廣東工業大學

本科毕业设计（论文）

基于Python的人脸识别系统设计与实现

学　　院　信息工程学院

专　　业　信息工程

（电子信息工程方向）

年级班别　2015级（7）班

学　　号　3115002473

学生姓名　陈以勒

指导教师　杨志景

2019年 6 月

摘 要

随着人工智能的发展,人脸识别系统在我们的生活中越来越被广泛应用。人脸识别系统是指能够从数字图像或视频源中识别人的技术。人脸识别系统可以通过多种方法工作，但是，它们通常是通过将给定图像中的面部特征与数据库中的人脸进行比较来工作的。人脸识别技术以前是作为一种计算机应用程序而发展起来的，但现在它在不同的移动平台上得到了广泛的应用，如人脸识别系统是一种比较常用的安全机制，可以与指纹和眼睛识别技术等生物识别系统进行比较。通过这项技术，用户可以轻松地解锁他们的移动电话，而不必输入他们的密码。同时它也被广泛应用于各种技术，如机器人开发等等..鉴于人工智能的主流发展,本设计主要对人脸识别系统进行开发和拓展。

本设计应用 python 语言编写程序，主要用到开源的 dlib 图像处理库。读取图片/视频或打开摄像头后，首先调用正脸检测器检测出人脸， 再用已经训练好的人脸关键点检测器定位人脸特征点，接着用 ResNet 人脸识别模型获得人脸的 128 个向量特征，再与数据库文件下所有 jpg 图片比较人脸的特征向量值并打印出来结果，结果小于阈值则时，判定为同一个人并弹出提示框显示识别结果，否则显示无识别结果。

除了基本的人脸识别功能，本设计还拓展了三个功能:美颜功能，瘦脸功能和人脸计数。界面上设置三个复选框，选定后读取图片或打开摄像头,可实现各个功能。

美颜功能主要是对人脸进行磨皮去痘，实际上是进行模糊处理。瘦脸功能则改变人的脸型,主要对相应特征点进行分割和压缩。而人脸计数返回图像中的人数。相关的算法会在下文中详细研究。

关键词：人工智能，Python，人脸识别，人脸美化

ABSTRACT

With the development of artificial intelligence, face recognition system is widely used in our life. Face recognition system refers to the technology that can recognize people from digital images or video sources. Face recognition systems can work in a variety of ways, but they usually work by comparing facial features in a given image to faces in a database. Face recognition technology used to be developed as a computer application, but now it has been widely used in different mobile platforms. For example, Facial recognition system is more commonly employed as a security mechanism and can be compared to biometric systems such as fingerprint and eye recognition technology. A more recent use of this technology is in the mobile world, where latest mobile phone models have a built-in face recognition application. Through this technology, users can easily unlock their mobile phones without having to enter their password. It is also widely used in various technologies, such as robot development, etc. In view of the mainstream development of artificial intelligence, this design mainly develops the face recognition system and expands its functions.

This design uses the python language to write programs and mainly using the open source image processing library : dlib. After reading the picture/video or opening the camera, first call the face detector to detect the face, then use the trained face key point detector to locate the face feature points, and then use the ResNet face recognition model to obtain the 128 vector features of the face, after that, compare the feature vector values of the face with all the jpg images in the database file and print the results. If the result is less than the threshold, it is determined to be the same person. Meanwhile a prompt box will be shown to display the recognition result.

In addition to the basic face recognition function, the design also expands three functions: face beautification, face-lifting function and face counting. Set three check boxes on the interface. After selecting the image or opening the camera, it can implement corresponding functions.

The beauty function is mainly to skin the face to remove acne, in fact, it is blurred. The face-lifting function is to change the face shape, mainly to segment and compress the corresponding feature points and the face count returns the number of people in the image. The relevant algorithms will be studied in detail below.

Keywords: Artificial Intelligence, Python, Face recognition, Face beautification

目　录

1 绪论

1.1本设计的目的和意义

面孔是我们在社会生活中首要关注的焦点，在传递身份和情感方面起着重要的作用。在我们的一生中，我们可以识别出许多面孔，并且即使在多年的分离之后，我们也能一眼识别出面孔。即使由于不断变化的条件，包括年龄和外界干扰干扰，如胡须，眼镜或发型使视觉刺激存在较大差异，这项功能仍然是十分强大的。

随着科技的发展，人类社会对人工智能的渴望程度越来越大。人工智能的发展可以提供我们极大的便利。特别是可检测和识别人脸的计算机可以应用于各种任务，包括刑事识别、安全系统、图像和胶片处理、身份核实和人机交互。不幸的是，建立人脸检测和识别的计算模型相当困难，因为人脸是复杂的、多维的和有意义的视觉刺激。现在很多地方都在使用人脸检测，特别是像 picassa、photobucket 和 facebook 这样的图片网站。自动标记功能为图片共享创造了一个新的维度，让其他人了解到在图片中的人是谁。作为图像分析和理解最成功的应用之一，人脸识别受到了人们的重视，尤其是在过去的几年里。造成这一趋势的原因至少有两个：第一是商业和执法应用范围广泛，第二是经过 30 年的研究后可行技术的可获得性。此外，人脸的机器识别问题继续吸引图像处理、模式识别、神经网络、计算机视觉、计算机图形学和心理学等学科的研究人员。对于用户友好型系统的强烈需求是显而易见的，这种系统能够保护我们的资产，保护我们的隐私，而不会因为一个数字失去我们的身份。比如，一个人需要一个 PIN 从 ATM 机获取现金，一个计算机的密码来访问互联网，等等。而且对于那些视网膜、虹膜扫描或者指纹识别，它强烈需要识别人的配合，比如睁开双眼进行扫描、录入指纹等等，即使这些特征十分适合于识别人类这种生物。相比而言，人脸识别系统通常能在识别人不知情的情况下进行，那么就不需要进行强迫也能实现识别功能，因此可更广泛的应用于各个领域。如:

1. 法律监督和执法:带有人脸识别功能的监控，预防盗窃和非法行为的发生、发生盗窃事件或其他非法行为后进行分析推理、识别到犯法人后在各个路口和海关进行人脸捕获，以此抓捕到犯罪嫌疑人。

2. 信息安全:智能手机解锁屏幕、手机支付(包括 Apple pay、支付宝、微信支付等)、各种社交软件的注册和登录、数据库登录、互联网登录、医疗和交易记录等等。

3. 智能卡:代替国家身份证、护照、社保卡、手机卡、银行卡、驾驶证等等。

4. 娱乐:视频游戏、虚拟现实、VR 真人游戏、室内人工智能。

与此同时，随着人脸识别技术慢慢成熟和数字图像处理技术不断的发展，人脸自动美化技术也逐渐受到人们的青睐。

1.2系统开发背景

在这个世纪的大部分时间，甚至上个世纪的部分时间(达尔文，1872 年)，对人脸识别的兴趣在各个科学学科中都发挥着重要的作用，认知心理学家对人脸识别这一现象很感兴趣，因为有证据表明，人们对人脸的感知与其他有图案的物体不同，因此，人脸可能代表了一类“特殊”的刺激。

人脸识别作为图像分析与理解中最成功的应用之一，近年来受到了广泛的关注。至少有两个原因可以解释这一趋势:第一个是广泛的商业和执法应用，第二个是经过 30 年的研究，可行技术的可用性。根据早期科学家对人脸识别的研究，人脸识别其实就是检测和提取人脸特征点，根据特征点进行识别。人脸识别技术在我国也被广泛的应用。人脸识别不仅在互联网、交通、安全等领域占据发展主流，同时还在建筑设计、金融、医疗保险、海外旅游移民等方面也得到广泛的应用和认可。

虽然目前的机器识别系统已经达到了一定的成熟程度，但是它们的成功受到许多实际应用所施加的条件的限制。例如，在光照和/或姿态发生变化的户外环境中获得的人脸图像的识别在很大程度上仍然是一个未解决的问题。换句话说，目前的系统距离人类感知系统的能力还很远。

因为国家的昌盛，我们的生活水平在不断的提高，我们不再满足于物资生活，精神需求也越来越广泛，比如我们更追求外表的美观。以至于现在市面上出现各种类型的 PS 软件。人们可以借助此类软件来处理自己的皮肤，或者身材，实际上就是使用了相应的图像处理技术来美化人脸或者身材，能展现更美丽的自己，这受众多女性朋友的青睐。于此同时，随着数码相机的发展，用相机拍下的照片的像素，或者说分辨率也越来越高，以至于能够清晰的展示出人脸的各个细节，当然包括了青春痘、雀斑、皱纹等等，这就暴露了一些人们不想暴露的点。因为爱美是人的本性，所以在得到这些照片后，大部分女性朋友都会在 PS 软件上重新进行美化处理，但是 PS 过程是相当繁琐和枯燥的，如果需要 PS 多张人脸，则需要重复一样的步骤，这真的十分浪费时间。事实上为了迎

合大众的需求，一部分数码相机，比如柯达相机中，就将这两个步骤结合在一起了。也就是在数码相机上加入自动人脸美化或者瘦脸的功能，人们只要拍照，拍摄完后相机会自动的处理图片中的人脸。怎么处理呢?其实就是利用数字图像处理的一些算法，去除人脸上的痘痘、雀斑、皱纹等等，达到磨皮的作用，同时又不会去除掉人脸的特征。除此之外还能够实现自动瘦脸等功能。

1.3存在的问题

虽然目前的机器识别系统已经达到了一定的成熟程度，但是它们的成功受到许多实际应用所施加的条件的限制。在变化的户外环境中获得的人脸图像的识别在很大程度上仍然是一个未解决的问题。换句话说，目前的系统距离人类感知系统的能力还很远。

例如影响人脸识别的诸多因素：

（1）复杂场景

图 1.1　复杂场景下的人脸识别

（2）表情变化

图 1.2 表情变化下人脸发生变化

（3）光照变化：

图 1.3 光照方向不同时人脸产生的不同视觉效果

（4）年龄变化：

图 1.4 随着年龄变化人脸的改变

（5）姿态变化：

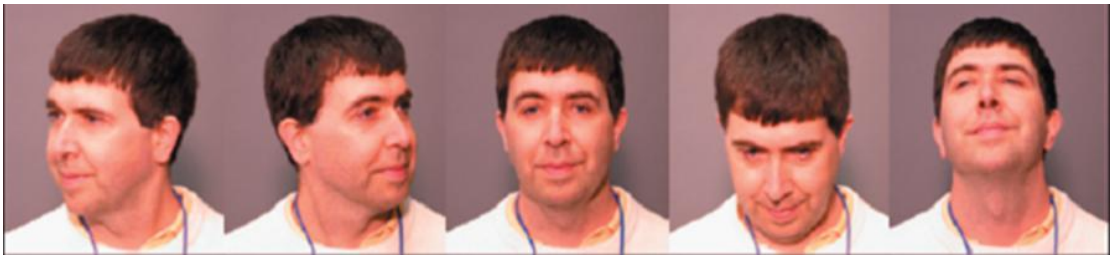


图 1.5 各种姿态变化

变化是绝对的，不变是相对的。模式识别(图像识别)的基本任务就是从变化中寻找不变的本质。图像处理不是科学，是门艺术。同时，虽然美化，瘦脸等功能技术已有产品上市，但是美化的同时会使图像有事产生过度模糊、或者人脸变形的现象，因此总的来说这类技术还是不够成熟，需要我们更深入的研究。

1.4.需达到的技术要求

本设计主要要求使用 python 语言实现人脸识别功能，因为本设计的研究方向不是人脸识别的底层算法，所以使用已经训练好的模型进行人脸识别。其中使用了 dlib 库中的人脸检测方法，它的算法可以是 HOG(HISTOGRAM OF ORIENTED GRADIENT)方向梯度直方图算法或者卷积神经网络方法、于此同时利用 2015 年提出的深度残差网络(ResNet)人脸识别方法。它的识别速度不仅快，可以达到 10 帧每秒，而且识别精度也相对较高，因此在开发中此方法深受人工智能爱好者的青睐[2]。

除此了基础的人脸识别功能，本设计还要实现的功能有：人脸美化、人脸计数及瘦脸功能。详细的算法和实现会在下文中提到。

1.5指导思想

人脸识别的商业和执法应用范围从静态、控制格式的照片到不受控的视频图像，构成了广泛的技术挑战，需要同样广泛的技术，从图像处理、分析、理解和模式识别。人脸识别问题的一般表述为：使用现有的、存储人脸的数据库，对给定的图像或者视频中的人脸进行检测，识别场景中的一个或多个人并返回结果，做出响应。可用诸如人种、年龄、性别、面部表情或语言等附带信息可用于缩小搜索范围（增强识别）。该问题的解决方案涉及从杂乱场景中分割人脸（人脸检测）、从人脸区域中提取特征、识别或验证（图 1.6）。在识别问题中，系统的输入是一个未知的面孔，系统从已知个体的数据库中验证和确定的身份。此外，图像处理不是科学，是门艺术。我们需要深刻领悟图像对象的特点，根据对象的本质特征选用针对性的图像处理方案。

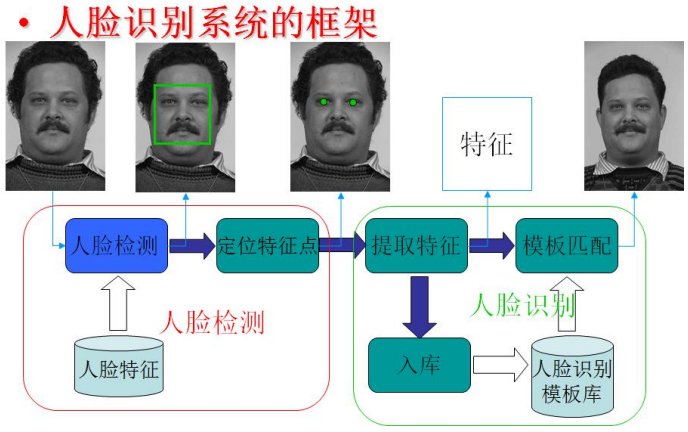


图 1. 6 人脸识别的框架

2 软件及关键技术简介

2.1 Python

Python是一种高级编程语言，同时它与java、C++语言一样也是面向对象的，但与另外的面向对象的编程语言区别却很大，使用过这几种语言的程序员都不得不赞赏Python语言的精简和方便[10]。不仅如此，Python可以同时在各个不同的操作系统中搭建环境和使用，而且用Python语言编写的代码不需要做改变，所以它给程序员提供的功能是毫无质疑的。此外，Python另外的强大之处在于，它的应用领域范围十分广，遍及人工智能、科学计算、Web开发、系统运维、大数据及云计算、金融、游戏开发等。为什么Python能实现这么强大的功能呢？主要是因为它具有庞大的库(包括标准库和第三方库)，并且它大部分的库都是开源的，开源的库经过不断的使用和完善是使Python持续发展的主要原因。通过引用库来实现不同领域的各种功能。然而，于此同时，正是由于库的数量庞大，对于管理这些库以及对库作及时的维护成为既重要但复杂度又高的事情。

Python是一种由Guido van Rossum开发的通用编程语言，它很快就变得非常流行，主要是因为它的简单性和代码可读性。它使程序员能够用更少的代码行表达思想，而不会降低可读性。与C／C++等语言相比，Python速度较慢。也就是说，Python可以使用C／C++轻松扩展，这使我们可以在C／C++中编写计算密集型代码，并创建可用作Python模块的Python包装器。这给我们带来了两个好处：首先，代码与原始C ／ C++代码一样快（因为它是在后台工作的实际C++代码），其次，在Python中编写代码比使用C／C++更方便和容易。

2.2 Anaconda

Anaconda的中文是森蚺，一种非常肥大的蟒蛇。简单来说你可以把Anaconda当作是Python的懒人包，支持Linux, Mac, Windows系统, 包含了Python常用的资料分析、机器学习、视觉化的套件。并且自带了专门用来解决软件环境依赖问题的 conda 包管理系统。可以在Anaconda下随意切换python的版本，并且可以在Anaconda prompt下方便的安装第三方包，其实在下载时就已经自动为我们装上除了标准库外其他常用的库了。比较特别一点的是，Anaconda是利用conda命令来管理包和环境，使用起来十分方便[6]。

2.3 conda

conda是用来管理包和环境的管理工具。它适用的语言有：Python，R，Ruby，Lua，Scala，Java，JavaScript，C/C++，FORTRAN。conda为Python项目而创造，但可适用于上述的多种语言。适用平台：Windows，macOS，Linux。它的作用有：①快速安装、运行和升级包及其依赖项。②在计算机中便捷地创建、保存、加载和切换环境。

2.4 dlib库

Dlib是一个C++包，它拥有各种机器学习、深度学习相关的库，而且它可以在不同环境下使用，特别是在图像处理领域运用十分广泛。同时dlib所有的设计都是高度模块化的，执行速度快，并且通过一个干净的、现代的C++ API简单地使用。特别的是，因为Dlib是开源的，我们可以免费的使用它，只需要直接在环境下安装即可，十分方便。并且Dlib不依赖第三方库，无须安装和配置，可用在window、Mac OS、Linux系统上。

为什么用Dlib来做人脸识别呢？因为我们需要3个重要的模型，分别是：人脸检测器，人脸关键点检测器、人脸识别模型，而这3个模型Dlib库中都有，并且后两者还是已经训练好的，比如此设计我使用的是训练好的ResNet人脸识别模型，我们只需直接调用就行，是相当方便的了。ResNet即深度残差网络，为什么用这个模型呢？如果我们让网络对残差进行学习，得到的人脸识别结果在深度和精度上都比 CNN 更加强大。因为它的强大使他的发明者何凯明在2015年获得ImageNet的冠军，因此目前此算法在人脸识别领域占据领先位置。于此同时，Dlib的拥有许多解析文档和开源、高级的图像处理算法。Dlib有debug模式，在debug模式下可以调试代码，查看变量和对象随着程序运行的变化，快速定位错误点。同时Dlib也有说明文档，每个类和函数都在文档中有做详细的说明。除此之外，Dlib还提供了大量的实例，作为程序员重要的参考资料。

2.5 OpenCV-Python

是OpenCV的Python API，结合了OpenCV C++ API和Python语言的最佳特性。OpenCV-Python是一个Python绑定库，旨在解决计算机视觉问题。OpenCV-Python是原始OpenCV C++实现的Python包装器。OpenCV-Python使用Numpy，这是一个高度优化的数据库操作库，具有MATLAB风格的语法。

3 基本概念和理论基础

3.1 HOG(HISTOGRAM OF ORIENTED GRADIENT)特征描述子

特征描述子是人脸检测的重要元素，它可以从图像中提取有用信息，同时剔除无关信息；典型的例子就是特征描述子从一张宽度 * 高度 * 3 （ 通道数 RGB ） 大小的图像中，提取出长度为 N 的特征向量/特征矩阵。

很明显，通过特征向量用来浏览图像是没用的，但是在图像识别或者目标检测中，特征向量会变得很有用；像是一些图像分类算法比如 SVM(Support Vector Machine)，基本上都用特征向量进行分类，效果是十分好的。

但是哪些是“有用信息”，哪些又是“无关信息”呢？假设现在我们想通过一个目标检测器，来检测人脸，人脸是由五官组成，五官又构成脸的基本形状，比如眼睑、鼻子、嘴巴的位置都是固定的，于是我们可以检测到五官的边缘，并标注为特征点。根据特征点的不同来识别检测人脸。这个例子中，边缘和位置信息是“有用的”，而颜色信息是 ”无用的“；除此之外，特征也需要有足够特殊的地方。比如一个好的特征，应该能够让你辨认出人脸的区别比如高鼻梁低鼻梁，大眼睛小眼睛等。

方向梯度直方图(HOG)是特征描述子，主要用于在计算机视觉和图像处理中用来进行物体检测，再进一步进行图像操作[7]。HOG描述子技术在图像检测窗口或感兴趣区域(ROI)的局部部分中取出特征向量。HOG描述符算法的实现如下：（1）图像灰度化，因为颜色是无用信息。(2)将图分为若干个cell，每个cell为8乘8像素，然后每4个cell组成一个block，每个cell由9个bin组成。划分单元根据实际情况来决定，相邻细胞群被认为是称为block的空间区域，将cell分组成块是分组和直方图规范化的基础。（3）利用三线性插值模板和公式求每个像素点的梯度值(相邻像素之差)和梯度方向(像素强度变化的方向，度数是由0~160组成，因为梯度的方向加180不变)，如果RGB图则选择RGB中最大的值作为幅值和方向（4）对每一个cell进行加权投影，得到9维特征向量，并对block内进行归一化梯度直方图，主要是因为在实际检测中，人脸会出现在不同地点，关照程度不同，所用进行归一化可以对光照的变化范围进行压缩（5）讲所用的block的HOG特征向量归一化，得出HOG最终的特征向量得到一个由标准化直方图组表示块直方图。这些直方图的集合表示描述符。下图演示了算法实现方案：

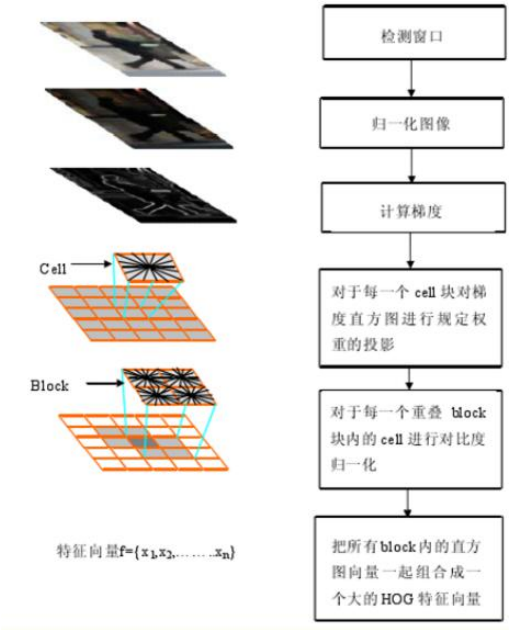


图3. 1 HOG算法实现过程

本设计就是应用dlib中提供的人脸检测方法dlib. get_frontal_face_detector()，用来检测人脸，返回值是一个矩形，坐标为[(x1， y1) (x2， y2)]，可以通过函数的left, right, top, bottom方法分别获取对应的x1， x2， y1， y2值，。其原理就是HOG算法。本论文不做深究。

3.2 深度残差网络(ResNet)

在2015年的ImageNet上，我国研究员何恺明、张翔宇、任少卿和孙剑公布了他们的研究成果深度残差网络，其实也就是深度卷积网络。为了训练更深的神经网络他们应用多个残缺块，将他们堆积在一起，以此构建一个ResNet网络，它的深度、速度和精度都是更加可观的，也因此他们靠着这个算法拿到了各项比赛的冠军，项目分别是图像分类，图像检测和图像定位。

现在简单介绍一下它的原理：比如这是一个普通的神经网络（Plain network）

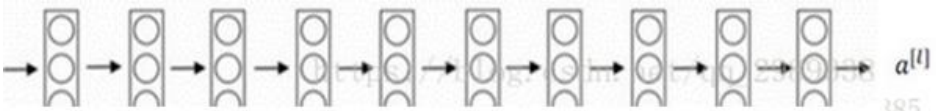


图3. 2 普通的神经网络（Plain network）

如果我们在每两层的中间插入一个跳跃点，也就是说可以第二层可以通过这个跳跃

点直接跳到第四层，其实就是一个捷径，残缺块就是这样构成的啦。

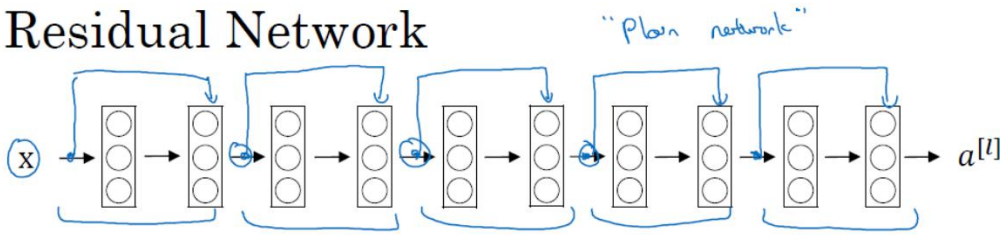


图3.3　5个残差块连接在一起构成一个残差网络

为了突出残差网络的作用，下图简单画出了CNN网络(比较传统的网络结构)和残差网络。在CNN网络中，目标f(x)=x。而为了使网络的输入和输出一样大，即实现恒等映射网络，x等于f(x)，在此残差网络中将上式改变为h(x)=f(x)−x，因为f(x)和x几乎相等，减去后输出h(x)就几乎趋近与0了，所以像残差网络这样的网络，不仅训练起来更加方便和容易，它的变化也更显而易见。也因为它的公式类似残差公式，所以被称为残差网络。

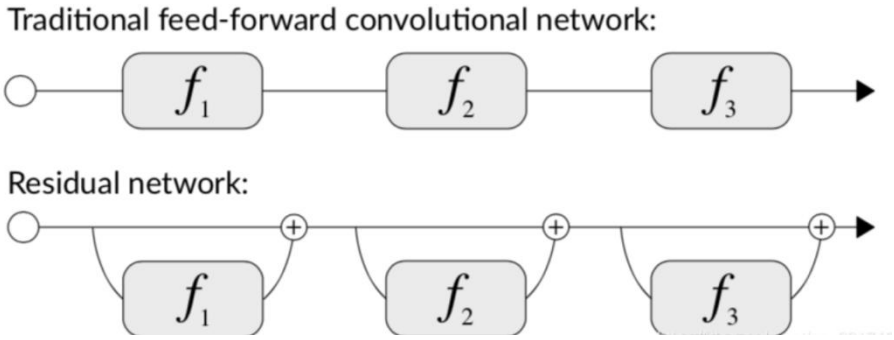


图3.4　CNN网络(上)和残差网络(下)

为什么说残差对训练网络这么重要呢？比如如果我们要训练一个普通网络，应用早期热门的一些算法去直接训练一个网络。根据先人的经验，如果我们训练的网络深度持续加深时，它的错误当然会先减少，但是到达一定程度后还训练深度再加深时，训练的错误会增多。这是在没有设置跳远点没有捷径的时候，也就是没有残差的时候会发生的。这不符合理论知识，正常情况下如果训练的网络深度越深，它的效果应该越来越明显，精度应该越来越高才对。但是事实证明当训练深度到达一定程度后，随着网络深度加深错误也会相继增加。但是有了残差就不一样了，看下图就是随着layers的加深training error的变化。

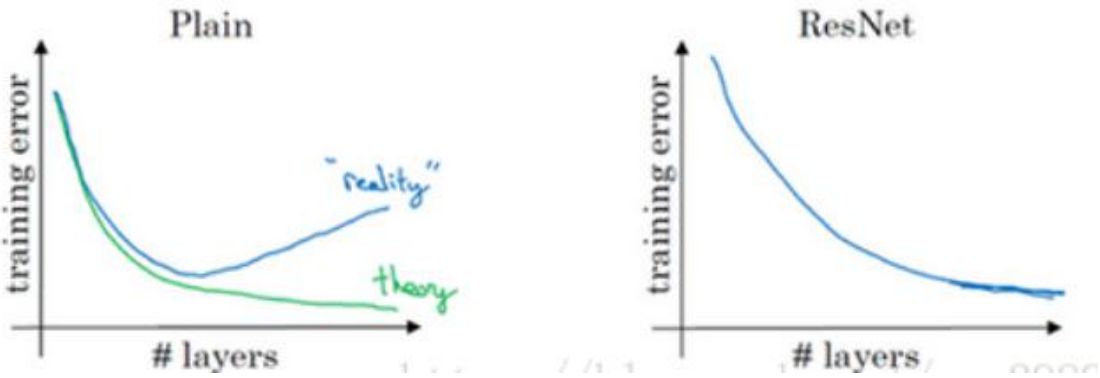


图3. 5　随着layers的加深，训练错误率在Plain和ResNet下的变化

从图中我们就知道不管训练的层数怎么增加，训练的错误量都不会再“反弹”，而是于我们理论相符，随着深度的增加一直减少，不论是层数多深都一样，以此保证良好的性能，同时能够解决各种训练网络的问题，比如梯度爆炸等等。所以 ResNet 十分适合于深度网络。本设计就是应用深度残差网络(ResNet)来实现人脸识别。它的识别速度不仅快，可以达到 10 帧每秒，而且识别精度也相对较高，因此在开发中此方法深受人工智能爱好者的青睐。也就是用 dlib 中已经训练好的 ResNet 模型，调用之后可以得到特征向量，比较向量可以实现识别，在下文中会详细谈到。

3.3 双线性插值法

当图像的大小发生改变时，它的像素会发生改变，比如由2x2图像变化到4x4图像，在这个过程就会产生一部分新的像素点，那么这些新的像素点的像素值要如何确认呢？在数字图像处理我们对这些新像素点求值的过程称为插值，它的算法我们称为图像缩放算法，事实上现已经有一系列的插值方法，而本设计应用的双线性内插值，它是利用原图中，距离改变后坐标最近的四个像素点，通过计算来得到该新像素点的值，也就是它使用4个最近的邻居，取它们的加权平均值来产生输出[8]。因此缩放效果是相对比较好的，算法也比较容易理解。

那么，让我们首先讨论什么是线性插值， 线性插值是用线性多项式估计值的方法。假设我们有两个值为10和20的点，我们想猜测它们之间的值。简单的线性插值如下所示：

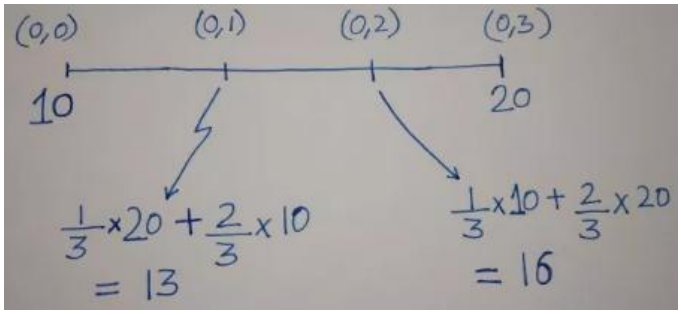


图3.6　计算线性插值的例子

更多的权重放在最近的值(见上图中的1/3和2/3)。对于2D(例如图像)，我们必须执行两次这个操作，一次沿着行，然后沿着列，这就是为什么它被称为双线性插值。

双线性内插值算法描述如下：如果原图为2x2，目标图像大小为4x4，那么原图和目标图像的边长比就为1/2，假设要求目标图像第（i, j）个像素点（i行j列）的像素值，我们只需用式子i*(1/2)和j*(1/2)来求得它在原图下对应的像素点。但是当目标图像大小变为3x3时，其对应的坐标（i*m/a, j*n/b）不为整数，这时候就需要用到双线性内插值，首先找到举例坐标（i*m/a, j*n/b）最近的4个像素点坐标(整数)，然后计算它们的加权平均值来得到该点的值（灰度值或者RGB值）。比如对应坐标是（1.5, 1.5）的话，那么最近的四个像素是（1，1）、（1，2）、（2，1），（2，2）。然后用一下公式进行计算可以得到插值的像素值：f(i, j)=q1*x1+q2*x2+q3*x3+q4*x4；(3-1) 其中，xi(i=1, 2, 3, 4)为最近的四个像素点，qi(i=1, 2, 3, 4)为各点相应权值。关于权值的计算，Matlab和Opencv有不同的算法，得到的权重分布也不同。这里就不做分析。

3.4 双边滤波

3.4.1 概述

在平滑或模糊图像(最流行的平滑目标是降低噪声)中，由于线性滤波器易于实现，且速度快，所以可以使用多种线性滤波器，其中最常用的是均匀滤波器、高斯滤波器、中值滤波器等[1]。

但如果利用上述三种滤波器平滑处理图像，消除噪声的同时，会平滑边缘，这使边缘更不锐化，甚至消失了。为了解决这一问题，我们可以使用一种称为双边滤波器的滤波器，它是高斯滤波器的一个高级版本，它引入了另一个权重，表示两个像素在值上如何接近(或相似)，并且通过考虑图像中的两个权重，双边滤波器可以在模糊图像的同时保持边缘锐利，即保存图像边缘细节而滤除掉低频分量的噪音，但是双边滤波器的效率

不是太高，花费的时间相较于其他滤波器而言也比较长。

假设我们有一个有噪音的原始图像，它的RGB图如下：

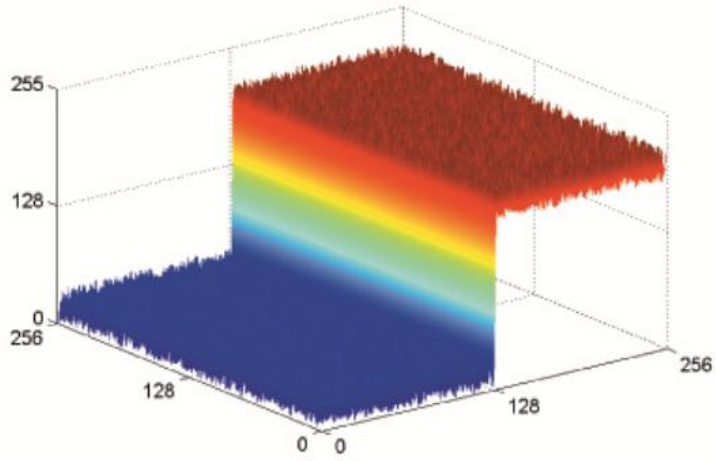
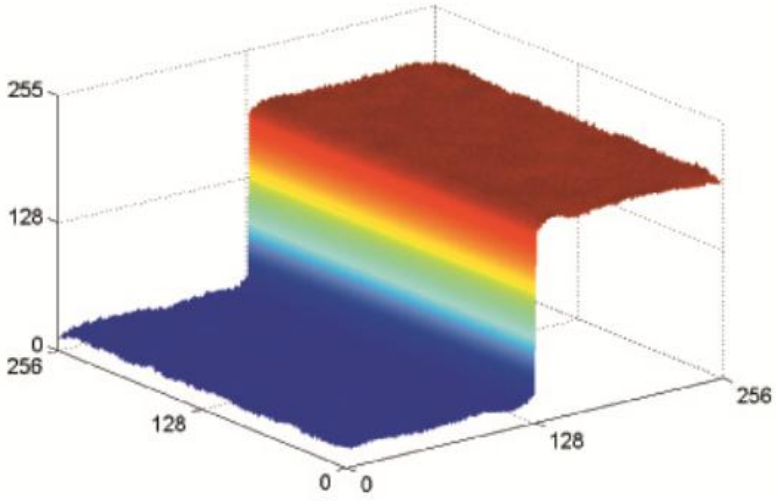


图3.7　有噪音的RGB原始图像



注：使用双边滤波器，图像更平滑，边缘也是尖锐的

图3.8 双边滤波处理后的RGB图像

3.4.2 实现方法

实现双边滤波的方法有很多，在opencv中，它提供了bilateralFilter()函数如下：

void bilateralFilter(src, dst, d, sigmaColor, sigmaSpace, borderType=BORDER_DEFAULT)

（1）src：待处理的原图，是一个矩阵，Mat类型，同时必须是浮点型单或者三通道的图像

（2）dst：处理后的图像，也是一个矩阵，Mat类型，尺寸和原图一样。

（3）d:整型，像素领域的直径范围，一般为正数，否则函数会从第五个参数计算该值。

（4）sigmaColor：double类型，控制颜色空间过滤器，值越大，能更轻易的混合周围的颜色。

（5）sigmaSpace：double类型，控制坐标空间滤波器，值越大，图像中颜色相似的像素将更容易受到影响，受影响的范围更大，取得相同的颜色[3]。

（6）int borderType=BORDER_DEFAULT:开启图像外部像素的边界模式

图3.9 未处理过的图像

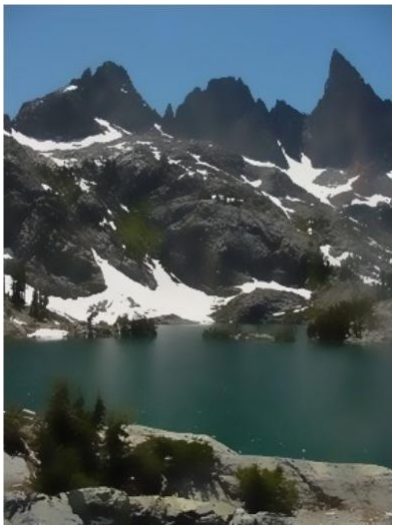


图3,10 双边滤波后的图像

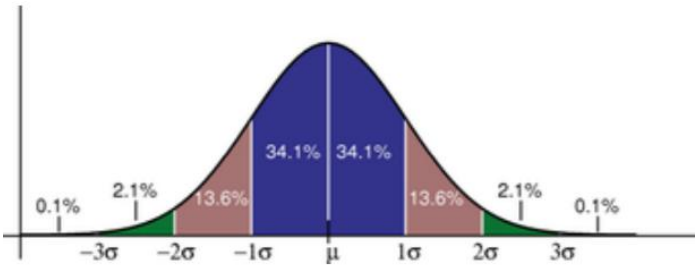
3.5 高斯滤波

3.5.1 概述

图像的高斯滤波，其实就是将二维分布作为点扩散的函数。使用二维分布函数与图像做卷积。我们需要产生一个高斯函数的离散近似值。理论上这需要一个无限大的卷积核，因为高斯分布在任何地方都是非零的。幸运的是，分布在距平均值大约三个标准偏差处接近于零。99%的分布在3个标准偏差内。这意味着我们通常可以将内核大小限制为只包含平均值的3个标准偏差内的值。因为正态分布也叫做高斯分布，所以我们一般把这种方法叫做高斯滤波。高斯滤波器是一类根据高斯函数的形状来选择权值得线性平滑滤波器，对于抑制服从正态分布的噪声十分有效[9]

3.5.2 高斯分布

（1）一维高斯分布函数：



Distribution of the Gaussian function values (Wikipedia)

图 3.11 一维高斯分布函数

$$G(x)=\frac{1}{\sqrt{2\pi\sigma^2}}e^{-\frac{x^2}{2\sigma^2}} \tag{3-2}$$

σ是分布的标准偏差，以图形方式显示，我们看到熟悉的钟形高斯分布，用概率论的术语，它描述了从负值到正值变化时任何给定空间的 100%可能值。高斯函数永远不等于零，并且它是对称函数。特别的，高斯函数的标准差在设计固定高斯核时起着重要的作用。

（2）二维高斯分布：它其实只是两个一维高斯函数的乘积，如果我们要谈到图像领域，就必须使用二维的。

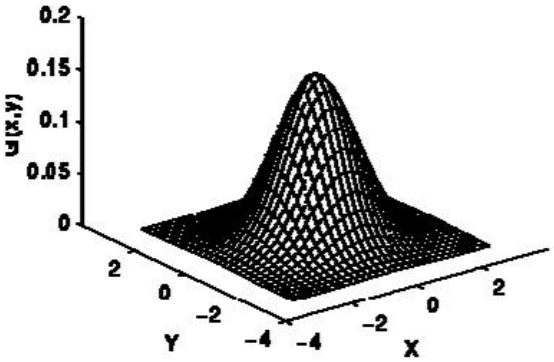


图 3.12　二维高斯分布函数图

$$G(x,y)=\frac{1}{2\pi\sigma^2}e^{-\frac{x^2+y^2}{2\sigma^2}} \tag{3-3}$$

3.5.3 高斯核

如果我们对高斯函数进行 sample 处理，则可以的得到。下面是一个 5 乘 5 的整数卷积核，近似于一个σ为 1 的高斯函数。

$$\frac{1}{273}\begin{bmatrix} 1 & 4 & 7 & 4 & 1 \\ 4 & 16 & 26 & 16 & 4 \\ 7 & 26 & 41 & 26 & 7 \\ 4 & 16 & 26 & 16 & 4 \\ 1 & 4 & 7 & 4 & 1 \end{bmatrix}$$

高斯核随距离中心的增加而减小。中心像素的权重高于外围像素。为了保持滤波器的高斯性质，必须随着σ值的增大而增大核大小。高斯核系数取决于σ值。在掩模边缘，系数必须接近 0。核是旋转对称的，没有方向偏差。高斯核是可分离的，允许快速计算。高斯滤波器可能不会保留图像亮度。

3.5.4 高斯滤波步骤

（1）将高斯核移动到需要被处理的像素上方。

（2）将需要被处理的像素值与高斯核相乘。

（3）将乘积的结果相加，作为结果

具体如何实现呢？本设计采用GaussianBlur()函数：

```
void GaussianBlur( InputArray src, OutputArray dst, Size ksize,
                                   double sigmaX, double sigmaY = 0,
                                   int borderType = BORDER_DEFAULT );
```

参数解释：

1. src：待处理的原图，是一个矩阵，Mat类型，同时必须是浮点型单或者三通道的图像

2. dst：处理后的图像，也是一个矩阵，Mat类型，尺寸和原图一样。

3. ksize：它表示高斯核的大小，必须为正数和奇数或者零，同时宽度和高度可以不同。

4. sigmaX：double类型，在X方向的的标准偏差。

5. sigmaY：double类型，在Y方向的的标准偏差。

6. int borderType=BORDER_DEFAULT:开启图像外部像素的边界模式

例子:

图3. 13 原始图像

图3. 14 高斯滤波处理后的图像

3.6 图片线性混合

线性混合操作是一种典型的二元（两个输入）的像素操作，它的理论公式是这样的：g(x) = (1-α)*f1(x) + α*f2(x) (3-4) 其中，α代表图像的权重。

就像放映幻灯片中的淡入功能或者电影制作中过度情节时出现的画面，前后的画面进行重叠，达到图像混合的效果。根据上面的公式，我们可以通过改变两幅图像的权值α，来分配两幅图像的权重，以产生不一样的混合效果。

那么在编程中如何实现呢？在OpenCV中，我们可以addWeighted()函数，通过这个函数我们可以求得两个图像阵列的加权和。它的原型如下：

```
1 void addWeighted(InputArray src1, double alpha, InputArray src2,
2                   double beta, double gamma, OutputArray dst, int dtype=-1);
```

（1）src1，待处理的原图的图像阵列，是一个数组，Mat类型，同时必须是浮点型单或

者三通道的图像

（2）alpha，第一个数组的权值

（3）src2，待处理的原图的图像阵列，是一个数组，Mat类型，和第一个数组必须拥有相同的尺寸和通道数。

（4）beta，第二个数组的权值。

（5）dst，处理后的图像阵列，是一个数组，Mat类型，和第一个和第二个数组必须拥有相同的尺寸和通道数。

（6）gamma，一个加到权重总和上的标量值。

（7）dtype，默认为-1

AddWeighted()函数公式：`dst = src1[I]*alpha+ src2[I]*beta + gamma;` (3-5) 其中的I是多维数组元素的索引值。

一个例子:

图3. 15　图像A

图3.16　图像B

图3.17　图像A与图像B线性混合后

4 软件功能设计

4.1 人脸识别模块

4.1.1功能概述

人脸识别实际上由三个步骤组成，分别是人脸检测、特征点检测和人脸识别，首先会先检测到人脸然后用绘图工具画出包围人脸区域的矩形窗口，接着提取68个特征点，将特征点信息传给人脸识别的函数。最后进行识别，通过ResNet返回人脸特征向量，依次计算两个特征向量的欧式距离，进行匹配[4]。

4.1.2 人脸检测和特征点检测阶段

人脸是如何进行检测的呢？它其实是在待处理的图像上生成不同大小的区域，如何遍历整个图像，看有没有能检测到人脸，有的话用绘图工具画出包围人脸区域的矩形窗口。早期比较热门的算法有特征提取+集成学习分类器(像是haar特征+adaboost级联分类器)，这种算法的检测效果不是很好，精度比较低，现在已经很少被使用了，实际上opencv就是用此类方法来实现人脸识别的，所以本设计采用的是dlib的HOG（histogram of oriented gradient）的方法，因为涉及到深度学习，所以使用dlib训练好的模型进行检测效果要好很多。

接着就是特征点的提取，使用预测算子获取得到的人脸区域中的五官的几何点区域，加载的是68特征点的landmark模型。

4.1.3 识别阶段

其实就是分类器，将检测到的人脸的特征向量与人脸库中的人脸的特征向量进行比较，再计算相似度，判断识别结果[5]。具体分为两步：

①特征向量抽取。本设计利用的是的，训练好的人脸识别模型，即深度残差网络(ResNet)方法。它的识别速度不仅快，可以达到10帧每秒，而且识别精度也相对较高，因此在开发中此方法深受人工智能爱好者的青睐。此接口会返回一个128D的人脸特征向量。

②返回欧式距离进行匹配。得到特征向量之后，我们可以计算欧式距离，计算得到相似度和匹配结果。

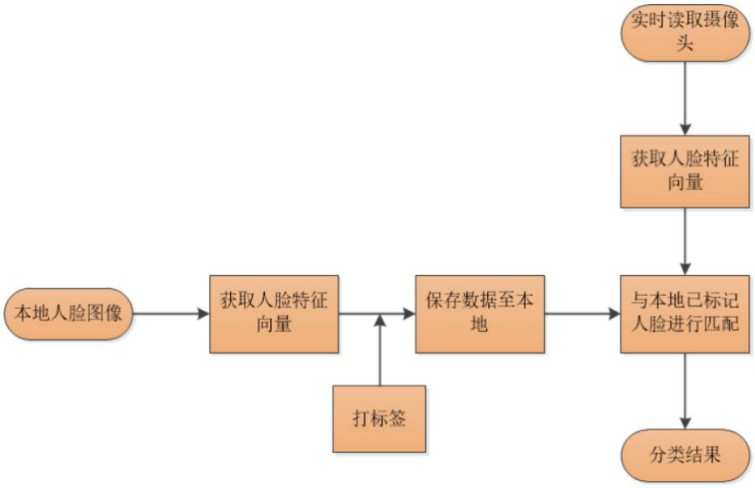


图4. 1人脸识别分类流程图

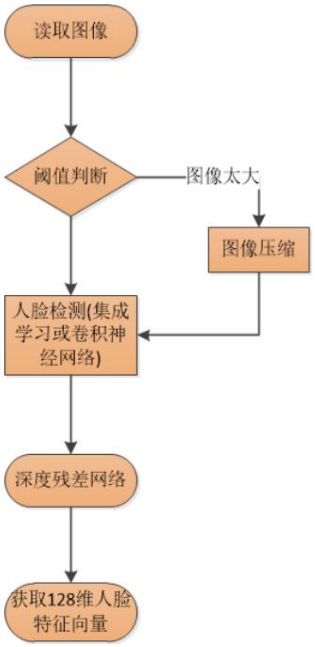


图4. 2获取人脸特征向量流程图

4. 1. 4具体实现方法

（1）首先加载需要的python库：

```
import  sys
import  os
import  cv2
import  numpy  as  np
import  dlib
import  math
```

然后加载模型参数：

```
detector = dlib.get_frontal_face_detector()
# 加载正脸检测器，返回值是一个矩形，坐标为[(x1, y1) (x2, y2)],
predictor = dlib.shape_predictor("face/shape_predictor_68_face_landmarks.dat")
#加载人脸关键点检测器(已经训练好的人脸关键点检测器)
facerec =
dlib.face_recognition_model_v1("face/dlib_face_recognition_resnet_model_v1.dat")
```

这里使用HOG特征级联分类的检测方法，可检测到图像中人脸的位置，效果略差于CNN。变量predictor，使用预测算子获取得到的人脸区域中的五官的几何点区域，这里加载的是68特征点的landmark模型；然后通过facerec得到ResNet模型，这里训练模型已经给出，因此不需要自己手动去训练了。

最后，每次检测到人脸，开启识别按钮，若进行识别，则会先对选择的图片进行处理，使用ResNet的接口获取人脸特性向量，保存到事先准备好的矩阵中。之后再对facelib目录下的所有图片进行处理，facelib目录下所有图像的命名方式均为姓名.jpg。首先读取目录下的每一张图片，然后检测图片中的人脸(facelib中默认一张图只有一张人脸)，接着同样使用ResNet的接口获取人脸特性向量，保存到矩阵中。然后比较两个图片“128向量”的欧式距离并打印出来结果，结果小于阈值则时，判定为同一个人并弹出提示框显示识别结果，否则显示无识别结果。

4.1.5 具体实现代码

```
 #识别  打开该目录下的人脸文件逐个比较，如果在设定的阈值范围内则弹窗提示
def _re_face(self):

    try:
        global a
        l=self.return_face_number(all_mat)
        print("There %s person in this pisture" %l)
        while a<l:
            now_vector = self.get_vector_origin(all_mat) #返回所选图片的特征向量

            flg = 0

            for root, dirs, files in os.walk('faceLib'):
                for file in files:
                    if os.path.splitext(file)[-1] == '.png'or os.path.splitext(file)[-1] == '.jpg':

                        #识别库为 faceLib 文件夹 里面为储存的人脸照片，识别时会与faceLib内的照片对比，格式为 名字+.png/jpg

                        print (os.path.splitext(file)[0])

                        name_mat = cv2.imdecode(np.fromfile("faceLib/"+file,dtype=np.uint8),cv2.IMREAD_COLOR)

                        # 先用np.fromfile()读取为np.uint8格式，再使用cv2.imdecode()解码,返回img,可读取中文路径

                        #name_mat = cv2.imread("faceLib/"+file)  中文路径报错

                        name_vector = self.get_vector(name_mat)

                        compare = self.return_euclidean_distance(name_vector, now_vector)

                        print("compare_result:" + str(compare))

                        if(compare < 0.4):
                            QMessageBox.about(None,'人脸识别结果:',"\n         识别到"+os.path.splitext(file)[0]+"           \n " )

                            flg = 1
                if flg == 0 :

                    QMessageBox.about(None,"人脸识别结果:","\n         Person%s没有对应结果              \n " %(a+1))

            a=a+1

        if a==l:
            a=0


    except Exception as e:
       print(e)
```

```
#返回人脸数
def return_face_number(self,frame):

    tem = detector(frame, 1)

    return len(tem)


#利用dlib_face_recognition_resnet_model_v1.dat得到原图人脸128向量特征
def get_vector_origin(self,frame):
    global a

    tem = detector(frame, 1)  #人脸检测

    face_vector = 0

    if (len( tem)>0):

        shape = predictor(frame, tem[a])

        face_vector = facerec.compute_face_descriptor(frame, shape)

    return face_vector

#利用dlib_face_recognition_resnet_model_v1.dat得到人脸库中图片人脸128向量特征

def get_vector(self,frame):

    tem = detector(frame, 1)  #人脸检测

    face_vector = 0

    if (len( tem)>0):

        shape = predictor(frame, tem[0])

        face_vector = facerec.compute_face_descriptor(frame, shape)

    return face_vector
```

```
# 比较两个“128向量”的欧式距离
def return_euclidean_distance(self,face_vector_0,face_vector_1):
    face_vector_0 = np.array(face_vector_0)
    face_vector_1 = np.array(face_vector_1)
    tem = np.sqrt(np.sum(np.square(face_vector_0 - face_vector_1))) #相减后求平方再相加再开方，欧式距离
    #print(tem)
    return tem
```

代码解析：

（1）def _re_face(self)：

全局变量l为所选图片的人脸数，由函数return_face_number(self, frame)返回。全局变量a初始值为0，为了产生循环，并且在get_vector_origin(self, frame) 函数中作为tem[a]的变量，对所选图片中每张人脸都进行检测并返回特征向量now_vector。

接着遍历facelib文件夹下中的每张人脸(默认每张图上只有单张人脸，并且是jpg或者png格式的图片)。对每张图片都做一下处理：先用numpy. fromfile()读取为uint8格式，然后使用cv2. imdecode()解码, 返回图像矩阵name_mat（可读取中文路径），再调用get_vector(self, frame)函数获得128特征向量name_vector，最后调用return_euclidean_distance(self, face_vector_0, face_vector_1)函数比较两个“128向量”，计算出欧式距离，如果欧氏距离小于阈值0. 4则判定为同一个人，弹出提示框显示识别结果，否则显示对选择图像中的第几个人没有识别结果。

（2）def get_vector(self, frame)：

人脸检测tem = detector(frame, 1)，关键点检测shape = predictor(frame, tem[0])首先通过dlib. shape_predictor(predictor_path)从路径中加载模型，返回的predictor就是特征提取器，对dets遍历，用predictor(img, d)计算检测到的每张人脸的关键点；获取每个关键点坐标shape. parts()的x, y值，存入landmark矩阵（模型默认提取68个关键点，所以landmark为68×2矩阵）。描述子提取，128D向量face_vector = facerec. compute_face_descriptor(frame, shape)

（3）def return_euclidean_distance(self, face_vector_0, face_vector_1)：

欧式距离的计算方式：矩阵相减后求平方再相加再开方，欧式距离。

4.2 人脸美颜模块

4.2.1 功能概述

美颜功能实际上就是先检测到人脸，再对人脸进行磨皮处理。具体的算法实现我参考了博客上一篇文为《简单探讨可牛影像软件中具有肤质保留功能的磨皮算法及其实现细节》的博文，文章中涉及的是ps技术，介绍了如何运用PS配合外挂磨皮滤镜(插件Portraiture)对人物皮肤进行磨皮美化。虽然非python代码，但他的磨皮算法缺十分具有参考价值，实现的效果也是十分可观的[12]。

4.2.2 具体实现方法

（1）假定原始图像层为Src层，首先遍历图像中所有检测到的人脸，设置roi(Region Of Interest)区域，也就是从图像中划出我们所关注的区域，将大图像变为小图形，再对小图形进行处理，节省了时间同时，简化了图像处理过程。

（2）之后对HighPass层磨皮：这个算法可以选择：表面模糊、导向滤波、双边滤波、各向异性扩散、BEEP、局部均方差、Domain transfer、 Adaptive Manifolds、 Local Laplacian Filters等任何具有保边效果的EPF-Filter，即边缘保留滤波。

表达式为： HighPass = EPF-Filter(HighPass) (4-1)；

本设计采用的是双边滤波，应用opencv自带的cv2. bilateralFilter(src, dst, d, sigmaColor, sigmaSpace, borderType=BORDER_DEFAULT)方法。双边滤波的功能与具体实现方法在上文中有提到过这里就不详细说明。

（3）然后再用高反差保留的算法的计算公式：HihgPass = HighPass - Src + 128 (4-2)；通过高反差保留算法可以保留图片的细节信息，特别是对图像中反差比较大的区域，比如人脸中的五官[11]。

(4) 接着进行高斯滤波，它的功能与具体实现方法在上文中也有提到这里也不过多阐述。表达式为：HighPass = GuassBlur(HighPass, Radius)；其中Radius为高斯模糊的半径，本设计应用OpenCV中的函数GaussianBlur实现了高斯滤波。

(5) 进行图层混合： 假定两个需要混合的图像X和Y，X在下方，Y在上方，X与Y混合，其计算公式为：Z = X + 2 * Y - 255 (4-3)； 再运用了OpenCV中addWeighted函数计算两个数组（图像阵列）的加权和，得到结果后替换原图的人脸区域。

4.2.3 具体实现代码

```
#检测美颜复选框
if (self.checkBox.isChecked()):

    for k, d in enumerate(face_number):

        #设置roi区域，即取出人脸区域
        face=frame[d.top():d.bottom(),d.left():d.right()]

        #双边滤波:对HighPass层磨皮 //or use EPF-filter :HighPass = EPF-Filter(HighPass);
        temp1 = cv2.bilateralFilter(face,15,37.5,37.5)

        #计算公式: HihgPass = HighPass - Src + 128;
        temp2 = (temp1 - face + 128);

        #高斯滤波
        temp3 = cv2.GaussianBlur(temp2 ,(1,1),0,0 )

        temp4 = face + 2 * temp3 - 255;

        #图片线性混合:
        face = cv2.addWeighted(face, 0.5, temp4, 0.5, 0) # 50%

        #人脸区域替换原图人脸区域
        frame[d.top():d.bottom(),d.left():d.right()]=face
```

4.3 人脸计数模块

4.3.1 概述和实现方法

人脸计数的前提是能检测到人脸，返回矩阵后，求矩阵或者说数组的长度len()就是人脸的个数了。最后只需要用opencv的putText()将人脸个数显示在左上角或者以其他方式显示出来。

4.3.2 具体实现代码

```
#检测人脸统计复选框
if (self.checkBox_2.isChecked()):
    #  len(face_number)即为人脸个数
    cv2.putText(frame, "face number:"+str(len(face_number)), (20, 50), cv2.FONT_HERSHEY_SIMPLEX, 2.0, (0, 255, 0), 5, cv2.LINE_AA)
```

4.4 瘦脸功能模块

4.4.1功能概述

当我们用dlib的方法检测到人脸，并提取68个特征点后，就可以进入瘦脸的程序了。本设计中应用的瘦脸算法主要是图像局部平移变形。

图4.3人脸的68个特征点

4.4.2 具体实现方法

图像要如何变形呢？根据收集的资料，图像变形基本都是变形前的坐标根据相关关系计算得到变形后的坐标，从而形成不懂得变形比如旋转、缩小增大、平移等等。而为了保证变形后的图像是连续、完整的。在实际中我们是先确定变形后坐标，在进行逆变换的到变形前坐标，之后进行插值，得到的rgb像素值作为变形后的像素值。

下面简单讲一下图像局部平移变形，该变形稍微复杂一些，公式如下：

$$\vec{u} = \vec{x} - \left(\frac{{r_{\max}}^2 - |\vec{x} - \vec{c}|^2}{({r_{\max}}^2 - |\vec{x} - \vec{c}|^2) + |\vec{m} - \vec{c}|^2} \right)^2 (\vec{m} - \vec{c}) \tag{4-4}$$

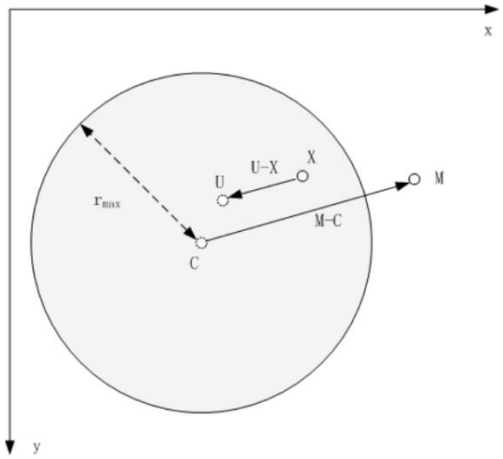


图4.4 公式图示

公式中，由于主要是像素点位置计算，因此涉及一些矢量运算，不过比较简单。其实上面公式就是逆变换公式了，x是变换后的位置，u是原坐标位置。整个计算在以c为圆心，r为半径的圆内进行。本设计就是应用第4个点到第6个点的距离作为左脸瘦脸距离，即CM，第14个点到第16个点的距离作为右脸瘦脸距离，先确定变形后坐标，在进行逆变换的到变形前坐标，之后进行插值，得到的rgb像素值作为变形后的像素值[13]。

4.4.3 具体实现代码

```
#检测瘦脸复选框

if (self.checkBox_3.isChecked()):
    try:
        land_marks = []
        for i in range(len(face_number)):
            land_marks_node = np.matrix([[p.x,p.y] for p in predictor(frame,face_number[i]).parts()])
            land_marks.append(land_marks_node)  #68个特征点标定功能
        if len(land_marks) == 0:
            return
        for landmarks_node in land_marks:
            left_landmark= landmarks_node[3]
            left_landmark_down=landmarks_node[5]

            right_landmark = landmarks_node[13]
            right_landmark_down = landmarks_node[15]

            endPt = landmarks_node[30]

            #计算第4个点到第6个点的距离作为瘦脸距离
            r_left=math.sqrt((left_landmark[0,0]-left_landmark_down[0,0])*(left_landmark[0,0]-left_landmark_down[0,0])+
                        (left_landmark[0,1] - left_landmark_down[0,1]) * (left_landmark[0,1] - left_landmark_down[0, 1]))

            # 计算第14个点到第16个点的距离作为瘦脸距离
            r_right=math.sqrt((right_landmark[0,0]-right_landmark_down[0,0])*(right_landmark[0,0]-right_landmark_down[0,0])+
                        (right_landmark[0,1] -right_landmark_down[0,1]) * (right_landmark[0,1] -right_landmark_down[0, 1]))

            #瘦左边脸
            frame = self.localTranslationWarp(frame,left_landmark[0,0],left_landmark[0,1],endPt[0,0],endPt[0,1],r_left)
            #瘦右边脸
            frame = self.localTranslationWarp(frame, right_landmark[0,0], right_landmark[0,1], endPt[0,0],endPt[0,1], r_right)
    except Exception as e:
        print(e)
```

代码解析：先创建一个空列表，将检测到的68个特征点进行标记后，按[x，y]形式存在列表中。之后计算第4个点到第6个点的距离作为左脸的瘦脸距离，也就是上面局部平移算法提到的半径r。同样也计算第14个点到第16个点的距离作为右脸的瘦脸距离。然后设中心点为特征点30，即鼻子的中部，上图的C点。再调用局部平移函数。

（1）局部平移函数：

```
#局部平移算法

  def localTranslationWarp(self,srcImg,startX,startY,endX,endY,radius):

      ddradius = float(radius * radius)
      copyImg = np.zeros(srcImg.shape, np.uint8)
      copyImg = srcImg.copy()

      # 计算公式中的|m-c|^2
      ddmc = (endX - startX) * (endX - startX) + (endY - startY) * (endY - startY)
      H, W, C = srcImg.shape
      for i in range(W):
          for j in range(H):
          #计算该点是否在形变圆的范围之内
           #优化，第一步，直接判断是否在（startX,startY)的矩阵框中
              if math.fabs(i-startX)>radius and math.fabs(j-startY)>radius:
                  continue

              distance = ( i - startX ) * ( i - startX) + ( j - startY ) * ( j - startY )

              if(distance < ddradius):
                  #计算出（i,j）坐标的原坐标
                  #计算公式中右边平方号里的部分
                  ratio=(  ddradius-distance ) / ( ddradius - distance + ddmc)
                  ratio = ratio * ratio

                  #映射原位置
                  UX = i - ratio  * ( endX - startX )
                  UY = j - ratio  * ( endY - startY )

                  #根据双线性插值法得到UX，UY的值
                  value = self.BilinearInsert(srcImg,UX,UY)
                  #改变当前 i ，j的值
                  copyImg[j,i] =value

      return copyImg
```

代码解析：获得半径和圆心之后，按若人脸每个像素点特征点4的距离是小于半径，若小于半径，则利用公式计算出原坐标，最后再调用双线性插值函数得到最终的rgb像素值。实际上就是应用此公式：

$$\vec{u} = \vec{x} - \left(\frac{{r_{\max}}^2 - |\vec{x} - \vec{c}|^2}{({r_{\max}}^2 - |\vec{x} - \vec{c}|^2) + |\vec{m} - \vec{c}|^2} \right)^2 (\vec{m} - \vec{c}) \quad (4\text{-}5)$$

（2）双线性插值法：

```
#双线性插值法

def BilinearInsert(self,src,ux,uy):
    w,h,c = src.shape
    if c == 3:
        x1=int(ux)
        x2=x1+1
        y1=int(uy)
        y2=y1+1

        part1=src[y1,x1].astype(np.float)*(float(x2)-ux)*(float(y2)-uy)
        part2=src[y1,x2].astype(np.float)*(ux-float(x1))*(float(y2)-uy)
        part3=src[y2,x1].astype(np.float) * (float(x2) - ux)*(uy-float(y1))
        part4 = src[y2,x2].astype(np.float) * (ux-float(x1)) * (uy - float(y1))

        insertValue=part1+part2+part3+part4

        return insertValue.astype(np.int8)
```

第三章对双线性插值的功能和实现方法有解析，这里不做更深的研究。

4.5 UI界面设计模块

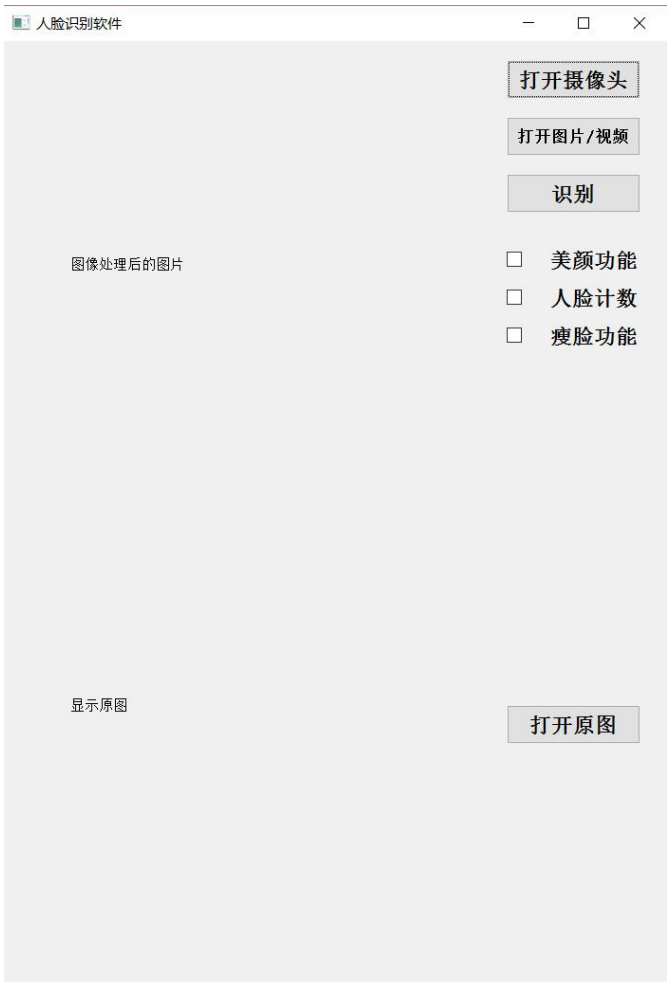


图4.5　软件界面

4.5.1 界面概述

界面由三个Button和三个checkbox组成，只有打开图片或摄像头后，系统检测到人脸时，识别按钮才可以使用。如果要实现拓展的三个功能，即美颜功能、人脸计数、或者瘦脸功能，则要先选着checkbox对应的功能，再打开图片或者摄像头。同时可以在界面下面打开原图，与处理后的图像进行对比。该界面应用PyQt5库来写，使用起来也比较容易

4.5.2 具体实现代码

```
from PyQt5 import QtCore, QtGui, QtWidgets
import sys

class Ui_Dialog(object):
    def setupUi(self, Dialog):
        Dialog.setObjectName("Dialog")
        Dialog.resize(700, 1000)

        self.pushButton_open = QtWidgets.QPushButton(Dialog)
        self.pushButton_open.setGeometry(QtCore.QRect(530, 20, 141, 41))
        font = QtGui.QFont()
        font.setPointSize(13)    #字体的点大小
        font.setBold(True)       #粗体
        font.setWeight(75)       #粗度
        self.pushButton_open.setFont(font)
        self.pushButton_open.setObjectName("pushButton_open")

        self.label = QtWidgets.QLabel(Dialog)
        self.label.setFixedSize(430, 430)
        self.label.move(70, 20)
        self.label.setText("图像处理后的图片")
        self.label.setObjectName("label")

        self.label1 = QtWidgets.QLabel(Dialog)
        self.label1.setText("显示原图")
        self.label1.setFixedSize(430, 430)
        self.label1.move(70, 485)
        self.label1.setObjectName("label1")

        self.pushButton_open_2 = QtWidgets.QPushButton(Dialog)
        self.pushButton_open_2.setGeometry(QtCore.QRect(530, 80, 141, 41))
        font = QtGui.QFont()
        font.setPointSize(10)
        font.setBold(True)
        font.setWeight(75)
        self.pushButton_open_2.setFont(font)
        self.pushButton_open_2.setObjectName("pushButton_open_2")

        self.checkBox = QtWidgets.QCheckBox(Dialog)
        self.checkBox.setGeometry(QtCore.QRect(530, 210, 160, 41))
        font = QtGui.QFont()
        font.setPointSize(13)
        font.setBold(True)
        font.setWeight(75)
        self.checkBox.setFont(font)
        self.checkBox.setObjectName("checkBox")

        self.pushButton_re = QtWidgets.QPushButton(Dialog)
        self.pushButton_re.setGeometry(QtCore.QRect(530, 140, 141, 41))
        font = QtGui.QFont()
        font.setPointSize(13)
        font.setBold(True)
        font.setWeight(75)
        self.pushButton_re.setFont(font)
        self.pushButton_re.setObjectName("pushButton_re")
```

```
        self.checkBox_2 = QtWidgets.QCheckBox(Dialog)
        self.checkBox_2.setGeometry(QtCore.QRect(530, 250, 160, 41))
        font = QtGui.QFont()
        font.setPointSize(13)
        font.setBold(True)
        font.setWeight(75)
        self.checkBox_2.setFont(font)
        self.checkBox_2.setObjectName("checkBox_2")

        self.checkBox_3 = QtWidgets.QCheckBox(Dialog)
        self.checkBox_3.setGeometry(QtCore.QRect(530, 290, 160, 41))
        font = QtGui.QFont()
        font.setPointSize(13)
        font.setBold(True)
        font.setWeight(75)
        self.checkBox_3.setFont(font)
        self.checkBox_3.setObjectName("checkBox_3")

        self.pushButton_open_3 = QtWidgets.QPushButton(Dialog)
        self.pushButton_open_3.setGeometry(QtCore.QRect(530, 700, 141, 41))
        font = QtGui.QFont()
        font.setPointSize(13)
        font.setBold(True)
        font.setWeight(75)
        self.pushButton_open_3.setFont(font)
        self.pushButton_open_3.setObjectName("pushButton_open_3")


        self.retranslateUi(Dialog)
        QtCore.QMetaObject.connectSlotsByName(Dialog)

    def retranslateUi(self, Dialog):
        _translate = QtCore.QCoreApplication.translate
        Dialog.setWindowTitle(_translate("Dialog", "人脸识别软件"))
        self.pushButton_open.setText(_translate("Dialog", "打开摄像头"))
        self.pushButton_open_2.setText(_translate("Dialog", "打开图片/视频"))
        self.checkBox.setText(_translate("Dialog", "  美颜功能"))
        self.pushButton_re.setText(_translate("Dialog", "识别"))
        self.checkBox_2.setText(_translate("Dialog", "  人脸计数"))
        self.checkBox_3.setText(_translate("Dialog", "  瘦脸功能"))
        self.pushButton_open_3.setText(_translate("Dialog", "打开原图"))


if __name__ == "__main__":
    app = QtWidgets.QApplication(sys.argv)
    MainWindow = QtWidgets.QMainWindow()
    ui = Ui_Dialog()
    ui.setupUi(MainWindow)
    MainWindow.show()
    sys.exit(app.exec_())
```

5 软件测试

软件测试是一个相当重要的环节，我们需要通过各种不同的角度和情况来验证该软件是否能达到我们预期的功能，同时在测试中发现BUG并及时修改完善，这是每个编程人员需要掌握的重要技能。越是对软件性能进行更深入的评估。

5.1测试环境

（1）硬件环境：Intel(R) Core(TM) i5-52000U CPU @ 2.20GHz 2.19 GHz

（2）操作系统：WINDOWS 10 PROFESSIONAL

（3）开发工具：Anaconda

（4）开发语言：python、C++

5.2 人脸识别功能模块测试

将Facelib文件夹作为人脸库，为了节省时间这里只采用3张人脸来测试：

图5.1 facelib文件夹下

5.2.1 单人识别

点击打开图片按钮，选择jpg或者png类型的图片，然后点击识别按钮进行识别。识别的同时打印出欧式距离，值越小代表越相似。小于0.47时判断为同一个人，即识别成功。测试结果如下：

```
========== RESTART: D:\pyqt-face\main.py
open file ...
There 1 person in this pisture
Emma Watson
compare_result:0.4108630695439857
JIREH
compare_result:0.8461870954163891
吴彦祖
compare_result:0.8954057363388737
```

图5.2 识别功能测试结果

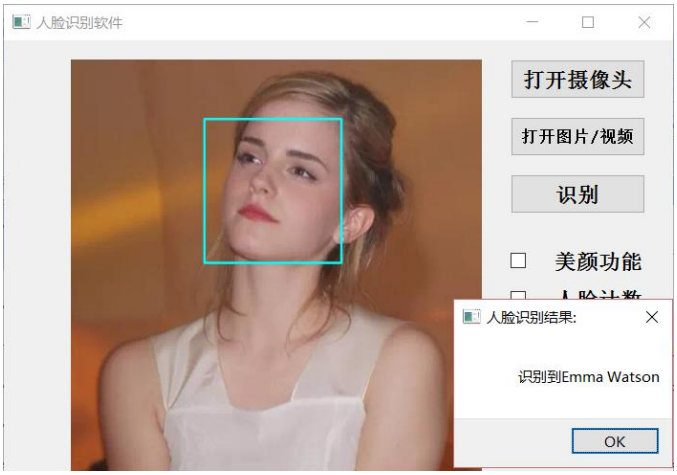


图5.3 识别功能测试结果

更换一张图片进行测试：

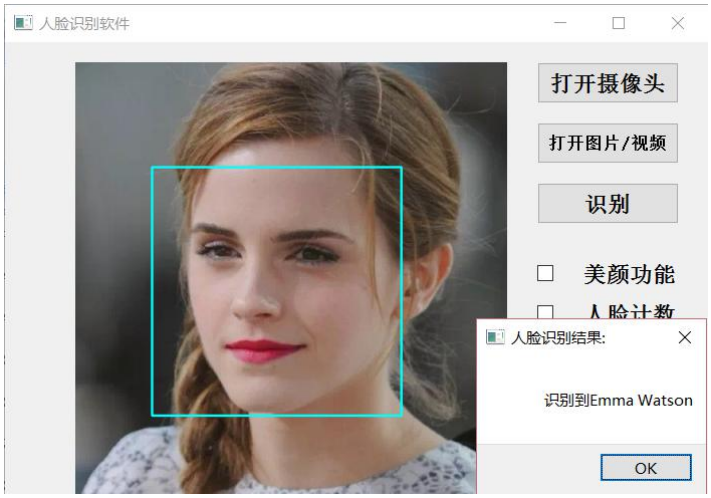


图5.4 识别功能测试结果

5.2.2 多人识别

（1）提取图片中的两个人脸后，分别与人脸库中的每个人脸进行比较，同时打印出比较结果。若识别成功则弹出对话框提示识别到xxx，否则提示几号人物没有识别结果。测试结果如下：人物1没有识别成功，人物2识别成功

```
open file ...
There 2 person in this pisture
Emma Watson
compare_result:0.9310827034024279
JIREH
compare_result:0.4904024774309338
吴彦祖
compare_result:0.6350505161803287
Emma Watson
compare_result:0.8757455530011197
JIREH
compare_result:0.35268607562086735
吴彦祖
compare_result:0.6514978314045274
```

图5.5 多人识别测试结果

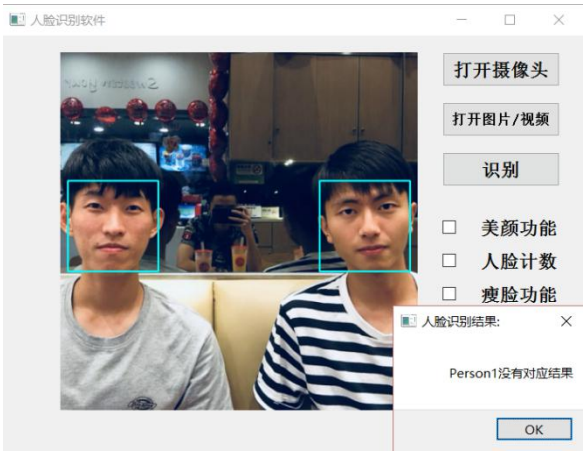


图5.6 多人识别测试结果

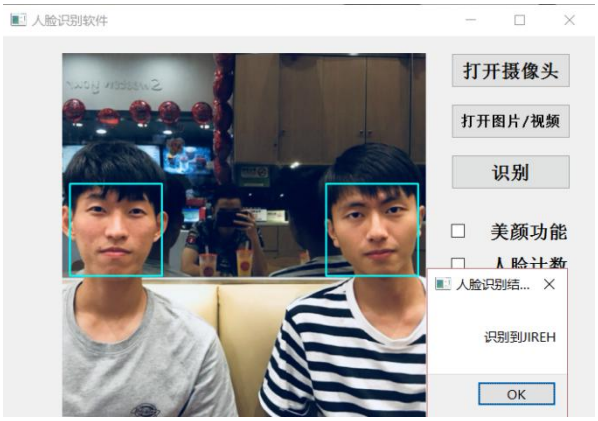


图5.7 多人识别测试结果

5.2.3 前置摄像头识别

打开前置摄像头后，如果检测到人脸，识别功能将被打开，也就是识别按钮会亮起，然后可点击识别进行识别，识别结果也会以提示框的形式显示。测试结果如下：

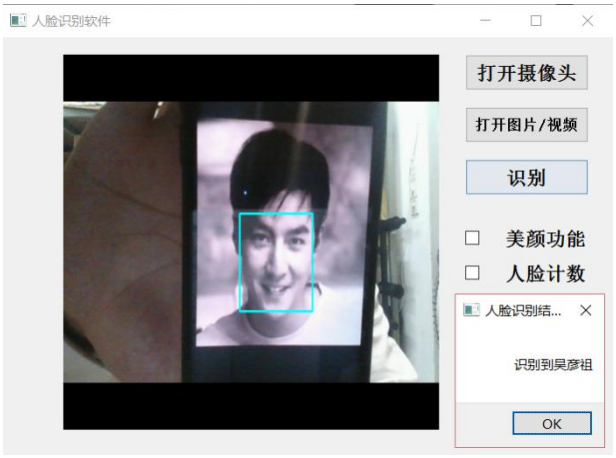


图5.7 前置摄像头识别测试结果

5.3 美颜功能模块测试

5.3.1 图像人脸美颜测试

选择Checkbox的美颜功能选项后，选择图片，图片也必须是jpg或者png格式的，打开后会先对图像中的人脸进行检测，然后自动对人脸进行美颜功能，也就是磨皮、去痘、去除雀斑等功能。为了进行对比，可以在下面打开原图。测试结果如下：

图5.8 美颜功能测试结果

5.3.2 前置摄像头美颜测试

打开前置摄像头后，点击Checkbox的美颜功能选项，只要检测到人脸，就可以实现磨皮功能，去除痘痘，雀斑，皱纹等等。同样在下方打开原图作为对比。测试结果如下：

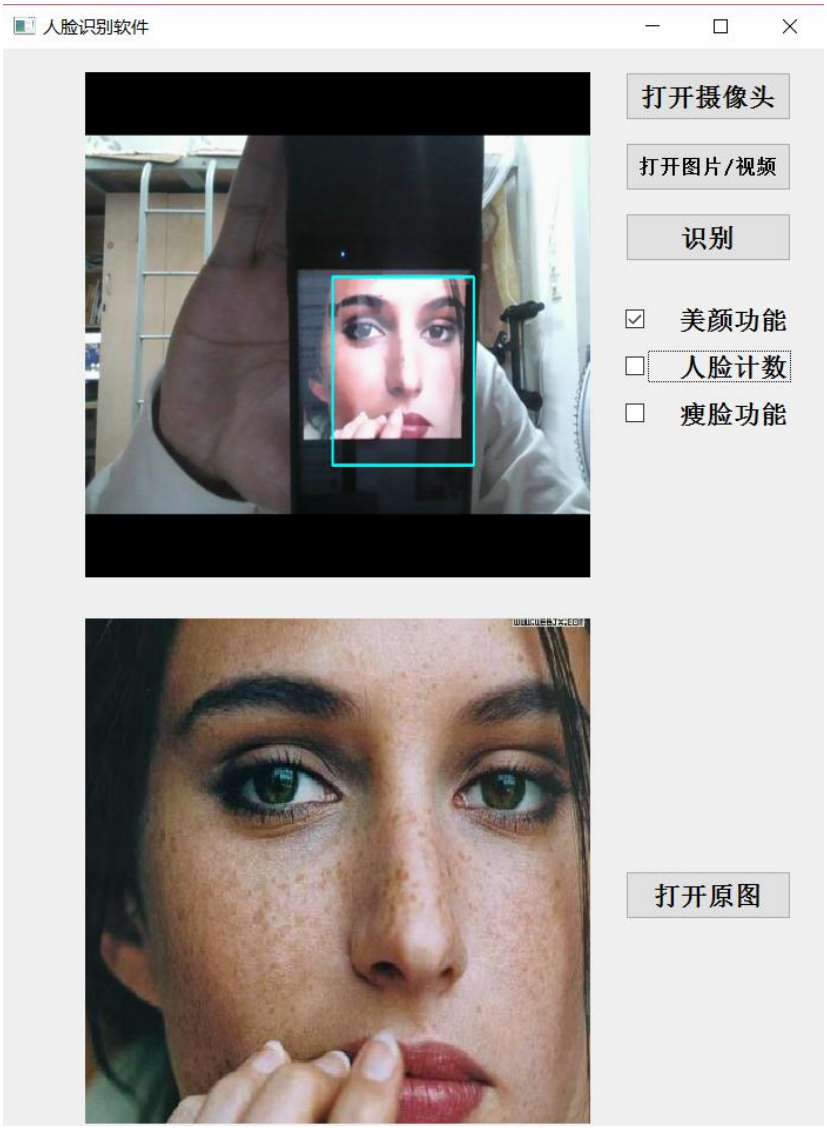


图5.8 前置摄像头美颜功能测试结果

5.4 人脸计数功能模块测试

5.4.1图片人脸计数

选择Checkbox的人脸计数选项后，选择图片，图片也必须是jpg或者png格式的，打开后会自动对图片上的人脸进行检测，检测后返回人脸数，用绿色粗体字显示在图片左上角。测试结果如下：

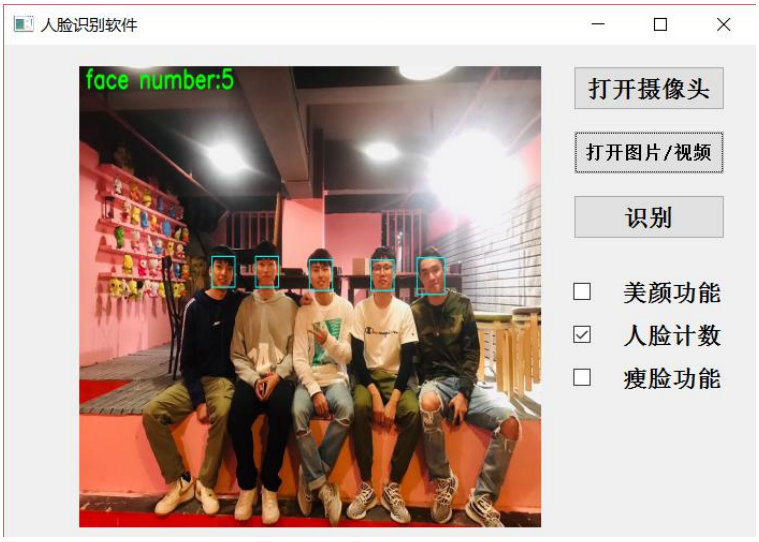


图5.9　人脸计数功能测试结果

5.4.2 前置摄像头人脸计数

打开前置摄像头后，点击Checkbox的人脸计数选项，马上就可以实现计数功能，同样在屏幕左上角显示人脸数。测试结果如下：

图5.10　前置摄像头人脸计数功能测试结果

5.5 瘦脸功能模块测试

5.5.1 图片瘦脸功能

选择Checkbox的瘦脸功能选项后，选择图片，图片也必须是jpg或者png格式的，打开后会自动对图片中的人脸进行瘦脸，也就是局部平移。为了进行对比，可以在下面打开原图。测试结果如下：

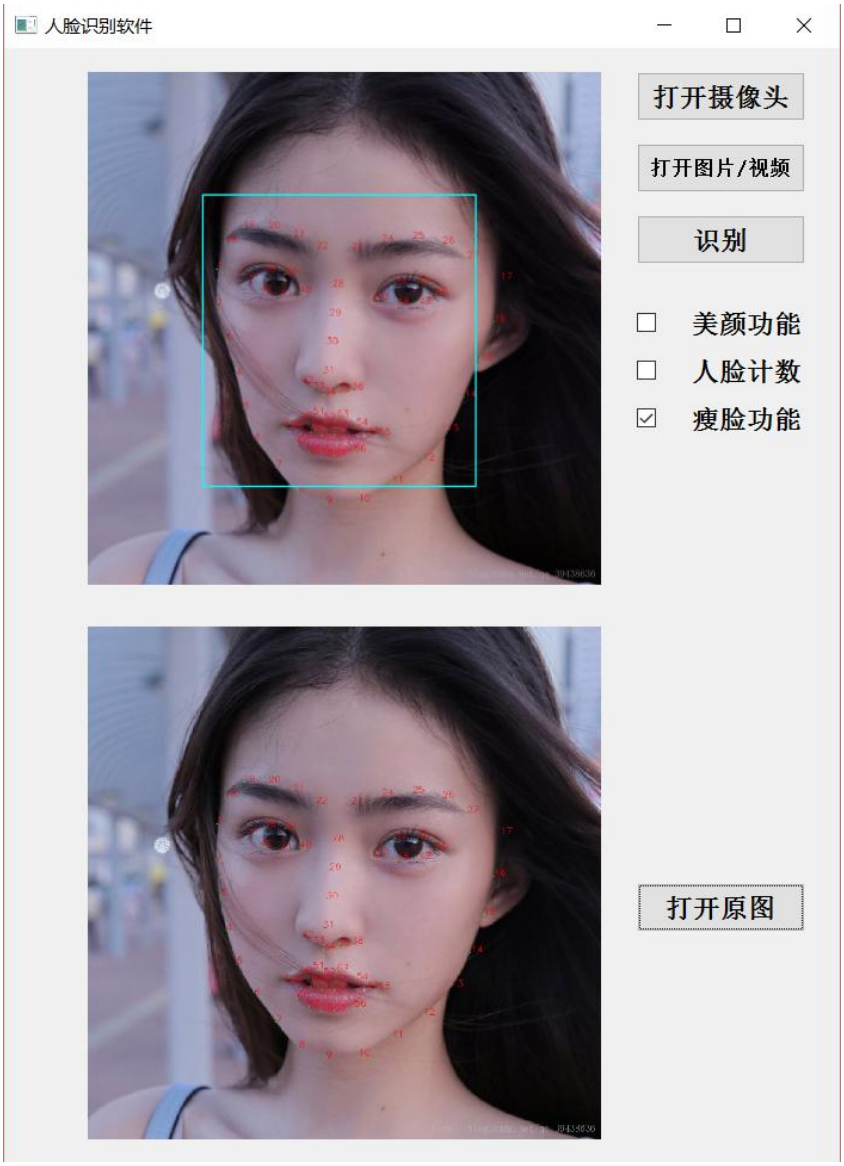


图5.11　瘦脸功能测试结果

5.5.2 前置摄像头瘦脸功能模块测试

打开前置摄像头后，点击Checkbox的瘦脸功能选项，只要检测到人脸，就可以实现瘦脸功能，同样为了进行对比，我在下方打开了原图测试结果如下：

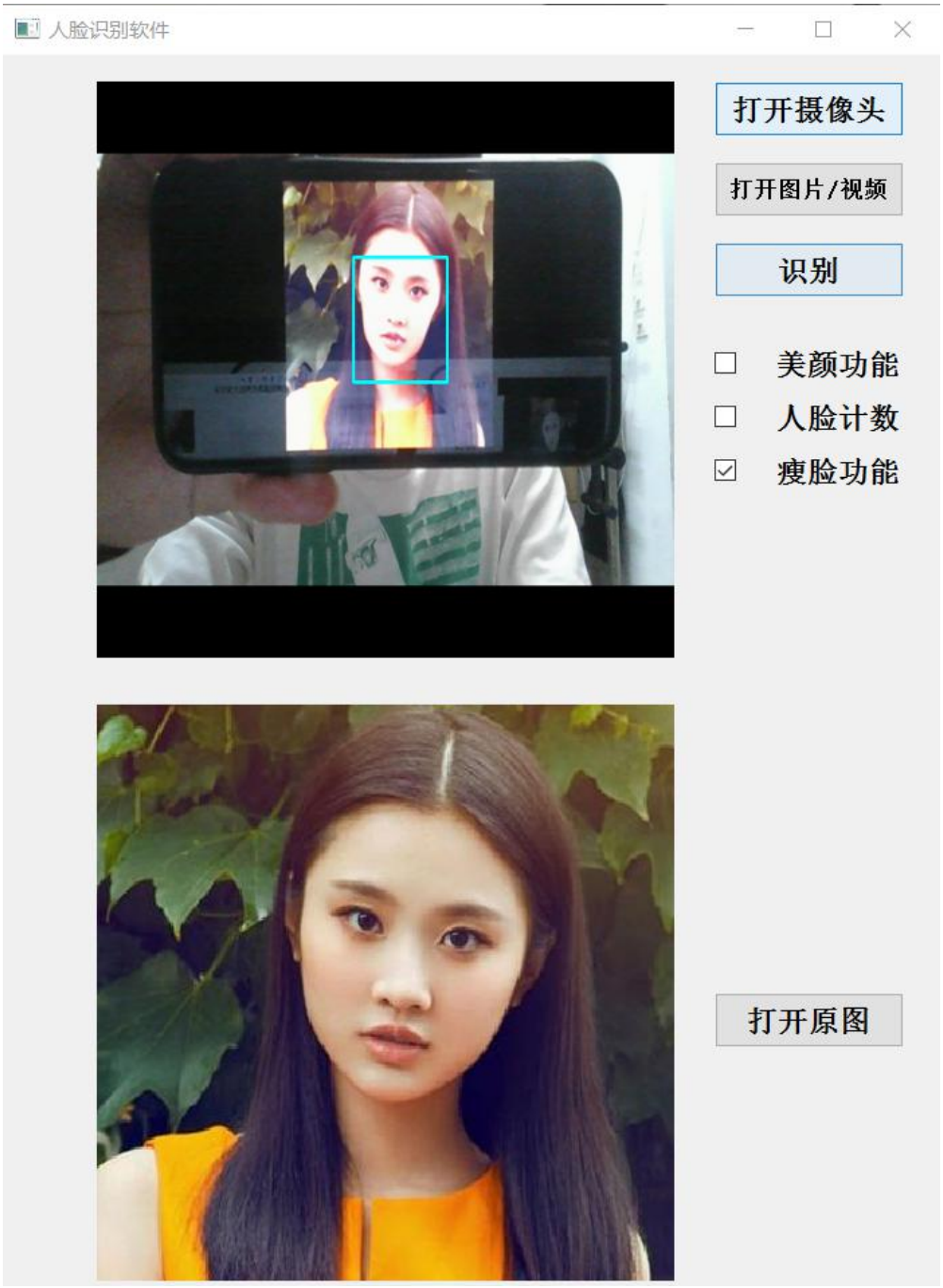


图5.12　前置摄像头瘦脸功能测试结果

6 总结与展望

本章主要总结一下本设计完成的工作，在软件开发的过程中遇到的困难，该软件的不足之处以及如何进行改进。此外还谈到通过这次设计我得到什么收获。最后对人脸识别和人脸美化等技术，以及数字图像处理技术未来发展的展望。

6.1 总结

本设计主要是为了迎合社会的发展、人工智能的需求和人们对美好外在形象的追求，而开发设计的软件。能实现基本的自动人脸识别功能，并且在检测到人脸的前提下能对人脸进行自动的美化、统计和瘦脸处理，因为是自动处理省去了PS的各种繁琐的步骤。虽然人脸识别和美化技术已经相对成熟，设计的代码比较开源，但在本设计中除了调用dlib中的模块来完成人脸识别和计数功能，其他人脸美化都是参考相关算法和计算公式，再查找各种函数用法后自己慢慢编程出来，因为大学学习的专业比较少涉及到人工智能方面，所以本设计对我来说还是花了很多精力。

特别是在设计的过程中遇到很多困难，比如初期对美颜没有具体的理解，因为美颜其实是相对比较大的领域，也可能是改变五官的标准或者自动美妆等等，所以选择和查资料时就比较混乱。同时由于经验的不足，在设计过程中需要经常的修改，比如很小的用cv2. imread()不能读取中文路径，改先用numpy. fromfile() 读取为np. uint8格式，再使用cv2. imdecode()解码、由只能识别一张人脸修改到可以识别多张人脸。以及处理图像前要先用cv2. cvtColor( )进行色彩空间的转换, 因为在opencv中默认的颜色空间是BGR等等，看起来很容易的细节都比较容易被忽略。

因为知识储备的不足，此软件虽然能实现功能，但还有很多的不足之处。比如因为手提电脑内存不足，调用了dlib中HOG+回归树的方法来检测检人脸，而没用使用卷积神经网络，它的检测效果要比前者好很多。但因为使用这个算法需要较大的内存，否则识别一张图片就要跑个几十秒。此外，尽管听说用已训练好的resnet模型识别的识别率可以达到99. 38%，但在测试中仍然会遇到识别不出的情况。还有美颜功能，暂时就只能磨皮，也就是去掉脸上的斑而已，而且磨皮的同时会降低图像的质量而使图像变得模糊一些，并且有些斑仍然不能彻底的去除。对于瘦脸功能，经过测试发现，这个功能更加适合于原来就脸很原胖的人。而对于脸本来就很瘦的人，再瘦脸的话可能会出现变形的情

况。因此还是有很多不足之处需要改善。但由于时间限制和本身知识储备的不足，暂时只能开发到这个程度。但本人对人工智能有着深厚情怀，接下来的时间我会对人脸识别与美化等领域做更进一步的研究。

但是通过这次毕业设计，我能够独立的设计出一款软件，我的编程能力和检索能力得到了很好的锻炼，阅读了大量关于人脸识别和人脸美化领域的相关知识，能更加熟练的使用python语言，懂得如何使用python语言调用opencv和dlib中各种函数来处理图像，同时对数字图像处理中底层的算法也有深入的学习，这同时也得以于老师的耐心指导和信任。

6.2 展望

事实上虽然目前的机器识别系统已经达到了一定的成熟程度，但是它们的成功受到许多实际应用所施加的条件的限制。在各种外界因素影响下获得的人脸图像的识别在很大程度上仍然是一个未解决的问题，换句话说，人脸识别技术发展还未饱和，目前的系统距离人类感知系统的能力还很远。

本软件只能检测到人的正脸，过分的侧脸会检测不到，同时也受各种外界因素的影响比如光照、表情变化等等。同时，本设计应用的磨皮算法和瘦脸算法都是比较基础的，即使效果不错可是实用性还不够高，仍然有各种各样的问题需要解决和相关算法需要优化。

当然，随着计算机网络的发展，各种社交软件也相继而出，人们都希望在设计软件上展现自己最好的一面，因为爱美之心人人皆有，所以人类对美的追求是永不停止的，所以，竟然有需求，就有很大的商业价值。我相信人脸美化领域在接下来的时间里也会不断发展和优化，希望自己也能为人工智能领域做一些贡献。

参 考 文 献

[1] 赵建章. 基于 FPGA 的红外图像采集与分层增强处理技术研究[D]. 燕山大学, 2017.

[2] supersayajin. 使用 dlib 中的深度残差网络(ResNet)实现实时人脸识别[DB/OL]. https://www.cnblogs.com/supersayajin/p/8489435.html, 2018-03-07.

[3] 高洁. 基于双目立体视觉的人脸三维建模方法研究[D]. 吉林大学, 2017.

[4] 顾徐鹏. 实用人脸识别系统：技术研究与系统实现[D]. 上海交通大学, 2011.

[5] 吕冰. 基于核技术的人脸识别应用研究[D]. 江南大学, 2007.

[6] 张文超, 胡玉兰. 基于 PyQt 的全文搜索引擎平台开发[J]. 软件导刊, 2018,17(09): 132-135 .

[7] 王喆. 面向自动柜员机智能安防的异常人脸检测技术和系统研发[D]. 中山大学, 2014.

[8] 丁宇胜. 数字图像处理中的插值算法研究[J]. 电脑知识与技术, 2010,6(16): 4502-4503

[9] 顾威威. 复杂背景下实时人脸检测技术的研究[D]. 河北工程大学, 2010.

[10] CainCary. Python3 基础:08_01_面对对象编程(OOP)[DB/OL] https://blog.csdn.net/qq_44713454/article/details/89762602,2019-05-23.

[11] ItchyHacker. 【OPENCV】高反差保留算法[DB/OL] https://www.jianshu.com/p/bb702124d2ad , 2018-08-06.

[12] laviewpbt. 简单探讨可牛影像软件中具有肤质保留功能的磨皮算法及其实现细节 [DB/OL] http://www.cnblogs.com/Imageshop/p/4709710.html,2015-08-06.

[13] grafx. 图像处理算法之瘦脸及放大眼睛[DB/OL] https://blog.csdn.net/grafx/article/details/70232797 , 2017-04-08.

致 谢

首先特别感谢杨志景老师在大四这一学期的指导和照顾，尽管在本设计的过程中出现各种出国考试的缠累，老师能够理解和耐心的指导，这使我十分感动。老师渊博的专业知识，诲人不倦的高尚师德，宽以待人的崇高风范，朴实无华、平易近人的人格魅力对我影响深远。不仅使我树立了远大的学术目标、掌握了基本的研究方法，还使我明白了许多待接物与为人处世的道理。本论文从选题到完成，每一步都离不开导师的指导，倾注了导师大量得心血，在此谨向导师表示崇高的敬意和衷心的感谢!

同时我也要感谢大学生涯以来一路上各位老师的教导，回顾大学四年，其实每位老师都是可爱的，同时他们为国家做的贡献是不容置疑的，在此献上我对他们深深的敬意。

最后感谢家人时刻的关怀与支持，感谢努力的自己。