

**本科生毕业论文（设计）**

**基于MTCNN算法的人脸检测系统设计和实现**

院 （系） **信息科技与技术学院**

专 业  计算机科学与技术

班 级  **F150112**

学 号  **F15011208**

学生姓名  **施阳**

指导教师及职称 **张璇**

定稿日期： 年 月 日

摘 要

随着计算机信息技术的发展，机器学习，深度学习等相关人工智能（**A**rtificial **I**ntelligence，以下简称AI）技术越来越成熟，相关产品也开始在生活中普及开来，如手机上的人脸解锁，人脸支付等。

本文通过对机器学习中的人脸检测（Face Detection）算法（Joint Face Detection and Alignment using Multi-task Cascaded Convolutional Networks[10]，多任务级联卷积网络对人脸检测和对齐，以下简称MTCNN）的学习和研究，使用Google开源的机器学习框架TensorFlow[2]对算法中的模型进行实现，并将训练好的模型应用于实际项目中，构建成一个人脸检测系统。

该系统主要分成三个服务部分。第一部分，便是训练好的MTCNN的神经网络模型，它作为剩下两个部分的基础；第二部分，是一个基于网络摄像头的实时人脸检测系统，可以对检测到的人脸进行计数和截图，它可以部署像在树莓派地性能设备上，也可以部署在具有高性能图形处理器（Graphic Processing Unit，以下简称GPU）的设备上；第三部分，是一基于Representational State Transfer[1]（表现层状态转换,以下简称rest）风格规范的超文本传输协议（Hypertext Transfer Protocol， 以下简称HTTP）应用程序编程接口（Application Programming Interface， 以下简称API）的图片检测服务。

其中整个系统主要采用Python语言进行编程，第一部分使用TensorFlow进行模型构建；第二部分使用开源软件OpenCV获取实时的摄像头，然后调用训练好的MTCNN的模型对实时输入的图像进行处理，检测人脸；第三部分采用前后端分离的设计方案，后端使用了基于Werkzeug，Jinja2的Python微型框架Flask[3]，api采用遵循restful风格进行设计， 前端主要使用了渐进式JavaScript框架—VueJS，和ElementUI[7]作为桌面UI组件库，使用axios[6]与后端api进行数据交互。

关键词：MTCNN，Python，VueJS，TensorFlow,系统设计，人脸检测，

ABSTRACT

With the development of computer information technology, related technologies such as machine learning and deep learning (Artificial Intelligence (hereinafter referred to as AI) technology are becoming more and more mature, and related products are beginning to spread in life, such as face unlocking on mobile phones. Face payment, etc.

This article uses Google's face detection and Alignment using Multi-task Cascaded Convolutional Networks (multi-tasking convolutional network for face detection and alignment, hereinafter referred to as MTCNN) learning and research, using Google’s open source machine learning framework TensorFlow implements the model in the algorithm and applies the trained model to the actual project to construct a face detection system.

The system is mainly divided into three service parts. The first part is the trained neural network model of MTCNN, which is the basis of the remaining two parts. The second part is a real-time face detection system based on webcam, which can count the detected faces and Screenshots, which can be deployed on a Raspberry pie performance device or on a device with a High Performance Graphics Unit; the third part is based on Representational State Transfer (performance) Layer state conversion, hereinafter referred to as the rest of the style of the hypertext transfer protocol (Hypertext Transfer Protocol, hereinafter referred to as HTTP) application programming interface (API) image detection service.

The whole system is mainly programmed in Python language. The first part uses TensorFlow to build the model. The second part uses the open source software OpenCV to get the real-time camera, and then calls the trained MTCNN model to process the real-time input image to detect the human face. The third part adopts the design of front and rear separation. The back end uses Werkzeug, Jinja2's Python micro-frame Flask. The api is designed in a restful style. The front end mainly uses the progressive JavaScript framework - VueJS, and ElementUI as the desktop UI component. Library, using axis for data interaction with the backend api.

**Key words**: MTCNN, Python, VueJS, TensorFlow, system design, face detection

目 录

目录

[摘 要 2](#_Toc4976683)

[ABSTRACT 2](#_Toc4976684)

[目 录 2](#_Toc4976685)

[第一章 绪论 4](#_Toc4976686)

[1.1 系统开发背景及意义 4](#_Toc4976687)

[1.2 系统开发任务和目标 4](#_Toc4976688)

[1.2.1第一部分 4](#_Toc4976689)

[1.2.2第二部分 4](#_Toc4976690)

[1.3 本章小结 5](#_Toc4976691)

[第二章 开发基础和相关技术理论 6](#_Toc4976692)

[2.1 开发基础和配置详情 6](#_Toc4976693)

[2.1.1 开发环境和相关配置 6](#_Toc4976694)

[2.1.2 开发技术简介 7](#_Toc4976695)

[2.2人脸检测简介和目前发展 8](#_Toc4976696)

[2.2.1问题由来 8](#_Toc4976697)

[2.2.2 检测的基本流程和发展。 9](#_Toc4976698)

[2.3 MTCNN介绍 11](#_Toc4976699)

[2.3.1 卷积神经网络简介 11](#_Toc4976700)

[2.3.2 MTCNN算法介绍和分析 13](#_Toc4976701)

[2.4本章小结 19](#_Toc4976702)

[第三章 基于MTCNN算法的模型构建和训练 20](#_Toc4976703)

[3.1 开发环境 20](#_Toc4976704)

[3.1.1 环境依赖 20](#_Toc4976705)

[3.1.2 安装依赖 21](#_Toc4976706)

[3.2 MTCNN模型构建和训练 21](#_Toc4976707)

[3.2.1 数据集介绍 21](#_Toc4976708)

[3.2.2 工程结构和文件目录说明 23](#_Toc4976709)

[3.2.3 模型构建 24](#_Toc4976710)

[3.2.4 模型训练 28](#_Toc4976711)

[3.3 本章小结 29](#_Toc4976712)

[第四章 人脸检测系统的设计和实现 30](#_Toc4976713)

[4.1 检测系统设计 30](#_Toc4976714)

[4.1.1 基于web的图片人脸检测部分的设计 30](#_Toc4976715)

[4.1.2 基于OpenCV网络摄像头的实时检测部分设计 31](#_Toc4976716)

[4.2. 检测系统实现 32](#_Toc4976717)

[4.2.1 检测系统的前后端实际实现 32](#_Toc4976718)

[4.2.2 检测系统的OpenCV实时监测实现 34](#_Toc4976719)

[4.3 系统测试 37](#_Toc4976720)

[4.4 优化 38](#_Toc4976721)

[4.3 本章小结 40](#_Toc4976722)

[第五章 总结 40](#_Toc4976723)

[5.1 分析总结 40](#_Toc4976724)

[5.2 展望 40](#_Toc4976725)

[参考文献 41](#_Toc4976726)

1. 绪论

## 系统开发背景及意义

随着计算机计算硬件的发展，AI技术的不断积累和突破，如今在各个行业都开始将AI技术进行落地，从手机拍照搜题到无人驾驶路测牌照的发放，AI技术将越来越贴近我们的日常生活，并逐渐成为我们生活的一部分。从手机解锁，人脸支付到车站安检，我们都可以发现少不了人脸识别这一项，而人脸检测便是人脸识别必不可少的一环之一，所以学习和开发相关的系统不仅具有很高的社会利用价值，同时也具有很高的实际应用需求。

本文便是以上面所提到的人脸识别中必不可少的人脸检测为出发点，从构建模型到实际应用系统的设计和开发为毕业设计课题。通过该课题的研究，学习和系统的开发，在逐渐了解图像识别领域的同时，也学习到了相关知识，掌握了相关工具和框架使用，从而让自己的职业发展也多了一个方向。

## 系统开发任务和目标

本文设计并开发的系统主要任务和目标分为如下两个个部分:

### 1.2.1第一部分

主要是使用TensorFlow实现MTCNN的模型，并且可以正常使用OpenCV调用：

使用OpenCV正常的获取网络摄像头，并进行人脸监测，完成摄像头的人脸计数以及摄像简单的人脸对齐后的截屏功能。

流程大致如下图：

调用MTCNN

MTCNN Model

训练获取MTCNN 模型参数

TensorFlow

OpenCV

IP 摄像头

从摄像头获取实时图像，并返回实时结果

### 1.2.2第二部分

完成前后端的设计并实现正常交互：

从前端上传图片到后端服务器，后端服务器正常接收，返回服务器生产的token参数（该图片在服务器上的保存路径的url和文件名），前端可以正常获取token 参数并且展示在页面上，前端再通过相关的参数返回给服务器进行图片检测。整个流程如下图：

MTCNN 调用TF获得结果

TensorFlow

MTCNN-Model

后台调用mtcnn模型，mtcnn返回结果

后台RESTful API

调用接口，获得结果

浏览器

Client

构建自己的Client或Server,进行二次开发

浏览器访问前端，通过前端进行接口的使用

用户

## 本章小结

本章先介绍了系统的开发的背景和实际意义，生活，学习和技术的角度上说明开发这个系统的社会应用价值，然后对系统设计和开发进行了切分，分成两个部分说明开发的具体任务和目标。

综上，不仅了解整个系统的开发背景及意义，也可以一窥系统的大致框架和运行流程，而下一章便是具体的说明该系统的设计和开发所涉及的相关基础知识理论和技术。

1. 开发基础和相关技术理论

上一章主要介绍了系统的开发背景和实际意义，以及大致的框架和运行流程。

这一章将从开发的基础技术和相关理论进行说明，将涉及到开发环境，选用的技术框架，以及它的优缺点，相关的神经网络简介，人脸检测的相关理论和基础等。

## 2.1 开发基础和配置详情

### 2.1.1 开发环境和相关配置

开发环境和技术框架详情见下表：

|  |  |  |  |
| --- | --- | --- | --- |
| 开发基础项 | 名称 | 版本 | 备注 |
| 环境 | Windows 10 | Profession x64 |  |
| 开发语言1 | Python | 3.6.8 |  |
| 开发语言2 | JavaScript | ECMAScript 6.0 |  |
| 主要技术框架1 | TensorFlow | Tensorflow(cpu-version) 1.13.0  tensorflow-gpu(gpu-verison) 1.13.0 | 二者具备其一即可 |
| 主要技术框架2 | Flask | 0.11.1 |  |
| 主要技术框架3 | VueJS | 3.5.1 |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

表2.1.1.1

相关硬件配置见下表：

|  |  |  |
| --- | --- | --- |
| 硬件配置 | 参数 | 备注 |
| CPU | Intel(R) Core(TM) i7-8750H CPU @ 2.20GHz \* 6 |  |
| GPU | NVIDIA GeForce RTX -2070 8GB |  |
| 摄像头 | 电脑默认 | 实时人脸检测必需要有 |
| RAM | 16g \* 1 |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

表2.1.1.2

### 2.1.2 开发技术简介

这里的简介将安装2.1.1中的表2.1.1.1的开发基础项进行介绍。

#### 开发语言1：Python

Python由 Guido van Rossum 在八十年代末和九十年代初，在荷兰国家数学和计算机科学研究所设计出来的。本身参考了很多其他语言发展而来的,这包括ABC、Modula-3、C、C++、Algol-68、SmallTalk、shell等等。[12]

Python特点如下：

* 一个高层次的结合了解释性、编译性、互动性和面向对象的脚本语言。
* 具有很强的可读性，相比其他语言经常使用英文关键字，其他语言的一些标点符号，它具有比其他语言更有特色语法结构。
* 一种解释型语言： 这意味着开发过程中没有了编译这个环节。类似于PHP和Perl语言。
* 是面向对象语言: 这意味着Python支持面向对象的风格或代码封装在对象的编程技术。

本系统中主要负责完成TensorFlow的MTCNN 模型训练，测试的编程，web后端的api的编程部分。

#### 开发语言2：JavaScript

JavaScript是互联网上最流行的脚本语言，这门语言可用于 HTML 和 web，更可广泛用于服务器、PC、笔记本电脑、平板电脑和智能手机等设备。[13]

JavaScript特点如下：

* 是一种轻量级的编程语言，很容易学习。
* JavaScript 是可插入 HTML 页面的编程代码。
* JavaScript 插入 HTML 页面后，可由所有的现代浏览器执行。

JavaScript主要负责前端界面的编程，

#### 开发技术框架1：TensorFlow

TensorFlow是Google开源的第二代用于数字计算（numerical computation）的软件库。它是基于数据流图的处理框架，图中的节点表示数学运算（mathematical operations），边表示运算节点之间的数据交互。TensorFlow从字面意义上来讲有两层含义，一个是Tensor，它代表的是节点之间传递的数据，通常这个数据是一个多维度矩阵（multidimensional data arrays）或者一维向量；第二层意思Flow,指的是数据流，形象理解就是数据按照流的形式进入数据运算图的各个节点。

它是一个非常灵活的框架，能够运行在个人电脑或者服务器的单个或多个CPU和GPU上，现在不仅支持移动设备上，还增加了浏览器的支持。

主要扮演快速的构建MTCNN模型，完成训练，测试和后续调用的角色。

#### 开发技术框架2：Flask

Flask是一个用Python编写的微web 框架。因为不需要特定的工具或库，没有数据库抽象层，表单验证或任何其他预先存在的第三方库提供常用功能的组件，所以是十分小巧的，而通过自身的扩展功能支持第三方插件，在开发的时候就显得的非常灵活。

在系统中主要承担web框架的角色，给前端做支撑。[13]

#### 开发技术框架3：VueJS

Vue (读音/vjuː/，类似于**view**) 是一套用于构建用户界面的**渐进式框架**。与其它大型框架不同的是，Vue 被设计为可以自底向上逐层应用。Vue 的核心库只关注视图层，不仅易于上手，还便于与第三方库或既有项目整合。另一方面，当与现代化的工具链以及各种支持类库结合使用时，Vue 也完全能够为复杂的单页应用提供驱动。[14]

## 2.2人脸检测简介和目前发展

人脸检测（Face Detection）是指在输入图像中确定所有人脸（如果存在）的位置、大小、 位姿的过程。人脸检测作为人脸信息处理中的一项关键技术，近年来成为模式识别与计算机 视觉领域内一项受到普遍重视、研究十分活跃的课题。[15]

### 2.2.1问题由来

来源于人脸识别（Face Recognition）的需求，比如我们现在实际生活中的支付宝里的人脸支付，手机上的人脸解锁等等应用。人脸识别系统中的首要问题就是解决人脸检测，因此从上个世纪70年代便开始研究这个问题。

因为人脸是一类具有相当复杂的细节变化的，以及检测的背景环境的不确定性，所以它的问题和挑战主要集中在一下这几个方面：

（1）人脸由于外貌、表情、肤色等不同，具有模式的可变性。比如白种人，颧骨较高，鼻梁高而窄，胡子和体毛发达，而黑种人则是鼻子宽扁，嘴唇厚而且外翻，胡子和体毛较少。；

（2）存在遮挡物，比如佩戴的眼镜、老人的长白胡须等等；

（3）人脸的3D结构较为复杂，所以同一个人在不同光源不同角度产生的图片差异也是很大的。

因此，如果能够找到解决这些问题的方法，成功构造出人脸检测与跟踪系统，将为解决其它类似的复杂模式检测问题提供重要的启示[15]。

### 2.2.2 检测的基本流程和发展。

简单的来说，人脸检测就是判定一张给定的图像上是否存在人脸，如果存在就给出人脸所在的位置。

尽管从上个世纪70年代便开始研究，但是受到技术条件和实际需求等因素影响，人脸检测技术的研究工作发展的并不是很迅速，直到90年代开始人脸检测的脚步才逐渐开始加快。千禧年过后的十几年间，出现了大量的人脸检测研究工作。

2001年Paul Viola和Michael Jones在CVPR上提出Viola-Jones算法，因其快速的检测被后续的科研工作者或项目广泛的采用与改进，为后面的发展奠定基础。例如《基于模板和支持矢量机[17-18]（Support Vector Machines，以下简称 SVM）的人脸检测[16]》，通过“人脸-双眼”构成模板对，滤波时先使用双眼模板再使用人脸模板，以进一步提高速度并取得更好的性能[16],再通过SVM提高人工神经网络泛化能力，使得检测速度平均提高了二十倍以上。[16]

虽然，这些技术放到现在还远远不能达到生活场景实际使用的要求，但是整体检测的思路和流程以及和现在的方法大相径庭了。即给定一张图片，完成人脸检测的大致步骤如下：

1. 选择一个n\*n大小的观察窗口；
2. 提取选定窗口中的一些特征，作为这个窗口的代表；
3. 根据代表描述的信息来判断这个窗口中是否存在人脸
4. 循环执行上述步骤直到遍历完整张图片
5. 如果存在，则判定存在并返回其位置信息，反之则判定不存在

下面为该流程做一个图片演示:

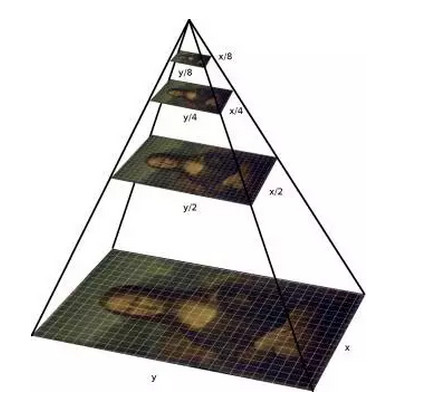
图片2.2.2.（1）

首先确定红色为观察窗口，从左到右，从上到下进行观察，提取特征进行对比，到达黄色框位置，最后获得两个绿色框的结果，返回存在人脸和两个绿色框的位置信息。

但是，新的问题便是如何确定这个观察窗口的大小，因为不同照片上的人脸大小是不同的。

如下，相同的窗口在图片2.2.2（2）上就不能正确返回结果，图片2.2.2（3）则是正常结果。

图片2.2.2.（2） 图片2.2.2（3）

因此，出现了一种图像处理方式—图像金字塔算法[19],图片2.2.2（4）便是其的变换过程

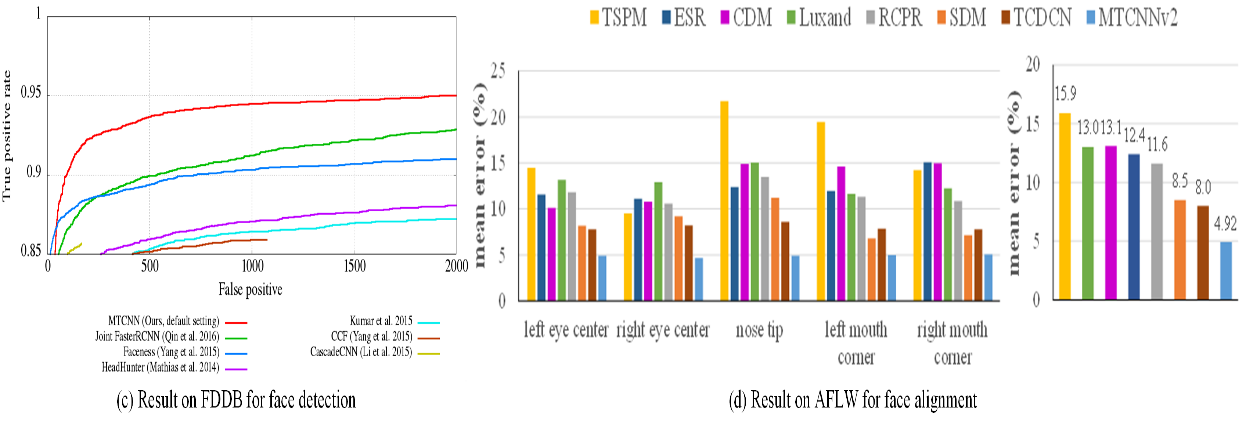
图片2.2.2（4）

通过构建图像金字塔，同时允许窗口和人脸的贴合程度在小范围内变动，这一能够检测到不同位置、不同大小的人脸。

然而，上面的图片的中的人脸都是一个姿态，遇到多姿态检测任务时可能不能得到很好的结果。为了解决这一问题，采用了分治策略，通过对不同姿态的人脸进行单个训练，获得多个分类器，将这些分类组合起来变成级联分类器,从而解决这个问题。

再然后，通过卷积神经网络（Convolutional neural network，以下简称CNN）[20]来处理图像更是得心应手。深度学习的到来，出现了R-CNN(Rich feature hierarchies for accurate object detection and semantic segmentation)[21], 抛弃了滑动窗口的方法，取而代之的是一个新的生成候选窗口的环节。即对于给定的图像，不再用一个滑动窗口去对图像进行扫描，枚举所有可能的情况，而是采用某种方式“提名”出一些候选窗口，在获得对待检测目标可接受的召回率的前提下，候选窗口的数量可以控制在几千个或者几百个。另一个很大改变是在特征提取上：不再采用人工设计的特征，而是用CNN来自动学习特征。特征提取过程就是从原始的输入图像（像素颜色值构成的矩阵）变换到特征向量的过程。R-CNN的改革带来了一系列的方法，Fast R-CNN[22]，Fast R-CNN[23]，YOLO[24]等，这些方法也不再仅限单一的人脸或物体检测，而是发展成通用目标检测。

在本文使用的MTCNN算法发表2016年，在人脸检测和特征点定位任务上，相比于R-CNN系列通用检测方法，它更加针对人脸检测这一专门的任务，在速度和精度都有足够的提升。，而且具有实时处理的性能。下面是来自官方的在FDDB 和WIDER FACE benchmarks 上的对比：



图片 result of benchmarks[25]

## 2.3 MTCNN介绍

### 2.3.1 卷积神经网络简介

卷积神经网络（Convolutional Neural Network, CNN）是一种前馈神经网络，它的人工神经元可以响应一部分覆盖范围内的周围单元[26]，对于大型图像处理有出色表现。

CNN由一个或多个卷积层和顶端的全连通层（对应经典的神经网络）组成，同时也包括关联权重和池化层（pooling layer）。这一结构使得卷积神经网络能够利用输入数据的二维结构。与其他深度学习结构相比，卷积神经网络在图像和语音识别方面能够给出更好的结果。这一模型也可以使用反向传播算法进行训练。相比较其他深度、前馈神经网络，卷积神经网络需要考量的参数更少，使之成为一种颇具吸引力的深度学习结构[27]。

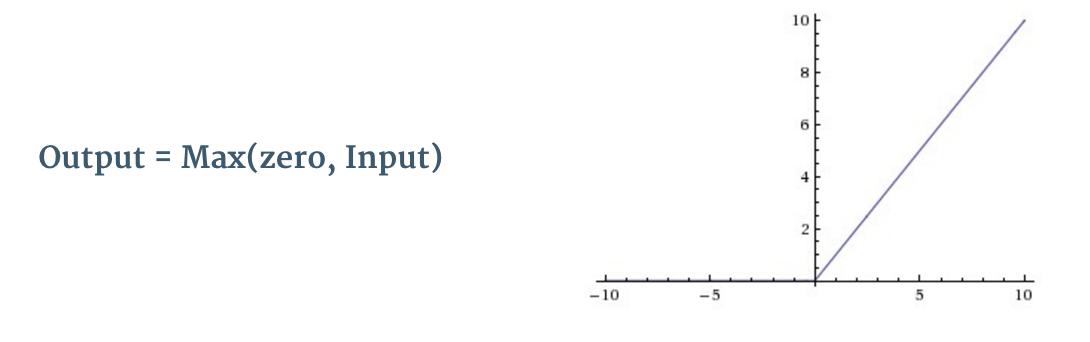
通常是由有四层组成。

1. 卷积层：

卷积层（Convolutional layer），卷积神经网络中每层卷积层由若干卷积单元组成，每个卷积单元的参数都是通过反向传播算法最优化得到的。卷积运算的目的是提取输入的不同特征，第一层卷积层可能只能提取一些低级的特征如边缘、线条和角等层级，更多层的网路能从低级特征中迭代提取更复杂的特征。

1. 线性整流层

线性整流层（Rectified Linear Units layer, ReLU layer）使用线性整流（Rectified Linear Units, ReLU）f(x)=max(0,x)作为这一层神经的激励函数（Activation function）。它可以增强判定函数和整个神经网络的非线性特性，而本身并不会改变卷积层。其他的一些函数也可以用于增强网络的非线性特性，如双曲正切函数f(x) = tanh(x)，f(x) = |tanh(x)|，或者Sigmoid函数f(x)=(1+e-x)-1。



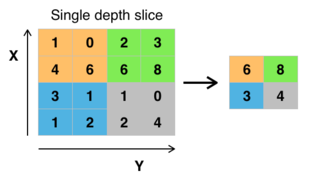
图片 ReLU函数

相比其它函数来说，ReLU函数更受青睐，这是因为它可以将神经网络的训练速度提升数倍[28]，而并不会对模型的泛化准确度造成显著影响。

1. 池化层：

池化（Pooling）是卷积神经网络中另一个重要的概念，它实际上是一种形式的降采样。有多种不同形式的非线性池化函数，而其中“最大池化（Max pooling）”是最为常见的。它是将输入的图像划分为若干个矩形区域，对每个子区域输出最大值。这种机制能够有效地原因在于，在发现一个特征之后，它的精确位置远不及它和其他特征的相对位置的关系重要。

池化层会不断地减小数据的空间大小，因此参数的数量和计算量也会下降，这在一定程度上也控制了过拟合。通常来说，CNN的卷积层之间都会周期性地插入池化层。池化层通常会分别作用于每个输入的特征并减小其大小。

 图片 max\_pooling[31]

当前最常用形式的池化层是每隔2个元素从图像划分出 2\*2 的区块，然后对每个区块中的4个数取最大值。这将会减少75%的数据量。除了最大池化之外，池化层也可以使用其他池化函数，例如“平均池化”甚至“L2-范数池化”等。过去，平均池化的使用曾经较为广泛，但是最近由于最大池化在实践中的表现更好，平均池化已经不太常用。

由于池化层过快地减少了数据的大小，当前文献中的趋势是使用较小的池化滤镜，[29]甚至不再使用池化层。[30]

1. 损失函数层

损失函数层（loss layer）用于决定训练过程如何来“惩罚”网络的预测结果和真实结果之间的差异，它通常是网络的最后一层。各种不同的损失函数适用于不同类型的任务。例如，Softmax交叉熵损失函数常常被用于在K个类别中选出一个，而Sigmoid交叉熵损失函数常常用于多个独立的二分类问题。欧几里德损失函数常常用于结果取值范围为任意实数的问题。

### 2.3.2 MTCNN算法介绍和分析

MTCNN由3个神经网络P-Net（Proposal Network， 以下简称P-Net）, R-Net（Refine Network， 以下简称R-Net）, O-Net(Output Network, 以下简称-Net)组成：

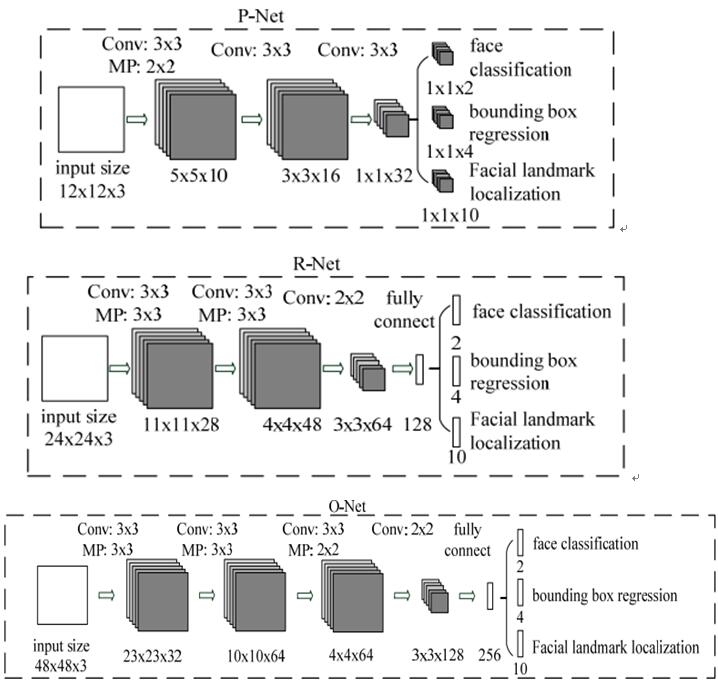
1. P-Net:

该网络结构主要获得了人脸区域的候选窗口和边界框的回归向量。并用该边界框做回归，对候选窗口进行校准，然后通过非极大值抑制（NMS）来合并高度重叠的候选框。

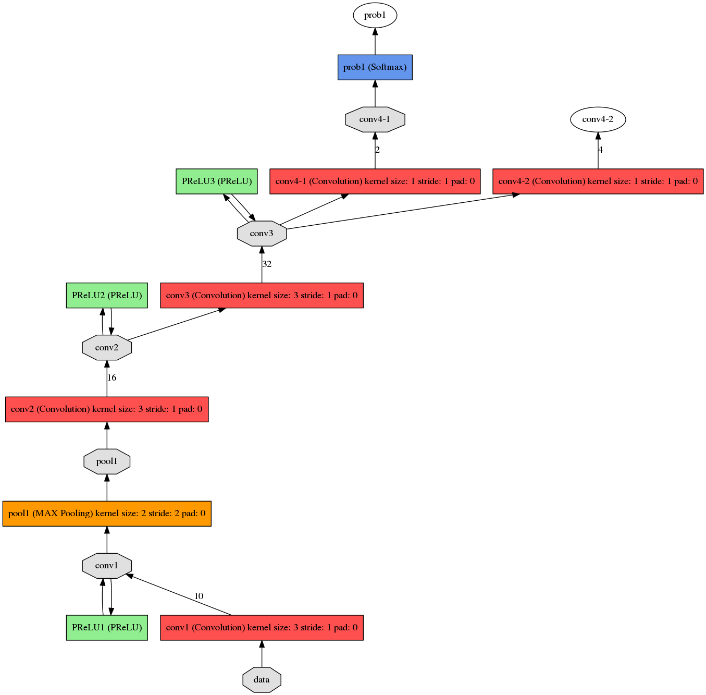
1. R-Net：

该网络结构通过边界框回归和NMS来去掉那些false-positive区域。只是由于该网络结构和P-Net网络结构有差异，多了一个全连接层，所以会取得更好的抑制false-positive的作用。

1. O-Net：

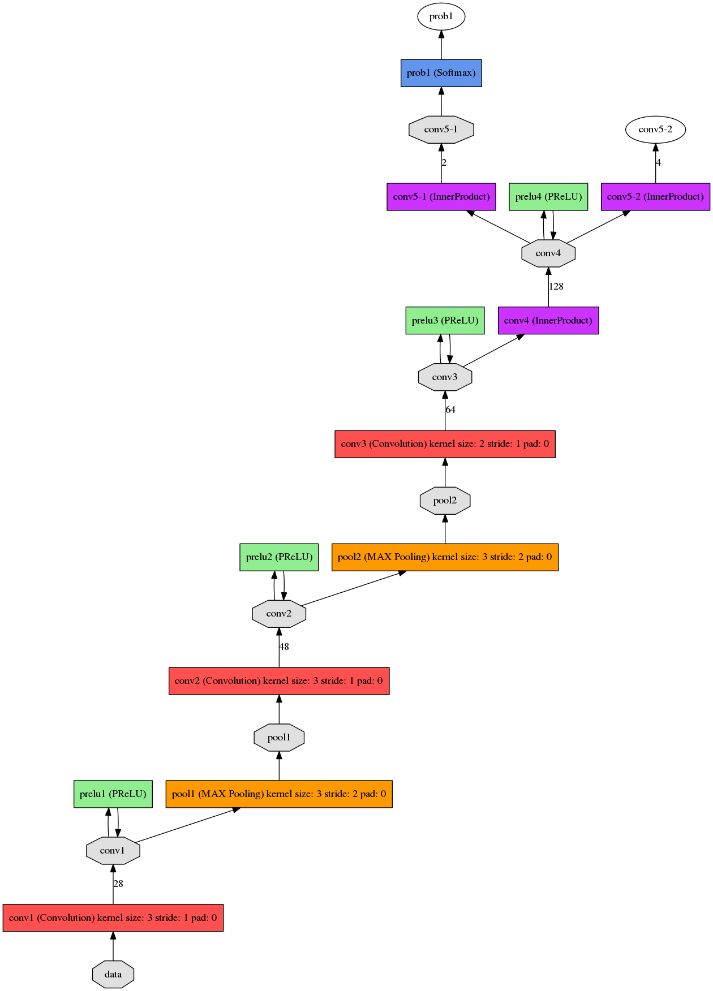
该层比R-Net层又多了一层卷基层，所以处理的结果会更加精细。作用和R-Net层作用一样。但是该层对人脸区域进行了更多的监督，同时还会输出5个地标（landmark）

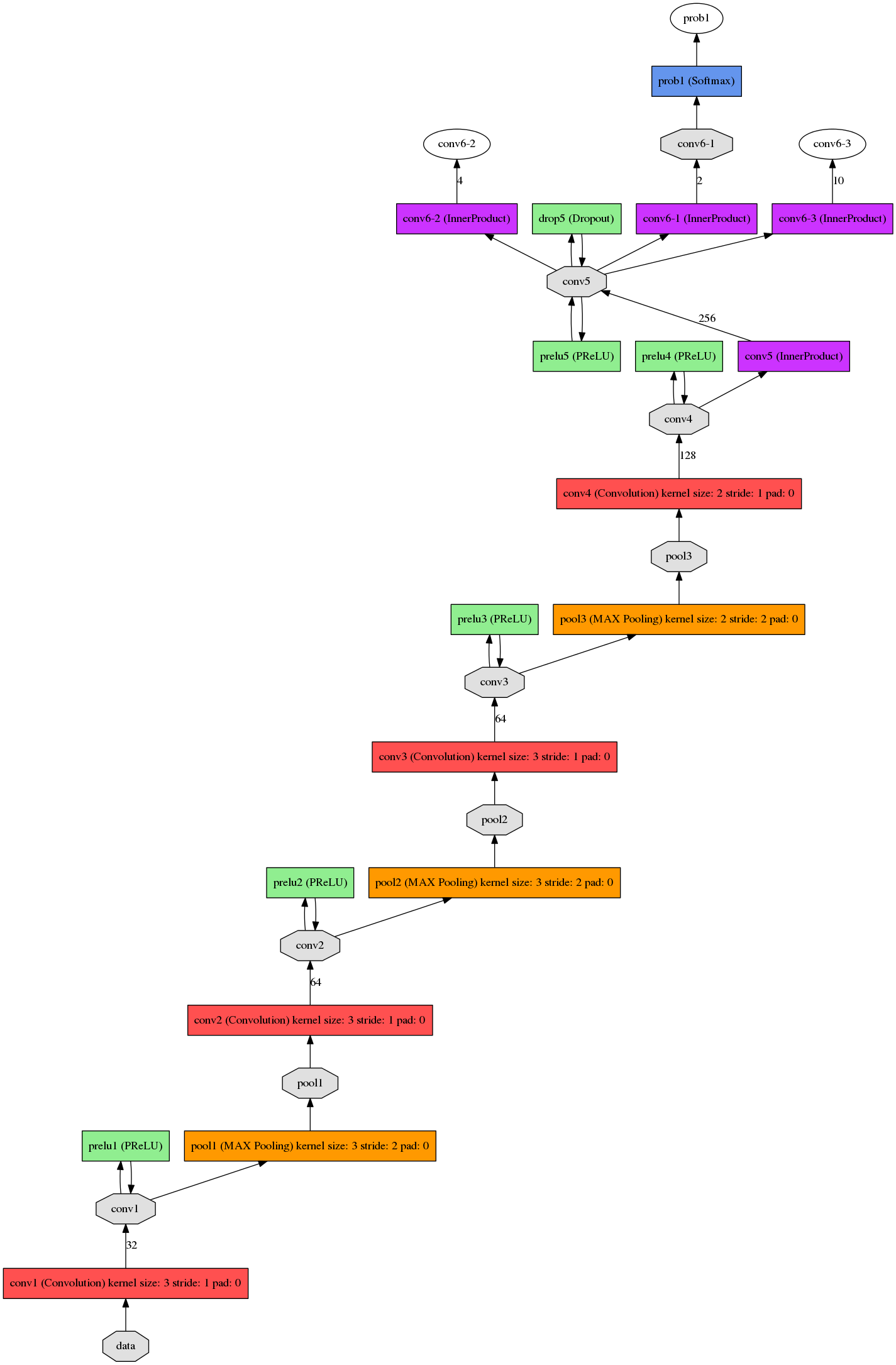
图片 MTCNN各层网络结构

P-Net 网络结构关系图：

图片 P-Net

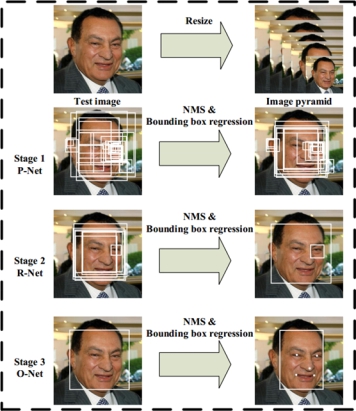
R-Net 网络结构关系图：

 图片 R\_Net

O-Net网络结构：

图片O\_Net

算法包括三个阶段[10]：首先，通过快速提案网络(P-Net)生成候选窗口。在此之后，将在下一阶段通过细化网络(R-Net)来细化这些候选项。在第三阶段，输出网络(O-Net)产生最终的人脸框与标志定位。算法流程如图所示[10]。



图片 算法阶段

总体框架[10]

对于一个给定的图像，首先将其调整到不同的比例，以构建一个图像金字塔，这是以下三级级联框架的输入，其推理流程如图所示[10]：

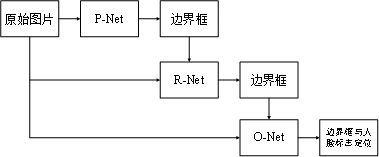


图 MTCNN的推理流程图[10]

第1阶段[10]：首先，利用一个名为提议网络(P-Net)的全连接卷积神经网络获得候选脸部窗口和边界坐标回归向量；然后，估计边界框回归向量进行候选窗口的校准；最后，使用了非最大抑制（NMS，Non-Maximum Suppression）进行合并重叠的候选窗口。

第二阶段[10]:所有候选窗口被传到下一层卷积神经网络——提炼网络（R-Net）该结构进一步排除大量错误候选窗口，进行边界回归校准，并进行NMS抑制。

第三阶段[10]:这一阶段与第二阶段相似，但这一阶段，用更多的监督去识别脸部区域。最终会在网络输出五个脸部标志位置——左/右眼、鼻尖、左/右嘴角。

MTCNN的优化

考虑到基础的卷积神经网络可能受到以下因素的影响[10]：

(1)一些卷积层的卷积核缺少多样性，可能限制了它们的识别能力[10]；

(2)与其他多类目标检测和分类任务相比，人脸检测是一个非常具有挑战性的二值类别任务，所以它每一层需要少量的卷积核。为此，需要减少核的数量并且将5x5的卷积核换成3x3的卷积核减少计算量，同时增加网络深度以取得更好的效果。有了这些改进，和先前的结构相比，我们用更少的时间取得了更好的效果（训练阶段的结果如表1所示。为了公平比较，我们在每个组中使用了相同的训练集和验证集）。基于多任务深度级联卷积神经网络的网络结构如图2所示。其使用PReLU[32]作为非线性激活函数，并应用在卷积层和全连接层之后（除了输出层）[10]。

表 CNN与MTCNN处理图像的速度与验证精度对比[10]

|  |  |  |  |
| --- | --- | --- | --- |
| Group | CNN | 300 × Forward  Propagation | Validation Accuracy |
| Group1 | 12-Net  P-Net | 0.038s  0.031s | 94.4%  94.6% |
| Group2 | 24-Net  R-NET | 0.738s  0.458s | 95.1%  95.4% |
| Group3 | 48-Net  O-Net | 3.577s  1.347s | 93.2%  95.4% |

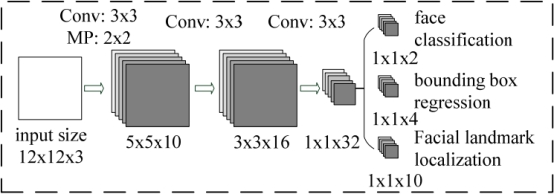


图 P-Net网络结构图[10]

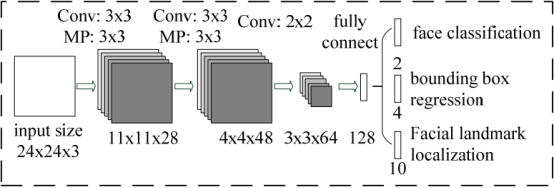


图 R-Net网络结构图[10]

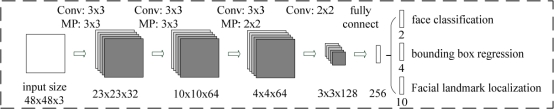


图 O-Net网络结构图[10]

## 2.4本章小结

本章通过开发环境和相关配置的说明，人脸检测发展简介，MTCNN算法介绍为接下来模型构建和系统设计实现奠定了基石，下一章就着重介绍MTCNN在TensorFlow上的实现。

1. 基于MTCNN算法的模型构建和训练

上一章主要是详细的介绍了开发的基础，以及设计系统和开发系统时所需要涉及到的基础技术和理论，而这章将紧接其后，利用这些相应的技术对整个系统的核心基石部分进行构建和开发，同时开始真正的接触深度学习啦！

## 3.1 开发环境

### 3.1.1 环境依赖

开发环境win10, Python3.6.8;

使用Tensorflow 框架开发，相关的依赖见下表：

表 MTCNN模型训练环境依赖表

|  |  |
| --- | --- |
| 依赖包 | 版本 |
| astor | 0.7.1 |
| gast | 0.2.2 |
| grpcio | 1.18.0 |
| h5py | 2.9.0 |
| Keras | 2.2.4 |
| Keras-Applications | 1.0.7 |
| Keras-Preprocessing | 1.0.9 |
| Markdown | 3.0.1 |
| mock | 2.0.0 |
| numpy | 1.16.1 |
| opencv-python | 4.0.0.21 |
| pbr | 5.1.3 |
| protobuf | 3.6.1 |
| PyYAML | 3.13 |
| scipy | 1.2.1 |
| six | 1.12.0 |
| tensorboard | 1.13.0 |
| tensorflow | 1.13.1 |
| tensorflow-estimator | 1.13.0 |
| tensorflow-gpu | 1.13.1 |
| termcolor | 1.1.0 |
| tqdm | 4.31.1 |
| Werkzeug | 0.14.1 |
| absl-py | 0.7.0 |

### 3.1.2 安装依赖

(1) 安装虚拟环境

使用“win + e”打开文件浏览器，创建好项目目录，如：

D:\2019SandaU\code\mtcnn

在地址栏输入“cmd”进入win10 cmd命令行，输入“virtualenv --no-site-packages venv”

等待片刻便创建好了，然后在输入“venv\Scripts\activate”激活虚拟环境

在当前路径下新建env.txt文件并将下面的依赖复制进去保存

absl-py==0.7.0

astor==0.7.1

gast==0.2.2

grpcio==1.18.0

h5py==2.9.0

Keras==2.2.4

Keras-Applications==1.0.7

Keras-Preprocessing==1.0.9

Markdown==3.0.1

mock==2.0.0

numpy==1.16.1

opencv-python==4.0.0.21

pbr==5.1.3

protobuf==3.6.1

PyYAML==3.13

scipy==1.2.1

six==1.12.0

tensorboard==1.13.0

tensorflow==1.13.1

tensorflow-estimator==1.13.0

tensorflow-gpu==1.13.1

termcolor==1.1.0

tqdm==4.31.1

Werkzeug==0.14.1

在cmd 里面输入“pip install -r env.txt”所有等待依赖安装完成即可。

综上，整个环境就部署完成了。

## 3.2 MTCNN模型构建和训练

### 3.2.1 数据集介绍

数据集使用公共数据集WIDER Face[33] 和 Deep Convolutional Network Cascade for Facial Point Detection [34]（使用Celeb Face[35]训练也可以）。

其中WIDER Face数据集用来进行人脸检测，Deep Convolutional Network Cascade for Facial Point Detection用来特征点检测。

#### 3.2.1.1 WIDER Face介绍

WIDER FACE是香港中文大学的一个提供更广泛人脸数据的人脸检测基准数据集，由YangShuo， Luo Ping ，Loy ，Chen Change ，Tang Xiaoou收集。

它包含32203个图像和393703个人脸图像，在尺度，姿势，闭塞，表达，装扮，关照等方面表现出了大的变化。WIDER FACE是基于61个事件类别组织的，对于每一个事件类别，选取其中的40%作为训练集，10%用于交叉验证（cross validation），50%作为测试集。和PASCAL VOC数据集一样，该数据集也采用相同的指标。和MALF和Caltech数据集一样，对于测试图像并没有提供相应的背景边界框。

#### 3.2.1.2 Deep Convolutional Network Cascade for Facial Point Detection介绍

Deep Convolutional Network Cascade for Facial Point Detection数据集分为训练集和测试集，其中

测试集：

它包含5,590个LFW图像和7,876个从网上下载的图像。训练集和验证集分别在trainImageList.txt和testImageList.txt中定义。这些文本文件的每一行都以图像名称开头，然后是我们的面部检测器修复的面部边界框的边界位置，然后是五个面部点的位置。

训练集：

它包含测试中使用的1,521个BioID图像，781个LFPW训练图像和249个LFPW测试图像，以及记录每个数据集由面部检测器修复的面部边界框边界位置的文本文件。我们的面部检测器失败的一些图像未在文本文件中列出。LFPW图像被重命名以便于处理。

#### 3.2.1.3 Celeb Face介绍

该数据集为香港中文大学汤晓鸥老师组开源的数据集，主要包含了5个关键点，40个属性值等，包含了202599张图片，图片都是高清的名人图片，可以用于人脸检测，5点训练，人脸头部姿势的训练等。

### 3.2.2 工程结构和文件目录说明

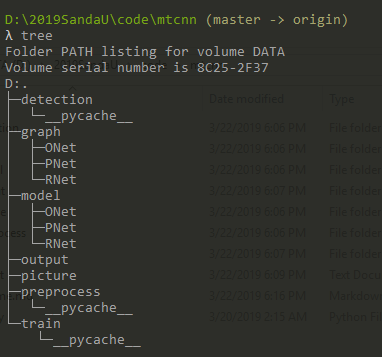
 如下图，是mtcnn 模型目录结构

图 mtcnn项目文件目录

目录说明：

codes MTCNN 论文作者基于caffe实现的代码

detection 用于检测相关操作的文件

env.txt 项目的环境依赖文件，便于后续的迁移和部署

graph 训练过程产生的日志和模型相应的图信息

model 存储训练更新后的模型权重值

output 放置识别图像或视频完成后存储放置的路径

picture 是要测试的图像放置路径

preprocess 主要是预处理数据程序脚步

Readme.md 项目说明

test.py 测试脚步

train 训练模型脚步

data 存放数据集和tfrecord

具体目录下文件说明:

detection下的fcn\_detector.py主要用于PNet的单张图片识别，detector.py用于RNet和ONet的一张图片通过PNet截取的多个人脸框的批次识别，MtcnnDetector.py为识别人脸和生成RNet，ONet输入数据

preprocess里是预处理数据程序，BBox\_utils.py和utils.py，loader.py是一些辅助程序，gen\_12net\_data.py是生成PNet的pos，neg,part的程序，gen\_landmark\_aug.py是生成landmark数据的程序，gen\_imglist\_pnet.py是PNet的四种数据组合一起，gen\_hard\_example.py是生成RNet, ONet的三种数据程序，gen\_tfrecords.py是生成tfrecord文件的程序

train中的config是一些训练和测试参数设定，model.py是模型,train.py是训练，train\_model.py针对不同网络训练

data下放置训练所用的原始数据和划分数据，生成的tfrecord

### 3.2.3 模型构建

##### 3.2.2.1.1 P-Net

模型代码：

def P\_Net(inputs, label=None, bbox\_target=None, landmark\_target=None, training=True):

'''pnet的结构'''

with tf.variable\_scope('PNet'):

with slim.arg\_scope([slim.conv2d], activation\_fn=prelu,

weights\_initializer=slim.xavier\_initializer(),

weights\_regularizer=slim.l2\_regularizer(0.0005),

padding='VALID'):

net = slim.conv2d(inputs, 10, 3, scope='conv1')

net = slim.max\_pool2d(

net, kernel\_size=[2, 2], stride=2, padding='SAME', scope='pool1')

net = slim.conv2d(net, 16, 3, scope='conv2')

net = slim.conv2d(net, 32, 3, scope='conv3')

# 二分类输出通道数为2

conv4\_1 = slim.conv2d(

net, 2, 1, activation\_fn=tf.nn.softmax, scope='conv4\_1')

bbox\_pred = slim.conv2d(

net, 4, 1, activation\_fn=None, scope='conv4\_2')

landmark\_pred = slim.conv2d(

net, 10, 1, activation\_fn=None, scope='conv4\_3')

if training:

cls\_prob = tf.squeeze(

conv4\_1, [1, 2], name='cls\_prob') # [batch,2]

cls\_loss = cls\_ohem(cls\_prob, label)

bbox\_pred = tf.squeeze(

bbox\_pred, [1, 2], name='bbox\_pred') # [bacth,4]

bbox\_loss = bbox\_ohem(bbox\_pred, bbox\_target, label)

landmark\_pred = tf.squeeze(

landmark\_pred, [1, 2], name='landmark\_pred') # [batch,10]

landmark\_loss = landmark\_ohem(

landmark\_pred, landmark\_target, label)

accuracy = cal\_accuracy(cls\_prob, label)

L2\_loss = tf.add\_n(slim.losses.get\_regularization\_losses())

return cls\_loss, bbox\_loss, landmark\_loss, L2\_loss, accuracy

else:

# 测试时batch\_size=1

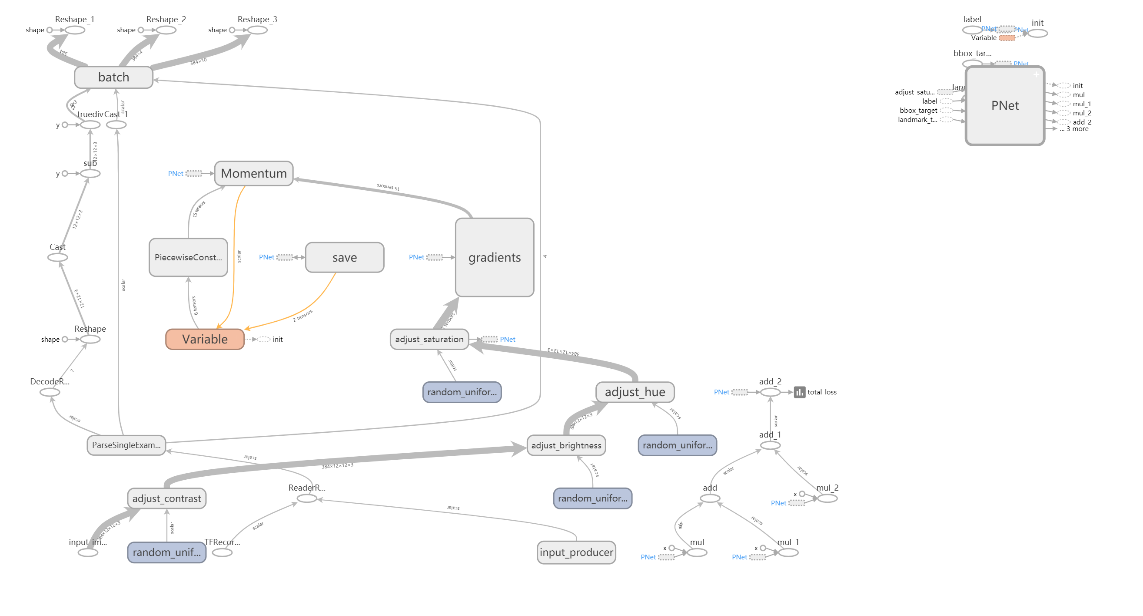
cls\_pro\_test = tf.squeeze(conv4\_1, axis=0)

bbox\_pred\_test = tf.squeeze(bbox\_pred, axis=0)

landmark\_pred\_test = tf.squeeze(landmark\_pred, axis=0)

return cls\_pro\_test, bbox\_pred\_test, landmark\_pred\_test

使用tTensorFlow中P-Net的graph图片如下：



##### 3.2.2.1.2 R-Net

def R\_Net(inputs, label=None, bbox\_target=None, landmark\_target=None, training=True):

'''RNet结构'''

with tf.variable\_scope('RNet'):

with slim.arg\_scope([slim.conv2d],

activation\_fn=prelu,

weights\_initializer=slim.xavier\_initializer(),

weights\_regularizer=slim.l2\_regularizer(0.0005),

padding='VALID'):

net = slim.conv2d(inputs, 28, 3, scope='conv1')

net = slim.max\_pool2d(

net, kernel\_size=[3, 3], stride=2, padding='SAME', scope='pool1')

net = slim.conv2d(net, 48, 3, scope='conv2')

net = slim.max\_pool2d(

net, kernel\_size=[3, 3], stride=2, scope='pool2')

net = slim.conv2d(net, 64, 2, scope='conv3')

fc\_flatten = slim.flatten(net)

fc1 = slim.fully\_connected(

fc\_flatten, num\_outputs=128, scope='fc1')

cls\_prob = slim.fully\_connected(

fc1, num\_outputs=2, activation\_fn=tf.nn.softmax, scope='cls\_fc')

bbox\_pred = slim.fully\_connected(

fc1, num\_outputs=4, activation\_fn=None, scope='bbox\_fc')

landmark\_pred = slim.fully\_connected(

fc1, num\_outputs=10, activation\_fn=None, scope='landmark\_fc')

if training:

cls\_loss = cls\_ohem(cls\_prob, label)

bbox\_loss = bbox\_ohem(bbox\_pred, bbox\_target, label)

landmark\_loss = landmark\_ohem(

landmark\_pred, landmark\_target, label)

accuracy = cal\_accuracy(cls\_prob, label)

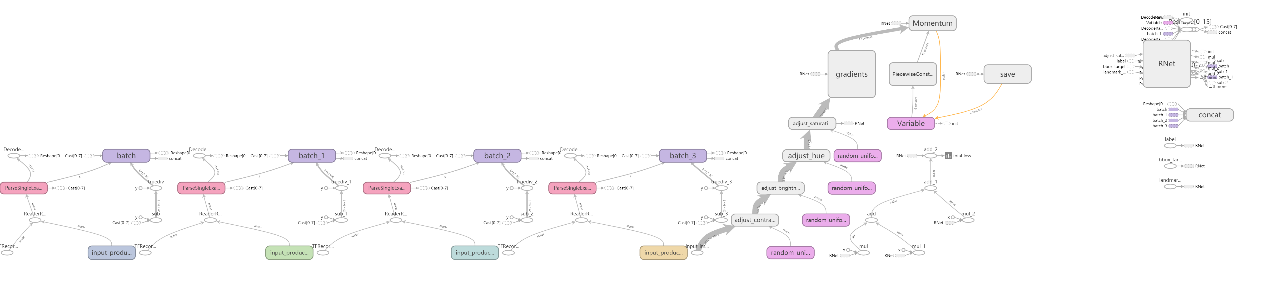
L2\_loss = tf.add\_n(slim.losses.get\_regularization\_losses())

return cls\_loss, bbox\_loss, landmark\_loss, L2\_loss, accuracy

else:

return cls\_prob, bbox\_pred, landmark\_pred

使用tensorflow中R-Net的graph图片如下:



##### 3.2.2.1.3 O-Net

def O\_Net(inputs, label=None, bbox\_target=None, landmark\_target=None, training=True):

'''ONet结构'''

with tf.variable\_scope('ONet'):

with slim.arg\_scope([slim.conv2d],

activation\_fn=prelu,

weights\_initializer=slim.xavier\_initializer(),

weights\_regularizer=slim.l2\_regularizer(0.0005),

padding='VALID'):

net = slim.conv2d(inputs, 32, 3, scope='conv1')

net = slim.max\_pool2d(

net, kernel\_size=[3, 3], stride=2, padding='SAME', scope='pool1')

net = slim.conv2d(net, 64, 3, scope='conv2')

net = slim.max\_pool2d(

net, kernel\_size=[3, 3], stride=2, scope='pool2')

net = slim.conv2d(net, 64, 3, scope='conv3')

net = slim.max\_pool2d(

net, kernel\_size=[2, 2], stride=2, padding='SAME', scope='pool3')

net = slim.conv2d(net, 128, 2, scope='conv4')

fc\_flatten = slim.flatten(net)

fc1 = slim.fully\_connected(

fc\_flatten, num\_outputs=256, scope='fc1')

cls\_prob = slim.fully\_connected(

fc1, num\_outputs=2, activation\_fn=tf.nn.softmax, scope='cls\_fc')

bbox\_pred = slim.fully\_connected(

fc1, num\_outputs=4, activation\_fn=None, scope='bbox\_fc')

landmark\_pred = slim.fully\_connected(

fc1, num\_outputs=10, activation\_fn=None, scope='landmark\_fc')

if training:

cls\_loss = cls\_ohem(cls\_prob, label)

bbox\_loss = bbox\_ohem(bbox\_pred, bbox\_target, label)

landmark\_loss = landmark\_ohem(

landmark\_pred, landmark\_target, label)

accuracy = cal\_accuracy(cls\_prob, label)

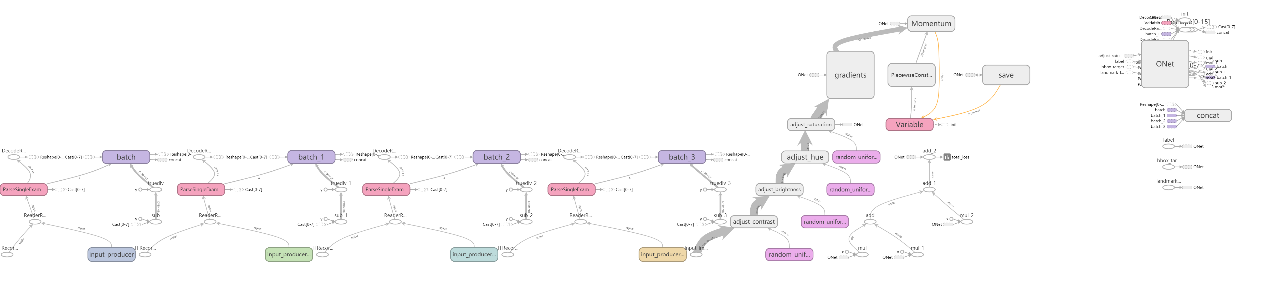
L2\_loss = tf.add\_n(slim.losses.get\_regularization\_losses())

return cls\_loss, bbox\_loss, landmark\_loss, L2\_loss, accuracy

else:

return cls\_prob, bbox\_pred, landmark\_pred

使用tensorflow 中O-Net的graph图片如下:



需要生产P-Net，R-Net，O-Net各自的训练数据然后进行训练，这三个神经网络需一个一个按顺序的训练。

1. 打开文件浏览器进入到相应目录，地址栏输入”cmd”进入cmd，输入”Python gen\_12\_data.py “命令生成三种PNet的数据
2. 接着上一步，输入“python gen\_landmark\_aug.py 12”生成PNet需要的landmark数据
3. 接着在输入“python gen\_imlist.py”将PNet数据整理到一起
4. 最后输入“python ../train/train.py 12”开始训练

### 3.2.4 模型训练

MTCNN中的三个神经网络P-Net，R-Net, O-Net需要依次单独的进行训练具体过程如下：

mtcnn\preprocess下:

python gen\_12net\_data.py 生成三种pnet数据，

python gen\_landmark\_aug.py 12 生成pnet的landmark数据，

python gen\_imglist\_pnet.py 整理到一起，

python gen\_tfrecords.py 12 生成tfrecords文件

切换路径到mtcnn\trainn上python train.py 12 训练pnet

mtcnn\preprocess下:，

python gen\_hard\_example.py 12 生成三种rnet数据，

python gen\_landmark\_aug.py 24 生成rnet的landmark数据,

python gen\_tfrecords.py 24 生成tfrecords文件

切换路径到mtcnn\trainn上python train.py 24 训练rnet

mtcnn\preprocess下:

python gen\_hard\_example.py 24 生成三种onet数据，

python gen\_landmark\_aug.py 48 生成onet的landmark数据,

python gen\_tfrecords.py 48 生成tfrecords文件

切换路径到mtcnn\trainn上 python train.py 48 训练onet

## 3.3 本章小结

本章对MTCNN的工程目录，数据集进行详细的说明，模型的构建和训练进行了演示，直观而形象的展示了MTCNN模型的构建和训练过程。下章将使用本章训练好的模型进行实际的模型调用。

# 第四章 人脸检测系统的设计和实现

上一章中，通过对MTCNN算法的学习和研究，完成了对MTCNN的模型的构建和训练，从而给整个系统铸造了核心的基石模块，为本章的系统的设计和实现奠定了成功的大梁。

本章的系统设计将分成两个小部分进行设计和实现。

## 4.1 检测系统设计

由于技术原因，无法完成Flask后端API 和前端实时的Web RTC(Real-Time Communications, 简称RTC)[36]交互，所以在这里将最初的web实时摄像头检测改为基于OpenCV的实时网络摄像头进行检测。

### 4.1.1 基于web的图片人脸检测部分的设计

#### 4.1.1.1 web接口架构设计

Web 整体采用 RESTful风格进行设计，从而实现简洁干练的访问接口，核心接口设计如下：

Flask Web RESTful API



Img files

MTCNN Model

Tensorflow

VueJS

Axios request/response

上传检测图片，获得服务文件token

浏 览 器

通过token获取img

发起检测请求发送token，获得检测结果

#### 4.1.1.2 前端页面设计

前端web界面，采用VueJS + Element UI的组合设计和编程实现。

ElementUI是由阿里巴巴的饿了么团队进行开发的UI组件，一套为开发者、设计师和产品经理准备的基于 Vue 2.0 的桌面端组件库[37]。而VueJS 是一个简单，灵活，高效的渐进式框架，其中大部分组件的桌面已经写好，在实际开发中只需直接引入即可。

综合起来，在降低学习曲线的同时也提升了开发效率。

页面的大致设计：

C

B

A

### 4.1.2 基于OpenCV网络摄像头的实时检测部分设计

人脸检测系统的实时摄像头检测设计如下：

1. 实时的对网络摄像头进行读取
2. 能进行人脸计数
3. 可以进行简单的人脸截图保存
4. 可以部署在物联网设备上，如树莓派

IP 摄 像 头

Tensorflow

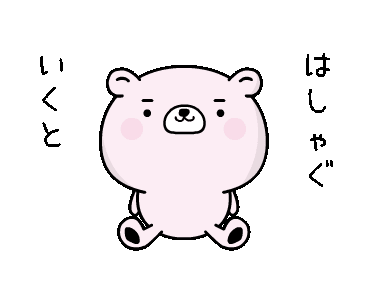
MTCNN

OpenCV

右图为OpenCV实时监控的架构

设计。

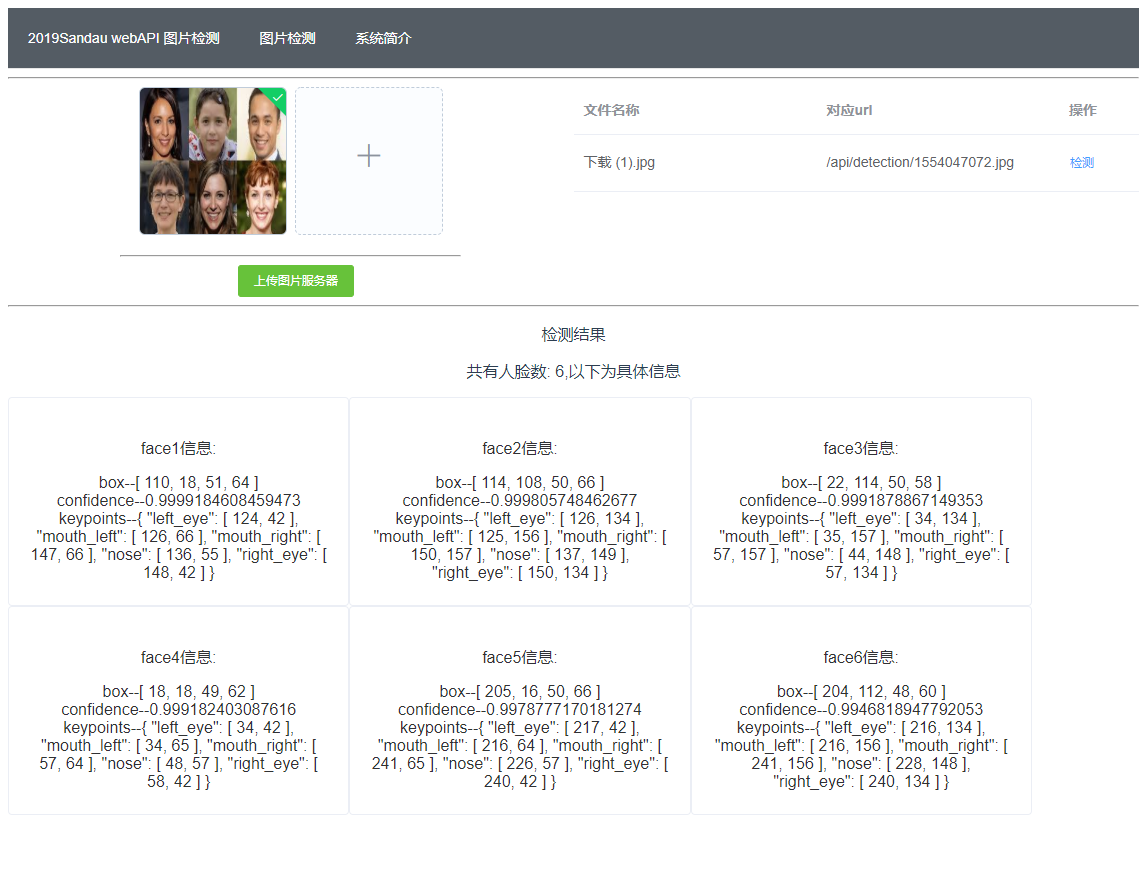
摄像头监测区域



监控屏幕输出

## 4.2. 检测系统实现

### 4.2.1 检测系统的前后端实际实现

前端界面实际实现：：

后端核心接口代码实现：

*import* os  
*import* time  
*import* cv2  
  
*from* flask *import* Flask, jsonify, request  
*from* flask\_cors *import* CORS  
*from* mtcnn.mtcnn *import* MTCNN  
  
app = Flask(\_\_name\_\_)  
UPLOAD\_FOLDER = 'upload'  
app.config['UPLOAD\_FOLDER'] = UPLOAD\_FOLDER  
CORS(app, supports\_credentials=*True*)  
  
  
@app.route('/')  
*def* hello\_world():  
 *return* 'Hello World!'  
  
  
@app.route('/api/upload', methods=['POST'], strict\_slashes=*False*)  
*def* api\_upload():  
 file\_dir = os.path.join(app.config['UPLOAD\_FOLDER'])  
 *if not* os.path.exists(file\_dir):  
 os.makedirs(file\_dir)  
  
 f = request.files['file'] *# 从表单的file字段获取文件，myfile为该表单的name值  
 if* f *is None*:  
  
 *return* jsonify({"code": 1001, "msg": "上传失败"})  
  
 *else*:  
 current\_time = *int*(time.time())  
 new\_filename = *str*(current\_time) + '.jpg'  
 f.save(os.path.join(file\_dir, new\_filename))  
  
 *return* jsonify({"code": 200,  
 "message": "上传成功, 已获取生成的服务器地址",  
 "filename": f.filename,  
 "token": "/api/detection/" + new\_filename})  
  
  
@app.route('/api/detection/<filename>', methods=['GET', 'POST'])  
*def* get\_img(filename):  
 img\_file = os.path.join(UPLOAD\_FOLDER, filename)  
 *if* os.path.isfile(img\_file):  
 img = cv2.imread(img\_file)  
 detector = MTCNN()  
 face\_info = detector.detect\_faces(img)  
 count = *len*(face\_info)  
 *if* count > 0:  
 *return* jsonify({"code": "200",  
 "message": "发现人脸",  
 "count": count,  
 "face\_info": face\_info})  
 *elif* count *is* 0:  
 *return* jsonify({"code": "200",  
 "message": "未检测出人脸"})  
 *else*:  
 *return* jsonify({"code": "500",  
 "message": "服务器内部错误"})  
 *else*:  
 *return* jsonify({"code": "404",  
 "message": "图片不存在或参数错误！"})  
  
  
@app.route('/api-test', methods=['POST', 'GET'])  
*def* api\_server\_test():  
 result = "The backend is connecting!"  
  
 *return* jsonify({"code": "200",  
 "message": result})  
  
  
*if* \_\_name\_\_ == '\_\_main\_\_':  
 app.run()

### 4.2.2 检测系统的OpenCV实时监测实现

主要实现代码

from detection.MtcnnDetector import MtcnnDetector

from detection.detector import Detector

from detection.fcn\_detector import FcnDetector

from train.model import P\_Net, R\_Net, O\_Net

from train import config

import cv2

import os

test\_mode = config.test\_mode

thresh = config.thresh

min\_face\_size = config.min\_face

stride = config.stride

detectors = [None, None, None]

# 模型放置位置

model\_path = ['model/PNet/', 'model/RNet/', 'model/ONet']

batch\_size = config.batches

PNet = FcnDetector(P\_Net, model\_path[0])

detectors[0] = PNet

if test\_mode in ["RNet", "ONet"]:

RNet = Detector(R\_Net, 24, batch\_size[1], model\_path[1])

detectors[1] = RNet

if test\_mode == "ONet":

ONet = Detector(O\_Net, 48, batch\_size[2], model\_path[2])

detectors[2] = ONet

mtcnn\_detector = MtcnnDetector(detectors=detectors, min\_face\_size=min\_face\_size,

stride=stride, threshold=thresh)

out\_path = config.out\_path

if config.input\_mode == '1':

# 选用图片

path = config.test\_dir

# print(path)

for item in os.listdir(path):

img\_path = os.path.join(path, item)

img = cv2.imread(img\_path)

boxes\_c, landmarks = mtcnn\_detector.detect(img)

for i in range(boxes\_c.shape[0]):

bbox = boxes\_c[i, :4]

score = boxes\_c[i, 4]

corpbbox = [int(bbox[0]), int(bbox[1]), int(bbox[2]), int(bbox[3])]

# 画人脸框

cv2.rectangle(img, (corpbbox[0], corpbbox[1]),

(corpbbox[2], corpbbox[3]), (255, 0, 0), 1)

# 判别为人脸的置信度

cv2.putText(img, '{:.2f}'.format(score),

(corpbbox[0], corpbbox[1] - 2),

cv2.FONT\_HERSHEY\_SIMPLEX, 0.5, (0, 0, 255), 2)

# 画关键点

for i in range(landmarks.shape[0]):

for j in range(len(landmarks[i]) // 2):

cv2.circle(img, (int(landmarks[i][2 \* j]), int(int(landmarks[i][2 \* j + 1]))), 2, (0, 0, 255))

cv2.imshow('im', img)

k = cv2.waitKey(0) & 0xFF

if k == 27:

cv2.imwrite(out\_path + item, img)

cv2.destroyAllWindows()

if config.input\_mode == '2':

cap = cv2.VideoCapture(0)

fourcc = cv2.VideoWriter\_fourcc(\*'XVID')

out = cv2.VideoWriter(out\_path + 'out.mp4', fourcc, 10, (1024, 720))

while True:

t1 = cv2.getTickCount()

ret, frame = cap.read()

if ret == True:

boxes\_c, landmarks = mtcnn\_detector.detect(frame)

t2 = cv2.getTickCount()

t = (t2 - t1) / cv2.getTickFrequency()

fps = 1.0 / t

for i in range(boxes\_c.shape[0]):

bbox = boxes\_c[i, :4]

score = boxes\_c[i, 4]

corpbbox = [int(bbox[0]), int(bbox[1]), int(bbox[2]), int(bbox[3])]

# 画人脸框

cv2.rectangle(frame, (corpbbox[0], corpbbox[1]),

(corpbbox[2], corpbbox[3]), (255, 0, 0), 1)

save\_img = cv2.rectangle(frame, (corpbbox[0], corpbbox[1]),

(corpbbox[2], corpbbox[3]), (255, 0, 0), 1)

# 画置信度

cv2.putText(frame, '{:.2f}'.format(score), (corpbbox[0], corpbbox[1] - 2),

cv2.FONT\_HERSHEY\_SIMPLEX, 0.5, (0, 0, 255), 2)

# 人脸截图保存

cv2.imwrite(out\_path + str(i) + ".jpg", save\_img)

croped\_img = cv2.imread(out\_path + str(i) + '.jpg')

crop\_img = croped\_img[corpbbox[1]:2\*corpbbox[1] - corpbbox[3], corpbbox[2]:2\*corpbbox[2] - corpbbox[0]]

cv2.imwrite(out\_path + 'crop -' + str(i) + ".jpg", crop\_img)

# 画fps值

cv2.putText(frame, '{:.4f}'.format(t) + " " + '{:.3f}'.format(fps), (10, 20),

cv2.FONT\_HERSHEY\_SIMPLEX, 0.5, (255, 0, 255), 2)

# 显示人脸数

cv2.putText(frame, 'total faces: %d' % boxes\_c.shape[0], (200, 20),

cv2.FONT\_HERSHEY\_SIMPLEX, 0.5, (100, 112, 255), 2)

# 画关键点

for i in range(landmarks.shape[0]):

for j in range(len(landmarks[i]) // 2):

cv2.circle(frame, (int(landmarks[i][2 \* j]), int(int(landmarks[i][2 \* j + 1]))), 2, (0, 0, 255))

cv2.imshow("result", frame)

if cv2.waitKey(1) & 0xFF == ord('q'):

break

else:

break

cap.release()

out.release()

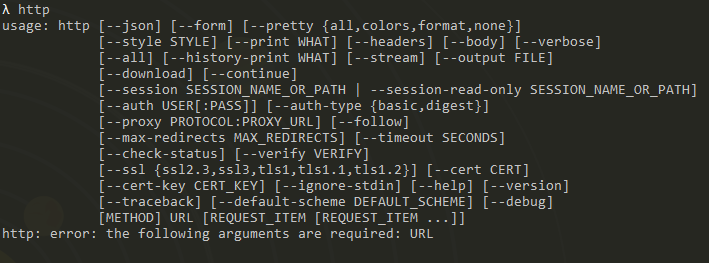
cv2.destroyAllWindows()

### 4.3 系统测试

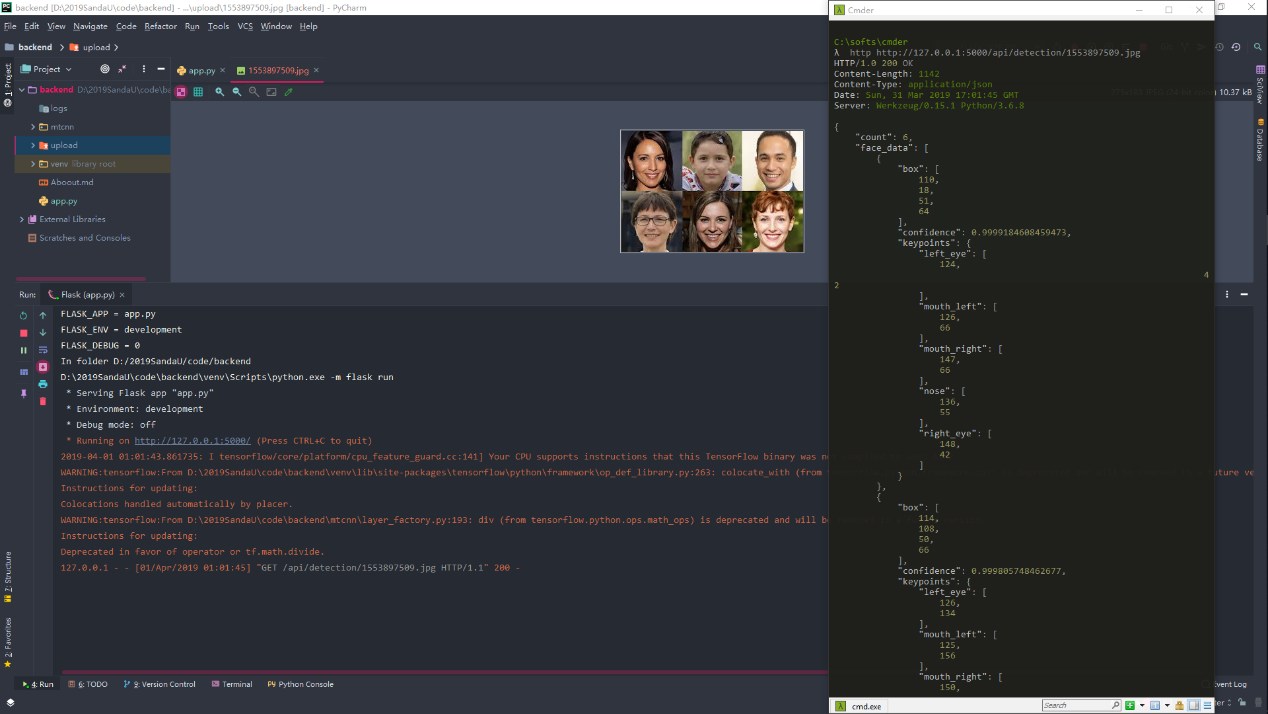
#### 4.3.1 网络摄像头实时检测功能测试

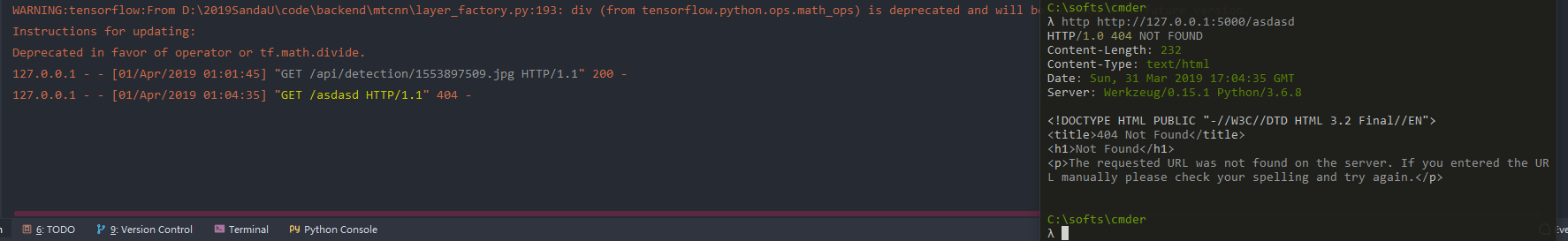
#### 4.3.2 web接口功能测试

测试工具： httpie

 具体使用如下：

测试1：



测试2：

### 4.4 优化

#### 4.4.1 实时检测优化

主要代码优化方式和4.2.2web接口优化一致

#### 4.4.2 web接口优化

通过删减冗余代码，减少过度封装，抽取通用函数在保证核心功能不受影响下，将flask 框架的app.py代码量保持在100行以下，使Python 的哲学思想得以在代码量上体现出来。

#### 4.4.3 架构层面优化

考虑到Flak对于多线程的处理并不是十分的出色，因而在这里使用Nginx 做为服务器，同时进行方向代理，统一管理前后端的请求，还可以提升后端服务的可靠性，当部署多个Flask 实例时，还可以进行负载均衡，保证服务的高可用和高性能。

如下为优化后的前后端架构：

Tensorflow

Img Files

Flask实例

NGINX

用户请求

Flask实例

Flask实例

用户请求

VueJS

MTCNN

用户请求

MTCNN

VueJS

#### 4.2.3 部署优化

由于Python的依赖环境相对较为复杂，即使使用虚拟环境仍然有一些小问题，所以这么为了加快部署，方便迁移，解决环境依赖冲突等问题，使用docker 作为部署工具。这里将Nginx 和VueJS打包成一个容器进行部署和运行，Flask和MTCNN，Tensoflow放在一起打包成容器。

综上，最终的部署简化成为两个docker 容器，这样减少部署问题的同时，也简化了后期的维护。

### 4.3 本章小结

本章通过实时人脸检测系统的设计和实现，MTCNN 模型的调用，功能接口的简单测试，代码上的优化，完成了整体毕设的课题的系统核心。同时在完成系统之后，利用之前在公司的实习时学习到的docker, Nginx等相关技术对整个系统进行了架构优化，运维优化，使整个系统在保持高性能的同时也能保持了高可用。

1. 总结

上一章完成整个系统最后的两个部分，从设计到实现，从实现到测试功能，从测试功能到分析优化，从分析优化到最后的架构和部署优化，完整的呈现了一个软件工程的开发过程，检验了自己所学的技术，利用了之前积累的经验，这样一路过来虽然踉踉跄跄但是还是充满动力呀。

本章便是最后一章，将从总体上来分析整个系统的设计和实现，总结出知识经验。

## 5.1 分析总结

以目标和结果为导向来看，整体上算是完成了开题报告上所立下的种种Flag。但是前端页面过于简单，没有充分的利用VueJS的优势，另一方面便是对自己的能力估计不足，在早期的设计中加入了自己不能解决的Web RTC技术，以试图解决web端实时监测人脸的问题，然后在投入大量时间后还是没有完成和解决这个问题，以至于导致了后面论文初稿不能尽早提交，打乱了后续的其他计划，另一方面，对于前期所学的机器学习相关知识没有趁热打铁，乘胜追击，将项目中所应用的MTCNN模型学习和理解的很透彻。

在实际编程中，遇到了太多的盲点，如根本不了解OpenCV，需要从零开始学习，TensorFlow的水平还停留在Hello Word 水平，Python web 开发上也不能吃透，前后端交互时，对于数据的处理也不能游刃有余，这些种种的迹象表明自己在一开始的课题选择和设计上过于盲目自信和不能真正的看清自己的分量，编程的基本功太差，不能熟练运用所学的数据结构和算法等等。

## 5.2 展望

从毕业课题开题，设计到最后的勉强完成，一路上暴露出的问题正是今后要解决的问题，只有这样大学四年才没有白过，毕业课题设计才具有它的实际意义。

也正是从这个最后一次的作业中，接触到了一些新的东西，希望自己可以继续坚持下去。在剩下的最后几个月，能把握住机会，最后努力一下。

参考文献

1. "What is REST API". RESTful API Tutorial. Retrieved
2. TensorFlow document：<https://tensorflow.google.cn/api_docs/python/tf>
3. Flask document：<http://flask.pocoo.org/>
4. Vue JS document：https://vuejs.org/
5. OpenCV document：https://opencv.org/
6. Axios document：https://github.com/axios/axios
7. ElementUI document: http://element-cn.eleme.io/#/zh-CN
8. Aurélien Géron:《Hands-On Machine Learning with Scikit-Learn and TensorFlow》 2017-1-25
9. Goodfellow / Yoshua Bengio / Aaron Courville：《Deep Learning》，2016-11-11
10. . Zhang，Z. Zhang ，Z. Li，Y. Qiao ：《Joint Face Detection and Alignment using Multi-task Cascaded Convolutional Networks》，2016，ISSN-1070-9908
11. Traveler: 《Vue.js，何謂漸進式框架？》 2017
12. Runoob python 简介：<http://www.runoob.com/python/python-intro.html>
13. unoob JavaScript简介：<http://www.runoob.com/js/js-intro.html>
14. VueJs 官网：<https://cn.vuejs.org/v2/guide/>
15. 梁路宏,艾海舟,徐光佑,张钹:《人脸检测研究综述》，2002,中图分类号：TP391.4
16. 梁路宏,艾海舟,肖习攀,叶航军,徐光佑,张钹：《基于模板匹配与支持矢量机的人脸检测》,2002
17. Gunn S.R.:《Support vector machines for classification and regression, Technical Report, Image Speech and Intelligent Systems Research group, University of Southampton》 1997. <http://www.isis.ecs.soton.ac.uk/resources/svminfo/svm.pdf>
18. Osuna E, Freund R, Girosi F:《Training support vector machines: an application to face detection. In Proc. of CVPR, Puerto Rico》,1997,130-136
19. <http://cs.haifa.ac.il/hagit/courses/ip/Lectures/Ip11_MultiscaleRepx4.pdf>
20. Yann LeCun, Yoshua Bengio:《Convolational Networks for Images, Speech, and Time-Series》,http://yann.lecun.com/exdb/publis/pdf/lecun-bengio-95a.pdf
21. Ross Girshick, Jeff Donahue, Trevor Darrell, Jitendra Malik:《Rich feature hierarchies for accurate object detection and semantic segmentation》，2013，arXiv:1311.2524 [cs.CV]
22. Ross Girshick： The IEEE International Conference on Computer Vision (ICCV), 2015, pp. 1440-1448
23. Shaoqing Ren, Kaiming He, Ross Girshick, Jian Sun: 《Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks》，Advances in Neural Information Processing Systems 28 (NIPS 2015)
24. <https://pjreddie.com/darknet/yolo/>
25. <https://kpzhang93.github.io/MTCNN_face_detection_alignment/index.html>
26. Convolutional Neural Networks (LeNet) - Deep Learning 0.1 documentation. Deep Learning 0.1. LISA Lab. [31 August 2013] ：http://deeplearning.net/tutorial/lenet.html
27. Convolutional Neural Network. [2014-09-16].：<http://ufldl.stanford.edu/tutorial/supervised/ConvolutionalNeuralNetwork/>
28. Krizhevsky, A.，Sutskever, I.，Hinton, G. E.: 《Imagenet classification with deep convolutional neural networks》，2012, 1，Advances in Neural Information Processing Systems. 1097–1105
29. Graham, Benjamin. Fractional Max-Pooling. 2014-12-18. arXiv:1412.6071 [cs.CV].
30. Springenberg, Jost Tobias, Dosovitskiy, Alexey. Brox, Thomas. Riedmiller, Martin. Striving for Simplicity: The All Convolutional Net. 2014-12-21. arXiv:1412.6806 [cs.LG].
31. <https://zh.wikipedia.org/wiki/File:Max_pooling.png>
32. K. He, X. Zhang, S. Ren, J. Sun: “Delving deep into rectifiers: Surpassing human-level performance on imagenet classification,” in IEEE International Conference on Computer Vision, 2015, pp. 1026-1034
33. Yang, Shuo and Luo, Ping and Loy, Chen Change and Tang, Xiaoou: WIDER FACE: A Face Detection Benchmark: http://mmlab.ie.cuhk.edu.hk/projects/WIDERFace/
34. Y. Sun, X. Wang, and X. Tang. Deep Convolutional Network Cascade for Facial Point Detection. In Proceedings of IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2013.
35. Ziwei Liu, Ping Luo, Xiaogang Wang, Xiaoou Tang: Large-scale CelebFaces Attributes (CelebA) Dataset: <http://mmlab.ie.cuhk.edu.hk/projects/CelebA.html>
36. <https://webrtc.org/>
37. http://element-cn.eleme.io/#/zh-CN

# 致 谢

首先，能完成这次毕业设计感性的是指导老师张老师，从选题到最后的论文编写，一直都对我谆谆教导，循循善诱，让我坚定了选择机器学习相关的课题，并坚持做了下去，最后虽然勉勉强强的完成了系统，但是还是收货了很多，技术上的，知识上的，沟通上的，受益匪浅。