



华南理工大学
South China University of Technology

课程设计报告书

题目： 饭堂人群密度检测

学 院	电子与信息学院
专 业	信息工程
姓名学号	李新鹏 201430251448
	陈威鹏 201430250175
	谢博 201430253213
	黄坚荣 201430251127
	谈裕锦 201430252452
指导教师	邢晓芬
课程编号	135165
课程学分	2.0
起始日期	2017.01.18

<p>教师评语</p>	<p>教师签名： 日期：</p>
<p>成绩评定</p>	
<p>备注</p>	

目录

饭堂人群密度检测.....	4
一、选题背景.....	4
二、方案论证(设计理念)	4
1、硬件摄像（谢博、黄坚荣）	4
1.1 方案 1.....	5
1.2 方案 2.....	5
1.3 方案优劣选择.....	5
2、网页终端（谈裕锦）	5
2.1 设计原理	5
2.2 方案选择	5
2.3 实现的功能	6
2.4 系统的安全性	6
2.5 数据的完整性	6
2.6 应用的运行环境	7
2.7 性能	7
3、核心算法（陈威鹏、李新鹏）	7
3.1 设计要求.....	7
3.2 手动提取特征回归人数.....	7
3.3 深度卷积神经网络.....	8
三、过程论述.....	10
1、硬件摄像（谢博、黄坚荣）	11
1.1 方案 1.....	11
1.2 方案 2.....	12
2、应用终端（谈裕锦）	12
2.1 项目架构.....	12
2.2 组件结构.....	14
2.3 页面切换与路由跳转.....	16
3、核心算法（陈威鹏、李新鹏）	18
3.1 手动提取特征.....	18
3.2 MSCNN	19

四、结果分析.....	24
1、硬件摄像（谢博、黄坚荣）	24
1.1 方案 1.....	24
1.2 方案 2.....	25
2、应用终端（谈裕锦）	26
3、核心算法（陈威鹏、李新鹏）	29
手动提取特征.....	29
MSCNN	30
五、课程设计总结.....	32
1、硬件摄像（谢博、黄坚荣）	33
2、应用终端（谈裕锦）	33
3、核心算法（陈威鹏、李新鹏）	34
手动提取特征.....	34
MSCNN	34
参考文献.....	36
六、附录.....	37
1、硬件摄像（谢博、黄坚荣）	37
1.1 方案 1.....	37
1.2 方案 2.....	37
2、应用终端（谈裕锦）	38
3、核心算法（陈威鹏、李新鹏）	38
手动提取特征应用代码.....	38
MSCNN 应用代码	39

饭堂人群密度检测

一、选题背景

在这个人工智能快速发展的时代，智能交通、智能机器人等人工智能化产品不断出现。作为人工智能的重要分支，计算机视觉起到了重要作用。它通过一系列的图像处理与模式识别手段，模拟现实世界中的生物视觉系统，并对捕捉到的图像进一步处理以满足不同领域的需求。

本项目从生活细节出发，将计算机视觉应用在饭堂人数检测上，结合软硬件设施：算法基于深度卷积神经网络模型，硬件基于树莓派 RaspberryPi3 Model B，终端为Web APP 或者公众号平台，学生可以通过终端获取饭堂人数密度热力图。一方面有助于师生根据饭堂人数避开就餐高峰期，另一方面将人群密度分布数据提供给饭堂设计人员，方便做出更好的设计。此外，本项目运用计算机视觉、模式识别领域的相关理论同样对火车站、地铁站、广场等人群容易集中的场所进行人群密度估计和人数统计。

二、方案论证(设计理念)

本项目需要实现的功能为：通过手机终端实时获取饭堂人群密度信息，并具备核心算法估计人数和密度信息。初步架构为：使用智能硬件树莓派，控制摄像头获取视频或者图片并传输至服务器平台，利用服务器平台的优秀性能实现基于深度卷积神经网络的人群密度估计，若有用户使用终端请求数据，则返回人群密度数据，利用热力图或者其余指标呈现饭堂人群情况。下图为硬软件设施器件框图，本项目的工作围绕此图展开。



1、硬件摄像（谢博、黄坚荣）

本次实验硬件通过组员手中原有的树莓派及购入的摄像头进行摄像，作为一

款信用卡大小的超小型电脑，安装好摄像头之后进行更新、设置便可以进行拍照，录影。

1.1 方案 1

Crontab 定时任务每分钟执行一次拍照和上传，树莓派通过 `raspistill` 命令拍照，Python 程序上传到七牛云，服务器可通过命令行工具 `qshell` 下载到服务端。

1.2 方案 2

Crontab 定时任务每分钟执行一次拍照和上传，树莓派通过 `raspistill` 命令拍照，编写基于 `http` 协议的 Python 程序将图片上传到服务器端。

1.3 方案优劣选择

由于七牛云是图床，云端除了上传图片不能做其他事情，得重新下载到服务端，所以采取步骤，费时较少的方案 2。

2、网页终端（谈裕锦）

2.1 设计原理

① 在开发模式上，采用 **MVVM** 的架构模式，实现数据、处理逻辑与视图的分离，方便实现项目的组件化与模块化，大大提高开发效率与系统的可维护性。

② 在开发工具上，采用 `angular 4 + TypeScript`、`jQuery`、`bootstrap` 等框架或库进行网页的编码实现，以 `angular 4` 作为主要的开发框架（**MVVM** 模式），以 `bootstrap`（依赖于 `JQuery`）作为主要的 UI 库，实现响应式设计（PC、移动端兼容）。采用 `webpack` 自动化构建工具（不使用 `angular 4` 官方提供的 `@angular/cli` 构建工具，而是自己搭建环境，这样有利于系统架构的灵活性，例如按项目的变化进行性能优化、自己配置一些参数等等），打包、压缩并优化整个项目。

③ 在服务器方面，有两个分支，一个是自己申请的阿里云服务器，项目完成后将上传到该服务器中就可以在外网访问；另一个是通过 `git` 将项目同步到远程仓库，且利用 `Github Pages` 将网页上传到 `Github` 提供的服务器中，作为备用。

④ 在应用功能的设计上，该 web 应用功能模块主要有两个：监控系统和记录中心。这些模块中数据部分的实时更新是采用 `ajax` 轮询的方式实现的，而图片的实时更新是每隔 20s 重新从服务器加载一次图片，实现网页上显示的图片的更新（监控摄像头通过程序实现 60s 给服务器上传一张新图片）。

2.2 方案选择

按照网站的类型来看，有两种方案可供选择：

① 多页面应用：只需要单独做好每一个页面，再使用 `<a>` 标签让各个页面链接在一起。优点：实现的难度较为简单，不硬性需要把重点放在开发模式上（组

件化、模块化)；导入框架和库的方式较为简单；具有很好的 SEO。缺点：用户体验不友好，例如从监控系统的界面切换到记录中心的界面时网页会刷新。打包配置较为复杂，入口文件有多个。

② 单页面应用 (SPA)：旧的超链接页面浏览模型给用户带来了不和谐的体验，而原则允许数据驱动时间在一个页面内创建，并让页面内容在需要的时候更新。这意味着应用似乎可以运行得更加流畅，乃至到达可仿真桌面与本地资源接口的地步。故 web 应用程序只有一个页面，各模块只充当该页面的局部部分，页面局部变化时浏览器是异步刷新的，即不会触发我们平时的刷新按钮，模块直接加载到该局部，整个页面在此过程中不会变成白屏。优点：用户体验很友好；打包项目时比较简单，只需要配置一个入口文件。缺点：SEO 非常差，很不友好；实现难度较高，需要考虑并配置很多多页面应用不需考虑的东西，如：路由机制，组件化编程，各组件或模块间的相互等等。

这里选择单页面应用，原因有两个：①该 web 应用程序是一个饭堂流量管理系统，理论上属于后台管理系统的类型，故 SEO 对于该系统而言并不重要；②该 web 应用程序变化的部分都是局部（监控系统和记录中心的部分）。

2.3 实现的功能

在应用功能的设计上，该 web 应用功能模块主要有两个：

① 监控系统和记录中心。监控系统中有当前日期、该日期下饭堂早午晚的人流量统计数据、监控图片等等，这些日期、数据和图片都会根据监控摄像头拍到的图像进行实时更新。

②记录中心有一个表格，记录了当前和以往日期的饭堂早午晚的人流量统计数据，所以我们可以从中查询到一些历史记录。

2.4 系统的安全性

该 web 应用程序含有一个登录系统，要使用该系统首先需要登录，直接通过一些特殊路由来访问该系统的一些模块都是会被拦截禁止的。

2.5 数据的完整性

该 web 应用程序记录的数据很全面，包含了一天中饭堂从开放到关闭的所有时间段（早上 06:30-09:00、中午 10:30-13:00、晚上 16:30-21:30）。而且，该程序还有一个记录中心记录了当前和以往日期的饭堂早午晚的人流量统计数据，我们可以从中查询到一些历史记录。

2.6 应用的运行环境

在开发调试中，我采用的运行环境是叫做 `webpack-dev-server` 的一个调试服务器，参数可以自己按需求配置。而打包应用时，我自己配置了两个环境，分为 `dev`（开发环境，该环境下主要为了调试和效率，故排除掉生产环境下要做的步骤）和 `prod`（生产环境，该环境意味着项目已经完成并准备上传到真实的服务器上，需要对代码进行压缩、混淆、删除注释等等）。两个环境下生产的文件都可以先通过调试服务器来打开预先观察其效果、正确性，确认系统无可用、正确后，再上传到服务器上，这样就使得整个开发流程即灵活高效，又稳定正确。

2.7 性能

由于该 `web` 应用程序用到的框架和库也不少，而且 `bootstrap` 在移动端的性能比较一般，这就使得程序的性能优化异常重要。所以在项目的架构上，对 `webpack` 打包项目上进行了不少的优化。例如：用 `CommonChunksPlugin` 提取各个被打包模块代码的公共部分。用 `DllPlugin` 把解决浏览器兼容性的一些文件（如：`core.js`、`polyfills`、`zone.js`、`ts-helpers` 等等），`angular 4`、`jquery`、`bootstrap` 等一系列框架或库文件打包到一个 `dll` 文件中，与业务代码分开打包编译，实现 `webpack` 的音速编译，提高调试服务器的运行速度，减少程序的 `http` 请求次数。总的来说，优化后程序的性能有了很大的提升，效果非常好。

3、核心算法（陈威鹏、李新鹏）

3.1 设计要求

核心算法应当解决人群密度检测两方面的问题：人数统计和密度图绘制，在此之上，算法追求克服摄像头偏差、人群拥挤等环境情况影响，更准确地形象地将人群信息返回给用户。

3.2 手动提取特征回归人数

解决人群计数问题的传统的做法是手动设计并提取各种特征(`Hand-crafted Features`)，然后再基于这些特征训练一个线性或非线性函数来回归人头数。此类算法大多三个步骤：1) 前景分割；2) 特征提取；3) 人数回归。接下来将对这三个步骤分别做介绍。

1) 前景分割

前景（人群）分割的目的是将人群从图像中分割出来便于后面的特征提取，

分割性能的好坏直接关系的最终的计数精度,因此这是限制传统算法性能的一个重要因素。常用的分割算法有:光流法(Optical Flow)、混合动态纹理(Mixture of Dynamic Textures)、小波分析(Wavelets)等。

2) 特征提取

在完成前景分割之后,紧接着就是从分割得到的前景(人群)提取各种不同的低层特征(Low-level Features),常用的特征有:人群面积和周长(Area and Perimeter of Crowd Mask)、边的数量(Edge Count)、边的方向(Edge Orientation)、纹理特征(Texture Features)、闵可夫斯基维度(Minkowski Dimension)等。

3) 人数回归

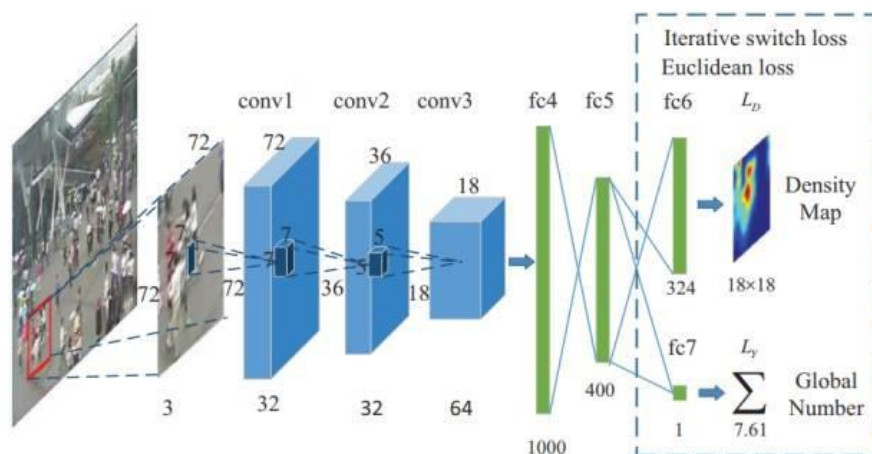
该步骤的目的是将上一步提取到的特征回归到图像中的人数,回归可以是简单的线性回归,也可以用复杂的非线性回归。常用的回归方法有:线性回归(Linear Regression)、分段线性回归(Piece-wise Linear Regression)、脊回归(Ridge Regression)、高斯过程回归(Gaussian Process Regression)等。

而对于单张图像而言没有运动信息,那么人群分割就显得非常困难,因此此类算法一般直接从整张图像或者其子区域提取特征,然后再计算图像中人群数量。

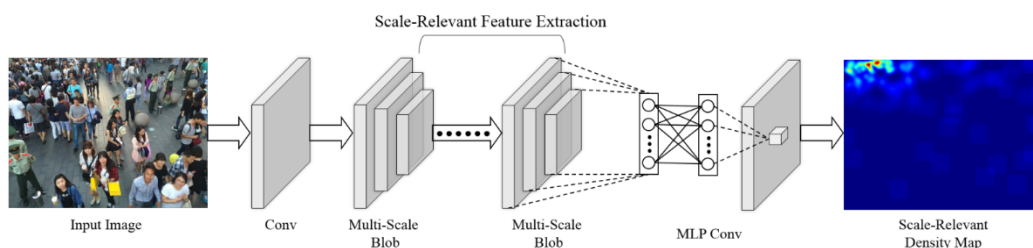
通过查询网上资料,最终确定为利用 python 以及 python-opencv 简单地实现算法,达到人脸检测、人头检测等基本目的。

3.3 深度卷积神经网络

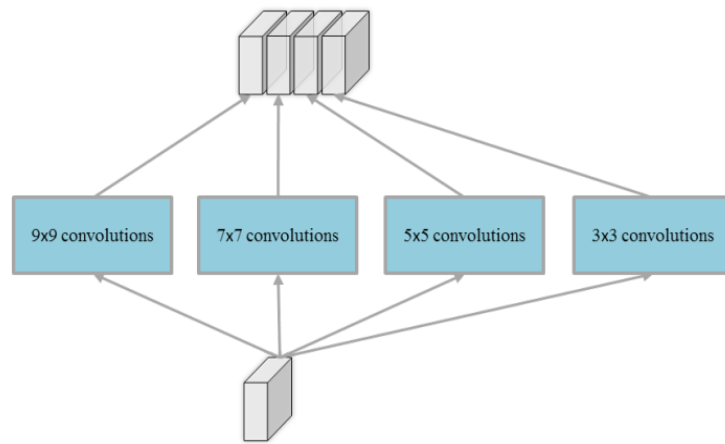
基于深度学习的人群分析技术,不再采用人为定义特征的方式去判断目标是否为“人”。通过使用大量数据训练模型,计算机可以自行学习并抽象出人群的概念,并有效提取出针对人群分析有效的特征。这种针对人群整体的分析方法有效克服了传统的基于人的个体分析方法所无法解决的大规模复杂场景的适应性问题,并且人数统计精确度达到 95% 以上。下图为针对人群密度估计的卷积神经网络结构图:



目前人群密度估计的深度卷积神经网络有多种，包括 single-CNN、multi-column CNN、multi-network CNN、multi-scale CNN、adapting CNN 等。single-CNN 采用单列神经网络估计人群密度图，但是受限于获取图片多尺度信息而难以达到更好地效果；multi-column CNN 采用多列卷积神经网络，每列网络使用不同尺寸的卷积核，更充分地获取图片尺度信息并且达到良好效果，此外 multi-network CNN 采用深浅两列网络以解决空间分辨率问题，但是 Multi-column/network 有两个缺点：一点是需要预训练单列网络以达到全局优化，为端到端训练增加复杂度，另一点是引入更多的参数，消耗更大的计算资源。而 multi-scale CNN 采用单列网络+多尺度卷积核的方式，下图为 MSCNN 的神经网络结构图：



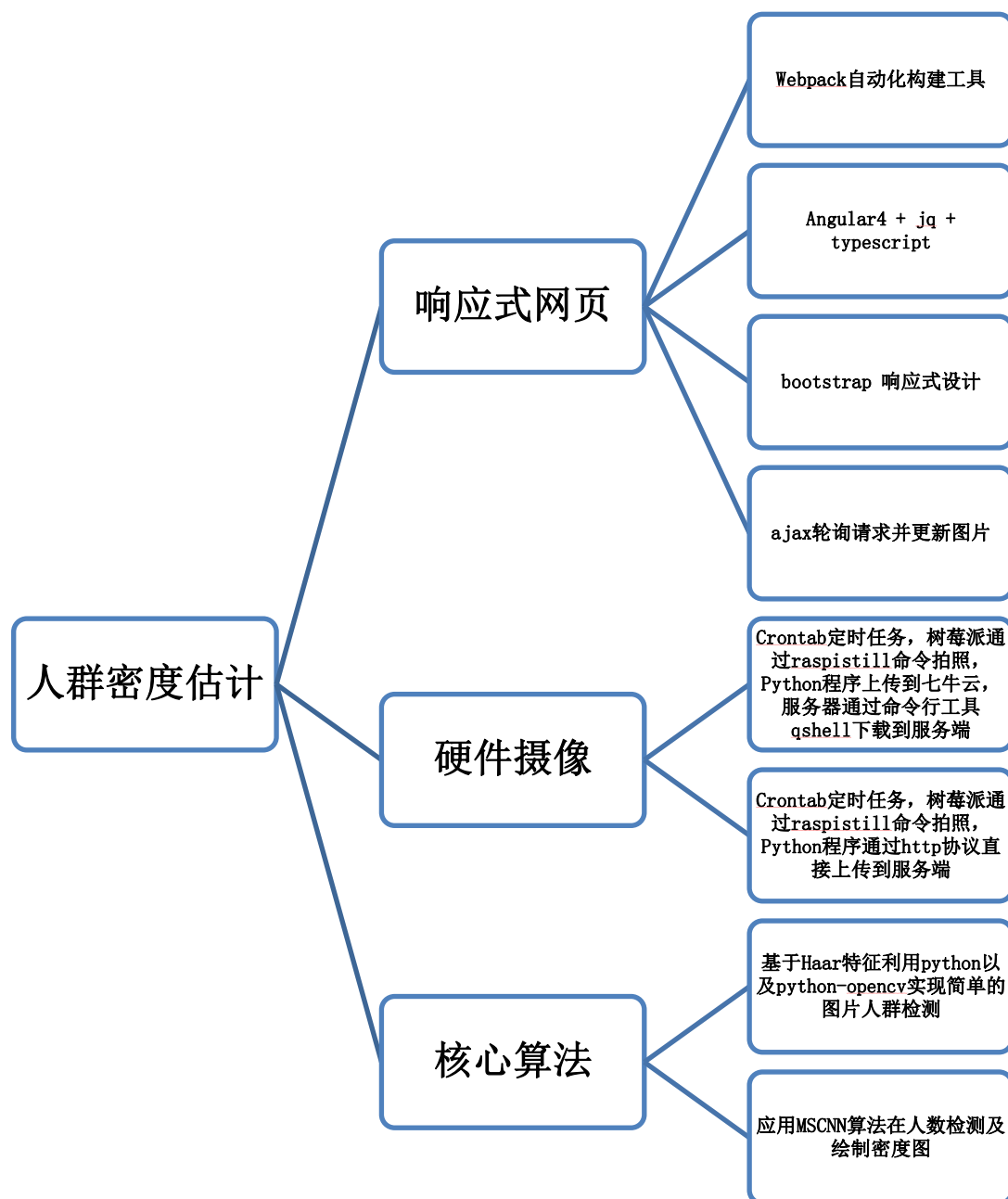
其中每层网络分为 9*9、7*7、5*5、3*3 四种尺度卷积核，如下图所示：



MSCNN 既克服图片尺度变化问题，又克服多元网络的参数过多、全局优化困难等问题，故在本项目最终采用 MSCNN 作为本项目的核心算法。而 ACNN 充分利用摄像头角度信息训练网络，但是由于精力有限没有进一步实践 ACNN 算法。

三、过程论述

经过近一个月的努力，团队基本完成人群密度检测的软硬件建设以及核心算法，整体流程为：使用树莓派拍摄图片，再通过 http 协议传至服务器端，每分钟拍摄一次并刷新服务器图片；服务器接收图片，并使用 MSCNN 算法估计人数以及绘制人群密度热力图，保存信息图片以供终端获取；手机等终端访问网站主页，可以实时获取服务器最新的人群信息图片，得知饭堂人群信息，实现框架如下：



1、硬件摄像（谢博、黄坚荣）

1.1 方案 1

①编写 python 上传程序 test.py

程序将根据秘钥及仓库名将目录下文件传输到七牛云端。并删去目录下同名文件。

②编写 shell 脚本

程序将操控树莓派摄像头拍照，存于特定目录下，并执行 Python 程序 test.py。

③设置定时任务

使用 crontab -e 指令编写树莓派定时任务，使树莓派每分钟执行一次脚本 take_photo.sh。

1.2 方案 2

①编写 Python 程序客户端程序 Client.py

程序根据 http 协议将本地文件上传到服务器端。服务器端判断上传的文件的大小，后缀名等决定是否接收。可以通过 <http://120.25.59.185:8081/pic> 查看上传的图片。

②编写 shell 脚本

程序将操控树莓派摄像头拍照，存于特定目录下，并执行 Python 程序 Client.py。

③运行脚本

程序将在本地目录下生成 current_photo.jpg，并在服务器端生成 1.jpg，可以通过 <http://120.25.59.185:8081/pic> 查看文件。

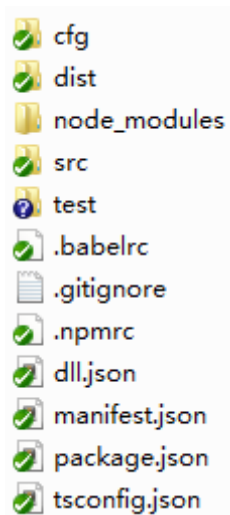
④设置树莓派定时任务，

使用 crontab -e 指令编写树莓派定时任务，使树莓派每分钟执行一次脚本 take_photo.sh，更新图片。

2、应用终端（谈裕锦）

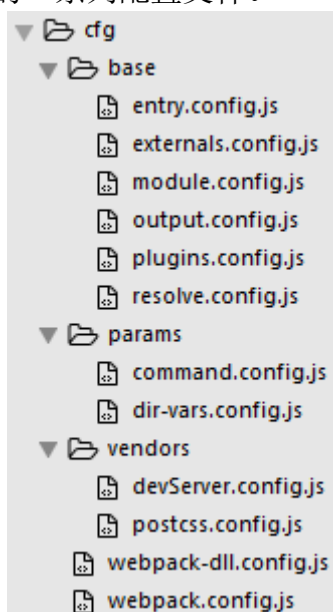
2.1 项目架构

整个项目架构使用 canteen-management-system，框图如下所示：

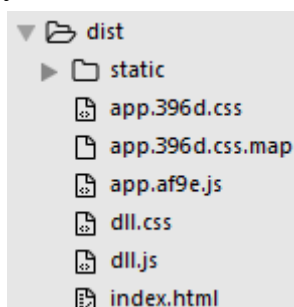


具体每个文件夹内容如下：

cfg 文件夹：存放 webpack 的一系列配置文件。



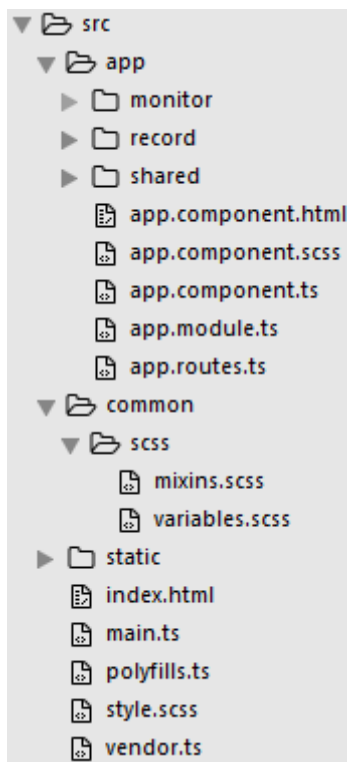
dist：存放打包后的生成文件。



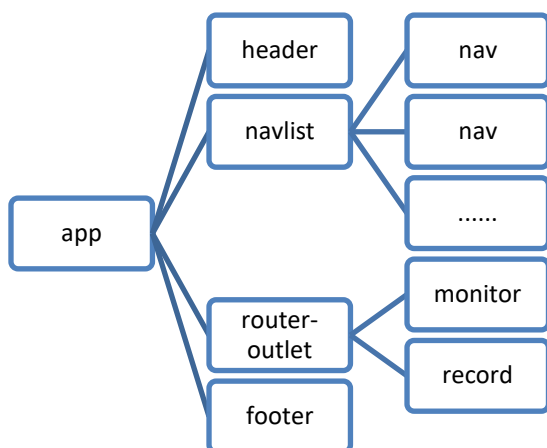
node_modules：存放用 npm 下载的各个开发模块，这么文件夹是通过命令行下载模块时自动生成的。

test：存放项目的测试文件。

src: 编写和存放业务代码的文件夹，该文件夹中的文件会通过 **webpack** 打包后，生成最终的文件到 **dist** 目录下。



2.2 组件结构



app 是程序的根组件，它由 **header** 组件、**navlist** 组件、**monitor** 组件或者 **record** 组件（**router-outlet** 意味着此处的视图或根据路由配置来确定到底采用哪一个组件）、**footer** 组件。而 **navlist** 组件由若干个 **nav** 组件组成（当前版本由两个 **nav** 组件，分别链接到监控系统和记录中心）。

这个组件结构图有助于我们划分好组件和模块，一个个攻破，实现好局部之后一起整合，从而使程序健壮又可维护。

实现出来，各组件的效果图如下：

header:



navlist:



nav:



router-outlet (monitor 和 record) :



2.3 页面切换与路由跳转

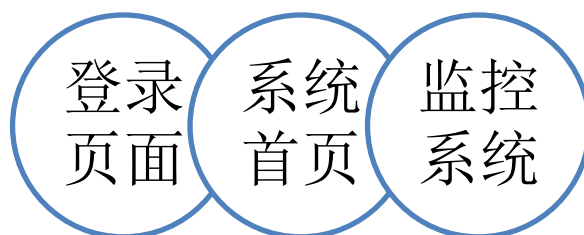


图 页面切换

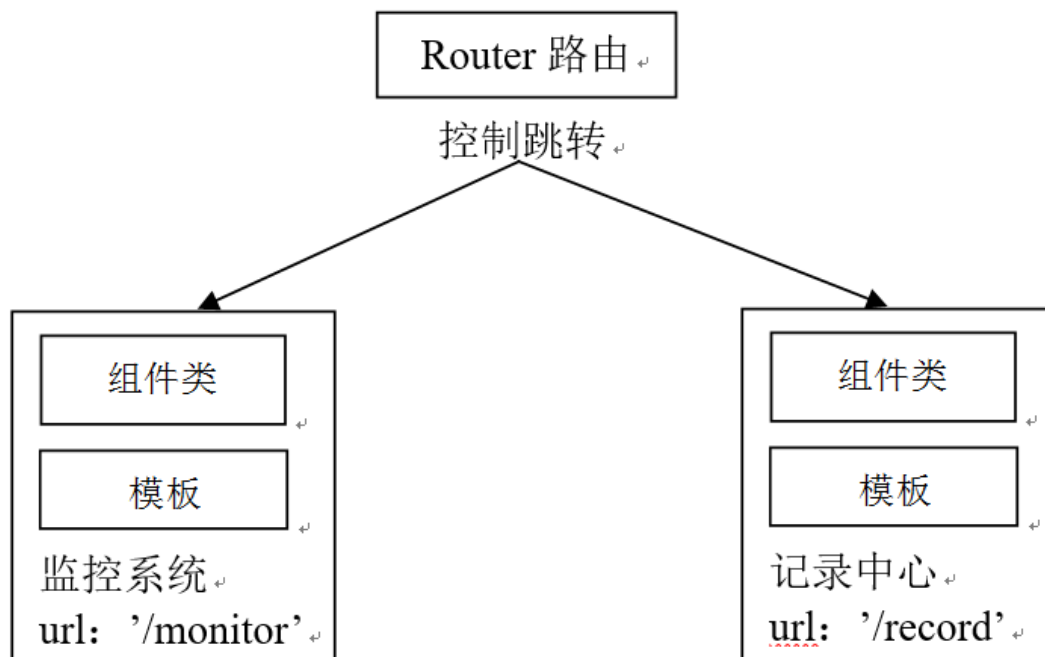
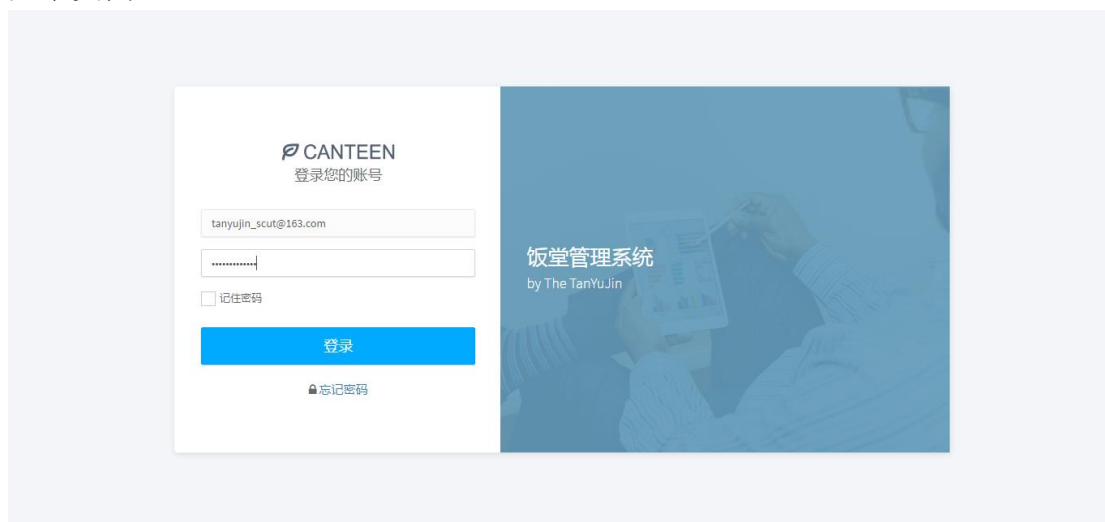


图 路由跳转

这两个图有助于我们理清页面跳转和路由跳转时视图的关系和变化，使我们编写网页的时候思路更为清晰，很好地掌握整体。这样在编程实现的时候，就会很清楚该用哪种方式使页面和组件模块联系在一起了。

登录页面：



系统首页（路由部分默认切换到监控系统视图）：



3、核心算法（陈威鹏、李新鹏）

3.1 手动提取特征

（1）查询资料，

通过网上资料学习了解传统人群检测算法，学习 python 以及 opencv 语言。OpenCV 人脸检测可简单概述为以下几个方面：

①Haar：Haar 特征也称 Haarlike 特征，是一种简单且高效的图像特征，其基于矩形区域相似的强度差异 Haar 小波。Haar 特征的特点为：高类的可变性；低类的可变性；而向局部的强度差异；多尺度不变性；计算效率高。②级联：OpenCV 在物体检测上使用的是基于 haar 特征的级联表，级联将人脸检测过程拆分成了多个过程。在每一个图像小块中只进行一次粗略的测试。如果测试通过，接下来进行更详细的细节测试，依次重复。检测算法中有 30 至 50 个这种过程或者级联，只有在所有过程成功后才会最终识别到人脸。③分类器：人们采用样本的 haar 特征进行分类器的训练，从而得到一个级联的 boost 分类器，实现时就是数据组成的 XML 文件，OpenCV 也自带了一些已经训练好的包括人眼、人脸和人体的分类器（位于 OpenCV 安装目录\data\haarcascades 目录下，分类器是 XML 类型的文件）。

（2）安装 python 以及 OpenCV

通过官网下载 python 2.7，再利用 cmd 命令完成环境搭建和 OpenCV 的下载安装，如下图所示（已安装完成）：

```
C:\Users\Roc-Chan.Chan-PC>python -m pip install opencv-python
Requirement already satisfied: opencv-python in c:\python27\lib\site-packages
Requirement already satisfied: numpy>=1.11.1 in c:\python27\lib\site-packages (from opencv-python)
```

(3) 学习编写代码

代码参考附录。

3.2 MSCNN

(1) 查询资料

寻找前沿的博客和论文，分别研究过 CrowNet、MCNN、MSCNN、ACNN 等算法，各算法的神经网络结构如下图：

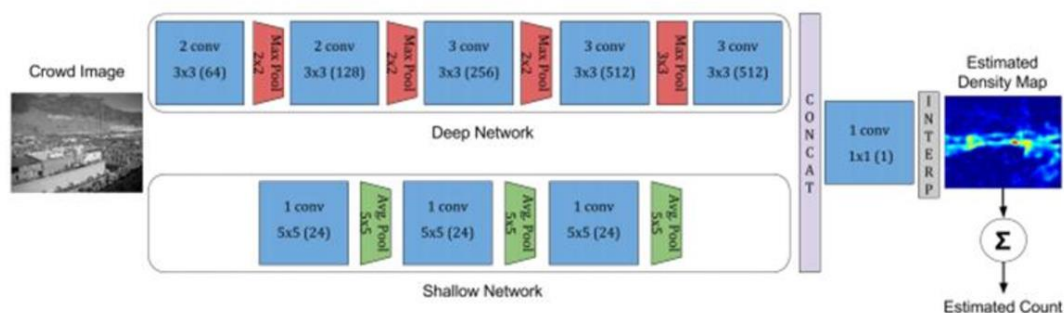


图 CrowNet

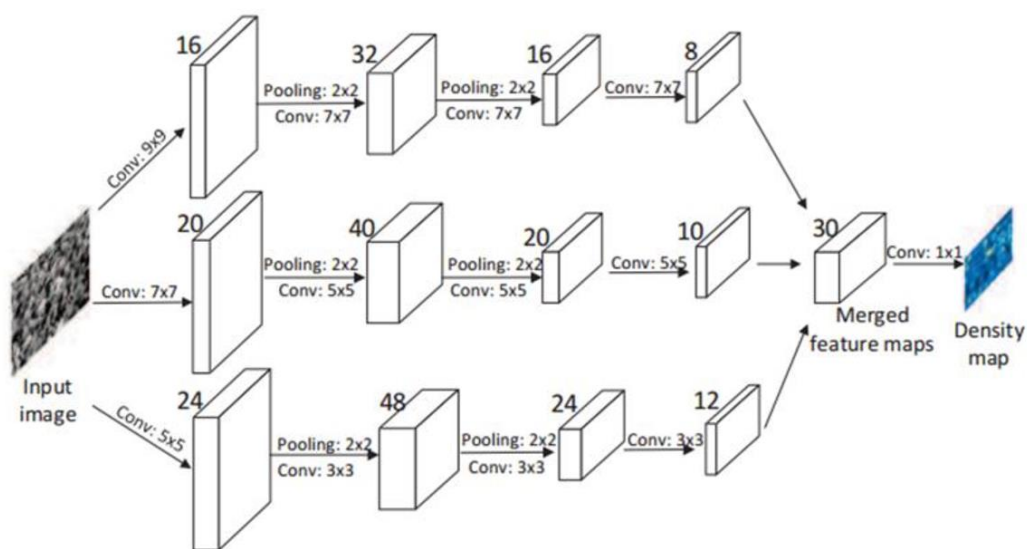


图 MCNN

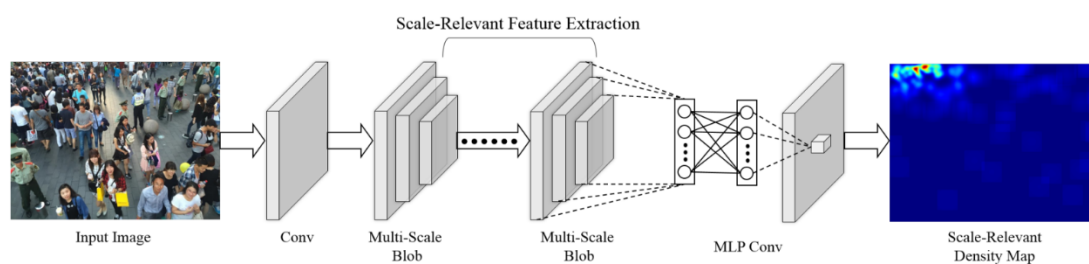


图 MSCNN

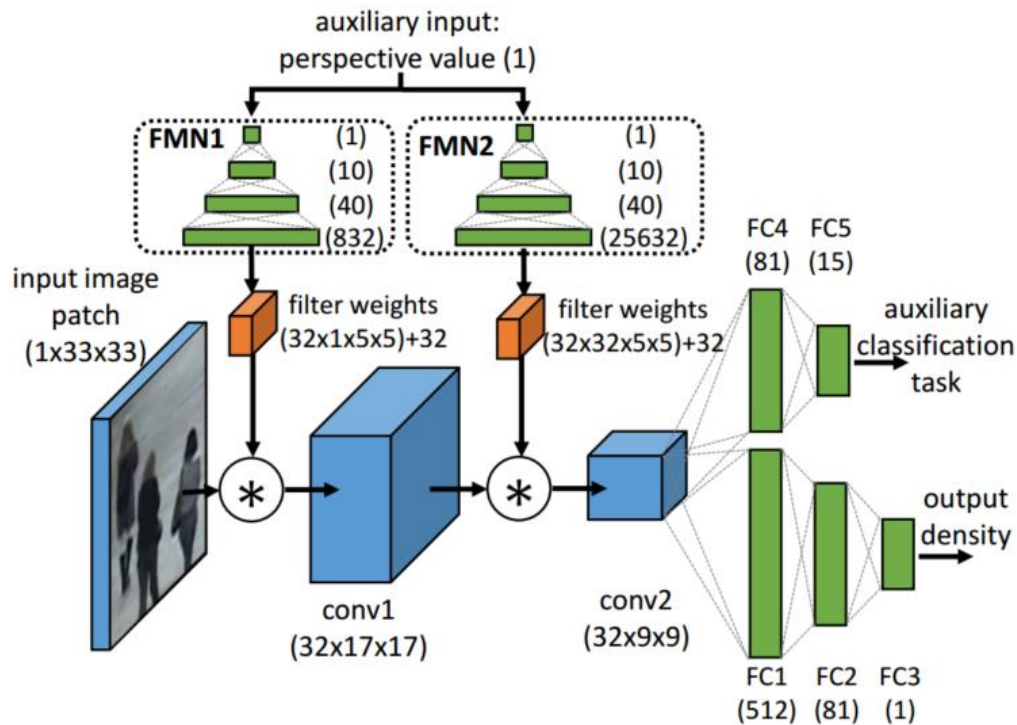


图 ACNN

通过对比几种网络结构，明白一些道理：多尺寸卷积核有助于适应不同图片间尺度差异，自适应高斯核编辑密度图有助于适应同一图片因视角原因产生的尺度差异，多元网络或者多种 Loss 结合对整体优化有帮助，充分利用摄像头姿态信息解决视角偏差问题。前面方案选择中对比过三类网络的性能，MSCNN 兼得减少网络复杂度，估计效果良好的优势，最终选为核心算法，在完成 MSCNN 后可以将 ACNN 思想搬移过来。

(2) 初步学习深度学习框架&卷积神经网络

tensorflow 社区活跃以及架构简洁，故从学习 tensorflow 以及训练 mnist 开始，逐步了解卷积神经网络，为使用 caffe 框架和理解 MSCNN 内部结构奠定基础。过程如下：

- 理解数据类型 Variable、Tensor、Graph，了解 tensorflow 基本操作比如 placeholder()、Session()等
- 预处理数据，利用 scipy、numpy、matplotlib 处理图像数据，查看数据维度

以及转换为适应的维度，为减轻训练的计算量，使用迭代器分批获取数据

- 理解卷积神经网络结构，包括 conv 层、relu 层、pooling 层、fc 层，以及层与层之间数据的转换，卷积层的 filter 可以看作 feature identifiers
- 亲自搭建卷积神经网络训练 mnist 手写数字识别，使用 dropout 层、滑动平均模型、学习率指数衰减率、正则化等手段优化网络得到较好的准确率
- 使用 TensorBoard 可视化 loss 及各类参数，使用 Saver 类保存模型参数

以下为训练 mnist 而创建的卷积神经网络以及 loss 可视化结果：

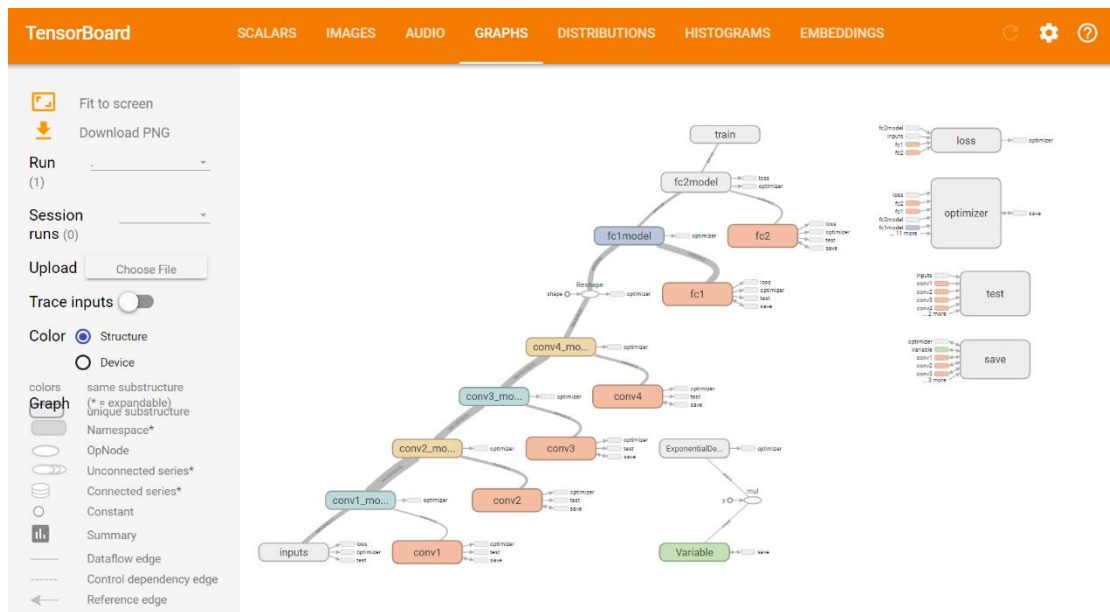


图 CNN 结构

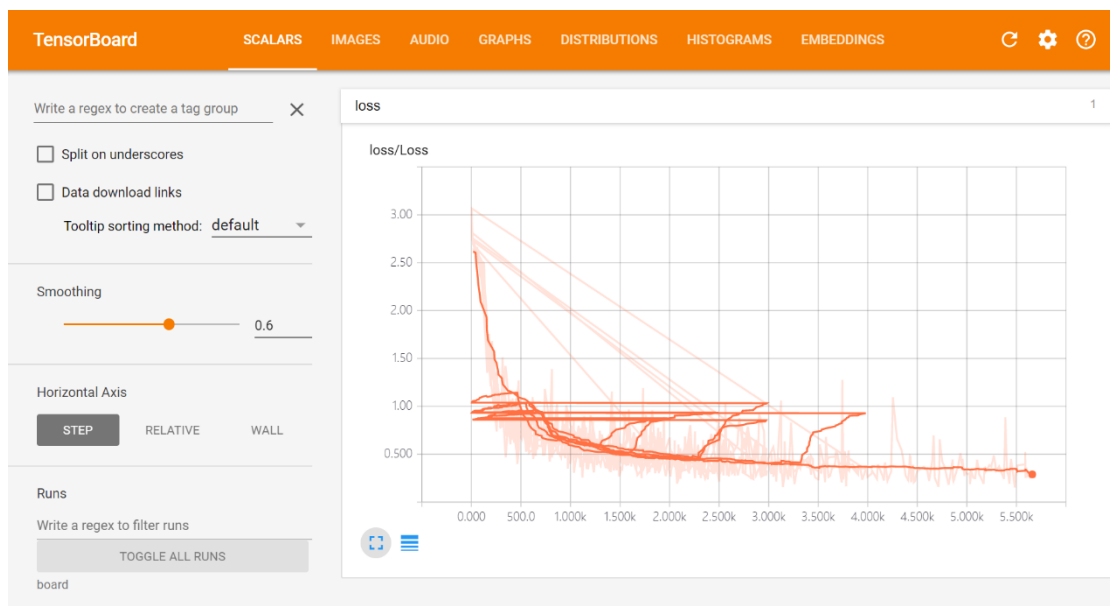
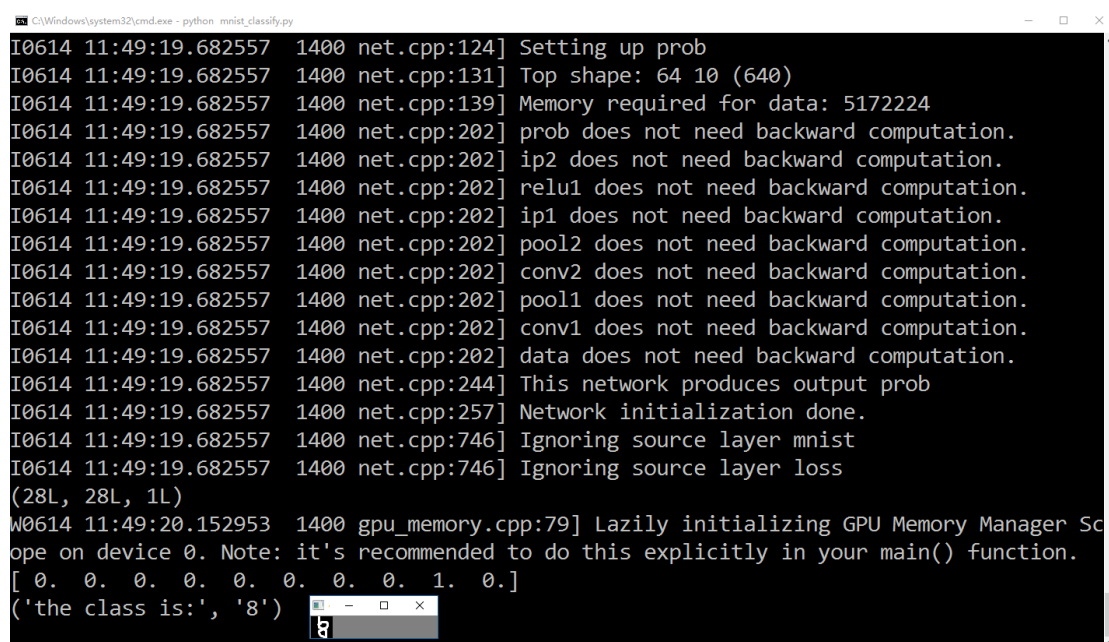


图 Loss 变化曲线

(3) 搭建 caffe 框架，熟悉 MSCNN

为方便工作，选择在 windows 下编译 caffe 框架，成功安装好并使用 caffe 框架，可以做到针对性修改源码再编译，编写 bat、sh、py 文件训练以及测试框架，结合 caffe 与其他模块实现拍摄照片->人群密度检测->保存信息图片并制作为视频。过程如下：

- 寻找 caffe-windows 源码，利用 VS2015 编译出 caffe.exe 及其他可执行文件，加入 python 接口和 matlab 接口再编译，方便结合使用 python 或者 matlab 的科学计算库。由于 caffe 使用在 linux 系统，所以在 windows 下编译需要解决诸多问题，经过多次尝试和修改之后成功编译好并使用，同时对编译和 caffe 框架更加熟悉。
- 通过训练 mnist 手写数字识别熟悉 caffe 框架，由于之前具备 tensorflow 基础，因此很快理解 caffe 的使用：理解 lmdb、leveldb 存储方式，熟悉 blob、layer、net、lr、loss、caffemodel 等关键词的含义，利用 python 编写 prototxt 文件，以及结合 opencv、matplotlib 将模型应用到实际检测中。下图为利用训练好的 caffemodel 测试自行制作的手写数字：



```
C:\Windows\system32\cmd.exe - python mnist_classify.py
I0614 11:49:19.682557 1400 net.cpp:124] Setting up prob
I0614 11:49:19.682557 1400 net.cpp:131] Top shape: 64 10 (640)
I0614 11:49:19.682557 1400 net.cpp:139] Memory required for data: 5172224
I0614 11:49:19.682557 1400 net.cpp:202] prob does not need backward computation.
I0614 11:49:19.682557 1400 net.cpp:202] ip2 does not need backward computation.
I0614 11:49:19.682557 1400 net.cpp:202] relu1 does not need backward computation.
I0614 11:49:19.682557 1400 net.cpp:202] ip1 does not need backward computation.
I0614 11:49:19.682557 1400 net.cpp:202] pool2 does not need backward computation.
I0614 11:49:19.682557 1400 net.cpp:202] conv2 does not need backward computation.
I0614 11:49:19.682557 1400 net.cpp:202] pool1 does not need backward computation.
I0614 11:49:19.682557 1400 net.cpp:202] conv1 does not need backward computation.
I0614 11:49:19.682557 1400 net.cpp:202] data does not need backward computation.
I0614 11:49:19.682557 1400 net.cpp:244] This network produces output prob
I0614 11:49:19.682557 1400 net.cpp:257] Network initialization done.
I0614 11:49:19.682557 1400 net.cpp:746] Ignoring source layer mnist
I0614 11:49:19.682557 1400 net.cpp:746] Ignoring source layer loss
(28L, 28L, 1L)
W0614 11:49:20.152953 1400 gpu_memory.cpp:79] Lazily initializing GPU Memory Manager Sc
ope on device 0. Note: it's recommended to do this explicitly in your main() function.
[ 0.  0.  0.  0.  0.  0.  0.  0.  1.  0.]
('the class is:', '8')
```

图 识别手写数字

- 熟悉 MSCNN 框架，结合论文研究框架的细节，修改部分源码编译出合适的

caffe 来运行 MSCNN，最终编写好估计代码，利用训练好的 caffemodel 预测拍摄图片中的人群密度。其中不再自行改进网络的原因有：目前整个网络较深且训练数据量大，身边缺乏足够的计算资源；MSCNN 当前网络已经达到世界前沿水准，且网络结构严谨，本人知识有限，短期内难以修改出性能飞跃的效果。在实践过程中，使用笔记本电脑内置摄像头拍摄饭堂图片，速率设置为 2 秒/张，再使用 MSCNN 预测人群密度并绘制出热力图，最后转换为视频方便得到人群流动信息。以下为部分人群密度预测照片：

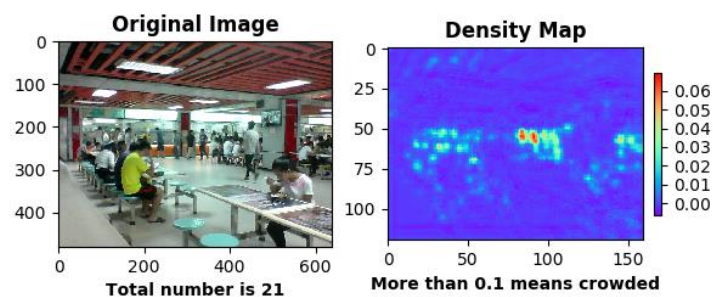


图 饭堂人群密度预测

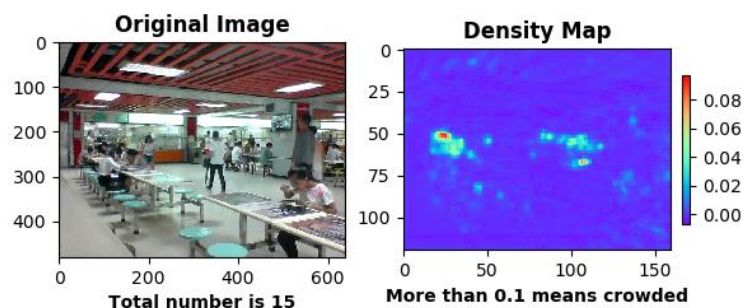


图 饭堂人群密度预测

- 结尾较为可惜的是，本项目租用廉价的服务器，性能甚至无法安装 **caffe** 框架，故没有在服务器上使用 **MSCNN** 算法预测人群密度，但可以保证非能力因素造成。同时，因为缺乏计算资源而无法尝试修改算法，在前进的道路有所止步，但根据实践结果，经过思考和探讨，得出一些修改 **MSCNN** 的想法，将在结果分析中详细说明。

四、结果分析

1、硬件摄像（谢博、黄坚荣）

1.1 方案 1

运行 shell 脚本

本地目录下有图片生成：

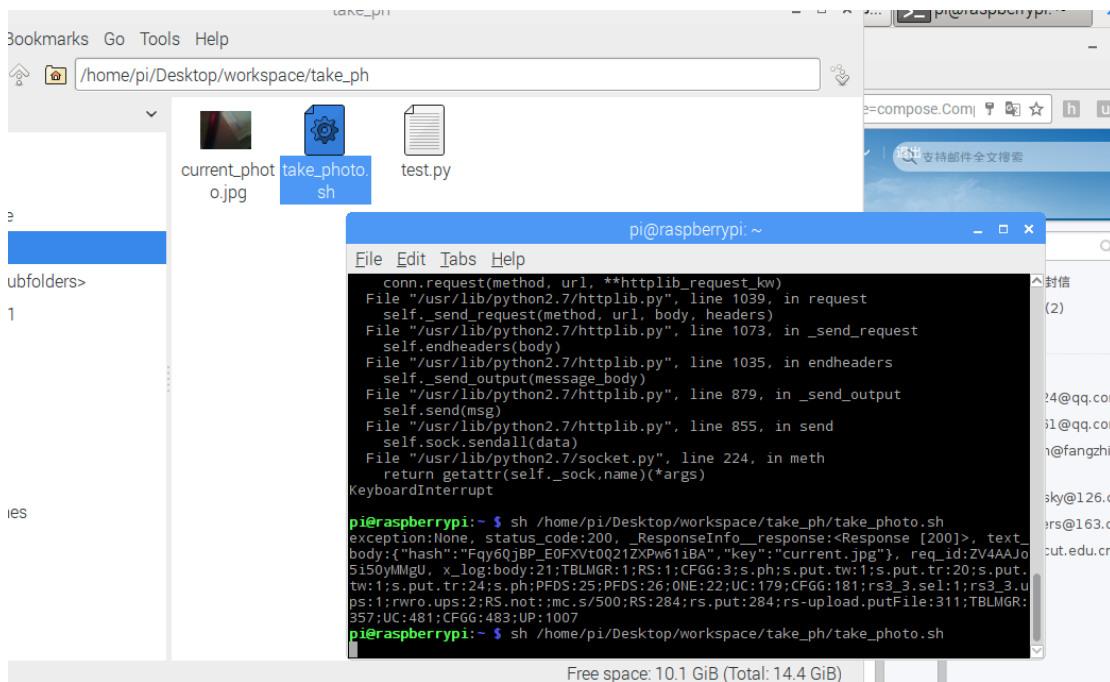


图 本地目录

七牛云端有图片上传：

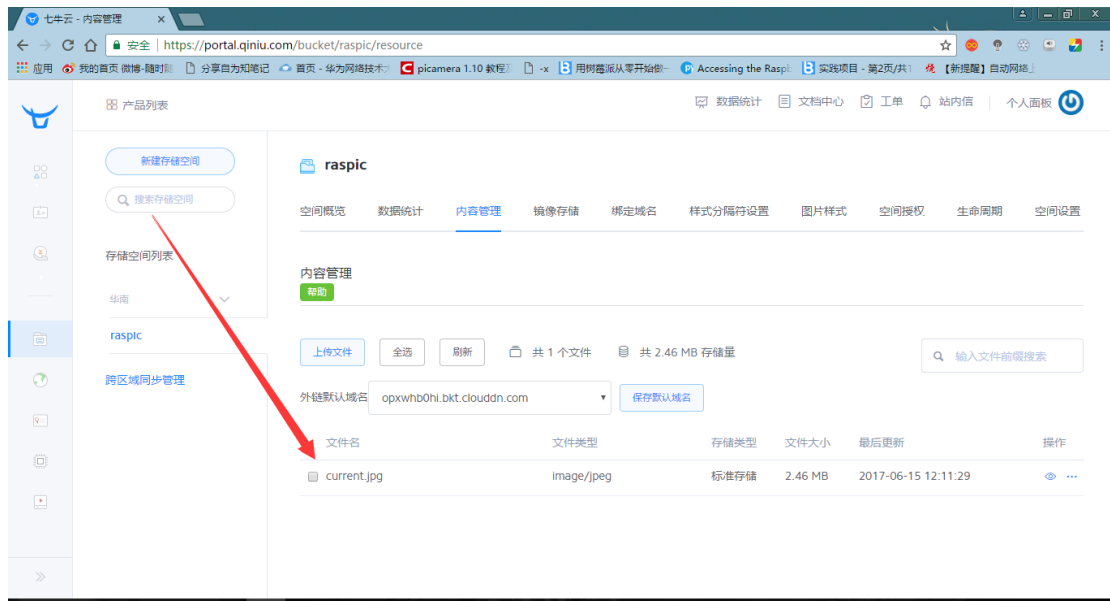


图 七牛云

1.2 方案 2

运行脚本 `take_photo.sh`, 如下图, 程序将在本地目录下生成 `current_photo.jpg`, 并在服务器端生成 `1.jpg`, 且可以通过 <http://120.25.59.185:8081/pic> 查看文件。

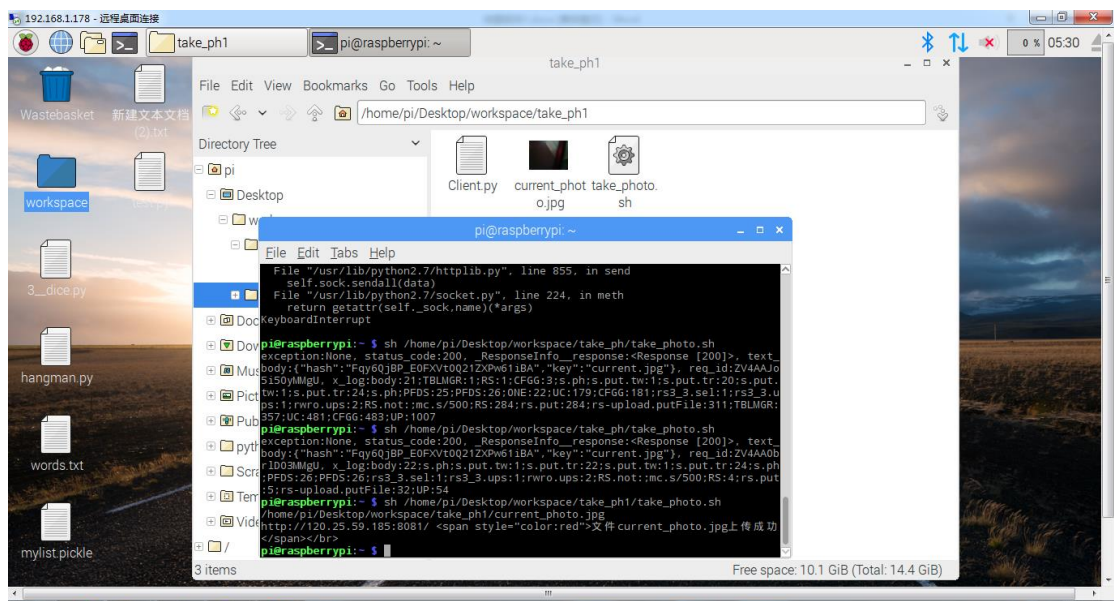


图 本地目录

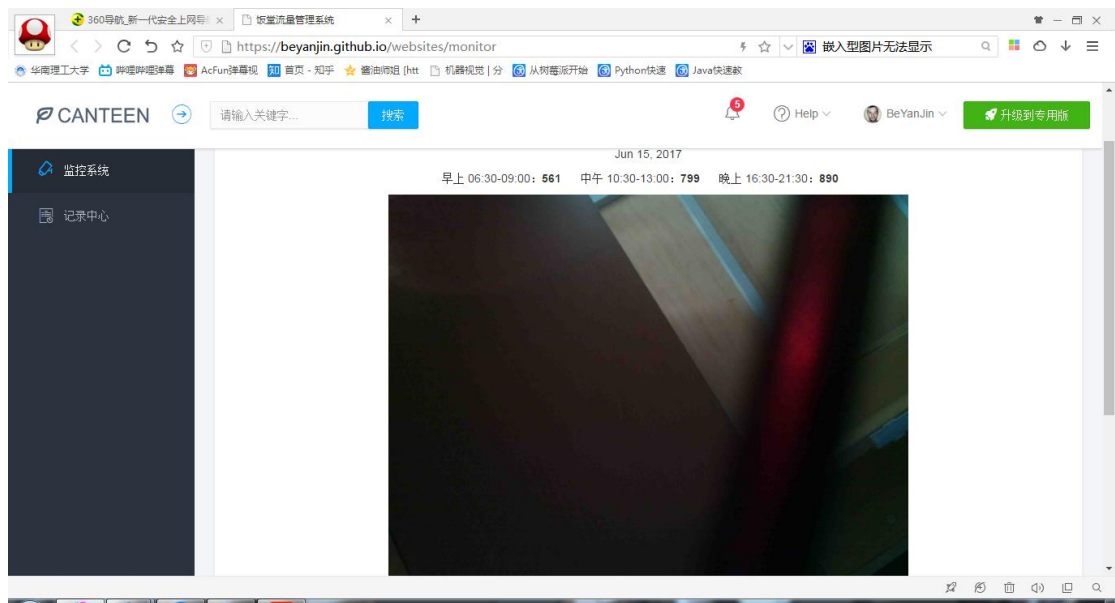


图 网页显示

2、应用终端（谈裕锦）

下面是成果展示，需要下载项目源代码的话可以访问本人 github 中的 websites 项目（<https://github.com/BeYanJin/websites>），若想直接访问网页，自己观看效果并进行操作的话，可以访问下面网页地址（<https://beyanjin.github.io/websites/>）：



图 PC 端效果 1

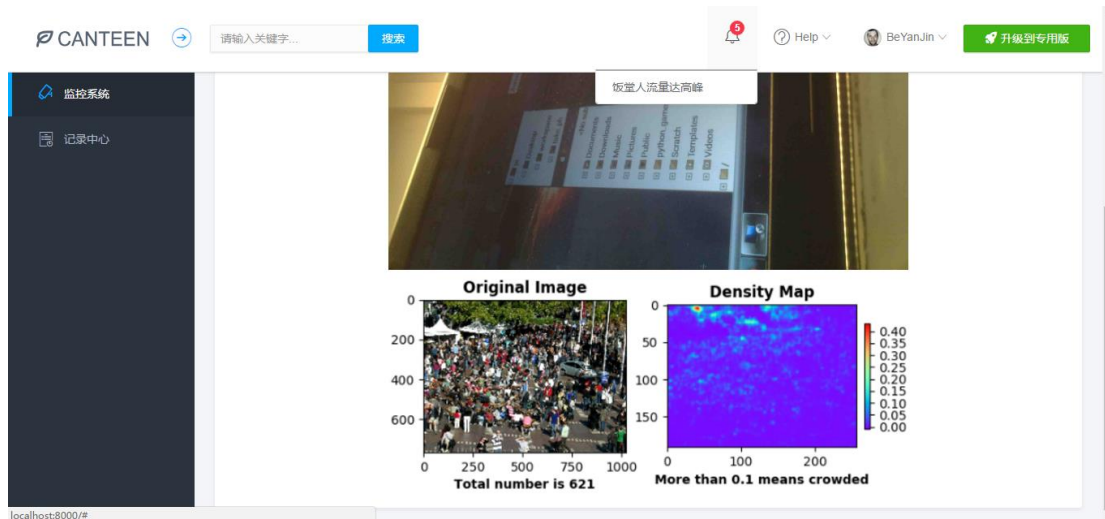


图 PC 端效果 2



图 PC 端效果 3

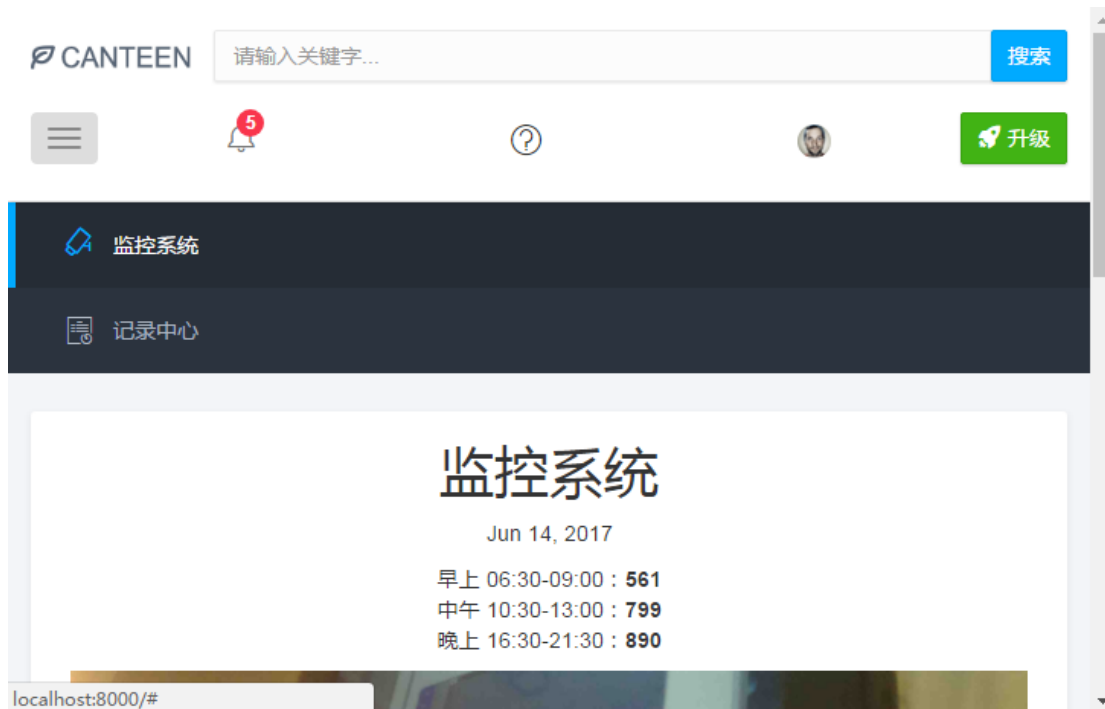


图 PC 端效果 4

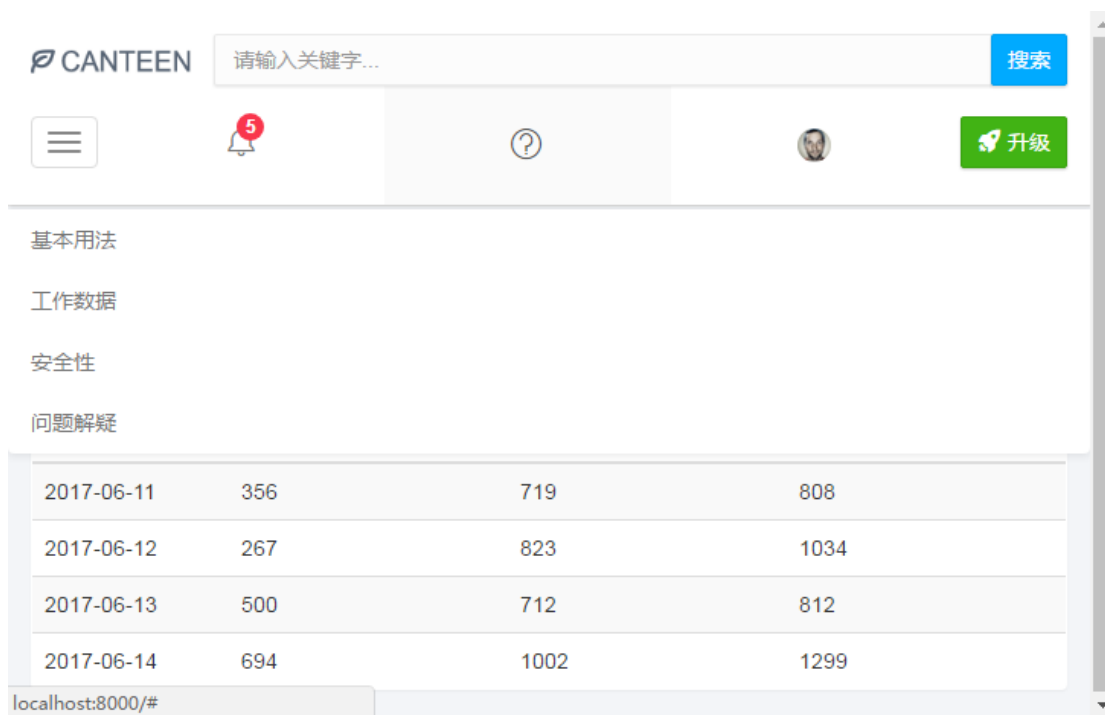


图 PC 端效果 5

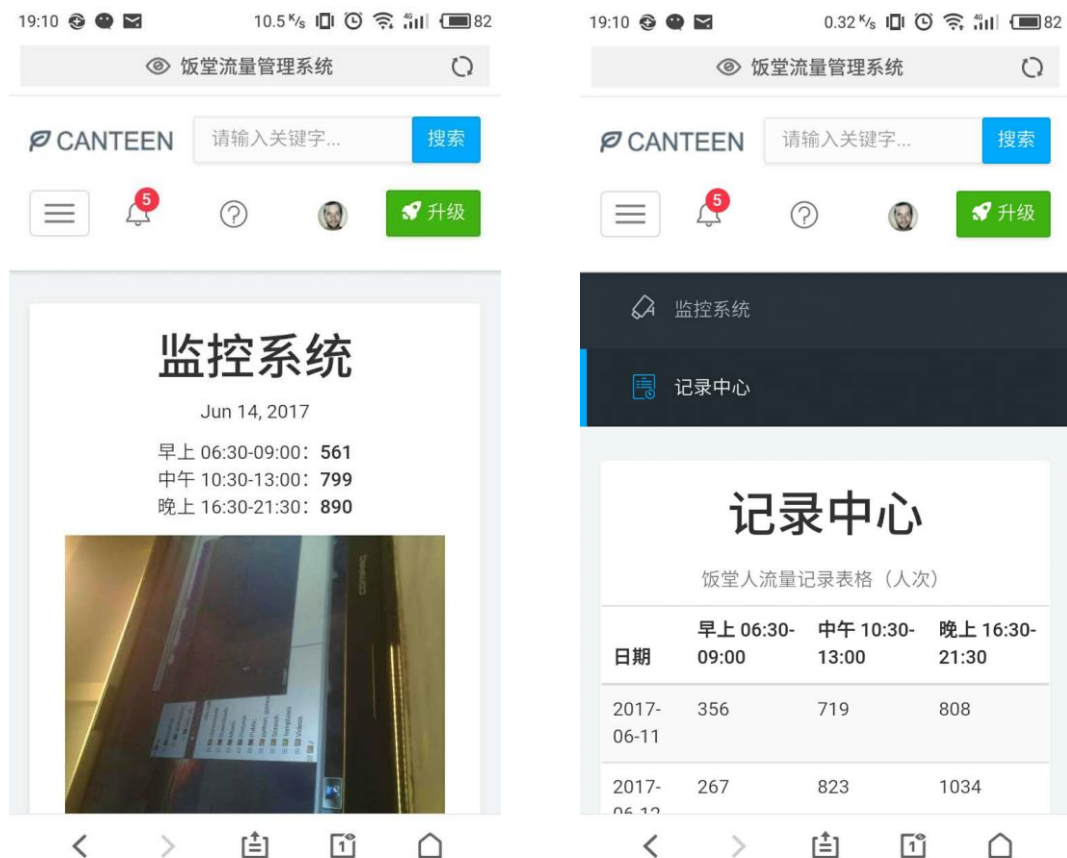


图 移动端效果图 1

3、核心算法（陈威鹏、李新鹏）

手动提取特征

如下图所示是检测算法实现的结果（以人脸检测为例），能正确地识别人脸

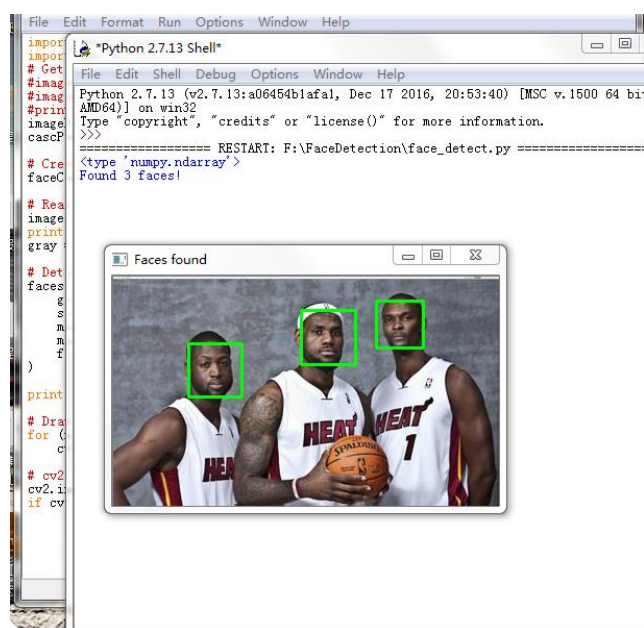


图 人脸检测 1

如下图所示是借助摄像头检测人脸算法实现的结果。

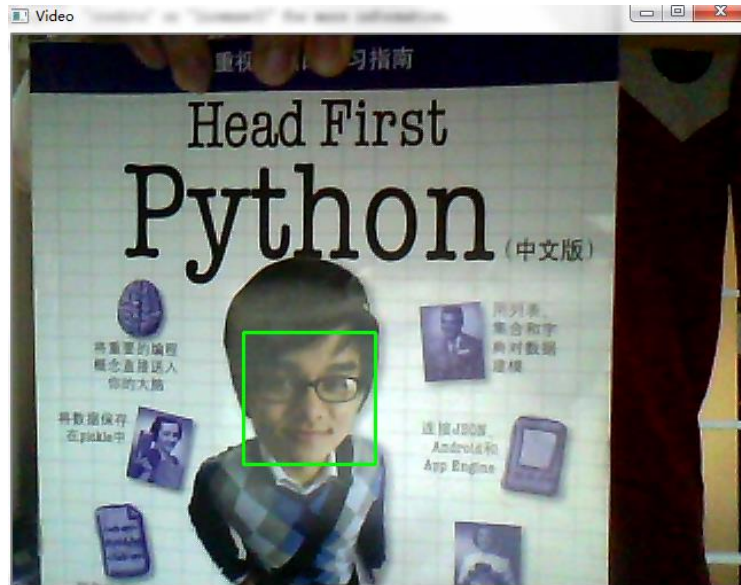


图 人脸检测 2

MSCNN

MSCNN 当属世界前沿算法，具备各方面较好的性能，下图为饭堂人群密度检测的图片之一，无论是人数还是密度图分布都反映真实情况，让人一目了然。

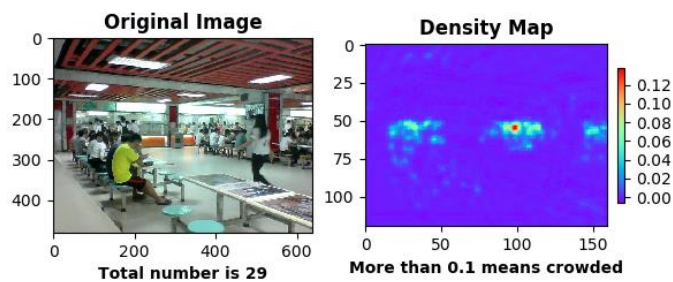


图 饭堂人群密度检测

但是应用在工程中有些许改进的地方。第一点是摄像头角度问题，由于拍摄角度不同人头大小有差异，得到的密度图仍无法反映出真实拥挤程度。下面图中各处拥挤程度一致，由于角度问题在密度图反映出左上方比右下方更为拥挤：

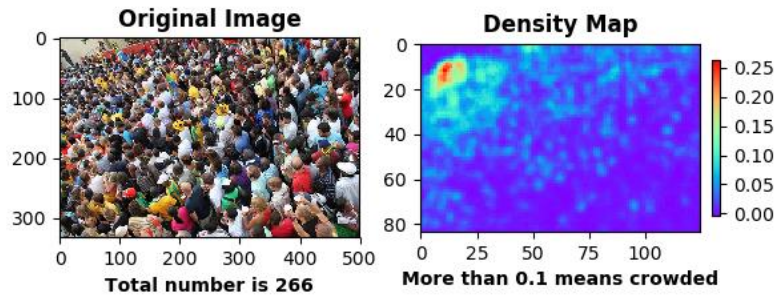


图 密度图无法反映拥挤程度

为改进结果，有两个想法（a）该误差原因不在算法，而应该从原始图片入手，根据摄像头姿态来修正原始图片，来减小照片的角度偏差，再把修正后的图片输入网络进行人群密度预测，这样基本能解决角度偏差带来的密度图偏差（b）其实可以利用摄像头姿态信息训练网络，虽然这个想法不能改进密度图反映拥挤程度的问题，但是可以引申出更好的网络结构，不妨参照 ACNN。

第二点，MSCNN 是解决图片有较多人数而难以统计的问题解决，训练和测试的图像库中，每张图片基本有 20 人以上，在一定程度造成当检测人数极少的图片会出现总人数小于 0 的结果，如下图所示，左图人数统计为-15，不过右图的密度图能较准确地反应信息。

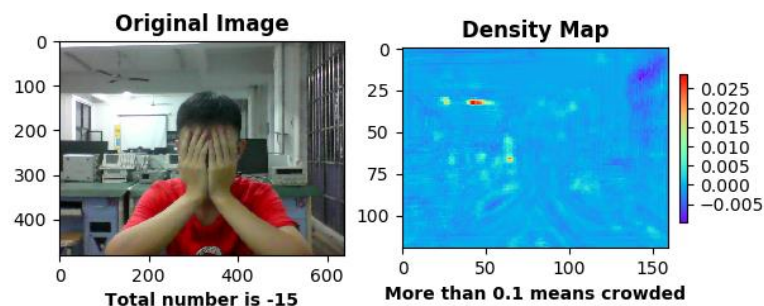


图 人数统计出错

为更准确地显示统计人数,有两个提议:(a)在 MSCNN 的统计结果出现负数时,改为使用其余合适的算法再次检测人数以修正,其余算法可以用传统的 upper-body 特征检测或者针对人数较少的图片训练出的新网络,这样就构成一个级联树 (b) 反思回归方程是否存在不合理性, MSCNN 的 loss 为密度图的欧式距离,但仔细思考直接回归欧式距离时,人数少和人数多对回归结果的敏感度是不同的,为能够更好地回归人数少时的密度图,应该利用非线性函数修正欧式距离:当人数较少时,较小的误差也能导致较大的损失,当人数增多时,较小的误差不会导致较大的损失,比如使用对数函数映射。

第三点, MSCNN 有 GoogleNet 多维度识别的影子,但是却没有优化其中的多维度结构而耗费大量计算资源。(a) 对于 9×9 、 7×7 大小的卷积核,可以借鉴 VGG 堆叠小卷积层的做法:串联 2 个 3×3 的卷积层取代 1 个 5×5 的卷积层,同样达到一个像素会跟周围 5×5 的像素产生关联的效果。这样做一方面是因为 2 个串联的 3×3 的卷积层的参数比 1 个 5×5 的卷积层更少,二是因为 3 个 3×3 的卷积层比 1 个 7×7 的卷积层有更多的非线性变换(前者可以用 2 次 ReLU 激活函数,而后者只有 1 次),对特征的学习能力更强。(b) 借鉴 GoogleNet 的做法,一方面利用 1×1 卷积核做特征图降维操作,另一方面提高卷积层输出特征图的维度,既能防止参数爆炸又能得到更好的提取特征效果,经典 CNN 有更多值得借鉴的技巧等待挖掘。

第四点, MSCNN 的网络架构也许难以有更大的突破,可以尝试 ResNet 的深度残差学习,或者使用 GoogleNet Inception v4 的架构。

五、课程设计总结

本次课程设计历时一个多月,每位组员都为“饭堂人群密度检测”付出许多精力,最后取得可观的成果,其中遇到的坎坷让我们收获良多,以下是我们的总结。

1、硬件摄像（谢博、黄坚荣）

通过本次课程设计，我得到而许多的收获。例如对于树莓派的使用方法增添了许多认识，增加了对多种方式传输文件的了解，同时，也提高了自己面对问题，解决问题的能力。

在设计过程中，也遇到了许多的问题，例如树莓派的无线网卡驱动安装不上、无法传输流媒体、定时任务的设置无法成功执行、在 http 协议传输中文件类型遭更改，服务器端拒绝访问等，在最后经过思考将其中的部分问题解决。

在完成 shell 脚本编写，并通过命令行执行成功后。定时任务设置却无法成功执行，这是由于在定时任务执行时，并没有进入到文件所存放的目录，所以需要脚本内所引用的其他文件加上绝对路径，或者在该定时任务前加上 cd 命令行转入文件所在目录。

而在 http 协议中，文件被转换成表单文件，需要在后面加上说明该文件为 JPEG 格式，使得文件能被服务器端接受。

其次，在本次开发中，不仅提高了我的团队合作能力，也使得我更了解团队之间沟通的重要性，在终端同学想要获得图片时，由于了解错了同学的要求，给了错误的回答，使得了过程多了一点曲折。

2、应用终端（谈裕锦）

通过本次课程设计，我收获颇多。

其一是解决问题的能力提升了。先说一下在项目开发到上线时我遇到的问题：

① 调试图片自动实时更新的功能时（不用手动刷新浏览器，原理是设置一个定时器，周期自己定，没到一个周期就修改图片的 src，修改方法可在路径后加一些随机的查询参数），我设置的周期太小了，设了 1s，所以每次发起请求的时候 1s 过了图片还没从服务器下载完，下一个周期又来了，就中断了图片的下载，又重新发起了一次请求，所以图片更新失败（网速肯定没这么快，1s 就把图片下载完）。

后来我查看控制台的请求情况，就发现图片不断显示发起请求、正在下载，然后状态又在 1s 后迅速变为发起请求，于是我就捉摸到了原因，把周期改为 20s（理论上应该设为 60s，但是怕一个周期错过之后要再等 60s，这样延迟就有点大了）！

② 上传到服务器的时候图片的路径出错了，每次都需要手动的改，后来打开并查阅编译后的生产文件，查出原来是 webpack 配置中 publicPath 还有 html 中的 base 路径配置错了。于是纠正，后面上传到服务器时所有文件的路径就都

对了,不用在编译生成生产文件(最终文件)后还要手动修改代码中的某些路径。

其二,提高了我的团队意识和协作能力。这次课程设计很接近一次真实的项目,大家各司其职,术业有专攻,要想把系统建好,就要了解队友们的工作,之间要有很好的沟通。这次课设之所以能够完成,很大一部分要归功于此。

其三,这次课设令我明白了开发环境的搭建,项目的架构,程序的调试与性能都是非常重要的要素,好的要素很可能让你只花 1 分钟就做完别人花 10 分钟才做完的事情。不管是课设还是工作,我们都在与时间进行赛跑,高效开发必不可少,但高效之中又要保证系统的可维护性、健壮性与好性能。这就要求我们付出更大的精力。永远不要让技术停留在浅显的层面,想要做好一个系统,就要学习更多更深入的东西,并灵活地把它用于实践。

3、核心算法(陈威鹏、李新鹏)

手动提取特征

由于之前没有接触过 python 以及 OpenCV 语言,万事开头难,刚开始去了解学习的时候会感到枯燥且困难,具体完成算法过程中遇到的问题有:

(a) 查询资料时,资料五花八门,需要一篇一篇地去看懂算法思想,了解语言的语法结构。幸亏班上有学习过 python 以及 OpenCV 的同学,期间在他的指导交流下,较快地掌握了语言的基本结构,能够独立地编写代码,对代码进行优化。

(b) 安装 python 以及 OpenCV 的时候掉进过很多坑,运行算法也出现很多报错,最后了解到是环境搭建的问题,最终通过不断地尝试才得以将环境完成搭建。

总之,这一次的课程设计让我受益匪浅。我从中明白了,对于一门语言,学习语法结构是最困难但又是最关键的部分,而利用语言去完成算法就需要软件环境的完美搭建才能运行。同时,我也明白到人多力量大,感谢组员和组外同学对我完成算法过程中的帮助以及指导,让我能较好地完成任务。

MSCNN

实现项目的过程总是坎坷的,一方面需要有恒心毅力,另一方面需要不断提高调适能力。使用 MSCNN 算法的过程中遇到的问题有:

(a) 查询资料时,开始沉陷于利用 opencv 的 haar 特征识别,后来改为研究 CNN 提取高维特征识别,并得到老师的关键指导而使用 MSCNN

(b) 学习 tensorflow 时,安装遇到许多环境错误,通过网络查找答案修改才安装成功,作为小白出发慢慢跟 tensorflow 教学视频理解,最终搭建出脚完善的卷积神经网络,深入理解深度学习的概念

(c) 学习 caffe 时,遇到最大的问题就是在 windows 安装 caffe 框架,当时有许

多依赖库出错，包括 `caffelib`、`pycaffe` 等，而博客或者 `github` 没有提供准确的解答，花了三天时间慢慢摸索出门路，最终成功安装并能够根据错误迅速就纠正错误，接着适应着将 `ubuntu` 运行的 `sh` 文件改为 `windows` 下运行的 `bat` 文件和 `py` 文件，能够做到熟练地在 `windows` 使用 `caffe`

(d) 研究 `MSCNN`，最初遇到自定义 `Layer` 而无法运行，查找教程添加对应层并重新编译 `caffe`，第一次失败，根据编译失败的提示逐步修改源码、清理生成文件，多次调试之后终于成功得到想要的 `caffe.exe`；接着遇到 `check failed:data_`：读取文件不正确的错误，但不是网络参数或者数据路径问题，根据调试信息解决信息慢慢解决问题，最后仍有部分错误没有解决，请教他人发现是笔记本显存不足够计算大量的数据；由于网络计算量较大，即便是 1 张图片都出现存储溢出问题，查找各方面资料没有解决，请教他人得知应除去内存检测重新编译 `caffe`，最后能够跑通 `MSCNN` 结构；为将 `MSCNN` 应用到项目，拍摄图片、绘制热力图和展示，使用 `matplotlib`、`opencv` 等模块编写代码使得完善整个过程，其中遇到大大小小的错误逐步解决。

这次项目我学习到前沿的计算机视觉知识并应用到真实生活，实现过程极大地提高调试能力，在后续思考改进网络的时候认识到计算机视觉广阔的一面，对于自己来说意义非凡，同时作为组长，锻炼出协调工作监督进度的能力。本项目历经艰难，各方面基本调试成功，是对每位组员极大的鼓励，望以后的日子再接再厉。

在总结的最后，感谢指导老师的辛勤付出！老师在整个课程中极度负责的态度值得一赞，开题时认真纠正提升项目质量，项目进行过程中督促并慷慨提供各种支持，结题对学生也是给予认可的鼓励，其中的细节对学生的影响是深远的，十分感谢！

参考文献

- [1] Crowd Counting by Adapting Convolutional Neural Networks with Side Information
- [2] Cross-scene Crowd Counting via Deep Convolutional Neural Networks
- [3] Privacy Preserving Crowd Monitoring: Counting People without People Models or Tracking
- [4] Deep Count: Fruit Counting Based on Deep Simulated Learning
- [5] END-TO-END CROWD COUNTING VIA JOINT LEARNING LOCAL AND GLOBAL COUNT
- [6] Deep Residual Learning for Image Recognition
- [7] Going deeper with convolutions
- [8] Very deep convolutional networks for large-scale image recognition
- [9] Single-Image Crowd Counting via Multi-Column Convolutional Neural Network

六、附录

1、硬件摄像（谢博、黄坚荣）

1.1 方案 1

①test.py

```
1. import time
2. from qiniu import Auth,put_file,etag,urlsafe_base64_encode
3. import qiniu.config
4. import os
5.
6. access_key='oL9QUnx4xpLWpJD5EzWL9JR6wAG53JqfiRCVGobw'
7. secret_key='qIspGG9a5KmP2pn_PFaUqaVo1bEQQ13vaSoG_MT9'
8.
9. q=Auth(access_key,secret_key)
10. bucket_name='raspic'
11. key='current.jpg'
12. token=q.upload_token(bucket_name,key,60)
13. localfile='current_photo.jpg'
14.
15. filename=r'/home/pi/Desktop/workspace/take_ph/current_photo.jpg'
16.
17. ret,info=put_file(token,key,localfile)
18. print(info)
19. assert ret['key']==key
20. assert ret['hash']==etag(localfile)
21.
22. if os.path.exists(filename):
23. os.remove(filename)
```

②take_photo.sh

```
1. raspistill -o /home/pi/Desktop/workspace/take_ph/current_photo.jpg -q 5
2. python2.7 /home/pi/Desktop/workspace/take_ph/test.py
```

1.2 方案 2

①Client.py

```

1. #coding=utf-8
2. import requests
3. import os
4. url = 'http://120.25.59.185:8081'
5. path = u'/home/pi/Desktop/workspace/take_ph1/current_photo.jpg'
6. print path
7. files= {'file':('current_photo.jpg',open(path, 'rb'),'image/jpeg')}
8. r = requests.post(url, files=files)
9. print r.url,r.text

```

②take_photo

```

1. raspistill -o /home/pi/Desktop/workspace/take_ph1/current_photo.jpg -q 5
2. python2.7 /home/pi/Desktop/workspace/take_ph/Client.py

```

2、应用终端（谈裕锦）

由于网页项目较庞大，不适合粘贴到文档，可以访问组员谈裕锦的 github: websites 项目（<https://github.com/BeYanJin/websites>）

3、核心算法（陈威鹏、李新鹏）

手动提取特征应用代码

facedetectCam.py

```

1. import cv2
2. import sys
3.
4. cascPath = "haarcascades/haarcascade_frontalface_default.xml"
5. faceCascade = cv2.CascadeClassifier(cascPath)
6.
7. video_capture = cv2.VideoCapture(0)
8.
9. while True:
10.     # Capture frame-by-frame
11.     ret, frame = video_capture.read()
12.     gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
13.     faces = faceCascade.detectMultiScale(
14.         gray,
15.         scaleFactor=1.1,

```

```

16.         minNeighbors=5,
17.         minSize=(30, 30),
18.         flags=cv2.CASCADE_SCALE_IMAGE
19.     )
20.
21.     # Draw a rectangle around the faces
22.     for (x, y, w, h) in faces:
23.         cv2.rectangle(frame, (x, y), (x+w, y+h), (0, 255, 0), 2)
24.
25.     # Display the resulting frame
26.     cv2.imshow('Video', frame)
27.     if cv2.waitKey(1) & 0xFF == ord('q'):
28.         break
29.
30. # When everything is done, release the capture
31. video_capture.release()
32. cv2.destroyAllWindows()

```

MSCNN 应用代码

PredictCam.py

```

1. # -*- coding: utf-8 -*-
2. import os
3. import sys
4. import caffe
5. import numpy as np
6. import cv2
7. import scipy.io
8. import time
9. from matplotlib import pyplot as plt # plt 用于显示图片
10. from matplotlib import cm as cm # cm 用于设置热力图
11. import matplotlib.image as img # img 用于读取图片
12.
13. interval = 2          # 捕获图像的间隔，单位：秒
14. num_frames = 1        # 捕获图像的总帧数
15. out_fps = 24          # 输出文件的帧率
16.
17. caffe.set_mode_gpu()
18. caffe.set_device(0)
19.
20. proto_use = 'proto/MSCNN_deploy.prototxt'

```



```

21. proto_file = 'proto/MSCNN.prototxt'
22. model_file = 'snapshot_aug/mscnn_partA_iter_380000.caffemodel'
23.
24. # VideoCapture(0)表示打开默认的相机
25. cap = cv2.VideoCapture(0)
26. # 获取捕获的分辨率
27. size =(int(cap.get(cv2.CAP_PROP_FRAME_WIDTH)),
28.         int(cap.get(cv2.CAP_PROP_FRAME_HEIGHT)))
29. # 设置要保存视频的编码, 分辨率和帧率
30. video = cv2.VideoWriter(
31.     "Crowd.avi",
32.     cv2.VideoWriter_fourcc('M','P','4','2'), #指定了视频编码的格式, 此处用的是
        MP42, 也就是 MPEG-4
33.     out_fps,
34.     size
35. )
36.
37. def edit_proto(img_height, img_width):
38.     with open(proto_file, 'r') as template_file:
39.         template_proto = template_file.read()
40.         # 写好测试模型文件, 即复制 MSCNN 到 MSCNN_deploy
41.         with open(proto_use, 'w') as out_file:
42.             out_file.write(template_proto.format(height=img_height, width=im
                g_width))
43.
44. def predict_count(frame):
45.     img_height = frame.shape[0]
46.     img_width = frame.shape[1]
47.     frame = np.reshape(frame, (img_height, img_width, 1))
48.     edit_proto(img_height, img_width)
49.     net_full_conv = caffe.Net(proto_use, model_file, caffe.TEST)
50.     transformers = caffe.io.Transformer({'data': net_full_conv.blobs['data']
        .data.shape})
51.     transformers.set_transpose('data', (2,0,1))
52.     # 转化为数组再传入网络
53.     out = net_full_conv.forward_all(data=np.array([transformers.preprocess('
        data', frame)]))
54.     # 输出密度图, (图片数量,通道,高度,宽度)
55.     estdmap = out['estdmap']
56.     print(estdmap.shape)
57.     # 1*1*height*width->height*width, 只要元素总量一致即可

```

```

58.     estdmap = np.reshape(estdmap, (estdmap.shape[2], estdmap.shape[3]))
59.     # sum 为求和, ceil 为向下取整
60.     count = int(np.ceil(np.sum(out['estdmap'])))
61.     return count, estdmap
62.
63. def Img2Map(originImg, count, densityImg):
64.     fig = plt.figure(facecolor="w")
65.     img1 = fig.add_subplot(121)#121 表示整体布局为 1 行 2 列, 此为第 1 副
66.     img1.imshow(originImg)
67.     # plt.axis('off') # 不显示坐标轴
68.     plt.title("Original Image", fontweight='bold')
69.     plt.xlabel('Total number is '+str(count), fontweight='bold')
70.
71.     img2 = fig.add_subplot(122)#121 表示整体布局为 1 行 2 列, 此为第 2 副
72.     cmap = cm.get_cmap('rainbow')# 设置 cmap 形式, 对于同一幅图, 彩虹图形象地显示
        密集的地方, 不同图像不行
73.     map = img2.imshow(densityImg, cmap=cmap) # 显示图片, 默认为热力图
74.     bar=plt.colorbar(mappable=map, fraction=0.026, pad=0.04)
75.     # bar.set_label('over 0.15 indicates crowdy', fontweight='bold')
76.     # plt.axis('off') # 不显示坐标轴
77.     plt.title("Density Map", fontweight='bold')
78.     plt.xlabel('More than 0.1 means crowded', fontweight='bold')
79.
80.     plt.savefig('test.png')
81.     # plt.show()
82.
83. if __name__ == '__main__':
84.     # 对于一些低画质的摄像头, 前面的帧可能不稳定, 略过
85.     for i in range(42):
86.         cap.read()#没有返回即放弃
87.
88.     # 开始捕获, 通过 read()函数获取捕获的帧
89.     try:
90.         for i in range(num_frames):
91.             _, frame = cap.read()#opencv 读取图片格式为 BGR
92.             count, estdmap=predict_count(cv2.cvtColor(frame, cv2.COLOR_BGR2GR
                AY))
93.             Img2Map(cv2.cvtColor(frame, cv2.COLOR_BGR2RGB), count, estdmap)
94.             time.sleep(interval/2) # 进程挂起, 若要显示图片需要多进程
95.             # Figure = cv2.cvtColor(cv2.imread('test.png'), cv2.COLOR_BGR2RGB
                )

```

```

96.         Figure = cv2.imread('test.png')
97.         video.write(Figure)
98.
99.         # 如果希望把每一帧也存成文件，比如制作 GIF，则取消下面的注释
100.        filename = '{:0>6d}.png'.format(i) #0>6d 表示 6 位数
101.        cv2.imwrite(filename, Figure)
102.
103.        print('Frame {} is captured.'.format(i))
104.        time.sleep(interval/2) # 进程挂起，若要显示图片需要多进程
105.    except KeyboardInterrupt: #用来获取用户 Ctrl+C 的中止
106.        # 提前停止捕获
107.        print('Stopped! {}/{} frames captured!'.format(i, num_frames))
108.
109.    # 释放资源并写入视频文件
110.    video.release()
111.    cap.release()

```

predictImg.py

```

1. # -*- coding: utf-8 -*-
2. import os
3. import sys
4. import caffe
5. import numpy as np
6. import cv2
7. import scipy.io
8. from matplotlib import pyplot as plt # plt 用于显示图片
9. from matplotlib import cm as cm # cm 用于设置热力图
10. import matplotlib.image as img # img 用于读取图片
11.
12. caffe.set_mode_gpu()
13. caffe.set_device(0)
14.
15. proto_use = 'proto/MSCNN_deploy.prototxt'
16. proto_file = 'proto/MSCNN.prototxt'
17. model_file = 'snapshot_aug/mscnn_partA_iter_380000.caffemodel'
18. images_path = 'D:/Caffe/caffe-windows-ms/data/ShanghaiTech/part_A/test_data/
    images/IMG_15.jpg'
19.
20. def edit_proto(img_height, img_width):
21.     with open(proto_file, 'r') as template_file:

```

```

22.         template_proto = template_file.read()
23.         # 写好测试模型文件，即复制 MSCNN 到 MSCNN_deploy
24.         with open(proto_use, 'w') as out_file:
25.             out_file.write(template_proto.format(height=img_height, width=img_width))
26.
27. def predict_count(image_path):
28.     frame = cv2.imread(image_path)
29.     if not frame.shape[2] == 1:
30.         frame = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
31.     img_height = frame.shape[0]
32.     img_width = frame.shape[1]
33.     frame = np.reshape(frame, (img_height, img_width, 1))
34.     edit_proto(img_height, img_width)
35.     net_full_conv = caffe.Net(proto_use, model_file, caffe.TEST)
36.     transformers = caffe.io.Transformer({'data': net_full_conv.blobs['data'].data.shape})
37.     transformers.set_transpose('data', (2,0,1))
38.     # 转化为数组再传入网络
39.     out = net_full_conv.forward_all(data=np.array([transformers.preprocess('data', frame)]))
40.     # 输出密度图，（图片数量,通道,高度,宽度）
41.     estdmap = out['estdmap']
42.     print(estdmap.shape)
43.     # 1*1*height*width->height*width, 只要元素总量一致即可
44.     estdmap = np.reshape(estdmap, (estdmap.shape[2], estdmap.shape[3]))
45.     # sum 为求和, ceil 为向下取整
46.     count = int(np.ceil(np.sum(out['estdmap'])))
47.     return count, estdmap
48.
49. def display(count, densityImg):
50.     fig = plt.figure(facecolor="w")
51.     img1 = fig.add_subplot(121)#121 表示整体布局为 1 行 2 列，此为第 1 副
52.     originImg = img.imread(images_path)
53.     img1.imshow(originImg)
54.     # plt.axis('off') # 不显示坐标轴
55.     plt.title("Original Image", fontweight='bold')
56.     plt.xlabel('Total number is '+str(count), fontweight='bold')
57.
58.     img2 = fig.add_subplot(122)#121 表示整体布局为 1 行 2 列，此为第 2 副

```

```

59.     cmap = cm.get_cmap('rainbow')# 设置 cmap 形式，对于同一幅图，彩虹图形象地显示
      密集的地方，不同图像不行
60.     map = img2.imshow(densityImg, cmap=cmap) # 显示图片，默认为热力图
61.     bar=plt.colorbar(mappable=map,fraction=0.026, pad=0.04)
62.     # bar.set_label('over 0.15 indicates crowded',fontweight='bold')
63.     # plt.axis('off') # 不显示坐标轴
64.     plt.title("Density Map",fontweight='bold')
65.     plt.xlabel('More than 0.1 means crowded',fontweight='bold')
66.
67.     plt.savefig('test.png')
68.     plt.show()
69.
70. if __name__ == '__main__':
71.     count,densityImg = predict_count(images_path)
72.     # print(count)
73.     # print(estdmap)
74.     display(count,densityImg)

```