嵌入式实验报告

封斯旸 计 61

1. 基础功能

(1) 视频传输

实现视频传输我使用的工具是 mjpg_streamer，mjpg-streamer 可以通过文件或者是 HTTP 方式访问 linux UVC 兼容摄像头，需要很少的 CPU 和内存资源就可以工作，大部分编码工作都是摄像头完成的，所以对于内存和性能都有限的树莓派十分适用。

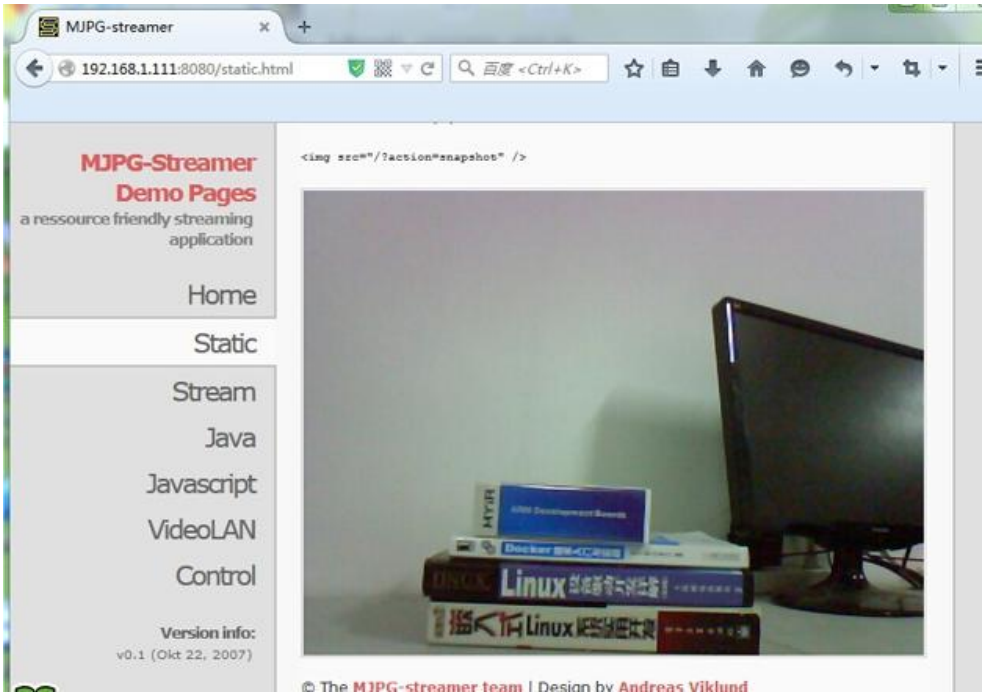
使用十分简单，首先 git 开源的 project 到本地，编译安装

```
sudo git clone https://github.com/jacksonliam/mjpg-streamer.git
cd mjpg-streamer/mjpg-streamer-experimental
make all
sudo make install
```

然后使用以下命令即可运行

```
./mjpg_streamer -i "./input_uvc.so" -o "./output_http.so -w ./www"
```

在我的电脑上访问树莓派对应的 IP 地址即可，效果如下图



以上方法简单便捷，延迟小，使用非常方便。

(2) 运动侦测

实现运动侦测我是每隔一秒获取一张图片，把当前图片和上一张图片进行比较，如果差异大于一个阈值，就把当前图像保存下来，代码逻辑如下

```
image1 = make_regalur_image(pic1)
image2 = make_regalur_image(pic2)
similarity = calc_similar(image1, image2)
# print("图片间的相似度为",similarity)
str_time = d.datetime.now().strftime("%Y.%m.%d-%H:%M:%S")
if (similarity > limit):
    image1.save("./Pic/Pos/"+str_time+"1.jpg")
    image2.save("./Pic/Pos/"+str_time+"2.jpg")
else:
    image1.save("./Pic/Neg/"+str_time+"1.jpg")
    image2.save("./Pic/Neg/"+str_time+"2.jpg")
```

如果两个图片的相似度较高，大于预设的阈值，就会将其保存在对应的文件夹中

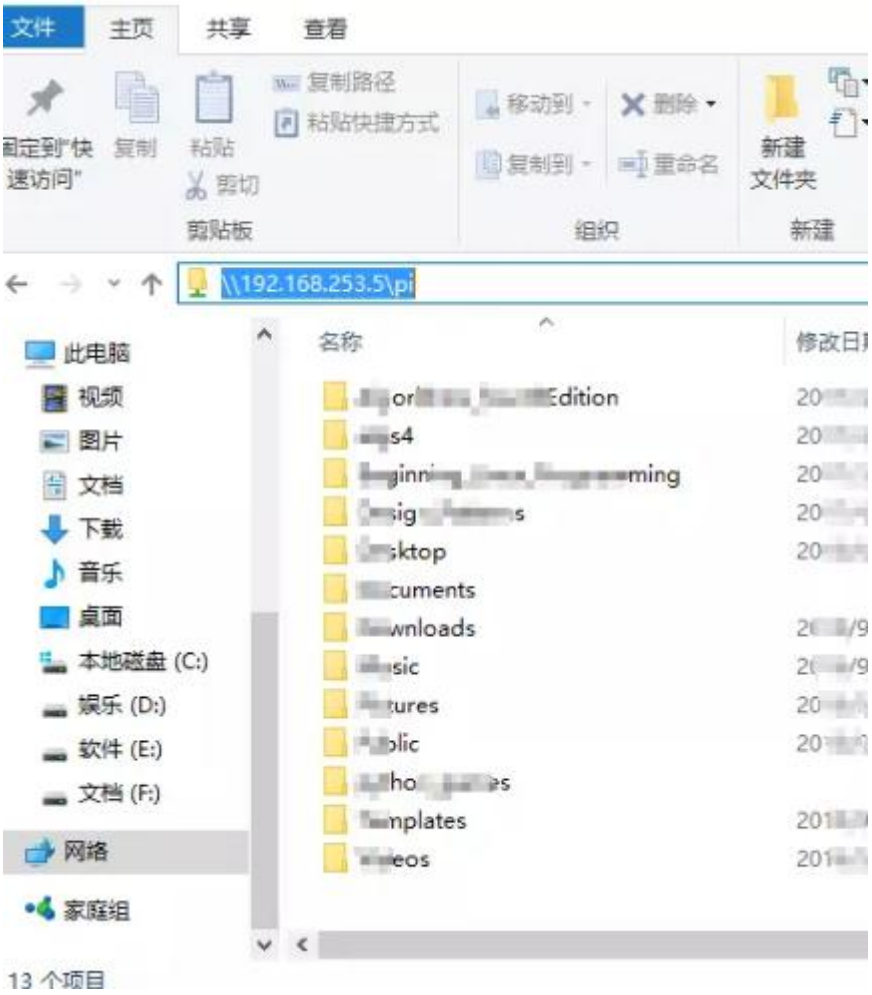
(3) 文件共享

为了实现 PC 开启浏览器可以查看树莓派上记录的图像，我使用了 Samba 工具。

SMB(Server Messages Block)协议:实现局域网内文件或打印机等资源共享服务的协议。

Samba 服务程序是一款基于 SMB 协议并由服务端和客户端组成的开源文件共享软件，实现了 Linux 与 Windows 系统间的文件共享。

效果如图，同时在浏览器也可以直接查看共享文件夹的文件。

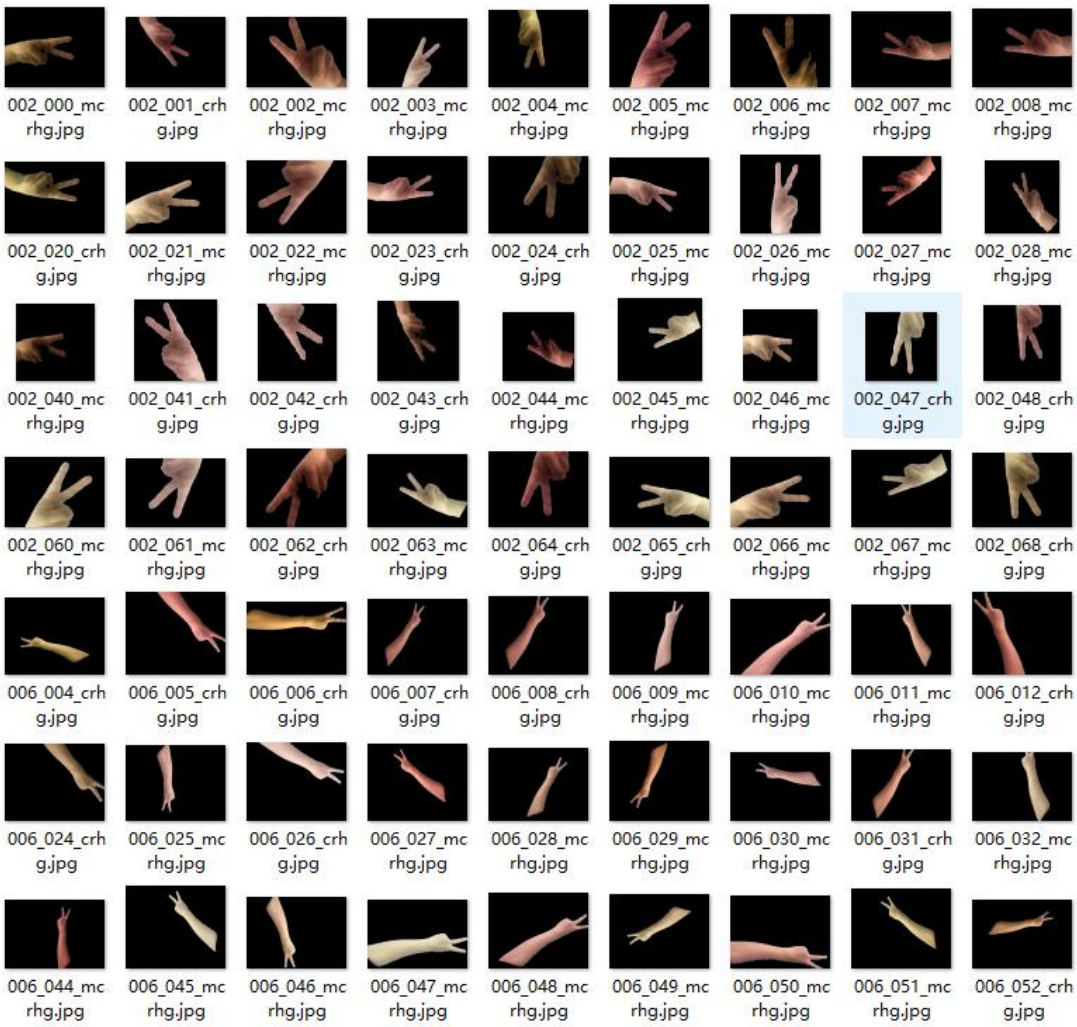


2. 剪刀石头布

(1) 数据扩充

由于给的数据集非常少，要想得到较好的训练效果必须使用数据增强的方法扩充数据集。

我使用随机的翻转，旋转，裁剪，变形，缩放等各类操作，以及噪声、模糊、颜色变换、擦除、填充等颜色数据增强操作，将原来的数据集扩充为每类 1000 个样本的大数据集，如下图：

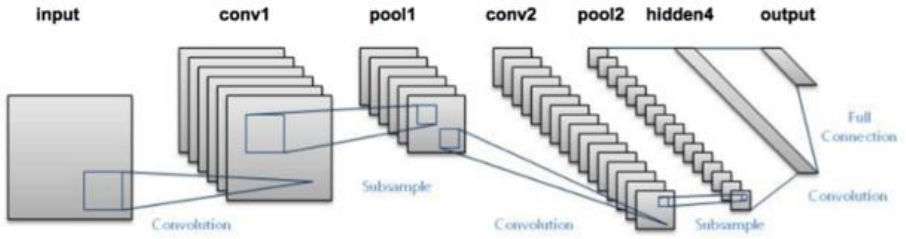


(2) 消除噪声

由于数据中背景带来的数据噪声较大，若是要想将手势作为唯一的判断标准，必须做噪声的消除，这里我采用了肤色识别来去除背景，原理如下：

(3) 模型训练

我使用的工具为 Keras，使用的网络模型为 LeNet，使用了 Adam 优化器，由于这个任务是一个多分类问题，所以使用类别交叉熵（categorical_crossentropy）。

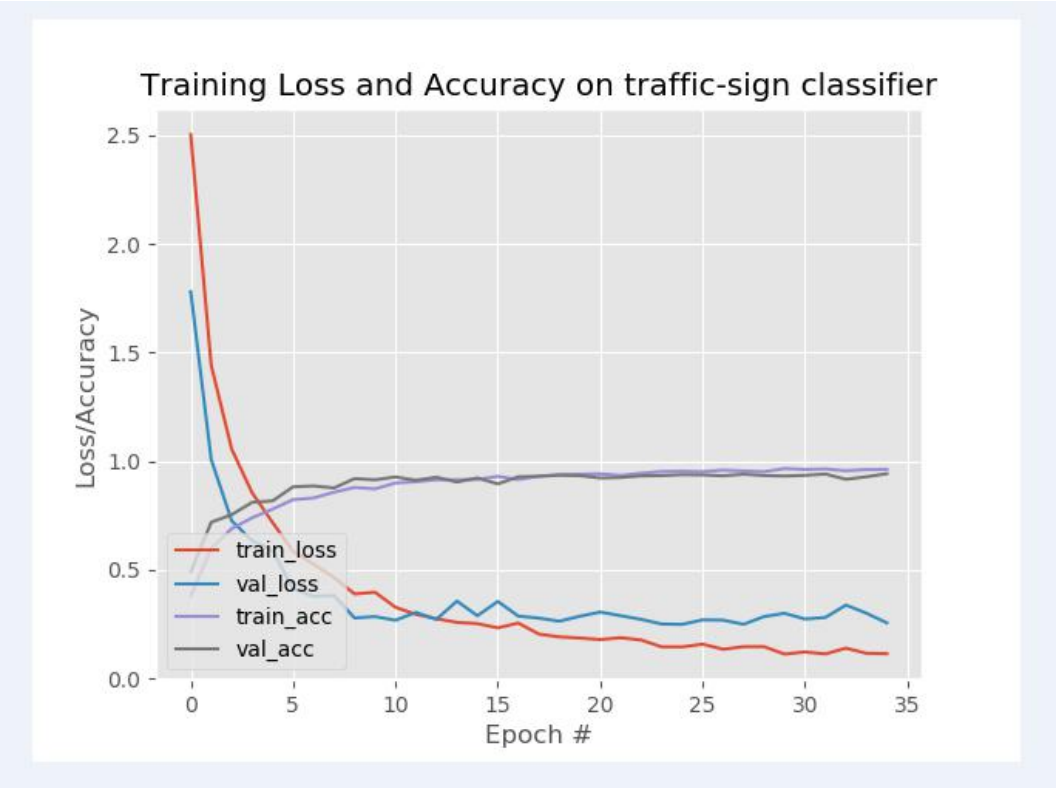


训练过程：

```
142/142 [==============================] - 13s - loss: 0.2948 - acc: 0.9051 - val_loss: 0.3039 - val_acc: 0.9119
Epoch 13/35
142/142 [==============================] - 13s - loss: 0.2746 - acc: 0.9144 - val_loss: 0.2717 - val_acc: 0.9258
Epoch 14/35
142/142 [==============================] - 13s - loss: 0.2581 - acc: 0.9143 - val_loss: 0.3561 - val_acc: 0.9036
Epoch 15/35
142/142 [==============================] - 13s - loss: 0.2522 - acc: 0.9157 - val_loss: 0.2885 - val_acc: 0.9226
Epoch 16/35
142/142 [==============================] - 14s - loss: 0.2327 - acc: 0.9298 - val_loss: 0.3542 - val_acc: 0.8952
Epoch 17/35
142/142 [==============================] - 15s - loss: 0.2546 - acc: 0.9155 - val_loss: 0.2869 - val_acc: 0.9278
Epoch 18/35
142/142 [==============================] - 14s - loss: 0.2041 - acc: 0.9306 - val_loss: 0.2775 - val_acc: 0.9302
Epoch 19/35
142/142 [==============================] - 15s - loss: 0.1911 - acc: 0.9375 - val_loss: 0.2633 - val_acc: 0.9353
Epoch 20/35
142/142 [==============================] - 14s - loss: 0.1858 - acc: 0.9399 - val_loss: 0.2860 - val_acc: 0.9333
Epoch 21/35
142/142 [==============================] - 14s - loss: 0.1788 - acc: 0.9416 - val_loss: 0.3055 - val_acc: 0.9222
Epoch 22/35
142/142 [==============================] - 15s - loss: 0.1875 - acc: 0.9341 - val_loss: 0.2886 - val_acc: 0.9246
Epoch 23/35
142/142 [==============================] - 14s - loss: 0.1767 - acc: 0.9440 - val_loss: 0.2717 - val_acc: 0.9321
Epoch 24/35
142/142 [==============================] - 14s - loss: 0.1456 - acc: 0.9519 - val_loss: 0.2505 - val_acc: 0.9333
Epoch 25/35
142/142 [==============================] - 13s - loss: 0.1451 - acc: 0.9533 - val_loss: 0.2487 - val_acc: 0.9377
Epoch 26/35
142/142 [==============================] - 14s - loss: 0.1577 - acc: 0.9513 - val_loss: 0.2694 - val_acc: 0.9365
Epoch 27/35
142/142 [==============================] - 13s - loss: 0.1345 - acc: 0.9586 - val_loss: 0.2684 - val_acc: 0.9325
Epoch 28/35
142/142 [==============================] - 13s - loss: 0.1459 - acc: 0.9549 - val_loss: 0.2492 - val_acc: 0.9397
Epoch 29/35
142/142 [==============================] - 14s - loss: 0.1466 - acc: 0.9511 - val_loss: 0.2846 - val_acc: 0.9337
Epoch 30/35
142/142 [==============================] - 14s - loss: 0.1120 - acc: 0.9656 - val_loss: 0.2997 - val_acc: 0.9313
Epoch 31/35
142/142 [==============================] - 14s - loss: 0.1220 - acc: 0.9617 - val_loss: 0.2736 - val_acc: 0.9345
Epoch 32/35
142/142 [==============================] - 15s - loss: 0.1127 - acc: 0.9639 - val_loss: 0.2806 - val_acc: 0.9405
Epoch 33/35
142/142 [==============================] - 14s - loss: 0.1393 - acc: 0.9558 - val_loss: 0.3376 - val_acc: 0.9163
Epoch 34/35
142/142 [==============================] - 14s - loss: 0.1158 - acc: 0.9610 - val_loss: 0.3002 - val_acc: 0.9278
Epoch 35/35
142/142 [==============================] - 14s - loss: 0.1138 - acc: 0.9621 - val_loss: 0.2557 - val_acc: 0.9421
[INFO] serializing network...

[python35] D:\CV数据集\classifier\traffic-sign-code>
```

Loss 和 Accuracy:



从训练效果看来，准确率在 94%左右，还阔以，实际在助教检查时能达到接近 70%的正确率。我自己在实际测试的时候也得到了较好的识别效果（可参看我提交的视频）。

3. 总结

这门课程结束了，我自己通过这门课从头开始接触和使用树莓派，从拿到一个全新的树莓派到将它改造成一个能识别手势的智能机器，这个过程给了我极大的收获了乐趣。其中我主要在图像识别训练上下了很大的功夫，在数据的处理以及模型的选择方面我获得了丰富的经验知识，谢谢老师助教一学期的大力帮助。