《移动计算技术》课程报告

题目：基于人脸情感识别的移动 App 应用

班级序号：_____

学生姓名：_____

任课教师：_____

中国地质大学地理与信息工程学院软件工程系

2019 年 12 月

一、引言

人脸的情感识别是计算机视觉的一个重要研究方向，在人机交互、情感机器人、陪伴机器人 等应用领域有重要的实际应用意义。目前，研究者们在研究面部表情识别时大多以传统七类人脸表情类别为基础，进行深度特征学习和深度特征分类，本题在这种理论的基础之上，采取 C/S 架构设计软件系统，使得在移动端也能够精准的识别情感。

二、关键技术方案

2.1 vggNet

VggNet 是牛津大学中 Visual Geometry Group 组提出的一种卷积神经网络，其目的是解决解决 ImageNet 中的 1000 类图像分类问题。VggNet 在 AlexNet 基础之上做了一定的改进，采用连续的几个 3x3 的卷积核代替 AlexNet 中的较大卷积核（11x11，7x7，5x5），简单的说来，在 VGG 中，使用了 3 个 3x3 卷积核来代替 AlexNet 中 7x7 卷积核，使用了 2 个 3x3 卷积核来代替 5*5 卷积核，这样做的主要目的是在保证具有相同感知野的条件下，提升了网络的深度，在一定程度上提升了神经网络的效果，并且小卷积核的使用带来参数量减少，可以更加 steadily 地增加层数得同时不会太过于担心计算量的暴增。本次使用的就是 vgg19，是 vggNet 中网络的一种，包含了 19 个隐藏层（16 个卷积层和 3 个全连接层），本次实习使用了前面的 16 个卷积层作为特征提取工具。

ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input (224 × 224 RGB image)					
conv3-64	conv3-64 **LRN**	conv3-64 **conv3-64**	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128	conv3-128	conv3-128 **conv3-128**	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128
maxpool					
conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256 **conv1-256**	conv3-256 conv3-256 **conv3-256**	conv3-256 conv3-256 conv3-256 **conv3-256**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					

Table 2: **Number of parameters** (in millions).

Network	A,A-LRN	B	C	D	E
Number of parameters	133	133	134	138	144

图 1：vgg 网络层数以及相对应的参数

2.2 pytorch 迁移学习

pytorch 是一个 python 的深度学习框架,它深入在 python 之上,PyTorch 具有轻巧的框架。我们集成了各种加速库，如 Intel MKL、英伟达的 CuDNN 和 NCCL 来优化速度。在 PyTorch 当中提供了各种已经预训练好的网络模型，可以作为特征提取的工具，本次实习就使用了 pytorch 网络提供的 vgg19，将 vgg19 作为特征提取的工具然后在后续添加了一层全连接层，进行相对应情感分类。

三、系统分析与设计

3.1 需求定义

功能需求：

1.激活人脸识别引擎

用例编号	UC_1	用例名称	激活人脸识别引擎
用例描述	该用例描述了用户如何使用软件激活人脸识别引擎		
主参与者	用户		
前置条件	用户进入软件，选择“active_engine”		
后置条件	无		
级别	无		
基本事件流程： 1.用户选择“active_engine”			
候选事件流程： 无			
特殊需求	无		
扩展点	无		
备注	无		

2.静态图片人脸特征的检测

用例编号	UC_2	用例名称	静态图片人脸特征检测
用例描述	该用例描述了用户如何使用软件进行静态图片人脸特征检测，在这里面会进行联网实现情感识别		
主参与者	用户		
前置条件	已经激活了人脸识别引擎		
后置条件	无		
级别	无		
基本事件流程： 1.用户选择“face attributes detect(image)” 2.用户从相册中选择图片 3.点击“processing”			
候选事件流程： 无			
特殊需求	无		
扩展点	无		
备注	无		

3.动态人脸特征检测

用例编号	UC_3	用例名称	动态人脸特征检测
用例描述	该用例描述了用户如何动态进行人脸特征识别		
主参与者	用户		
前置条件	已经激活了人脸识别引擎		
后置条件	无		
级别	无		
基本事件流程： 1.用户进入程序，选择对应的识别角度 2.用户选择“face attributes detect(image)”			
候选事件流程： 1.变换摄像头			
特殊需求	无		
扩展点	无		
备注	无		

系统用例图如下：

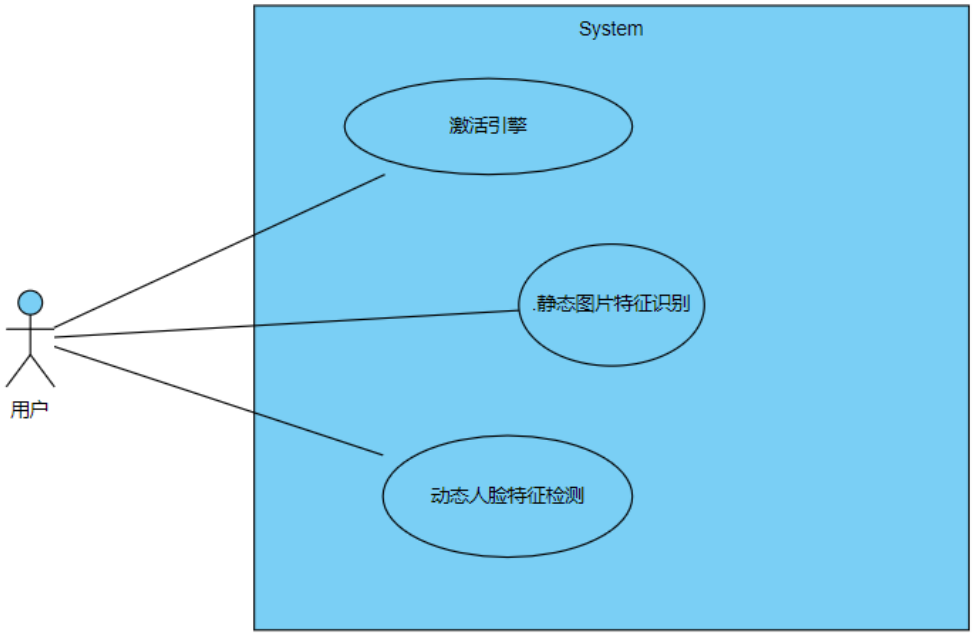


图 2：系统用例图

非功能需求：

1.性能需求（RELIABILITY）：

a.动态识别情感时，识别时间不能太长，需要在一个可接受时间内实现特征以及情感的识别。

b.情感识别的准确度不能太低，能够识别大多数人脸的情感。

2. 用户体验/易用性（User Experience）：

a.比较简单，操作容易

b.界面美观，体验良好

3.2 架构设计

1.架构设计：

使用 C/S 架构，采用客户端服务器模型，将人脸检测等功能在客户端自己本地实现，而人脸情感识别在服务器端实现，通过网络实现传输图片，然后将识别结果再通过网络传输给客户端，通过这种设计，实现简单的在移动端进行人脸情感识别。将移动端部署在 Android 系统上，将服务器部署在自己的电脑上，可以实现在局域网内的移动端人脸情感识别。

2.逻辑设计：

整个系统分为两个部分，Android 客户端和服务器。在 Android 客户端中又分为两个部分，一部分是 Activity，负责和用户产生交互，并且调用相应的接口提供服务；一部分为 FaceEngine，提供人脸识别的相对应服务，这部分依赖于虹软 SDK 提供的服务，实现对人脸识别的检测，是 SDK 中以及提供的编辑好的类。在客户端方面也分为两部分，一部分是 Server，是服务器和客户端进行通讯交互的部分；一部分是 Emotion_Recognition,用于训练模型，然后提供训练好的网络给 Server 调用，进行人脸的情感识别。系统的逻辑视图如图 3 所示.

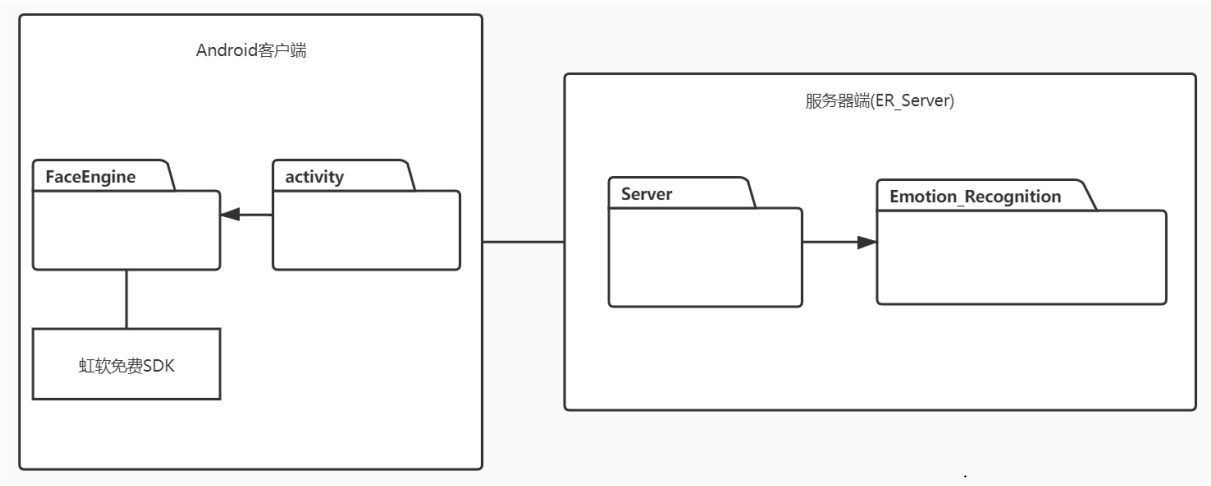


图 3：系统逻辑视图

3.构件设计：

Android 客户端，主要的构件有 4 个，对构件的介绍，如表 1 所示

构件名	描述
FaceEngine	依赖于虹软提 SDK，对功能进行了对应的封装，可以通过创建对应的对象进行使用，更加适合别人对其进行开发。
SingleImageActivity	单张图片处理活动，可以对单张的人脸图片进行人脸检测以及特征提取（识别对应的年龄、性别等），依赖于 FaceEngine 提供的接口。并且可以使用网络和服务器进行通讯，从而能够识别人脸的情感。
FaceAttrPreviewActivity	人脸特征检测活动，通过打开摄像头能够动态的进行人脸检测以及特征提取。
ChooseFunctionActivity	程序进入时的构件，会在里面进行功能的选择以及人脸引擎的激活。

表 1

服务器端：

对构件的讲解如表 2 所示：

构件名	描述
Er_server	服务器端实现服务器功能的构件，能够实现和客户端通过套接字进行交互，并且通过相对应的协议接收客户端发送过来的文件，然后进行图片的人脸情感识别
vggModel	训练好的网络模型，可以用于人脸情感的识别，提供接口给 Er_server 使用，用于人脸情感识别

表 2

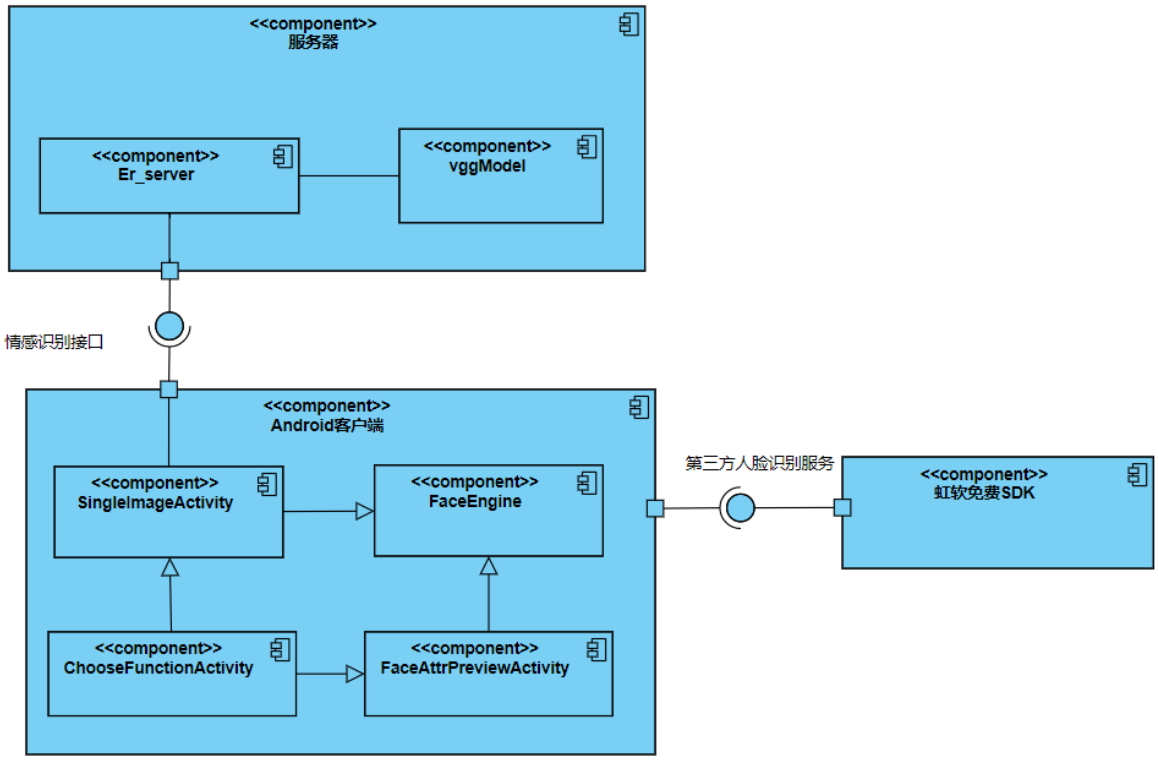


图 4：系统构件图

4.物理设计：

将客户端部署在 Android 上，将服务器部署在自己的电脑上，两者之间通过 socket 套接字进行相对应的通讯，具体的部署图如图 5 所示。

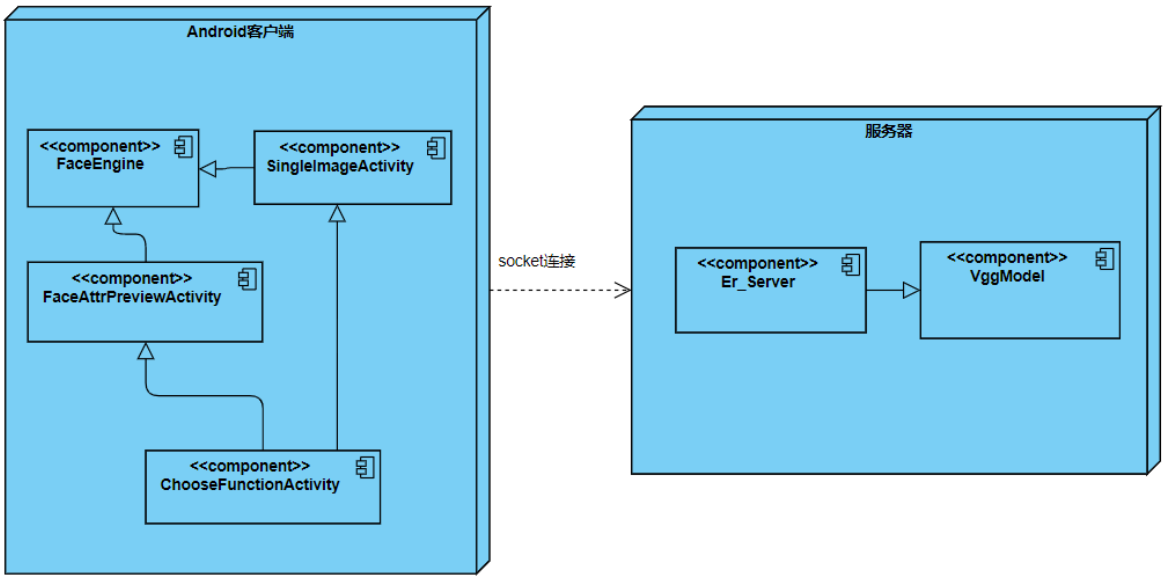


图 5：系统部署图

3.3 程序设计

Android 客户端：

1. FaceEngine：

该类提供了基本所有的人脸检测的接口，只需要实例化该类的对象，就可以配合着使用接口进行人脸特征的检测，其接口如表 3 所示。

接口名	参数说明	返回值说明	功能介绍
int getAge(List<AgeInfo> ageInfoList)	List 类型，将年龄信息存放在其中	Int 类型，根据参数可以判断年龄特征是否提取成功	提取人脸的年龄特征
int getGender(List<GenderInfo> genderInfoList)	List 类型，将性别信息存放在其中	Int 类型，根据参数可以判断性别特征是否提取成功	提取人脸的性别特征
int getLiveness(List<LivenessInfo> livenessInfoList)	List 类型，将人脸活体信息存放在其中	Int 类型，根据参数可以判断活体特征是否提取成功	提取人脸的活体特征

表 3

2. SingleImageActivity：

实现静态的人脸特征识别以及人脸情感识别，里面有两个 AsyncTask，反别为 FindFaceTask 和 NetTask，功能分别为本地的人脸特征提取以及通过网络进行人脸情感分析。其中重要的接口如表 4 所示。

接口名	参数说明	返回值说明	功能介绍
SpannableStringBuilder processImage()	无	SpannableStringBuilder 类型，用于显示特征类型	调用 FaceEngine 的接口实现对应的人脸特征识别，使用需要开线程，因为计算时间比较长，需要预防界面假死
void sendFile(DataOutputStream out)	DataOutputStream 类型，用于发送消息	无	用于向服务器发送图片文件,使用了自己定义的协议

表 4

服务器端：

1. Er_server 是服务的实现，因为使用 python 实现，所以没有使用具体的类，写的比较简单直接在里面提供接口进行使用。具体介绍如表 5 所示。

接口名	参数说明	返回值说明	功能介绍
deal_data(conn, addr)	Conn 是传进来的套接字，addr 是地址	无	处理新连接的客户端，进行文件接收处理
predict(image_path)	path 传进来的文件路径	Int 类型，代表情感的类型	对图片进行情感识别

表 5

四、实现与实验

4.1 软件实现

1.使用了 Android 开发相关技术
2.使用了深度学习相关的框架 pytorch
3.使用 python 网络编程的相关技术

4.2 实验环境

[包括使用的硬件、软件（服务器/客户端操作系统，服务器、数据库、虚拟机等支撑软件）、实验场景（简单描述实验场景情况，并给出现场照片）]
客户端：
手机型号：Le X620
运行内存：3GB
总存储:32.0GB
Android 版本：6.0

服务器端：
电脑型号：Lenovo R720
处理器：Intel(R)Core(TM)i5-7300HQ 2.5GHZ 4 核
内存 RAM:8.00GB
硬盘:1T+128G
显卡：NVIDIA 1050
操作系统：Windows10
Pytorch 版本：1.3.1

使用的网上服务器训练，具体配置可以参照 https://www.kaggle.com/

4.3 实验步骤

实验采用的数据集为人脸数据集 fer2013 以及人脸数据集 CK+,针对这两种数据集进行了相对应的训练，训练了两个模型，然后利用最优模型进行人脸情感检测。

1.先对数据集进行处理，fer2013 文件是一个 csv 文件，所以需要先进行处理将其处理成 h5 文件。CK+人脸数据只是从其中挑选出了一些峰值图片，然后通过 python 处理成了 48*48 大小的图片，再将其转化成相对应的 h5 文件格式。

2.针对 fer2013 和 CK+分别写相对应的数据集类（继承 dataSet）,这是 pytorch 框架使用时，必须实现，这样才能通过 pytorch 框架进行训练。

3.编写网络模型 vggModel，采取的方式是继承了 vgg19 前面的 36 层，然后添加一层平均池化层以及全连接层
4.写 main 函数，在里面实现训练，如果可以跑通，将文件上传到网上的服务器上，进行相对应的训练。在服务器上设置 Bathch_size=128,学习率 learn_rate=0.001，对于图片使用 pytorch 提供的函数进行随机裁剪，然后进行随机的旋转增加鲁棒性。

5.编写 android 客户端的程序

6.编写服务器端的程序

五、结果与讨论

5.1 结果

1.使用 fer 数据集进行训练结果
Fer2013 数据集是一套人脸数据集，其中图像大小为 48*48，数据集中包含训练数据集 28709 个例子，公开测试集（public test）包含 3589 个例子，私有测试集（private test）3589 个例子，本次实习采用 Fer2013 作为其中的一个数据集进行了训练，训练了 250 轮，发现最终的测试集的准确度最高为 70%左右。其最终结果展示如下：

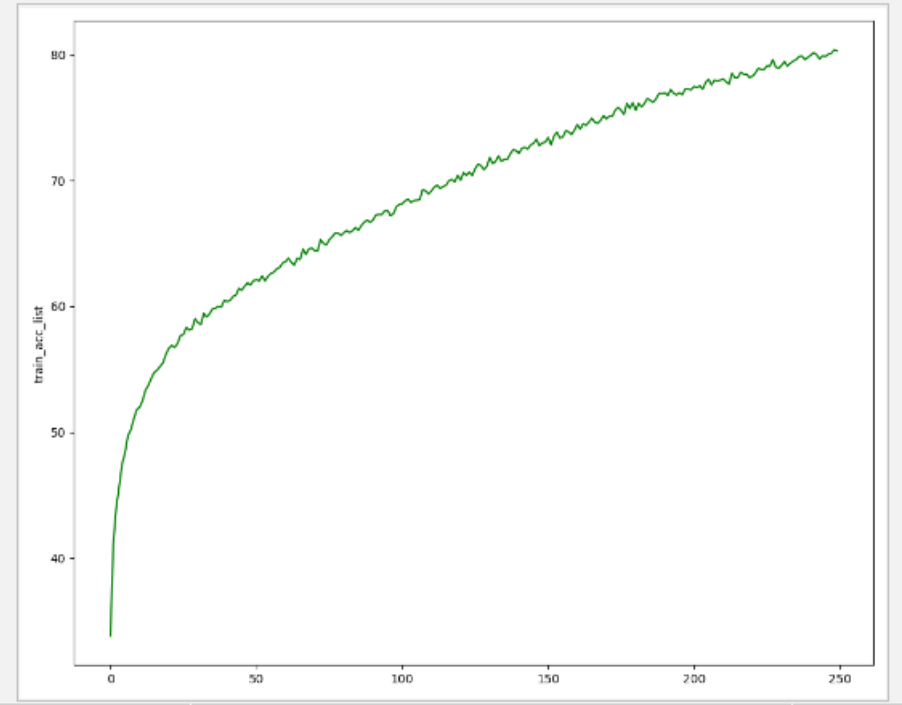


图 6：训练的准确度

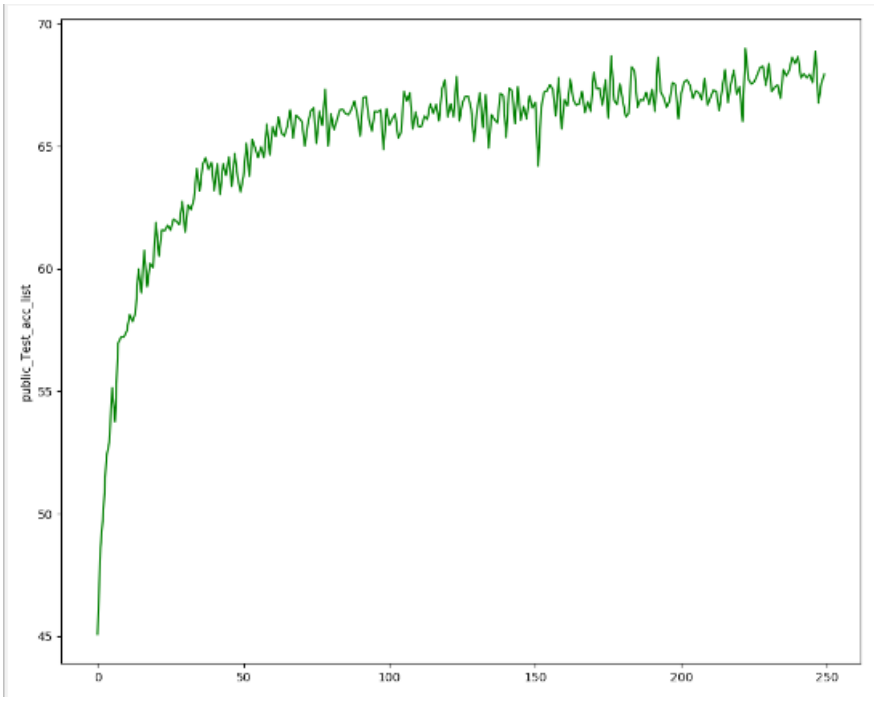


图 7：.PublicTest 的准确度

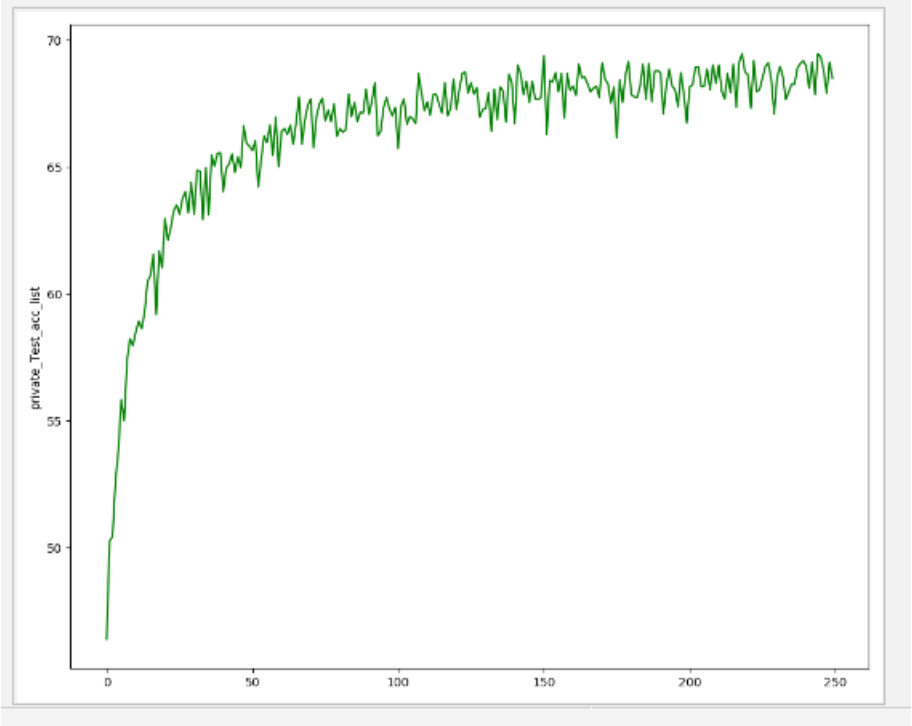


图 8：PrivateTest 准确度

2. 使用 CK+数据集进行训练结果

CK+是 CK 数据集的扩展，本次实习使用了 CK+中的部分图片，将其变成了 48*48 的大小，然后进行相对应的训练，将其以 8：2 的比例分成了训练集和测试集。最终训练 300 轮，在测试集上准确度为 95%左右。其具体结果如下图所示：

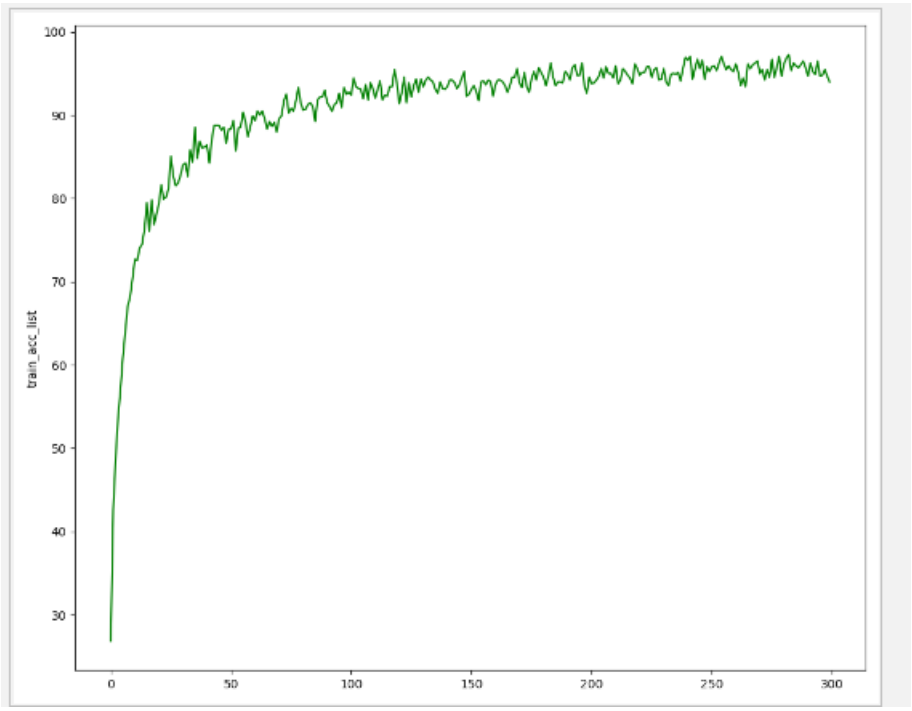


图 9：训练的准确度

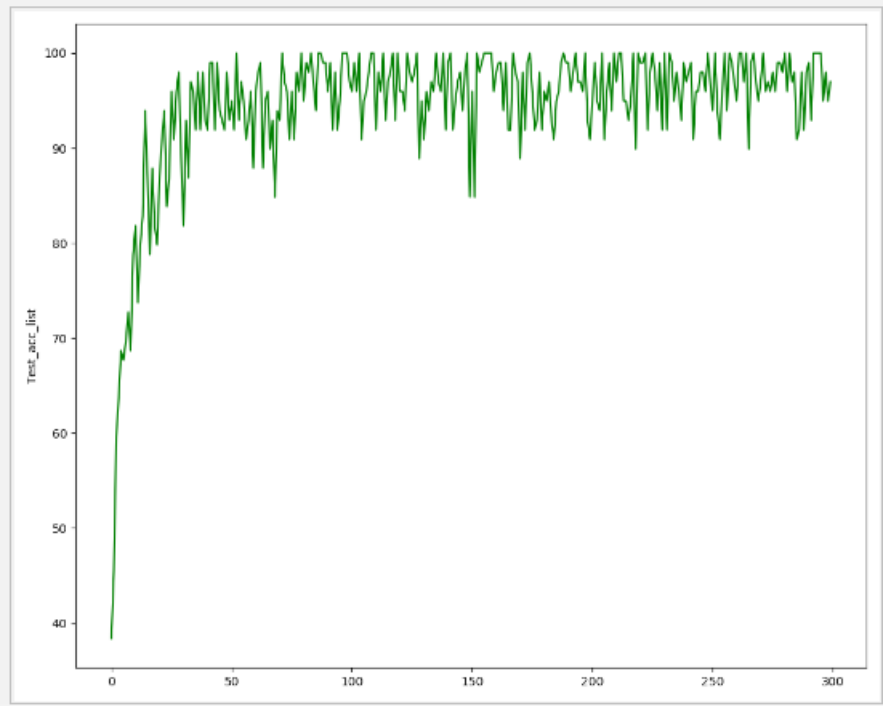


图 10：测试的准确度

3.APP 运行

客户端运行结果如下图所示：

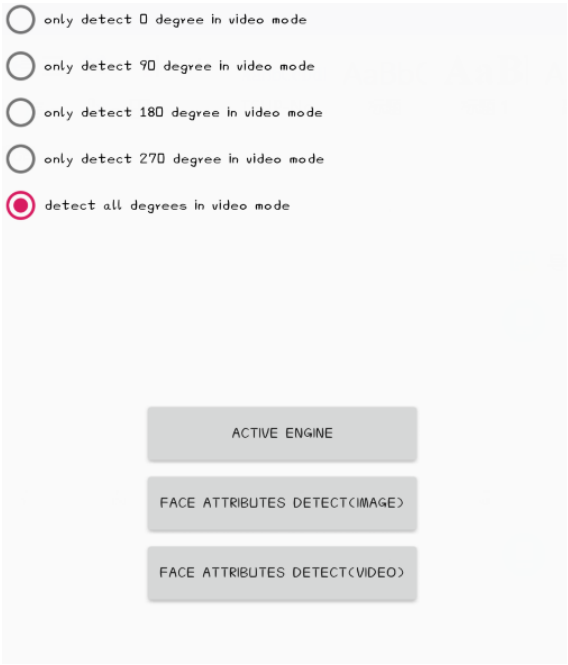


图 11：开始界面

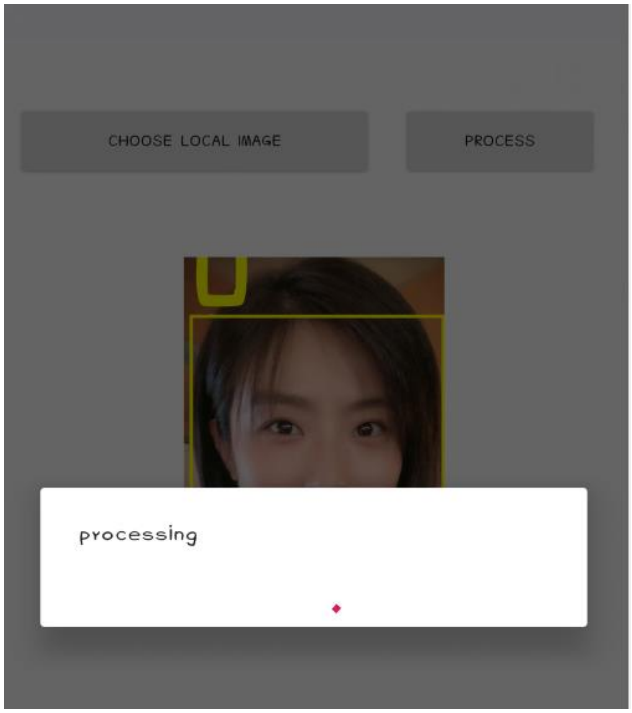
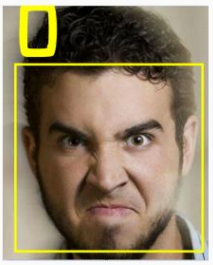


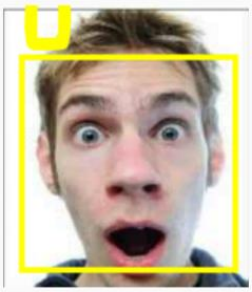
图 12：人脸情感识别时

图 13：识别结果 1



```
start face detection,imageWidth is 424, imageHeight is 524
detect result:
errorCode is :0   face Number is 1
face list:
face[0]:Rect(26, 121 - 411, 505),1

age of each face:
face[0]:33

gender of each face:
face[0]:MALE

face3DAngle of each face:
face[0]:Face3DAngle(yaw=-2.8274999, roll=1.3396235,
pitch=-11.445926, status=0)

liveness of each face:
face[0]:ALIVE

faceFeatureExtract:
faceFeature of face[0] extract success

the emtion is Angry
```

图 14：识别结果 2



```
start face detection,imageWidth is 212, imageHeight is 244
detect result:
errorCode is :0   face Number is 1
face list:
face[0]:Rect(16, 45 - 200, 229),1

age of each face:
face[0]:28

gender of each face:
face[0]:MALE

face3DAngle of each face:
face[0]:Face3DAngle(yaw=0.46905386, roll=1.5346426,
pitch=-0.38037735, status=0)

liveness of each face:
face[0]:ALIVE

faceFeatureExtract:
faceFeature of face[0] extract success

the emtion is Surprise
```

图 15：识别结果 3

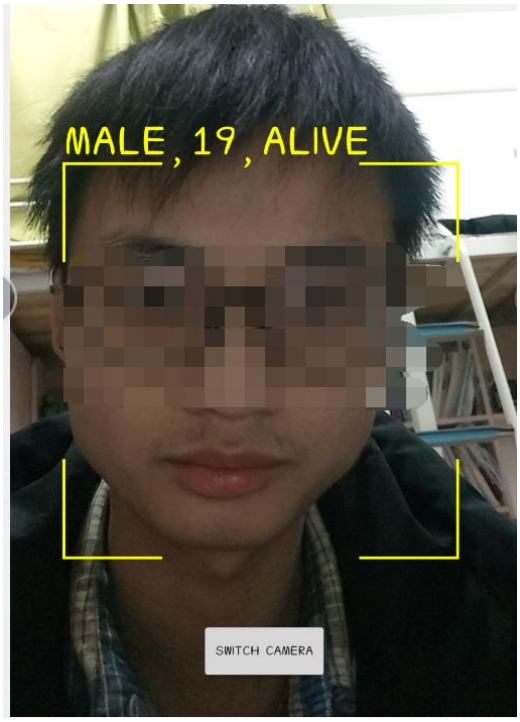


图 16：动态人脸特征提取（因为是现实中的人所以进行打码）

服务器端：

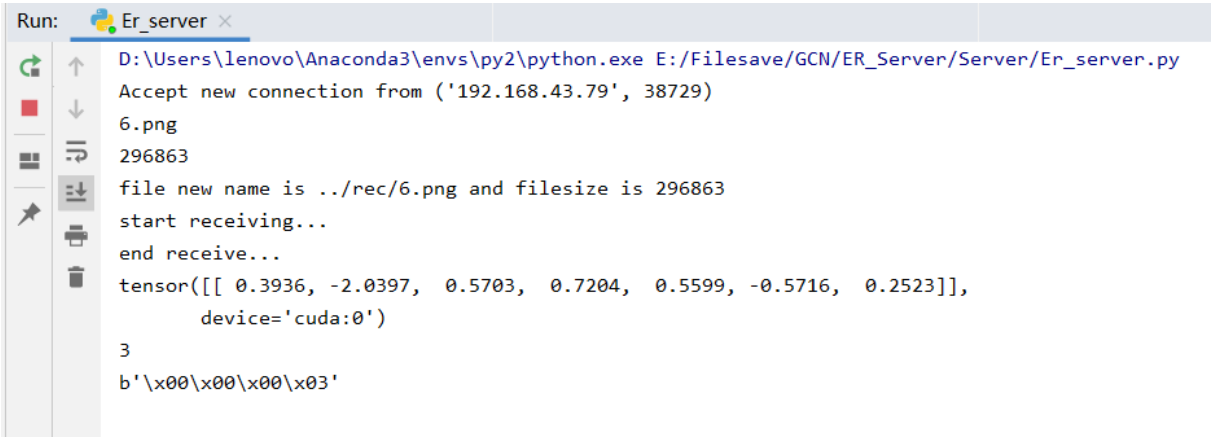


图 17：运行结果展示

5.2 讨论

1.对于结果准确度进行分析可知，使用 fer2013 数据集效果不如 CK+数据集训练的结果并不是特别好，进行查询之后好像说是 fer2013 训练集不好而 CK+因为都是实验室中得到的数据所以效果比较好。
2.目前使用人脸照片进行测试，发现如果照片中人脸比例大，测试的准确度要高很多，但是如果就是一张普通照片，当中有人脸，效果比较差，准确度下降很多，说明训练好的模型鲁棒性并不好。
3.目前客户端向移动端发送图片未进行压缩，是整张进行发送，比较消耗流量，应该实现进行压缩之后再进行发送。
4.目前程序还需要解决的问题是无法实现一张照片多个人脸同时进行人脸情感检测，这个目前未实现，想法是先通过人脸检测将人脸图像截取出来再进行发送，这样既能减小图片传输的大小也能够识别多张人脸的情感但是由于时间问题，并未进行实现。

5.3 特色与创新

1.能够在移动端实现人脸特征以及情感的提取，如果能够做的好以后可以运用到家庭智能机器人等方面，应用场景比较广泛

2.因为是在自己电脑端通过模型进行情感识别，如果以后训练出了更好的模型，可以及时的进行模型更换，耦合性比较好。比如：在产学研中我们就在研究更好的模型，研究出来后可以及时的进行替换。

六、总结

本次实习我将产学研目前学习到的知识实现了一次运用，将其通过网络实现了在移动端也能够识别情感，但是实现的并没有想象中那么准确，其实准确率还是比较低，但是整个流程进行了一遍对于产学研的学习以及 Android 编程方面的学习都是一个不小的提升。

目前移动端的人脸情感识别应用前景比较广泛，像推荐方面，通过识别情感之后推荐歌曲，社交方面，微博发照片自动识别情感等都有应用，腾讯在这方面曾经出过一款视频互动游戏，就是在通过开视频做出给定的表情，是我看过比较好的应用之一。我目前的研究方向是继续创建更好的模型，提升识别的准确度，希望在之后的学习当中能够顺利实现产学研，并将其想办法部署到移动端。

附件：设计图表和/或程序核心代码

VggModel 具体层数展示：

```
vggModel(
  (features): Sequential(
    (0): Conv2d(3, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (1): ReLU(inplace=True)
    (2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (3): ReLU(inplace=True)
    (4): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (5): Conv2d(64, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (6): ReLU(inplace=True)
    (7): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (8): ReLU(inplace=True)
    (9): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (10): Conv2d(128, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (11): ReLU(inplace=True)
    (12): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (13): ReLU(inplace=True)
    (14): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (15): ReLU(inplace=True)
    (16): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (17): ReLU(inplace=True)
    (18): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
```

```
    (19): Conv2d(256, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (20): ReLU(inplace=True)
    (21): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (22): ReLU(inplace=True)
    (23): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (24): ReLU(inplace=True)
    (25): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (26): ReLU(inplace=True)
    (27): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (28): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (29): ReLU(inplace=True)
    (30): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (31): ReLU(inplace=True)
    (32): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (33): ReLU(inplace=True)
    (34): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (35): ReLU(inplace=True)
    (36): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  )
  (avgpool): AdaptiveAvgPool2d(output_size=(7, 7))
  (classifier): Sequential(
    (0): Linear(in_features=25088, out_features=7, bias=True)
  )
)
```