

圣彼得堡国立理工大学

学士学位论文

论文题目：

《使用神经网络进行商店客流量统计》

**ПРИМЕНЕНИЕ НЕЙТРОННЫХ СЕТЕЙ ДЛЯ ОЦЕНКИ
ТРАФИКА МАГАЗИНА**

作者：李逸嘉

摘要

图片，表格，页数

关键词：计算机视觉，神经网络，目标检测，多目标跟踪，行人识别

毕业论文题目：《神经网络在店铺客流量评估中的应用》。

本文致力于实现用于统计商店客流量的神经网络算法。为了解决任务，使用了深度学习和计算机视觉方法。

本工作要解决的任务：

1. 实现 YOLOv5 算法用于检测行人
2. 为了在多个摄像头下建立行人身份之间的关联，实现了行人重识别算法 (ReID)。
3. 在行人重识别的基础上实现了 DeepSORT 算法，用于在视频流上对行人跟踪
4. 基于上述方法，开发了一个超市访客统计系统。

作为研究的结果，我们开发了超市的顾客计数系统，其识别人员进出的精确度达到 95.5%，行人重识别的准确度达到 81.4%。最后在其他大型公开数据集上，我们将 DeepSORT 实现、FastReID 与其他主流算法做了比较，均取得了不错的成绩。

Реферат

На - с., - рисунков, - таблицы, - приложение

KEY WORDS: Computer vision, Neural networks, Object detection, Multiple object tracking, Pedestrian re-identification)

КЛЮЧЕВЫЕ СЛОВА: КОМПЬЮТЕРНОЕ ЗРЕНИЕ, НЕЙРОННЫЕ СЕТИ, ОБНАРУЖЕНИЕ ОБЪЕКТОВ, ОТСЛЕЖИВАНИЕ НЕСКОЛЬКИХ ОБЪЕКТОВ, ПЕРЕИДЕНТИФИКАЦИЯ ПЕШЕХОДОВ

Выпускная квалификационная работа на тему: «Применение нейронных сетей для оценки трафика магазина».

Данная статья посвящена реализации алгоритмов нейронных сетей для оценки трафика в магазине. Для решения поставленных задач были использованы методы глубокого обучения и компьютерного зрения.

Задачи, решаемые в ВКР:

1. Для обнаружения пешеходов был использован алгоритм YOLOv5
2. Для установления связи между персонажами при работе с несколькими камерами был реализован алгоритм повторного распознавания (ReID).
3. Для отслеживания локальных зон был реализован алгоритм DeepSORT он основе повторного распознавания.
4. На основе методики, описанных в данной статье, была разработана система подсчета посетителей для супермаркета.

В результате проведенных исследований была разработана система подсчета посетителей для супермаркета, которая демонстрирует точность 81.4%.

Abstract

目录

摘要.....	2
Реферат.....	3
绪论（Введение）	6
ГЛАВА 1 Обзор литературы （文件综述）	7
目标检测.....	7
基于传统方法的目标检测.....	7
基于深度学习的目标检测.....	8
YOLOv5 目标检测算法.....	8
多目标跟踪.....	9
基于目标检测的多目标跟踪 (Tracking By Detection, TBD).....	10
联合检测和跟踪的多目标跟踪（Join Detection and Tracking, JDT）	11
DeepSORT 目标跟踪算法	11
行人重识别.....	12
基于表征学习(Representation learning)的方法	14
基于度量学习 (Metric learning)的方法.....	15
Fast ReID 算法	17
人流量统计.....	19
本文的主要研究内容.....	19
ГЛАВА 2 Обучение модели для отслеживания людей в нескольких видеопотоках （训练模型）	20
ReID 模型的训练.....	20
数据集准备.....	20
训练原理.....	21
构建模型.....	21
损失函数和优化算法.....	22
训练步骤.....	22
训练过程.....	23
训练结果.....	23

基于 FastReID+YOLOv5 搭建 DeepSORT 模型	26
构建模型	26
测试数据集 MOT-16	27
实验 1：对比测试经过我们改进后的 DeepSORT 算法与原 DeepSORT 算法	28
实验 2：改进算法与几种主流算法对比	30
ГЛАВА 3 Система оценки трафика магазина основа на нейронных сетей （商店客流量统计系统）	31
项目背景	31
商店客流量统计系统简介	31
系统整体架构	31
开发环境	33
商店客流量统计系统展示与测试	34
客流量统计算法	34
测试环境	34
跨摄像头行人跟踪功能	35
准确性、效率、稳定性	39
ГЛАВА 4 Анализ полученных результатов （结果分析）	45
ReID 模型在其他数据集上的测试结果	45
在 Market1501、DukeMTMC 数据集上的测试结果	45
DeepSORT 模型的评价指标	46
在 MOT20 数据集上的测试结果	46
ЗАКЛЮЧЕНИЕ （结论）	47
参考文献	48

绪论（Введение）

（研究的主要背景）

随着零售业的发展和竞争加剧，目前国内很多的商场已经意识到了客流数据对于商业决策的重要性。客流量是指单位时间进出商场的顾客人数。通过对客流量的深层分析，可以对商场方方面面的管理提供科学依据。客流量分析对于提高商场日常经营决策的科学性、购物环境舒适度、人力资源调配的合理性等方面起着重要的作用。

（现有研究的不足）

最早出现的是人工统计方法，商场派若干员工在出入口，在一段时间内，连续地对进入商场的顾客进行目视计数。该方法的优点在于不需要进行信息采集设备的投资，调查资料比较全面、灵活，但缺点是耗费人力、物力，并且容易出现漏记的情况。

随着机械触碰设备、红外传感设备和监控摄像头设备的出现，人流量统计方法依次经历了物理触碰统计和红外统计两个阶段。物理触碰统计多用在进出口，行人依次通过，但有设备安装要求较高、使用起来不方便、通行速度慢以及不能大范围使用等弊端；红外统计安装要求低、使用相对方便，但其统计误差大、不能用于大区域场景；除此之外，这些方法都有非常明显的缺点：无法知道每个顾客出入商店的时间。用这种方法得到的数据，只可以用于定性的了解，没有实际的统计分析价值。在经济竞争如此激烈的今天，这些统计方法已完全不满足市场的需求。

传统的人工检测方法效率低下、成本高昂，因此需要一种更加高效、准确且自动化的方法来满足快速变化的市场需求。基于此，我们开展了本研究，旨在实现利用神经网络方法对超市客流量进行监控与计算的算法。

本文的主要目的是提出一种结合了行人检测、行人跟踪和行人重识别算法的方法，以实现商店客流量的统计。我们通过实现YOLOv5 算法、DeepSORT跟踪算法和行人重识别算法，完成了对不同角度和光照条件下的顾客出入商店准确的跟踪与计数，并开发了一个面向超市的客流量统计系统。

在可行性分析方面，我们认为本文提出的方法具有较高的可行性。首先，所使用的算法均为当前深度学习领域内较为成熟和广泛应用的技术，已经得到了充分实验验证。其次，在硬件和软件环境上，本文所需的配置相对较低，能够在普通 PC 上运行。最后，我们还对所提出方法的准确性、效率和稳定性进行了全面的实验评估，证明了其可行性。

为了实现本文目标，我们需要完成以下任务：

- 1) 采集超市摄像头监控画面，并进行预处理；
- 2) 设计并实现基于 YOLOv5 算法的行人检测模块；
- 3) 设计并实现基于行人重识别算法的 DeepSORT 跟踪算法，应用于视频流中行人的跟踪；
- 4) 设计并实现行人重识别算法以进行多摄像头下指定顾客的人物关联；
- 5) 存储记录出入顾客的照片和出入时间，并进行客流量统计。

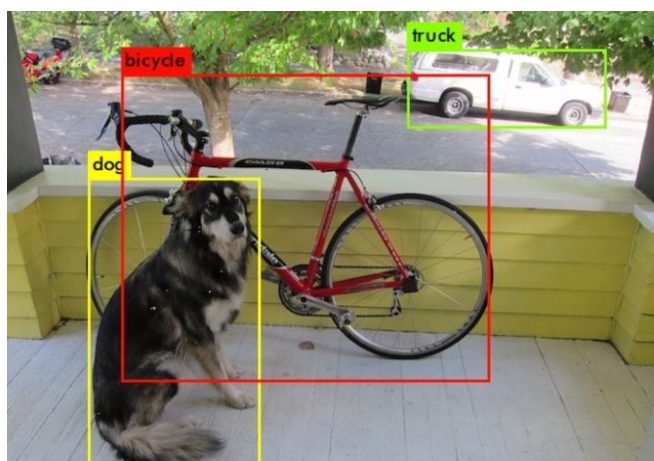
在此基础上，我们开发了一个实用的面向超市的客流量统计系统，可为超市提供有效的管理和决策支持。

ГЛАВА 1 Обзор литературы （文件综述）

人流量统计是一个集目标检测、多目标跟踪、行人重识别技术于一体的综合性任务。鉴于此，本文首先叙述目标检测和多目标跟踪的研究现状，然后介绍行人重识别算法，最后介绍人流量统计的研究现状。行人重识别是解决跨摄像头轨迹关联问题的核心技术，也是本研究工作的核心。所以在我们将会重点地、详细地介绍我们实现的行人重识别算法：FastReID。

目标检测

目标检测是多目标跟踪任务的基础，其主要目的是把需要被检测的目标从每一帧的静态图片中用矩形框标记出来，得到目标在图片中的具体位置。如果目标检测中出现了漏检，那么后期也就不会对目标进行跟踪，所以目标检测在多目标跟踪任务中起着非常关键的作用。纵观目标检测的整个发展过程，可将其分为基于传统方法的目标检测和基于深度学习的目标检测。



基于传统方法的目标检测

在深度学习发展之前，传统的目标检测算法通常遵循基于手工特征设计的思想，此类方法大致包括三个步骤，分别为区域选择、特征提取和分类器分类。该时期主流的检测算法包括 Viola-Jone 检测器^[2]、HOG 检测器^[3]和变形部件模型^[4]（Deformable Parts Model, DPM）。Viola 等人提出的 Viola-Jone 检测器采用积分图像、特制选择和检测级联三种有效的策略，并通过从左到右、从上到下的滑动方式，判断是否有窗口包含目标，该检测器的提出使得目标检测的速度大大提高。HOG 检测器最初由 Dalal 等人在 2005 年提出，其核心思想是局部物体形状和外观能够被局部梯度或边缘的密度分布所描述，利用方向梯度直方图特征描述子来构建特征，并与 SVM 分类器结合，有效地解决了行人检测任务。另外，DPM 检测器由 Pedro 等人在 2008 年提出，遵循“分而治之”的理论对目标进行检测，通过检测目标包含的各个部件从而完成对目标整体的检测。

基于深度学习的目标检测

随着传统目标检测算法的性能趋于饱和,目标检测的研究也进入了一个平稳的发展期,直到卷积神经网络的出现,使得目标检测技术的性能获得了显著的提高。此类算法大致可分为以下两类,一类是两阶段检测算法,先在图像中选择候选区域,然后再完成这些区域的分类任务,该类算法的特点是精度高但速度慢。如Ross 等人^[5]所提出的 R-CNN,首先以选择性搜索操作预先定位图片中的待检测区域,然后通过卷积神经网络提取这些区域上的特征,最后进行分类。虽然 R- CNN 算法大大提高了目标检测精度,但其存在计算冗余的问题。He 等人^[6]提出的 SPPNet 算法,通过加入空间金字塔池化层解决了 R-CNN 反复缩放图像造成的影响。此外, Girshick 等人^[7]提出 Fast R-CNN 算法,通过加入区域池化层进一步优化了 R-CNN 与 SPPNet 算法。Ren 等人^[8]提出 Faster R-CNN 算法,将选择性搜索用区域提案网络代替,对网络进行端到端地训练,极大地提高了检测器的速度。另一类是一阶段检测算法,在产生候选区域的同时进行回归输出检测对象的位置和类别,其代表性算法就是 YOLO 系列算法^[9]和 SSD 算法^[10]。YOLO 系列算法首先由 Redmon 等人于 2016 年提出,其抛弃了“提案检测+验证”的两阶段检测范式,通过将图像分割成多个网格,直接预测每个网格所属的类别概率和位置信息。针对 YOLO 算法检测精度低的缺点,Liu 等人提出了 SSD 算法,引入了多尺度特征图检测技术,使得检测器的精度得到了极大提高。

从上面的介绍我们得出结论:传统方法的目标检测通常需要手动设计特征和分类器,这些方法在处理复杂的场景时表现不佳,并且需要大量的专业知识和时间来调整参数。相比之下,基于深度学习的目标检测技术利用深度神经网络自动学习特征和分类器,具有更高的准确性和鲁棒性,适用于各种复杂场景。目标检测是目标跟踪的前提,我们选择 YOLOv5 作为研究项目的目标检测器,主要原因是 YOLOv5 具有高精度,高效率,简单易用,而且可定制性强。

综合以上优点, YOLOv5 比较适合我们的使用场景。

YOLOv5 目标检测算法

【韩晓冰】

YOLOv5 是由作者 Glenn Jocher 提出,一共有四种大小模型,分别为 YOLOv5s、YOLOv5m、YOLOv5l、YOLOv5x。其中 YOLOv5s 的卷积层数最少、检测速率最快、检测精度最低;另外的 3 个模型,卷积层数从小到大依次排列,随着模型复杂度增大检测速率逐渐降低,检测精度逐渐提高。

YOLOv5 结构包括四部分: Input、Backbones、Neck、Prediction, 如图 1 所示

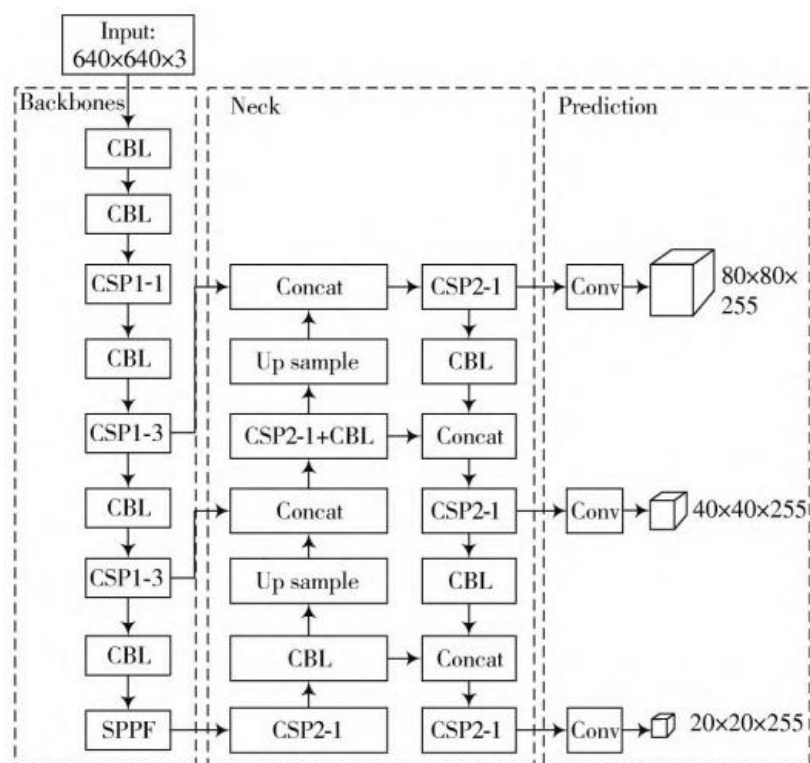


图1 YOLOv5网络总结构图

Input 端包含 Mosaic 增强、动态锚框、自适应图片处理等；Backbones 包括跨阶段局部网络 CSP、空间金字塔池化 SPPF，前者有助于减少计算量，后者提高检测精确度；Neck 采用了 FPN+PAN 的结构，下采样增强语义信息，上采样增强定位信息；Prediction 是最后检测输出端。

为了追求更快的速度，我们选用 YOLOv5 开源网站【注释】发布的预训练好的 yolov5s 模型。

多目标跟踪

多目标跟踪的主要目的是对视频中的多个目标同时进行轨迹跟踪，并维持对应的身份信息。随着近年来多目标跟踪研究的不断增多，其相关方法层出不穷，当前主流的方法可分为以下两类：基于目标检测的多目标跟踪和联合检测和跟踪的多目标跟踪。



基于目标检测的多目标跟踪 (Tracking By Detection, TBD)

在基于目标检测的多目标跟踪算法中，目标检测和特征提取被分为两步，首先对当前帧的图像执行目标检测，然后提取检测框的特征表示，最后将检测框与已有的运动轨迹进行关联。在 2016 年，Alex 等人^[11]提出了一种在线和实时跟踪方法（Simple Online and Realtime Tracking, SORT）利用卡尔曼滤波预测每条跟踪轨迹在当前帧中的运动状态，将预测值与当前帧中候选检测框的交并比作为相似度矩阵得分，最后通过匈牙利算法解决轨迹和候选检测框间的关联任务。然而 SORT 在数据关联时只使用目标的运动信息，因此在遇到频繁遮挡时容易出现身份信息交换的情况。Wojke 等人^[12]提出的 DeepSORT 算法解决了该问题，在 DeepSORT 中使用结合运动和外观信息的联合度量来完成数据关联，大大减少了行人身份信息变化的次数，还引入级联匹配解决遮挡导致的概率弥散问题。另外，为解决检测器漏检、误检等问题，Chen 等人^[13]设计了一个基于卷积神经网络的评分函数，利用该函数从检测器和跟踪器的输出中生成最优的候选框，进一步拓展了基于检测的多目标跟踪算法。

联合检测和跟踪的多目标跟踪（Join Detection and Tracking, JDT）

联合检测和跟踪的多目标跟踪算法采用了多任务联合学习的思想，该类算法通常训练一个可以同时输出目标检测框位置和外观特征的模型，提高了多目标跟踪的速度。Wang 等人^[14]提出将行人重识别模型合并到 YOLOv3 检测器中，并设计一个联合检测框回归、anchor 分类以及嵌入学习的损失函数完成同时学习外观特征表示和目标检测的任务。作为第一种联合检测和特征提取的方法，该算法存在很多不足，比如外观信息中缺少多层特征的融合、基于锚框的检测器不适合提取目标外观特征等问题。为了解决这些问题，Zhang 等人^[15]提出了 FairMOT，使用深层聚合网络来实现不同尺度的检测，以便提取多层次的行人重识别信息，同时使用了基于关键点的检测方式，缓解了目标中心偏移问题。此外，研究者们还提出将数据关联也统一到单个网络中，如 Zhou 等人^[16]提出的 CenterTrack，通过将检测模型应用于相邻图像帧，结合前一帧的目标检测结果来预测当前帧的目标运动状态并完成数据关联，巧妙地将数据关联问题转为相邻帧之间的检测框回归问题。Peng 等人^[17]提出了链式跟踪算法（Chained-tracker），业内首创统一目标检测、特征提取、数据关联三个任务的端到端多目标跟踪网络，并设计了联合注意力模块，相较于其他算法集成度更高。

综上所述，联合检测和跟踪（JDT）和基于目标检测的多目标跟踪（TBD）都是针对视频中出现的多个目标进行跟踪的算法。其中，JDT 算法将目标检测和跟踪过程结合在一起，先通过目标检测得到目标的位置信息，再在下一帧图像中通过匹配和预测得到目标的轨迹；而 TBD 算法则是首先利用目标检测算法检测出所有目标，然后利用某些模型或者算法进行轨迹匹配和更新。相比较而言，JDT 算法更加高效准确，但是在目标密度较大时，算法的计算量会变得非常大，从而降低了跟踪的精度。考虑到零售场景会出现人员密集的情况，我们在研究项目中使用 DeepSORT 作为目标跟踪算法，它能有效解决人员密集的问题。DeepSORT 使用卷积神经网络来提取目标的特征，然后利用卡尔曼滤波和匈牙利算法来进行多个目标的跟踪，并且具有较高的准确性和鲁棒性。此外，DeepSORT 还具有较好的扩展性和可调节性，可以根据不同的应用场景进行优化和改进。

DeepSORT 目标跟踪算法

【郑繁亭】 【韩晓冰】

DeepSORT 是对 SORT 追踪算法的改进，为了降低跟踪目标身份编号切换（ID switch）次数，DeepSORT 引入了外观特征和级联匹配。DeepSORT 的主要目的是生成视频流中多个人物运动的连续轨迹，其总工作流程如下图。DeepSORT 算法以目标检测网络检测结果的检

测框坐标和置信度作为输入，通过卡尔曼滤波算法对下一帧行人的位置进行预测，再进行级联匹配，然后利用匈牙利算法进行数据关联，最后进行卡尔曼滤波更新。

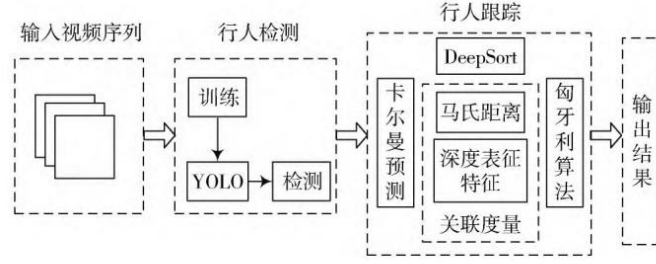


图2 行人检测总流程图

DeepSORT 计算相似度利用了目标的运动信息和外观信息。对于运动信息，用马氏距离判断预测目标与检测目标的关联度。马氏距离表达式为：

$$d^{(1)}(i, j) = (d_j - y_i)^T S_i^{-1} (d_j - y_i)$$

式中， d_j 是检测框 j 的位置； y_i 是跟踪器 i 预测的位置； S_i 是检测与预测位置的协方差矩阵。

当目标长久遮挡或视角抖动，则要引入外观信息，通过余弦距离来解决因遮挡带来身份切换的问题。

余弦距离表达式为：

$$d^{(2)}(i, j) = \min \{1 - r_j^T r_k^{(i)} \mid r_k^{(i)} \in R_i\}$$

式中， r_j 是检测框 d_j 的本征向量； $r_k^{(i)}$ 是跟踪器 i 对应最近 100 帧本征向量的集合； R_i 为外观特征向量库。

为了充分利用两种信息，采用线性加权的方式求和：

$$c_{i,j} = \lambda d^{(1)}(i, j) + (1 - \lambda) d^{(2)}(i, j)$$

式中： λ 为超参数，当且仅当度量值 $c_{i,j}$ 存在于 $d^{(1)}(i, j)$ 和 $d^{(2)}(i, j)$ 之间，才认为目标关联。

值得注意的是，由于卡尔曼滤波算法是基于线性匀速运动的，所以 DeepSORT 算法只能对线性环境下的目标进行状态预测，在行人轨迹非线性时也许不能很好地进行预测。

行人重识别

【来源：<https://zhuanlan.zhihu.com/p/456060221>】

行人重识别（Person Re-identification 也称行人再识别，简称为 ReID，是近几年智能视频分析领域兴起的一项新技术。它旨在解决这样的问题：给定一个监控行人图像，检索多个视频中的该行人图像。在监控视频中，由于相机分辨率和拍摄角度的缘故，通常无法得到质量非常高的人脸图片。当人脸识别失效的情况下，ReID 就成为了一个非常重要的替代品技术。ReID 有一个非常重要的特性就是跨摄像头，所以学术论文里评价性能的时候，是要检索出不同摄像头下的相同行人图片。

以下介绍 3 个最常用到的行人重识别概念：



probe: 待检索的人物图像

Gallery: 查找的图像集合

Query: 所有 probe 组成的集合

行人重识别任务主要包含特征提取和相似度度量两个步骤。特征提取：学习能够应对在不同摄像头下行人变化的特征。度量学习：将学习到的特征映射到高维空间，使相同的人更近不同的人更远。

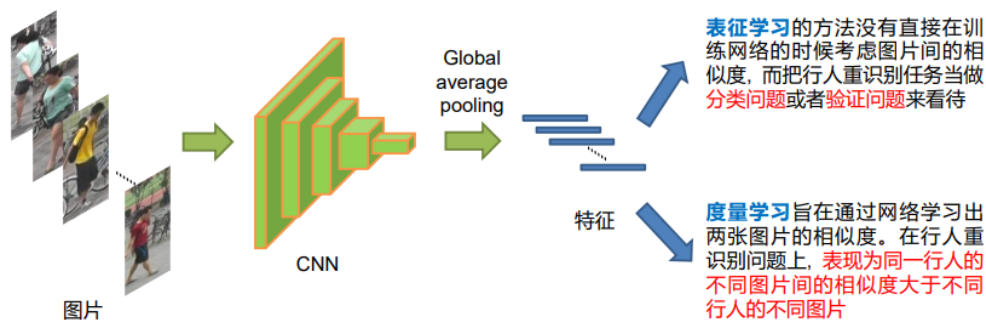
传统的方法思路为手工提取图像特征，例如：颜色、HOG (Histogram of oriented gradient)[4]、SIFT (Scale invariant feature transform)[5]、LOMO (Local maximal occurrence) 等。之后，利用 XQDA (Cross-view quadratic discriminant analysis)[6] 或者 KISSME (Keep it simple and straightforward metric learning)[7] 来学习最佳的相似度度量。然而，传统的手工特征描述能力有限，很难适应复杂场景下的大数据量任务。并且，在数据量较大的情形下，传统的度量学习方法求解也会变得非常困难。

近年来，以卷积神经网络为代表的深度学习在计算机视觉领域取得了极大的成功，在多项任务上都击败传统的方法，甚至一定程度上超越了人类的水平[8-9]。在行人重识别问题上，基于深度学习的方法可以自动学习出复杂的特征描述，并且用简单的欧氏距离进行相似度度量便可以取得很好的性能。换句话说，深度学习可以端对端地实现行人重识别任务，这使得任务变得更加简单。目前，基于深度学习的行人重识别方法已经在性能上大大超越了传统的方法。这些优势使得深度学习在行人重识别领域变得流行。

【来源：罗浩-基于深度学习的行人重识别研究进展】

上述基于深度学习的行人重识别方法根据训练损失可以分为基于表征学习和度量学习，在这里我们介绍这两种不同的学习方法。

根据损失分类

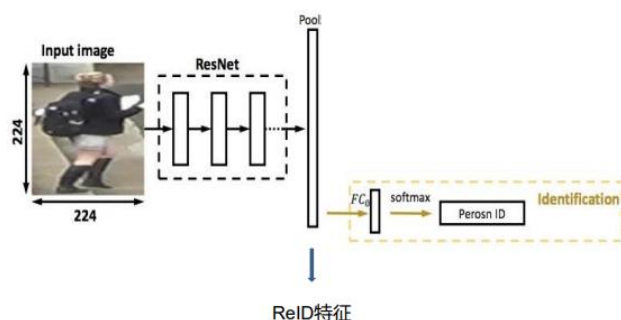


基于表征学习(Representation learning)的方法

基于表征学习 (Representation learning) 的方法是一类非常常用的行人重识别方法。虽然行人重识别的最终目标是为了学习出两张图片之间的相似度, 但是表征学习的方法并没有直接在训练网络的时候考虑图片间的相似度, 而把行人重识别任务当做分类 (Classification) 问题来看待。这类方法的特点就是网络的最后一层全连接 (Fully connected, FC) 层输出的并不是最终使用的图像特征向量, 而是经过一个 Softmax 激活函数来计算表征学习损失, 前一层 (倒数第二层) FC 层通常为特征向量层。具体言之, 分类问题是指利用行人的 ID 或者属性 (Attribute) 作为训练标签来训练模型, 每次只需要输入一张图片。

分类网络常用的两种损失分别是行人 ID 损失 (Identification loss) 和属性损失 (Attribute loss). 文献 [3, 29] 将每一个行人当做分类问题的一个类别, 用行人的 ID 作为训练数据的标签来训练 CNN 网络, 训练集中行人的 ID 数就是网络的类别数。在特征层后接一个分类 FC, 经过 softmax 激活函数计算交叉熵损失。这个网络损失被称为 ID 损失。而在测试阶段, 分类 FC 层将会被丢弃, 使用倒数第二层的特征向量进行检索。这种网络被称为 IDE (ID embedding) 网络。IDE 网络是行人重识别领域非常重要的 baseline 基准。

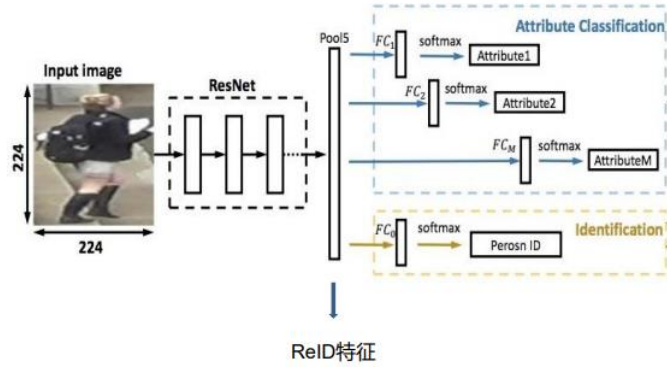
分类损失 (Classification/Identification Loss)



后来部分研究者认为, 光靠行人的 ID 信息不足以学习出一个泛化能力足够强的模型。因此他们利用了额外标注的行人图片的属性信息, 例如性别、头发、衣着等属性, 通过引入行人属性标签计算属性损失。训练好的网络不但要准确地预测出行人 ID, 还要预测出各项行人属性, 这大大增加了网络的泛化能力。具体的实现上, 可以同时连接几个属性分类损失

增强 ReID 特征的性能，每一个属性损失都是一个分类的交叉熵，可以等效于一个 multi-task 的网络。在测试阶段将所有分类 FC 丢弃，只使用 ReID 特征。

属性损失 (Attribute Loss)



基于度量学习 (Metric learning)的方法

度量学习 (Metric learning)旨在通过网络学习出两张图片的相似度。在行人重识别问题上，表现为同一行人的不同图片间的相似度大于不同行人的不同图片。具体为，定义一个映射 $f(x)$:

$$f(x): R^F \rightarrow R^D$$

将图片从原始域映射到特征域，之后再定义一个距离度量函数

$$D(x, y): R^D \times R^D \rightarrow R$$

来计算两个特征向量之间的距离。最后通过最小化网络的度量损失，来寻找这样的最优映射 $f(x)$ ，使得相同行人两张图片（正样本对）的距离尽可能小，不同行人两种图片（负样本对）的距离尽可能大。而这个映射 $f(x)$ ，就是我们训练得到的深度卷积网络。

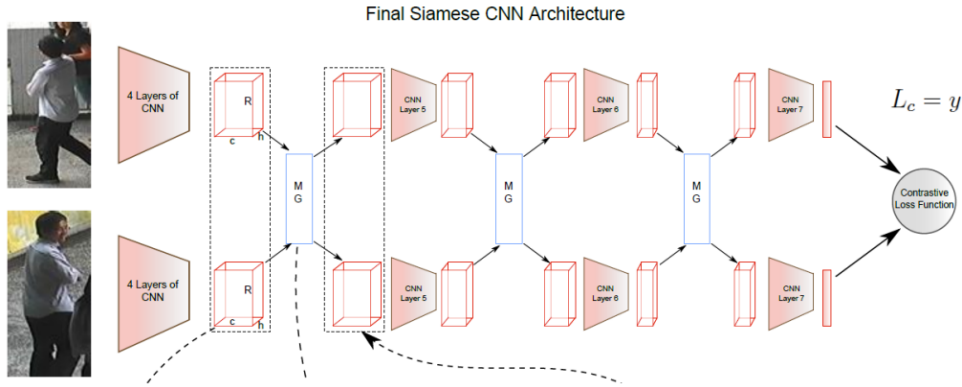
常用的度量学习损失方法包括对比损失(Contrastive loss)、三元组损失(Triplet loss)。首先，假如有两张输入图片 I_1 和 I_2 ，通过网络的前向传播我们可以得到它们（归一化后）的特征向量 f_{I_1} 和 f_{I_2} 。之后我们需要定义一个距离度量函数，通常我们使用特征的欧氏距离或者余弦距离作为度量函数，即两张图片在特征空间的距离定义为：

$$d_{I_1, I_2} = \|f_{I_1} - f_{I_2}\|_2$$

$$d_{I_1, I_2} = 1 - \frac{f_{I_1} \cdot f_{I_2}}{\|f_{I_1}\|_2 \|f_{I_2}\|_2}$$

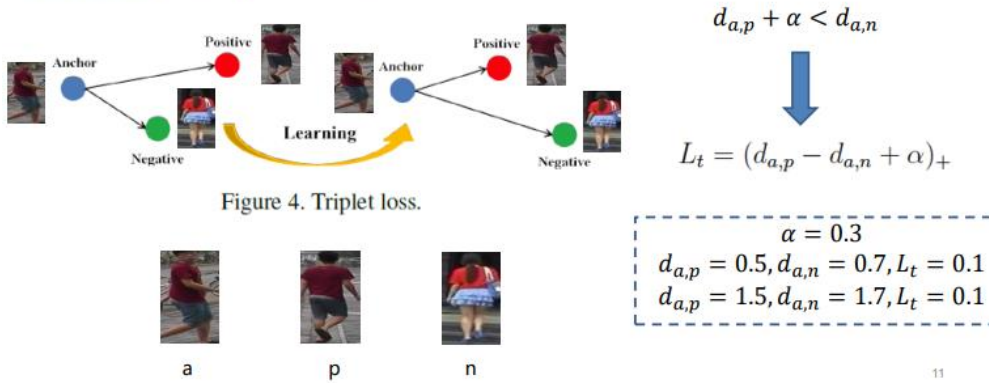
对比损失就是两种图片之间的度量距离。

对比损失 (Contrastive loss)



三元组损失是最被广泛应用的一种度量学习损失，之后的大量度量学习方法也是基于三元组损失演变而来。顾名思义，三元组损失需要三张输入图片。和对比损失不同，一个输入的三元组 (Triplet) 包括一对正样本对和一对负样本对。三张图片分别命名为固定图片 (Anchor) a ，正样本图片 (Positive) p 和 负样本图片 (Negative) n 。图片 a 和图片 p 为一对正样本对，图片 a 和图片 n 为一对负样本对。

三元组损失 (Triplet loss)



则三元组损失表示为:

$$L_t = (d_{a,p} - d_{a,n} + \alpha)_+$$

在计算度量损失时，样本对都是从训练集中随机挑选。随机挑选样本对的方法可能经常挑选出一些容易识别的样本对组成训练批量 (Batch)，使得 网络泛化能力受限。为此，部分学者提出了难样本采样 (Hard sample mining) 的方法，来挑选出难样本对训练网络[37, 41]。常用的思路是挑选出一个训练 Batch 中特征向量距离比较大 (非常不像) 的正样本 对和特征向量距离比较小 (非常像) 的负样本对来训练网络。难样本采样技术可以明显改进度量学习方法的性能，加快网络的收敛，并且可以很方便地在原有度量学习方法上进行扩展，是目前广泛采用的一种技术。

综上所述，度量学习可以近似看作为样本在特征空间进行聚类，表征学习可以近似看作为学习样本在特征空间的分界面。正样本距离拉近的过程使得类内距离缩小，负样本距离推开使得类间距离增大，最终收敛时样本在特征空间呈现聚类效应。度量学习和表征学习相比，优势在于网络末尾不需要接一个分类的全连接层，因此对于训练集的行人 ID 数

量并不敏感，可以应用于训练超大规模数据集的网络。总体而言，度量学习比表征学习使用的更加广泛，性能表现也略微优于表征学习。我们选择了基于度量学习的 FastReID 方法作为研究项目的行人重识别方法，实验证明，它在我们的项目使用上发挥了非常不错的效果。

Fast ReID 算法

DeepSORT 之所以能有效解决遮挡问题，是因为其搭建了一个重识别神经网络 WRN。WRN 主要由残差块组成，参数多、计算量大。为了增强重识别网络的速度和识别能力，我们引入行人重识别模型 FastReID 替代 WRN，可以在实现同样功能的前提下降低计算量，增加表达能力，达到更快的速度，更好的识别效果。

FastReID 的整体由图像预处理、主干网络、特征集成和网络头四大部分组成。整体架构如下图所示：

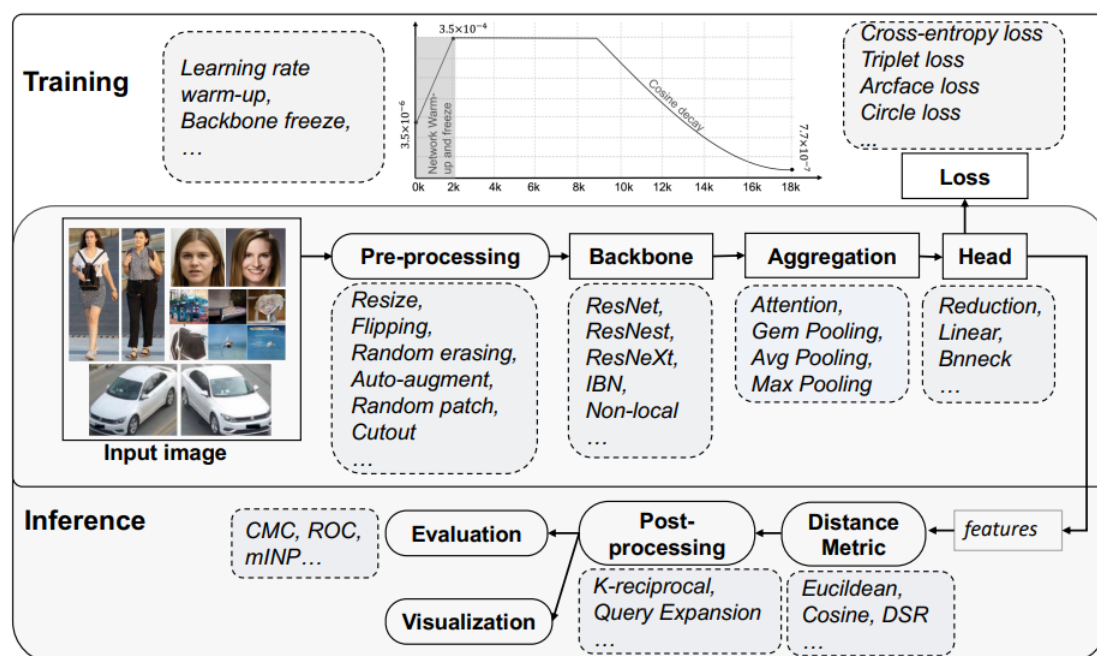


Figure 1. The Pipeline of FastReID library.

训练（Training）阶段

数据预处理模块（Pre-processing）:

在数据预处理中，FastReID 使用了各种数据增广方法，如 Resize, Flipping, Random erasing, Auto-augment, Random patch, Cutout 等；这些技巧提高了特征的鲁棒性。

骨干网模块（Backbone）

在骨干网中，实现了 ResNet 特征提取模型。我们还将 instance batch normalization (IBN) 模块添加到主干中，以学习更健壮的特性。

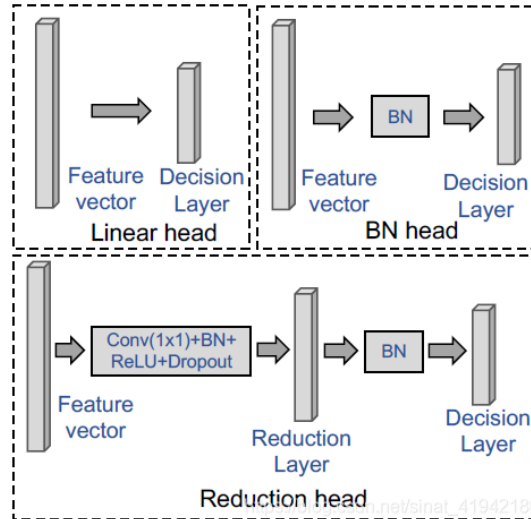
特征集成模块（Aggregation）

特征集成部分用于将骨干网生成的特征聚合成一个全局特征。实现了最大池化、平均池

化、Gem 池化和 Attention 池化四种池化操作。

头部模块 (Head)

头部模块是对聚合模块生成的全局向量进行处理的部分，包括批归一化 (BN) Head (batch normalization head)、线性 Head (Linear head) 和简化 Head (Reduction head)。三种类型的头如图 3 所示，线性头只包含一个 decision 层，BN 头包含一个 BN 层和一个 decision 层，简化头包含 conv+BN+relu+dropout 操作，一个 reduction 层和一个 decision 层。



批量归一化用于解决内部协变位移，因为很难训练具有饱和和非线性的模型。

简化层 (Reduction layer) 旨在将高维特征变成低维特征，即 2048 维-512 维。

决策层 (Decision layer) 输出不同类别的概率，区分不同类别，用于后续模型训练。

损失函数:

作者实现了四种 loss，包括:

- 交叉熵损失 (Cross-entropy loss)
- 三元组损失 (Triplet loss)
- Arcface loss
- Circle loss, 旷视于 2020 年提出的, 被认为是目前在各种度量学习任务中表现最好的

推理 (Inference) 阶段

度量模块 (Distance Metric):

输入的图片顺序经过 pre-processing, backbone, aggregation, head 之后会成为一个 512 维的特征向量。度量部分执行特征向量与 Gallery 向量库的对比，得到检索结果：一组于原图相似的图片。

后处理模块 (Post-processing):

后处理指的是对检索结果的处理，包括两种重排序 (re-rank) 方法: k-倒数编码[11] (K-reciprocal coding) 和查询扩展[12] (Query Expansion)。

性能评估模块 (Evaluation)

对于性能评估,我们采用了大多数人再识别文献中的标准度量,即累积匹配曲线(CMC)和平均平均精度(mAP)。此外,我们还增加了两个指标:受试者工作特性(ROC)曲线和平均逆负惩罚(mINP)。

可视化模块 (Visualization)

提供了检索结果的排名列表工具,有助于展示行人重识别的结果。

人流量统计

【沈爽】

基于视觉的人流量统计是指通过对监控视频进行一定的处理,能够估计出其中的人数。视觉人流量统计在许多领域均有广泛的应用,同时,监控摄像头广泛部署,人流量统计应用场景的不断扩充,引起了研究学者对人流量统计问题的极大关注,研究出了许多基于监控视频的人流量统计方法。

国际上许多公司在人数统计应用产品方面进行了大量研究与实验,比如飞瑞斯公司的人流量统计 Wise Count 系列产品,统计准确度超过 95%,可以一天不间断的工作,在实际应用中有很高的实时性和统计准确度。

针对商量客流量统计这一课题,我们研究了一种基于卷积神经网络的算法,可以记录每个客人进出商店的具体时间,用于商业决策和分析。

本文的主要研究内容

本文的主要研究内容是设计一种 YOLOv5 算法、DeepSORT 跟踪算法和 fastreid 行人重识别算法相结合的方法,以实现商店客流量的统计。我们通过对行人检测、跟踪和重识别算法的优化,实现了对不同角度和光照条件下的顾客进行准确的跟踪与计数,并开发了一个面向商店的客流量统计系统。

ГЛАВА 2 Обучение модели для отслеживания людей в нескольких видеопотоках（训练模型）

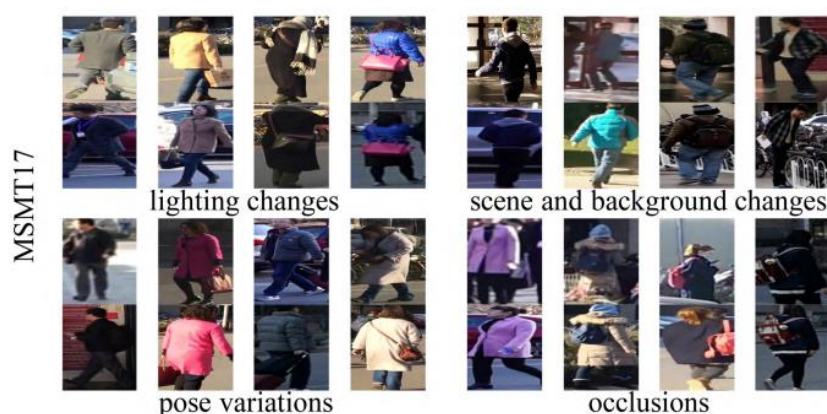
ReID 模型的训练

数据集准备

MSMT17 数据集

在CVPR2018会议上，提出了一个新的更接近真实场景的大型数据集MSMT17，即 Multi-Scene Multi-Time，涵盖了多场景多时段。

MSMT17数据集采用了安防在校园内的15个摄像头网络，其中包含12个户外摄像头和3个室内摄像头。为了采集原始监控视频，在一个月里选择了具有不同天气条件的4天。每天采集3个小时的视频，涵盖了早上、中午、下午三个时间段。因此，总共的原始视频时长为180小时。



基于Faster RCNN作为行人检测器，三位人工标注员用了两个月时间查看检测到的包围框和标注行人标签。最终，得到4101个行人的126441个包围框。和其它数据集的对比以及统计信息如下图所示。

Dataset	<i>MSMT17</i>	<i>Duke</i> [41, 27]	<i>Market</i> [39]	<i>CUHK03</i> [20]	<i>CUHK01</i> [19]	<i>ViPeR</i> [8]	<i>PRID</i> [10]	<i>CAVIAR</i> [3]
BBoxes	126,441	36,411	32,668	28,192	3,884	1,264	1,134	610
Identities	4,101	1,812	1,501	1,467	971	632	934	72
Cameras	15	8	6	2	10	2	2	2
Detector	Faster RCNN	hand	DPM	DPM, hand	hand	hand	hand	hand
Scene	outdoor, indoor	outdoor	outdoor	indoor	indoor	outdoor	outdoor	indoor

MSMT17与其他数据集相比的优势如下：

- （1）数目更多的行人、包围框、摄像头数；
- （2）复杂的场景和背景；
- （3）涵盖多时段，因此有复杂的光照变化；
- （4）更好的行人检测器（faster RCNN）， bounding box更精准。

评估协议

按照训练-测试为1: 3的比例对数据集进行随机划分, 而不是像其他数据集一样均等划分。这样做的目的是鼓励高效率的训练策略, 由于在真实应用中标注数据非常昂贵。

最后, 训练集包含1041个行人共32621个包围框, 而测试集包括3060个行人共93820个包围框。对于测试集, 11659个包围框被随机选出来作为query, 而其它82161个包围框作为gallery。

测试指标为CMC曲线和mAP. 对于每个query, 可能存在多个正匹配。

训练原理

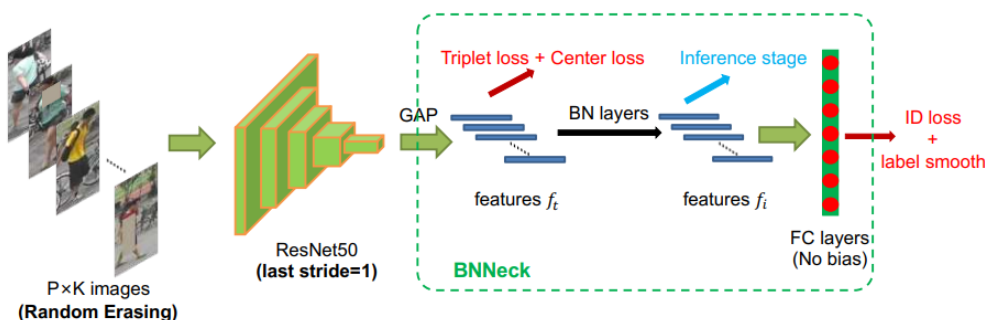
Reid 模型的训练原理如下:

1. 数据预处理: 首先将数据集中的图片进行 0.5 概率的裁剪、缩放、对称和数据增强等预处理操作, 以便提高模型对不同姿态和光照条件下的人员图像的鲁棒性。
2. 特征提取: FastReID 使用训练好的 ResNet 系列卷积神经网络作为特征提取器, 从输入图像中提取出 512 维特征向量。同时 FastReID 还引入一些新的特征提取方法, 例如 Scale-Aware Alignment (SAA) 和 random-erasing 等, 以进一步提高特征表达能力。
3. 度量学习: 通过度量学习的方式将特征向量映射到一个低维空间中, 并采用 triplet loss 和 Cross Entropy Loss 损失函数来优化模型, 使得同一身份的样本在特征空间中距离更近, 不同身份的样本在特征空间中距离更远。
4. 模型优化: 使用 Adam 优化器对模型参数进行优化, 以提高模型的训练效率和精度

构建模型

为了加速模型的收敛和提高模型性能, 我们使用了在 ImageNet 数据集上预训练的卷积神经网络模型 ResNet50。ImageNet 是一个广泛使用的图像分类数据集, 其中包含超过 1000 个类别的 140 万张图像。ResNet50 模型使用了深度残差网络结构, 可以有效地处理大规模图像分类任务。在训练完成后, 改模型的权重可以被用于其他计算机视觉任务中以提高任务的性能和效率, 比如我们的行人重识别。

预训练的 ResNet50 模型输出特征维度是 2048。我们在主干网络后面继续接入了一个平均池化层和一个线性分类器。线性分类器的输出维度是 N 。 N 表示训练数据的 ID 数目。



损失函数和优化算法

损失函数：

我们的 reid 模型会产生两个输出： features f 进行预测 logits p。features f 被用于计算 triplet loss，logits p 用于计算交叉损失熵。

Cross Entropy Loss

CrossEntropyLoss 是一个常用的分类损失函数，通常用于训练深度神经网络进行图像分类任务。具体地，对于一个输入样本 x 和其真实标签 y，Cross Entropy Loss 的计算公式如下：

$$\text{Cross Entropy Loss} = - \sum_{i=1}^C y_i \log(\hat{y}_i)$$

其中 C 是类别数， y_i 表示第 i 个类别的概率（即真实标签）， \hat{y}_i 表示模型预测 x 属于第 i 个类别的概率。

CrossEntropyLoss 的目标是最小化预测与真实标签之间的差距，使得模型能够更准确地预测出每个样本所属的类别(训练集中每一个人看做一个单独的类别)。

Triplet Loss

Triplet Loss 是在人脸识别领域中广泛应用的损失函数之一，其目的是将同一个人的图像嵌入向量（embedding）拉近，不同人的图像嵌入向量拉远。具体来说，Triplet Loss 会为每个样本学习一个嵌入向量，该向量能够使得同一个人的图像距离更近，不同人的图像距离更远。

对于一个 triplet 包括 anchor、positive 和 negative 三张图片，TripletLoss 的计算公式如下：

$$\text{Triplet Loss} = \max(0, d(a, p) - d(a, n) + m)$$

其中， $d(a, p)$ 表示 anchor 和 positive 之间的欧几里得距离， $d(a, n)$ 表示 anchor 和 negative 之间的欧几里得距离，m 为 margin，是一个预先设定的超参数，通常为正数。这个公式的含义是：如果当前的 anchor 和 positive 之间的距离减去 anchor 和 negative 之间的距离加上 margin 小于等于 0，则说明当前的嵌入向量已经足够好，不需要再进行优化；否则，需要更新模型参数以获取更好的嵌入向量。

Triplet Loss 通常与 Batch Hard Triplet Mining 结合使用，即在每个 batch 中选择最难的 triplet 进行训练。具体方法是选择与 anchor 非常相似但不同类别的行人图片作为负样本，选择与 anchor 差别很大但属于同一类别的行人图片作为正样本，以提高模型的性能。

训练步骤

- 1.使用 ResNet50（初始化权重来自于 ImageNet 的预训练模型），然后改变其全链接层为 N。N 表示训练数据的 ID 数目。

- 2.我们随机采样 P 个身份 ID，并且对每个 ID 采集 K 张，最后一个 batch size $B = P * K$ ，在这篇论文中，我们设置 $P=16, K=4$

- 3.我们改变每张图像的大小为 256×128 ，并且使用 0 值填充 10 个像素，然后使用随

机剪切的方式，重新剪切 256×128 大小的图像。

4. 每张图像以 0.5 的概率值，随机进行水平反转。

5. 每幅图像都被解码为[0,1]中 32 位浮点原始像素值，然后通过减去 0.485,0.456,0.406，再除以来对 RGB 通道进行归一化分别为 0.229、0.224、0.225。

6.模型输出的 ReID features f 进行预测 logits p。

7.ReID features f 被用于计算 triplet loss，logits p 用于计算交叉损失熵，triplet loss 的 margin m 设置为 0.3

8.采用 Adam 方法对模型进行优化。初始学习率设为 0.00035，在第 40 个 epoch 和第 70 个 epoch 分别降低 0.1。总共有 120 个 epoch

【添加伪代码】 或者【流程图】

训练过程

```
Terminal: local - Command Prompt
[05/08 03:10:20 fastreid.utils.events]: eta: 12:07:38 epoch/iter: 21/9999 total_loss: 1.474 loss_cls: 1.344 loss_triplet: 0.1238 time: 0.9364 data_time: 0.0008 lr: 3.50e-04 max_mem: 2539M
[05/08 03:10:33 fastreid.utils.events]: eta: 12:04:28 epoch/iter: 21/10399 total_loss: 1.474 loss_cls: 1.371 loss_triplet: 0.122 time: 0.9364 data_time: 0.0012 lr: 3.50e-04 max_mem: 2539M
[05/08 03:12:25 fastreid.utils.events]: eta: 12:01:20 epoch/iter: 21/10383 total_loss: 1.442 loss_cls: 1.331 loss_triplet: 0.1098 time: 0.9363 data_time: 0.0009 lr: 3.50e-04 max_mem: 2539M
[05/08 03:12:140 fastreid.utils.events]: eta: 12:01:04 epoch/iter: 22/10399 total_loss: 1.441 loss_cls: 1.329 loss_triplet: 0.1126 time: 0.9363 data_time: 0.0009 lr: 3.50e-04 max_mem: 2539M
[05/08 03:12:43 fastreid.utils.events]: eta: 11:57:29 epoch/iter: 22/10399 total_loss: 1.433 loss_cls: 1.326 loss_triplet: 0.1204 time: 0.9363 data_time: 0.0013 lr: 3.50e-04 max_mem: 2539M
[05/08 03:12:54 fastreid.utils.events]: eta: 11:54:17 epoch/iter: 22/10399 total_loss: 1.449 loss_cls: 1.341 loss_triplet: 0.1108 time: 0.9363 data_time: 0.0010 lr: 3.50e-04 max_mem: 2539M
[05/08 03:12:46 fastreid.utils.events]: eta: 11:51:41 epoch/iter: 22/10855 total_loss: 1.445 loss_cls: 1.332 loss_triplet: 0.1148 time: 0.9363 data_time: 0.0008 lr: 3.50e-04 max_mem: 2539M
[05/08 03:13:01 fastreid.utils.events]: eta: 11:51:16 epoch/iter: 23/10999 total_loss: 1.453 loss_cls: 1.33 loss_triplet: 0.1251 time: 0.9362 data_time: 0.0013 lr: 3.50e-04 max_mem: 2539M
[05/08 03:14:08 fastreid.utils.events]: eta: 11:47:51 epoch/iter: 23/11399 total_loss: 1.448 loss_cls: 1.34 loss_triplet: 0.1049 time: 0.9362 data_time: 0.0010 lr: 3.50e-04 max_mem: 2539M
[05/08 03:14:17 fastreid.utils.events]: eta: 11:45:48 epoch/iter: 23/11217 total_loss: 1.432 loss_cls: 1.316 loss_triplet: 0.115 time: 0.9362 data_time: 0.0008 lr: 3.50e-04 max_mem: 2539M
[05/08 03:17:15 fastreid.utils.events]: eta: 11:44:32 epoch/iter: 24/11399 total_loss: 1.434 loss_cls: 1.310 loss_triplet: 0.1221 time: 0.9362 data_time: 0.0014 lr: 3.50e-04 max_mem: 2539M
[05/08 03:17:11 fastreid.utils.events]: eta: 11:41:18 epoch/iter: 24/11399 total_loss: 1.453 loss_cls: 1.342 loss_triplet: 0.1033 time: 0.9361 data_time: 0.0010 lr: 3.50e-04 max_mem: 2539M
[05/08 03:14:28 fastreid.utils.events]: eta: 11:37:56 epoch/iter: 24/11799 total_loss: 1.413 loss_cls: 1.290 loss_triplet: 0.1203 time: 0.9361 data_time: 0.0018 lr: 3.50e-04 max_mem: 2539M
[05/08 03:14:28 fastreid.utils.events]: eta: 11:37:56 epoch/iter: 24/11799 total_loss: 1.413 loss_cls: 1.290 loss_triplet: 0.1203 time: 0.9361 data_time: 0.0018 lr: 3.50e-04 max_mem: 2539M
[05/08 03:14:58 fastreid.utils.events]: eta: 11:34:54 epoch/iter: 25/11999 total_loss: 1.460 loss_cls: 1.352 loss_triplet: 0.1079 time: 0.9361 data_time: 0.0012 lr: 3.50e-04 max_mem: 2539M
[05/08 03:14:52 fastreid.utils.events]: eta: 11:31:54 epoch/iter: 25/12199 total_loss: 1.434 loss_cls: 1.299 loss_triplet: 0.1185 time: 0.9360 data_time: 0.0008 lr: 3.50e-04 max_mem: 2539M
[05/08 03:15:09 fastreid.utils.events]: eta: 11:30:28 epoch/iter: 25/12271 total_loss: 1.418 loss_cls: 1.298 loss_triplet: 0.1127 time: 0.9360 data_time: 0.0014 lr: 3.50e-04 max_mem: 2539M
[05/08 03:15:24 fastreid.utils.events]: eta: 11:28:26 epoch/iter: 26/11399 total_loss: 1.454 loss_cls: 1.341 loss_triplet: 0.09971 time: 0.9360 data_time: 0.0010 lr: 3.50e-04 max_mem: 2539M
[05/08 03:15:25 fastreid.utils.events]: eta: 11:25:03 epoch/iter: 26/11399 total_loss: 1.42 loss_cls: 1.311 loss_triplet: 0.1054 time: 0.9359 data_time: 0.0009 lr: 3.50e-04 max_mem: 2539M
[05/08 03:15:39 fastreid.utils.events]: eta: 11:22:45 epoch/iter: 26/12743 total_loss: 1.424 loss_cls: 1.313 loss_triplet: 0.1171 time: 0.9359 data_time: 0.0014 lr: 3.50e-04 max_mem: 2539M
[05/08 03:15:52 fastreid.utils.events]: eta: 11:21:51 epoch/iter: 27/12799 total_loss: 1.434 loss_cls: 1.331 loss_triplet: 0.1091 time: 0.9359 data_time: 0.0012 lr: 3.50e-04 max_mem: 2539M
[05/08 03:16:02 fastreid.utils.events]: eta: 11:18:22 epoch/iter: 27/12999 total_loss: 1.416 loss_cls: 1.31 loss_triplet: 0.1034 time: 0.9358 data_time: 0.0009 lr: 3.50e-04 max_mem: 2539M
[05/08 03:16:15 fastreid.utils.events]: eta: 11:15:09 epoch/iter: 27/13199 total_loss: 1.444 loss_cls: 1.317 loss_triplet: 0.1161 time: 0.9358 data_time: 0.0014 lr: 3.50e-04 max_mem: 2539M
[05/08 03:16:29 fastreid.utils.events]: eta: 11:14:54 epoch/iter: 27/13115 total_loss: 1.444 loss_cls: 1.311 loss_triplet: 0.1151 time: 0.9358 data_time: 0.0012 lr: 3.50e-04 max_mem: 2539M
[05/08 03:16:45 fastreid.utils.events]: eta: 11:11:45 epoch/iter: 28/13399 total_loss: 1.434 loss_cls: 1.310 loss_triplet: 0.1088 time: 0.9357 data_time: 0.0010 lr: 3.50e-04 max_mem: 2539M
[05/08 03:17:27 fastreid.utils.events]: eta: 11:08:35 epoch/iter: 28/13399 total_loss: 1.428 loss_cls: 1.307 loss_triplet: 0.112 time: 0.9356 data_time: 0.0013 lr: 3.50e-04 max_mem: 2539M
[05/08 03:17:06 fastreid.utils.events]: eta: 11:07:06 epoch/iter: 28/13487 total_loss: 1.438 loss_cls: 1.331 loss_triplet: 0.1019 time: 0.9356 data_time: 0.0012 lr: 3.50e-04 max_mem: 2539M
[05/08 03:14:36 fastreid.utils.events]: eta: 11:05:19 epoch/iter: 29/13799 total_loss: 1.434 loss_cls: 1.319 loss_triplet: 0.09049 time: 0.9356 data_time: 0.0010 lr: 3.50e-04 max_mem: 2539M
[05/08 03:12:48 fastreid.utils.events]: eta: 11:02:19 epoch/iter: 29/13999 total_loss: 1.407 loss_cls: 1.291 loss_triplet: 0.107 time: 0.9355 data_time: 0.0014 lr: 3.50e-04 max_mem: 2539M
[05/08 04:12:09 fastreid.engine.defaults]: Prepare testing set
[05/08 04:12:10 fastreid.data.datasets.bases]: == Loaded HMNT17 in csv format:
# sunset | # ids | # images | # cameras |
+-----+-----+-----+-----+
| query | 3040 | 11488 | 15 |
| gallery | 3040 | 82161 | 15 |
[05/08 04:18:22 fastreid.evaluation.evaluator]: Start inference on 93820 images
[05/08 04:18:46 fastreid.evaluation.evaluator]: Inference done 11/733. 0.699 s / batch. ETA=0:08:04
[05/08 04:19:18 fastreid.evaluation.evaluator]: Inference done 36/733. 0.671 s / batch. ETA=0:07:35
[05/08 04:19:42 fastreid.evaluation.evaluator]: Inference done 101/733. 0.674 s / batch. ETA=0:07:07
[05/08 04:19:48 fastreid.evaluation.evaluator]: Inference done 144/733. 0.674 s / batch. ETA=0:06:37
[05/08 04:19:48 fastreid.evaluation.evaluator]: Inference done 191/733. 0.679 s / batch. ETA=0:06:10
[05/08 04:19:48 fastreid.evaluation.evaluator]: Inference done 236/733. 0.675 s / batch. ETA=0:05:36
[05/08 04:19:48 fastreid.evaluation.evaluator]: Inference done 281/733. 0.679 s / batch. ETA=0:05:06
[05/08 04:19:48 fastreid.evaluation.evaluator]: Inference done 325/733. 0.676 s / batch. ETA=0:04:36
[05/08 04:19:48 fastreid.evaluation.evaluator]: Inference done 369/733. 0.678 s / batch. ETA=0:04:07
[05/08 04:19:48 fastreid.evaluation.evaluator]: Inference done 413/733. 0.678 s / batch. ETA=0:03:37
[05/08 04:19:48 fastreid.evaluation.evaluator]: Inference done 457/733. 0.676 s / batch. ETA=0:03:08
[05/08 04:19:48 fastreid.evaluation.evaluator]: Inference done 501/733. 0.680 s / batch. ETA=0:02:35
[05/08 04:19:48 fastreid.evaluation.evaluator]: Inference done 544/733. 0.682 s / batch. ETA=0:02:09
[05/08 04:19:48 fastreid.evaluation.evaluator]: Inference done 587/733. 0.683 s / batch. ETA=0:01:40
[05/08 04:19:48 fastreid.evaluation.evaluator]: Inference done 631/733. 0.684 s / batch. ETA=0:01:10
[05/08 04:19:48 fastreid.evaluation.evaluator]: Inference done 675/733. 0.684 s / batch. ETA=0:00:39
[05/08 04:19:48 fastreid.evaluation.evaluator]: Inference done 719/733. 0.683 s / batch. ETA=0:00:09
[05/08 04:19:48 fastreid.evaluation.evaluator]: Total Inference Time: 0:08:45.339901 (0.721615 s / batch per device, on 1 devices)
[05/08 04:19:48 fastreid.evaluation.evaluator]: Total Inference Time: 0:08:45.339901 (0.721615 s / batch per device, on 1 devices)
[05/08 04:19:48 fastreid.evaluation.evaluator]: Total Inference pure compute Time: 0:08:14 (0.702107 s / batch per device, on 1 devices)
C:\sources\11\fastreid-fast-reid\fastreid\evaluation\eval.py:14: UserWarning: Cython num evaluation (very fast so highly recommended) is unavailable, now use python evaluation.
warnings.warn(
[05/08 04:19:48 fastreid.engine.defaults]: Evaluation results for HMNT17 in csv format:
# sunset | # query | # gallery | # hits | # miss | # miss | # miss |
+-----+-----+-----+-----+-----+-----+-----+
| HMNT17 | 86.19 | 89.95 | 89.16 | 82.89 | 14.94 | 45.34 |
[05/08 04:19:48 fastreid.engine.defaults]: Evaluation results for HMNT17 in csv format:
# sunset | # query | # gallery | # hits | # miss | # miss | # miss |
+-----+-----+-----+-----+-----+-----+-----+
| HMNT17 | 86.19 | 89.95 | 89.16 | 82.89 | 14.94 | 45.34 |
```

训练结果

评价指标

在行人重识别研究中，主要采用累计匹配曲线（Cumulative Matching

Characteristics, CMC) 和平均精度均值 (Mean Average Precision, mAP) 这两种评价指标来评估模型的性能。

累计匹配曲线 (CMC)

累计匹配曲线具体体现为第 k 位匹配率 (Rank- k)，具体指的是在候选图像库 G 中选取与待检索图片 $probe$ 最相似的 k 张图像中找到相同行人图片的概率，用公式表达如下：

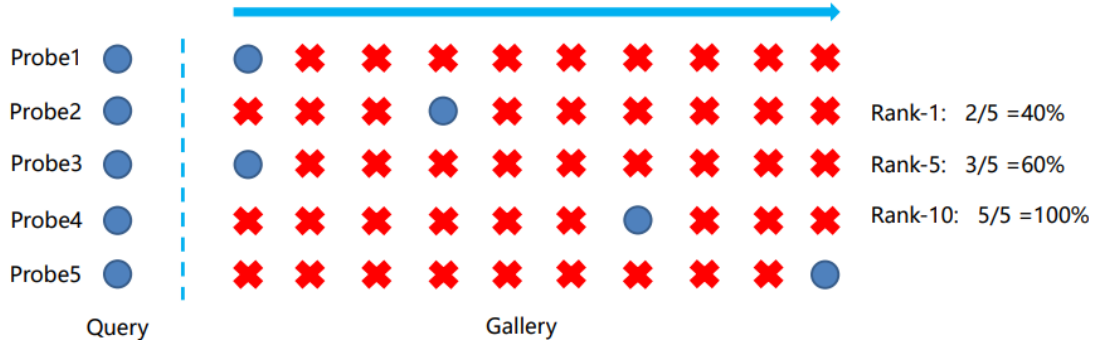
$$Rank-k = \frac{\sum_{probe \in Q} f(k, index_{probe})}{m}$$

$$f(k, index_{probe}) = \begin{cases} 0, & k \geq index_{probe} \\ 1, & k < index_{probe} \end{cases}$$

其中， $index_{probe}$ 表示检索结果中与待查询图像 $probe$ 属于同一行人的第一张图片所在的位置， m 表示待检索库 Q 中待检索图片的数目。通常来说，Rank-1 是评估行人重识别模型性能最关键的指标，其可以表示为检索得到的排序列表中首张图片与待检索图片是相同行人的概率。在实际情况中，除了使用 rank-1 之外，其他常用的几个 Rank- k 包括 Rank-5，Rank-10，Rank-20 等。

常用的评价指标——rank-k

- rank-k: 算法返回的排序列表中，前 k 位为存在检索目标则称为 rank-k 命中



平均精度均值 (mean average precision, mAP)

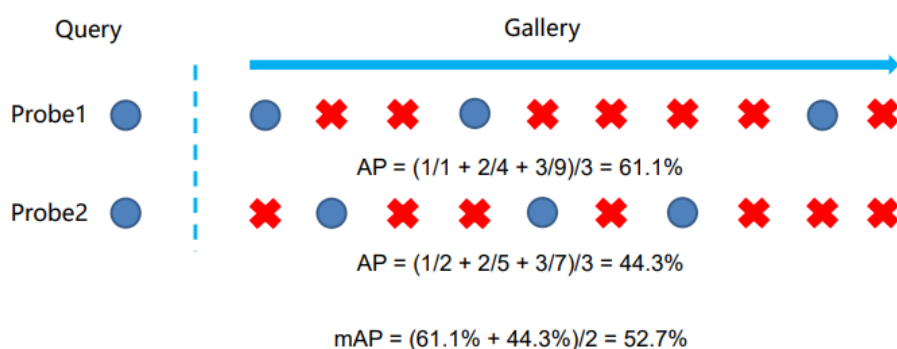
在早期的行人重识别数据集中，在图像候选库 G 中有且仅有一张和检索图片 $probe$ 相同的行人图片，但是随着 Market-1501 和 DukeMTMC-reID 等大型数据集的提出，一张待检索图像 $probe$ 在候选图像库 G 中通常可以找到多张相同身份的匹配图片，仅使用 CMC 评价指标难以评估难检索样本对模型性能的影响。因此，为了更加全面地评价行人重识别模型的性能，行人重识别模型的评价指标中加入了平均精度均值。平均精度均值就是一种能够评价全部正样本的排序结果的指标，只有被检索人在候选库中所有的图片都排在最前面时，mAP 的指标才会高，因此它能更全面地反映行人重识别模型的性能。在计算 mAP 时，首先会计算每个待检索图像 $probe$ 所对应的平均精度 (Average Precision, AP)，它是用来衡量模型在单个查询样本上的识别精度的，计算过程如式 (3.19) 所示：

$$AP(q) = \frac{\sum_{k \in k_1, k_2, \dots, k_S} \frac{k_r}{k}}{S}$$

其中， S 表示候选库 G 中对应检索图像 $probe$ 的正样本数量， $\{k_1, k_2, \dots, k_S\}$ 是 S 个正样本在排序结果中的索引位置， k_r 表示前 k 个结果中正样本的数量。最后，在计算包含 m 张图片的待检索库 Q 中所有行人图像的平均精度后，对所有样本的 AP 值作均值即可得到 mAP ，计算过程如式（3.20）所示：

$$mAP = \frac{\sum_{q_i \in Q} AP(q_i)}{m}$$

- mAP (mean average precision): 反应检索的人在数据库中所有正确的图片排在排序列表前面的程度，能更加全面的衡量ReID算法的性能。

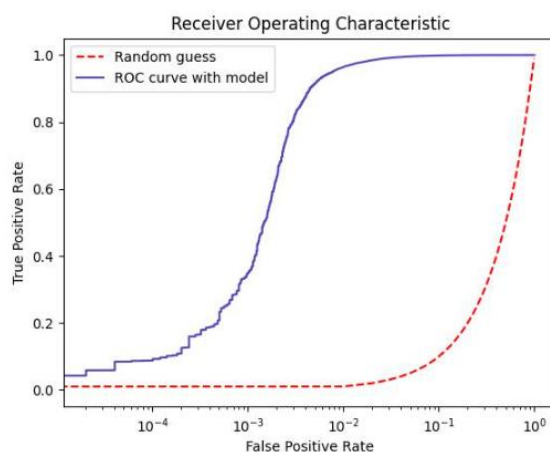


均值逆负惩罚（ $mINP$ ）：表示找到所有正确匹配的成本，越低越好。

在数据集 MSMT17 上的训练结果

我们使用 MSMT17 中 1/3 的数据作为样本去训练 reid 模型，然后用剩下的 2/3 数据作为验证集检验模型的能力，得到结果如下：

Dataset	Rank-1 ↑	Rank-5 ↑	Rank-10 ↑	mAP ↑	mINP ↓
MSMT17	84.19	90.95	95.10	62.89	14.94



根据给出的数据，可以看出 FastReID 模型在 MSMT17 数据集上获得了不错的结果。具

体来说,它在 Rank-1 准确率上达到了 84.19%,在 Rank-5 和 Rank-10 准确率上分别为 90.95% 和 95.10%。同时,平均精度 (mAP) 也达到了 62.89%,这意味着模型在整个数据集上的表现都比较稳定。需要指出的是,在逆平均负样本惩罚率 (mINP) 方面,模型仅为 14.94%,这意味着在一些难以识别的情况下,模型的表现可能会下降。总的来说,这些结果显示了 FastReID 模型在 MSMT17 数据集上表现良好,但如果应用到特别难识别的场景时,还有一些改进空间。

横向对比其他 reid 模型在该数据集上的表现:

Methods	MSMT17	
	Rank-1	mAP
IANet(IVPR'19)	75.7	45.8
Auto-ReID(ICC'19)	78.2	52.5
OSNet(ICC'19)	78.7	52.9
ABDNet(ICC'19)	82.3	60.8
Circle Loss[(CVPR'20)	76.9	52.1
ours	84.19	62.89

经过横向对比得知,我们的模型准确率远高于其他同时期发布的模型,准确率性能达到最佳。

基于 FastReID+YOLOv5 搭建 DeepSORT 模型

构建模型

1. 使用 DeepSORT 作为跟踪网络,来跟踪每个人物并将其与之前的轨迹相关联
2. 使用 YOLOv5s 作为 DeepSORT 的检测部分,检测每帧图像中的人物,输出检测框
3. 使用 FastReID 作为特征提取器提取行人的外观特征,使得 DeepSORT 可以区分不同的人物
4. 确定模型超参数。训练模型的参数如下:

DEEPSORT:

```

REID_CHKPT: "./fast-reid/checkpoint/model-final.pth"
MAX_DIST: 0.2
MIN_CONFIDENCE: 0.3
NMS_MAX_OVERLAP: 0.5
MAX_IOU_DISTANCE: 0.7
MAX_AGE: 140

```

N_INIT: 3
NN_BUDGET: 100

测试数据集 MOT-16

我们采用行人公共数据集 MOT-16 来测试我们的多目标跟踪模型（改进行人重识别模块后的 DeepSORT）。该数据集是用于评价多目标跟踪算法性能的常用数据集，有多种多样行人场景，包括 14 个视频序列，有 7 个视频序列详细标注了行人身份和边框位置，用于训练多目标行人跟踪算法；另外 7 个视频序列作为测试集。



图 4-1 MOT 16 数据集示意图

实验采用评价标准为：跟踪准确度（MOTA）、跟踪精度（MOTP）、目标轨迹正确跟踪 80% 以上的个数（MT）、目标轨迹正确跟踪 20%以下的个数（ML）、目标 ID 变换次数（IDSW），跟踪准确度与跟踪精度计算公式如下：

$$MOTA = 1 - \frac{\sum_t (m_t + n_t + s_t)}{\sum_t g_t}$$
$$MOTP = \frac{\sum_{i,t} d_i^t}{\sum_t c_t}$$

式中：t 表示第 t 帧， m_t 表示 t 帧漏检目标数目； n_t 表示 t 帧误检目标数目； s_t 表示 t 帧身份切换数目； g_t 表示 t 帧出现目标总数目； d_i^t 表示第 t 帧目标 i 预测位置与真实位置之间的距离； c_t 表示第 t 帧中成功匹配的目标数。

该次实验均在 MOT-16 的测试集视频序列上进行，为了更好地凸显本文算法的性能，设计了两组不同实验进行对照比较。

实验 1：对比测试经过我们改进后的 DeepSORT 算法（YOLOv5+FastReID+DeepSORT）和原 DeepSORT 算法。将两者放在测试集视频上测试，分析改进的 DeepSort 算法的优劣性。

实验 2：分析在多样场景下改进算法的鲁棒性。将改进算法与几种主流算法对比，分析改进算法的优劣性。

实验1：对比测试经过我们改进后的DeepSORT算法与原DeepSORT算法

实验过程

我们将对 MOT16 数据集中所有拥有行人身份和边框位置人工标注（gt.txt）的视频序列进行测试，一共 7 个。分别是：MOT16-02, MOT16-04, MOT16-05, MOT16-09, MOT16-10, MOT16-11, MOT16-13。这里以 MOT16-13 为例，对应其他视频操作相同。

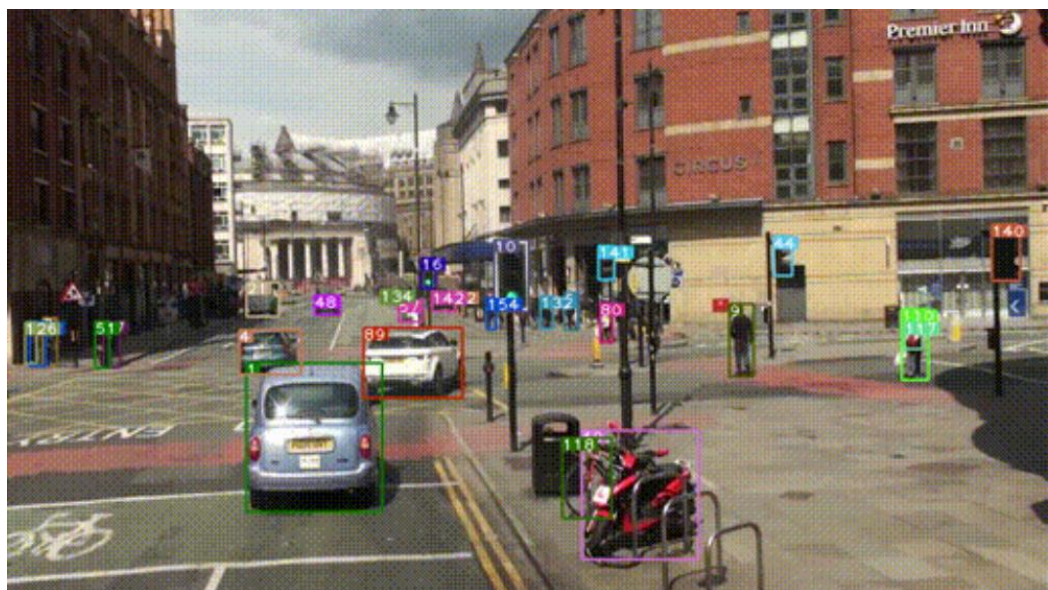
1. 下载 MOT16 数据集，获取视频序列和各自对应的 gt.txt 文件。MOT16-13 中 gt.txt 文件如下(部分)，各参数间用逗号分隔：

```
1,1,1376,485,37,28,0,11,1
2,1,1379,486,37,28,0,11,1
3,1,1382,487,38,29,0,11,1
4,1,1386,488,38,29,0,11,1
5,1,1389,490,38,29,0,11,1
```

gt.txt 文件是 CSV 文本文件，每行包含一个对象，描述其中一帧中的一个跟踪对象，有 9 个值，用逗号分隔。TrackEval 只用到前 6 个，帧序号，目标 ID，跟踪框 4 个坐标，后 3 个（目标置信度，目标类别，可见性）不参与运算，可忽略，如下：

<frame>, <id>, <bb_left>, <bb_top>, <bb_width>, <bb_height>, <conf>, <class>, <visibility>

2. 运行我们构建的 DeepSORT 模型中的 track.py 程序



```
Terminal - Command Prompt (2)
(1, 256, 320, 3)
[INFO] len(strongsort.list): 2
C:\Users\yeyoung\OneDrive\Documents\StrongSORT_Video\Scripts\run_eval.py:505: UserWarning: torch.meshgrid: in an upcoming release, it will be required to pass the indexing argument. (Triggered internally at C:\actions-runner\_work\python\python\build\install\python\lib\tensor\shape.cpp:3491.)
  return _VF.meshgrid(tensors, **kwargs) # type: ignore[attr-defined]
video 1/1 (1/750) C:\source\GitRepo\reid\Volov5_DeepSORT_FastReID\videos\MOT16\MOT16-13.mp4: 384x640 9 persons, Done. VLOv5:(0.0793), DeepSORT:(0.2036)
video 1/1 (2/750) C:\source\GitRepo\reid\Volov5_DeepSORT_FastReID\videos\MOT16\MOT16-13.mp4: 384x640 9 persons, Done. VLOv5:(0.0783), DeepSORT:(0.3286)
video 1/1 (3/750) C:\source\GitRepo\reid\Volov5_DeepSORT_FastReID\videos\MOT16\MOT16-13.mp4: 384x640 9 persons, Done. VLOv5:(0.0626), DeepSORT:(0.2036)
video 1/1 (4/750) C:\source\GitRepo\reid\Volov5_DeepSORT_FastReID\videos\MOT16\MOT16-13.mp4: 384x640 9 persons, Done. VLOv5:(0.0625), DeepSORT:(0.3555)
video 1/1 (5/750) C:\source\GitRepo\reid\Volov5_DeepSORT_FastReID\videos\MOT16\MOT16-13.mp4: 384x640 8 persons, Done. VLOv5:(0.0635), DeepSORT:(0.1546)
video 1/1 (6/750) C:\source\GitRepo\reid\Volov5_DeepSORT_FastReID\videos\MOT16\MOT16-13.mp4: 384x640 8 persons, Done. VLOv5:(0.0473), DeepSORT:(0.1876)
video 1/1 (7/750) C:\source\GitRepo\reid\Volov5_DeepSORT_FastReID\videos\MOT16\MOT16-13.mp4: 384x640 9 persons, Done. VLOv5:(0.0473), DeepSORT:(0.2036)
video 1/1 (8/750) C:\source\GitRepo\reid\Volov5_DeepSORT_FastReID\videos\MOT16\MOT16-13.mp4: 384x640 10 persons, Done. VLOv5:(0.0626), DeepSORT:(0.2206)
video 1/1 (9/750) C:\source\GitRepo\reid\Volov5_DeepSORT_FastReID\videos\MOT16\MOT16-13.mp4: 384x640 10 persons, Done. VLOv5:(0.0535), DeepSORT:(0.2406)
video 1/1 (10/750) C:\source\GitRepo\reid\Volov5_DeepSORT_FastReID\videos\MOT16\MOT16-13.mp4: 384x640 11 persons, Done. VLOv5:(0.0626), DeepSORT:(0.2386)
video 1/1 (11/750) C:\source\GitRepo\reid\Volov5_DeepSORT_FastReID\videos\MOT16\MOT16-13.mp4: 384x640 10 persons, Done. VLOv5:(0.0596), DeepSORT:(0.2086)

Python Packages TO DO Python Console Problems Terminal Services
Localized PyCharm 2022.1.3 is available // Switch and restart // Don't ask again (12 minutes ago)
```

```
Terminal - Command Prompt (2)
video 1/1 (736/750) C:\source\GitRepo\reid\Volov5_DeepSORT_FastReID\videos\MOT16\MOT16-13.mp4: 384x640 7 persons, Done. VLOv5:(0.0626), DeepSORT:(0.1516)
video 1/1 (735/750) C:\source\GitRepo\reid\Volov5_DeepSORT_FastReID\videos\MOT16\MOT16-13.mp4: 384x640 6 persons, Done. VLOv5:(0.0473), DeepSORT:(0.1566)
video 1/1 (736/750) C:\source\GitRepo\reid\Volov5_DeepSORT_FastReID\videos\MOT16\MOT16-13.mp4: 384x640 4 persons, Done. VLOv5:(0.0643), DeepSORT:(0.0966)
video 1/1 (737/750) C:\source\GitRepo\reid\Volov5_DeepSORT_FastReID\videos\MOT16\MOT16-13.mp4: 384x640 4 persons, Done. VLOv5:(0.0473), DeepSORT:(0.0896)
video 1/1 (738/750) C:\source\GitRepo\reid\Volov5_DeepSORT_FastReID\videos\MOT16\MOT16-13.mp4: 384x640 6 persons, Done. VLOv5:(0.0626), DeepSORT:(0.1416)
video 1/1 (739/750) C:\source\GitRepo\reid\Volov5_DeepSORT_FastReID\videos\MOT16\MOT16-13.mp4: 384x640 6 persons, Done. VLOv5:(0.0626), DeepSORT:(0.1256)
video 1/1 (740/750) C:\source\GitRepo\reid\Volov5_DeepSORT_FastReID\videos\MOT16\MOT16-13.mp4: 384x640 4 persons, Done. VLOv5:(0.0483), DeepSORT:(0.0896)
video 1/1 (741/750) C:\source\GitRepo\reid\Volov5_DeepSORT_FastReID\videos\MOT16\MOT16-13.mp4: 384x640 3 persons, Done. VLOv5:(0.0573), DeepSORT:(0.0866)
video 1/1 (742/750) C:\source\GitRepo\reid\Volov5_DeepSORT_FastReID\videos\MOT16\MOT16-13.mp4: 384x640 3 persons, Done. VLOv5:(0.0513), DeepSORT:(0.0666)
video 1/1 (743/750) C:\source\GitRepo\reid\Volov5_DeepSORT_FastReID\videos\MOT16\MOT16-13.mp4: 384x640 4 persons, Done. VLOv5:(0.0506), DeepSORT:(0.0966)
video 1/1 (744/750) C:\source\GitRepo\reid\Volov5_DeepSORT_FastReID\videos\MOT16\MOT16-13.mp4: 384x640 4 persons, Done. VLOv5:(0.0616), DeepSORT:(0.1606)
video 1/1 (745/750) C:\source\GitRepo\reid\Volov5_DeepSORT_FastReID\videos\MOT16\MOT16-13.mp4: 384x640 4 persons, Done. VLOv5:(0.0576), DeepSORT:(0.1216)
Speed: 0.5ms pre-process, 56.0ms inference, 2.0ms NN, 122.7ms strong sort update per image at shape (1, 3, 640, 640)
Results saved to runs\track\exp15\tracks
1 tracks saved to runs\track\exp15\tracks

Python Packages TO DO Python Console Problems Terminal Services
Localized PyCharm 2022.1.3 is available // Switch and restart // Don't ask again (12 minutes ago)
```

得到兼容 MOT16 格式的跟踪数据文件 MOT16-13.txt。文件 MOT16-13.txt:

```
3 7 726 247 12 40 -1 -1 -1 0
3 8 269 276 16 45 -1 -1 -1 0
3 9 744 256 13 37 -1 -1 -1 0
4 1 1 341 13 93 -1 -1 -1 0
4 2 773 279 22 57 -1 -1 -1 0
```

MOT16-13.txt 格式与 gt.txt 略有差别，号称与 MOT16 格式“兼容”。“兼容”格式共 10 个值，其参数间以空格分隔。参与 MOT16 运算的前 6 个参数与 gt.txt 相同，后 4 个不参与指标运算，均为-1。格式如下：

<frame>, <id>, <bb_left>, <bb_top>, <bb_width>, <bb_height>, <conf>, <x>, <y>, <z>

3. 运行评估程序，得到结果：

All sequences for deepsort finished in 0.91 seconds															
MOTA: deepsort-pedestrian		MOTA	Delta	AssA	DetRe	DetPr	AssRe	AssPr	LocA	OMTA	HOTA(0)	LocA(0)	HOTALocA(0)		
MOT16-13		57.313	59.474	56.029	66.611	73.942	63.201	75.428	81.504	68.957	74.316	76.452	56.816		
COMBINED		57.313	59.474	56.029	66.611	73.942	63.201	75.428	81.504	68.957	74.316	76.452	56.816		
CLEAR: deepsort-pedestrian		MOTA	MOTP	MODA	CLR_Re	CLR_Pr	MTR	PTR	MLR	sMOTA	CLR_TP	CLR_FN	CLR_FP	IDSW	MT
MOT16-13		72.154	78.316	72.338	81.212	90.149	50	31.25	18.75	54.544	2654	614	290	6	8
COMBINED		72.154	78.316	72.338	81.212	90.149	50	31.25	18.75	54.544	2654	614	290	6	8
Count: deepsort-pedestrian		Dets	GT_Dets	IDs	GT_IDs										
MOT16-13		2944	3268	18	16										
COMBINED		2944	3268	18	16										

4. 分别对 MOT16-02, MOT16-04, MOT16-05, MOT16-09, MOT16-10, MOT16-11 进行同样的操作，遍历整个 MOT16 数据集。得到这个 MOT16 数据集上最终的结果。

实验结果

对比原 DeepSORT 算法的表现，见下表：

Algorithm	MOTA ↑	MOTP ↑	MT / % ↑	ML / % ↓	IDs / % ↓
-----------	--------	--------	----------	----------	-----------

Original DeepSORT	61.4	79.1	32.8	18.2	781
Ours	66.2	80.8	35.3	17.6	760

可以看出,改进了 DeepSORT 算法后,各项指标都略有提升。跟踪准确度 (MOTA) 提升了 5.2%,跟踪精度 (MOTP) 提升了 1.7%,目标轨迹正确跟踪 80%以上的个数 (MT) 提升了 3.5%,目标轨迹正确跟踪 20%以下的个数 (ML) 降低了 0.6%,目标身份切换总数 (IDs) 减少了 21 次,可见改进 DeepSORT 能减少行人切换次数。

实验2: 改进算法与几种主流算法对比

实验 2 结果见表。

Algorithm	MOTA ↑	MOTP ↑	MT / % ↑	ML / % ↓	IDs / % ↓	FPS / Hz ↑
SORT	59.8	79.6	25.4	22.7	1423	8.6
Original DeepSORT	61.4	79.1	32.8	18.2	781	6.4
JDE	64.4	-	35.4	20	1544	18.5
Ours	66.2	80.8	35.3	17.6	760	5.8

我们在 DeepSORT 的基础上改良了行人重识别的算法 (fastREID), 因此模型的准确度、精度和维持行人 ID 的能力都有提升。我们的模型与其他模型相比具有全面的优势,跟踪准确度 (MOTA)、跟踪精度 (MOTP) 都是最高,目标轨迹正确跟踪 80%以上的个数 (MT) 与 JDE 几乎持平并列第一,目标轨迹正确跟踪 20%以下的个数 (ML) 和目标身份切换总数 (IDs) 都是最低的。

此外注意到与 JDE 算法相比实时性略差,这是因为 JDE 属于一阶段的跟踪算法,实时性跟踪相对较高,但是 ID 切换相对频繁,这是由于目标相互遮挡导致的。在零售的环境下经常出现大量密集行人的场景,所以此时我们的算法是一个合适的选择。

ГЛАВА 3 Система оценки трафика магазина основа на нейтронных сетей （商店客流量统计系统）

项目背景

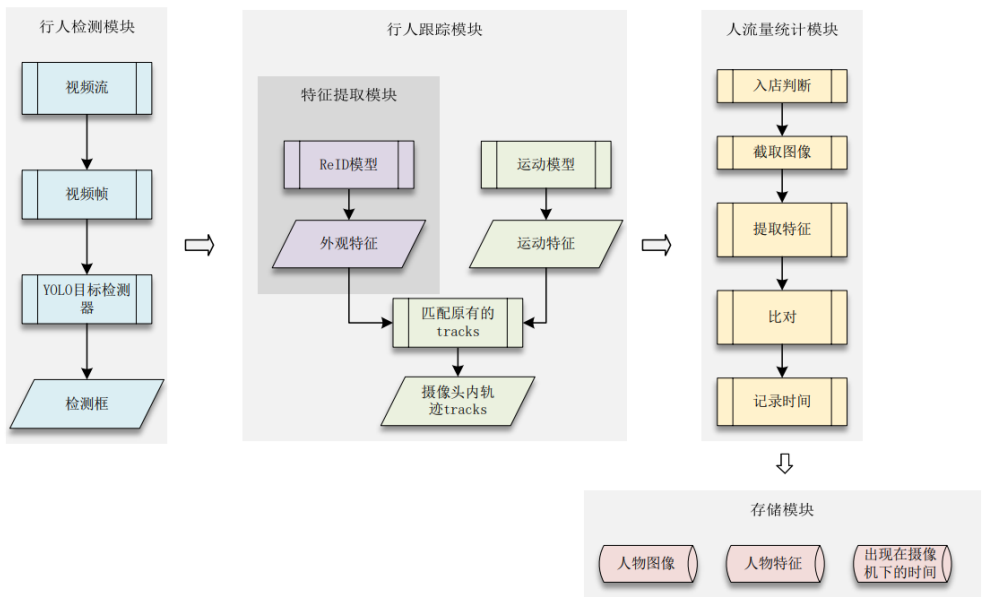
基于监控视频的人流量统计是资源合理分配、收集商业信息和进行智能管理等任务的基础，它也是现代视频智能监控重要的功能组成部分。本项目将采用 YOLOv5、DeepSORT 和 Fastreid 等先进的计算机视觉和深度学习技术,实现对商店内顾客的自动识别、追踪和分析。具体而言，该系统将通过摄像头获取商店内的视频流，利用 YOLOv5 对视频中出现的顾客进行自动检测和识别，然后使用 DeepSORT 对顾客进行目标跟踪，最终使用 fastreid 对顾客进行身份识别和行为分析，从而实现对商店客流量的自动统计和分析。

通过本项目的研究和实践，可以帮助商家更加精准地了解客户的行为习惯和偏好，制定更加科学有效的销售策略，提升商店的盈利能力和市场竞争力。同时，该系统还可以为商家提供实时的客流量监控和预测功能，帮助商家进一步优化运营管理，提高服务水平和用户体验。

商店客流量统计系统简介

系统整体架构

本文在第三章提出的跨域行人重识别方法和第四章提出的跨摄像头多目标跟踪方法的基础上，设计了一个面向社区的跨摄像头行人跟踪系统。该系统由四个模块组成：行人检测模块、特征提取模块、单摄像头跟踪模块以及跨摄像头轨迹关联模块。系统的整体架构如图 5.1 所示。



行人检测模块

行人检测模块的输入为单个摄像头中的视频流，在实现过程中本文采用YOLO目标检测算法。利用YOLO目标检测算法对输入的图像序列依次进行行人检测，将其中的行人用检测框标记出来，并返回检测框的具体位置信息，定义如式（5.1）所示：

$$b = [x, y, w, h] \quad (5.1)$$

其中， (x, y) 为检测框左上角像素坐标， (w, h) 为检测框的宽度和高度。

特征提取模块

特征提取模块旨在提取行人的外观特征。在这里将使用 FastReID 进行。

特征提取模块的输入为行人检测模块中输出的行人检测框，经过 ReID 模型后，得到行人的外观特征。外观特征将会为后面的行人跟踪模块进行轨迹和检测框匹配提供参考。

行人跟踪模块

行人跟踪模块采用了DeepSORT模型，其输入为行人检测模块提供的检测框和特征提取模块提取的行人特征，目的是将当前帧中的行人检测框与前一帧中现存的运动轨迹进行关联。在得到行人检测模块提供的检测框后，首先利用特征提取模块和运动信息提取模块提取当前帧检测框和现存运动轨迹的外观特征以及运动特征，然后计算它们之间的距离矩阵，最后使用和匈牙利算法将当前帧中的候选检测框与前一帧中现存的运动轨迹进行关联，构建视频中所有行人的轨迹。

人流量统计模块

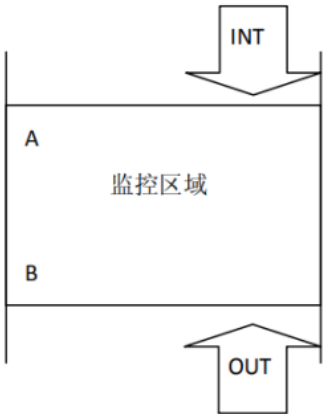


图 3-5 计数区域

在考虑了现实作业需求后，本研究使用“双线法”来完成人流量计数。双线法的具体原理如下：在图像中标记出两条计数线，分别记做 A、B。以身体运动轨迹为基础，通过轨迹与线相交的情况来表示从此经过的具体人数。

举个例子，我们假设商店的门位于 B 线的下方。那么顾客的轨迹会先接触 A 线再接触 B，这被称为进门行为。反之，如果行人的轨迹先接触 B 线再解除 A 线，则被称为出门行

为。每发生一次进门行为，行人检测、行人追踪、特征提取模块会依次工作，以获得顾客截图和特征，并在 log 中记录进入时间。当出门行为发生时，获取顾客的截图和特征，与数据库中的特征集进行比对，获取顾客 id，并根据 id 记录顾客离开的时间。于是我们就完成了记录顾客进入和离开超市具体时间的工作。

人流量统计模块的输入为所有人在单摄像头下的轨迹。在人物进入监控区域时会触发 ReID 的特征提取和比对行为。在进行特征比对的时候，需要用到先前存储的所有顾客的特征集合，可以通过访问数据库获取。

存储模块

存储模块负责将顾客进入商店、出现在摄像机下和离开商店的时间进行记录。

每次有人进入计数区域的时候，记录该人的出现时间，同时将截取对应的人物图像，并根据图像提取特征向量。如果该摄像头是店内或出口摄像头，则还会根据 reid 模型计数该行人的身份。最终将人物身份、出现在哪个镜头下、出现时间、人物图像存储在磁盘上，以便后续统计分析。

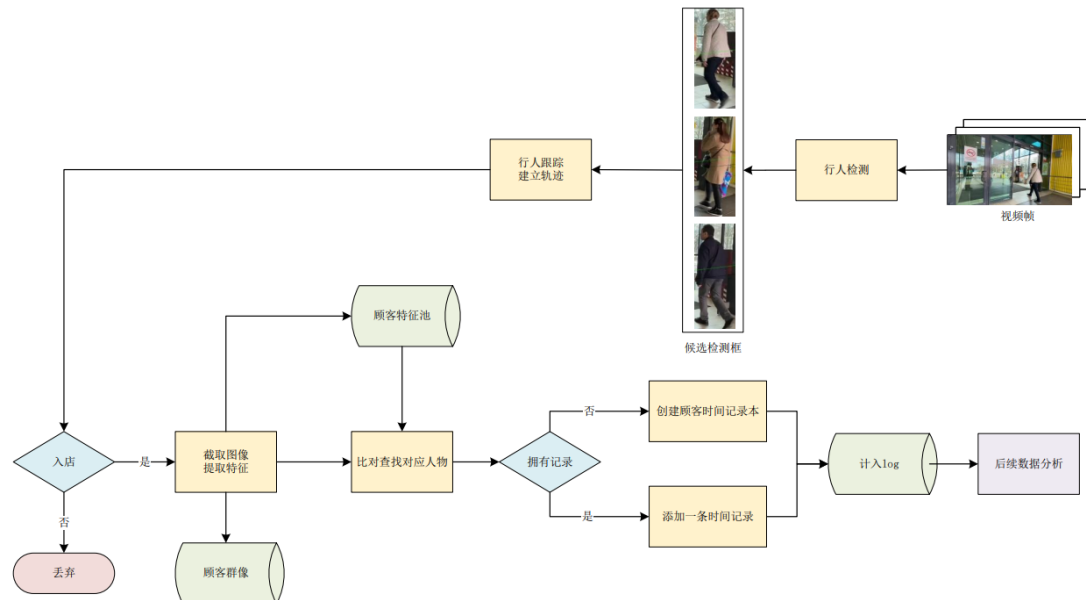
开发环境

实验环境如表所示：

硬件与软件	规格
操作系统	Windows 10 Professional
系统类型	64 位操作系统，基于 x64 的处理器
处理器	AMD Ryzen 5 2600 Six-Core Processor 3.40 GHz
CPU 物理存储	16 GB
GPU 型号	Nvidia GeForce RTX 1050Ti
CPU 内存	4 GB
编译语言	Python 3.8.16
实验框架	Pytorch 深度学习框架
存储地址	本地磁盘

商店客流量统计系统展示与测试

客流量统计算法



测试环境



为了验证本客流量统计系统中各种功能的可行性，本章的测试选在了大型超市 ОКЕЙ 的出入口。超市的平面图和架设的 3 台摄像机布置如上。

跨摄像头行人跟踪功能

单摄像头下跟踪同一人的功能

下面三张图分别选自摄像机 3 下的第 300、第 330、第 360、第 390 帧图像，抽取的时间间隔约 1.5 秒。下图主要显示了对 881 号人物的追踪。





可以看出，在 100 帧内 881 号的人物检测始终良好，并且 ID 始终保持，没有切换的情况。即便在第 3 帧画面出现了遮挡，但在第 4 帧遮挡消失的时候，行人的 ID 仍然保持在 881 号，这就是 FastReID 提供的外观特征关联的作用。此外，对除了 881 之外的其他人物，比如 803,895,869 等，也始终保持了良好的追踪。

测试结果证明了本系统可以在单摄像头下对多个行人进行持续追踪。

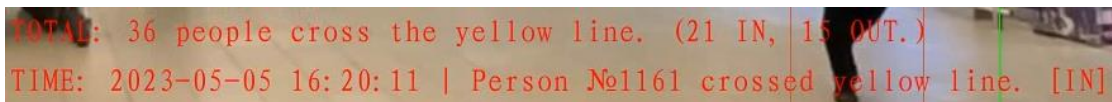
多摄像头下追踪同一人的功能





上图为行人 cus-120 按时间顺序排序依次经过三个摄像头下的跟踪结果。通过测试结果可以看出，当行人第一次出现在摄像头 1 的监控范围时，系统会为其赋予一个身份 ID: cus-120, 此 ID 是全局唯一的。当行人 cus-120 从摄像机 1 的范围内消失，进入摄像机 2 的时候，系统通过 reid 提取其外观特征进行比对，找出了该行人的身份信息 cus-120 并恢复了跟踪。同样的，当她离开摄像头 2 进入摄像头 3 的时候，系统也成功地在摄像头 3 的监控区域内将其识别出来。

当行人通过统计线时，画面的左下角会打印通过时间、进入人员身份、出入总人数的提示。



记录顾客出入时间（打印表格）

PERSON-ID	CAMERA-ID	TIME	CAMERA-ID	TIME	CAMERA-ID	TIME
cus-1	cam-1	2023-05-10 03: 31: 27	cam-2	2023-05-10 03: 37: 59	cam-3	2023-05-10 03: 46: 04
cus-5	cam-1	2023-05-10 03: 31: 37	cam-2	2023-05-10 03: 50: 03		
cus-35	cam-1	2023-05-10 03: 32: 03	cam-2	2023-05-10 03: 38: 10	cam-3	2023-05-10 03: 46: 19
cus-71	cam-1	2023-05-10 03: 32: 11	cam-2	2023-05-10 03: 38: 15	cam-3	2023-05-10 03: 46: 23
cus-82	cam-1	2023-05-10 03: 32: 37	cam-2	2023-05-10 03: 38: 26	cam-3	2023-05-10 03: 46: 35
cus-94	cam-1	2023-05-10 03: 32: 48	cam-2	2023-05-10 03: 38: 32	cam-3	2023-05-10 03: 46: 42
cus-101	cam-1	2023-05-10 03: 32: 53	cam-2	2023-05-10 03: 41: 28		
cus-103	cam-1	2023-05-10 03: 32: 53	cam-2	2023-05-10 03: 41: 39	cam-3	2023-05-10 03: 50: 03
cus-108	cam-1	2023-05-10 03: 33: 00	cam-2	2023-05-10 03: 41: 02	cam-3	2023-05-10 03: 49: 20
cus-120	cam-1	2023-05-10 03: 33: 10	cam-2	2023-05-10 03: 42: 06	cam-3	2023-05-10 03: 50: 23
cus-134	cam-1	2023-05-10 03: 33: 14	cam-2	2023-05-10 03: 42: 30	cam-3	
cus-147	cam-1	2023-05-10 03: 33: 21	cam-2	2023-05-10 03: 42: 57	cam-3	2023-05-10 03: 51: 18
cus-203	cam-1	2023-05-10 03: 34: 15	cam-2	2023-05-10 03: 42: 59	cam-3	2023-05-10 03: 51: 16
cus-253	cam-1	2023-05-10 03: 35: 21	cam-2		cam-3	
cus-248	cam-1	2023-05-10 03: 35: 24	cam-2		cam-3	
cus-257	cam-1	2023-05-10 03: 35: 28	cam-2		cam-3	
cus-306	cam-1	2023-05-10 03: 36: 17	cam-2		cam-3	
cus-314	cam-1	2023-05-10 03: 36: 30	cam-2		cam-3	
cus-331	cam-1	2023-05-10 03: 36: 53	cam-2		cam-3	
cus-335	cam-1	2023-05-10 03: 36: 56	cam-2		cam-3	

我们使用 Python 中的数据结构记录了所有顾客在一段时间内的出入记录。打印出来如

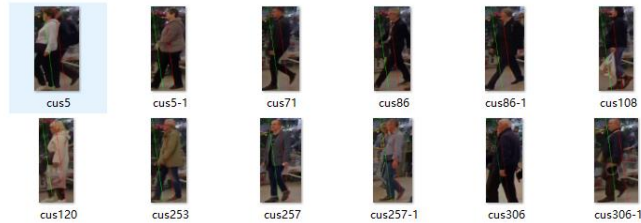
上图所示。

顾客群像

摄像机 1 捕获的顾客图像（部分）：



摄像机 2 捕获的顾客图像（部分）：



摄像机 3 捕获的顾客图像（部分）：



准确性、效率、稳定性

为评估统计方法的效果，本文使用 3 个摄像头对大型超市 ОКЕЙ 进行了视频采样。其中每个视频长度约为 2 分钟。统计测试结果如下表所示：

人员进出计数模块

摄像头	真实值		统计值		精度
	方向	人数	方向	人数	
cam1	进	30	进	30	100.00%
	出	16	出	15	93.75%
	总和	46	总和	45	97.83%
cam2	进	30	进	28	93.33%
	出	23	出	20	86.96%
	总和	53	总和	48	90.57%
cam3	进	28	进	27	96.43%

	出	25	出	25	100.00%
	总和	53	总和	52	98.11%
总计				进入	96.59%
				离开	93.57%
				总共	95.50%

【分析】

这份数据是使用双线法对客流量进行统计得到的，其中包括了三个摄像头的真实值和统计值。可以看到，每个摄像头在进入和离开方向上的精度有所不同，但总体来说，平均精度较高，分别为 93.33%（进入），93.57%（离开）和 93.59%（进入和离开总共）。

在具体分析每个摄像头的情况时，可以发现 cam2 在出口方向上检测的精度较低，只有 86.96%。这与摄像头设置、环境等因素有关。因为在 in2 中，检测人员离开的区域较小。而且在一个时间段内又突然出现了大批人群同时进店和离店的情形（5 秒内 10 个人），所以人员之间相互重叠严重，就出现了 3 个漏检人员，于是拉低了平均精度。

而 cam1、cam3 的精度则相对较高，都在 97% 以上。这是因为在这两个场景下几乎没有大批人群集中进出店的情况。

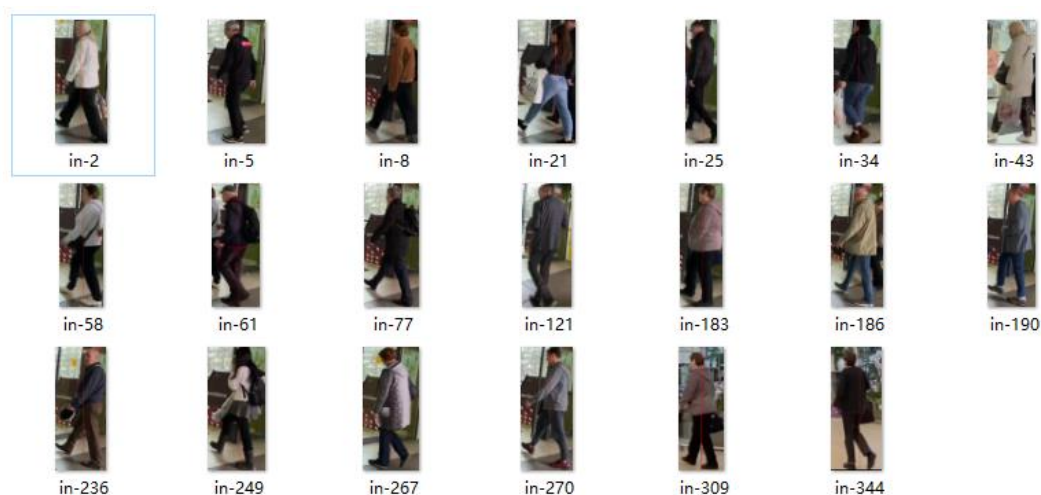
综合 3 个摄像头下的表现，

综合来看，虽然精度还有提升的空间，但整体表现已经比较稳定和准确，可以帮助场所管理者更好地掌握人流动态。

跨摄像头行人重识别模块

本轮，我们总共考核了 20 个人。这 20 个人从摄像机 1 经过摄像机 2 到达摄像机 3，也就是说拥有完整的跨摄像头轨迹。

在 cam1 中出现了 20 个人，形成了 20 张顾客截图，此时特征池里有 20 个外观向量。



图表 1 cam-1 20 张顾客图像

在 cam2 中依次经过了 20 个人，其中只形成 19 张截图，因为存在两个行人并行，互相遮挡，无法正确形成两个检测框的情况。因此出现了一个漏检测。在 19 张待重识别的图像中，18 个人经过重识别得到正确身份，1 人无法获得身份（与原来的身份外貌相似度过低，被过滤成负样本）。因此在 cam-2 下重识别的准确率为 $18/20=90.0\%$



图表 2 cam-2 相互遮挡



图表 3 cam-2：找不到对应的身份（相似度不足）

其真实的身份是：



图表 4 图表 3 中行人对应的真实身份

在 cam-3 中依次经过了 20 人，其中只形成 19 张截图，因为存在两个行人并行，互相遮挡，无法正确形成两个检测框的情况。因此出现了一个漏检测。在 19 张待重识别的图像中，19 张经过重识别得到正确身份，0 张得到了错误身份。因此在 cam-3 下重识别的准确率为 $19/20=95.0\%$



图表 5 cam-3：图中有 3 个人，但只能检测到其中的两个

完整的实验得到如下表格：

行人 ID	cam-1	cam-2	cam-3
2	成功进入统计	成功重识别，得到正确身份	成功重识别，得到正确身份
5	成功进入统计	成功重识别，得到正确身份	成功重识别，得到正确身份
8	成功进入统计	成功重识别，得到正确身份	被遮挡而无法截图
21	成功进入统计	成功重识别，得到正确身份	成功重识别，得到正确身份
25	成功进入统计	成功重识别，得到正确身份	成功重识别，得到正确身份
34	成功进入统计	成功重识别，得到正确身份	成功重识别，得到正确身份
43	成功进入统计	成功重识别，得到正确身份	成功重识别，得到正确身份
58	成功进入统计	成功重识别，得到正确身份	成功重识别，得到正确身份
61	成功进入统计	被遮挡而无法截图	成功重识别，得到正确身份
77	成功进入统计	重识别失败	成功重识别，得到正确身份
123	成功进入统计	成功重识别，得到正确身份	成功重识别，得到正确身份
183	成功进入统计	成功重识别，得到正确身份	成功重识别，得到正确身份
186	成功进入统计	成功重识别，得到正确身份	成功重识别，得到正确身份
188	成功进入统计	成功重识别，得到正确身份	成功重识别，得到正确身份
190	成功进入统计	成功重识别，得到正确身份	成功重识别，得到正确身份
236	成功进入统计	成功重识别，得到正确身份	成功重识别，得到正确身份
249	成功进入统计	成功重识别，得到正确身份	成功重识别，得到正确身份
250	成功进入统计	成功重识别，得到正确身份	成功重识别，得到正确身份
267	成功进入统计	成功重识别，得到正确身份	成功重识别，得到正确身份
270	成功进入统计	成功重识别，得到正确身份	成功重识别，得到正确身份

并且统计了他们进入商店的时间（json 文件）：





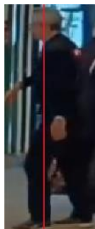
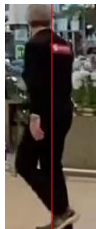


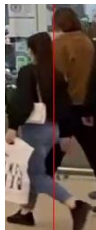
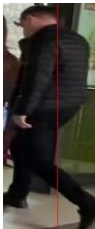



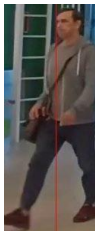
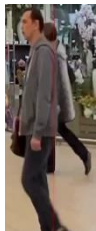
```

1 [{"customer-2": {"in": "2023-05-15 13: 26: 36", "in2": "2023-05-15 13: 33: 20", "in3": "2023-05-15 13: 40: 24"},
2   }, {"customer-5": {"in": "2023-05-15 13: 26: 45", "in2": "2023-05-15 13: 33: 43", "in3": "2023-05-15 13: 42: 01"},
3   }, {"customer-8": {"in": "2023-05-15 13: 26: 55", "in2": "2023-05-15 13: 33: 31"},
4   }, {"customer-21": {"in": "2023-05-15 13: 26: 59", "in2": "2023-05-15 13: 36: 31", "in3": "2023-05-15 13: 45: 21"},
5   }, {"customer-25": {"in": "2023-05-15 13: 27: 00", "in2": "2023-05-15 13: 33: 37", "in3": "2023-05-15 13: 40: 38"},
6   }, {"customer-34": {"in": "2023-05-15 13: 27: 11", "in2": "2023-05-15 13: 33: 47", "in3": "2023-05-15 13: 40: 55"},
7   }, {"customer-43": {"in": "2023-05-15 13: 27: 25", "in2": "2023-05-15 13: 33: 56", "in3": "2023-05-15 13: 41: 01"},
8   }, {"customer-58": {"in": "2023-05-15 13: 27: 28", "in2": "2023-05-15 13: 34: 10", "in3": "2023-05-15 13: 44: 10"},
9   }, {"customer-61": {"in": "2023-05-15 13: 27: 30",
10  }, {"customer-77": {"in": "2023-05-15 13: 27: 39",
11  }, {"customer-123": {"in": "2023-05-15 13: 28: 39", "in2": "2023-05-15 13: 39: 09", "in3": "2023-05-15 13: 42: 44"},
12  }, {"customer-183": {"in": "2023-05-15 13: 30: 16", "in2": "2023-05-15 13: 37: 35", "in3": "2023-05-15 13: 45: 21"},
13  }, {"customer-186": {"in": "2023-05-15 13: 30: 19", "in2": "2023-05-15 13: 37: 51", "in3": "2023-05-15 13: 44: 04"},
14  }, {"customer-188": {"in": "2023-05-15 13: 30: 40", "in2": "2023-05-15 13: 38: 04", "in3": "2023-05-15 13: 43: 25"},
15  }, {"customer-190": {"in": "2023-05-15 13: 30: 26", "in2": "2023-05-15 13: 37: 00", "in3": "2023-05-15 13: 43: 09"},
16  }, {"customer-236": {"in": "2023-05-15 13: 31: 37", "in2": "2023-05-15 13: 38: 30", "in3": "2023-05-15 13: 44: 41"},
17  }, {"customer-249": {"in": "2023-05-15 13: 31: 56", "in2": "2023-05-15 13: 39: 02", "in3": "2023-05-15 13: 47: 47"},
18  }, {"customer-267": {"in": "2023-05-15 13: 32: 27", "in2": "2023-05-15 13: 45: 59", "in3": "2023-05-15 13: 48: 49"},
19  }, {"customer-270": {"in": "2023-05-15 13: 32: 32", "in2": "2023-05-15 13: 45: 57", "in3": "2023-05-15 13: 53: 03"}
20  ]
21

```

成功重识别的人员共（部分表格）：

ID	Cam-1	Cam-2	Cam-3
----	-------	-------	-------

2			
5			
21			
25			
270			

实验现象：观察到以上现象：在一段时间内有 20 个行人通过 cam-1, cam-2, cam-3，其中 17 个人被完整地跟踪，拥有在 3 个摄像头的记录，2 个人因为被其他行人遮盖而无法被检测到，1 个人重识别识别，因此他们只拥有不完整的 2 个摄像头下的时间记录，不计入完整跟踪当中。

实验结论：我们的客流量统计系统的跟踪精准度达到 $17/20=85.0\%$ 左右。

ГЛАВА 4 Анализ полученных результатов （结果分析）

ReID 模型在其他数据集上的测试结果

在 Market1501、DukeMTMC 数据集上的测试结果

在完成了模型的训练之后，我们又选择在另一个数据集 Market1501、DukeMTMC 上测试我们的模型性能，并与同时期出现的其他 reid 模型做了横向对比：

Methods	Market1501		DukeMTMC	
	Rank-1	mAP	Rank-1	mAP
IANet(IVPR'19)	94.4	83.1	87.1	73.4
Auto-ReID(ICCV'19)	94.5	85.1	-	-
OSNet(ICCV'19)	94.8	84.9	88.6	73.5
ABDNet(ICCV'19)	95.6	88.3	89.0	78.6
Circle Loss(CVPR'20)	96.1	87.4	89.0	79.6
ours	95.7	88.4	90.1	81.3

根据上面提供的数据，我们可以看到 FastReID 模型在 Market1501 和 DukeMTMC 两个数据集上都表现出色。其在 Market1501 数据集上的 Rank-1 精度为 95.7%，mAP 为 88.4%，在 DukeMTMC 数据集上的 Rank-1 精度为 90.1%，mAP 为 81.3%。

相比于其他算法，FastReID 取得了很好的性能表现。例如，在 Market1501 数据集上，FastReID 的 Rank-1 精度和 mAP 分别超过了 IANet、Auto-ReID、OSNet 等算法；甚至在 DukeMTMC 数据集上，它的 Rank-1 精度和 mAP 超过了其他所有的算法。这说明 FastReID 在行人重识别任务上具有很强的鲁棒性和泛化能力。

总之，基于这些评估指标，我们可以认为 FastReID 是一种有效的行人重识别模型，有望应用于实际场景中。

DeepSORT 模型的评价指标

在 MOT20 数据集上的测试结果

为了验证基于 fastreid 改进的 deepsort 多目标跟踪方法的有效性，我们在又在更具挑战性的新数据集 MOT20 上展开了对比实验。

该数据集使用静态和移动相机拍摄的不受约束的环境中注释了 8 个具有挑战性的视频序列（4 个训练，4 个测试）。相比于此前的 MOT Challenge 系列数据集，MOT20 关注人群密集的场景，其视频最多可达单帧 246 人。所以跟踪显得更加困难。它比较考验多目标跟踪模型对小目标的追踪能力。



图表 6 MOT20-01 截取片段

我们将多目标跟踪算法放到在 MOT20 数据集中四个子集上进行了测试，并得到的结果与世界范围的 MOT Challenge 比赛中的前 10 名优胜者作比较，结果如下表：

Tracker	MOTA ↑	IDF1↑	MT/% ↑	ML/% ↓	IDs ↓	FPS ↑
ByteTrack	67	70.2	47.7	21.2	680	17.7
OUTrack	65.4	65.1	49.5	13.3	2,885	5.1
UTM	64.4	65.9	65	8.7	2,592	22.4
RFTracker	62.4	53	48.7	15.5	3,804	0
SUSHI	61.6	71.6	47.6	19.2	1,053	5.3
TransCenter	61	49.8	48.4	15.5	4,493	1
MPTC	60.6	59.7	51.1	16.7	4,533	0.7
TMOH	60.1	61.2	46.7	17.8	2,342	0.6
OCSORT	59.9	67	38.5	26.6	554	27.6
MFI	59.3	59.1	41.1	17.3	191	0.5
ours	60.4	60.9	35.2	17.4	2021	8.6

首先，需要说明的是在 MOT20 数据集上，跟踪器的表现被衡量为 MOTA、IDF1、MT 和 ML 等指标。其中，MOTA 是多目标跟踪的主要评估指标，它综合了检测、匹配和跟踪的

精度和召回率。IDF1 是一种更加注重跟踪器的准确性的指标，MT 和 ML 分别代表正确跟踪的目标数量占总数的比例和漏跟踪目标数量占总数的比例，IDs 表示错误标识符数量，FPS 表示每秒能够处理的帧数。

从数据可以看出，ByteTrack 表现最好，其次是 OUTrack 和 UTM，这三个跟踪器的 MOTA 均超过了 64。其他跟踪器的表现相对较差，但也有一些跟踪器在某些指标上表现突出。例如，SUSHI 和 OCSORT 在 IDF1 和 MOTA 上得分较高，但是在 ML 方面表现较差，而 TMOH 则在 IDF1 上表现较好。

我们的跟踪器对比其他 10 名世界范围内的优胜者，有如下的优点和缺点：

优点：

1. 该跟踪器在 IDF1 指标上表现不错，得分 60.4，说明其基本能够完成多目标跟踪任务。
2. 该跟踪器在 FPS 方面表现也比较良好，达到了 8.6 帧/s，这意味着它可以处理高速动态场景。

缺点：

1. 在 MOTA 指标上，该跟踪器只有 60.4 的得分，说明它在数据集 MOT20 的表现上相对较弱。
2. ML/%指标较高，为 17.4，说明它对于漏跟踪目标数量占总数的比例也没有很好的控制。
3. 该跟踪器在 IDs 两个指标上的表现都不如其他一些跟踪器，IDs 为 2021，MT/%为 35.2，这说明它存在一定的误检、漏检和误跟踪等问题。

虽然我们的跟踪器在 FPS 和 IDF1 指标上表现不错，并且具有一定的实时性，但在正确跟踪目标数量占总数的比例和漏跟踪目标数量占总数的比例上表现相对较差。经过分析，我认为表现差和 YOLO 目标检测算法有关。我们的 YOLO 检测算法属于一阶段的目标检测算法，优点是运行速度很快，节省计算资源。而缺点也非常明显：位置精确性差，对于小目标物体以及物体比较密集的也检测不好，比如一群小鸟，一大群密集人群。而在前 10 名中，除了 OCSORT、ByteTrack 和 UTM 之外，都是用了二阶段的目标检测算法，这保证了他们针对小目标也有较高的目标检测质量。

因此我认为，如果后续要提升模型对小目标的追踪能力，可以从 YOLO 入手，有以下策略：

- 调整模型结构：可以考虑使用更深的网络结构，以提高模型的识别能力。
- 多尺度检测：可以设计多尺度检测策略，对不同大小的目标进行识别和检测，提高模型的检测精度。例如在 YOLO 网络的颈部（neck）增加 FPN（Feature Pyramid Networks）结构，将顶层特征和底层特征融合，以提高网络检测不同尺度目标的能力。

ЗАКЛЮЧЕНИЕ（结论）

参考文献