上海交通大學

SHANGHAI JIAO TONG UNIVERSITY

学士学位论文

THESIS OF BACHELOR

论文题目：面向海量数据的高效模糊搜索

学生姓名：刘韵

学生学号：5140309455

专　　业：计算机科学与技术

指导教师：沈耀

学院(系)：电子信息与电气工程学院

上海交通大学
学位论文原创性声明

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

学位论文作者签名：_____

日　期：_____年 _____月 _____日

上海交通大学
学位论文版权使用授权书

本学位论文作者完全了解学校有关保留、使用学位论文的规定，同意学校保留并向国家有关部门或机构送交论文的复印件和电子版，允许论文被查阅和借阅。本人授权上海交通大学可以将本学位论文的全部或部分内容编入有关数据库进行检索，可以采用影印、缩印或扫描等复制手段保存和汇编本学位论文。

本学位论文属于

保　密 □，在 _____ 年解密后适用本授权书。

不保密 □。

（请在以上方框内打 ✓）

学位论文作者签名：_____　　指导教师签名：_____

日　期：_____年 _____月 _____日　　日　期：_____年 _____月 _____日

面向海量数据的高效模糊搜索

摘　要

模糊搜索是一类搜索相同或相似数据的搜索方法。本文针对无感知环境下的大规模人脸数据的模糊搜索，从人脸检测，特征提取与向量匹配三方面进行研究，最终实现了完整的无感知人脸识别系统。

无感知人脸识别系统是一种不依赖待检测者主动配合的人脸识别系统。这种系统利用监控摄像头的实时画面，对其中的人脸进行检测、计数和识别。在教室点名、公司打卡签到等实际场景中有广泛的应用。由于不依赖于被检测人的配合，人脸的图像会呈现偏转、模糊和遮挡等特征，这对检测与识别算法都提出了非常高的要求。

深度学习在人脸检测和特征提取方面有着广泛的应用。本文选取 SSH 检测器[1] 进行人脸检测，InsightFace[2] 进行特征提取。我们还从实际监控摄像中提取数据，建立了无感知人脸检测的测试集 CFDDB。并在 CFDDB 上定义了基准参数，以分析不同人脸检测算法的优劣。

大规模特征向量的匹配是人脸模糊搜索需要解决的另一个问题。本文讨论基于局部敏感哈希与基于有向图的近似搜索算法进行高效的向量匹配，同时给出基于局部敏感哈希匹配算法的优化思路。

无感知人脸识别系统的实现充分考虑了数据处理流程、设备网络拓扑结构、以及系统逻辑架构等的设计，从而使系统具有稳定和可扩展的特性。本文在最后对这部分内容进行了阐述。

关键词： 模糊搜索　无感知人脸检测　大规模数据　向量匹配　深度学习

EFFICIENT FUZZY SEARCH OF LARGE-SCALE DATA

ABSTRACT

Fuzzy search is a type of search method that searches for the same or similar data. This paper focuses on the fuzzy search of large-scale face data, from the aspects of face detection, feature extraction and vector matching, and finally implements a complete imperceptible face recognition system.

Imperceptible face recognition system is a face recognition system which does not require intentional cooperation from users. This system pulls live images of faces from surveillance cameras at first. Then it performs face detection, face counting and face recognition to get face embeddings. Finally, it matches face embeddings to database contents. This system can be widely used for student attendance management and access control in companies. Since this system does not rely on users' cooperation, the facial images received would be angled, blurred and occluded, which makes detection and recognition very challenging.

Deep learning has a wide range of applications in face detection and feature extraction. In this paper, we use SSH detector[1] for face detection and InsightFace[2] for feature extraction. We also pull data from surveillance cameras and present an imperceptible face detection benchmark CFDDB. The benchmark parameters are defined to analyze the performance of different face detection algorithms.

The matching of large-scale feature vectors is another challenge that needs to be solved for approximate search of face data. In this paper, efficient vector matching algorithms based on LSH or directed graph are discussed. At the same time, the optimized method for algorithm based on LSH is given.

We present our system's data architecture, physical architecture and logical architecture at the end. This architecture design keeps imperceptible face recognition system efficient, stable and scalable.

KEY WORDS: fuzzy search, imperceptible face detection, large-scale data, vector matching, deep-learning

目 录

插图索引

表格索引

算法索引

主要符号对照表

$recall$	召回率
$errorrate$	误判率
α	检测参数
$\mathbf{q}, Q$	查询向量
$\mathbf{p}$	最邻近向量
L	Loss 函数
$\mathcal{F}, \mathcal{G}$	LSH 函数族
h, g	哈希函数
d	距离函数
$\mathcal{M}$	矩阵空间

第一章 绪论

1.1 设计背景和应用场景

模糊搜索是一类搜索相同或相似数据的搜索方法。大规模人脸数据的模糊搜索，指的是给定人脸数据返回其在大规模数据集中多个可能匹配项的过程，是人脸识别系统的核心。其实现涉及人脸检测，特征提取与向量匹配三方面。

而本文中的人脸识别系统为无感知条件下的人脸识别系统。其中无感知条件，指的是不依赖于用户主动配合即可完成人脸检测和识别的系统。而目前常见的人脸识别系统在使用时往往需要用户主动配合，如需要用户将面部对准摄像设备的中心区域，或者需要用户调整特定的角度以获得多角度的人脸信息。

无感知人脸识别有丰富的实际应用，其中最典型的应用场景有三个。

第一是教室点名。目前课堂点名的主要的方法有签到和教师口头点名两种方式，但是这两种方式都有很大的缺陷。首先，在大学课堂这种学生流动性较大的场景，教师很难记住每个学生的长相和姓名，如果一名学生实际未到，而请其他同学代为签到或者口头答道，教师很难分辨。其次，课堂时间是有限而宝贵的，如果同一课堂的人数较多，签到点名的时间是不可接受的。最后，签到点名无法调查学生在课堂上的学习状况。于此相比，使用无感知人脸识别系统不仅可以节约签到点名的时间，而且可以避免作弊现象。如果可以引入一些衡量标准例如学生的抬头率，抬头听课的时间等，则还可以调查学生在课堂上的学习状况，方便学校和教师调整教学策略。

第二是公司签到。目前已经有部分公司采用了人脸识别系统进行打卡签到，但是这种方法依然存在一些不足。这类打卡签到系统需要每一位员工使用打卡机，而当上下班高峰期时，会有非常多员工同时使用打卡机，从而需要排队等待。而使用无感知人脸检测则不仅可以避免排队等待，而且可以实时的掌握公司内部的人员信息。

第三是展会安保。大型展会和演出往往伴随者大量的人流，对集中出现的大量人流使用有感知的人脸检测逐一排查不仅耗时长，还容易加剧现场的拥堵现象。而监控室的工作人员也无法做到同时监控多路摄像头获取的视频数据。此时使用无感知人脸检测系统不仅可以避免现场的拥堵，而且可以做到实时筛选可能造成危险的人员，减轻会场安保系统的负担。

1.2 研究的现状和研究成果

1.2.1 人脸的检测与特征提取

人脸检测是指将人脸图像区域从一张图片中标注出来的过程。由于不同图片中人脸的姿势、大小和光照条件完全不同，使得这一过程非常具有挑战性。

早期的想法是请专家设计一些特征分类器，然后将一张图片的每一块区域都使用特征分类器进行分类，进而区分出人脸区域和非人脸区域。而这种人工设计分类器的分类效果并不能令人满意，因此研究者们开始利用机器学习的算法来训练分类器。这种方法首先需要寻找一类区分度强的，而且不易被光照强度等外界环境干扰的特征。然后将某一个训练数据集的图片分为含有人脸和不含人脸

两种集合，分别从两种集合上提取某类特征。最后将区分效果较好的特征组合起来，得到完整的人脸特征分类器。使用的时候，将图片的每一个区域使用特征分类器进行分类，将分类的结果组合后得到图片中的人脸区域。

经过机器学习得到的分类器比起人工设计的分类器效果更好，其中的代表有基于 HAAR 特征级联分类器[4] 和基于 HOG 特征的分类器[5]。这两种分类器直到现在依然在一些计算机视觉的开源工具包中被使用。

随着时间的推移，深度学习在计算机视觉领域开始大放异彩。由于深度神经网络有着非常强的表征能力和学习能力，使用深度学习的人脸检测算法在检测精度上迅速超过了传统的方法。其中具有代表性的基础网络结构包括 VGG 系列网络[6]，ResNets 系列网络[7] 等。而基于这些网络，研究者们针对不同场景下不同大小的人脸不断进行优化，在提升精度的同时提升运算速度，训练出了非常高效的人脸检测器。而我们尤其关注针对小脸检测进行优化的检测器，其代表为 SSH 检测器[1] 和 Tiny face 检测器[8]。

关注这两个基于卷积神经网络的检测器是因为它们在非受控环境下的人脸检测测试集上取得了非常好的测试结果。我们希望通过实验来探究这两个检测器在无感知的人脸检测环境下是否依然有足够高的召回率与较低的计算代价。

人脸检测技术的飞速发展的同时，研究者们为了衡量算法的优劣，建立了多个公开的数据集。其中有代表性的有 LFW 数据集[9]、WIDER FACE 数据集[10] 等。这些数据集中大多会区分训练集和测试集两种子集。其中 WIDER FACE 数据集还包括了人脸大小、姿势、遮挡、表情、妆容、光照条件等标记，被许多人脸检测算法广泛使用。而这些数据集中，并没有针对无感知人脸检测所设计的数据集。因此，我们自己建立了符合无感知人脸检测这一应用场景的数据集 CFDDB。

在这个数据集中，我们使用了从教室摄像头获取的人脸图像，以期最接近实际使用场景。同时请几位志愿者进行人工标注并纠错，确保数据集中标记的准确性。同时我们在设计时充分考虑了数据集的可扩展性，使得数据集中的标记内容可以非常容易的获得扩充。

目前数据集的规模并不大，作为训练集使用容易出现过拟合等现象。因此我们将整个数据集用作测试集，并定义了测试人脸检测算法的多种参数，以衡量不同的人脸检测算法在执行无感知人脸检测时的优劣。

通常而言，检测出来的人脸区域并不能直接用于识别。为了保证识别的准确性，普遍的做法时找到检测算法得到的人脸区域的多个特征点，然后根据特征点的位置对人脸进行姿势校正。MtCNN 网络[11] 不仅能够用于人脸的检测，而且在同时可以识别出人脸的多个特征点，被广泛地运用于人脸姿势的校正中。

特征提取的过程在这里被定义为从检测到的人脸区域产生人脸特征向量的过程。产生人脸特征向量的方法多种多样，其中使用深度网络提取人脸特征的 InsightFace[2] 具有极高的精确性，在一些公开测试集上的精度达到了世界先进水平。

1.2.2 人脸特征向量的匹配

特征提取的结果为从人脸图像区域提取出的特征向量，想要将人脸与身份绑定，则需要匹配数据库中的人脸特征与检测得到的人脸特征。而根据实际应用场景的不同，数据库中的数据会有不同的规模。

针对小规模数据，线性搜索和多类支持向量机就有较好的匹配效果。而匹配大规模数据则需要依靠近似搜索的方法，以牺牲一定的精度与空间为代价获取较高的搜索效率。

目前广泛使用的近似搜索算法有基于有向图、基于 kNN 图、基于局部敏感哈希和基于矢量量化等多种类型的方法，每个算法在时间效率和空间效率方面各有千秋。

1.3 设计思路

在无感知条件下，人脸信息的获取主要依赖于监控摄像头提供的视频流。由于不依赖于用户的主动配合，从视频流中抽取到的人脸区域往往是大角度、有遮挡的。如果使用全景视频流，人脸区域往往非常小，一帧图片中含有的人脸数目往往较多。为了解决这些问题，我们采用了以下方法：

首先是频繁地从视频流中截取关键帧，这样做的目的是保证我们从视频流中提取足够的样本，从而最大限度的减少人脸的漏检。而这种方法则对后端服务器会产生极大的压力，不仅要求服务器的带宽足够支持上千路图像的不间断输入，而且要求在服务器上运行的算法速度足够快从而保证等待队列不会溢出。

其次需要选择优秀的人脸检测算法，在计算效率足够高的同时保证能够对大角度、有遮挡的人脸有较高的检测效率并尽量避免将非人脸图像区域误判为人脸区域。

再次需要选择优秀的人脸识别算法，在人脸图像区域不完整、有噪声的情况下能够准确快速的提取人脸的特征，并同数据库进行交互判断人脸身份。

最后需要摄像头不仅能够提供全景的图像信息，而且能够以某一预先设定好的顺序将某个固定空间分割成多个子空间提供每个子空间的视频流。

针对大规模人脸数据库，还需要考虑使用怎样的近似搜索算法快速在数据库中匹配人脸特征。

1.4 设计工具

在计算机视觉领域的发展过程中，很多优秀的工程师开发了丰富的开源工具，其中应用最为广泛的就是 OpenCV[12]。OpenCV 最初由 Intel 公司组织开发，现在已经演化成为了一个跨平台的开源计算机视觉实时演算工具包。OpenCV 功能强大，已经将图片读写、图片格式转换、HAAR 特征级联分类器等常用功能封装成类或者类函数，而且底层使用 C++ 实现，有着优秀的计算性能。于此同时，其还实现了 Python 脚本语言的调用接口，简单易用，因此被本系统的图像处理部分使用。

目前多数利用深度神经网络的算法需要一定的框架支持，而本系统中使用了 Caffe[13] 和 TensorFlow[14] 两种开源框架。

Caffe 由 Berkeley Vision and Learning Center 和众多社区贡献者共同开发，以表达式、速度和模块化为核心的深度学习框架。本系统中使用卷积神经网络的 SSH 检测器就使用了这种开源框架。由于 Caffe 支持使用 GPU 加速，使得 SSH 检测器的时间性能大大提升。但是 Caffe 缺乏一套成熟的部署方案而且对嵌入式设备的支持力度较小，给系统的架构设计带来了一定的限制。

TensorFlow 最早是由 Google Brain 团队开发的一个使用数据流图进行数值计算的深度学习开源框架。TensorFlow 以数据流图作为核心，图中的节点代表数学运算，而图中的边则代表在这些节点之间传递的多维数组。本系统中 MtCNN[11]、Tiny face 检测器[8] 与 InsightFace[2] 均使用了这种框架。TensorFlow 支持使用 GPU 进行加速运算，同时支持使用 TFserving 和 Docker 容器进行规模化部署和

更新。针对嵌入式设备，TensorFlow 提供了 TensorFlow Lite 优化深度网络在移动设备、嵌入式设备上的运行效率。使用 TensorFlow 框架，不仅提升了运算速度，而且为系统的架构设计和部署都带来了很大的方便。

深度学习框架进行运算加速往往依赖于 GPU，而我们在这个系统中使用了 Nvidia 公司开发的通用并行计算架构 CUDA[15]。这个架构赋予了 GPU 解决复杂计算问题的能力，尤其是图像处理中常见的矩阵运算，因此提高了整个系统的运算效率。

CuDNN[16] 是 Nvidia 公司推出的针对深度神经网络的 GPU 加速库，Caffe 和 TensorFlow 等深度学习框架的加速都依赖于它。CuDNN 对常见的深度神经网络中的操作使用 CUDA 进行了精细优化，使得研究者们可以专心构建网络而不用花费大量时间优化网络。

1.5 研究内容与主要工作

研究工作主要分为四个阶段，每阶段的主要研究内容和成果如下：

1.5.1 数据集设计与标注

这一阶段共有五项研究内容。第一，根据无感知人脸识别的场景，从监控数据中抽样，得到测试集图片。第二，标注每一张图片的人脸位置与人脸标签并对标注进行纠错和预处理。第三，建立测试集 CFDDB，分析 CFDDB 中的人脸属性，定义合理的测试参数，制定人脸检测算法的测试策略。第四，建立人脸特征向量数据库，制定人脸识别算法的测试策略。第五，分析改进 CFDDB 的方法，设计扩充 CFDDB 的流程。

完成这一阶段的研究工作后，我们发现 CFDDB 中的非正脸图像占比超过 70%，有遮挡人脸占比接近 60%，这给后续优化人脸检测与识别算法带来了极大的挑战。

1.5.2 人脸检测与特征提取算法的测试与调参

这一阶段共有五项研究内容。第一，封装基于 HAAR 特征的级联分类器[4]，实现基于 CFDDB 的测试接口，并根据召回率与时间效率分析其性能。第二，封装 MtCNN[11]，使用 GPU 服务器对其进行加速，并根据其深度网络的结构与功能分析其性能。第三，测试 SSH 检测器[1]，实现基于 CFDDB 的测试接口，并进行参数调优。第四，测试 Tiny face 检测器[8]，实现基于 CFDDB 的测试接口，并于 SSH 检测器调优后的结果进行对比。第五，测试高精度的特征提取算法 InsightFace[2]，确保其在 CFDDB 上有较高的召回率。

完成本阶段工作后，我们取得了如下三点研究成果。首先，基于 HAAR 特征的级联分类器与 MtCNN 召回率过低，不符合要求。其次，Tiny face 检测器时间效率过低，不符合要求。最后，即使不考虑时间效率，SSH 检测器在综合考虑召回率与误判率的条件下依然优于 Tiny face 检测器，因此选择 SSH 检测器作为人脸检测模块的核心算法。

1.5.3 匹配算法的设计与优化

这一阶段共有两项研究内容。第一，设计小规模数据集的匹配算法，分析其时间效率与空间效率。第二，设计大规模数据集的匹配算法，选择合理的优化策略。

我们发现，在小规模数据集上利用人脸特征向量的性质，使用线性搜索或多类支持向量机都可以获得较好的匹配效果。而如果数据规模过大，则可以考虑使用基于多探针的局部敏感哈希与基于有向图的近似搜索算法进行高维向量匹配。

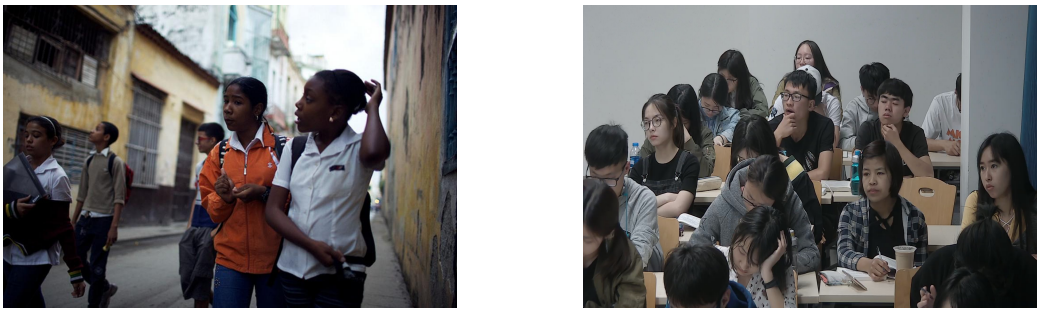
1.5.4 无感知人脸识别系统的架构设计

这一阶段的研究内容主要是设计良好的系统架构，保证系统高效稳定运行的同时具有可扩展的特性。

在本阶段，我们设计了系统的数据处理流程、设备网络拓扑结构与逻辑架构，并在 GPU 服务器上实现了无感知人脸识别系统。

第二章 CFDDB 测试集

在选择正确的算法之前，我们应当选择或者建立一套测试集，以检测算法执行的效果。在人脸检测的领域，已经有数量众多的公开数据集用于训练和测试，常见的有 WIDER FACE[10]，FDDB[17]，LFW[9] 等等。然而如图2–1所示，这些数据集中的人脸大多是欧美人，而且每张照片中的人脸数量也有限。而我们从监控录像中提取的图片亚洲人偏多，人脸往往是模糊的，大角度的，或是被遮挡的。因此，现有的数据集无法满足我们的测试需求，据此我们建立了自己的数据集 CFDDB(Classroom Face Detection Database)。



(a) WiderFace 测试集样例图片　　(b) CFDDB 测试集样例图片

图 2–1 WIDER FACE 与 CFDDB 样例图片对比图

Figure 2–1 Sample picture comparision between WIDER FACE and CFDDB

2.1 CFDDB 测试集图片标注规则

测试集图片由从一段 15 分钟的监控录像中每隔 100 帧截取一帧产生，共有 149 张图片。标注人需要在一张图片中使用矩形框标注全部可识别人脸。标注的矩形框应当以人脸的鼻子为中心，矩形框区域面积应当大概是人脸区域面积的两倍。图2–2为 CFDDB 中一张图片的标注可视化之后的效果，其中，绿色的矩形框为标注人标注的人脸区域。

矩形框标注完毕之后需要给每一个矩形框添加一个标签，这个标签是人脸的全局名称。标签添加完毕之后，将一张图片中所有矩形框的位置信息和标签信息写入与图片同名的 XML 文件进行保存。

为了保证视频中每位老师和同学的隐私，我们给每张人脸添加的全局标签均为一定规则生成的独特的数字组合而不是姓名，以保证数据集不被滥用。

2.2 CFDDB 测试集的纠错与预处理

测试集的标注是人为进行的，有可能会出现差错，所以在将标注信息加入测试集之前，还需要对标注的信息进行认真反复的比对，以确保标注信息的正确性。在本次比对中，共发现六处错误。修

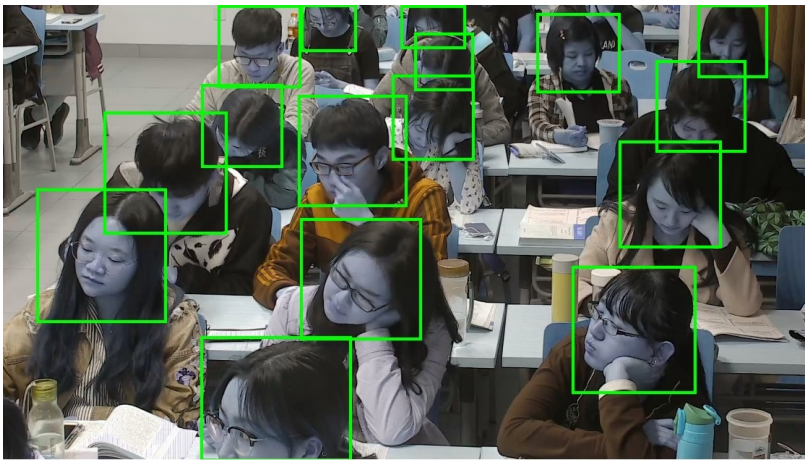


图 2–2 CFDDB 标注示例

Figure 2–2 A lable example from CFDDB

改所有错误之后，测试集中共有可用图片 134 张，人脸 1087 个，总人数 84 人。

测试集的预处理分为四步。第一步是将所有人脸图像区域切割出来，并将相同的人脸放到以该人脸的标签为名的文件夹下。这么做的目的是方便建立之后的识别数据库，而且通过对每个文件夹中的内容的比对，可以再次检验人为标注是否准确。

第二步是对齐所有人脸。由于人为标注的人脸有大有小，所以需要将所有截出的人脸图像对齐到相同大小。CFDDB 在建立时保持了原有人脸的长宽比，不足的地方以空白图像补全，之后使用线性插值的方式将所有人脸缩放到了宽 150 像素，高 150 像素的大小。

第三部是将截取出的所有人脸进行人工筛选。对于每一张图像，如果其人脸露出面积占总图像面积不超过 30% 或者原始人脸大小的高度小于 10 像素的区域，则将其标记为忽略。

第四步是将所有对齐的人脸进行预识别，得到特征向量。CFDDB 在建立时使用了 MtCNN[11] 进行校正，使用 InsightFace[2] 提取特征向量。最后，每一张截出的人脸图像都转化为了一个 256 维的特征向量。

最后一步是对所有抽取出的特征向量进行聚类，使用算法2–1计算出每个类的类中心作为该人脸的标准特征向量，并保存到以该人脸标签为文件名的 Numpy 向量文件中。

算法 2–1 求类中心

输入： $Array$ 类成员向量数组，n 类成员个数

输出： $center$ 类中心向量

```
1: function CalcAverage(Array, n)
2:     result ← 0
3:     for i = 0 → n − 1 do
4:         result ← result + Array[i]
5:     end for
```

```
    return result
end function

function NORMALIZE(vector)
    l ← 128
    norm ← 0
    for j = 0 → l − 1 do
        norm ← norm + vector[j] ∗ vector[j]
    end for
    norm ← √norm
    norm ← vector/norm
    return norm
end function

function CALCCENTER(Array, n)
    center ←CLACAVERAGE(Array, n)
    center ←NORMALIZE(center)
    return centor
end function
```

2.3 CFDDB 测试集的使用

CFDDB 测试集既可以用来测试无感知人脸检测算法的准确性，又可以粗测无感知人脸识别算法的准确性。CFDDB 中的所有内容都可以在 CFDDB 的 Github 主页[1]中找到，主页中还包含了针对不同算法检测的示例程序。下面介绍如何使用 CFDDB 数据集：

2.3.1 使用 CFDDB 测试人脸检测算法

下述检测方法中 *ImageList* 表示 CFDDB 测试集的图像列表，*GroundTruth*[*image*] 表示对应 *image* 图片的人为标注区域列表，*Detect*[*image*] 表示对应 *image* 图片使用人脸检测算法得到的区域列表，*TargetIOU* 表示人为设定的标定区域和检测区域的交叠率。检测方法如下：

1. 从 *ImageList* 中取出待检测的图像 *image*。
2. 调用待测的检测算法得到 *Detect*[*image*]，同时读出人为标注信息 *GroundTruth*[*image*]。
3. 计算 *Detect*[*image*] 中每一项与 *GroundTruth*[*image*] 中每一项的交叠率，如果最大的交叠率超过了预设的 *TargetIOU* 则认为成功的检测到了一张人脸。
4. 计算所有检测到的人脸数目，并将检测错误的信息输出分析检测算法存在的缺陷。

[1] https://github.com/liuwuliuyun/CFDDB

2.3.2 使用 CFDDB 测试人脸识别算法

下述检测方法中 $FaceImage$ 表示待检测的人脸图片区域，$GroundTruth[FaceImage]$ 表示人为标注的人脸编号，$Recognize[FaceImage]$ 表示识别得出的人脸向量，$FaceList$ 表示经过预处理后得到的每一个人脸编号对应的类中心向量。检测方法如下：

1. 取出待检测人脸区域的人为标注编号 $GroundTruth[FaceImage]$。
2. 调用人脸识别算法得到 $Recognize[FaceImage]$。
3. 计算 $Recognize[FaceImage]$ 到 $FaceList$ 中每一项的欧氏距离，将距离最小的一项对应的标签与 $GroundTruth[FaceImage]$ 比对，如果相同，认为识别正确；如果不同，认为识别错误。
4. 计算所有识别正确的人脸数目，并将错误信息输出分析识别算法存在的缺陷。

2.4 CFDDB 测试参数定义

以下三个参数均用于测试人脸检测算法的准确性，其中参数 α 为结合召回率和误判率两种参数的综合测试指标。

CFDDB 测试集中全部的人脸数量为 $face_{all}$，检测正确的人脸数量为 $face_{right}$，定义召回率 $recall$ 为：

$$recall = \frac{face_{right}}{face_{all}} \tag{2–1}$$

CFDDB 测试集中全部的人脸数量为 $face_{all}$，检测错误的人脸数量为 $face_{wrong}$，定义误判率 $errorrate$ 为：

$$errorrate = \frac{face_{wrong}}{face_{all}} \tag{2–2}$$

CFDDB 测试集中召回率为 $recall$，误判率 $errorrate$，定义检测参数 α 为：

$$\alpha = \frac{1}{recall} + 10 \cdot errorrate \tag{2–3}$$

2.5 CFDDB 测试集属性分析

CFDDB 测试集由于专注于课堂教室这个特殊的场景，适用于对教室点名的人脸识别系统进行测试，整体难度与 WIDER FACE[10] 的 Hard 测试集相当。但是，CFDDB 又与 WIDER FACE 数据集在人脸大小、人种等属性上有着明显的差别。下面我们对 CFDDB 测试集做属性分析。由于 CFDDB 测试集是从视频流中截取的，因此我们采用平均取样的方法，从测试集中每隔 20 张图片选取一张作为属性分析的样本。

2.5.1 大小

我们根据人脸区域的高度将人脸分为三种大小：较大人脸，中等人脸和较小人脸。这里沿用 WIDER FACE 数据集中的定义，将较小人脸定义为高 10 像素到高 50 像素的人脸；将中等人脸定义为高 50 像素到高 300 像素的人脸；将较大人脸定义为高度超过 300 像素的人脸。

2.5.2 肤色

与 WIDER FACE 数据集中多种肤色的人脸不同，CFDDB 中的人脸全部是肤色为黄色的亚洲人。未来可能添加包括多种肤色的图片以提供不同测试环境下的需求。

2.5.3 遮挡

遮挡在评估人脸检测算法中的表现中有着不可替代的作用。这里同样沿用 WIDER FACE 中的定义，将部分遮挡定义为人脸被遮挡的区域占全部人脸区域的 1% 至 30%；严重遮挡定义为人脸被遮挡的区域占全部人脸区域超过 30%。

2.5.4 姿势

这里类比 WIDER FACE 中姿势的定义。若满足一下两中条件中的任意一种，则一张人脸被定义为非正脸。条件一是俯仰角超过 30 度。条件二为偏航角超过 90 度。若两个条件都不满足，则该人脸区域被定义为正脸。

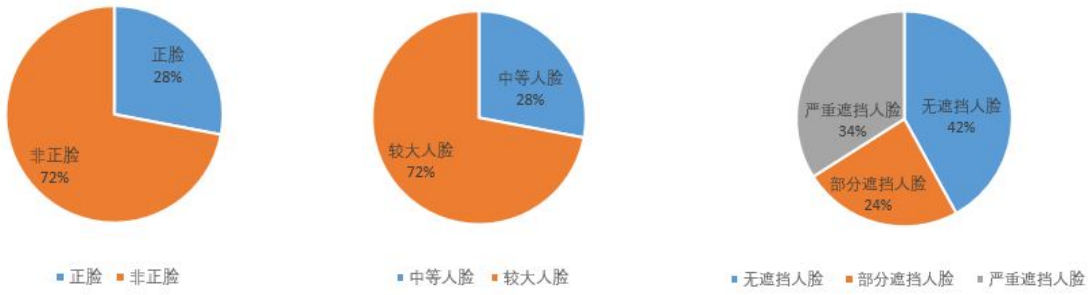
2.5.5 教室类型

在这里我们定义三种不同的教室类型：大型教室、中型教室和小型教室。其中大型教室为学生数量超过 100 人的教室；中型教室为学生数量介于 30 至 100 人之间的教室；小型教室为学生数量少于 30 人的教室。目前，CFDDB 中的区域仅有中等教室，后续会在 CFDDB 测试集的扩充中陆续添加从小型教室和大型教室获取的人脸图像与标记。

2.5.6 分析结果

对取样的到的 7 张图片进行分析，结果如表2–1所示：

由于取样的均匀性，这 7 张图片可以一定程度上代表整个 CFDDB 测试集的属性情况。从图2–3a可以看出 CFDDB 中非正脸的图像占据了超过 70%，从图2–3c可以看出，有遮挡人脸占比达到了 58%。这些特性使得 CFDDB 测试集对任何人脸检测算法和识别算法都是极大的挑战。



(a) 人脸姿势占比图　(b) 人脸大小占比图　(c) 人脸遮挡占比图

图 2–3 CFDDB 属性分析

Figure 2–3 CFDDB properties analysis

2.6 CFDDB 测试集的改进

CFDDB 由于采用了从视频流中截取连续帧的方法，使得有部分图片之间差异性较小。从而导致在测试检测算法时，如果在某张图片上效果不好，则在与之相邻的相同场景的图片上效果依然不好。这样会导致容易积累同样的错误从而使得 CFDDB 检测结果出现偏差。对于这种问题，可以通过使用更多的资源对数据库进行扩充来解决。如果数据量足够大，就可以稀释相似图片的积累错误，从而使得评价结果更加客观准确。

目前 CFDDB 采用通过剪裁截取到的图片计算类中心的方法建立人脸标准数据库。事实上这种方法只能粗略测得识别算法的准确性。更符合无感知检测场景的方法应当是由视频中出现人物的一张或多张证件照提取人脸特征向量，在对这些向量求类中心之后，得到人脸标准数据库中的人脸特征信息。目前受限于隐私保护，尚无法获得数据库中出现人脸的证件信息。这个问题有两种解决方法。第一种方法为招募志愿者，并提供完整的隐私保护政策，收集证件的照片信息。第二种方法为与受信任的机构合作，例如学校、医院，在学生或员工同意的情况下请机构提供证件的照片信息。

2.7 CFDDB 测试集的扩充

限于人力和物力，CFDDB 测试集目前数据量并不多。但是 CFDDB 在建立时就为可扩展性提供了广泛的支持。

首先，在标注人脸信息是使用了开源的工具 labelImg[1] 与通用的 XML 模板。这使得即使对 CFDDB 测试集没有过多了解的人也可以非常容易的参与到 CFDDB 测试集的扩充中。其次，我们收集了多种教室类型的视频数据，并设计了开源的抽取程序，使得任何人都可以非常简单的按照与我们同样的标准抽取视频数据，在保证测试集数据一致性的基础上提升了效率。最后，我们提供了开源的 XML 处理程序和算法测试样例，方便更多的研究者参与和使用 CFDDB 数据集。

未来更多人有望能够参与到 CFDDB 的使用和扩充中，使得 CFDDB 可以成为特定场景的非受

表 2–1 CFDDB 测试集属性

Table 2–1 CFDDB test dataset properties

人脸属性	004	024	044	064	084	104	124	总计
较小人脸	0	0	0	0	0	0	0	0
中等人脸	3	4	3	0	0	0	4	14
较大人脸	2	9	7	5	4	3	6	36
无遮挡	3	7	3	1	3	2	2	21
部分遮挡	1	3	2	2	0	1	3	12
严重遮挡	1	3	5	2	1	0	5	17
正脸	2	5	1	2	0	2	2	14
非正脸	3	8	9	3	4	1	8	36
总计	5	13	10	5	4	3	10	50

[1] https://github.com/tzutalin/labelImg

控无感知人脸检测和识别的先进测试集。如果数据量继续增加到数万张标记图片的规模，则可以将 CFDDB 中的数据按照某种合理的方式分为测试集和训练集，以方便对针对非受控无感知环境下人脸检测和识别算法的训练。

第三章 人脸检测算法

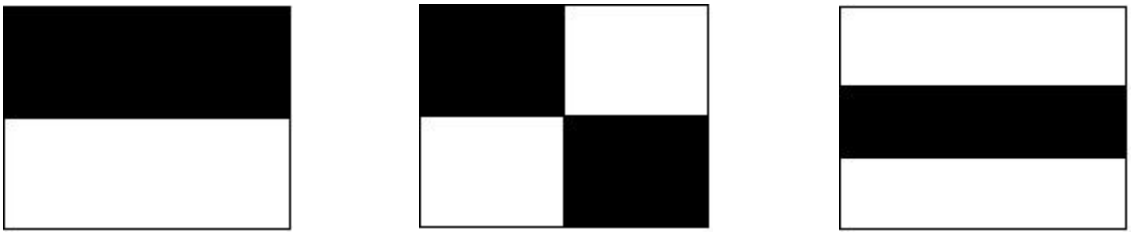
人脸检测算法自上世纪以来已经被无数研究者探究过，检测的准确率也逐年提高。在本系统中，我们需要的是一个能够优秀的处理大角度人脸、模糊人脸甚至部分被遮挡人脸的检测算法。在保证非常高的准确率的同时，该算法需要有非常快的执行速度，以适应同时处理大量图片的需求。

以 CFDDB 中的图片为标准，我们要求人脸检测算法需要在 CFDDB 中定义的 $recall \geq 80\%$ 的情况下，在一张 Nvidia 1080 Ti 显卡的支持下处理一张宽 1600 像素，高 1200 像素的图片的平均速度不超过 500ms。

3.1 基于 HAAR 特征的级联分类器

3.1.1 概述

HAAR 特征在物体检测领域有着广泛的应用，在 2001 年，Paul Viola 和 Michael Jones 在论文中提出了使用 HAAR 特征来检测人脸的方法[4]。这种方法的基本思想是在一系列含有人脸区域的正图像和不含人脸的负图像上训练一系列的分类器，然后将分类器级联组合，得到完成的人脸检测函数。在分类器的训练中，我们通过提取提取如图3–1所示的三种不同种类 HAAR 特征来判断一个特定的区域是否属于人脸区域。



(a) 边特征　(b) 四矩形特征　(c) 线特征

图 3–1 三种 HAAR 特征

Figure 3–1 Three HAAR features

在提取特征前，需要将图像转化为灰度图。三种特征中，图3–1a的边特征和图3–1c的线特征在顺时针旋转 90 度之后提取的特征归为同类特征。特征值则是将某一个特征中黑色部分区域的灰度值之和减去白色区域灰度值之和得到的。

同一种类的特征区域的大小可以不同，提取特征的图片区域可以不同，这样就导致了即使是非常小的一张图片，其所能够提取到的 HAAR 特征也是非常多的。例如，在一张宽 24 像素，高 24 像素的图片中，可以提取的特征可以超过了 16 万种。如果选取所有的特征进行比对，那么所花费的计算代价是不可接受的。因此，必须对特征进行选择。

首先，对每个特征进行训练，让它可以在训练集上尽可能区分开有人脸的图片和没有人脸的图片。接着计算这个特征的错误率。在所有特征都训练完毕之后选取错误率最小的部分特征作为需要

的特征保留。最后，将选取的特征加权求和，作为一个分类器使用。

每一个特征单独的分类效果较差，但是将所有特征加权求和得到的分类器则非常强大。使用 200 特征则可达到 95% 以上的准确率。最终的分类器使用了约 6000 个特征，具有非常强大的检测能力[4]。

然而，对于一张图片所有可能的区域计算 6000 个特征仍然是一件非常耗时的工作，对此，作者提出了使用级联分类器的方法来解决。所谓级联，就是将分类器串接起来，只有通过前面分类器检测的图片区域才会继续进行下一分类器的检测。最终的分类器共有 38 级，平均每个区域的识别所需要提取约 10 个特征[4]。这样的方式极大的加快了分类器的处理速度。

3.1.2 在 CFDDB 上的测试结果

我们使用 OpenCV 预训练好的 HAAR 级联分类器进行检测。这个分类器共有两个可变参数，minNeighbor 和 scaleFactor。其中 minNeighbor 表示构成人脸区域的相邻的小矩形个数，scaleFactor 表示在两次相继的扫描中，搜索框的比例系数。为了更好的检测 HAAR 级联分类器的效果，我们选取了不同的 minNeighbor 和 scaleFactor 参数在 CFDDB 上进行了测试，表3–1为测试结果：

表 3–1 HAAR 级联分类器在 CFDDB 上的测试结果

Table 3–1 HAAR cascade classifier results on CFDDB

minNeighbor	scaleFactor	faceDetected	faceRight	totalTime	recall
2	1.1	624	534	27.21s	49.13%
3	1.1	451	394	28.62s	36.25%
4	1.1	379	334	26.70s	30.73%
5	1.1	331	297	26.32s	27.32%
6	1.1	302	272	26.37s	25.02%
2	1.2	383	333	17.03s	30.63%
3	1.2	294	259	17.06s	23.83%
4	1.2	237	213	16.70s	19.60%
5	1.2	207	187	16.73s	17.20%
6	1.2	181	167	17.50s	15.36%
2	1.3	295	261	13.02s	24.01%
3	1.3	222	201	13.24s	18.49%
4	1.3	186	172	12.90s	15.82%
5	1.3	161	148	13.07s	13.62%
6	1.3	144	132	13.19s	12.14%

3.1.3 结果分析

3.1.3.1 处理速度

从图3–2中可以清楚的看出，scaleFactor 是影响处理速度的主要因素，scaleFactor 越大，处理速度越快。而 minNeighbor 参数则几乎不影响处理速度。HAAR 级联分类器在召回率最高时，处理 CFDDB

中的一张图片的平均速度没有超过 250ms，符合本系统的需求。

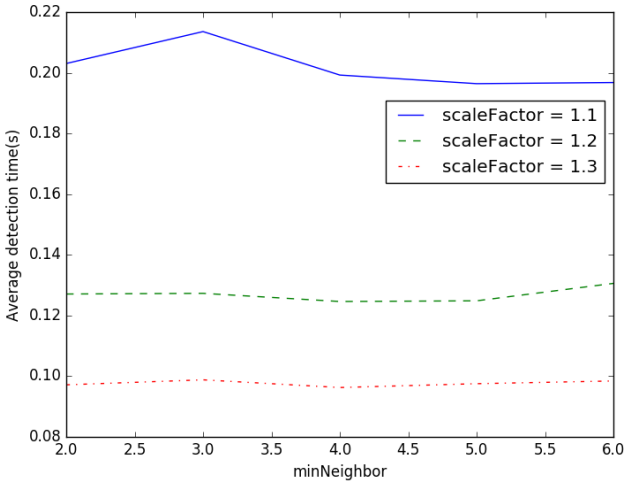


图 3–2 scaleFactor 与 minNeighbor 参数对于处理速度的影响

Figure 3–2 How parameter scaleFactor and minNeighbor affects processing speed

3.1.3.2 准确率

HAAR 级联分类器在 CFDDB 上召回率最高尚未超过 50%，没有达到本系统的需求。经过分析，共有两种原因影响了 HAAR 分类器在 CFDDB 上的检测效果：

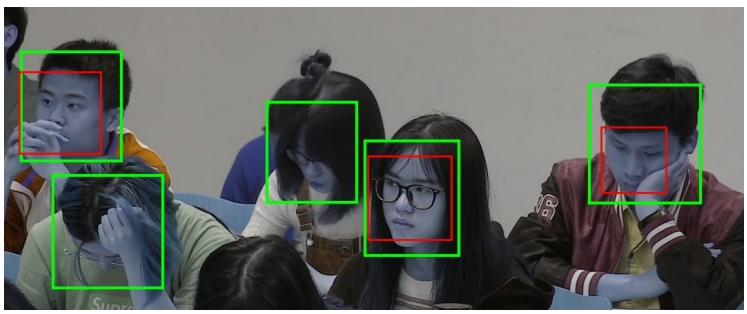


图 3–3 有角度和被遮挡的人脸识别结果示例

Figure 3–3 Example of angled faces and partially blocked faces

第一，这个预训练的 HAAR 分类器是针对正脸训练的，对侧脸和被遮挡人脸的检测效果非常差。我们将检测结果可视化之后取其中的一张图片分析，如图3–3所示。红色的矩形框是 HAAR 分类器检测到的人脸区域，绿色的矩形框是人工标注的人脸区域。可以看到，只要人脸的角度过大，或者被其他人或物体遮挡，HAAR 分类器的检测效果就会大打折扣。

第二，由于 HAAR 分类器对边缘信息特别敏感，使得在有些情况下会产生错误的识别结果，当 minNeighbor=2 并且 scaleFactor=1.1 时，如图3–4所示，HAAR 分类器经常会将衣服的褶皱误识别为

人脸或将特殊的边缘误识别成人脸。

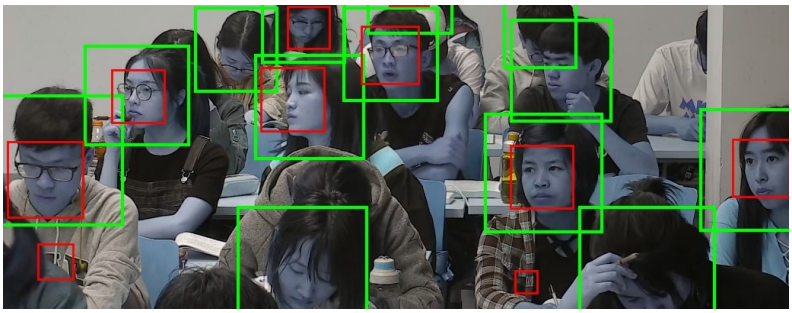


图 3–4 HAAR 分类器误识别示例

Figure 3–4 Haar cascade classifier misidentification example

综上所述，基于 HAAR 特征的级联分类器存在诸多限制，准确率难以达到要求，并不适合作为本系统的人脸检测模块使用。

3.2 MtCNN

3.2.1 概述

传统的人脸识别方法大多是基于某种分类器从图像中提取特征，根据特征或者特征组合来判断某一图像区域是否属于人脸。然而就像上文所述的基于 HAAR 特征的级联分类器，传统的人脸识别的方法往往无法胜任从监控录像所拍摄的图像中准确提取多个人脸的工作。因此，我们将目光投向了深度学习的方法。

近年来，以卷积神经网络为代表的深度神经网络在计算机视觉的各个领域都有了丰硕的成果，人脸检测也不例外。下面我们就将介绍一种利用深度神经网络执行人脸检测和校正的网络：MtCNN[11]。

MtCNN 的全称是多任务级联卷积网络，由 Kaipeng Zhang 等人中提出。这个网络一经提出就被广泛应用在多种人脸识别的神经网络的训练和测试输入中，是目前广泛应用的人脸检测网络之一。

在 MtCNN 之前，也有学者尝试使用卷积神经网络进行人脸检测。然而由于他们所使用的卷积神经网络过于复杂，导致计算的时间复杂度极高，无法实用[18]。MtCNN 不仅大大提升了人脸检测的速度，而且创造性的将人脸检测和人脸校正合二为一，丰富了网络的功能，为下一步的识别系统提供了极大的方便。

3.2.2 总体架构

MtCNN 中共包括三个不同的网络，三个不同的网络，即 P-Net, R-Net 与 O-Net，如图3–5[11] 所示：

在接收到一张图片之后，首先将其缩放得到图片金字塔。所得到的图像金字塔是三个级联卷积网络的输入。

第一阶段，我们使用一种全卷积网络叫做 Proposal Network(P-Net) 来获得候选人脸区域和它们的边界框回归向量。接着使用边界框回归向量去校准候选区域。最后使用非极大值抑制 (NMS) 整合高度重合的区域。

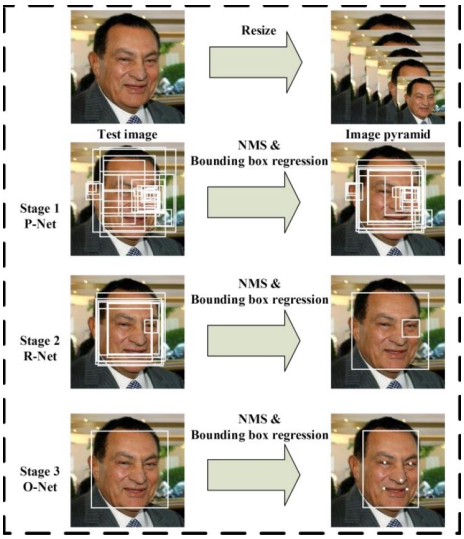


图 3–5 MtCNN 总体架构图[11]

Figure 3–5 MtCNN overall pipeline[11]

第二阶段，我们使用另一种卷积网络叫做 Refine Network(R-Net) 来进一步筛选得到的区域。在筛选结束之后，与第一阶段相同，使用边界框回归对人脸边界进行进一步校准。最后使用 NMS 整合重合区域。

第三阶段与第二阶段基本相似，但是在这一阶段我们主要对脸部更精细的特征进行描述。这一阶段使用 Output Network(O-Net) 得到人脸区域的五个特征点。

3.2.3 Loss 函数设计

对于每一个样本 x_i，人脸区域区分部分的 Loss 函数其中 p_i 是网络输出结果是人脸区域的概率，$y_i^{det} \in \{0, 1\}$ 是该区域是否是人脸的标签[11]。

$$L_i^{det} = -((y_i^{det} log(p_i)) + (1 - y_i^{det})(1 - log(p_i))) \tag{3–1}$$

边界框回归的 Loss 函数其中 $\hat{y}_i^{box}$ 是由网络得出的结果，y_i^{box} 是实际的边界框[11]。

$$L_i^{box} = ||\hat{y}_i^{box} - y_i^{box}||_2^2 \tag{3–2}$$

脸部特征回归的 Loss 函数与边界框类似，使用了 Euclidean Loss[11]。

$$L_i^{landmark} = ||\hat{y}_i^{landmark} - y_i^{landmark}||_2^2 \tag{3–3}$$

3.2.4 在 CFDDB 上的测试结果

我们使用预训练完成的 MtCNN 在 CFDDB 上进行测试，测试中使用了 TensorFlow 1.7.1 版本，并使用 GPU 加速网络运算。测试结果如表3–2所示，其中 confidence 表示所取 MtCNN 输出的人脸区域信任率，averageTime 表示平均处理一张图片所需要的时间。

3.2.5 结果分析

3.2.5.1 处理速度

不同的 confidence 参数处理时间基本一致，均为 350ms 左右的时间处理一张宽 1600 像素，高 1200 像素的图片。由于使用了 GPU 加速运算，并且将原有的 Matlab 平台的代码成功移植到了更先进的 TensorFlow 平台，使得 MtCNN 处理速度符合我们的要求。

3.2.5.2 准确率

在 CFDDB 测试集中，MtCNN 最多正确检测出了 423 个人脸区域，召回率最高为 38.91%，召回率较低，无法符合我们的要求。但是在 WIDER FACE 中的 Hard 测试集上，MtCNN 都有着超过 60% 的召回率[11]，然而在 CFDDB 上却无法体现。经过分析，可能的原因有以下两条：

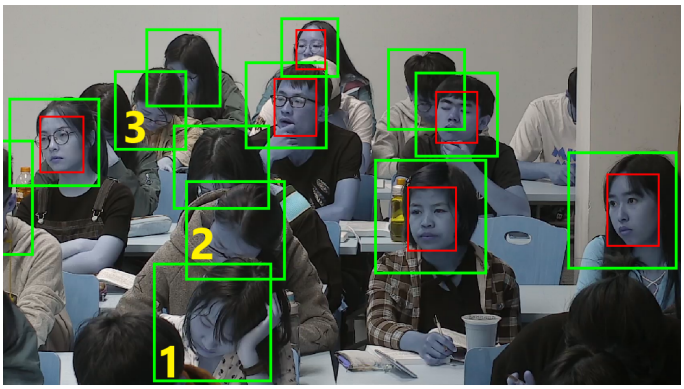


图 3–6 MtCNN 识别结果可视化样例

Figure 3–6 MtCNN detect results example

原因一：对侧脸的识别，五个特征点不完整的人脸召回率较低。CFDDB 中有大量侧脸和不完整人脸，如图3–6所示，其中人脸 1 俯仰左右都有较大的角度，难以识别；人脸 2 部分被遮挡，无法提取完整的五个特征点，因此被舍弃；人脸 3 俯仰角度过大，难以识别。

原因二：CFDDB 数据分布的问题，由于数据中样本采集采用的从连续的视频中截取帧的方法，前后两次截取得到的图片差距不大，这样容易导致前一张图片中识别不出的图像往往在下一张图像中一样难以识别。这个问题需要通过扩充 CFDDB 的样本规模，去除相似度高的图像来解决。

表 3–2 MtCNN 在 CFDDB 上的测试结果

Table 3–2 MtCNN test results on CFDDB

confidence	faceDetected	faceRight	averageTime	recall
0.90	433	388	344.38ms	35.69%
0.80	465	410	325.71ms	37.71%
0.70	483	423	357.41ms	38.91%
0.60	483	423	333.66ms	38.91%

3.3 SSH 检测器

3.3.1 概述

SSH 的全称为 Single Stage Headless face detector[1]，由 Mahyar 等人提出。SSH 检测器针对小脸进行了优化，对处于自然环境的人脸识别有着非常好的效果。在 WIDER FACE Hard 测试集的测试中，SSH 检测器的召回率达到了 0.844[1]，从检测效果来看，更加符合我们这个系统的需求。

与其他同样基于 CNN 的人脸检测器相比 SSH 检测器有如下四个特点：

首先，大多数基于 CNN 的人脸检测器只针对一种特定大小的人脸进行了训练。因此，在使用的时候需要先建立图像金字塔，然后对图像金字塔中的每一张图片在训练好的网络中前向传播，合并所有前向传播的输出得到结果。而 SSH 检测器设计了针对三种不同大小人脸的检测器，从不同的网络层中提取信息，使得 SSH 检测器在使用的时候只需要一次前向传播就可以完成所有大小的人脸检测。

其次，大多数基于 CNN 的人脸检测器将人脸检测分为两个阶段。第一阶段，浅层的卷积特征图产生候选边界框集合。第二阶段，剩余的分类网络用于从集合中提取局部特征并将其分类。在这样的设计中，在第二阶段必须对第一阶段所有产生的候选框进行计算，从而需要极高的计算代价。而 SSH 检测器在一个阶段同时完成了边界框回归和最后的分类运算，因此计算代价更小。

再次，在达到国际领先的检测水平的同时，SSH 检测器舍弃了基础 VGG-16 网络[6] 中含有大量参数的全连接层，使得训练过程更加容易。同时，也使得网络整体变得轻量快速。

最后，SSH 检测器针对每个检测模块分别采用了 Online hard negative and positive mining[19] 的方法进行训练，使得训练得到的模型误报率更低，更加精确。

3.3.2 网络结构

图3–7[1] 展示了 SSH 检测器的总体架构。从图中可以看出，SSH 检测器是一个全卷积深度网络，通过在步长分别为 8、16 和 32 的特征图上添加图3–8[1] 所示的检测模块，实现了利用网络定位和区分人脸区域的功能。检测模块则由二元卷积分类器和回归运算模块组成。

SSH 检测器为单阶段检测器，并没有产生边界框候选集的阶段。事实上，SSH 检测器通过对预先定义的边界框集进行回归而得到人脸区域的。而边界框集是通过密集的、重叠的滑动窗口所定义的。

而对于检测模块而言，有一系列的卷积层用于检测人脸的面部特征和位置定位，其中就包括了一个背景环境模块，如图3–9[1]。这个背景环境模块用于提高感受野的有效性，从而提高网络的检测精度。背景环境在人脸较小或者不清晰的时候可以向检测器提供非常重要的信息。因为当人脸面部信息被遮挡或者不清晰的时候，人脸周围的背景环境例如衣领、头发等信息可以作为面部区域的关键的辅助信息传入网络中，帮助网络断定这个区域是否为人脸区域。

事实上，在另一个由 Peiyun Hu 等人提出的 Tiny face 检测器[8] 在论文中详细叙述了背景区域对于小脸检测的重要性。根据论文中的叙述，增加背景环境信息几乎总是有效的，无论对于大脸还是小脸。Peiyun Hu 等人在实验中发现唯一降低了识别准确率的情况是向宽 25 像素，高 25 像素的小脸区域加入了超过宽 300 像素，高 300 像素的背景信息区域。而这种情况召回率降低的主要原因是向感受野中加入了明显过多的背景信息而导致了检测器出现了过拟合现象。

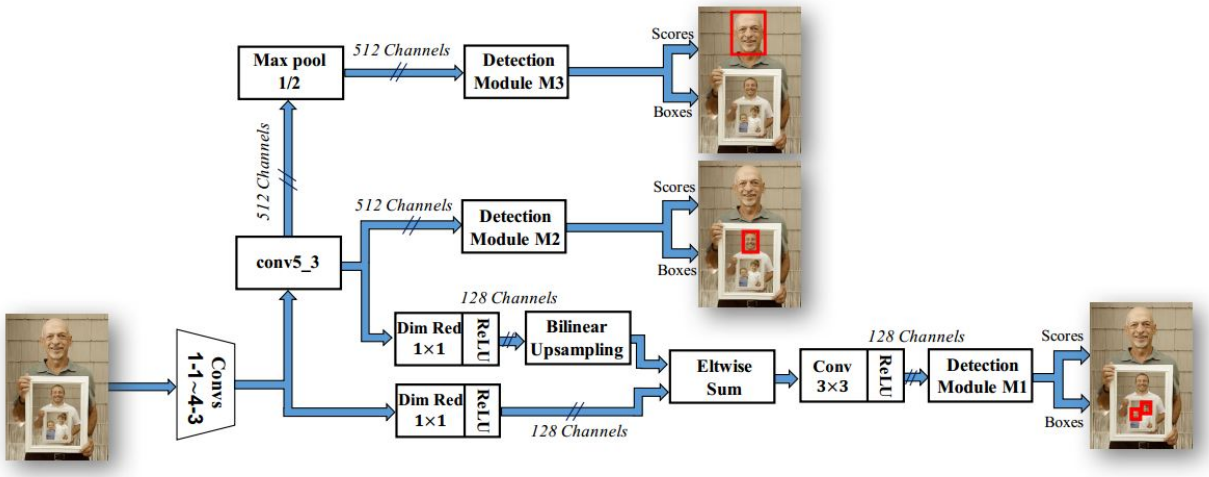


图 3–7 SSH 检测器总体架构[1]

Figure 3–7 SSH face detector general architecture[1]

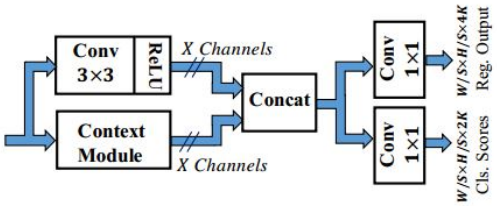


图 3–8 SSH 检测模块[1]

Figure 3–8 SSH detection module[1]

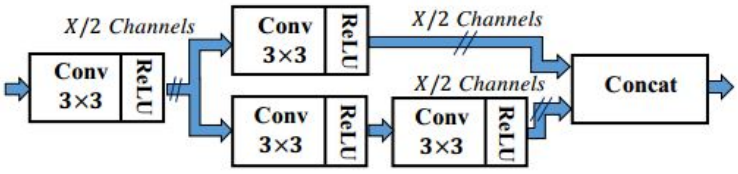


图 3–9 SSH 背景环境模块[1]

Figure 3–9 SSH context module[1]

3.3.3 在 CFDDB 上的测试结果

我们使用预训练完成的 SSH 检测器在 CFDDB 上进行测试，选用 Caffe 作为框架平台，测试中使用了 GPU 加速网络运算。测试结果如表3–3所示，其中 confidence 表示所取 SSH 检测器输出的人脸区域信任率，averageTime 表示平均处理一张图片所需要的时间。

表 3–3 SSH 在 CFDDB 上的测试结果

Table 3–3 SSH test results on CFDDB

confidence	faceDetected	faceRight	averageTime	recall
0.90	921	854	235.93ms	78.56%
0.80	957	885	235.41ms	81.42%
0.70	975	900	236.19ms	82.80%
0.60	989	913	224.73ms	83.99%
0.50	1008	929	227.33ms	85.46%

3.3.4 结果分析

3.3.4.1 处理速度

不同的 confidence 参数处理时间基本一致，均为 230ms 左右的时间处理一张宽 1600 像素，高 1200 像素的图片。由于使用了 GPU 加速运算，而且 SSH 检测器使用了轻量的网络，使得处理时间完全满足系统需求。

3.3.4.2 准确率

在 CFDDB 测试集中，SSH 检测器最多正确检测出了 998 个人脸区域，召回率高达 91.81%。即使将 confidence 参数调高至 0.9，召回率仍然达到了 85.1%，超过了 80% 的要求。因此，SSH 检测器的准确率完全可以满足系统的需求，适合作为人脸检测模块的核心算法。

3.3.5 网络优化与平台移植

虽然 SSH 检测满足了我们的检测条件，但是我们在测试中也发现了 SSH 存在的一些待改进的问题。

首先，SSH 检测器目前只有 Caffe 平台的实现，这对于测试和部署都是一项极大的挑战。Caffe 平台虽然功能强大，但是搭建 GPU 支持的 Caffe 框架却必须从源码进行编译。这就意味和每次部署搭建环境时都必须安装数量庞大的依赖库，并下载源码编译。整个过程需要耗费大量的时间，不利于规模性部署。而且随着 Caffe 平台的维护力度逐渐减小，最新版本的显卡驱动和深度神经网络加速库均无法使用，使得网络的运行效率下降。同时，Caffe 平台对于嵌入式设备的支持优化较少，不利于将识别模块移植到嵌入式平台上使用。

其次，SSH 检测器基于性能较弱的 VGG-16 网络[6] 改进而来，存在提升的空间。如果将其中的 VGG-16 网络替换为对应的 MobileNets 网络[20]，或者 MobileNetsV2 网络[21]，则不仅可以提升网络的

运行速度，而且更方便将网络移植到嵌入式设备等没有 GPU 支持的设备中使用。

最后，SSH 检测器难以兼容使用 TensorFlow 平台实现的人脸识别算法。由于 Caffe 平台和 TensorFlow 平台使用了不同版本的 Protocol Buffer 依赖库，如果希望能够同时运行两个平台就需要安装老旧版本的 TensorFlow 平台并进行一些繁琐的设置。这不仅损失了人脸识别算法的运行效率，而且给规模化部署带来了更多的挑战。

3.4 Tiny face 检测器

3.4.1 概述

Tiny face 检测器[8]，是由 Peiyun Hu 等人提出的一种针对小脸进行优化处理的人脸检测器，也是最早针对小脸进行检测的深度网络之一。Tiny face 使用了图像金字塔和多种不同大小、不同宽高比的模板来检测不同大小人脸，并基于 VGG-16 网络[6]，ResNet-101 和 ResNet-50 网络[7] 结构都进行了实现。其中基于 ResNet-101 的实现在 WIDER FACE 测试集 Hard 部分的召回率达到了 0.823[8]。

3.4.2 在 CFDDB 上的测试结果

我们使用在 TensorFlow 平台上预训练完成的 Tiny face 检测器在 CFDDB 上进行测试，测试中使用了 GPU 加速网络运算。测试结果如表3–3所示，其中 confidence 表示所取 Tiny face 检测器输出的人脸区域信任率，averageTime 表示平均处理一张图片所需要的时间。

表 3–4 Tiny face 检测器在 CFDDB 上的测试结果

Table 3–4 Tiny face detector test results on CFDDB

confidence	faceDetected	faceRight	averageTime	recall
0.90	1068	953	1906.05ms	87.67%
0.80	1070	953	1792.24ms	87.67%
0.70	1074	952	1859.22ms	87.58%
0.60	1077	958	1888.82ms	88.13%
0.50	1082	961	1801.81ms	88.41%

3.4.3 结果分析

3.4.3.1 处理速度

不同的 confidence 参数处理时间基本一致，均为 1850ms 左右的时间处理一张宽 1600 像素，高 1200 像素的图片。时间大约为 SSH 检测器处理时间的 8 9 倍，而且远超大于系统所要求的 500ms 时间，不符合系统的要求。

3.4.3.2 准确率

在 CFDDB 测试集中，Tiny face 检测器最多正确检测出了 1082 个人脸区域，召回率高达 88.41%。即使将 confidence 参数调高至 0.9，召回率仍然达到了 87.67%，超过了 80% 的要求。因此，Tiny face

检测器的准确率满足系统的需求。

3.4.3.3 与 SSH 检测器的对比分析

从表3–3和表3–4可以看出，在运算效率上，SSH 检测器优势非常大。原因主要是 SSH 能够一次性检测多种大小的人脸而不依赖于图像金字塔，避免了在多个尺度处理同一张图片的运算，从而获得了更高的运算效率。

如果不考虑运算效率，而希望得到召回率高而误判率低的算法，则需要计算两种算法在 CFDDB 上的参数 α 得到表3–5所示的结果：

表 3–5 Tiny face 检测器与 SSH 检测器对比分析表

Table 3–5 Tiny face detector and SSH detector comparision table

confidence	recall(Tiny)	errorrate(Tiny)	α(Tiny)	recall(SSH)	errorrate(SSH)	α(SSH)
0.90	87.67%	10.58%	2.1986	78.56%	6.16%	1.8892
0.80	87.67%	10.76%	2.2170	81.42%	6.62%	1.8906
0.70	87.58%	11.22%	2.2642	82.80%	6.90%	1.8978
0.60	88.13%	10.95%	2.2294	83.99%	6.99%	1.8898
0.50	88.41%	11.13%	2.2443	85.46%	7.27%	1.8968

根据 α 的定义知，当召回率越高的时候 α 的值越小，当误判率越低时 α 的值越小，因此，α 的值越小，代表了当前检测算法的性能越优秀。从表3–5中可以清晰的看出，无论参数 confidence 的值是多少，SSH 检测器的 α 值总是小于 Tiny face 检测器的 α 值。因此，综合考虑召回率和误判率的情况下，SSH 检测器依然优于 Tiny face 检测器。

第四章 特征提取算法

在检测到人脸区域的之后就需要对区域内的人脸图像进行特征提取。目前最先进的特征提取思路是将人脸图片经过深度神经网络的处理得到高维的人脸特征向量。然后使用这些特征向量进行人脸搜索与匹配。

事实上，我们可以将得到高维人脸特征向量的过程看作是一种映射。我们使用的深度神经网络就是一种映射。相同或相似的人脸图像由这种映射得到的向量具有较近的距离，而差异较大的人脸图像经过映射得到的向量具有较远的距离。

特征向量间的距离定义也有很多种，有些是欧式距离，有些是角度距离。而如果网络输出的特征向量都进行了标准化处理，就可以用两个特征向量的点积作为距离的衡量标准。在这种情况下，两个特征向量的点积越接近 1，则两个特征向量对应的人脸就越相像；越接近 −1，则两个特征向量对应的人脸差异越大。

在实际算法中，大多数算法在识别之前都有将人脸校正的需要。校正操作就是通过检测到人脸的多个特征点进行运算，从而将带有较大角度的人脸转换为相应正脸的过程。由于这种过程需要检测人脸的特征点，许多算法都使用了在人脸检测算法中提到的 MtCNN 网络[11] 进行校正操作。

如果全部可识别的人脸数量为 $face_{all}$，识别正确的人脸数量为 $face_{right}$ 定义召回率 $recall$ 为：

$$recall = \frac{face_{right}}{face_{all}} \tag{4–1}$$

由于 CFDDB 目前只能粗略测得特征提取算法的准确性，我们要求合格的特征提取算法需要 $recall > 50\%$。

4.1 InsightFace[2] 简介

使用深度神经网络检测人脸的方法有很多，它们主要的不同点有三个：

第一，训练数据。对于深度神经网络而言，通常训练数据越多，得到的模型就越精确。特征提取中使用的公开训练数据集主要有 VGG-Face[22]，VGG2-Face[23]，CASIA-WebFace[24]，UMDFaces[25]，MS-Celeb-1M[26] 和 Megaface[27] 等。而 Google 等公司的私有数据集甚至拥有数百万的不同人脸数据，可以训练出商用的特征提取网络。

第二，网络结构。与使用 CNN 网络的人脸检测器类似，研究者们也为特征提取设计了许多不同的网络。常见的有 ResNet 系列网络[7]，VGG 系列网络[6] 和 MobileNets[20] 等等。

第三，Loss 函数。不同的神经网络有着不同的 Loss 函数设计，常见的有 Euclidean margin based loss 和 Angular and cosine margin based loss 两大类。

而我们使用的 InsightFace 其训练数据为 MS-Celeb-1M[26] 与 VGG2[23] 数据集，网络结构基于 MXNet[28]，使用 additive angular margin 作为 Loss 函数。由于 InsightFace 在 Megaface 测试集上取得了国际领先的测试结果，我们选择它作为特征提取模块的核心方法。

4.2 在 CFDDB 上的测试结果

下表4–1展示了 ArcFace 在 CFDDB 上的测试结果。其中参数 confidence 表示两特征向量的最小信任值。即如果一张图片的特征向量与人脸标准数据库中所有的特征向量的点积值均低于 confidence，则认为该图片中的人脸不属于人脸标准数据库中的任何一张人脸。参数 faceRecongnized 参数表示当前人脸标准数据库中的可识别的人脸数量，参数 faceRecognizeRight 表示识别正确的人脸数量。

表 4–1 ArcFace 在 CFDDB 上的测试结果

Table 4–1 ArcFace test results on CFDDB

confidence	faceRecognized	faceRecognizeRight	recall
0.90	249	193	77.51%
0.80	412	558	73.84%
0.70	543	711	76.37%
0.60	838	649	77.44%
0.50	916	691	75.44%

分析表4–1中的测试结果可以看出，调整 confidence 参数从 0.5 至 0.9，召回率均超过 70%，符合系统的要求。

第五章 人脸匹配算法

5.1 概论

由特征提取模块得到人脸特征向量之后，接下来就需要将这个特征向量与人脸标准数据库中已有的特征向量进行匹配，找到距离其最近的特征向量，从而确定待检测人脸的身份。如果数据库中的数据量较少，线性搜索的性能就足以满足系统需求。但在数据量较大，例如数据库中有百万以上的特征向量时，线性搜索需要较多的时间，无法满足系统实时性的需求。因此，我们需要更高效的搜索算法加速这一过程。

我们对合格的人脸匹配算法有四点要求：

第一是准确，一次查询的结果应当非常接近线性搜索的结果。第二是快速，算法的时间复杂度为不超过 $O(n)$。第三是高效，即理想情况下，数据的索引应当和数据集成线性关系。第四是高维，由于人脸特征向量的维度常在 100 维以上，算法需要支持对高维特征的高效搜索。

除了线性搜索，常见的精确搜索算法有基于树形结构的 R-tree 和 K-D tree 等算法，根据一项研究结论，在维度超过 10 的时候，这些精确算法的时间效率已经超过了线性搜索[29]，而人脸特征向量的维度基本上在 100 的级别。所以使用这些精确搜索算法无法满足高维的要求，这些算法都不合适用作本系统的匹配算法。

而在机器学习的领域，支持向量机 (SVM) 常用于高维向量的分类和回归。而在人脸向量的匹配阶段，如果人脸特征数据库内存在 n 个特征向量，那么我们使用 SVM 将找到查询向量 **q** 所匹配的特征向量共有两种思路：

思路一，训练 n 个分类器，每个分类器都将人脸特征库中的某一个特征向量与剩余的所有特征向量区分开。在使用时，需要每个分类器对查询向量 **q** 进行分类，如果 **q** 与某个特征向量分到了同一类，则将该特征向量作为 **q** 的匹配候选项。

思路二，训练 C_n^2 个分类器，将 n 个特征向量中任意挑出的一对不同的特征向量区分开。需要找到查询向量 **q** 的匹配项时，使用每个训练好的分类器对 **q** 进行分类，每次将分类器输出结果所表示的特征向量类别权重加一。所有的分类器分类完毕时，选择权重最高的一类中的特征向量即为匹配结果。

这两种思路在 n 不大的时候都是可行的。事实上，在一些简单的人脸识别系统中，使用多类支持向量机 (Multi-Class SVM) 匹配高维人脸特征向量的做法非常普遍。

然而对于思路一中的方法，当 n 非常大的时候，需要训练的分类器的数量为 $O(n)$，而且每次查询的时间复杂度也为 $O(n)$；对于思路二中的方法，当 n 非常大的时候，需要训练的分类器的数量为 $O(n^2)$，每次查询的时间复杂度也为 $O(n^2)$。在简单分析后我们可以看出，多类支持向量机不符合快速的要求，当人脸特征向量的数量非常大的时候，不适合作为人脸匹配算法使用。

我们注意到人脸特征向量的具有这样的特性：相同身份的人脸会产生距离较近的特征向量，而不同身份的人脸往往会产生距离较远的特征向量。由此可将人脸特征向量的匹配过程等同于在人脸特征向量数据库中寻找距离查询向量 **q** 距离最近的一项。通过这样的转换，我们将匹配问题等价于

最近邻搜索问题。

最近邻搜索问题的解决主要有两大类方法，精确搜索和近似搜索。其中近似搜索是一类能够以非常高的概率找到最近邻的搜索方法。

5.2 基于局部敏感哈希的近似搜索

5.2.1 LSH 函数族的定义与生成

局部敏感哈希 (LSH) 在最近邻的搜索问题上有着广泛的应用，是一种近似搜索方法。

LSH 算法的核心是 LSH 函数族，以下为 LSH 函数族的定义：

一个 LSH 函数族 $\mathcal{F}$ 定义在矩阵空间 $\mathcal{M} = (M, d)$ 上，其中阈值 $R > 0$，渐进系数 $c > 1$。LSH 函数族中的每一个函数 $h : \mathcal{M} \to S$ 都将元素从矩阵空间映射到桶 $s \in S$ 中，并满足使用随机平均抽取的一个函数 $h \in \mathcal{F}$，为对任意的两个点 $p, q \in \mathcal{M}$，都有如下特性：

- 若 $d(p, q) \leq R$，那么 $h(p) = h(q)$ 的概率至少为 P_1。
- 若 $d(p, q) \geq cR$，那么 $h(p) = h(q)$ 的概率至多为 P_2。

在最邻近搜索中使用的 LSH 函数族均满足 $P_1 > P_2$ 的条件，此时，这个 LSH 函数族被称为 $(R, cR, P_1, P_2) - sensitive$。

如果给定一个 $(d_1, d_2, p_1, p_2) - sensitive$ 的 LSH 函数族 $\mathcal{F}$，我们可以通过与或的操作创建新的 LSH 函数族。

采用求与的方法，定义一个新的函数族 $\mathcal{G}$，函数族中每一个哈希函数为 g。每一个 g 均由函数族 $\mathcal{F}$ 中的 k 个随机选取的哈希函数 $h_1, ..., h_k$ 构成。对于哈希函数 $g \in \mathcal{G}$，$g(x) = g(y)$ 当且仅当 $i = 1, 2, ..., k$ 时，所有的 $h_i(x) = h_i(y)$。此时生成 $\mathcal{G}$ 为 $(d_1, d_2, p_1^k, p_2^k) - sensitive$ 的 LSH 函数族。

采用求或的方法，定义一个新的函数族 $\mathcal{G}$，函数族中每一个哈希函数为 g。每一个 g 均由函数族 $\mathcal{F}$ 中的 k 个随机选取的哈希函数 $h_1, ..., h_k$ 构成。对于哈希函数 $g \in \mathcal{G}$，$g(x) = g(y)$ 当且仅当存在一个或者多个 i，使得 $h_i(x) = h_i(y)$。此时生成 $\mathcal{G}$ 为 $(d_1, d_2, 1 - (1 - p_1^k), 1 - (1 - p_2^k)) - sensitive$ 的 LSH 函数族。

5.2.2 基础 LSH 最邻近搜索算法

利用 LSH 函数族进行最近邻搜索的算法有两个主要参数，宽度 k 和哈希表的数量 L，其中宽度 k 由创建过程中随机选取哈希函数的数量 k 定义。

第一步，采用上述的求与或求或的方法创建新的 LSH 函数族 $\mathcal{G}$，接着由 LSH 函数族 $\mathcal{G}$ 中随机选区 L 个不同的哈希函数，创建 L 个哈希表。

第二步，对于查询向量 $\mathbf{q}$，遍历每一个哈希表，取出与其分配到同一桶中的的向量将其放到集合 S 中。

第三步，使用线性搜索的方法找到 S 中最邻近 $\mathbf{q}$ 的向量。

5.2.3 基础 LSH 最邻近搜索算法复杂度分析

令

$$k = \frac{logn}{log1/P_2}, L = n^{\rho} \tag{5–1}$$

其中

$$\rho = \frac{logP_1}{logP_2} \tag{5–2}$$

则该算法的空间复杂度为 $O(n^{1+\rho})$，时间复杂度为 $O(n^{\rho})$。由于 $0 < \rho < 1$，所以该算法的搜索的时间效率高于线性搜索。

从工程实践的角度出发，基础 LSH 算法的最大问题是需要非常多的哈希表，才能在保证准确性的同时找到绝大多数的邻近元素。而每个哈希表的空间大小是和数据集的大小成比例的，所以数据量越大，需要的存储空间就越大。如果哈希表的存储空间超过了内存的容量，那么每一次查询操作就会涉及到磁盘的读取，从而极大的增加每次查询操作的时间开销。

5.2.4 基础 LSH 最邻近算法的优化思路

优化基础 LSH 最邻近算法的一种思路是减少查询时所需要的哈希表的数量。假设我们知道待求的最邻近向量 **p** 和查询向量 **q** 之间的距离 R_p，那么原则上，我们可以计算出 **p** 与 **q** 分在同一个桶中的概率。如果定义这个概率叫做成功率，那么查询的时候只需要查找成功率高的桶就可以避免使用过多无用的哈希表，从而提高存储的利用率。

然而预先计算每个桶的成功率在实际情况下是十分复杂的，所以，我们需要一种方法来高效的进行这种计算。每次查询之前，随机生成一个与查询向量 **q** 相距 R_p 的点 **p**′。标记 **p**′ 经过哈希后落到的桶。这样保证了所有的桶取样概率的正确性。多次执行同样的操作，即可找出所有高成功率的桶。

这样的思路以牺牲时间效率为代价换取了更高的空间利用率，避免了内存不够而需要访问硬盘的情况。这样的优化方式使得需要的哈希表数量减少了 1/2 到 2/3[30]。不过由于每次查询前都增加了取样的过程，使得每次查询的时间增加了 30% 到 210% 不等。

除了每次查询需要更多的时间之外，在实际测试中还发现了上述思路存在其他一些缺陷。

第一，在一次查询中高成功率的桶会被不同的扰动点 **p**′ 产生多次，即在一次查询中做了多次重复计算。通过记住每个扰动点生成的桶可以避免此问题，而其需要的空间开销在高并发量的查询时仍然是不可接受的。

第二，对于每一个待检测的数据集，参数 R_p 需要人工调整。若 R_p 偏大，则取样结果过多，容易产生误判；若 R_p 偏小，则取样结果过少，容易产生缺漏。

针对上述的缺陷，Qin Lv 等人提出了多探针 LSH[30]，以在减少哈希表数量的同时可以高效的执行查询操作。

设 **a** 为从正态分布中随机取出的 d 维向量，b 是从 $[0, W]$ 均匀分布中随机取出的一个实数，则 LSH 函数族中的每一个哈希函数被定义为：

$$h_{\mathbf{a},b}(\mathbf{v}) = \left\lfloor \frac{\mathbf{a} \cdot \mathbf{v} + b}{W} \right\rfloor \tag{5–3}$$

根据这个哈希函数的性质，如果 **p**，**q** 两点为邻近点，而经过哈希函数的计算后 **p** 没有和 **q** 分在同一个桶中，那么 **p** 一定与 **q** 分到了靠近的桶中。事实上，如果参数 W 选取合适，**p** 会有极大的概率分到与 **q** 紧邻的桶中。

根据邻近点在桶中的分布特性，使用合适的方法推导出探针序列，即可使用探针将绝大多数查询向量 **q** 的邻近点找出，从而得到最邻近点。

多探针 LSH 算法的测试采用了两个数据集，一个是包含 130 万图片的数据集，另一个是含有 260 万音频的数据集。测试结果表明，多探针 LSH 算法的空间利用率是基础 LSH 算法的 14 至 18 倍，而查询时间基本接近[30]。

5.3 基于邻近图的近似搜索

5.3.1 概述

基于 LSH 进行近似搜索的主要思想是相似的向量经过哈希函数的计算后仍然在相邻或者相同的桶里。这类算法在数据集中的高维向量距离相距较远的情况下存在明显的缺陷，即相邻的样本在计算会分布在稀疏的、不相邻的桶中，从而降低搜索效率。相比于 LSH，数据集中的每个特征向量都由有向图中的一个顶点表示，构建有向图来描述数据集则可以很好的解决这个问题。Ben Harwood 与 Tom Drummond 提出了一个高效的利用邻近图执行近似搜索的算法 FANNG[3]。

5.3.2 理想图的构建

如何构建邻近图是这类算法的核心。而最优的邻近图是在理想图的基础上优化而来。因此，在得到邻近图之前首先需要构建一个理想图。当查询点 Q 为数据集中的一个点时，无论初始查询下标从哪里开始，算法5–1总能正确的找到正确的匹配结果。符合这样要求的最小图定义为理想图。

算法 5–1 向下搜索算法[3]

输入： P 理想图的顶点集，E 理想图的边集，Q 查询点，v 初始查询下标

输出： v 最邻近下标

```
1: for each edge E_i with start vertex P_v do
2:     u ← index of end vertex of E_i
3:     if distance(Q, P_u) < distance(Q, P_v) then
4:         v ← u
5:     end if
6: end for
```

在这样的搜索过程中，每次迭代后当前的点与查询点之间的距离都会减小。因此，整个算法在有限次迭代之后一定会终止。这也同时意味着如果图中存在一条从 p_1 到 p_2 的边，那么任何一条从 p_1 出发到 p_3 的边，当 p_1 到 p_3 的距离比 p_2 到 p_3 的距离远时都没有存在的意义。

在理想图中，给定数据点 p_i 与距离函数 $d : \mathbf{R} \times \mathbf{R} \to \mathbf{R}$，定义边 (p_1, p_2) 阻塞边 (p_1, p_3) 当且仅当 $d(p_1, p_2) < d(p_1, p_3)$ 并且 $d(p_2, p_3) < d(p_1, p_3)$。

明确了如何精简图之后，使用算法5–2构建理想图。

建立理想图之后我们需要探究其存储空间是否合理。尽管数据集中的数据可能是高维的，例如 SIFT[31] 数据集中的数据有 128 维，但是其固有维度却只有 11 维[3]。这使得使用 SIFT 数据集构建的理想图中每个点的度平均值为 25 左右。因此，图的存储空间是可以接受的。

算法 5–2 理想图构建算法[3]

输入： P 理想图的顶点集

输出： E 理想图的边集

```
1: for each vertex P_i do
2:     e ← empty sorted list of edges
3:     for each vertex P_j ≠ P_i do
4:         u ← index of end vertex of e_j
5:         L ← length of e_j
6:         occluded ← false
7:         for each edge E_k with start vertex P_i do
8:             v ← index of end vertex of E_k
9:             if d(P_u, P_v) < L then
10:                occluded ← true
11:            end if
12:        end for
13:    end for
14: end forreturn E
```

不过此时我们要求查询点 Q 必须在数据集中，这样的要求限制了算法的运用场景。所以，我们将阻塞的定义修改为：边 (p_1, p_2) 阻塞边 (p_1, p_3) 当且仅当 $d(p_1, p_2) < d(p_1, p_3)$ 并且 $d(p_2, p_3)^2 < d(p_1, p_3)^2 - 2\tau d(p_1, p_2)$。此时再使用算法5–2即可构建寻找与查询点 Q 距离小于 τ 的最邻近点的邻近图。

需要注意的是，如果使用了新的阻塞定义，会增加理想图中每个点的度，使得整个图的存储代价变大，同时也会使一次搜索的时间效率下降。因此，在实际使用时，需要根据需要权衡 τ 的取值。如果 τ 值选的过小，则很可能降低可能的最邻近点的候选数量；如果 τ 值选的过大，则很可能增大存储图所需要的空间并降低搜索的时间效率。

5.3.3 使用回溯进行快速搜索

一个提高搜索效率的方法通过修改算法5–1。新的算法5–3不再像算法5–1当无法继续时终止，而是使用了深度优先的回溯搜索从第二靠近的点出发，检查依然没有被搜索过的边。如果第二近的点所有的边都被搜索过了，那么检查第三靠近的点，以此类推。具体实现时需要维护一个尚未搜索过边的优先级队列。搜索一条边包括了首先计算查询点到有向边的终点的距离，然后根据距离长短替换优先级队列中的边，短边优先。

这里需要权衡的一项参数是距离的计算次数，当计算次数达到一个上限 M 后，返回距离查询点最近的点。

算法 5–3 快速搜索算法[3]

输入： P 邻近图的顶点集，E 邻近图的边集，Q 查询点，v 初始查询下标，M 最大计算次数
输出： n 最邻近下标

```
X ← empty priority queue
add edge e_0 with start vertex P_v to X
m ← 1
n ← v
while m < M do
    e_i ← remove top of X
    u ← index of end vertex of e_i
    if P_u has not been searched yet then
        add edge e_0 with start vertex P_u to X
        m ← m + 1
        if d(Q, P_u) < d(Q, P_n) then
            n ← u
        end if
    end if
    v ← index of start vertex of e_i
    if i < number of edges with start vertex P_v then
        add edge e_{i+1} with start vertex P_v to X
    end if
end whilereturn n
```

5.3.4 邻近图的快速构建

由于对 n 个点中的每个点产生的 n 种距离列表进行了排序，算法5–2构建图的时间复杂度为 $O(n^2 \log n)$，这样的时间复杂度使得建立邻近图的时间代价非常高，因此我们需要一种更有效的思路来构建邻近图。

这种思路使用了同时使用了两种不同的方法。其中第一种方法首先从数据集中随机选取了两个点 v_1 和 v_2，然后使用算法5–1尝试从 v_1 出发搜索 v_2。如果目前的图为空图，这种方法会直接在 v_1 和 v_2 之间添加一条边。如果不是空图，则需要检验新加入的边是否造成了边的阻塞，如果有阻塞发生，则将较长的边移除以消除阻塞。如果数据集的大小为 N, 重复这种操作直至 $50N$ 次，直至调用算法5–1的平均成功率超过 90% 为止。

当算法5–1被不断调用时，图的构建过程会渐渐减慢，此时使用第二种方法来进一步提升图的完整性。第二种方法调用搜索算法返回几千个点的近邻点以构建近邻列表。遍历这个近邻列表并调用算法5–2去除阻塞边。并行对数据集中的每个点都应用这种方法可以快速高效的构建邻近图。

5.3.5 复杂度

基于邻近图的近似搜索算法的时间复杂度约为 $O(N^{0.2})$，其中 N 为数据集中的数据量[3]。因此，这种方法符合我们的系统准确、快速、高效、高维的要求。

第六章 系统架构设计

6.1 概述

无感知人脸识别系统的架构设计主要分为三部分：数据处理流程、设备网络拓扑结构和系统逻辑架构。其中，数据处理流程主要描述系统中的数据存储结构，数据格式和数据流动情况。设备网络拓扑结构主要描述系统中所有机器的物理部署特征，网络拓扑等。系统逻辑架构主要描述系统实现代码中所有的模块和组件，以及不同模块组件之间的交互与职能分配。

6.2 数据处理流程

无感知人脸识别系统的数据处理流程如图6–1所示：

图 6–1 系统数据处理流程图

Figure 6–1 System data architecture

图6–1中标注为 1 的部分表示视频流的帧截取过程；标注 2 的部分代表人脸区域检测的过程；标注 3 的部分代表特征提取的过程；标注 4 的部分代表人脸匹配的过程。

最先进入系统的是由摄像设备和嵌入式设备获取的视频流数据，这部分数据在本地进行帧截取操作后会将获得的图片上传到图片队列中。图片队列是一个被服务器维护的优先级队列，其长度根据系统的需求和服务器的性能确定。每接收到一张新的图片时，首先检查队列是否已满。如果队列已满，则将该图片丢弃，并向用户发送服务器繁忙的信息。如果队列未满，则根据该图片的优先级将该图片插入到队列中。队列中的每张图片都应有唯一的标记，以表明这张图片的发送源。

进行人脸检测时，每次从图片队列的前端取出一张图片进行检测，检测完成后根据图片标记将得到的人脸区域做好标记插入到人脸队列中。如果人脸队列已满，则将带标记的人脸区域存储在服务器中的特定硬盘区域。当人脸队列有空余位置时，从该硬盘区域中读取存储的内容加入人脸队列的尾部，并删除硬盘中的相应内容。

进行特征提取时，每次从人脸队列的前端取出批尺寸大小的人脸图像及其标记加载到特征提取模块中，得到批尺寸大小的特征向量及标记。接着将这些特征向量及其标记发送到人脸数据库中进行匹配。由于不同的请求之间没有数据依赖性，匹配模块可以在高并发量的情况下运行，得到匹配的身份信息与每条信息的初始标记。

最后，根据每条身份信息的标记，将某一用户请求的所有身份信息返回给用户。

6.3 设备网络拓扑结构

无感知人脸识别系统的设备网络拓扑结构如图6–2所示：

图 6–2 系统设备网络拓扑结构图

Figure 6–2 System physical architecture

图6–2中的标注 1 的部分表示摄像设备与 GPU 服务器集群之间通过有线网络连接，数据从摄像设备传输到 GPU 服务器集群；标注 2 表示嵌入式设备与 GPU 服务器集群之间通过无线网络连接，数据从嵌入式设备传输到 GPU 服务器集群；标注 3 表示 GPU 服务器集群与人脸数据库之间通过内部的有线网络连接；标注 4 表示人脸数据库与交互服务器通过内部的有线网络连接。

其中，人脸检测模块和特征提取模块部署在 GPU 服务器集群中，人脸匹配模块部署在人脸数据库中。交互服务器负责分析匹配结果，并将分析后的信息发送到相应的用户设备中。

在系统的实际部署中，由于摄像设备和嵌入式设备的输入众多，网络拓扑结构会更为复杂，网络中很可能需要增加一些中间节点。如果 GPU 服务器集群不在本地而在云端，则整个网络传输就无法在一个局域网内进行。此时则需要考虑网络带宽对数据传输的影响。如果网络带宽不足，可会影响获取到的图片的传输速度，进而影响总体的识别速度。同时，如果 GPU 服务器集群和人脸数据库之间的传输需要经过公有网络，则考虑网络带宽的同时还需要考虑传输安全的问题。

6.4 系统逻辑架构

无感知人脸识别系统的系统逻辑架构如图6–3所示：

图 6–3 系统系统逻辑架构图

Figure 6–3 System logical architecture

图6–3中标注为 1 的部分为 GPU 服务器集群的负载均衡模块；标注为 2 的部分为使用 Docker 封装的人脸检测和特征提取模块；标注为 3 的部分为人脸数据库服务器的负载均衡模块。

图像获取模块共有两种实现，一种部署到连接摄像设备的盒子中，另一种部署在嵌入式设备上。两种实现的功能都时从视频流中截取图像信息。根据实际需要，截取信息的操作可以是定时自发执行的，也可以是由用户或者服务器请求执行的。

通过第三章的对比，经过 CFDDB 的测试，SSH 检测器[1] 是目前唯一符合我们系统人脸检测需求的人脸检测器。因此，人脸检测模块中的算法优先选择 SSH 检测器。部署时需要考虑每一个 SSH 检测器需要的显存空间以最大化利用 GPU 服务器集群中的资源。如果未来有更优的算法，可以直接将其模块化，实现需要的接口后进行替换，而无需重写整个系统。

通过第四章的叙述，InsightFace[2] 符合我们系统的要求，因此将其封装为特征提取模块。一个较好的系统设计是将人脸检测模块和特征提取模块使用同一个 Docker 容器封装，外部程序通过使用 GRPC 框架调用容器内部的程序进而进行检测和识别。这样当算法有更新时，可以将更新完的 Docker 容器重新部署，加速部署的过程，减少服务器的停机时间。

如果人脸检测或特征提取的算法使用了 TensorFlow 实现，可以使用 TFserving 进行封装。使得算法和模型的更新与新旧版本的管理变得更加便捷。

如果同时间有大量的匹配请求，在匹配模块之前的负载均衡模块也必不可缺。如果实际中人脸数据库较小，匹配模块内部可以采用线性搜索的方法。而如果数据库非常大，则可以使用近似搜索的算法加快搜索的效率。

全文总结

本文针对无感知环境下的大规模人脸数据的模糊搜索，从五个方面开展研究：

第一，建立无感知人脸检测的数据集 CFDDB。并在 CFDDB 上定义召回率 *recall*、误判率 *errorrate* 与检测参数 α 等测试基准。之后进行人脸数据属性分析，探讨数据集的改进方法，制定数据集的扩充流程。

第二，根据 CFDDB 测试集的基准定义，完成四种不同的人脸检测算法的封装测试与参数调优。根据测试结果，选择 SSH 检测器[1] 作为系统人脸检测模块的核心。

第三，利用 CFDDB 中的数据，对 InsightFace[2] 进行测试，并将其封装为系统的人脸识别模块。

第四，针对小规模数据匹配，提出线性搜索和多类支持向量机两种方案。针对大规模数据匹配，研究基于局部敏感哈希与基于有向图的两种近似搜索算法，并提出优化思路。

第五，高效、稳定、可扩展的系统架构设计。

综合以上五方面的研究成果，最终在 GPU 服务器上实现了无感知人脸识别系统。

参考文献

[1] NAJIBI M, SAMANGOUEI P, CHELLAPPA R, et al. Ssh: Single stage headless face detector[C]// Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. [S.l.]: [s.n.], 2017: 4875–4884.

[2] DENG J, GUO J, ZAFEIRIOU S. Arcface: Additive angular margin loss for deep face recognition[J]. ArXiv preprint arXiv:1801.07698, 2018.

[3] HARWOOD B, DRUMMOND T. Fanng: Fast approximate nearest neighbour graphs[C]// Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. [S.l.]: [s.n.], 2016: 5713–5722.

[4] VIOLA P, JONES M J. Robust real-time face detection[J]. International journal of computer vision, 2004, 57(2): 137–154.

[5] DALAL N, TRIGGS B. Histograms of oriented gradients for human detection[C]// Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on. Vol. 1. IEEE. [S.l.]: [s.n.], 2005: 886–893.

[6] SIMONYAN K, ZISSERMAN A. Very deep convolutional networks for large-scale image recognition[J]. ArXiv preprint arXiv:1409.1556, 2014.

[7] 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. [S.l.]: [s.n.], 2016: 770–778.

[8] HU P, RAMANAN D. Finding tiny faces[C]// 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR). IEEE. [S.l.]: [s.n.], 2017: 1522–1530.

[9] HUANG G B, RAMESH M, BERG T, et al. Labeled faces in the wild: A database for studying face recognition in unconstrained environments[R]. Technical Report 07-49, University of Massachusetts, Amherst, 2007.

[10] YANG S, LUO P, LOY C.-C, et al. Wider face: A face detection benchmark[C]// Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. [S.l.]: [s.n.], 2016: 5525–5533.

[11] ZHANG K, ZHANG Z, LI Z, et al. Joint face detection and alignment using multitask cascaded convolutional networks[J]. IEEE Signal Processing Letters, 2016, 23(10): 1499–1503.

[12] BRADSKI G. The OpenCV Library[J]. Dr. Dobb's Journal of Software Tools, 2000.

[13] JIA Y, SHELHAMER E, DONAHUE J, et al. Caffe: Convolutional Architecture for Fast Feature Embedding[J]. ArXiv preprint arXiv:1408.5093, 2014.

[14] Martn Abadi, Ashish Agarwal, Paul Barham, et al. TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems. Software available from tensorflow.org. 2015. `https://www.tensorflow.org/`.

[15] NICKOLLS J, BUCK I, GARLAND M, et al. Scalable parallel programming with CUDA[C]// ACM SIGGRAPH 2008 classes. ACM. [S.l.]: [s.n.], 2008: 16.

[16] CHETLUR S, WOOLLEY C, VANDERMERSCH P, et al. Cudnn: Efficient primitives for deep learning[J]. ArXiv preprint arXiv:1410.0759, 2014.

[17] JAIN V, LEARNED-MILLER E. FDDB: A Benchmark for Face Detection in Unconstrained Settings[R]. UM-CS-2010-009. University of Massachusetts, Amherst, 2010.

[18] YANG S, LUO P, LOY C.-C, et al. From facial parts responses to face detection: A deep learning approach[C]// Proceedings of the IEEE International Conference on Computer Vision. [S.l.]: [s.n.], 2015: 3676–3684.

[19] SHRIVASTAVA A, GUPTA A, GIRSHICK R. Training region-based object detectors with online hard example mining[C]// Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. [S.l.]: [s.n.], 2016: 761–769.

[20] HOWARD A G, ZHU M, CHEN B, et al. Mobilenets: Efficient convolutional neural networks for mobile vision applications[J]. ArXiv preprint arXiv:1704.04861, 2017.

[21] SANDLER M, HOWARD A, ZHU M, et al. Inverted Residuals and Linear Bottlenecks: Mobile Networks for Classification, Detection and Segmentation[J]. ArXiv preprint arXiv:1801.04381, 2018.

[22] PARKHI O M, VEDALDI A, ZISSERMAN A, et al. Deep Face Recognition.[C]// BMVC. Vol. 1. 3. [S.l.]: [s.n.], 2015: 6.

[23] CAO Q, SHEN L, XIE W, et al. VGGFace2: A dataset for recognising faces across pose and age[J]. ArXiv preprint arXiv:1710.08092, 2017.

[24] YI D, LEI Z, LIAO S, et al. Learning face representation from scratch[J]. ArXiv preprint arXiv:1411.7923, 2014.

[25] BANSAL A, NANDURI A, CASTILLO C D, et al. Umdfaces: An annotated face dataset for training deep networks[C]// Biometrics (IJCB), 2017 IEEE International Joint Conference on. IEEE. [S.l.]: [s.n.], 2017: 464–473.

[26] GUO Y, ZHANG L, HU Y, et al. MS-Celeb-1M: A Dataset and Benchmark for Large Scale Face Recognition[C]// European Conference on Computer Vision. Springer. [S.l.]: [s.n.], 2016.

[27] NECH A, KEMELMACHER-SHLIZERMAN I. Level Playing Field For Million Scale Face Recognition[C]// Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. [S.l.]: [s.n.], 2017.

[28] CHEN T, LI M, LI Y, et al. Mxnet: A flexible and efficient machine learning library for heterogeneous distributed systems[J]. ArXiv preprint arXiv:1512.01274, 2015.

[29] WEBER R, SCHEK H.-J, BLOTT S. A quantitative analysis and performance study for similarity-search methods in high-dimensional spaces[C]// VLDB. Vol. 98. [S.l.]: [s.n.], 1998: 194–205.

[30] LV Q, JOSEPHSON W, WANG Z, et al. Multi-probe LSH: efficient indexing for high-dimensional similarity search[C]// Proceedings of the 33rd international conference on Very large data bases. VLDB Endowment. [S.l.]: [s.n.], 2007: 950–961.

[31] ASUNCION A, NEWMAN D. UCI machine learning repository. 2007.

致　谢

感谢沈耀老师提供实验场所和悉心指导！
感谢郁东风同学提供训练好的人脸识别模型！
感谢沈国栋学长在实验环境搭建和服务器搭建给予我的帮助！
感谢龚桂学长在小脸检测和嵌入式设备运行方面给予我的帮助！
感谢孙泽堃、郭远帆、丁志勇和罗翔中同学志愿参与 CFDDB 测试集的标注！
感谢其他所有帮助我的同学和老师！

EFFICIENT FUZZY SEARCH OF LARGE-SCALE DATA

Fuzzy search is a type of search method that searches for the same or similar data. This paper focuses on the fuzzy search of large-scale face data, from the aspects of face detection, feature extraction and vector matching, and finally implements a complete imperceptible face recognition system.

Imperceptible face recognition system is a face recognition system which does not require intentional cooperation from users. This system can be widely used for student attendance management and access control in companies because it requires less human resources and is time-efficient.

This system pulls live images of faces from CCTV or embedded devices at first. Then images are sent to GPU server at where face detection, face counting and face recognition programs run. After that, face embeddings extracted by face recognition module are sent to vector matching module which performs nearest neighbor search. Recognition module also connects each face embedding with an identity. Finally, deliver module sends identities required back to user.

The research work is mainly divided into four phases. The main research contents and achievements of each phase are as follows:

The first phase is to establish CFDDB dataset. Before choosing the right face detection and recognition algorithm, we should choose or build a benchmark which is suited for imperceptible face recognition. In the area of face detection, there have been a large number of public data sets for training and testing. Commonly used are WIDER FACE[10], FDDB[17], LFW[9] and so on. However, the faces in these data sets are mostly European and American, and the number of faces in each photo is also limited. The images extracted from the surveillance videos are mostly Asian, and the faces are often blurred, angled, or occluded. Therefore, the existing test set cannot meet our test requirements. According to this, we have established our own test set CFDDB (Classroom Face Detection Database).

The test set images were generated by capturing one frame every 100 frames from a 15-minute surveillance video and has 149 images. After preprocessing, there were 134 images available in the test set, with 1,087 faces and a total of 84 people. Analysis of the 7 images taken in the sample shows that the angled face images in CFDDB are more than 70% and occluded faces are about 58%. These features make the CFDDB test set a great challenge for any face detection algorithm and recognition algorithm.

There are five parts in the second phase. First, we encapsulate the cascading classifier[4] based on the HAAR feature, implement a CFDDB-based test interface, and analyze its performance based on recall rate and time efficiency. Second, MtCNN[11] is packaged, accelerated using a GPU server, and its performance is analyzed based on its network structure and functionality. Third, test the SSH detector[1], implement a CFDDB-based test interface, and perform parameter tuning. Fourth, test the Tiny face detector[8], implement the CFDDB-based test interface, and compare the results after the SSH detector is tuned. Fifth, test the high-precision face recognition algorithm InsightFace[2] to ensure that it has a high recall rate on CFDDB.

After completing this phase of work, we have achieved the following three points of research results. First, the HAAR cascade classifier and MtCNN recall rate are too low to meet the requirements. Second, Tiny face detectors are not time-efficient. Finally, even without considering the time efficiency, the SSH detector is still better than the Tiny face detector when considering the recall rate and the misjudgment rate comprehensively. Therefore, the SSH detector is selected as the core algorithm of the face detection module.

The main content of the next phase is the design and optimization of face embedding matching algorithms. First, the matching algorithm for small-scale datasets is designed and analyzed based on its time efficiency and space efficiency. Second, matching algorithms for large-scale data sets are designed and we provide some optimization strategies for them.

We have four requirements for a qualified face embedding matching algorithm:

The first is accuracy. The result of a query should be very close to the result of a linear search. The second is fast. The time complexity of the algorithm should not be more complicated than $O(n)$. The third is high efficiency, that is, ideally, the index of data should be linear to the size of dataset. The fourth is high dimension. Since the dimensions of face embedding vectors are often more than 100, the algorithm needs to support efficient search for high-dimensional features.

In addition to linear search, search algorithms are usually based on tree-structured algorithms such as R-tree and KD tree. However, when the dimension exceeds 10, the time efficiency of these precise algorithms has exceeded linear search[29]. Since the dimension of the face embedding vector is basically at the level of 100. Using these exact search algorithms cannot meet the high-dimensional requirements, these algorithms are not suitable for the system's matching algorithm.

In the field of machine learning, support vector machines (SVMs) are often used for the classification and regression of high-dimensional vectors. In the matching phase of face embeddings, if there are n embedding vectors in the face feature database, then we use SVM to find the embedding vector matched by the query vector $\mathbf{q}$ in two ways:

The first way is to train n classifiers. Each classifier distinguishes one embedding vector in the face feature database from all remaining embedding vectors. In use, each classifier is required to classify the query vector $\mathbf{q}$. If $\mathbf{q}$ is in the same class with a embedding vector, then the embedding vector is marked as $\mathbf{q}$ matching candidates.

The second way is to train C_n^2 classifiers to distinguish any randomly selected pair of embedding vectors from the n embedding vector database. When we need to find a match for the query vector $\mathbf{q}$, we use each trained classifier to classify $\mathbf{q}$, each time adding the weight of the embedding vector category represented by the output of the classifier. When all the classifiers have been classified, the embedding vector in the class with the highest weight is selected as the matching result.

These two ideas are feasible when database size is not large. In some simple face recognition systems, it is very common to use multi-class SVMs to match high-dimensional face embedding vectors.

We notice that face embedding vectors have following feature. Faces with the same identities will generate embedding vectors with closer distances, while faces with different identities will often produce embedding vectors with relatively long distances. Thus, the matching process of the face embedding vector

can be equated to finding the nearest neighbor to the query vector **q**. Through this transformation, we will match the problem to the nearest neighbor search problem.

LSH-based approximate search algorithm or directed-graph based algorithm to match large-scale data. At the same time, the optimization method of LSH-based approximate search algorithm is discussed.

The final phase is system architecture design. The architecture design of the non-perceived face recognition system is mainly divided into three parts: data processing flow, device network topology and system logic architecture. Among them, the data processing flow mainly describes the data storage structure, data format and data flow in the system. The device network topology mainly describes the physical deployment features and network topology of all the machines in the system. The system logic architecture mainly describes all the modules and components in the system, as well as the interaction and function assignment between different module components.

The first to enter the system is the video stream data acquired by the camera and the embedded device. This part of the data is uploaded to the image queue after the frame is locally intercepted. The image queue is a priority queue maintained by the server. Its length is determined based on the system's requirements and the server's performance. Whenever a new image is received, first check if the queue is full. If the queue is full, the image is discarded and the server is sent a busy message back to user. If the queue is not full, the image is inserted into the queue according to the priority of the image. Each image in the queue should have a unique tag to indicate the source of the image.

When face detection is performed, an image is taken from the front of the image queue for detection each time. After the detection is completed, the obtained face region is marked and put into the face queue according to the image tag. If the face queue is full, the marked face area is stored in a specific area in the server. When there are free positions in the face queue, the stored content is read from the hard disk and added to the tail of the face queue, and the corresponding content in the hard disk is deleted.

When face recognition is performed, batch-sized face images and their tags are loaded from the front end of the face queue into the face recognition module to obtain batch size feature vectors and tags. These feature vectors and their tags are then sent to the face database for matching.

Finally, according to the tag of each piece of identity information, all identity information requested by a certain user is returned to the user.

There are two implementations of the image acquisition module, one deployed to the box that connects the camera devices and the other to the embedded device.

Through the comparison of the second phase, the SSH detector[1] is face detector that meets our system's face detection requirements. The memory space needed for each SSH detector needs to be considered during deployment to maximize the utilization of resources in the GPU server.

InsightFace[2] is packaged as a face recognition module. A good system design is to use the same Docker container for the face detection module and the face recognition module. The external program then uses the GRPC framework to call the internal program of the container to perform detection and recognition.

If the algorithm for face detection or face recognition uses TensorFlow as framework, TFserving can be used for encapsulation which make algorithm update and model management easier.

In the end, we implement the non-perceived face recognition system and keep it highly efficient, stable, and scalable.