【2022秋】嵌入式系统实验报告

周永潇 2019011323 *2022/12/31*

实验内容

构建在树莓派3B+上能高效运行基于深度学习的即时图像识别程序的轻量级嵌入式系统。

> 网络模型可自行选择或训练，可利用计算神经棒NCS2、量化、多线程等多种方式加速网络推理过程，并采用从大到小或从小到大的方式剪裁系统，最终获得识别准确率高、cpu和内存占用小的轻量级嵌入式系统。

实验思路

我认为**轻量易用是嵌入式系统的基础**，因此神经网络训练、量化等操作并不是我实验的重点。我也尝试过利用神经棒对推理进行加速，但openvino的坑太多，最终放弃了这个想法。**我的工作量集中在系统剪裁上，并且后续同学能以此为基础，搭建更轻量、更高效的嵌入式系统。**

实验过程主要包括**配置剪裁系统镜像、剪裁图像识别程序、实验测试**等步骤，并且**使用了2张SD卡，各自安装1个系统，即插即用，以方便开发。**报告中部分名词解释如下

名词	定义
buildroot	用于构建自定义系统的工具
最小系统	使用buildroot构建的较小自定义系统。安装在SD卡1上。
生成系统	Raspberry Pi OS Lite，Debian Buster系统，安装了python3.7、opencv、openvino等开发环境，用于生成最小系统所需的可执行文件/动态链接库。安装在SD卡2上。
rpi-image-locating	实验过程中参考的github仓库，url：https://github.com/MashPlant/rpi-image-locating
raspberrypi3_defconfig	buildroot配置文件
rootfs_overlay	自定义目录，buildroot会在构建系统的最后一步将该目录复制到系统根路径，因此我们可以添加/覆盖某些系统文件。

1. 配置、剪裁系统

1.1 概述

采用从小到大的方式，利用buildroot构建较小的自定义系统。最小系统的大小约18MB。实现过程中参考了rpi-image-locating、https://github.com/vladreznikov/minimal_raspberrypi_config，buildroot的git仓库commit id为f298729fc3e3d71c01eab9e2939661fb50a7fe5a（截至2022年12月5日）。

1.2 剪裁linux

在raspberrypi3_defconfig中写入

```
BR2_LINUX_KERNEL_USE_CUSTOM_CONFIG=y
BR2_LINUX_KERNEL_CUSTOM_CONFIG_FILE="$(BR2_EXTERNAL)/board/raspberrypi3/linux.config"
```

buildroot会使用linux.config文件自定义linux系统。

我使用的linux.config基于rpi-image-locating修改，它去掉了

> - 一些不需要的硬件驱动，如蓝牙， NFC， USB， I2C， SPI， W1 等。
> - 一些不需要的协议，如 HID， IIO， HWMON 等；无用的网络协议，例如 IPSec。
> - File systems 下各种文件系统的支持，只保留 ext4 即可，也不需要 Native language support。
> - 不需要大部分加密算法，留下的几个都是无法关闭的。需要先关闭一些网络协议才能关闭一些加密算法。
> - 不需要 trace， profile， kprobe 等功能，因此也不需要内核符号 KALLSYMS 。

但由于rpi-image-locating最终构建的系统不含ssh远程登陆功能，并且关闭了linux的tty选项，导致安装openssh后无法远程登陆树莓派。查阅各种资料后发现，我们需要在linux.config写入

```
CONFIG_TTY=y
```

才能正常进行ssh登陆。

当然，busybox自带的linux配置开启了tty选项，不过自行剪裁linux过程繁杂，不建议直接使用。

1.3 剪裁BusyBox

> 百度百科：*BusyBox*是一个集成了三百多个最常用Linux命令和工具的软件。*BusyBox* 包含了一些简单的工具，例如ls、cat和echo等等，还包含了一些更大、更复杂的工具，例grep、find、mount以及telnet。

busybox会默认开启BusyBox，并使用buildroot仓库中的默认配置文件进行安装。我们可以在make后build文件夹.config文件夹中发现最终生成的buildroot配置，包含

```
BR2_PACKAGE_BUSYBOX=y
BR2_PACKAGE_BUSYBOX_CONFIG="package/busybox/busybox.config"
```

我们可以在raspberrypi3_defconfig中写入

```
BR2_PACKAGE_BUSYBOX_CONFIG="你的busybox config文件路径"
```

以剪裁busybox。

rpi-image-locating中的busybox.config去掉了一些无用的软件和库函数，如编辑器，计算器， find工具等，可以作为参考。不过busybox并不大，约1M左右，剪裁收益有限。

1.4 设备管理

busybox默认会使用busybox中的mdev工具管理系统设备，如网卡、摄像头等。我们需要在/etc/init.d文件夹下放置mdev配置文件，树莓派启动后才会识别到各种系统设备，**否则会导致找不到wlan0、eth0、video0等情况。**

我参考rpi-image-locating，**使用eudev自动管理系统设备**，安装后无需其他配置即可正常识别系统设备。
raspberrypi3_defconfig配置如下

```
# eudev for wifi-driver and camera management
BR2_ROOTFS_DEVICE_CREATION_DYNAMIC_EUDEV=y
BR2_PACKAGE_EUDEV=y
BR2_PACKAGE_PROVIDES_UDEV="eudev"
BR2_PACKAGE_HAS_UDEV=y
BR2_PACKAGE_HOST_EUDEV=y
```

1.5 摄像头

为了使用摄像头设备/dev/video0，树莓派需要在启动时使用带"X"后缀的固件，树莓派启动配置和其余类型的固件含义可参考官方文档。

我们在raspberrypi3_defconfig中配置

```
BR2_PACKAGE_RPI_FIRMWARE=y
BR2_PACKAGE_RPI_FIRMWARE_BOOTCODE_BIN=y
BR2_PACKAGE_RPI_FIRMWARE_VARIANT_PI_X=y #  (x for camera support)
BR2_PACKAGE_RPI_FIRMWARE_CONFIG_FILE="$(BR2_EXTERNAL)/board/raspberrypi3/config.txt"
```

并在config.txt中写入

```
start_file=start_x.elf
fixup_file=fixup_x.dat
```

即可。

1.6 连接wifi

1.6.1 wifi固件安装

我使用buildroot版本和rpi-image-locating使用版本的不同，移除了某些旧配置。**为了让树莓派连上wifi，需要安装wifi固件。我还使用了wpa_supplicant工具（也可使用busybox自带的iw工具，具体操作可参考rpi-image-locating），并安装NL80211相关包，否则会在连接时报错。**

```
BR2_PACKAGE_BRCMFMAC_SDIO_FIRMWARE_RPI=y
BR2_PACKAGE_BRCMFMAC_SDIO_FIRMWARE_RPI_WIFI=y
BR2_PACKAGE_WPA_SUPPLICANT=y
BR2_PACKAGE_WPA_SUPPLICANT_NL80211=y
```

1.6.2 配置wpa_supplicant

在rootfs_overlay创建配置文件/etc/wpa_supplicant.conf，树莓派开机后即可自动连接wifi。

```
network={
        ssid="wifi名称"
        psk="wifi密码（记得加引号）"
}
```

1.6.3 配置网络接口

在rootfs_overlay中创建文件/etc/network/interfaces，进行网络接口配置，具体语法可参考https://wiki.debian.org/NetworkConfiguration。iface用于描述一个网络接口，`pre-up wpa_supplicant -B -Dnl80211 -iwlan0 -c/etc/wpa_supplicant.conf`表示使用wpa_supplicant，根据wpa_supplicant.conf的配置并通过NL80211连接wifi。

```
auto lo
iface lo inet loopback
auto eth0
iface eth0 inet dhcp
    pre-up /etc/network/nfs_check
    wait-delay 15
auto wlan0
iface wlan0 inet dhcp
    pre-up wpa_supplicant -B -Dnl80211 -iwlan0 -c/etc/wpa_supplicant.conf
    post-down killall -q wpa_supplicant
    wait-delay 15
iface default inet dhcp
```

1.7 ssh登陆

1.7.1 安装openssh

在raspberrypi3_defconfig中配置

```
BR2_PACKAGE_OPENSSH=y
BR2_PACKAGE_OPENSSH_CLIENT=y
BR2_PACKAGE_OPENSSH_SERVER=y
BR2_PACKAGE_OPENSSH_KEY_UTILS=y
BR2_PACKAGE_OPENSSH_SANDBOX=y
```

1.7.2 配置ssh和root密码

在rootfs_overlay中创建文件/etc/ssh/sshd_config，基于openssh默认配置，修改一行

```
PermitRootLogin yes
```

以允许root登陆。

在raspberrypi3_defconfig中设置root密码

```
BR2_TARGET_ENABLE_ROOT_LOGIN=y
BR2_TARGET_GENERIC_ROOT_PASSWD="root"
```

即可正常进行ssh登陆。

1.8 根文件系统配置

在raspberrypi3_defconfig中配置ext4文件系统，如果在buildroot过程发生文件系统空间不足的错误，修改`BR2_TARGET_ROOTFS_EXT2_SIZE`扩大根文件系统空间即可，**不过这会直接影响最终生成的系统镜像大小。**

```
# Filesystem / image
BR2_TARGET_ROOTFS_EXT2=y
BR2_TARGET_ROOTFS_EXT2_4=y
BR2_TARGET_ROOTFS_EXT2_SIZE="216M"
# BR2_TARGET_ROOTFS_TAR is not set
BR2_ROOTFS_POST_BUILD_SCRIPT="$(BR2_EXTERNAL)/board/raspberrypi3/post-build.sh"
BR2_ROOTFS_POST_IMAGE_SCRIPT="$(BR2_EXTERNAL)/board/raspberrypi3/post-image.sh"
BR2_ROOTFS_OVERLAY="$(BR2_EXTERNAL)/board/raspberrypi3/rootfs_overlay"
```

`BR2_ROOTFS_OVERLAY`可以配置用以覆盖系统根文件系统的目录路径。

`BR2_ROOTFS_POST_BUILD_SCRIPT`会在buildroot build完成后运行，可以利用它进行进一步的系统剪裁，如去掉eudev中某些不需要的功能。

2. 剪裁图像识别程序

2.1 程序实现

编程语言	优点	缺点
python	通过tensorflow、socketio等包可以快速实现深度学习、与服务器交互等功能	需要安装opencv、numpy、tflite_runtime、openvino等多个包，**会显著增加disk usage。**以opencv为例，自行在树莓派上编译需要安装相关工具链，而且编译速度慢(40min+)，最终的剪裁收益也不高。如果在最小系统上安装python执行环境以及相关包，不仅操作复杂，最终镜像大小会增加到几个G，那么使用buildroot剪裁系统镜像就变得没有意义。
C/C++	编译成可执行文件后，**在最小系统上运行无需安装其他依赖**	需要使用C/C++重新编写识别程序，并实现与服务器网络交互、运行深度学习模型等功能，工作量比较大。

最终**我选择使用python的pyinstaller包，在生成系统上打包图像识别程序，并通过rootfs_overlay部署到最小系统上。**pyinstaller会将python源文件编译成一个可执行文件，并打包相关的动态链接库。我们也可以添加-F参数，将python源文件和相关动态库一起打包成一整个可执行文件，进一步减少空间占用。我们只需将得到的可执行文件部署到最小系统上，既无需安装其他依赖，又可以减少空间占用，兼具上述两种编程语言的优点。**最终图像识别程序的大小约为100MB。**

注意事项

pyinstaller在打包过程中会缺失某些动态链接库，如libGL.so.1等，导致图像识别程序运行在最小系统上时报错。缺失的库数量并不多，约10个左右，我们只需根据其报错信息，在生成系统的/usr/lib等路径中找到缺失的动态链接库，复制到最小系统的/usr/lib路径下即可。

2.2 剪裁streamer

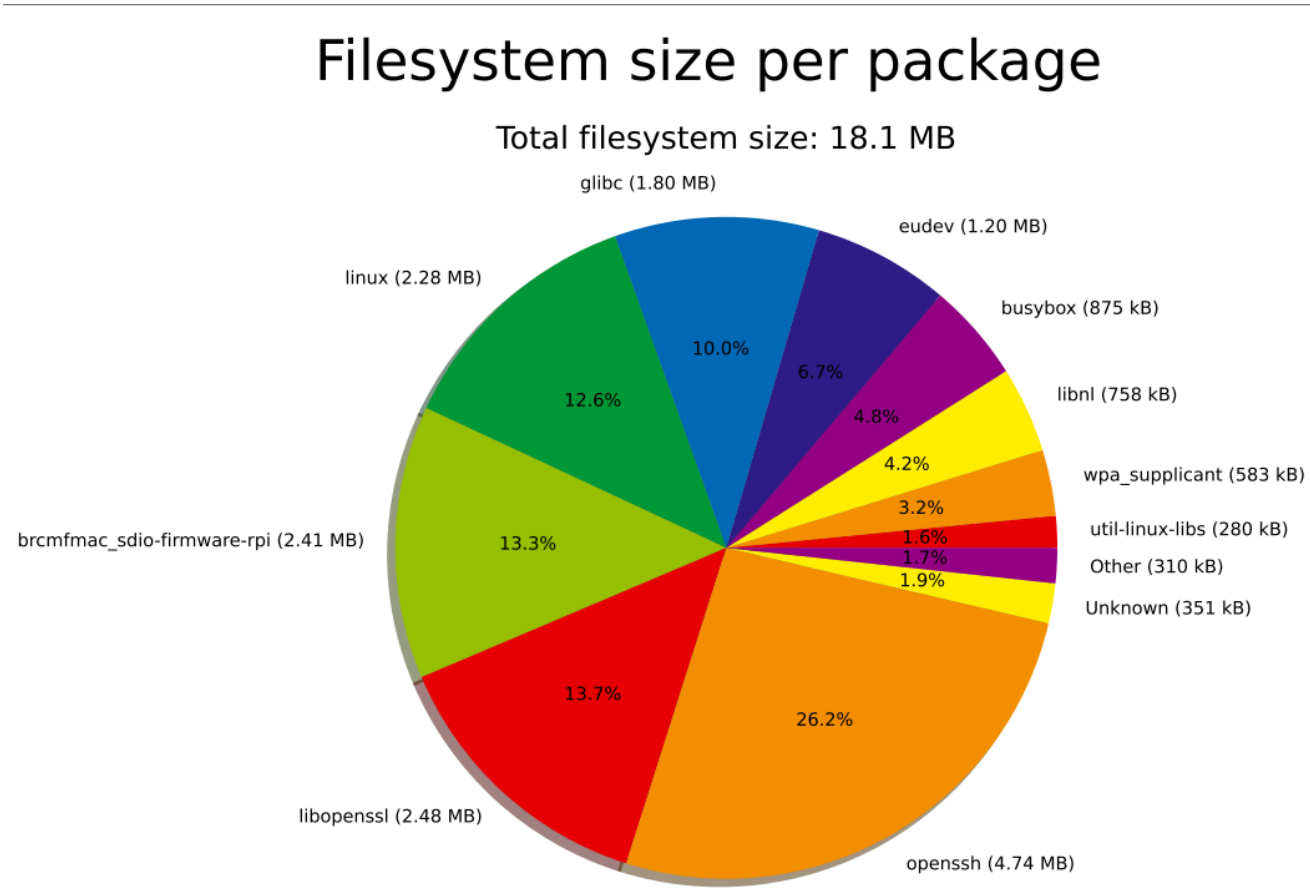
识别程序需要将视频流发送到服务器，否则无法从网页上获取此时树莓派拍摄到的图像，不便于实验。

以mjpg-streamer为例，可在buildroot配置中进行安装，**但buildroot使用的mjpg-streamer版本有未知bug，不能正常运行，**需要我们使用新版本自行编译。由于最小系统不含编译工具链，我们需要在生成系统上提前编译好，安装教程参考 https://github.com/cncjs/cncjs/wiki/Setup-Guide:-Raspberry-Pi-%7C-MJPEG-Streamer-Install-&-Setup-&-FFMpeg-Recording

和图像识别程序类似，直接将mjpg-streamer可执行文件放置到/usr/bin，input_uvc.so和output_http.so放置到/usr/lib，运行mjpg-streamer也会缺少某些动态链接库，如libv4l2.so.0，因为我们的最小系统上并没有安装这些功能。同样，我们从生成系统上将缺失的动态链接库复制到最小系统即可。

3. 系统镜像大小

描述	空间占用	备注
根文件系统(linux、各种系统功能等，不含图像识别程序)	**约18MB**	**可进一步剪裁**，如linux、eudev、busybox等。如果不需要ssh远程登陆功能，也可以去掉openssh和openssl。不过剪裁过程比较繁杂，收益不大。
图像识别程序（以及相关动态链接库）	**约100MB**	**可进行大量剪裁。**我利用apt安装的opencv包含许多不需要的功能，pyinstaller将python源代码编译成的可执行文件会加载所有opencv相关的动态链接库（即使python代码中没使用到相关功能，应该是opencv的python binding会加载这些库），如aruco等。numpy、tflite_runtime等其他python包的情况类似，应该也有办法可以剪裁。受时间所限，没有对这部分内容进行深入研究。
神经网络模型	**约60MB**	有`mobilenet-v1-1.0-224.tflite`、`mobilenet-v2-1.0-224.tflite`、`mobilenet-v3-small-1.0-224.tflite`和`mobilenet-v3-large-1.0-224.tflite`一共4个模型。可根据需要添加/删除。
df命令得到的Used Size	**183MB**	\
boot.vfat文件	**32MB**	buildroot打包的树莓派启动文件，包含start_x.elf等。
rootfs.ext2/rootfs.ext4	**216MB**	`BR2_TARGET_ROOTFS_EXT2_SIZE`设置的根文件系统大小，需要能容纳linux、各种系统功能、图像识别程序和神经网络模型。
sdcard.img	**248MB**	boot.vfat + rootfs.ext2，**最终系统镜像大小，可直接刷入SD卡中。**



4. 实验测试

我尝试了`mobilenet-v1-1.0-224.tflite`、`mobilenet-v2-1.0-224.tflite`、`mobilenet-v3-small-1.0-224.tflite`和`mobilenet-v3-large-1.0-224.tflite`四个模型，**并未进行fine-tune、量化、多线程等其他优化，仅使用助教提供的实验框架进行测试。**

4.1 各模型在valid数据集上的测试结果

编写脚本model_test.py，分别读取valid数据集中的原始图片，测试各模型的识别准确率。

模型名称	识别准确率	fps
`mobilenet-v1-1.0-224.tflite`	**61%**	1.3
`mobilenet-v2-1.0-224.tflite`	53%	1.7
`mobilenet-v3-large-1.0-224.tflite`	50%	2.0
`mobilenet-v3-small-1.0-224.tflite`	40%	**4.0**

4.2 各模型在test数据集上的测试结果

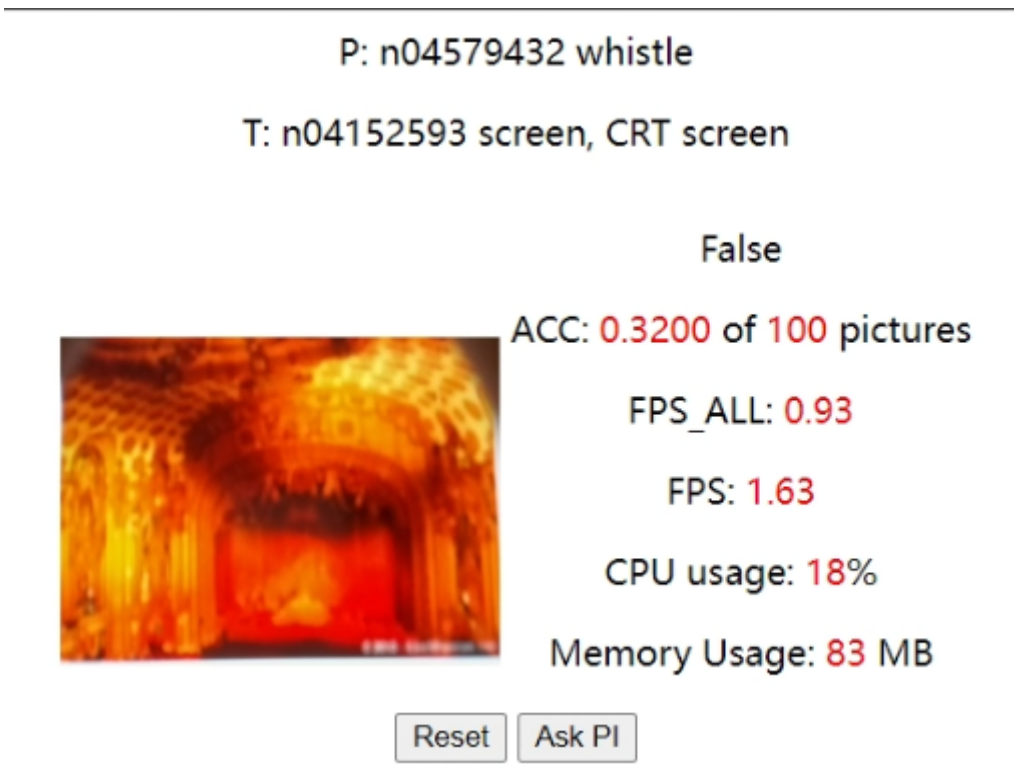
分别读取test数据集中的原始图片，测试各模型的识别准确率。

模型名称	识别准确率	fps
`mobilenet-v1-1.0-224.tflite`	**64%**	1.3

模型名称	识别准确率	fps
`mobilenet-v2-1.0-224.tflite`	58%	1.7
`mobilenet-v3-large-1.0-224.tflite`	58%	2.0
`mobilenet-v3-small-1.0-224.tflite`	46%	**4.0**

4.3 真实环境测试结果

树莓派通过摄像头即时拍摄屏幕上的图像，进行识别并与服务器进行交互。**使用mobilenet-v1-1.0-224.tflite模型，并调整display_delay为0.2**，在test集上识别结果如下：



可以发现，利用最小系统部署程序，**内存使用比较小（约80MB）**。

可能由于摄像头模糊、未进行fine-tune等原因，即使将display_delay调得很大，各种模型在真实环境中的识别准确率仍有显著下降，除mobilenet-v1-1.0-224.tflite外其他3个模型平均识别准确率都在15%~25%之间，并且经常波动，鲁棒性比较差。

5. 实验总结

我在实验过程中进行了多种系统级和程序级的剪裁，得到了250MB的轻量级嵌入式系统镜像，并且支持ssh、wifi、摄像头等多种功能。最小系统的镜像可在20秒内烧录进SD卡中，方便系统部署。经过实验测试，系统运行过程中能显著减少内存占用（从180MB到80MB）。后续同学能以此为基础，继续剪裁系统，或者利用神经棒加速推理过程（在生成系统上编写完python代码后，只需将利用pyinstaller打包并部署到最小系统）、部署量化后的神经网络等，优化图像识别速度和准确率。