**基于openCV和tensorflow的人脸识别系统 设计与实现**

蒋 拓

# 摘 要

近些年来，随着大数据（Big Data）和人工智能(Artificial Intelligence)行业的兴起于广泛应用，计算机图形处理领域有了空前的进展。但由于用户量之大，对人工智能系统的要求也越发严格。而在图像处理领域，主要表现在目标预测准确度和计算耗费的时间与计算机空间资源。为此，需要对图像识别系统进行设计与优化。文中主要研究人脸识别系统的设计与优化，设计了以OpenCV为图片数据采集框架，TensorFlow为神经网络计算框架的人脸识别系统。还运用到了较新的Nvidia CUDA加速技术用来加速神经网络的训练。该系统主要包括了数据采集、标注、卷积神经网络搭建、训练、预测，等几个步骤。其中卷积神经网络包括三层卷积层、三层池化层、两层全连接层。文中设计的系统能够自动检测人脸，保存为一张一张的图片并归纳在不同文件夹下作为数据集，可实现训练集的标注，分割。并且编写了测试脚本用来检测训练出的模型准确度，文中还将对训练耗时和预测每一张照片耗时进行计算（GPU下）。经过检验，4人左右的类别共1000张左右的图片的数据量，预测准确度可达到99.9%以上。因此该文具有一定的应用价值和社会价值。

**关键词**：人脸识别；卷积神经网络；TensorFlow；CUDA；OpenCV

# **Abstract**

In recent years, with the rise of big data and artificial intelligence industry, the field of computer graphics processing has made unprecedented progress. But for many users, the requirements of artificial intelligence systems are more and more stringent. In the field of image processing, we focused on the accuracy of target prediction, the time required for calculation, and computer space resources. Therefore, design and optimization of the image recognition system is needed. Focusing on the design and optimization of the face recognition system, we designed a face recognition system using opencv as a frame for image data collection and a tensorflow as a framework for neural network calculation. The new nvidia cuda acceleration technology is also used to accelerate the training of neural networks. The system mainly includes data collection, labeling, convolutional neural network construction, training, prediction and other steps. A convolutional neural network includes three layers of convolutional layers, three layers of pooling layers and two layers of fully connected layers. The system designed in this paper can detect faces automatically and save them as images, which can be summarized as datasets in different folders, which enables annotation and segmentation of training sets. Then write a test script to test the accuracy of the trained model. After testing, the data volume of about 1000 photos in 4 categories can reach an accuracy of 99.9%. Therefore, it has specific applied value and social value.  
  
**Keywords**: Face Recognition; Convolutional Neural Network; TensorFlow; CUDA; OpenCV

目 录

[摘 要 I](#_Toc40624234)

[**Abstract** III](#_Toc40624235)

[第1章 绪 论 5](#_Toc40624236)

[1.1 课题研究背景 5](#_Toc40624237)

[1.2 国内外研究现状 5](#_Toc40624238)

[1.3 课题研究重点和思路 6](#_Toc40624239)

[第2章 卷积神经网络 7](#_Toc40624240)

[2.1 神经网络 7](#_Toc40624241)

[2.2 卷积 9](#_Toc40624242)

[2.3 卷积和神经网络 11](#_Toc40624243)

[2.3.1多通道图片卷积 11](#_Toc40624244)

[2.3.2 same padding 与 valid padding 12](#_Toc40624245)

[2.4 池化 13](#_Toc40624246)

[2.5 过拟合及其处理方法 14](#_Toc40624247)

[2.6 激活函数的选择 16](#_Toc40624248)

[2.7 损失计算与Cross Entropy 交叉熵 18](#_Toc40624249)

[2.8 Gradient Descent Optimizer和 Adam Optimizer 18](#_Toc40624250)

[第3章 系统实现 20](#_Toc40624251)

[3.1 opencv与haar 20](#_Toc40624252)

[3.2 数据采集和标注 20](#_Toc40624253)

[3.3 CNN网络的构建 23](#_Toc40624254)

[3.4 测试脚本 27](#_Toc40624255)

[3.5 实时检测脚本 28](#_Toc40624256)

[3.6性能评估 30](#_Toc40624257)

[参考文献 31](#_Toc40624258)

# 第1章 绪 论

1.1 课题研究背景

在当今信息化时代，人们对生活便捷程度要求愈发变高。而早在2015年，我国出台了《安全防范视频监控人脸识别系统技术要求》等文件。无疑是在规范计算机视觉领域这个行业的技术发展倾向。五年之后的今天，我们满大街的手机上都可以看到指纹识别，虹膜识别，面部识别等等一系列计算机视觉领域的成功成功应用。但是作为本科生，很少有人愿意去接触这一行业，甚至只愿意钻进传统信息产业的泥坑里。为此，我觉得有必要对计算机视觉领域做较为靠谱的理解，要去了解其中的数学奥秘，也算是给自己打开计算机图形处理的一个小窗。

1.2 国内外研究现状

单说人脸识别撇开人工神经网络，我们得从1960年Bledsoe的工作开始。他发了明了一种可在在RAND平板电脑上工作的人脸识别系统，该系统需要依靠用户手工标注人的面部特征，然后系统根据这种特征来进行识别。实际上这一个系统非常接近今天我们常用的系统了，他的缺点是标注人脸特征需要依靠人手工。

在1980~1990年，通过Sirovich与Kirby的努力，数学的方法被带上人脸识别的舞台,即用线性代数来解决面部识别问题。他们用一些列特征向量来表示一张人脸,即用“特征脸”方法开始寻找用低维度表示面部图像特征的方法。

到2015年，Deep Residual Network在ImageNet测试上，第一次实现超过人类分类能力的计算机算法。

而在中国，由于众所周知的原因，我国计算机事业起步慢，鼓励少。光起步时期就到了上世纪九十年代末。而且还没有任何技术支持，我们一开始得从国外引进这一技术。到2014年之前这一技术的应用基本等于空白，但当年FaceBook发表了Deep Face系统，极大的促进了全球人脸识别技术的发展，我国也乘上了顺风车。加之国家大力推广，到2018年，全国基本无人不晓这一名词。当年多省甚至还用人脸识别技术在高考期间做身份识别一用。可见人脸识别技术在中国的应用之广阔以及市场前进之大

## 1.3 课题研究重点和思路

文章主要研究如何使用Python语言，借助OpenCV库存与TensorFlow框架编写程序，完成一个人脸识别项目所需要的所有功能。并且要用TensorFlow计算框架搭建卷积神经网络，建立具有一定识别能力的人脸检测模型。能够实时的检测和识别人脸，争取做到高准确率，低延迟。

文章思路围绕建立卷积神经网络模型进行代码编写。要建立可靠的模型，首先要有足够的数据。因此我们要使用到OpenCV进行人脸检测，用OpenCV捕捉笔记本电脑（也可以预先提供视频）的摄像头成像，实时的捕捉每一帧，分析其中的人脸，再通过其他逻辑编写代码，归类好数据集。再编写模型训练脚本、测试脚本、和实时检测脚本。由于数据集有上千张，要调整的参数有上兆，我们必须用到GPU辅助加速，这就要用NVIDIA的CUDA技术进行加速训练。

# 第2章 卷积神经网络

## 2.1 神经网络

让我们从最简单的一个例子：《手写数字识别》开始。

最基础的手写数字识别，它的数据集一般选择著名的MNIST中的手写数字图片集合。它是由海量的单通道颜色（即黑白）图片所构成的，每张图片对应一个预测值，这个预测值为一个向量，一般的，这个向量维度为10。若预测值为n，则第（n-1）的维度上的值为1，其他维度向量值为0。即One-Hot向量

MNIST手写数字识别的训练集图片大小为28\*28\*1，即单通道28\*28大小的图片。为了让我们的电脑认识手写数字。我们将一张图片的信息28\*28共784个位置的信息拉伸转换为784\*1维向量。如图：

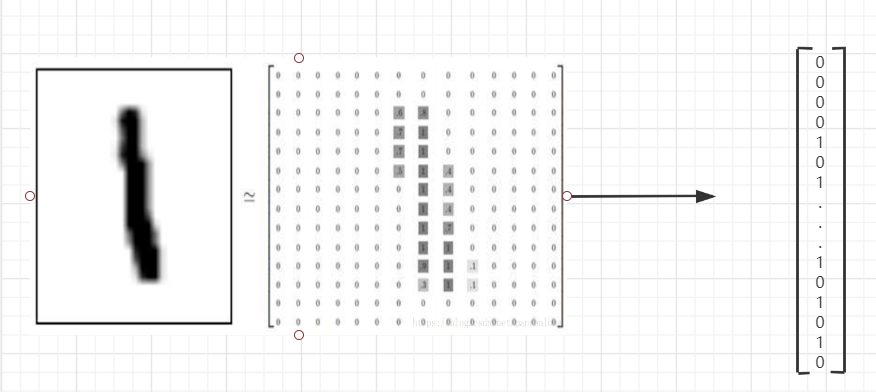


图 2.1单通道图片拉伸示意图

将这784个数当成输入，我们将其变成行向量，认为他是1\*784大小的向量。设有为第一层网络的权重，其大小为，为第一层网络的偏置矩阵，他的维度应该为，但是基于python的广播机制，我们可以直接认为他的维度是out[1]。

那么到这里你肯定要问第一层神经网络是什么。简单的来说，可以将人工神经网络理解为众多神经元的集合（有序），他们按一定的方式组合起来。这些神经元里面所含有的东西就是一个数，这些数其实就是我们刚刚所说的权重矩阵中的数（事实上应该是两个神经元之间的连线），一般的都为浮点数据。

我们将这些一层的网络里面的权重当作参数，设最终得到的结果为，则我们认为，其中可以认为是某种规则，通过对权重矩阵和偏置矩阵的某些固定运算规则能表达出。但是很明显的，这784个参数加上偏置矩阵的参数并不足以体现一张手写数字识别图片的所有特征。为了解决这个问题，我们可以继续添加第二层网络：设第层网络的输出为，则第层网络为。其中和为第层的权重矩阵和偏置矩阵，且矩阵列向量维度必为的行向量维度。

但是单纯的数学计算并不足以表示人类脑中的神经元工作，因此，引入激活函数（activation function）这个概念。所谓的激活函数，就是把每一层神经神经网络的输出结果当作输入，引入规则，设第层神经网络的输出为，那么事实上，第层神经网络应该为，其中为固定运算规则，其作用是将中的所有元素(神经元)都做一次非线性运算，例如函数：

图片包含 游戏机, 物体

描述已自动生成

图 2.2 ReLU函数图像

对于函数，也称其为线性修正线性函数，它是现在人工神经网络中常用的激活函数（以前多用sigmoid与tanh，ReLU包括ReLU和它衍生的变种函数。）。如假设为的运算规则，那么经过处理过的第层神经网络的输出应该为，其中是第层输出，是第层神经网络的权重矩阵，为第层神经网络的偏置矩阵。而函数为a，b两数之间的最大数值。

由此递归，构建具有n层的神经网络，第n+1层即为预测层（整个网络的输出），他是一个10\*1维的One-Hot向量。我们通过调整这n层神经网络的权重和偏置，最终可以得到类似一个如图的神经网络：

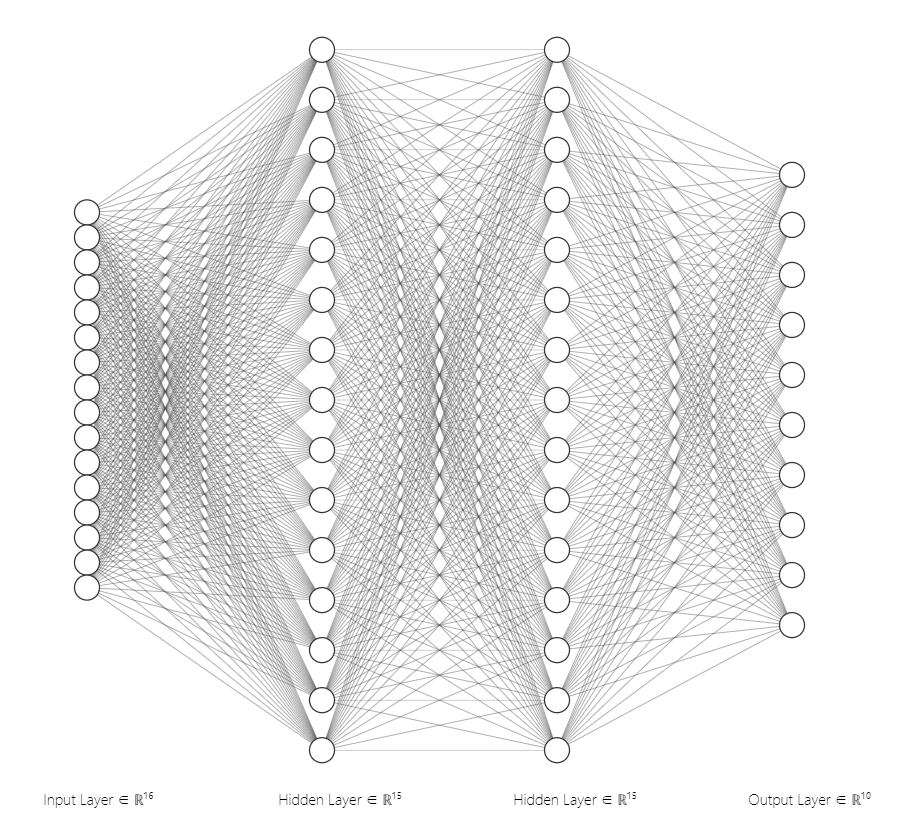


图 2.3 神经网络示意图

当然输入层应该有784个，上图只表示该网络可视化后的大概状况。

## 2.2 卷积

要理解卷积(Convolution)，我们要从另一个例子《边缘检测》入手。

假设有一张的单通道图：

(2.1)

每个像素点上数值越大，表示这个位置颜色越深。设有大小为的矩阵：

(2.2)

我们的目标是要检测到单通道图上面的竖直黑白边缘（改图中竖直边缘为中切线）。为了达到这个目的，我们进行如下操作：将矩阵与矩阵左上角9个元素重合，再将重合的元素相乘，例如与相乘，与相乘，由此得到的个相乘后的数相加起来得到，其中为一个的矩阵。再将往右移动一个单位，做同样的运算，得到，由此往复填满整个矩阵。经过计算后我们得到：

(2.3)

如果我们把矩阵B当成图像的话，那么其中间就会有一条很亮的线条。我们认为这个线条就是我们所需要找的边界。但是我们可能认为这个边界有点粗，其实只是因为我们原始图片大小不够，如果是很大的图片，比如100\*100的图片，那么这个边界可以说寻找的非常准确。

很明显对于矩阵P来说，我们只用了一种数字组合来做边缘检测，那么对应的我们可以用其他数字组合，让他效果更好，历史上计算机视觉论文曾经就争论过哪种数字组合是最好的。比如著名的Sobel算子（或滤波算子，过滤器，filter）：

(2.4)

和著名的Scharr算子：

(2.5)

不过当然这些算子都是用于垂直边缘检测，若想进行水平边缘检测只需要将他们旋转90度就可以了。但是在计算机视觉领域里，我们有另外一种思维：将这个矩阵*P*看作是带有9个参数的矩阵，其中这九个参数可以任意设置，即：

(2.6)

我们可以通过调整这个矩阵中的参数来让他实现某些功能，对于图片来说就相当于让计算机了解矩阵所对应图片的某个特征（或提取/抽象某个特征，）。那么通常我们将这种操作成为相关（Cross-Correlation），设A为单通道图片对应的矩阵，F为过滤器，为用F对A作相关性运算之后的结果，则有公式：

(2.7)

其中k表示矩阵F的大小（设长宽都为k），m、n为矩阵C的第m行n列上的元素。

而我们说的卷积运算，与相关性运算非常相似，其公式为：

(2.8)

其中CONV（A，F）表示对A，F做卷积运算的结果。可能乍一看与相关性运算非常相似，我们可以先认为其两公式中的*F*不为同一个矩阵。设有，即用对A作相关性运算的结果和用*F*对A作卷积运算的结果一样，则必有，其中表示对F中的元素位置作180°旋转。

在文中我们的卷积概念与概率论与数理统计中的卷积公式有所不同，由于文中描述的项目特殊性，除特殊说明以外一般认为卷积操作与相关性操作等同。

## 2.3 卷积和神经网络

### 2.3.1多通道图片卷积

上节中，我们已经了解单通道矩阵图片的卷积操作。现在就让我们来尝试对于RGB通道的卷积操作。设有大小为6\*6\*3的矩阵A，我们使用大小为3\*3\*3的矩阵f对其进行卷积。在矩阵A的每个颜色通道（以后称通道）上，矩阵f都有与之对应的一个通道，对其进行卷积，其计算方式与单通道图片类似。设第i层卷积结果为b[i]，则3通道图片A卷积后每个位置上的元素为

(2.9)

其中B为大小4\*4的矩阵。如图所示：

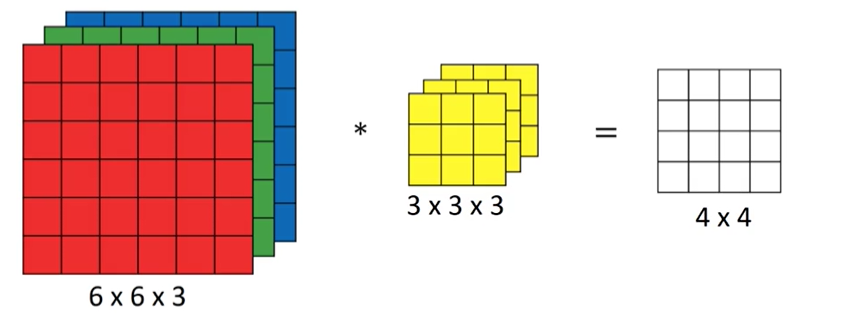


图 2.4 多通道卷积示意图

在本文中，训练数据保证是长宽大于64的R,G,B三通道图片。对于公式（2.9）可能没有实际的印象，事实上对于多通道矩阵卷积，我们只需用相同通道数量的卷积核f，在每一个通道上对其分别卷积，再将得到的三个矩阵对应位置上的元素相加就可以了。

### 2.3.2 same padding 与 valid padding

如上小结所述，我们初步了解了单通道图片与多通道图片卷积的基本要义。现在让我们来研究一个问题：设有大小6\*6的矩阵A和3\*3的矩阵f，将矩阵A与矩阵f作卷积运算。我们得到矩阵B，其大小为4\*4。很明显，如果让计算机继续去完成图像识别任务，B矩阵已经丢失了A矩阵中部分原有的信息，这对于图像识别任务来说无疑是不可取的。我们从这个角度来思考原因：由于卷积运算的特殊性，对于A矩阵来说，为了第一次卷积，使A的左上角与f的左上角重合进行卷积运算，再往右滑动进行卷积运算，设矩阵A大小为，f大小为，一次完整的卷积运算下来我们得到矩阵B，其大小为，如上述边缘检测例子：

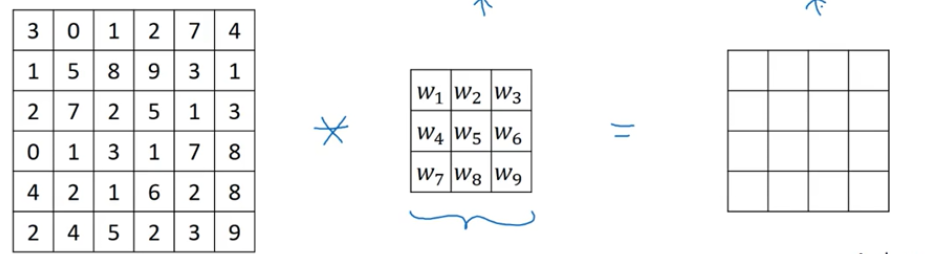


图 2.5 边缘检测样例图

位于左上角的第一个元素在整个卷积过程中只在第一次中参与卷积，而其他位置的元素例如4行4列的元素在卷积过程中被多次运用。如图：

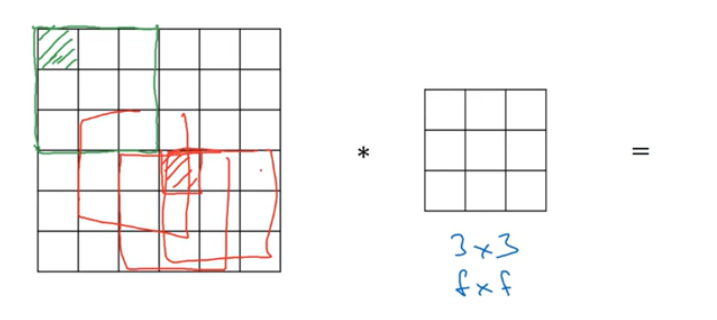


图 2.6 卷积过程中边缘像素未被充分利用

事实上，对于矩阵A上所有边缘上的元素，在整个卷积过程中都只参与了一次。这样我们计算机对图像进行计算时候，有一部分的元素没有被充分利用，这也将会导致计算机对图形理解的缺失。

为了解决这个问题，引入padding概念。Padding即在卷积之前，用部分元素，填充被积矩阵（一般在边界上增加随机元素），其填充元素数据根据实际而变化，一般选择0。

Padding分为same padding 和 valid padding，即同等填充和有效填充。在本文中，我们认为有效填充就是不做任何的元素填充，即直接用原图对应矩阵来作卷积，而同等填充则有所不同。所谓同等填充，就是一种使矩阵卷积后得到的矩阵与原矩阵大小相同的填充方式。对于上述矩阵A，我们可以在其周围填充一圈的随机数字，使其大小变成8\*8，根据上节所提供的公式，可以得出其卷积后的结果为（8-3+1）\*（8-3+1），也就是6\*6，其大小与原矩阵大小相同，如图：

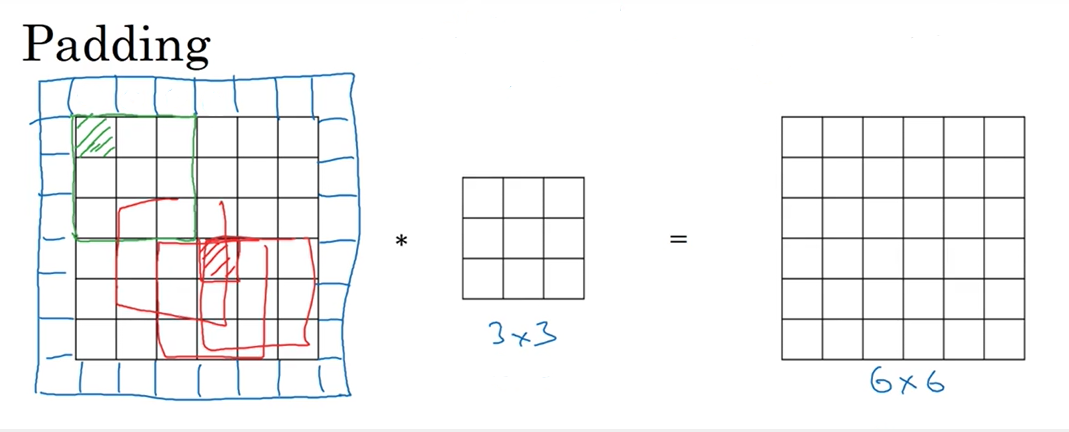


图 2.7 SAME PADDING示意图

在本文中，我们使用同等填充作为积分前的操作。

## 2.4 池化

我们已经了解了多通道图片卷积和卷积前padding的两种方式。为了轻量化和减少训练时的参数，优化模型训练速度，我们要对他进行特征提取和参数缩减[2]。为此，我们引入pooling（池化）概念。

与卷积类似，池化中也设有类似卷积核f的池化核p。设有4\*4大小的矩阵B，使用大小为2\*2的矩阵p对其进行池化操作，若池化步长（strides）为2，在p所覆盖的矩阵B的区域上，每次池化后滑动的距离为2个单位（包括垂直方向和水平方向）。池化操作有两种，一种是max pooling，即每次在p所覆盖的矩阵位置上提取其中的最大值。我们认为这个最大值能代表该次池化的区域内矩阵在该位置的特征。例如：

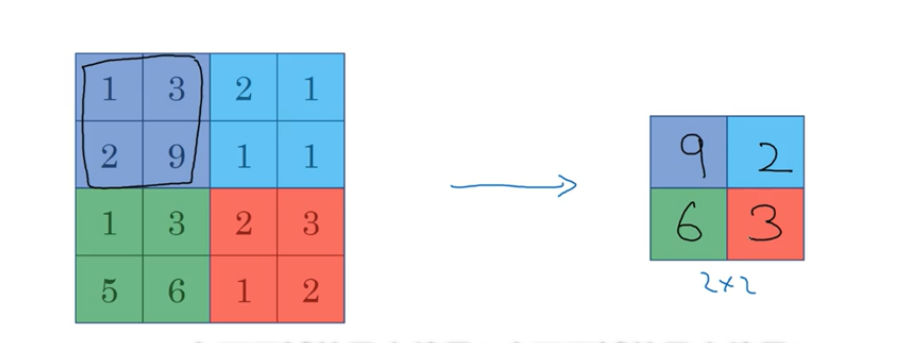


图 2.8 max pooling 原理图

另外一种则为average pooling，即在每个p所覆盖的位置上提取所有元素数值的平均数，例如：

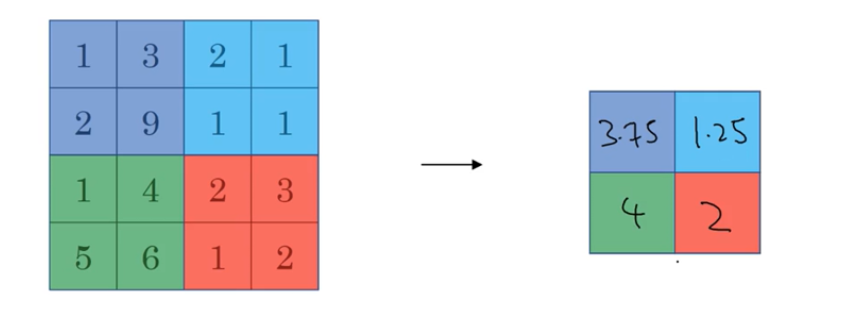


图 2.9 average pooling 原理图

目前来说，最大池化比平均池化更常用。但也有例外，比如深度很深的神经网络，我们可以用平均池化来分解规模为7\*7\*1000的矩阵，在整个空间内求平均值，得到1\*1\*1000的矩阵。池化的参数一般包括池化核p的大小f，和池化步长s。一般将f和s设置为2，2。其直观效果相当于被池化矩阵高度和宽度缩小一半。像卷积一样，池化也可以设置padding。也有valid和same 两种padding设置方案。但是我们一般不在池化的时候设置padding。若有需要，我们一般不随机设置padding值，而是直接将其置0。设最大池化中，被池化矩阵大小为n\*n\*h，池化核大小为f\*f\*h（上文可得出两矩阵h必相等的结论），则其池化后输出大小为：

(2.10)

并且向下取整。

## 2.5 过拟合及其处理方法

过拟合（over-fit）也称为过学习，它的直观表现是训练出来的模型在测试集上表现好，但是模型实际使用时表现欠缺，实际预测或分类效果很差。大型的人工神经网络中，过拟合的现象非常常见（大型的神经网络中参数可达数亿）。抽象的来说。大量的参数让神经网络记住更多不必要的细节，实际上我们训练出来的网络应该要去学习具有更加鲁棒性(可以理解为更加具有特性的，例如忽略光线外，发色外)的特征。如果参数过多，很可能导致模型的泛化效果差。

例如对于二分类来说，事实上我们需要的曲线是一条光滑的，只需要能较好的分开两个类别即可，这样既可以加快训练速度，也能减少过拟合现象。如果模型参数设置太多，训练出来的曲线将严格划分两个类别的对象（指丝毫不差），直观上来说，这样会导致我们训练出来的是一条非常不光滑的曲线。如下图：

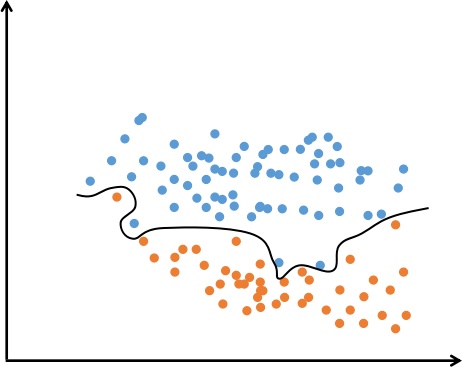


图 2.10 过拟合示意图

为了解决过拟合问题，我们引入dropout概念。dropout的做法是在训练时随机的选择一部分神经元进行正向传播和反向传播（即BP），另外一些神经元的参数值保持不变，以减轻过拟合[3]。一般情况下，越复杂的模型越容易陷入过拟合[4]。Dropout使得每个神经元在训练过程中没有用到所有的特征值检测器的数据，而是随机抽取一部分。人工智能之父Hinton在其2012年发表的《Improving neural networks by preventing co-adaptation of feature detectors》中认为大型神经网络对小型数据集进行训练的时候容易造成过拟合，而通过隐藏部分（选择性忽略）神经元，减少其共同作用可以提高整个神经网络的作用[5]。其示意图如下：

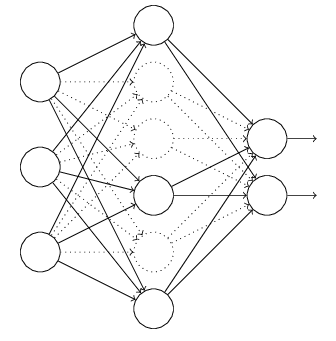


图 2.11 dropout 示意图

对于上图所示的网络，在训练开始时，可以随机删除一些神经元，一般取0.5，0.75。即认为这些神经元不存在，同时保持输入层与输出层神经元的个数不变。

吴恩达（Andrew Ng）认为：对于单个神经元来说，它的工作就是接收很多输入再生成有作用的输出，且这个输出依赖于他对每个输入的权重值。假设我们将他的输入随机撤销掉一些，他就不敢直接依赖于某些特定的权重。这将导致它会更加均匀的把权重分配给各个输入。

举个例子来说，种类A的样本光线较为充足，种类B的光线较暗，那么神经网络就会直接使用光线的亮度对数据进行分类。而到了实际应用中，如果种类A的光线不再充足，它就会被辨认为种类B。

也就是说我们的网络应该去学习在多种类别中具有普遍性的特征，它不应该去学习某些类别特定的特征，这样才能提高整个神经网络的鲁棒性[6]（健壮性）

## 2.6 激活函数的选择

在本章第一节曾经介绍过神经网络中的一个最基础的公式，从数学角度上来说，每上一层的输出将作为下一层的输入。最后一层为整个神经网络的输出，第一层的输入为训练样本的数据。不但如此，在大型的神经网络中，为了更好的模拟人脑的工作状态（本人非生物学专业，无法验证这一说法），在每两层之间加入激活函数（一般为非线性函数），每上层网络的输出经过激活函数才能充当下一层网络的输入。

在早期的神经网络研究和训练过程中，多半为sigmoid函数和 tanh函数。其均为非线性函数，其在早期的网络中表现较好，但是如sigmoid函数，在海量的测试中表明它在训练过程中会出现梯度消失和梯度爆炸等问题。

所谓梯度消失，即当网络训练到一定程度时，神经网络的权重更新缓慢，甚至无法更新的问题。而梯度爆炸是指在训练过程中，梯度呈指数形的增长，导致学习率更新非常困难或者或者相对应的它要求学习率非常的低。

对于Sigmoid函数：

(2.11)

其图像：

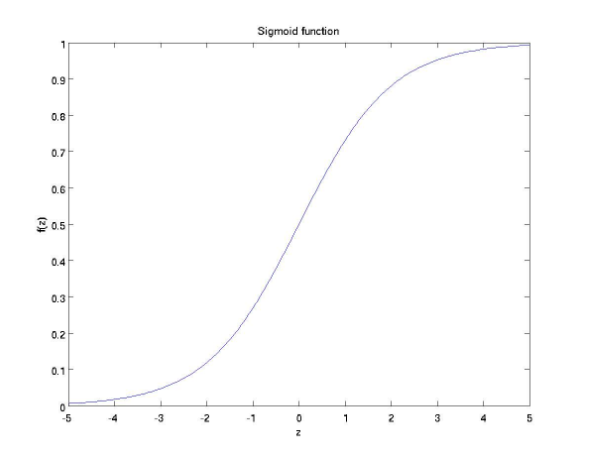


图 2.12 sigmoid函数图像

对于tanh函数：

(2.12)

其图像：

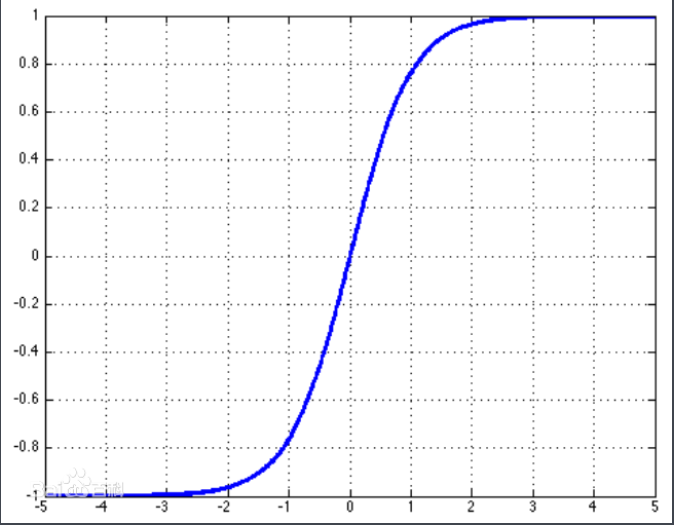


图 2.13 tanh(双曲正切)函数图像(注意零点位置)

对于在上文中提到的ReLU函数我们在本章第一节已经简述过。此处不做赘述。由于ReLU函数的仿生物学原理比Sigmoid与tanh更好，加上它ReLU能很好的避免梯度爆炸和梯度消失，结合本文描述的人脸识别项目，本文选择ReLU作为激活函数。

## 2.7 损失计算与Cross Entropy 交叉熵

对于人工神经网络来说，其设计之初的一个理念是为了让计算机如同人脑一样学习和思考。这就要求我们建立一个可靠的神经网络模型给计算机使用。而在一个神经网络训练过程中，无疑就是不断地减少训练值和真实值之间的差距。为此，我们需要合理计算训练过程中的损失。

损失计算中最常见到的一种计算方式为均方误差(mean squared error)。顾名思义，其原理就是计算目标值和真实值之间的平方差的平均值。其公式为：

(2.13)

其中为输出，*f*表示固定运算，本文表示卷积神经网络的输出，表示正确的预测值（真实值）。

熵是一种对不确定问题的度量准则[7]，即对问题混乱程度的度量准则。Cross Entropy 交叉熵也可以在神经网络中作为损失函数。如使用的激活函数为sigmoid，交叉熵能避免如均方差一样在用梯度下降方法进行训练时，初期训练速度过慢的问题。交叉熵对于离散变量采用以下的方式计算：

(2.14)

由于在分类任务中，用均方差作为损失函数会导致某一些数据在一开始训练的时候速度非常慢，而交叉熵能很好的解决这个问题，所以文中的损失函数使用交叉熵。

## 2.8 Gradient Descent Optimizer和 Adam Optimizer

由于神经网络的训练过程就是一个不断减少预测值和真实值之间的差距，上文已经选定好了损失函数的计算方式为Cross Entropy，本节介绍Gradient Descent Optimizer[9]（梯度下降优化器）和 Adam Optimizer（自适应矩估计）。

首先我们来介绍梯度下降法。对多元函数的各个自变量求偏导，将求出来的偏导数以向量的方式写出，称之为梯度。设有函数，我们分别对x，y求偏导，得出 ∂f/∂x 与 ∂f/∂y。那么对于函数在处，其梯度则为：

(2.15)

而从几何角度来观察，函数在某一点处的梯度所指的方向就是在这一点处函数值增长最快的方向。反过来说，函数在该点处梯度所对应的反方向就是在这一点处函数值下降的最快的方向。例如对于上述梯度公式的和点，就是函数在减少或者下降的最快方向。

Adam Optimizer则为梯度下降法的一种变形，或者说是优化。Adam即为adaptive moment estimation。它主要的不同是利用了梯度的一阶矩和二阶矩对学习率进行动态调整以优化训练效果。

鉴于本次项目的数据量大小，为了提高训练速度，文中的优化器将选择Adam Optimizer。

# 第3章 系统实现

## 3.1 opencv与haar

OpenCV（开源计算机视觉库）是一个开源计算机视觉和机器学习软件库。 OpenCV的构建旨在为计算机视觉应用程序提供通用的基础结构，并加速在商业产品中使用机器感知。

而Haar(哈尔)[10]特征值有点类似我们上一章所说的滤波算子（filter），也是由很多不同的算子构成的特征，使用的时候类似卷积操作一样滑动的对矩阵计算一遍。滑动窗口[8]走到一个位置就会计算出这个区域内所有的特征，接着再用已经训练好的级联分类器[11]（Cascade Classifier）进行特征提取。与卷积和相关性操作不一样的是，Haar特征在OpenCV里面使用的时候有个参数叫缩放倍数。在级联分类器提取特征的时候我们都会先对图像作缩放，然后滑动窗口再对图片进行多次计算。这个缩放倍数由开发者控制。通过控制缩放倍数等参数，可以提高目标检测准确性，文中描述的系统采用OpenCV的级联分类器来检测目标。

## 3.2 数据采集和标注

人工神经网络的训练需要大量的人脸数据，为此我们使用OpenCV，结合haar特征值和adaboost级联分类器来检测目标。首先是对摄像头和源视频的图像截取，我们按一帧一帧截取所给出的视频源或者读取本地摄像头的数据。在预设的图像保存目录按顺序保存不同用户的人脸数据。在实时检测时，将检测到的人脸目标划上矩阵边缘以提醒用户检测到的目标，按“q”键可退出人脸采集。其实现如下：

import cv2

import time

import os

PATH\_PREFFIX = "faces/"

PATH\_SUFFIX = "\_img"

PATH\_LABLE = 0

PATH = PATH\_PREFFIX+str(PATH\_LABLE)+PATH\_SUFFIX

classifer = cv2.CascadeClassifier("haarcascade\_frontalface\_default.xml")

while True:

if os.path.exists(PATH\_PREFFIX+str(PATH\_LABLE)+PATH\_SUFFIX):

PATH\_LABLE+=1

else:

PATH = PATH\_PREFFIX+str(PATH\_LABLE)+PATH\_SUFFIX

os.makedirs(PATH)

print("path:=======",PATH)

break

# if not os.path.exists(SAVE\_PATH):

# os.mkdir(SAVE\_PATH)

cap = cv2.VideoCapture(0)

tag = 0

while(cap.isOpened() and cv2.waitKey(2)!=ord("q")):

(flag,frame) = cap.read()

frame\_gray = cv2.cvtColor(frame,cv2.COLOR\_RGB2GRAY)

faces = classifer.detectMultiScale(frame\_gray,1.15,6,minSize=(64,64))

if len(faces) == 1:

(x,y,w,h) = faces[0]

face = frame[y:y+h,x:x+w]

cv2.imwrite(PATH+"/img\_%02d.jpg"%tag,face)

cv2.rectangle(frame,(x,y),(x+w,y+h),(255,0,0),2)

tag = tag+1

print(tag)

cv2.imshow("camera",frame)

cap.release()

其中PATH\_PREFFIX常量为数据存储目录前缀，依据用户需求可以更改。PATH\_SUFFIX为不同用户人脸保存目录的后缀，用于统一标识文件夹。比如在本源码设置下，我们进行捕捉会时在faces文件夹下有“0\_img”和“1\_img”文件夹，这说明这两个文件夹中的数据来自不同的用户并且按顺序存储。要注意每次捕捉都会生成新的文件夹，即使本次收集人脸没有得到数据也会生成一个文件夹，只是这个文件夹是空的。所以在训练的时候一定要确认无空文件夹，否则直接报错使脚本退出。该脚本将捕捉的到图像先转换为灰度图片，再通过OpenCV的detectMultiScale接口（因为只OpenCV为了提高检测速度都需要将图片转换为单通道的灰度图片），检测出一张图片人脸位置和长款。得到位置和长宽后，到原图上面截取对应的RGB图片按顺序命名好并存储到对应的文件夹。值得注意的是detectMultiScale中的scaleFactor和minNeighbors要根据用户使用的环境进行修改，其分别代表缩放因子和构成检测目标的相邻矩形的最小个数。采集效果如图：

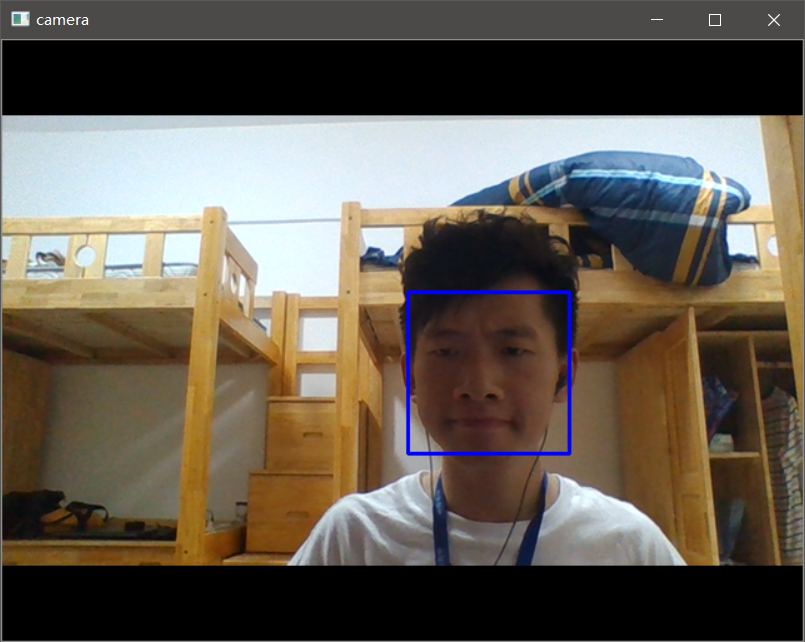


图 3.1 人脸采集效果图

在控制台可看收集到的图片数量和保存的路径：

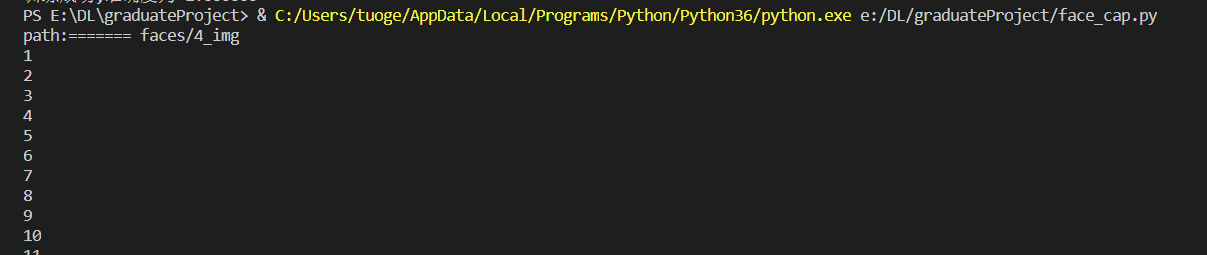


图 3.2 人脸采集控制台输出信息

再到资源管理器中查看对应的图像资源（部分截取）：

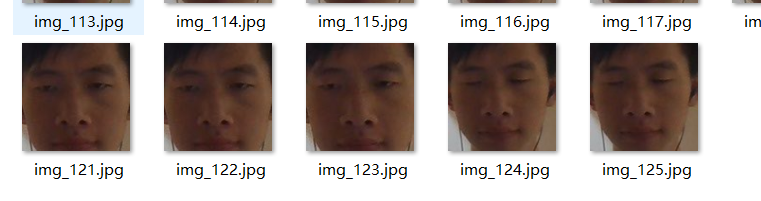


图 3.3 人脸资源部分截取

## 3.3 CNN网络的构建

收集完人脸数据之后我们要做的就是训练网络了。本文设计了三层卷积层，每层卷积后加有池化层并对其进行dropout，激活函数选择ReLU。三层卷积层的卷积核大小均为3\*3。第一层卷积层输入通道为3（RGB），输出通道为32；第二层卷积层输入通道为32（上层输出），输出通道为64；第三层卷积层输入通道为64，输出通道为128。接着是两层全连接层，第一层全连接层输入大小为8192，输出大小为512；第二层输入大小为512，输出大小为1024。最后一层的大小依据用户数量自动判断。其可视化后效果如下：

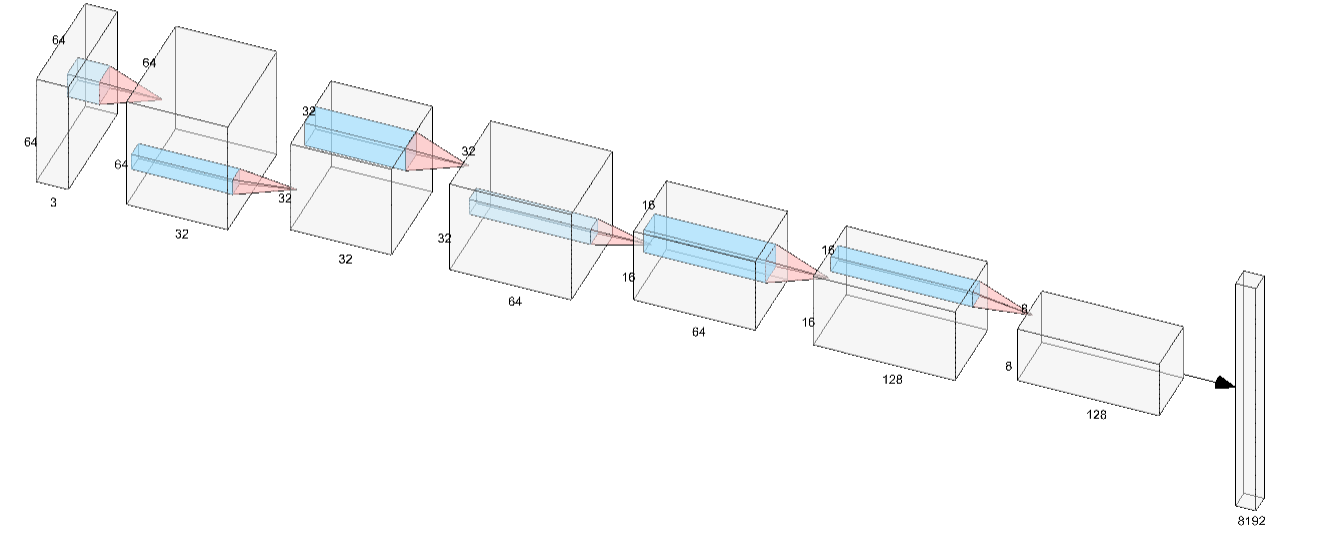


图 3.4 卷积层架构图

其中由于高度的问题未画出两层全连接层。参考上一章节的《手写数字识别》案例配图。现在编写train.py脚本，其目的是为了读取数据，标注数据，训练和保存网络。其基本内容如下：

import cv2

import tensorflow as tf

import os

import numpy as np

import random

import time

from sklearn.model\_selection import train\_test\_split

PATH\_PREFFIX = "./faces/"

MODEL\_PATH = "./model/"

imgs = []

labels = []

size = 64 #规范图片长度

out\_size = None

batch\_size = 100

batch\_num = None

CONV\_KEEP\_1 = 1

CONV\_KEEP\_2 = 1

CONV\_KEEP\_3 = 1

FC1\_OUT\_KEEP = 1

FC2\_OUT\_KEEP = 0.8

def read\_data():

if not os.path.exists(PATH\_PREFFIX):

print("img path not found")

exit()

else:

len\_ = len(os.listdir(PATH\_PREFFIX))

global out\_size

out\_size = len\_

for i in os.listdir(PATH\_PREFFIX):

print("found path %s"%i)

cur\_index = int(i.split("\_")[0])

y\_ = np.zeros([len\_])

y\_[cur\_index]=1

print(y\_)

for j in os.listdir(PATH\_PREFFIX+i):

path = PATH\_PREFFIX+i+"/"+j

print("reading %s"%path)

imgs.append(cv2.resize(cv2.imread(path,1),(size,size)))

labels.append(y\_)

def weight\_var(shape):

# w = tf.random\_normal(shape,stddev=0.1) #标准差0.1

w = tf.truncated\_normal(shape,stddev=0.1)

return tf.Variable(w)

def bias\_var(shape):

b = tf.random\_normal(shape) #正态分布取值

return tf.Variable(b)

def conv2d(x,w):

#1 4位置固定数据1 2是水平方向抽取数据跨度 3是竖直方向跨度

return tf.nn.conv2d(x,w,strides=[1,1,1,1],padding="SAME")

def max\_pool(x):

return tf.nn.max\_pool(x,ksize=[1,2,2,1],strides=[1,2,2,1],padding="SAME")

def drop\_out(x,keep):

return tf.nn.dropout(x,keep) #drop out 放弃一些权重的变化，避免over fitting

def add\_cnn\_layer ():

# 第一层卷积 核心大小3x3 输入通道3(RGB) 输出 通道32

w1 = weight\_var([3,3,3,32])

b1 = bias\_var([32])

conv1 = tf.nn.relu(conv2d(x\_holder,w1)+b1)

# 第一层池化

pool\_1 = max\_pool(conv1)

# 第一层 drop out

out1 = drop\_out(pool\_1,CONV\_KEEP\_1)

# 第二层卷积

w2 = weight\_var([3,3,32,64])

b2 = bias\_var([64])

conv2 = tf.nn.relu(conv2d(out1,w2)+b2)

pool\_2 = max\_pool(conv2)

out2 = drop\_out(pool\_2,CONV\_KEEP\_2)

# 第三层卷积

w3 = weight\_var([3,3,64,128])

b3 = bias\_var([128])

conv3 = tf.nn.relu(conv2d(out2,w3)+b3)

pool\_3 = max\_pool(conv3)

out3 = drop\_out(pool\_3,CONV\_KEEP\_3)

# 全连接 1

wf1 = weight\_var([8\*8\*128,512])

bf1 = bias\_var([512])

out3\_flat = tf.reshape(out3,[-1,8\*8\*128])

flatw\_plus\_b = tf.nn.relu(tf.matmul(out3\_flat,wf1)+bf1)

f1out = drop\_out(flatw\_plus\_b,FC1\_OUT\_KEEP)

# 全连接 2

wf2 = weight\_var([512,1024])

bf2 = bias\_var([1024])

f1out\_mut\_wf2\_plus\_bf2 = tf.nn.relu(tf.matmul(f1out,wf2)+bf2)

f2out = drop\_out(f1out\_mut\_wf2\_plus\_bf2,FC2\_OUT\_KEEP)

# 输出

w\_out = weight\_var([1024,out\_size])

b\_out = bias\_var([out\_size])

out = tf.add(tf.matmul(f2out,w\_out),b\_out)

return out

def get\_acc():

return sess.run(acc,feed\_dict={x\_holder:test\_x,y\_holder:test\_y})

def save\_model(sess):

if not os.path.exists(MODEL\_PATH):

os.mkdir(MODEL\_PATH)

time\_str = time.strftime("%Y\_%m\_%d\_%H\_%M\_%S", time.localtime())

saver.save(sess,MODEL\_PATH+"face\_model\_"+time\_str+".model")

read\_data()

print("total img count %s"%len(imgs))

print("total label count %s"%len(labels))

imgs = np.array(imgs)

labels = np.array(labels)

train\_x,test\_x,train\_y,test\_y=train\_test\_split(imgs,labels,test\_size=0.2,random\_state=random.randint(0,100),stratify = labels)

#为卷积设定维度

train\_x = train\_x.reshape(train\_x.shape[0], size, size, 3)

test\_x = test\_x.reshape(test\_x.shape[0], size, size, 3)

batch\_num = train\_x.shape[0] // batch\_size

#RGB小于256 转换为 0~1的浮点，归一化

train\_x = train\_x.astype("float32")/255.0

test\_x = test\_x.astype("float32")/255.0

print("leng test img : %d"%len(test\_y))

print("leng train img : %d"%len(train\_y))

print("out size: %d"%out\_size)

#定义 holder

x\_holder = tf.placeholder(tf.float32,[None,size,size,3])

y\_holder = tf.placeholder(tf.float32,[None,out\_size])

pred = add\_cnn\_layer()

#损失计算

cross\_entropy = tf.reduce\_mean(tf.nn.softmax\_cross\_entropy\_with\_logits\_v2(logits=pred,labels=y\_holder))

# cross\_entropy = tf.reduce\_mean(-tf.reduce\_sum(y\_holder\*tf.log(pred),axis=1))

#训练

train\_step = tf.train.AdamOptimizer(0.01).minimize(cross\_entropy)

#准确度

acc = tf.reduce\_mean(tf.cast(tf.equal(tf.argmax(pred,1),tf.argmax(y\_holder,1)),tf.float32))

saver = tf.train.Saver()

init = tf.global\_variables\_initializer()

print("数据堆个数%d"%batch\_num)

with tf.Session() as sess:

sess.run(init)

for i in range(100):

for j in range(batch\_num):

X = train\_x[j\*batch\_size:(j+1)\*batch\_size]

Y = train\_y[j\*batch\_size:(j+1)\*batch\_size]

sess.run(train\_step,feed\_dict={x\_holder:X,y\_holder:Y})

print("第%d次训练，准确率：%f"%(i,get\_acc()))

if get\_acc()<0.9:

print("准确度小于0.9,训练失败")

else :

save\_model(sess)

print("训练成功,准确度为 %f"%get\_acc())

其中CONV\_OUT\_KEEP(1,2,3),FC1\_OUT\_KEEP，FC2\_OUT\_KEEP应该根据用户使用环境进行调节，以达到最优效果和减少拟合。影响此参数的条件主要为用户使用时环境的光线。要注意开发使用的tensorflow为1.80版本，在高版本中有部分方法已经弃用或者更改名字，为了减少训练参数和加快训练速度，我们读入faces文件夹下的图象时都将其归并为64\*64大小。事实上，为了提高精准度，一般不对卷积层进行dropout，所以本文卷积层dropout的概率为1，即不丢弃任何神经元。我们还可以通过增加卷积层和全连接层的数目。至于保存模型的问题，实际上我们保存的就是卷积层和全连接层的权重和偏置，其存储在Tensorflow.Session()返回的对象内，通过Tensorflow.Session().save()方法即可保存当前训练（调整）出来的权重与偏置。其目录由MODEL\_PATH指定。模型训练正确度未达到90%认为训练失败，不保存session。

运行train.py，将输出GPU（需要安装CUDA驱动以及Cudnn库，篇幅问题本文不做介绍）型号，可用显存，时钟频率。训练过程中可看到每一次训练后的正确率。训练后可看到耗时（耗时为自己计算非tensorflow功能）。

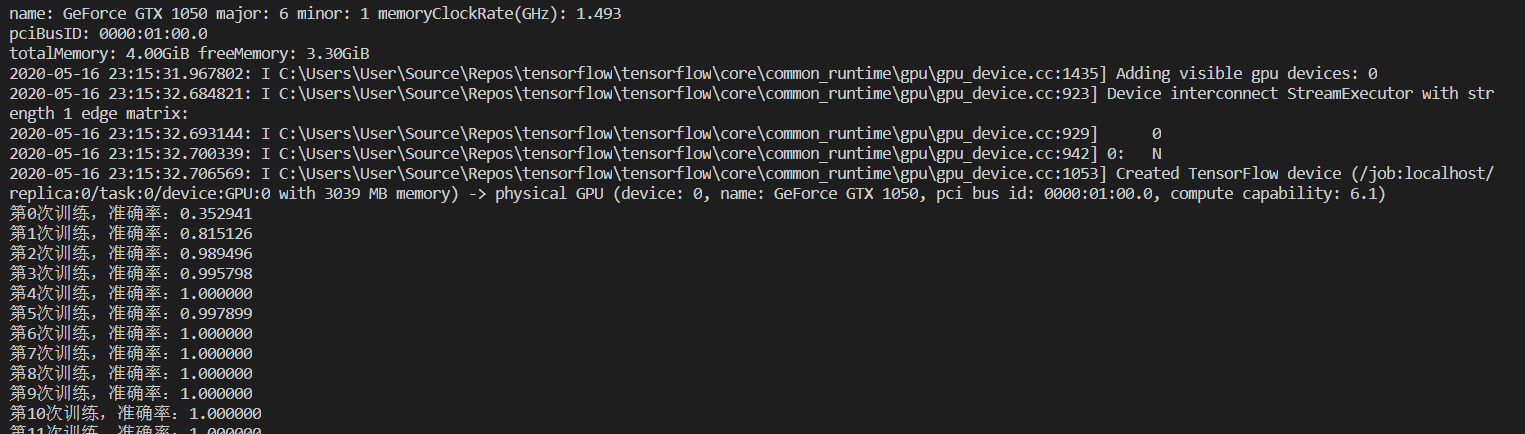


图 3.5 GPU信息图

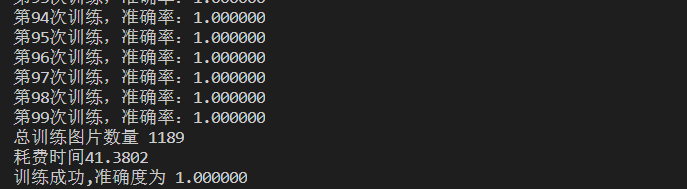


图 3.6 训练结果和耗时

检查当前目录下的model路径应该按训练完成的时间保存了checkpoint，data，index，meta文件。其中checkpoint为检查点，MODEL\_PATH可能有多个模型组合，但检查点只记录最后一次训练的结果。DATA-00000-OF-00001与INDEX文件共同组ckpt文件。ckpt文件存储我们所搭建的网络中的权重和偏置。Meta文件则保存的网络结构图已经定义在张量上的操作。

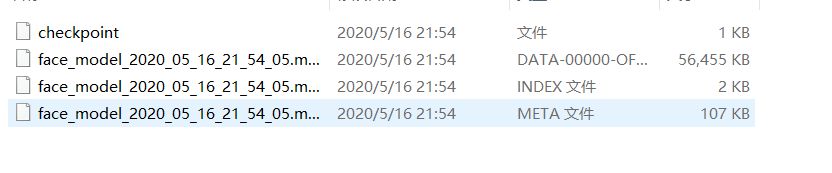


图 3.7 模型文件保存示意图

## 3.4 测试脚本

训练模型后为了测试训练成果，我们必须编写测试脚本。文中设计的训练脚本不需要指定参数，由python产生随机数用来选定某一个用户的脸部图片目录，接着对该用户的脸部图片随机检测1000次，如有一次预测值与真实值不一致，即认为测试失败。此脚本同样也需要先构建和train.py参数一致的卷积神经网络，再通过tf.train.saver()来恢复上次训练的权重。其核心内容如下：

pred = tf.argmax(out,1)

saver = tf.train.Saver()

sess = tf.Session()

saver.restore(sess,tf.train.latest\_checkpoint(MODEL\_PATH))

path\_no = random.randint(0,out\_size-1)

faces\_path = PATH\_PREFFIX+os.listdir(PATH\_PREFFIX)[path\_no]

print("随机读取: %s %s"%(faces\_path,names[path\_no]))

imgs = []

for i in os.listdir(faces\_path):

imgs.append(cv2.imread(faces\_path+"/"+i))

flag = 0

for i in range(1000):

img\_no = random.randint(0,len(imgs)-1)

img = imgs[img\_no]

name = recog\_face(img)

print("选中%s"%os.listdir(faces\_path)[img\_no])

print("识别 %s"%name)

if not name == names[path\_no]:

flag+=1

if flag != 0 :

print("测试失败,%d样例未通过"%flag)

else :

print("测试成功")

其中add\_cnn\_layer与train.py中设计一致，主要是用python对其进行图片目录随机抽取和预测。为此，我们的CONV\_KEEP参数与FC1\_OUT\_KEEP参数应该全部置1，抽象的说，这是让计算机看的更清楚图片。而在训练的时候是让计算机减少对图片细节的抽取。如果训练数据是片边缘有突刺的树叶，若训练时dropout的参数为1，则计算机将认为边缘没有突刺的树叶不是树叶。这其实就是过拟合问题。但是在测试和实际使用中，我们就是应该让计算机看到这些细节，给计算机提供更多的特征，因此在实际使用模型时这些KEEP参数都应该置1。

## 3.5 实时检测脚本

实时检测功能实际上应该是对于用户来说接触最直接的东西，其基本思路为1.读取本地摄像头的数据流。2.提取通过train.py训练出来模型。3.利用OpenCV的接口实时截取人脸。4.使用模型对截取的人脸进行预测。但为了告知用户该图片检测结果的正确可能性（非预测准确度），我们需要在预测出人脸后对此结果做可能性判断。其基本算法如下：假设预测值向量pred为n\*1维向量，我们首先将其中值为负数的置0，再用np计算出该向量累和sum，用其中最大的浮点数p除以sum得到一个0~1的的值，即可能性。其原理是因为文中神经网络的输出并不为One-Hot向量，甚至还有负值，预测值所在的位置的值并不为1，但却是整个向量中最大的值。而对于那些负值所对应的人脸则根本不可能为真实预测结果，因此将其置0。其核心代码如下：def get\_face\_info(face):    face = cv2.resize(face,(size,size))    result,out\_ = sess.run((pred,out),feed\_dict={x\_holder:[face/255.0]})    return names[result[0]],compute\_acc(out\_)def img\_with\_face\_info(frame,face,pos):    name,acc = get\_face\_info(face)    st = "%s : %.3f%%"%(name,acc\*100)    img = cv2.putText(frame,st,pos, cv2.FONT\_HERSHEY\_SIMPLEX, 1, (0, 255, 0), 2)    return imgdef compute\_acc(out\_):    out\_ [out\_<0]=0    print(out\_[0])    acc\_= np.max(out\_)/np.sum(out\_)    return acc\_

cap = cv2.VideoCapture(0)

while(cap.isOpened() and cv2.waitKey(2)!=ord("q")):

flag,frame = cap.read()

img\_gray = cv2.cvtColor(frame,cv2.COLOR\_BGR2GRAY)

faces = classifer.detectMultiScale(img\_gray,1.1,5,minSize=(100,100))

for (x,y,w,h) in faces:

face = frame[y:y+h,x:x+w]

cv2.rectangle(frame,(x,y),(x+w,y+h),(0,0,255),2)

frame = img\_with\_face\_info(frame,face,(x,y))

cv2.imshow("face",frame)

其中get\_face\_info的输入为用户的脸部数据或者矩阵。输出为人名和对应的预测可能性为一个元组。img\_with\_face\_info的输入为用户图片，输出为带有用户人名和预测值可能性（百分数）的图片。compute\_acc为计算当前预测值可能性的方法。当然，也是要构建与train.py中结构一致的神经网络才能正常运行。其运行效果如下：



图 3.8 live\_detect.py 运行效果图（截取）

## 3.6性能评估

本节对文中的网络参数复杂程度和预测图片速度进行数据上的计算。

第一层卷积层卷积核大小为3\*3\*3\*32，加上偏置矩阵共1728个参数。第二层卷积层卷积核大小为3\*3\*32\*64，加上偏置矩阵共36864个参数。第三层卷积核大小为3\*3\*64\*128，加上偏置共147456个参数。两层全连接共8388608+1048576=9437184个参数。整个网络共9623232个参数。

对总共1189张图片进行100次训练。每次对将近九百六十万个参数调整（不考虑dropout），总共耗时41.38秒。每次训练约0.4138秒，对一张图片调整参数约0.0004138秒。这对于没有GPU辅助计算[12]的系统来说基本不可想象。而对于实时检测脚本来说，由于opencv没有运用GPU辅助运算，所以在检测人脸的时候耗费的时间比人脸识别更多。经过计算，识别一张图片耗时约在0.00199~0.00499秒内。人眼每秒看到约20~40张图片，也就是说在低于0.025秒的延时内有图像的变换人眼将忽略（不考虑人脑思考图片内容）。结合本文设计的系统的性能，完全可以做到用户直观感受上的不卡顿，因为最慢0.005秒内识别一张人脸，大大小于人眼时间差。此处不考虑OpenCV的目标检测的耗时，本测试环境为奔腾G5420，GPU为GTX1050。考虑高版本tensorflow的问题（高版本下tensorflow为AVX指令集下编译，而奔腾全系列CPU不支持AVX指令集，需要自行编译tensorflow）不建议使用奔腾系列处理器。

# 参考文献

[1] 陈磊. 卷积神经网络语言模型研究分析 2018

[2] Hinton. Imagenet classification with deep convolutional neural networks. 2012

[3] Hinton. Reducing the dimensionality of data with neural networks. 2006

[4] 周志华.《机器学习》 清华大学出版社 ISBN:9787302423287

[5] Hinton. Improving neural networks by preventing co-adaptation of feature detectors. 2012

[6] 赵永强，饶元，董世鹏，张君毅. 深度学习的目标跟踪算法综述 2020

[7] 刘鱼勍,马义德,钱志柏. 一种基于交叉熵的改进型PCNN图像自动分割新方法 2005

[8] 齐向明,高婷. 图像块的不可见性与鲁棒性均衡水印算法 2017

[9] 余烨,徐京涛,贺敏雪,路强. 增强边缘梯度特征局部量化策略驱动下的车标识别 2019

[10] 薛一哲,王拓. 基于代价敏感Adaboost目标跟踪 2016

[11] 江伟坚,郭躬德. 复杂环境下高效物体跟踪级联分类器 2014

[12] 齐美彬,李佶,蒋建国,王慈淳. 改进特征与GPU加速的行人检测 2018