特征的分支和两个用于提取局部特征的分支。因此我们选择用 resnet50 为底层模型，在 res_conv4_1 之后分成三个不相关的分支进行训练，分别是 p1、p2 和 p3。p1 是 global（part-1）的分支，p2 的 part-2 的分支，p3 是 part-3 的分支。p1 分支因为是 global 分支，其处理方式和 resnet-50 是无异的，因此仍然使用 resnet-50 的网络的结构， p1 的特征 zg_p1 的特征图是经过 maxpool 降维的，大小为 12*4；p2 和 p3 因为需要处理粒度特征，因此要保证特征图的大小，所以不进行下采样，因此重新定义网络，使最后特征图输出为：对于 p2， zg_p2 的特征图未经过 maxpool 降维的，大小为 24*8，其下两个粒度 z0_p2 、z1_p2 的特征图是 zg_p2 均分的，大小都为 12*8；对于 p3，zg_p3 的特征图是未经过 maxpool 降维的，大小为 24*8，其下三个粒度 z0_p3 、z1_p3、 z2_p3 的特征图是 zg_p3 三等均分的，大小都为 8*8。

网络最后的输出特征图谱 predict 就是由这 8 个特征经过一次卷积降维组合而成的，维度为 2048。而损失函数的构成就要相对复杂一点，zg_p1、zg_p2、zg_p3 在通过全局最大化池化后经过 1*1 卷积降维处理后也变成了输出 fg_p1、fg_p2、fg_p3，这三个输出经过 softmax 和 triplet loss 后产生了 6 个损失函数，加上 zg_p2、zg_p3 下 5 个粒度经过全局最大化池化和 1*1 卷积降维后经过 softmax loss 后产生的 5 个损失函数，组成了 mgn 原始网络的 11 个损失函数。

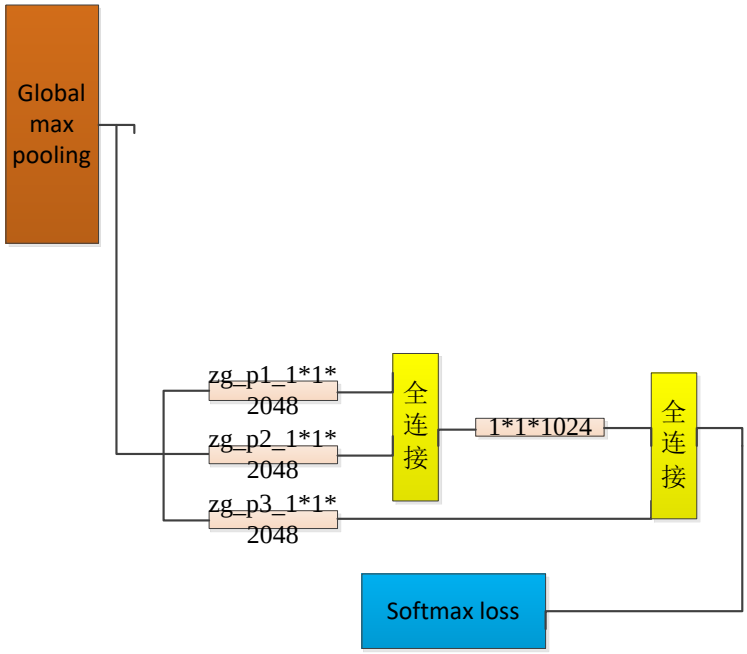

图 3-4　新增损失函数结构

原始的 mgn 网络一共有 8+3 共 11 个损失函数，我们对其进行改进。原有的 mgn 网络只是将三个分支分开、分小块进行 softmax 或者 triplet loss，并没有融合多支路的输出进行 triplet 或者 softmax，因此加上了个全支路输出的 softmax loss，将全局最大化池化后输出的三个网络分支进行两次全连接操作，最后将输出的 1*1*3072 的特征使用 softmax 进行约束，增加了第 12 个损失函数，略微提升了网络的性能。

新增加的损失函数可以在一定程度上弥补因粒度划分导致的准确率丢失的现象（图像某一部分缺少行人，纯背景，这将导致准确率的丢失），可以提升网络的性能。

改进后的网络相比改进前总体上有 1.07%的提升（比较 rank1、rank3、rank5）。

3.2.2 数据集介绍

market1501 数据集：Market-1501 数据集在清华大学校园中采集，夏天拍摄，在 2015 年构建并公开。它包括由 6 个摄像头（其中 5 个高清摄像头和 1 个低清摄像头）拍摄到的 1501 个行人、32668 个检测到的行人矩形框。每个行人至少由 2 个摄像头捕获到，并且在一个摄像头中可能具有多张图像。

关于数据集目录：

bounding_box_test 中存放着用于测试集的 750 人，包含 19,732 张图像。

bounding_box_train 中存放着用于训练集的 751 人，包含 12,936 张图像。

query 和 gt_query 是用来方便进行 triplet loss 的文件夹，query 中为 750 人在每个摄像头中随机选择一张图像作为 query，gt_query 中的文件用于判断一个 query 的哪些图片是好的匹配（同一个人不同摄像头的图像）和不好的匹配（同一个人同一个摄像头的图像或非同一个人的图像）。

3.2.3 mgn 网络训练

由于网络是基于 resnet50 进行改建的，因此开始时候加载 resnet50 已经在 ImageNet 数据集上训练过的参数。在训练的时候对图像进行 HSVAdjust（利用 HSV 颜色空间来调整图像，先将灰度归一化，之后将 rgb 色域转换为 hsv 色域，再对转化后的灰度进行 0-1 化调整，让灰度保持在[0.0,1.0]的范围内，之后再从 hsv 色域转换到 rgb 色域，再对灰度进行还原）、VerticalFlip（图像垂直翻转）、Crop（图像随机裁剪）和 Resize（调整大小）的操作。使用 adam 优化算法计算反向传播时参数的变化量，初始学习率为 0.0003，训练 160 个 epoch，训练批次大小设置为 32。

改进前后的网络训练结果为：

表 3-1　改进 mgn 网络精度对比

指标精度	原网络	添加全体输出的 triplet loss
Rank-1	94.60%	94.40%
Rank-3	96.53%	98.00%
Rank-5	97.06%	99.00%

改进后的网络性能相比改进前的网络平均有 1.07%的提升，符合预期。

3.3　动作识别网络搭建及训练

3.3.1　数据集介绍

BOT2018：该训练集是 BOT2018 新零售技术挑战赛的训练集，训练集的内容是商场里监控摄像头拍摄到的图片集，记录了监控摄像头中行人的各种动作，其 label 中记录了每个行人的坐标、行人身份、以及行人站立还是坐下，行人是否有玩手机。训练集总共有 4500 张图片，测试集有 500 张图片，由于本次目的是员工行为检测，该数据集非常适合目的。使用该数据集训练网络，提升网络对识别玩手机动作的准确率，有助于找出员工玩手机偷懒的画面。

3.3.2　网络结构

网络使用 resnet50，输出一个 2048 神经元的全连接层，使用 softmax 函数进行约束。

3.3.3　动作识别网络训练

由于网络是基于 resnet50 进行改建的，因此开始时候加载 resnet50 已经在 ImageNet 数据集上训练过的参数。在训练的时候对图像进行 HSVAdjust（利用 HSV 颜色空间来调整图像，先将灰度归一化，之后将 rgb 色域转换为 hsv 色域，再对转化后的灰度进行 0-1 化调整，让灰度保持在[0.0,1.0]的范围内，之后再从 hsv 色域转换到 rgb 色域，再对灰度进行还原）、VerticalFlip（图像垂直翻转）、Crop（图像随机裁剪）和 Resize（调整大小）的操作。

由于网络结构比较简单，最开始训练的时候效果不是很好，使用学习率 0.0005 训练 30 个 epoch 后识别看手机动作的精确率只有 93.43%，通过阅读[18]，论文中提出了一种将视觉特征和姿态估计的骨骼点信息融合提升准确率的方法：输入的图像（或视频）通过一个多任务学习卷积网络得到视觉特征、概率图和姿势估计的骨骼点，将视觉特征和概率图进行融合并经过外观识别，与经过姿态识别的骨骼点进行联合识别人体的动作。

姿态估计的骨骼点信息在没有视觉特征的情况下是一种“多解释的信息”，人类有许多相似的动作，比如喝水和吃东西，如果没有视觉特征辅助的话，仅仅依靠姿态估计很难判断到底是在做什么事；而引入了视觉特征之后就可以有效提高准确率。由这篇论文提供的思路方法为出发点进行思考，对单一视觉特征来判断动作，如果引入骨骼点是不是会对准确率有所提升。

借此启发，使用了 AlphaPose 对训练集图片进行处理，让图片上的行人加上骨骼点，再次训练网络验证。经过学习率 0.0005 的 30 个 epoch 的训练后，最好准确率为 94.52%，之后又经过学习率 0.0001 的 30 个 epoch 训练及学习率 0.00001 的 30 个 epoch 训练后，准确率保持在 95.00%以上，最高准确率为 95.14%，相同的训练量相比较之前有 1.2%的提升。

3.4 本章小结

本章主要介绍了 yolo 和 mgn 的网络结构，介绍了对 mgn 网络所做的改进，并且总结了网络的训练效果，证实了对 mgn 网络的改进是有效的。

第 4 章　员工行为分析功能实现

4.1　结构总览

图 4-1　员工行为分析

员工行为分析功能实现程序所实现的功能为：输入一段视频，在视频中对特定的行人进行监控，一旦他有偷懒的动作就记录下来。程序的结构分为三个模块，如图 4-1，下面我们将按模块进行介绍。

4.2　前期准备

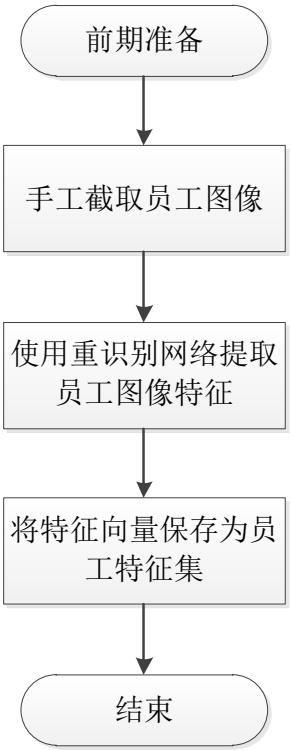

图 4-2　前期准备流程图

要实现预计功能，难点之一就是如何检测出店员。本毕设功能主要针对的是具有统一制服的店员（包括服装店、餐饮店等）该类店铺需要员工保持认真工作，服务于客户的氛围，对员工是否有认真工作十分看重，如果店员玩手机的话容易让顾客产生反感。根据[8]中的思想，第一步就是要收集员工的图片库（主要是收集制服的标准）。

图 4-3　手工截取的员工图片

在视频中截取清晰的、不同方位的视频员工截图多张，如图 4-3 中截取的图片，正身、侧身、背身、半身都应该进行截取。将截图保存到 uniform 文件夹中，之后使用训练好的行人重识别网络对图片进行处理，提取图片中的特征向量保存到本地 cvs 文件中，方便之后网络结果的对照。这样做的好处是可以在实际测试过程中减少程序对这几张图进行特征提取的操作，需要进行特征比对时直接与 cvs 文件中记录的特征向量进行比对，而不用每次比对都重新进行一次员工特征提取，提高了软件运行效率。

4.3　图像分析

图像分析是实现功能的重要环节，利用三种网络依次对图像进行分析处理。Yolo 网络负责行人定位，为后续网络提供行人坐标；mgn 重识别网络负责筛选行人，在行人中筛选出店员，只有店员才能进行下一个网络的分析，这样 可以避免算力浪费，节省分析时间；动作识别网络对店员图像进行分析，判断其是否在玩手机。下面将分网络进行具体讲解。

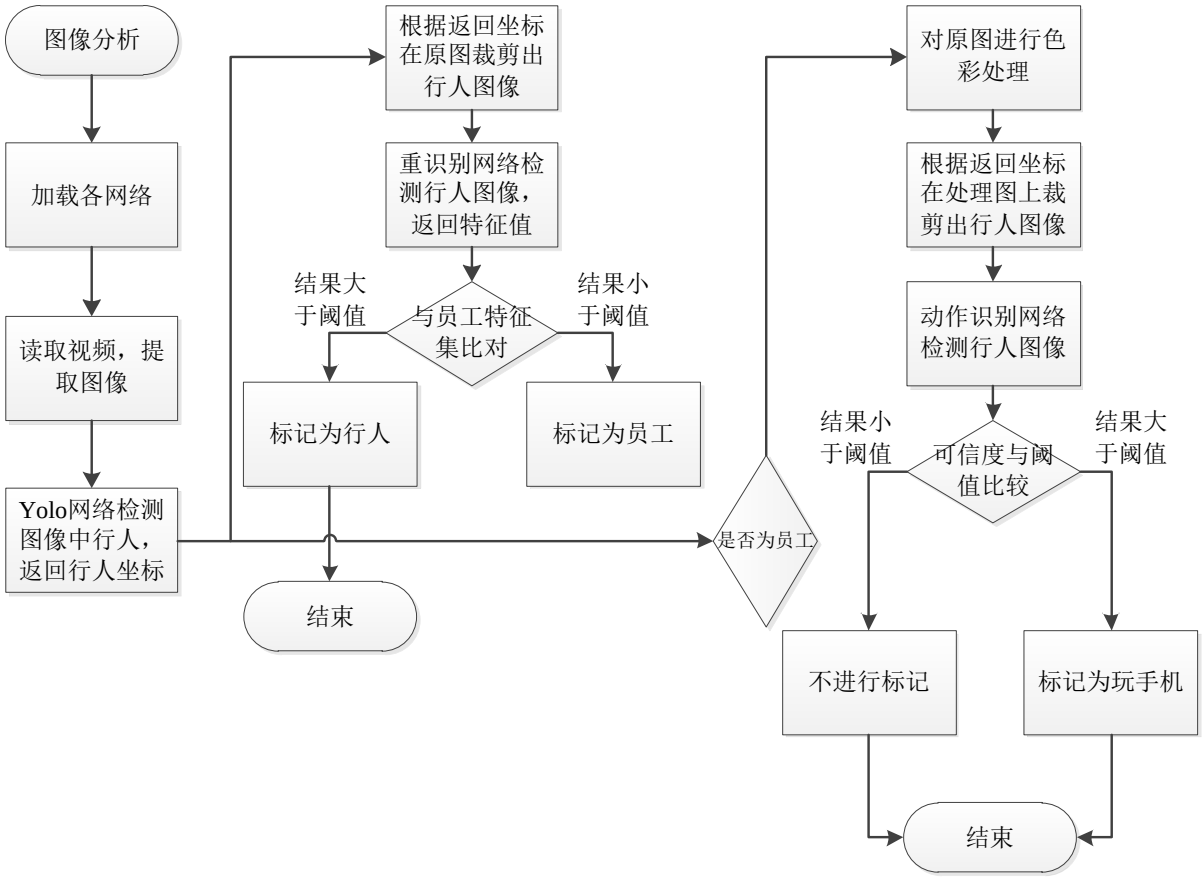

图 4-4　图像分析流程

4.3.1　提取行人坐标

将视频读入后每隔 10 帧提取 1 张图片，对图片进行 HSVAdjust（利用 HSV 颜色空间来调整图像，先将灰度归一化，之后将 rgb 色域转换为 hsv 色域，再对转化后的灰度进行 0-1 化调整，让灰度保持在[0.0,1.0]的范围内，之后再从 hsv 色域转换到 rgb 色域，再对灰度进行还原）、VerticalFlip（图像垂直翻转）、Crop（图像随机裁剪）和 Resize（调整大小）的操作，之后将图片输入给训练好的 yolo 网络，网络返回预测坐标和坐标转换率。由于输入网络时会对图像有大小调整，因此返回的坐标并非为原图中的坐标，需要利用坐标转换率将坐标转换为实际坐标。

4.3.2　判断行人类型

由于图像色彩对行人重识别不是干扰项，因此在提取到行人坐标后，直接在原图上根据行人坐标进行裁剪。将裁剪好的行人图片作为输入送入训练好的行人重识别网络中，

网络输出的特征向量会跟前期准备中产生的特征库中的特征向量进行对比，两者计算欧式距离，只要输出的特征与特征库中任意一个特征向量的欧氏距离小于阈值 α，就可以判断该行人应该属于员工，并置标志位为员工，进行下一步的操作；如果欧氏距离大于阈值 α，就可以认为该行人不是员工，置标志位为行人，停止进行下一步的操作。

4.3.3 判断员工行为

由于图像色彩对动作识别来说是干扰项，因此在提取到行人坐标后，应对原图进行色彩调整，保存为一个图片副本，在提取到行人坐标后，且经过重识别网络判断，该坐标对应行人为员工后，应在图片副本上根据坐标进行裁剪。将裁剪得到的员工图像作为输入送入动作检测网络进行分析，网络返回玩手机动作置信度，对置信度大于阈值的图片进行标记，标记为玩手机。

4.4 结果输出

结果输出就是一个根据网络检测到的标记在原图上使用不同颜色，不同名称的框对行人进行标记的过程，在这个过程中还会对检测到有玩手机员工的图片进行保存。

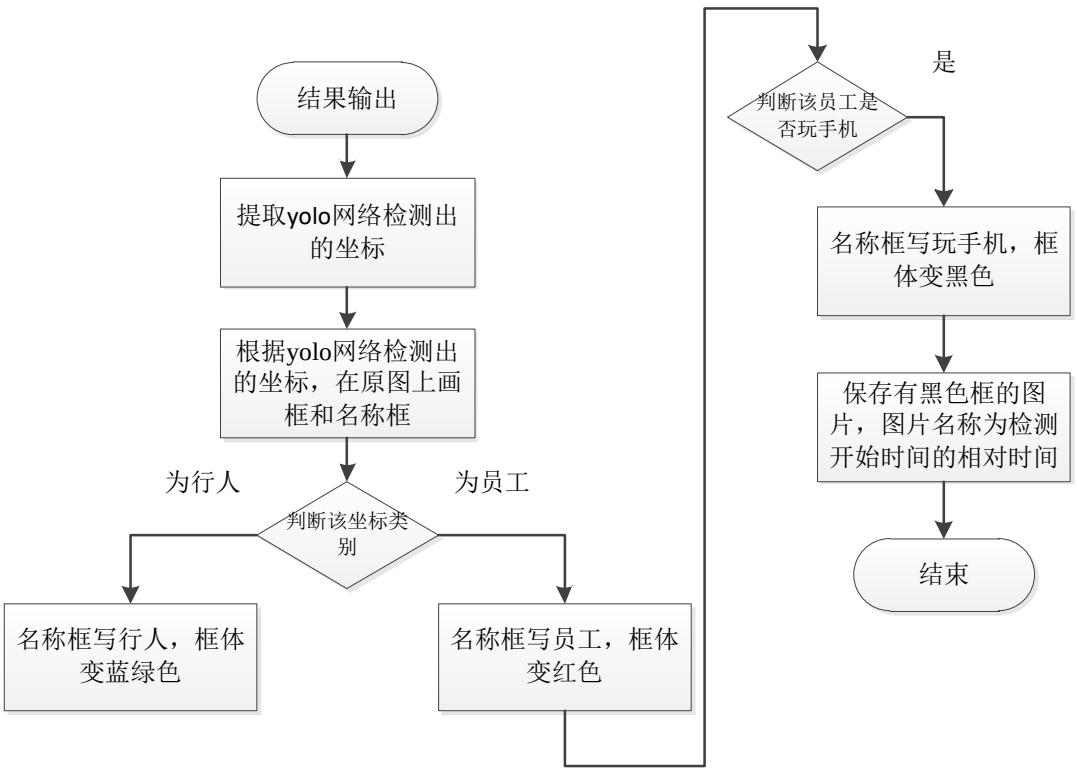

图 4-5 结果输出流程

当三个网络全部处理完一张图片的所有行人后，根据标志位：

1、如果行人截图有员工标志位和玩手机标志位都置位，就将对应的行人截图保存下来，作为检测出偷懒行为的结果；并且在视频原图上对应坐标处画一个黑框，表明员工在玩手机偷懒。

图 4-6　标记出在玩手机的员工

2、如果行人截图只有员工标志位，说明没有检测出员工在玩手机，这样就在视频原图的对应坐标处画一个红框，表明是员工。

图 4-7　标记出员工

3、如果标志位是行人的话，就在视频原图上对应坐标处画一个蓝色的框，表明当前行人为行人。

图 4-7 标记出一般行人

处理完一张图片上所有行人之后进行下一张图片的处理。

当处理完所以图片之后，将处理过的图片合成一段新的视频（因为是跳帧读取视频图片，虽然提升了处理速度，但是输出的视频会产生快进的效果）。

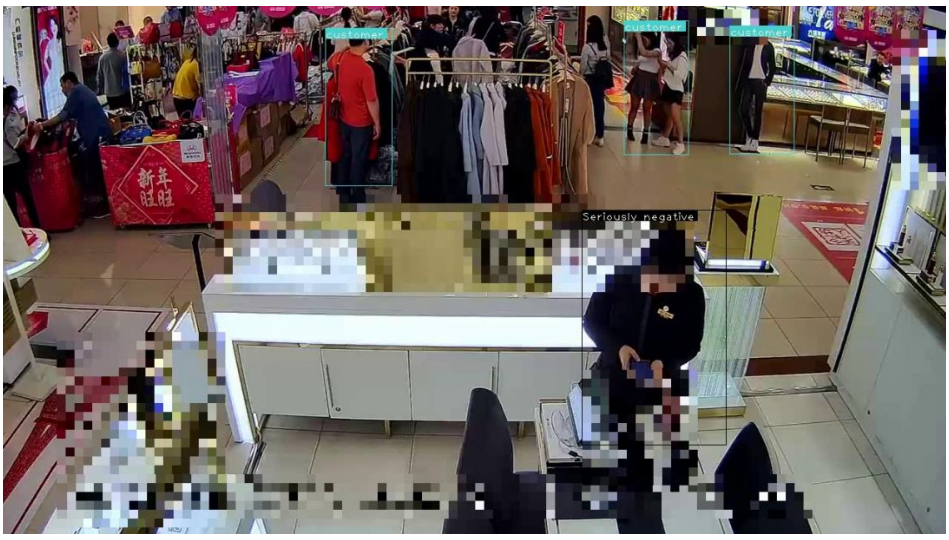

图 4-8 检测到店员玩手机的检测图

第 5 章　总结与展望

5.1　毕设工作总结

行人重识别和动作识别是实现员工行为分析的关键步骤，随着人工智能的进一步发展，未来行为分析将成为一种热门技术，不光是管理层对员工的行为监测，同时也可以应用到监狱等重要场所。因此，对行为分析的研究具有重要的现实意义。

本文首先介绍了本课题的研究背景和意义，综述了在进行目标时候必要的知识和方法；然后重点介绍了所搭建的深度学习网络的结构以及网络的训练；最后总结了功能实现的方法步骤，以下是总结：

1、本文研究了 yolo 的网络模型、原理及其训练的过程；研究并改进了 mgn 网络模型使其提升 1.2%的精度；使用姿态识别预处理数据，搭建并训练了动作识别网络，使其对特定动作的识别精度达到了 95.14%。

2、使用所训练的网络搭建了一个功能实现模块，实现了从输入的视频中检测员工是否玩手机偷懒的功能，并能使检测结果顺利保存。

3、所用方法在员工样本不够多的情况下容易将员工误判成行人，但样本太多又容易将行人误判成员工。

5.2　未来展望

本次毕业设计选取的是一个非常宽泛的题目：基于深度学习的员工行为分析系统。最完美最理想的员工行为分析不应该是只检测员工是否玩手机，而是对员工的一切行为都进行监控。但是目前并没有人能实现这个目标，而且目前姿态识别对精确区分人类行为的能力不强，大都停留在检测姿态的程度，在现有数据集的基础上我选择使用一般的卷积网络进行少量动作的识别，并用成熟的姿态估计网络预处理其训练集提升网络准确度，但是也提升有限。

1、希望未来能够将姿态识别和图像特征进一步的融合，提升网络的性能，实现高精度的多动作识别。

2、同时能寻找新的员工检测方法，目前的员工库方法在每次更换场景或者地点的时候都需要重新设置，同时用这个方法进行判断虽然具有一定的普适性但精度不高。

参 考 文 献

[1] 陈波, 余秋婷, 陈铁明. 基于传感器人体行为识别深度学习模型的研究[J]. 浙江工业大学学报, 2018, 46(4): 375-381.

[2] 尤磊. 基于深度学习的人体异常行为检测技术研究[D]. 哈尔滨工业大学, 2018.

[3] 廖鹏, 刘宸铭, 苏航, 等. 基于深度学习的学生课堂异常行为检测与分析系统[J]. 电子世界, 2018 (8): 97-98.

[4] 陆晴. 基于深度学习的异常行为识别算法研究[D]. 哈尔滨工业大学, 2018.

[5] 王忠勇, 陈恩庆, 葛强, 等. 误差反向传播算法与信噪分离[J]. 河南科学, 2002, 20(1): 7-10.

[6] 魏秀参. 解析深度学习：卷积神经网络原理与视觉实践[M]. 北京：电子工业出版社, 2018: 10, 14, 24-25, 30, 44-46, 48-54.

[7] 刘娜. 基于卷积神经网络的行人重识别算法[D]. 2017.

[8] 王金, 刘洁, 高常鑫, 等. 基于姿态对齐的行人重识别方法[J]. 控制理论与应用, 2017, 34(6): 837-842.

[9] Redmon J, Divvala S, Girshick R, et al. You only look once: Unified, real-time object detection[C]//Proceedings of the IEEE conference on computer vision and pattern recognition. 2016: 779-788.

[10] Redmon J , Farhadi A . YOLO9000: Better, Faster, Stronger[J]. 2017:6517-6525.

[11] Wang G , Yuan Y , Chen X , et al. Learning discriminative features with multiple granularities for person re-identification[C]// 2018 ACM Multimedia Conference. ACM, 2018.

[12] He K, Zhang X, Ren S, et al. Deep residual learning for image recognition[C] //Proceedings of the IEEE conference on computer vision and pattern recognition. 2016: 770-778.

[13] LeCun Y, Bengio Y, Hinton G. Deep learning[J]. nature, 2015, 521(7553): 436.

[14] Krizhevsky A, Sutskever I, Hinton G E. Imagenet classification with deep convolutional neural networks[C]//Advances in neural information processing systems. 2012: 1097-1105.

[15] Simonyan K, Zisserman A. Very deep convolutional networks for large-scale image recognition[J]. arXiv preprint arXiv:1409.1556, 2014.

[16] Lin M, Chen Q, Yan S. Network in network[J]. arXiv preprint arXiv:1312.4400, 2013.

[17] Srivastava R K, Greff K, Schmidhuber J. Training very deep networks[C]//Advances in neural information processing systems. 2015: 2377-2385.

[18] Luvizon D C, Picard D, Tabia H. 2d/3d pose estimation and action recognition using multitask deep learning[C]//Proceedings of the IEEE conference on computer vision and pattern recognition. 2018: 5137-5146.

[19] Liu W, Wen Y, Yu Z, et al. Large-margin softmax loss for convolutional neural networks[C]//ICML. 2016, 2(3): 7.

[20] Schroff F, Kalenichenko D, Philbin J. Facenet: A unified embedding for face recognition and clustering[C]//Proceedings of the IEEE conference on computer vision and pattern recognition. 2015: 815-823.

附录：网络搭建

Yolo 检测网络搭建：

```
class Yolo_Person2(nn.Module):
    def __init__(self, num_classes,
                 anchors=[(1.3221, 1.73145), (3.19275, 4.00944), (5.05587, 8.09892), (9.47112, 4.84053),
                          (11.2364, 10.0071)]):
        super(Yolo_Person2, self).__init__()
        self.num_classes = num_classes
        self.anchors = anchors
        #定义网络，每一层都使用 BN，ReLU
        self.stage1_conv1 = nn.Sequential(nn.Conv2d(3, 32, 3, 1, 1, bias=False), nn.BatchNorm2d(32),
                                          nn.LeakyReLU(0.1, inplace=True), nn.MaxPool2d(2, 2))
        self.stage1_conv2 = nn.Sequential(nn.Conv2d(32, 64, 3, 1, 1, bias=False), nn.BatchNorm2d(64),
                                          nn.LeakyReLU(0.1, inplace=True), nn.MaxPool2d(2, 2))
        self.stage1_conv3 = nn.Sequential(nn.Conv2d(64, 128, 3, 1, 1, bias=False), nn.BatchNorm2d(128),
                                          nn.LeakyReLU(0.1, inplace=True))
        self.stage1_conv4 = nn.Sequential(nn.Conv2d(128, 64, 1, 1, 0, bias=False), nn.BatchNorm2d(64),
                                          nn.LeakyReLU(0.1, inplace=True))
        self.stage1_conv5 = nn.Sequential(nn.Conv2d(64, 128, 3, 1, 1, bias=False), nn.BatchNorm2d(128),
                                          nn.LeakyReLU(0.1, inplace=True), nn.MaxPool2d(2, 2))
        self.stage1_conv6 = nn.Sequential(nn.Conv2d(128, 256, 3, 1, 1, bias=False), nn.BatchNorm2d(256),
                                          nn.LeakyReLU(0.1, inplace=True))
        self.stage1_conv7 = nn.Sequential(nn.Conv2d(256, 128, 1, 1, 0, bias=False), nn.BatchNorm2d(128),
                                          nn.LeakyReLU(0.1, inplace=True))
        self.stage1_conv8 = nn.Sequential(nn.Conv2d(128, 256, 3, 1, 1, bias=False), nn.BatchNorm2d(256),
                                          nn.LeakyReLU(0.1, inplace=True), nn.MaxPool2d(2, 2))
        self.stage1_conv9 = nn.Sequential(nn.Conv2d(256, 512, 3, 1, 1, bias=False), nn.BatchNorm2d(512),
                                          nn.LeakyReLU(0.1, inplace=True))
        self.stage1_conv10 = nn.Sequential(nn.Conv2d(512, 256, 1, 1, 0, bias=False), nn.BatchNorm2d(256),
                                           nn.LeakyReLU(0.1, inplace=True))
        self.stage1_conv11 = nn.Sequential(nn.Conv2d(256, 512, 3, 1, 1, bias=False), nn.BatchNorm2d(512),
                                           nn.LeakyReLU(0.1, inplace=True))
        self.stage1_conv12 = nn.Sequential(nn.Conv2d(512, 256, 1, 1, 0, bias=False), nn.BatchNorm2d(256),
                                           nn.LeakyReLU(0.1, inplace=True))
        self.stage1_conv13 = nn.Sequential(nn.Conv2d(256, 512, 3, 1, 1, bias=False), nn.BatchNorm2d(512),
                                           nn.LeakyReLU(0.1, inplace=True))
        #产生分支，此处输出为输出 0
```

```
        self.stage2_a_maxpl = nn.MaxPool2d(2, 2)
        self.stage2_a_conv1 = nn.Sequential(nn.Conv2d(512, 1024, 3, 1, 1, bias=False),
                                            nn.BatchNorm2d(1024), nn.LeakyReLU(0.1,
inplace=True))
        self.stage2_a_conv2 = nn.Sequential(nn.Conv2d(1024, 512, 1, 1, 0, bias=False),
nn.BatchNorm2d(512),
                                            nn.LeakyReLU(0.1, inplace=True))
        self.stage2_a_conv3 = nn.Sequential(nn.Conv2d(512, 1024, 3, 1, 1, bias=False),
nn.BatchNorm2d(1024),
                                            nn.LeakyReLU(0.1, inplace=True))
        self.stage2_a_conv4 = nn.Sequential(nn.Conv2d(1024, 512, 1, 1, 0, bias=False),
nn.BatchNorm2d(512),
                                            nn.LeakyReLU(0.1, inplace=True))
        self.stage2_a_conv5 = nn.Sequential(nn.Conv2d(512, 1024, 3, 1, 1, bias=False),
nn.BatchNorm2d(1024),
                                            nn.LeakyReLU(0.1, inplace=True))
        self.stage2_a_conv6 = nn.Sequential(nn.Conv2d(1024, 1024, 3, 1, 1, bias=False),
nn.BatchNorm2d(1024),
                                            nn.LeakyReLU(0.1, inplace=True))
        self.stage2_a_conv7 = nn.Sequential(nn.Conv2d(1024, 1024, 3, 1, 1, bias=False),
nn.BatchNorm2d(1024),
                                            nn.LeakyReLU(0.1, inplace=True))
        #此处为输出 1
        self.stage2_b_conv = nn.Sequential(nn.Conv2d(512, 64, 1, 1, 0, bias=False),
nn.BatchNorm2d(64),
                                           nn.LeakyReLU(0.1, inplace=True))
        #此处为输出 2
        self.stage3_conv1 = nn.Sequential(nn.Conv2d(256 + 1024, 1024, 3, 1, 1, bias=False),
nn.BatchNorm2d(1024),
                                          nn.LeakyReLU(0.1, inplace=True))
        #融合输出 1 和输出 2，得到输出
        self.new_stage3_conv2 = nn.Conv2d(1024, len(self.anchors) * (5 + num_classes), 1, 1, 0,
bias=False)
        #用 1*1 卷积代替全连接
    def forward(self, input):


        output = self.stage1_conv1(input)
        output = self.stage1_conv2(output)
        output = self.stage1_conv3(output)
        output = self.stage1_conv4(output)
        output = self.stage1_conv5(output)
        output = self.stage1_conv6(output)
        output = self.stage1_conv7(output)
        output = self.stage1_conv8(output)
        output = self.stage1_conv9(output)
        output = self.stage1_conv10(output)
        output = self.stage1_conv11(output)
        output = self.stage1_conv12(output)
        output = self.stage1_conv13(output)

        residual = output#网络中的分支

        output_1 = self.stage2_a_maxpl(output)
        output_1 = self.stage2_a_conv1(output_1)
        output_1 = self.stage2_a_conv2(output_1)
        output_1 = self.stage2_a_conv3(output_1)
```

```
        output_1 = self.stage2_a_conv4(output_1)
        output_1 = self.stage2_a_conv5(output_1)
        output_1 = self.stage2_a_conv6(output_1)

        output_1 = self.stage2_a_conv7(output_1)
        output_2 = self.stage2_b_conv(residual)

        batch_size, num_channel, height, width = output_2.data.size()
        #对输出 2 进行变换
        output_2 = output_2.view(batch_size, int(num_channel / 4), height, 2, width, 2).contiguous()
        output_2 = output_2.permute(0, 3, 5, 1, 2, 4).contiguous()
        output_2 = output_2.view(batch_size, -1, int(height / 2), int(width / 2))
        #连接输出 1 和输出 2，通过卷积层得到最后输出
        output = torch.cat((output_1, output_2), 1)
        output = self.stage3_conv1(output)
        output = self.new_stage3_conv2(output)
        return output
```

改进 mgn 网络搭建：

```
class MGN(nn.Module):
    def __init__(self, args):
        super(MGN, self).__init__()
        num_classes = args.num_classes

        resnet = resnet50(pretrained=True)

        self.backone = nn.Sequential(
            resnet.conv1,
            resnet.bn1,
            resnet.relu,
            resnet.maxpool,
            resnet.layer1,
            resnet.layer2,
            resnet.layer3[0],
        )
        #使用 resnet 作为前 4 层
        res_conv4 = nn.Sequential(*resnet.layer3[1:])
        #定义分支网络
        res_g_conv5 = resnet.layer4

        res_p_conv5 = nn.Sequential(
            Bottleneck(1024, 512, downsample=nn.Sequential(nn.Conv2d(1024, 2048, 1, bias=False),
nn.BatchNorm2d(2048))),
            Bottleneck(2048, 512),
            Bottleneck(2048, 512))
        res_p_conv5.load_state_dict(resnet.layer4.state_dict())

        self.p1 = nn.Sequential(copy.deepcopy(res_conv4), copy.deepcopy(res_g_conv5))
        self.p2 = nn.Sequential(copy.deepcopy(res_conv4), copy.deepcopy(res_p_conv5))
        self.p3 = nn.Sequential(copy.deepcopy(res_conv4), copy.deepcopy(res_p_conv5))

        if args.pool == 'max':
            pool2d = nn.MaxPool2d
        elif args.pool == 'avg':
            pool2d = nn.AvgPool2d
        else:
            raise Exception()
```

```
        #global maxpool
        self.maxpool_zg_p1 = pool2d(kernel_size=(12, 4))
        self.maxpool_zg_p2 = pool2d(kernel_size=(24, 8))
        self.maxpool_zg_p3 = pool2d(kernel_size=(24, 8))
        self.maxpool_zp2 = pool2d(kernel_size=(12, 8))
        self.maxpool_zp3 = pool2d(kernel_size=(8, 8))
        #降维
        reduction = nn.Sequential(nn.Conv2d(2048, args.feats, 1, bias=False),
nn.BatchNorm2d(args.feats), nn.ReLU())
        self.red = nn.Sequential(nn.Linear(3072, 256), nn.BatchNorm1d(256), nn.ReLU())


        self._init_reduction(reduction)
        self.reduction_0 = copy.deepcopy(reduction)
        self.reduction_1 = copy.deepcopy(reduction)
        self.reduction_2 = copy.deepcopy(reduction)
        self.reduction_3 = copy.deepcopy(reduction)
        self.reduction_4 = copy.deepcopy(reduction)
        self.reduction_5 = copy.deepcopy(reduction)
        self.reduction_6 = copy.deepcopy(reduction)
        self.reduction_7 = copy.deepcopy(reduction)
        #全连接层，进行 softmax
        #self.fc_id_2048_0 = nn.Linear(2048, num_classes)
        self.fc_id_2048_0 = nn.Linear(args.feats, num_classes)
        self.fc_id_2048_1 = nn.Linear(args.feats, num_classes)
        self.fc_id_2048_2 = nn.Linear(args.feats, num_classes)

        self.fc_id_256_1_0 = nn.Linear(args.feats, num_classes)
        self.fc_id_256_1_1 = nn.Linear(args.feats, num_classes)
        self.fc_id_256_2_0 = nn.Linear(args.feats, num_classes)
        self.fc_id_256_2_1 = nn.Linear(args.feats, num_classes)
        self.fc_id_256_2_2 = nn.Linear(args.feats, num_classes)
        self.fc_fuse = nn.Sequential(nn.Linear(4096, 1024), nn.BatchNorm1d(1024), nn.ReLU())

        self._init_fc(self.fc_id_2048_0)
        self._init_fc(self.fc_id_2048_1)
        self._init_fc(self.fc_id_2048_2)

        self._init_fc(self.fc_id_256_1_0)
        self._init_fc(self.fc_id_256_1_1)
        self._init_fc(self.fc_id_256_2_0)
        self._init_fc(self.fc_id_256_2_1)
        self._init_fc(self.fc_id_256_2_2)

    @staticmethod
    def _init_reduction(reduction):
        # conv
        nn.init.kaiming_normal_(reduction[0].weight, mode='fan_in')
        #nn.init.constant_(reduction[0].bias, 0.)

        # bn
        nn.init.normal_(reduction[1].weight, mean=1., std=0.02)
        nn.init.constant_(reduction[1].bias, 0.)

    @staticmethod
    def _init_fc(fc):
        nn.init.kaiming_normal_(fc.weight, mode='fan_out')
```

```
        #nn.init.normal_(fc.weight, std=0.001)
        nn.init.constant_(fc.bias, 0.)

    def forward(self, x):

        x = self.backone(x)

        p1 = self.p1(x)
        p2 = self.p2(x)
        p3 = self.p3(x)
        #global maxpool 输出
        zg_p1 = self.maxpool_zg_p1(p1)
        zg_p2 = self.maxpool_zg_p2(p2)
        zg_p3 = self.maxpool_zg_p3(p3)

        zp2 = self.maxpool_zp2(p2)
        z0_p2 = zp2[:, :, 0:1, :]
        z1_p2 = zp2[:, :, 1:2, :]

        zp3 = self.maxpool_zp3(p3)
        z0_p3 = zp3[:, :, 0:1, :]
        z1_p3 = zp3[:, :, 1:2, :]
        z2_p3 = zp3[:, :, 2:3, :]
        #  降维输出
        fg_p1 = self.reduction_0(zg_p1).squeeze(dim=3).squeeze(dim=2)
        fg_p2 = self.reduction_1(zg_p2).squeeze(dim=3).squeeze(dim=2)
        fg_p3 = self.reduction_2(zg_p3).squeeze(dim=3).squeeze(dim=2)
        f0_p2 = self.reduction_3(z0_p2).squeeze(dim=3).squeeze(dim=2)
        f1_p2 = self.reduction_4(z1_p2).squeeze(dim=3).squeeze(dim=2)
        f0_p3 = self.reduction_5(z0_p3).squeeze(dim=3).squeeze(dim=2)
        f1_p3 = self.reduction_6(z1_p3).squeeze(dim=3).squeeze(dim=2)
        f2_p3 = self.reduction_7(z2_p3).squeeze(dim=3).squeeze(dim=2)

        '''
        l_p1 = self.fc_id_2048_0(zg_p1.squeeze(dim=3).squeeze(dim=2))
        l_p2 = self.fc_id_2048_1(zg_p2.squeeze(dim=3).squeeze(dim=2))
        l_p3 = self.fc_id_2048_2(zg_p3.squeeze(dim=3).squeeze(dim=2))
        '''
        #softmax 输出
        l_p1 = self.fc_id_2048_0(fg_p1)
        l_p2 = self.fc_id_2048_1(fg_p2)
        l_p3 = self.fc_id_2048_2(fg_p3)

        l0_p2 = self.fc_id_256_1_0(f0_p2)
        l1_p2 = self.fc_id_256_1_1(f1_p2)
        l0_p3 = self.fc_id_256_2_0(f0_p3)
        l1_p3 = self.fc_id_256_2_1(f1_p3)
        l2_p3 = self.fc_id_256_2_2(f2_p3)
        #添加一个全局特征 fg_p4，输出进行 triplet
        midfeat = torch.cat((fg_p1, fg_p2), dim=1)# shape:[1,4096]
        midfeat = self.fc_fuse(midfeat)    # shape:[1,1024]
        combofeat = torch.cat((x5c_feat, midfeat), dim=1)    # shape：[1,3072]
        fg_p4=self.red(combofeat)#256
        predict = torch.cat([fg_p1, fg_p2, fg_p3, fg_p4, f0_p2, f1_p2, f0_p3, f1_p3, f2_p3], dim=1)

        return predict, fg_p1, fg_p2, fg_p3, fg_p4, l_p1, l_p2, l_p3, l0_p2, l1_p2, l0_p3, l1_p3, l2_p3
```

动作识别网络搭建：

```
class ResNet50_Phone_BOT(nn.Module):
    def __init__(self, classes = 2, loss={'xent'}, **kwargs):
        super(ResNet50_Phone_BOT, self).__init__()
        self.loss = loss
        resnet50 = torchvision.models.resnet50(pretrained=True)
        self.base = nn.Sequential(*list(resnet50.children())[:-2])   # 去掉最后两层，fc 和 pooling
        self.phone_classifier = nn.Linear(2048, classes)#添加全连接层
        self.feat_dim = 2048    # feature dimension

    def forward(self, x):
        x = self.base(x)
        x = F.avg_pool2d(x, x.size()[2:])
        f = x.view(x.size(0), -1)
        phone = self.phone_classifier(f)#softmax 调用

        return phone
```

致 谢

大学四年真就印证了一句从小就学会的谚语，光阴似箭，日月如梭。回首这段时光，所有得到与失去，留下来的都是自己的经历。在这告别之际，我想对陪伴我走过这美好四年的师长和亲友表达诚挚的谢意。

首先，我要感谢我的母校浙江工业大学。她提供给了我一个优越的学习环境，使我可以安心沉浸于研究当中，没有收到外界的干扰。

然后，我要感谢我的指导教师赵云波教授。赵老师严谨的治学态度深深吸引着我。从最初的选题，初期答辩，到后面的中期答辩，毕业论文的撰写，赵老师给予了我很大的帮助。每次遇到瓶颈的时候，与赵老师的交流总能给我一种豁然开朗的感觉，明确了前进的方面。没有赵老师的指导，我很难完成毕业设计工作和毕业论文的撰写。

最后，我想特别感谢的是我的父母。是你们提供了我优异的学习条件，使我不用担心除学习以外的事情。每次与你们的交流都会成为我继续前进的动力。你们总要我不用担心你们，要好好照顾自己，不要熬夜。这些我都知道，我也会好好加油的，再次感谢我的父母。

马上就要步入社会，面对未知生活带来的压力，我内心中涌出的是不惧挑战的勇气。在未来一定会遇到更多的困难和失败，但只要坚持下去，总能找到自己的道路！