# 心情日记

## 整体设计



图1 整体设计

如图1所示为本项目整体设计，分为三个部分：硬件端、服务器端和App端。

* App端：实现用户的注册、登录和可视化等功能。App注册新用户后，将新用户的信息和一张参考照片上传给服务器。服务器保存该数据。用户登录时，App端向服务器发送请求，验证用户的合法性和校验密码。App端根据用户需求向服务器发送请求，获取近期的心情分析结果，对结果能进行间接明了的显示。
* 硬件端：负责人脸的检测。每注册一个新用户，就有一个可联网摄像头可接入服务器，向服务器上传采集的人脸照片。需要说明的是，硬件端不需要校验照片是否是对应用户的照片，只需要上传图片。由服务器端识别图片中的人是否是用户，是哪一个用户。
* 服务器端：整体分为算法部分和存储部分。
* 存储部分：A、建立mysql数据库win2020，含两个表。一个表存储用户信息，一个表存储每一张正确照片的心情结果，包括：用户-表情-时间。B、存储App端上传的参考用户图片在pre\_face文件夹里。C、存储硬件端上传的待识别图片在faces文件夹里。
* 算法部分：需要完成两个工作。A、读写数据库。注册新用户时写入、登录验证时比较、表情识别时验证用户存在与否、获取情绪结果时读取存储结果并进行简单计算分析。B、表情识别。主要完成人脸识别和表情识别。

根据上面说明，可整理出如下关系（接口设计）。



图2 接口设计

## 服务器端

如上整体设计中说明，服务器端分为算法和存储两部分。先介绍存储部分，再介绍算法部分。

1. **存储部分**

存储部分有图片存储和数据库记录存储两部分。图片存储有文件夹faces，保存所有由硬件端上传的采集的人脸图片，命名包括中包含识别；由文件夹pre\_face将所有的参考图片存储，命名为用户设置的名称。（我们不会将用户信息外泄）

数据库部分：建立数据库名win2020。算法访问用户为guest，仅赋予权限select、insert、updata、delete等对记录操作的权限，以防误删数据库和数据表。建立数据表user\_info和picture\_emotion，设置的字段和数据类型如表1和表2。

表1 User\_info

|  |  |  |
| --- | --- | --- |
| 字段名 | 数据类型 | 其他 |
| user (KEY) | VARCHAR(20) | NOT NULL |
| pwd | VARCHAR(20) | NOT NULL |

数据表usr\_info用来存储注册的用户。包括用户名称和登录密码。其中用户名称为主键，保证每个用户在此表中中只出现一次。

表2 Picture\_emotion

|  |  |  |
| --- | --- | --- |
| 字段名 | 类型 | 其他 |
| Id(KEY) | INT | AUTO\_INCREMENT |
| user | VARCHAR(20) | NOT NULL |
| date | DATE | NOT NULL |
| hour | VARCHAR(2) | NOT NULL |
| emotion | VARCHAR(10) | NOT NULL |

数据表picture\_emotion用来存储faces文件夹中合理的照片的识别结果，每一个结果对应一条记录，主键对为增长的整数，每条记录是唯一的。其他字段有用户名、时间、和表情结果。即一条记录包含用户和它对应某时刻的表情结果。

数据库设置比较简单。为了保证表user\_info和表picture\_emotion之间的联动性，采取了如下措施。A、当用户注销时，删除两个表所有user为注销用户的记录。B、当往数据库存储用户的表情结果时，会先检查表user\_info中是否存在该用户，如果不存在，则记录不保存。这样就可以保证表picture\_emotion中保存的记录全部是我们注册了的合法用户。

展示：

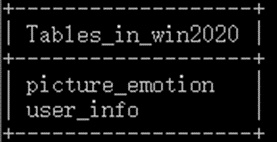


图3 数据库和数据表

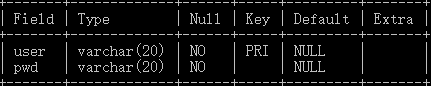


图4 数据表user\_info

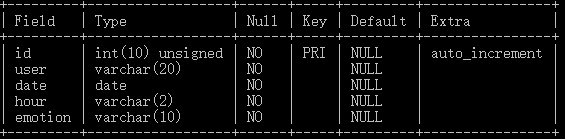


图5 数据表picture\_emotion

1. **算法部分**

算法部分包括对数据库的读写操作和表情识别部分。先介绍表情识别部分。

1. **表情识别部分**

表情识别部分的处理流程如图3。首先从图片存储文件夹中faces中读取一张图片(说明：这里文件读取过程为逐一遍历整个文件夹，为了方便说明，我们以一张图片为例)。根据图片的名称，获取图片的一些基本信息：采集的时间。之后我们将此图片与参考图片文件夹pre\_face中的图片，通过人脸识别算法进行比较。如果，不存在与之对应的用户图片，说明硬件端采集的图片不是我们注册的用户的图片，那么删除改图片，继续下一张图片。如果，存在与之对应的图片，将进行表情识别，同时将对应的用户名称返回(这里就知道了一张图片属于哪一个用户)。表情识别结束后，删除对应的图片。在数据库user\_info中查找是否存在对应的用户，如果存在则保存一条记录：用户名+时间+识别结果，如果不存在，则不保存记录。一轮识别结束，重新开始。直到faces中图片为空。

以上实现过程分成模块由几个函数各自完成，下面对些模块进行介绍。



图6 服务器端表情识别算法实现流程

* **模块1：函数recognition**

**功能描述：**人脸识别。将从faces中读取的图片与参考图片文件夹pre\_face中的图片一一比对，判断是否是某一个用户的图片，是则返回对应的用户名，不是就删除图片，返回’False’。

**接口描述：**输入I为待识别的图片路径，输出O为用户名或’False’。

**调用模块：**os、face\_recognition(依赖于dlib)、dlib

**实现思路：**通过遍历pre\_face中的参考图片，借助face\_recognition库识别人脸。读取pre\_face中的一张图片，用load\_image\_file函数加载图片，再对其进行编码，再将待检测图片加载编码。通过比较两个图片之间的向量距离进行判断是不是同一个人，如果是，则停止遍历返回用户名，不是则继续下一张，直到遍历结果返回False(string型)。

在识别人脸时，没有借助face\_recognition中已有的识别模块，因为根据常识，准确度不高，笔者将自己的照片与妹妹的照片进行比较，发现不能区分，故选用向量距离比较，距离相差小，则判断是通一个人，相差大，则判断不是同一个人。此判断阈值设置不能过大，过大则不能区分很相似的人，也不能过小，否则不能包容角度光线或者拍摄出现的不可抗因素。这里设置的是0.4。

另外要注意的一点是，这里返回的False不是布尔型，因为若是存在就要返回用户名，这两者的数据类型应该一致。

* **模块2：表情识别**

**功能描述：**表情识别过程，包含函数face\_detect(人脸检测)、generate\_faces(将探测到的人脸进行增广)、predict\_expression(表情预测)、CNN(搭建的卷积神经网络)、index2emotion(将识别的表情下标转换成对应字符串)。共能识别'发怒': 'anger'，'厌恶': 'disgust'，'恐惧': 'fear'，'开心': 'happy'，'伤心': 'sad'，'惊讶': 'surprised'，'中性': 'neutral'，'蔑视': 'contempt'八种表情。

**接口描述：**这里predict\_expression为主函数，通过调用其他函数整体实现表情识别的过程。

表3 接口说明

|  |  |  |
| --- | --- | --- |
| 函数名 | I | O |
| predict\_expression | 待检测图片路径、加载的卷积模型包含对应以及训练好的权值 | 表情识别的结果和各个表情的概率 |
| face\_detect | 待检测图片路径 | 检测出的灰度化处理后的人脸 |
| generate\_faces | 检测出的灰度化处理后的人脸 | 增广处理后的人脸4848大小 |
| Cnn | 各种处理后的人脸 | 表情下标和各个表情的概率 |
| index2emotion | 表情下标 | 对应的表情 |

**调用模块：**opencv、numpy、keras、PIL、os、tensorflow

**外部文件：**cnn3\_best\_weights.h5为卷积神经网络的权值

**实现思路：**这里主要借助opencv进行人脸的检测等处理，利用keras（依赖于tensorflow）搭建卷积神经网络，numpy主要进行一些数组切片等的处理以实现图像大小的增广等。首先利用opencv进行人脸标定(检测)和灰度化处理，再将灰度化已将标定出的人脸处理成4848大小的数据，传递入卷积神经网络。通过搭建好的神经网络和训练好的最佳权值进行运算，最后以编号的形式输出表情结果，对表情结果进行转化，即可得到识别出的表情。卷积神经网络的结构如下图所示。

最后，卷积神经网络的实现参考了github中开源的代码（附上链接https://github.com/luanshiyinyang/ExpressionRecognition.git）。利用其中训练好的权值和对应的网络结构模型。一开始尝试找了训练集自己训练，但是电脑设备吃不消，得出的结果不是很好，并且发现训练源本身也存在数据量少获取正确率低问题。模型的构建参考2018年CVPR几篇论文以及谷歌的Going Deeper设计。图4中网络结构，输入层后加入(1,1)卷积层增加非线性表示且模型层次较浅，参数较少（大量参数集中在全连接层）。在FER2013、JAFFE、CK+上进行训练，JAFFE给出的是半身图因此做了人脸检测。最后在FER2013上Pub Test和Pri Test均达到67%左右准确率（该数据集爬虫采集存在标签错误、水印、动画图片等问题），JAFFE和CK+5折交叉验证均达到99%左右准确率（这两个数据集为实验室采集，较为准确标准）。

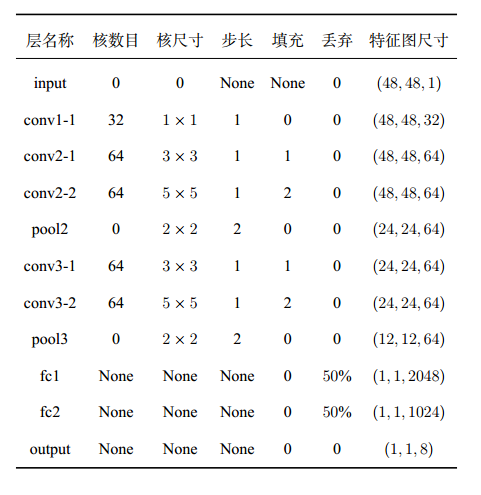


图7 卷积神经网络结构



图8 识别结果实例

* **模块3：写入数据库**

**功能说明：**将识别结果写入数据库。包含research\_user(查找用户是否存在)和write\_emotion(将结果写入数据库)两个函数。

**接口说明：**write\_emotion调用research\_user。

表5 接口说明

|  |  |  |
| --- | --- | --- |
| 函数名 | I | O |
| research\_user | 文件名、对应用户名、结果 | 无 |
| write\_emotion | 用户名 | True/False(bool型) |

**调用模块：**pymysql实现python对mysql的读写操作

**实现思路：**首先通过research\_user检测识别出的用户是否还在user\_info中，即判断用户合法性。若用户合法，则根据文件名提取文件的时间，然后将用户、时间和结果写入数据库。这里进行了数据的记录查找search和插入insert操作。主要流程是：根据数据库名、用户和密码建立连接，在指定表查找满足条件的数据或者在指定表插入数据。检查操作是否完成，之后关闭和数据库的连接。整体实现有套路，熟悉了数据库的结构和简单的语法即可。

本地模拟的结果展示：

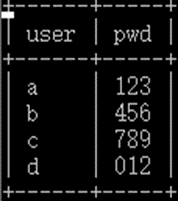
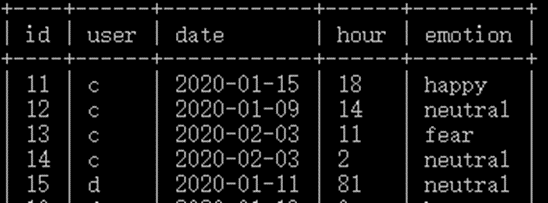
 

图9 本地模拟结果展示

1. **数据库读写部分**

这部分整体结构很类似，划分成两部分，用户的注册和注销部分，以及返回用户心情结果的部分。

* **模块1：用户的注册和注销**

**功能说明**：App端发送注册和注销信号后，执行的操作，将新用户的信息写入数据库表usr\_info中，或者将指定的注销用户的两个表中所有相关记录删除。由函数add\_user和delete\_user分别实现

调用模块：pymysql

**接口说明：**

表6 接口说明

|  |  |
| --- | --- |
| 函数名 | I |
| add\_user | Id，pwd(用户名称和密码) |
| delete\_user | Id(用户名称) |

* **模块2：结果分析**

**功能说明：**根据用户的请求，返回某一天或过去30天的心情结果。分别由函数day\_analyse和month\_analyse实现。

**调用模块：**pymysql

**接口说明：**

|  |  |  |
| --- | --- | --- |
| 函数名 | I | O |
| day\_analyse | user, date(用户名称和指定日期) | userName\_day.json |
| month\_analyse | User(用户名称) | userName\_month.json |

**实现思路：**首先与数据库建立连接，然后根据用户名和日期条件筛选出符合的记录，对记录的表情结果进行简单的统计计算。然后写成以用户名\_month或用户名\_day命名的.json文件。之后发送给App端。.json文件格式示例：

{

'user\_name': 'name',

"anger"：0.0,

"disgust"：0.1,

"fear"：0.1,

"happy"：0.8,

"sad"：0.0,

"surprised"：0.0,

"neutral"：0.0,

"contempt"：0.0

}

1. **需要改进的地方**

判断用户合法性这一部分是在写入数据库那里判断的，这时候已经进行了一轮的表情识别。如果用户不合法，那么之前的表情是没有意义的。故可以将用户合法性识别放在人脸识别之后，一检测处是合法的存在的用户图片，就是判断是否还在数据库中。这样就可以减少多余的运算。

1. **技术上遇到的问题**

* **表情识别部分：**

**安装库：**这是一开始遇到的问题，在安装dlib和face\_recognition这两个的时候一直不成功，根据多方查找，知道原因是dlib安装在windows上依赖于cmake和vs2015，且版本之间由对应关系。所以先装了cmake(vs2015已经有了)，变成功了。而face\_recognition依赖于dlib，所以得先装好dlib才行。此外，由于库传输距离很远，pip很慢，可以选择镜像源，或者先从github上下载好文件，然后解压，再setup.py的形式安装。

**代码部分：**在调用开源代码cnn时，一开始无法成功使用。先将一个变量指向我们搭建好的神经网络，然后变量名.load\_weights就可以正确的加载权值，然后用变量名.predict进行预测。此外，图片的前期处理也很重要，从上面的说明我们可以知道，cnn的输入是一个4848尺寸的数据，所以一定要先处理成这种形式，不然无法运行。这里变用来opencv进行人脸标定检测和numpy处理数组大小。一定要熟悉opencv一些常用函数的参数。在使用face\_recognition时还出现了参考图片编码维度的问题，需要对应，对于人脸识别选用了距离比较的方式，前已说明。此外，将不同的功能写在一个个独立的函数里是一个很好的习惯，，进行一些文字输出可以提示报错问题。

**对接部分：**对于和硬件端的对接上如何分配功能实现和对接细节也遇到了一些问题，现行解决方案认为是最好的。一开始想把人脸识别也放在硬件端，但这样硬件端就需要和app通信获得用户的信息或者参考照片，这样很麻烦。后来想到了集中管理的方式，由服务器来人脸识别，硬件端不需要知道采集的照片是否正确，只要是人脸就可以，这样便省去了刚才的问题。弊端是可能硬件端会采集很多不需要的图片，上传到服务器会占用内存，且花费流量。但是考虑到，拟定的场景，摄像头在私人场所，所以，误采集的比重不大，同时，通过运行代码的及时删除，也可以解决占用内存的问题。

* **数据库部分**

对于数据库搭建成什么形式进行了一定的思考，后来考虑到需求的简单性和明确性，仅保留了必要的字段，没有建立联结表与表之间的联结关系。此外，注销用户部分的函数，发现使用代码无法同时删除两个表，就只能一个表一个表的删，且语法里的“`”,不是单引号，而是另外一个英文字符，这个很坑，要注意。

其他问题不多说了，基本GitHub或者官网教程写的很详细，我也是看着学的。