基于 keras 和 tensorflow 的人脸识别技术

摘要

人脸识别， 是基于人的脸部特征信息进行身份识别的一种生物识别技术。用摄像机或摄像头,采集含有人脸的图像或视频流， 并自动在图像中检测和跟踪人脸,进而对检测到的人脸进行脸部识别的一系列相关技术。人脸与人体的其它生物特征一样与生俱来,它的唯一性和不易被复制的良好特性为身份鉴别提供了必要的前提。

人脸识别技术看似是近年来比较火的一个话题,但是对人脸识别的研究却是从 1888 年就开始了。但是当时还不涉及到人脸的自动识别。而真正进入人脸识别研究高潮期则是从 1991 年开始的。从 91 年到 97 年期间不仅诞生了若干代表性的人脸识别算法,还出现了很多商业化运作的人脸识别系统。到了 21 世界，人脸识别在商业系统中得到了进一步发展。其实不仅是人脸识别技术整个生物识别领域都在迅猛发展。如今人脸识别技术在公共安全、执法、移动互联网和娱乐领域都有大量应用。它也成为检验人工智能是否可以在解决某些特定智能问题上达到甚至超越人的重要测试基准。

我们通过人工智能学习系统 TensorFlow 平台，兼顾视频人脸识别中识别准确率和实时性,基于卷积神经网络 CNN 人脸识别方法。构建了一个多层结构的 CNN 人脸识别网络,通过神经网络的算法检测到的人脸输入所构建的 CNN 中进行视频人脸识别。此外为了更适用于实际视频监控情况,通过对 CNN 网络结构末尾 Softmax 分类器的分类结果进行多级判决引入了开集人脸识别功能。从多个角度对该方法进行了实验验证,结果证明,此方法可满足识别准确率和实时性要求,同时对于视频中人脸姿态变化、光照变化、距离远近等都具有良好的鲁棒性。

关键字：Keras、tensorflow、Python、人脸识别、卷积神经网络、机器学习

一、作品介绍

1.1 知识背景

随着安全入口控制和金融贸易方面应用需要的快速增长，生物统计识别技术得到了新的重视，而人脸识别是所有的生物识别方法中应用最广泛的技术之一。人脸识别因其在安全验证系统、信用卡验证、医学、档案管理、视频会议、人机交互、系统公安(罪犯识别等)等方面的巨大应用前景而越来越成为前模式识别和人工智能领域的一个研究热点。经过近四十年的研究发展，人脸识别技术得到了长足的发展并投入到了商业应用。但实践表明人脸识别技术还远不够成熟，特别是在大规模数据库条件下，存在识别率较低、识别时间较长等严重问题。

人脸识别技术看似是近年来比较火的一个话题,但是对人脸识别的研究却是从 1888 年就开始了。但是当时还不涉及到人脸的自动识别。而真正进入人脸识别研究高潮期则是从 1991 年开始的。从 91 年到 97 年期间不仅诞生了若干代表性的人脸识别算法,还出现了很多商业化运作的人脸识别系统。到了 21 世界，人脸识别在商业系统中得到了进一步发展。其实不仅是人脸识别技术整个生物识别领域都在迅猛发展。如今人脸识别技术在公共安全、执法、移动互联网和娱乐领域都有大量应用。它也成为检验人工智能是否可以在解决某些特定智能问题上达到甚至超越人的重要测试基准。

1.2 模型训练流程

（1）配置、获取实时视频流：利用 python 写一个程序，读取 USB 摄像头的视频流。

（2）建立人脸识别模型：从实时视频流中识别出人脸区域，从原理上看，其依然属于机器学习的领域之一，本质上与谷歌利用深度学习识别出猫并无区别。程序通过大量的人脸图片数据进行训练，利用数学算法建立建立可靠的人脸特征模型，如此即可识别出人脸。

（3）为模型训练准备人脸数据： 机器学习最本质的地方就是基于海量数据统计的学习。我们利用了 keras 这个深度学习库来训练人脸识别模型。keras 是一个上层的神经网络学习库，纯 python 编写，被集成进了 Tensorflow 和 Theano 这样的深度学习框架。

（4）利用卷积神经网络（CNN）和 Keras 库训练人脸识别模型：CNN 擅长图像处理，keras 库的 tensorflow 版亦支持此种网络模型。将训练数据准备好，并将其输入给 CNN。

二、模型建立与求解

2.1Python 语言与其环境配置

2.1.1Python 语言简介

Python，是一种面向对象的解释型计算机程序设计语言，由荷兰人 Guido van Rossum 于 1989 年发明，第一个公开发行版发行于 1991 年。

Python 是纯粹的自由软件， 源代码和解释器 CPython 遵循 GPL(GNU General Public License)协议。Python 语法简洁清晰，特色之一是强制用空白符(white space)作为语句缩进。

Python 具有丰富和强大的库。它常被昵称为胶水语言，能够把用其他语言制作的各种模块（尤其是 C/C++）很轻松地联结在一起。常见的一种应用情形是，使用 Python 快速生成程序的原型（有时甚至是程序的最终界面），然后对其中有特别要求的部分，用更合适的语言改写，比如 3D 游戏中的图形渲染模块，性能要求特别高，就可以用 C/C++重写，而后封装为 Python 可以调用的扩展类库。需要注意的是在使用扩展类库时可能需要考虑平台问题，某些可能不提供跨平台的实现。

由于 Python 语言的简洁性、易读性以及可扩展性，在国外用 Python 做科学计算的研究机构日益增多，一些知名大学已经采用 Python 来教授程序设计课程。例如卡耐基梅隆大学的编程基础、麻省理工学院的计算机科学及编程导论就使用 Python 语言讲授。众多开源的科学计算软件包都提供了 Python 的调用接口，例如著名的计算机视觉库 OpenCV、三维可视化库 VTK、医学图像处理库 ITK。而 Python 专用的科学计算扩展库就更多了，例如如下 3 个十分经典的科学计算扩展库：NumPy、SciPy 和 matplotlib，它们分别为 Python 提供了快速数组处理、数值运算以及绘图功能。因此 Python 语言及其众多的扩展库所构成的开发环境十分适合工程技术、科研人员处理实验数据、制作图表，甚至开发科学计算应用程序。

本项目中就使用到了 OpenCV 和 TensorFlow 等技术。

2.1.2 环境配置方法

Python 可应用于多平台包括 Linux 和 Mac OS X。

你可以通过终端窗口输入 "python" 命令来查看本地是否已经安装 Python 以及 Python 的安装版本。

Python 已经被移植在许多平台上（经过改动使它能够工作在不同平台上）。

需要下载适用于使用平台的二进制代码，然后安装 Python。

如果平台的二进制代码是不可用的，需要使用 C 编译器手动编译源代码。

编译的源代码，功能上有更多的选择性， 为 python 安装提供了更多的灵活性。

下面几个重要的环境变量，它应用于 Python：

变量名	描述
PYTHONPATH	PYTHONPATH是Python搜索路径，默认我们import的模块都会从PYTHONPATH里面寻找。
PYTHONSTARTUP	Python启动后，先寻找PYTHONSTARTUP环境变量，然后执行此变量指定的文件中的代码。
PYTHONCASEOK	加入PYTHONCASEOK的环境变量, 就会使python导入模块的时候不区分大小写.
PYTHONHOME	另一种模块搜索路径。它通常内嵌于的PYTHONSTARTUP或PYTHONPATH目录中，使得两个模块库更容易切换。

2.2TensorFlow 及其部署

2.2.1TensorFlow 简介

TensorFlow 是谷歌基于 DistBelief 进行研发的第二代人工智能学习系统，其命名来源于本身的运行原理。Tensor（张量）意味着 N 维数组，Flow（流）意味着基于数据流图的计算，TensorFlow 为张量从流图的一端流动到另一端计算过程。TensorFlow 是将复杂的数据结构传输至人工智能神经网中进行分析和处理过程的系统。

TensorFlow 可被用于语音识别或图像识别等多项机器学习和深度学习领域，对 2011 年开发的深度学习基础架构 DistBelief 进行了各方面的改进，它可在小到一部智能手机、大到数千台数据中心服务器的各种设备上运行。TensorFlow 将完全开源，任何人都可以用。

2.2.2TensorFlow 部署方法

目前可支持如下几种方式：

1. “原生” pip
2. Anaconda

原生 pip 会直接在系统上安装 TensorFlow，而不是通过虚拟环境。原生 pip 安装并未隔离在单独的容器中进行，因此可能会干扰系统中其他基于 Python 的安装。但是，如果熟悉 pip 和 Python 环境，通常只需一条命令即可进行原生 pip 安装。此外，如果使用原生 pip 安装，用户可以从系统上的任何目录运行 TensorFlow 程序。

在 Anaconda 中，可以使用 conda 来创建一个虚拟环境。但是，在 Anaconda 内部，建议使用 pip install 命令来安装 TensorFlow，而不要使用 conda install 命令

本项目中使用的是支持 GPU 的 TensorFlow。因为 TensorFlow 程序在 GPU 上的运行 速度通常要比在 CPU 上快得多。

三、模型介绍

3.1 神经网络构建

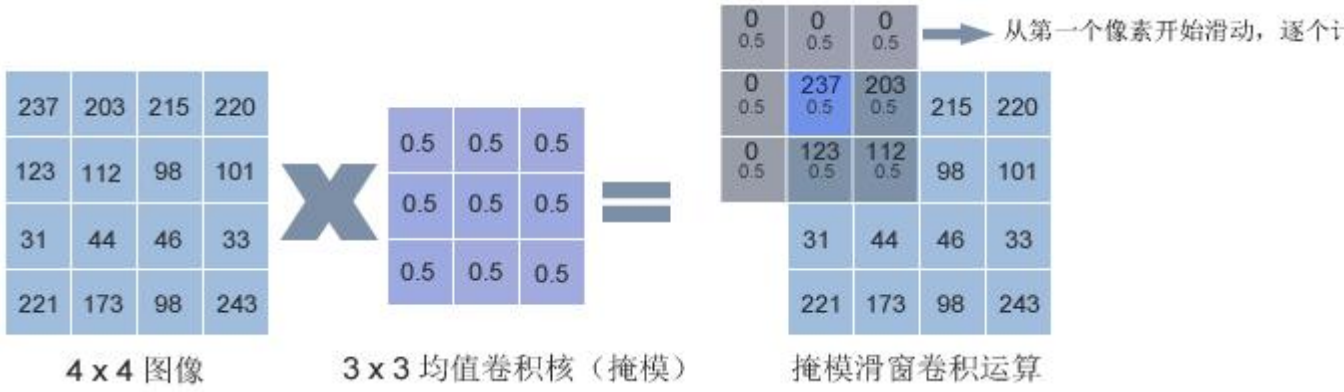
关于预处理，我们做了几项工作：
1）按照交叉验证的原则将数据集划分成三部分：训练集、验证集、测试集；
2）按照 keras 库运行的后端系统要求改变图像数据的维度顺序；
3）将数据标签进行 one-hot 编码，使其向量化
4）归一化图像数据
关于第一项工作，交叉验证属于机器学习中常用的精度测试方法，它的目的是提升模型的可靠和稳定性。我们会拿出大部分数据用于模型训练，小部分数据用于对训练后的模型验证，验证结果会与验证集真实值（即标签值）比较并计算出**差平方和**，此项工作重复进行，直至所有验证结果与真实值相同，交叉验证结束，模型交付使用。
关于第二项工作，keras 建立在 tensorflow 或 theano 基础上，换句话说，keras 的后端系统可以是 tensorflow 也可以是 theano。后端系统决定了图像数据输入 CNN 网络时的维度顺序，tensorflow 的维度顺序为行数(rows)、列数(cols)、通道数(颜色通道，channels)；theano 则是通道数、行数、列数。所以，我们通过调用 image_dim_ordering()函数来确

定后端系统的类型（‘th’代表 theano，'tf'代表 tensorflow），然后我们再通过 numpy 提供的 reshape()函数重新调整数组维度。

关于第三项工作，对标签集进行 one-hot 编码的原因是我们的训练模型采用 categorical_crossentropy 作为损失函数（多分类问题的常用函数，后面会详解），这个函数要求标签集必须采用 one-hot 编码形式。所以，我们对训练集、验证集和测试集标签均做了编码转换。

关于第四项工作，数据集先浮点后归一化的目的是提升网络收敛速度，减少训练时间，同时适应值域在（0,1）之间的激活函数，增大区分度。其实归一化有一个特别重要的原因是确保特征值权重一致。举个例子，我们使用 mse 这样的均方误差函数时，大的特征数值比如$(5000-1000)^2$与小的特征值$(3-1)^2$相加再求平均得到的误差值，显然大值对误差值的影响最大，但大部分情况下，特征值的权重应该是一样的，只是因为单位不同才导致数值相差甚大。因此，我们提前对特征数据做归一化处理，以解决此类问题。

卷积层（convolution layer）：Convolution2D()函数。根据 keras 官方文档描述，2D 代表这是一个 2 维卷积，其功能为对 2 维输入进行滑窗卷积计算。我们的脸部图像尺寸为 64*64，拥有长、宽两维，所以在这里我们使用 2 维卷积函数计算卷积。所谓的滑窗计算，其实就是利用卷积核逐个像素、顺序进行计算，如下图：



上图选择了最简单的均值卷积核，3x3 大小，我们用这个卷积核作为掩模对前面 4x4 大小的图像逐个像素作卷积运算。首先我们将卷积核中心对准图像第一个像素，在这里就是像素值为 237 的那个像素。卷积核覆盖的区域（**掩模**之称即由此来），其下所有像素取均值然后相加：

C(1) = 0 * 0.5 + 0 * 0.5 + 0 * 0.5 + 0 * 0.5 + 237 * 0.5 + 203 * 0.5 + 0 * 0.5 + 123 * 0.5 + 112 * 0.5

结果直接替换卷积核中心覆盖的像素值，接着是第二个像素、然后第三个，从左至右，由上到下……以此类推，卷积核逐个覆盖所有像素。整个操作过程就像一个滑动的窗口逐个滑过所有像素，最终生成一副尺寸相同但已经过卷积处理的图像。上图我们采用的是均值卷积核，实际效果就是将图像变模糊了。显然，卷积核覆盖图像边界像素时，会有部分区域越界，越界的部分我们以 0 填充，如上图。对于此种情况，还有一种处理方法，就是丢掉边界像素，从覆盖区域不越界的像素开始计算。像上图，如果采用丢掉边界像素的方法，3x3 的卷积核就应该从第 2 行第 2 列的像素（值为 112）开始，到第 3 行第 3 列结束，最终我们会得到一个 2x2 的图像。这种处理方式会丢掉图像的边界特征；而第一种方式则保留了图像的边界特征。在我们建立的模型中，卷积层采用哪种方式处理图像边界，卷积核尺寸有多大等参数都可以通过 Convolution2D()函数来指定：

```
self.model.add(Convolution2D(32, 3, 3, border_mode='same', input_shape =
dataset.input_shape))
```

第一个卷积层包含 32 个卷积核，每个卷积核大小为 3x3，border_mode 值为“same”意味着我们采用保留边界特征的方式滑窗，而值“valid”则指定丢掉边界像素。根据 keras 开发文档的说明，当我们将卷积层作为网络的第一层时，我们还应指定 input_shape 参数，显式地告知输入数据的形状，对我们的程序来说，input_shape 的值为(64,64,3)，来自 Dataset 类，代表 64x64 的彩色 RGB 图像。

有关卷积层的内容在下文有更加具体的介绍。

激活函数层：代码中采用的 relu（Rectified Linear Units，修正线性单元）函数，它的数学形式如下：

$f(x) = \max(0, x)$

这个函数非常简单，其输出一目了然，小于 0 的输入，输出全部为 0，大于 0 的则输入与输出相等。该函数的优点是收敛速度快，除了它，keras 库还支持其它几种激活函数，如下：

softplus

softsign

tanh

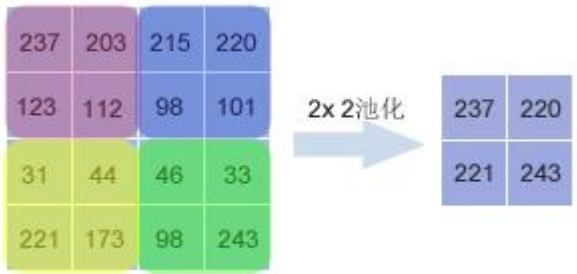
sigmoid

hard_sigmoid

linear

对于不同的需求，我们可以选择不同的激活函数，这也是模型训练可调整的一部分。

池化层（pooling layer）：池化层存在的目的是缩小输入的特征图，简化网络计算复杂度；同时进行特征压缩，突出主要特征。我们通过调用 MaxPooling2D()函数建立了池化层，这个函数采用了最大值池化法，这个方法选取覆盖区域的最大值作为区域主要特征组成新的缩小后的特征图：



显然，池化层与卷积层覆盖区域的方法不同，前者按照池化尺寸逐块覆盖特征图，卷积层则是逐个像素滑动覆盖。对于我们输入的 64x64 的脸部特征图来说，经过 2x2 池化后，图像变为 32x32 大小。

Dropout 层：随机断开一定百分比的输入神经元链接，以防止过拟合。那么什么是**过拟合**呢？一句话解释就是训练数据预测准确率很高，测试数据预测准确率很低，用图形表示就是拟合曲线较尖，不平滑。导致这种现象的原因是模型的参数很多，但训练样本太少，导致模型拟合过度。为了解决这个问题，Dropout 层将有意识的随机减少模型参数，让模型变得

简单，而越简单的模型越不容易产生过拟合。代码中 Dropout()函数只有一个输入参数——指定抛弃比率，范围为 0～1 之间的浮点数，其实就是百分比。这个参数亦是一个可调参数，我们可以根据训练结果调整它以达到更好的模型成熟度。

Flatten 层：截止到 Flatten 层之前，在网络中流动的数据还是多维的（对于我们的程序就是 2 维的），经过多次的卷积、池化、Dropout 之后，到了这里就可以进入全连接层做最后的处理了。全连接层要求输入的数据必须是一维的，因此，我们必须把输入数据“压扁”成一维后才能进入全连接层，Flatten 层的作用即在于此。该层的作用如此纯粹，因此反映到代码上我们看到它不需要任何输入参数。

全连接层（dense layer）：全连接层的作用就是用于分类或回归，对于我们来说就是分类。keras 将全连接层定义为 Dense 层，其含义就是这里的神经元连接非常“稠密”。我们通过 Dense()函数定义全连接层。这个函数的一个必填参数就是神经元个数，其实就是指定该层有多少个输出。在我们的代码中，第一个全连接层（#14 Dense 层）指定了 512 个神经元，也就是保留了 512 个特征输出到下一层。这个参数可以根据实际训练情况进行调整，依然是没有可参考的调整标准，自调之。

分类层：全连接层最终的目的就是完成我们的分类要求：0 或者 1，模型构建代码的最后两行完成此项工作：

```
self.model.add(Dense(nb_classes))                                    #17 Dense 层
self.model.add(Activation('softmax'))                                #18 分类层，输出
```

3.2 网络模型

3.2.1Keras 简介

Keras 的底层库使用 Theano 或 TensorFlow，这两个库也称为 Keras 的后端。无论是 Theano 还是 TensorFlow，都是一个“符号式”的库。

因此，这也使得 Keras 的编程与传统的 Python 代码有所差别。笼统的说，符号主义的计算首先定义各种变量，然后建立一个“计算图”，计算图规定了各个变量之间的计算关系。建立好的计算图需要编译以确定其内部细节，然而，此时的计算图还是一个“空壳子”，里面没有任何实际的数据，只有当你把需要运算的输入放进去后，才能在整个模型中形成数据流，从而形成输出值。

Keras 的模型搭建形式就是这种方法，在搭建 Keras 模型完毕后，模型就是一个空壳子，只有实际生成可调用的函数后（K.function），输入数据，才会形成真正的数据流。

Keras 有两种类型的模型，序贯模型（Sequential）和函数式模型（Model），函数式模型应用更为广泛，序贯模型是函数式模型的一种特殊情况。

3.2.2 卷积层

使用卷积层的原因是卷积运算的一个重要特点，通过卷积运算，可以使原信号特征增强，并且降低噪音用 6 个 5x5 的过滤器进行卷积，结果是在卷积层 C1 中，得到 6 张特征图，特征图的每个神经元与输入图片中的 5x5 的领域相连，即用 5x5 的卷积核去卷积输入层，由卷积运算可得 C1 层输出的特征图大小为（32-5+1）x（32-5+1）=28x28。

1.2 降采样层

使用将采样的原因是，根据图像局部相关性的原理，对图像进行子采样可以减少计算量，同时保持图像旋转不变性。降采样后，降采样层 S2 的输出特征图大小为（28÷2）x（28÷2）=14x14。

1.3 全连接层

采用 softmax 全连接，得到的激活值即卷积神经网络提取到的图片特征。

(1) Conv1D 层

一维卷积层（即时域卷积），用以在一维输入信号上进行邻域滤波。当使用该层作为首层时，需要提供关键字参数 input_shape。例如(10,128)代表一个长为 10 的序列，序列中每个信号为 128 向量。而(None, 128)代表变长的 128 维向量序列。

该层生成将输入信号与卷积核按照单一的空域（或时域）方向进行卷积。如果 use_bias=True，则还会加上一个偏置项，若 activation 不为 None，则输出为经过激活函数的输出。

(2) Conv2D 层

二维卷积层，即对图像的空域卷积。该层对二维输入进行滑动窗卷积，当使用该层作为第一层时，应提供 input_shape 参数。例如 input_shape = (128,128,3)代表 128*128 的彩色 RGB 图像（data_format='channels_last'）

(3) SeparableConv2D 层

该层是在深度方向上的可分离卷积。

可分离卷积首先按深度方向进行卷积（对每个输入通道分别卷积），然后逐点进行卷积，将上一步的卷积结果混合到输出通道中。参数 depth_multiplier 控制了在 depthwise 卷积（第一步）的过程中，每个输入通道信号产生多少个输出通道。

直观来说，可分离卷积可以看做讲一个卷积核分解为两个小的卷积核，或看作 Inception 模块的一种极端情况。

当使用该层作为第一层时，应提供 input_shape 参数。例如 input_shape = (3,128,128)代表 128*128 的彩色 RGB 图像

(4) Conv2DTranspose 层

该层是转置的卷积操作（反卷积）。需要反卷积的情况通常发生在用户想要对一个普通卷积的结果做反方向的变换。例如，将具有该卷积层输出 shape 的 tensor 转换为具有该卷积层输入 shape 的 tensor。同时保留与卷积层兼容的连接模式。

当使用该层作为第一层时，应提供 input_shape 参数。例如 input_shape = (3,128,128)代表 128*128 的彩色 RGB 图像

(5) Conv3D 层

三维卷积对三维的输入进行滑动窗卷积，当使用该层作为第一层时，应提供 input_shape 参数。例如 input_shape = (3,10,128,128)代表对 10 帧 128*128 的彩色 RGB 图像进行卷积。数据的通道位置仍然有 data_format 参数指定。

(6) Cropping1D 层

在时间轴（axis1）上对 1D 输入（即时间序列）进行裁剪

(7) Cropping2D 层

对 2D 输入（图像）进行裁剪，将在空域维度，即宽和高的方向上裁剪

(8) Cropping3D 层

对 2D 输入（图像）进行裁剪

(9) UpSampling1D 层

在时间轴上，将每个时间步重复 length 次

(10) UpSampling2D 层

将数据的行和列分别重复 size[0]和 size[1]次

(11) UpSampling3D 层

将数据的三个维度上分别重复 size[0]、size[1]和 ize[2]次

本层目前只能在使用 Theano 为后端时可用

(12) ZeroPadding1D 层

对 1D 输入的首尾端（如时域序列）填充 0，以控制卷积以后向量的长度

(13) ZeroPadding2D 层

对 2D 输入（如图片）的边界填充 0，以控制卷积以后特征图的大小

（14）ZeroPadding3D 层

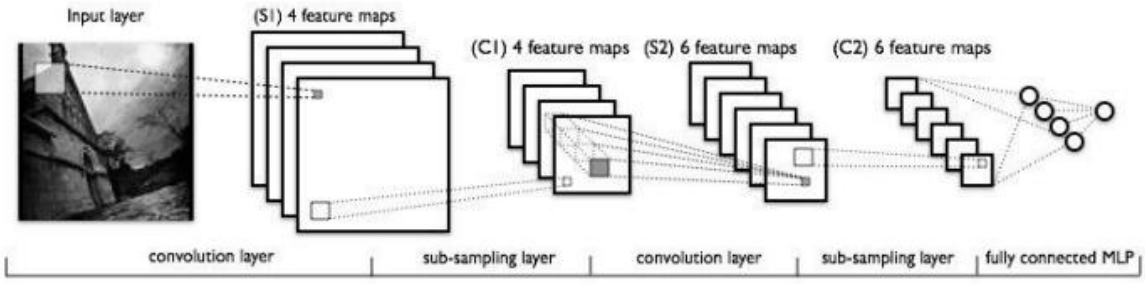
将数据的三个维度上填充 0

本层目前只能在使用 Theano 为后端时可用

3.3 卷积神经网络（CNN）

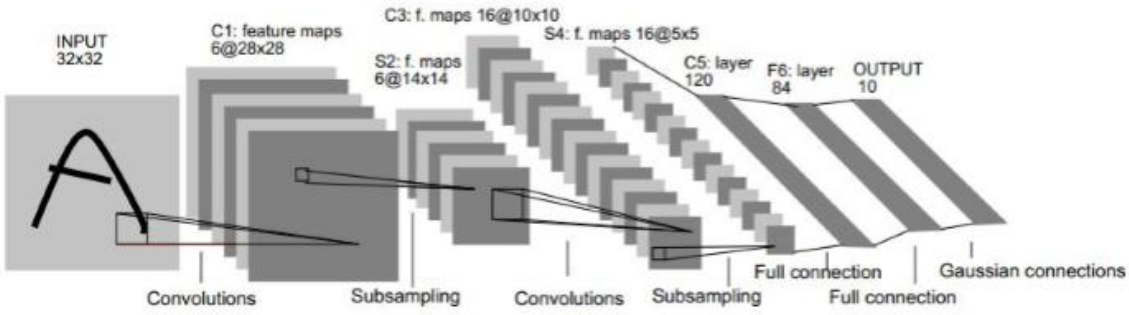
3.3.1 卷积神经网络的结构

对于图像特征提取任务，卷积神经网络的一般结构如图：



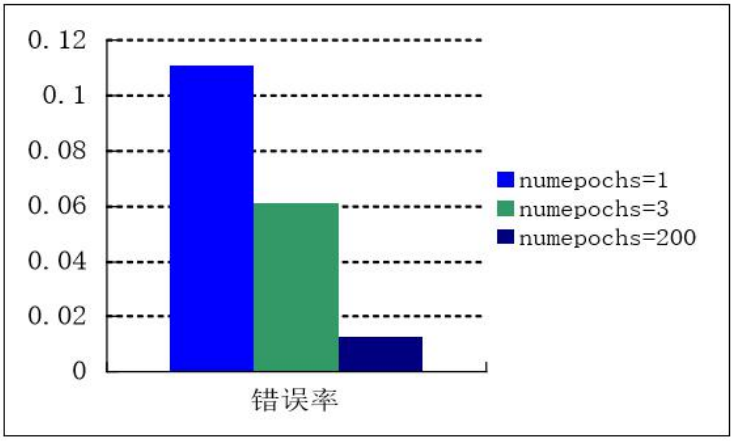
卷积神经网络结构包括：卷积层，降采样层，全连接层。每一层有多个特征图，每个特征图通过一种卷积滤波
器提取输入的一种特征，每个特征图有多个神经元。

如图来解释各层的具体操作。



3.3.2 手写数字库

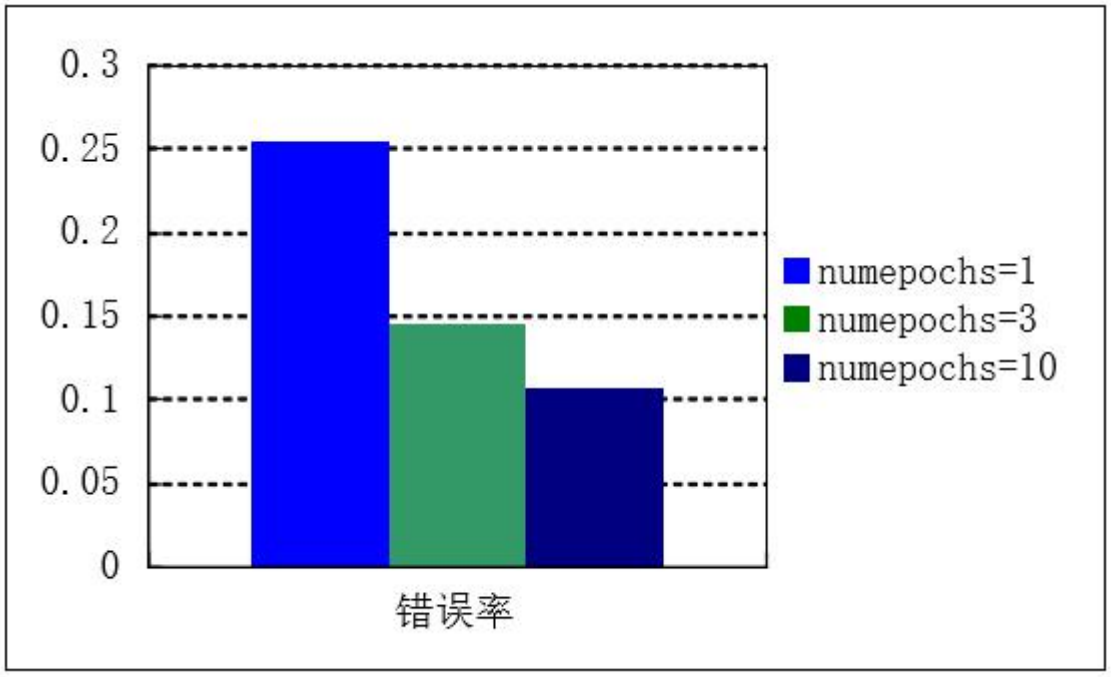
手写数字库分为训练部分和测试部分。训练部分包括 10 个阿拉伯手写数字（0 到 9）共 60000 张图片，均带有标签；测试部分包括 10 个阿拉伯手写数字 (0 到 9) 不同的 10000 张图片和标签。测试结果如图。



epoch 代表训练的次数，平均每次训练耗时 120ms。由图可见 CNN 在手写数字库的数据库上识别效果很好，进而验证了 CNN 在提取图像特征上的高准确率。

2.2 风景人脸图片混合库

创建一个数据库包含 600 组训练图片和 440 组测试图片，包括均是人脸图片和不全是人脸图片的图片对。测试结果如图：



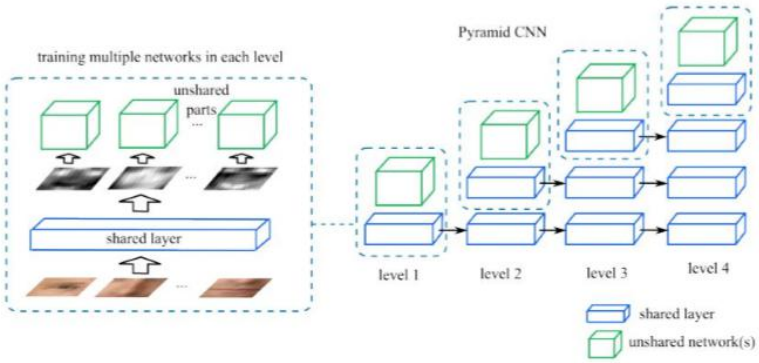
epoch 代表训练的次数。由图可见 CNN 在风景人脸混合数据库这样两种图片特征差别较大的数据上识别效果依然很好，表明 CNN 在提取图像特征上准确率较高。

3.3.3 PCNN 人脸识别

验证了 CNN 在提取图像特征上较高的准确性以后，根据深度人脸表示的相关研究，利 用一种对 CNN 进行一定的改进，名为“金字塔”型 CNN（Pyramid CNN，后简称 PCNN）
的算法结构进行人脸识别。

PyramidCNN 是一种理论上非常易于理解且效果优良的深度学习算法结构，同样采用过 滤器和降采样交替的操作，使训练过程非常快速且具有高效率的计算。另外，PCNN 可以自
然地做到多尺度脸部特征的共享，增加了结果表示的识别能力。

PCNN 理论[12-13]如下图所示：



将两张图片送入同一个 CNN 中，得到各自相应的表示结果。然后通过一个输出神经元
利用一个距离函数来比较这两个表示并作出预测是否这组图片属于同一个人。采用的损失函数如下：

$$L = \sum_{I_1, I_2} \log(1 + \exp(\delta(I_1, I_2) D(I_1, I_2)))$$

$$D(I_1, I_2) = \alpha \cdot d(f_\theta(I_1), f_\theta(I_2)) - \beta$$

其中 $\delta(I_1, I_2)$ 表示这两张图片是否输入同一个人，在我们的实验中 0 代表不是同一个人，1 代表是同一个人。 f_θ 代表神经网络中做的运算，α、β 都是训练参数

损失函数鼓励了属于同一个人的不同特征之间的微小距离，对不匹配的组队之间的似予以惩罚。这样学习到的特征表现出了优良的 ID-preserving 性质，即映射空间中的距离应
该基本可以反映真实的语义距离，不相关因素的影响应被减小。我们的理解是，通过这样的损失函数得到的特征应该能比较全面地概括一个人的人脸，由于个体内部差异导致的因素被压制了。

PCNN 的动机在于利用脸部的多层结构来加快深度神经网络的训练速度。

PCNN 包含了数个 CNN 网络，它们被分成了数级，每级包含一个或多个 CNN 网络，每层 CNN 网络有不同的深度和输入大小，但它们之间共享一些训练层。每个网络都包含几
层共享层和一个在所有级都具有相同结构的不共享部分。

第一层被所有级的网络共享，第二层被从第二级开始的所有网络共享，这个共享理不断重复。由于共享层的降采样操作，且更高级的网络的输入尺寸更大，在高级网络组中可能存在不止一个网络，它们虽然也共享共享层的参数，但却作用在人脸的不同区域上。

下面是具体操作：

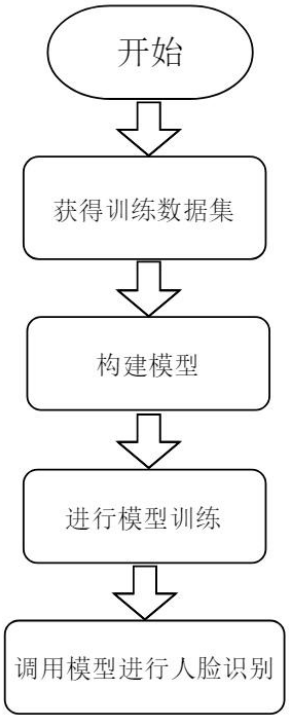
第一级 CNN 网络在人脸图片的部分内容上训练，然后其第一组卷积层和降采样层的各 项参数训练完毕后固定不变，传递到下一级的 CNN 网络，并被之后所有的 CNN 网络共享， 用于过滤和下采样处理高级 CNN 网络的输入。

类似的，第二级 CNN 网络训练完毕后其第二组卷积层和降采样层（即第二级 CNN 网 络相较于第一级 CNN 网络新增的第一组卷积层和降采样层）各项参数固定不变，和之前共 享的一组卷积层和降采样层一起传递到下一级组的 CNN 网络，并被之后所有的高级 CNN
网络共享。

这样的训练方式在 PCNN 网络中继续，知道所有的网络都被训练完毕，并且最后一个有额外深度的网络已经获得。采取这种方式后，实际需要训练的网络的大小大大减小，并没有像简单的 CNN 叠加那样随着层数的递增而增加。而在一个级别中可包含超过一个网络的 目的是补偿低级别网络无法覆盖输入图片所有区域的缺陷。因为 PCNN 是在强力的监督模式下学习的，这样一来各个级次的网络学习到的特征都与图像识别直接相关所以能够保证最终能提取出具有强辨识度的信息；其次 PCNN 是一种多层特征提取结构，能够自然地处理在人脸识别中常见的多层输入数据块，利用多层结构实现对于越大的输入区域便使用越深的的网络。网络深度的增加使得更高级的网络能够处理在更大的图像块上更复杂和更抽象的运算。

四、软件设计

4.1 软件总体设计流程：



4.1.1 环境配置

支持的版本为本文使用到的环境及其版本为：

软件	版本号
Anaconda	1.8.7
tensorflow	1.8
python	3.5.4
opencv	3.3.0
Scikit-learn	不限
Spyder（）	3.2.8

注：在安装 python 时一定要将所有插件安装完成，尤其是 PIP 插件。

4.1.2 训练人脸模型

通过人脸照片数据集 FDDB 中大量的人脸图片，通过我们先前介绍的 keras 库下的 CNN，对照片数据进行训练，这一段时间会很长。第二种方法，opencv 已经将人脸数据模型封装好，如果之前一步出现问题，可以直接调用 opencv 的函数。

4.1.3 训练自己的数据模型

使用自己的图片（为了确保精度，要控制到 500-1000 张图片）进行训练，并通过之前训练好的模型文件，综合基于 tensorflow 的 keras 模型下的卷积层网络，进行人脸数据训练，由于这一步头像采集量较少，所以训练时间会比较短（当然为了模型的精准，最好使用大量的照片模型）。

经过前面稍显罗嗦的准备工作，现在，我们终于可以尝试训练我们自己的卷积神经网络模型了。CNN 擅长图像处理，keras 库的 tensorflow 版亦支持此种网络模型，万事俱备，就放开手做吧。前面说过，我们需要通过大量的训练数据训练我们的模型，因此首先要做的就

是把训练数据准备好，并将其输入给 CNN。前面我们已经准备好了 2000 张脸部图像，但没有进行标注，并且还需要将数据加载到内存，以方便输入给 CNN。因此，第一步工作就是加载并标注数据到内存。

首先我们建立一个空白的 python 文件，文件名为：load_face_dataset.py，

上面给出的代码主函数就是 load_dataset()，它将图片数据进行标注并以多维数组的形式加载到内存中。我实际用于训练的脸部数据共 **1200** 张，我去掉了一些模糊的或者表情基本一致的头像，留下了清晰、脸部表情有些区别的，我和闺女各留了 600 张，所以训练数据变成了 1200。上述代码注释很清楚，不多讲，唯一一个理解起来稍微有点难度的就是 resize_image()函数。这个函数其实就做了一件事情，判断图片是不是四边等长，也就是图片是不是正方形。如果不是，则短的那两边增加两条黑色的边框，使图像变成正方形，这样再调用 cv2.resize()函数就可以实现等比例缩放了。因为我们指定缩放的比例就是 64 x 64，只有缩放之前图像为正方形才能确保图像不失真。

4.1.4 获取并显示 USB 摄像头

Opencv 在这里起了关键性的作用，利用 opencv 加几行代码，可以读取摄像头信息。

其实，利用 OpenCV 获取 USB 摄像头的视频流并展示出来非常简单，拢共不超过 30 行代码：

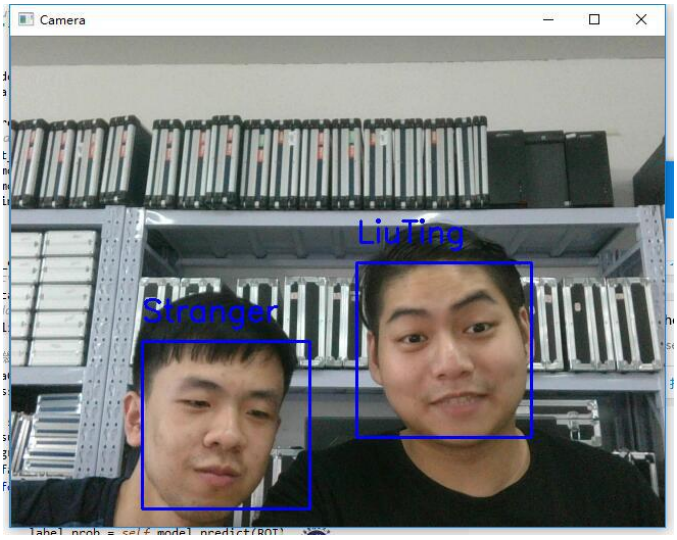
其中，唯一需要解释的就是 camera_id，这个就是 USB 摄像头的索引号，一般是 0，如果 0 不行可以试试 1、2 等，除非你的摄像头已经坏了，根本不能用。

4.1.5 从实时视频流识别出“我”

将所有的模型放到一个目录下，运行 read_camera.py，就可以得到我们的人脸模型

终于到了最后一步，激动时刻就要来临了，先平复一下心情，把剩下的代码加上，首先是为 Model 类增加一个预测函数：

这个函数是提供给外部模块使用的，外部模块用它来预测哪个是“**我**”，哪个不是“**我**”。代码很简单，注释也很详细，就不多解释了。接下来我们新建一个 python 文件：face_predict_use_keras.py，然后为这个文件添加如下代码：这个就是我们的最终程序，它能够从 USB 拍摄的实时视频流中找出哪一个是我，先看识别结果：



五、作品评价

5.1 作品优点

可以较好地展示并实现人脸，并且因为模型数据量较多，所以人像辨别精度较高。

5.2 作品不足

1 完成人脸数据集的神经网络优化程度不够高，需要训练较长的时间。

2. 关于人脸识别由于在 CNN 下进行，对初始数据量要求较大，由于小组成员照片有限，在实验过程中会出现人名显示错误等问题，这个问题现象较为严重。

5.3 作品应用领域推广

1、刷脸支付

通过应用人脸识别系统,我们就可以走上靠脸吃饭的路。在结账的时候, 我们只需对着刷卡设备的摄像头,系统就会扫描消费者面部,再把图像与数据库中的存储信息进行对比。当两者信息匹配的时候就能完成交易。

2、刷脸解锁屏幕

相信这个功能大家并不陌生, 在指纹解锁手机火热的同时, 通过面部识别来解锁手机同样受到追捧。两者都是通过生物识别技术来达到简化操作的目的。

3、身份识别

在机场、体育场、超市等公共场所对人群进行监视,例如在机场安装监视系统以防止恐怖分子登机;在机场或车站安装人脸识别系统以帮忙抓捕嫌犯。

4、门禁

可用于企业、住宅等受安全保护的地区,通过人脸识别辨认进入者身份以达到安全保护的目的。

六、参考文献

[tensorflow 文档](https://tensorflow.google.cn/deploy/)

[tensorflow 安装](https://tensorflow.google.cn/install/install_windows)

[tensorflow 入门](https://blog.csdn.net/lengguoxing/article/details/78456279)

[CUDA](http://www.stae.com.cn/ch/reader/create_pdf.aspx?file_no=1607470&flag=1&journal_id=jsygc&year_id=2016)

[keras 中文文档](http://keras-cn.readthedocs.io/en/latest/layers/pooling_layer/)

[WIDERFace](http://mmlab.ie.cuhk.edu.hk/projects/WIDERFace/)

[你的颜值打几分？——基于 tensorflow 实现人脸打分模型](https://blog.csdn.net/roguesir/article/details/77164578)

[代码](https://github.com/roguesir/DL-ML-project/tree/master/Face-Scoring)

[华东理工大学人机交互实验室](http://www.hcii-lab.net/data/SCUT-FBP/EN/download.html)

[tensorflow 模型](https://github.com/tensorflow/models)

[基于卷积神经网络（CNN）和 CUDA 加速的实时视频人脸识别](http://www.stae.com.cn/ch/reader/create_pdf.aspx?file_no=1607470&flag=1&journal_id=jsygc&year_id=2016)

[利用 Python、openCV 打造自己的人脸识别 AI 系统
](https://blog.csdn.net/weixin_37554177/article/details/72884682)