

项目 9 嵌入式 Python 开发

项目描述

目前物联网、人工智能已经深入到医疗、家居、交通、教育和工业等多个领域，正在极大改变人们的日常生活。树莓派（Raspberry Pi）设计用于教育，目前已进化到第 4 代。树莓派不仅廉价而且周边设备多，互联网上有各种各样的接口设备、有趣的项目应用案例资料。诞生了大量的应用项目，作为可轻松开发程序的平台，知名度越来越高，受到物联网与人工智能项目开发人员的欢迎。在自己项目上采用采用 Raspberry Pi 时，“想使用特定的传感器”、“想与周边装置通信”、“想连接云服务”等，都能够快速的获得现存的解决方案，可以节约为实现这些功能所需要的巨大的开发成本。

传统的软件开发以瀑布模型的开发手法为主，如今要求缩短开发周期，要求开发新颖性高的软件，因此能够缩短开发周期的敏捷开发受到关注。敏捷开发是一种应对快速变化需求的一种软件开发模式，描述了一套软件开发的价值和原则。此模式中，自组织的跨功能团队在紧密的协作中发掘用户或顾客的需求以及改良解决方案，此模式也强调适度的项目、进化开发、提前交付与持续改进，并且鼓励快速与灵活的面对开发与变更。这些原则支持许多软件开发方法的定义和持续进化。即使采用这样的开发手法，树莓派的开发环境是 Linux，Python 是树莓派的官方编程语言，可简单获取各种开源项目案例，使得树莓派的开发项目从原型的验证到实际运行得以顺利推进。

项目目标

知识目标

1. 掌握树莓派开发板
2. 掌握 NVIDIA Jetson Nano 开发
3. 掌握 [TensorFlow Lite、OpenCV](#)

技能目标

1. 掌握配置树莓派 Python 环境
2. 掌握安装与配置 Jupyter lab
3. 掌握通用输入/输出接口（GPIO）
4. 掌握配置 NVIDIA Jetson Nano 开发环境
5. 掌握[人脸识别的门禁系统](#)
6. 掌握花卉识别

项目 9.1 配置树莓派开发环境

9.1.1 配置树莓派 Python 环境

Python 是一种功能强大的编程语言，易于使用，易于阅读和编写，Python 与树莓派结合可以将您的项目与现实世界轻松的联系起来。

树莓派默认已安装了 Python，打开终端窗口，执行 `python` 来测试是否安装了 Python 开发环境，并查看当前的 Python 版本。

```
# python
Python 3.9.2 (default, Feb 28 2021, 17:03:44)
[GCC 10.2.1 20210110] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

Python2.7 官方已经停止维护了，树莓派目前安装的是 Python 3.9.2 版本。如果你的系统预装的 Python 版本不是 3.9.2，请将其升级。升级最简单的方法是直接更新树莓派系统，如果由于各种原因，不能升级树莓派系统，可以使用以下方法升级，这也适用于安装其它版本的 Python，只要下载时选择特定版本可以了。

1. 更新树莓派系统，整个系统升级到最新

```
# sudo apt-get update
# sudo apt-get upgrade
# sudo apt-get dist-upgrade
```

更新系统需要 root 权限，如果更新数据慢可以考虑跟换源，国内有很多源可以选择，例如阿里、清华等，其中清华源如图 9-1 所示。



图 9-1 清华 Raspbian 软件仓库镜像

使用 nano 或者 vi 编辑工具修改软件源的配置文件 `/etc/apt/sources.list`。

2. 安装依赖

安装 python3.9.2 需要的依赖。有些软件已经存在，会自动忽略。

```
# sudo apt-get install build-essential libncurses-dev libreadline-dev  
libsqlite3-dev libssl-dev libexpat1-dev zlib1g-dev libffi-dev  
...  
建议安装:  
ncurses-doc readline-doc sqlite3-doc libssl-doc  
下列【新】软件包将被安装:  
libffi-dev libncurses-dev libreadline-dev libsqlite3-dev libssl-dev  
升级了 0 个软件包, 新安装了 5 个软件包, 要卸载 0 个软件包, 有 1 个软件包未被升级。  
...
```

3. 下载解压 Python 源码包

从 Python 官网下载 3.9.2 版源码，并把源码解压到当前目录下。

```
# cd ~  
# wget https://www.python.org/ftp/python/3.9.2/Python-3.9.2.tgz  
# tar zxvf Python-3.9.2.tgz
```

4. 配置、编译、安装

如果顺利的话整个编译过程需要 1 个小时，编译后源码的目录会增加到 130 MB。可以选择把新版 Python 安装到 /opt/python3.9 目录下，或者安装在 /usr/bin/python3.9 目录下。

```
# cd ./Python-3.9.2  
# ./configure --prefix=/opt/python3.9  
# make  
# make  
# sudo make install
```

在 make 命令后如果提示 Python 模块无法编译，需要按照错误提示排查原因，通常是没安装相应的依赖包。

5. 创建软链接

make install 成功运行后，Python 相关程序模块会拷贝到 /opt/python3.9。在创建链接后就可以启动 Python 3.9.2 了。

创建 /usr/bin/python 软链接指向 python 3.9，并创建一个 pip 的软链接。pip3 已经被官方集成到 Python 3.9 里，它用于安装第三方模块。

```
# sudo ln -s /opt/python3.9/bin/python3.9 /usr/bin/python  
# sudo ln -s /opt/python3.9/bin/pip3 /usr/bin/pip3
```

9.1.2 安装与配置 Jupyter lab

提起 Jupyter Notebook，很多学习过 Python 的同学都不陌生。Jupyter 优点很多，例如功能强大，交互式、富文本，还有丰富的插件、主题修改、多语言支持等等。

JupyterLab 是 Jupyter Notebook 的全面升级。JupyterLab 是一个集 Jupyter Notebook、文本编辑器、终端以及各种个性化组件于一体的全能 IDE，下面介绍如何安装与配置 Jupyter lab。

1. 安装 Jupyter lab

通过 pip 安装 Jupyter lab，如果网络环境较差导致下载软件包慢，可以考虑更换源。

```
# pip3 install jupyterlab -i https://pypi.tuna.tsinghua.edu.cn/simple
```

jupyterlab-3.6.2 文件大小是 9.9 MB，安装成功后就可以进行下一步配置了。

如果安装出现提示“ERROR: THESE PACKAGES DO NOT MATCH THE HASHES FROM THE REQUIREMENTS FILE.”则可在 pip install 添加--no-cache-dir 参数，如果问题依旧，可手动下载安装需要的 Package 后用 pip 安装。

2. 配置文件

安装好以后，就是配置环节了，首先需要创建配置文件。

```
# jupyter notebook --generate-config
Writing default config to: /home/pzy/.jupyter/jupyter_notebook_config.py
```

配置文件不需要自己在某个文件夹下创建，是由 jupyter 软件生成的。运行成功后配置文件是/home/pi/.jupyter/jupyter_notebook_config.py。如果没有该文件则需要检查是否输入正确或重新安装一下 jupyterlab。

使用 vi 修改配置文件，配置文件路径，请使用修改为对应的路径。

```
# vi/home/pzy/.jupyter/jupyter_notebook_config.py
## notebook 服务会监听的 IP 地址。
c.NotebookApp.ip = '*'
## 用于 notebooks 和内核的目录。
c.NotebookApp.notebook_dir = '/home/pzy'
## The port the notebook server will listen on (env: JUPYTER_PORT).
c.NotebookApp.port = 8888
## Whether to open in a browser after starting.
c.NotebookApp.open_browser = False
```

配置文件内容比较多，需要修改运行服务监听的 IP 地址，端口，用于 notebooks 内核的目录，是否打开浏览器。

然后设置 Jupyter lab 的访问密码，这一步并不是必须做的，访问密码为空也是正常的，但是建议使用。

```
# jupyter notebook password
```

输入密码，输入后回车即可。在输入密码的状态下，键盘按下字符是没有任何显示的，继续输入最后回车即可。如果配置文件中有错误，这时会提示，请注意提示信息。

重启树莓派后就可以尝试启动 Jupyter lab。

```
# jupyter lab
...
[I 19:08:53.896 ServerApp] notebooks 运行所在的本地路径: /home/pzy
[I 19:08:53.898 ServerApp] Jupyter Server 2.5.0 is running at:
[I 19:08:53.900 ServerApp] http://localhost:8888/lab
[I 19:08:53.902 ServerApp] http://127.0.0.1:8888/lab
[I 19:08:53.904 ServerApp] 使用 Control-C 停止此服务器并关闭所有内核（连续操作两次便可跳过确认界面）。
[I 19:10:29.431 LabApp] 302 GET /lab (@192.169.3.130) 54.90ms
[I 19:10:34.120 ServerApp] Generating new user for token-authenticated request: 16381b3386174456af66f876af1575f8
[I 19:10:34.121 ServerApp] User 16381b3386174456af66f876af1575f8 logged in.
[I 19:10:34.124 ServerApp] 302 POST /login?next=%2Flab (16381b3386174456af66f876af1575f8@192.169.3.130) 570.49ms
```

最后在树莓派浏览器中输入 http://127.0.0.1:8888 就可以正常运行了，如图 9-2 所示。

Jupyter lab 支持远程访问，如果树莓派 ip 地址为 192.169.3.159，则输入 http://192.169.3.159:8888/lab。

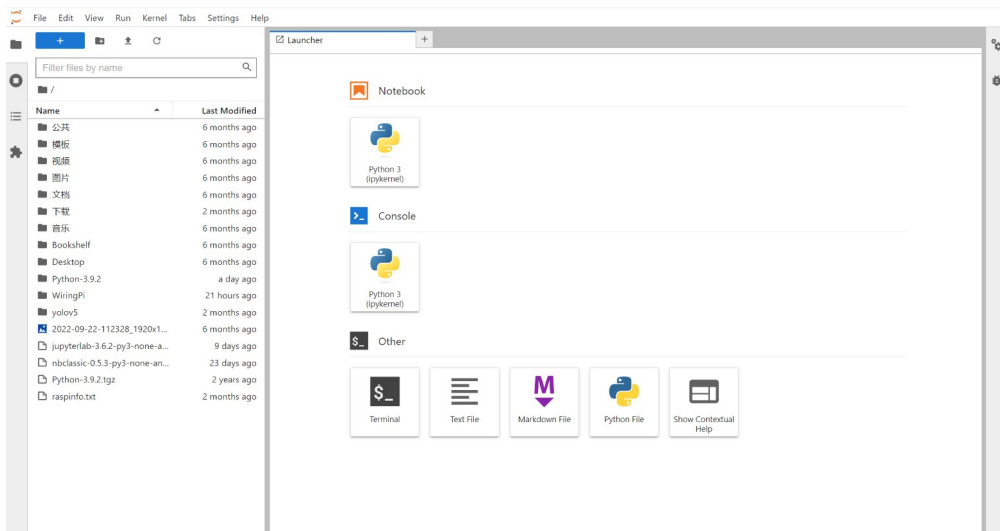


图 9-2 Jupyter lab 界面

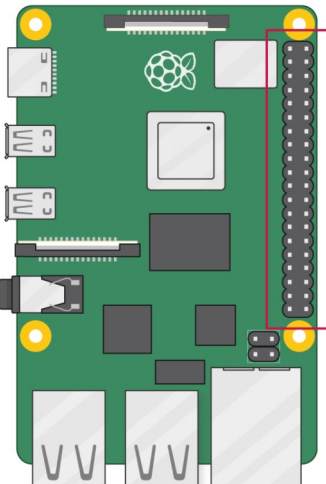
9.1.3 树莓派通用输入/输出接口（GPIO）

树莓派通用输入/输出接口（GPIO）是一组数字引脚，可用于将树莓派连接到其他电子设备。GPIO 引脚可以配置为输入或输出，因此可以用于读取传感器数据，控制 LED 等外部设备。

GPIO 引脚可以通过软件编程进行控制，例如使用 Python 或其他编程语言编写程序。树莓派还提供了各种库和工具，使编程更加方便。

使用 GPIO 需要小心谨慎，因为错误的连接和编程可能会导致设备损坏或故障。建议在使用 GPIO 之前仔细阅读相关文档，并确保采取适当的安全措施。

树莓派的 GPIO 引脚位于其引脚排针上，可以通过跳线线连接到其他电路板或设备。树莓派目前有 40 个 GPIO 引脚，其中 26 个引脚可以用作数字输入或输出，另外 14 个引脚用于其他功能，如图 9-3。其中包括有电源接口（5V、3.3V、GND）、有复用功能的 GPIO 口包含 I2C 接口（SCL、SDA），SPI 接口（MISO、MOSI、CLK、CS 片选信号 SPICE0_N），UART 串口接口（TXD、RXD），PWM 接口以及普通 GPIO 口。



wiringPi 编码	BCM 编码	功能名	物理引脚 BOARD 编码		功能名	BCM 编码	wiringPi 编码
		3.3V	1	2	5V		
8	2	SDA.1	3	4	5V		
9	3	SCL.1	5	6	GND		
7	4	GPIO.7	7	8	TXD	14	15
		GND	9	10	RXD	15	16
0	17	GPIO.0	11	12	GPIO.1	18	1
2	27	GPIO.2	13	14	GND		
3	22	GPIO.3	15	16	GPIO.4	23	4
		3.3V	17	18	GPIO.5	24	5
12	10	MOSI	19	20	GND		
13	9	MISO	21	22	GPIO.6	25	6
14	11	SCLK	23	24	CE0	8	10
		GND	25	26	CE1	7	11
30	0	SDA.0	27	28	SCL.0	1	31
21	5	GPIO.21	29	30	GND		
22	6	GPIO.22	31	32	GPIO.26	12	26
23	13	GPIO.23	33	34	GND		
24	19	GPIO.24	35	36	GPIO.27	16	27
25	26	GPIO.25	37	38	GPIO.28	20	28
		GND	39	40	GPIO.29	21	29

图 9-3 Raspberry pi 40 引脚对照表

树莓派接口的三种命名方案：Wiring Pi 编号、BCM 编号、物理引脚 Broad 编号。Wiring Pi 编号是功能接线的引脚号（如 TXD、PWM0 等等）；BCM 编号是 Broadcom 针脚号，也即是通常称的 GPIO；物理编号是 PCB 板上针脚的物理位置对应的编号（1~40）。

Wiring Pi 是应用于树莓派的 GPIO 控制库函数，是由 Gordon Henderson 所编写维护的。它刚开始是作为 BCM2835 芯片的 GPIO 库，现在已经除了 GPIO 库，还包括了 I2C 库、SPI 库、UART 库和软件 PWM 库等。Wiring Pi 使用 C、C++ 开发并且可以被其他语言包使用，例如 python、ruby 或者 PHP 等。Wiring Pi 库包含了一个命令行工具 gpio，它可以用来设置 GPIO 管脚，可以用来读写 GPIO 管脚，甚至可以在 Shell 脚本中使用来达到控制 GPIO 管脚的目的。

可以通过下载源代码来安装 Wiring Pi，使用 GIT 工具下载代码，然后编译安装。

```
# git clone https://github.com/WiringPi/WiringPi.git
# cd WiringPi/
# ./build
#
```

build 脚本将编译并安装所有内容。或者在官网下载安装包后安装 Wiring Pi。请注意 Pi 4B 与 Pi v3+ 安装包是不同的。使用 gpio readall 命令可以查看树莓派的 GPIO 引脚信息，如图 9-4。

```
# gpio readall
```

Pi 3B												
BCM	wPi	Name	Mode	V	Physical	V	Mode	Name	wPi	BCM		
		3.3v			1	2		5v				
2	8	SDA.1	IN	1	3	4		5v				
3	9	SCL.1	IN	1	5	6		0v				
4	7	GPIO. 7	IN	1	7	8	0	IN	TxD	15	14	
		0v			9	10	1	IN	RxD	16	15	
17	0	GPIO. 0	IN	0	11	12	0	IN	GPIO. 1	1	18	
27	2	GPIO. 2	IN	0	13	14		0v				
22	3	GPIO. 3	IN	0	15	16	0	IN	GPIO. 4	4	23	
		3.3v			17	18	0	IN	GPIO. 5	5	24	
10	12	MOSI	IN	0	19	20		0v				
9	13	MISO	IN	0	21	22	0	IN	GPIO. 6	6	25	
11	14	SCLK	IN	0	23	24	1	IN	CE0	10	8	
		0v			25	26	1	IN	CE1	11	7	
0	30	SDA.0	IN	1	27	28	1	IN	SCL.0	31	1	
5	21	GPIO.21	IN	1	29	30		0v				
6	22	GPIO.22	IN	1	31	32	0	IN	GPIO.26	26	12	
13	23	GPIO.23	IN	0	33	34		0v				
19	24	GPIO.24	IN	0	35	36	0	IN	GPIO.27	27	16	
26	25	GPIO.25	IN	0	37	38	0	IN	GPIO.28	28	20	
		0v			39	40	0	IN	GPIO.29	29	21	
BCM	wPi	Name	Mode	V	Physical	V	Mode	Name	wPi	BCM		
Pi 3B												

图 9-4 树莓派的 GPIO 引脚信息

Wiring Pi 编号模式只使用在 C 语言中，在 Python 程序中使用 BCM 编号、物理引脚 Broad 编号。下面将以 BCM 编号模式演示在 Python 程序中控制硬件。

项目 9.2 Python 控制树莓派 GPIO 引脚

下面用树莓派点亮 LED 灯，所需硬件包括 1 个面包板，2 根杜邦线公对母，1 个 LED 灯，1 个 330 欧姆电阻。用树莓派点亮一个 LED 灯虽然很简单，但却很重要，这是利用 GPIO 控制外部硬件设备的基础，能控制一个 LED 灯，就能让机器人动起来。

电路原理图如图 9-5 所示，实物图如图 9-6 所示。一个 LED 灯通过一个限流电阻串连到树莓派的 GPIO21 上，负极则连接到树莓派的 GND 上，从而形成一个完整的回路。

GPIO 引脚的输出电压约为 3.3V，如果直接串联，会有一个非常大的电流通过 LED，这个电流通常大到可以损坏 LED，甚至供电设备。因此，需要在 LED 和电源（GPIO 引脚）间串联一个电阻限制电流，从而对 LED 和为其供电的 GPIO 引脚提供保护。

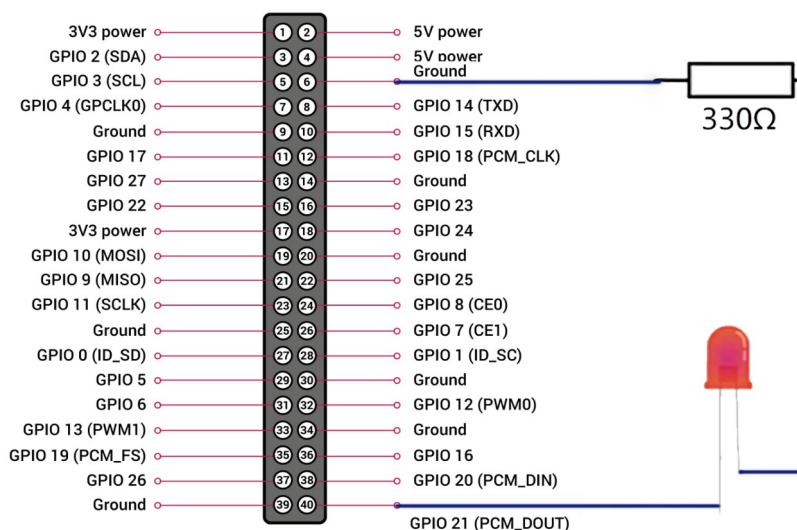


图 9-5 树莓派点亮 LED 电路图

可以在终端中控制 GPIO 口，测试线路是否连接正确。Linux 一切都是文件，对文件的读写操作就相当于向外设输出或者从外设输入数据。树莓派的 GPIO 端口文件在 `/sys/class/gpio` 目录下。

首先激活 GPIO21，然后把 GPIO21 置于输出状态，最后向 GPIO21 写入 1，让 PIN 处于高电压，这时就会看见 LED 灯被点亮了。

```
# echo 21 > /sys/class/gpio/export
# echo out > /sys/class/gpio/gpio1/direction
# echo 1 > /sys/class/gpio/gpio1/value
# echo 0 > /sys/class/gpio/gpio1/value
```

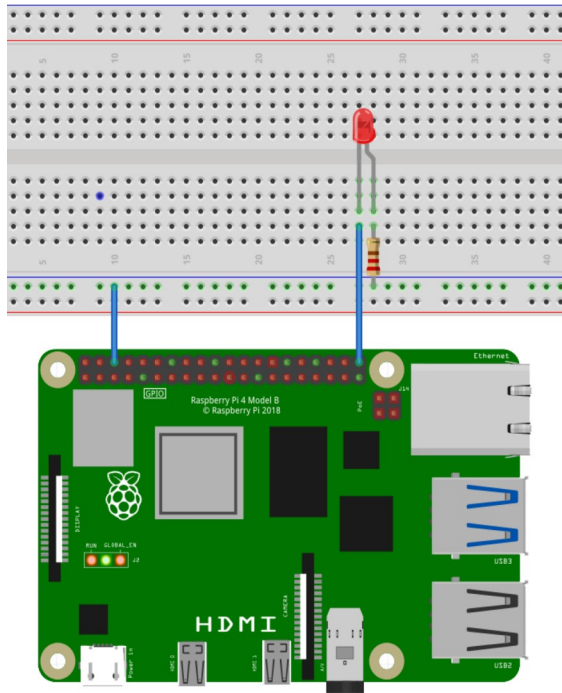


图 9-6 树莓派点亮 LED 连线图

在 Python 程序中定义的 GPIO 引脚有两种模式：BCM 编号模式、物理引脚 Broad 编号模式。使用 `GPIO.setmod()` 方法指定引脚编号系统，引脚编号系统一旦指定就不能改变。使用 `GPIO.setup()` 方法设置系统引脚是作为输入还是输出。

9.2.1 点亮单个 LED 灯，并让其亮暗闪烁

在这个例子中，首先导入 GPIO 库，然后设置 GPIO 引脚编号模式为 BCM 编号方式。使用 GPIO21 作为 LED 的控制引脚，使用 `GPIO.setup()` 方法将 GPIO21 设置为输出模式，并将其输出状态设置为 HIGH 电平，以点亮 LED 灯。

```
import RPi.GPIO as GPIO
import time

GPIO.setmode(GPIO.BCM)
GPIO.setup(21, GPIO.OUT)
GPIO.output(21, GPIO.LOW)
for i in range(10):
    GPIO.output(21, GPIO.HIGH)
    time.sleep(0.3)
    GPIO.output(21, GPIO.LOW)
    time.sleep(0.3)
GPIO.cleanup()
```

使用 `time.sleep()` 方法延迟一秒钟，然后再将 GPIO21 的输出状态设置为 LOW 电平，以关闭 LED 灯。最后使用 `GPIO.cleanup()` 方法清理 GPIO 引脚的设置，以便它们可以被其他程序使用。

8.2.2 按键控制 LED 灯的亮暗

通过按键来控制 LED 灯的亮暗，所采用的按键为四引脚按键。当按下按键时，LED 会亮，当再次按下 LED 灯时，LED 会熄灭。

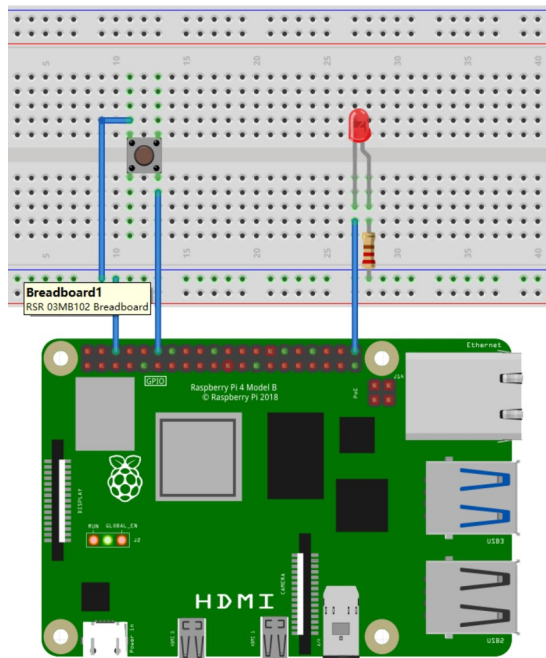


图 9-7 按键控制 LED 灯联线图

按键的一个引脚与树莓派 4B 的 18 号引脚连接，对角引脚接地。LED 灯的正极与树莓派的 21 号引脚连接。四脚按键的工作原理与普通按钮开关的工作原理类似，由常开触点、常闭触点组合而成。四脚按键开关中常开触点的作用是当压力向常开触点施压时，这个电路就呈现接通状态，当撤销这种压力的时候，就恢复到了原始的常闭触点。

四脚按键开关需要设置上拉电阻，在 18 号引脚处设置上拉电阻使用代码 `GPIO.setup(18, GPIO.IN, pull_up_down=GPIO.PUD_UP)`。

```
import RPi.GPIO as GPIO
import time

GPIO.setmode(GPIO.BCM)           #引脚是 BCM 编号方式
GPIO.setup(18, GPIO.IN, pull_up_down=GPIO.PUD_UP)  #在 18 引脚处设置上拉电阻
GPIO.setup(21, GPIO.OUT)
GPIO.output(21, GPIO.LOW)        #初始化，21 号引脚设置为低电平
ledStatus = 0                    #led 灯的初始状态默认为暗

try:
    while True:
        if (GPIO.input(18) == 0):
            time.sleep(0.2)
            print("button pressed!")
            ledStatus = not ledStatus
            if ledStatus:
                GPIO.output(21, GPIO.HIGH)
            else:
                GPIO.output(21, GPIO.LOW)
```

```
except KeyboardInterrupt:
```

```
    GPIO.cleanup()    #程序中止时清理 GPIO 资源
```

time.sleep(0.2)作用是开关去抖，忽略由于开关抖动引起的小于 0.2s 的边缘操作。
print("button pressed!") 用于观测开关去抖效果，如果忽略开关抖动的话，理论上每按下一次开关会输出一次。

可以使用 RPi.GPIO 库中的 wait_for_edge()函数和 add_event_detect()函数进行边缘检测。边缘的定义为电信号从低电平到高电平，或从高电平到低电平状态的变化。在需要场景中更关系输入状态的变化。

wait_for_edge() 函数是阻塞函数，会阻塞程序执行，直到检测到一个边沿。
add_event_detect()函数是增加一个事件的检测函数。

```
import RPi.GPIO as GPIO
```

```
import time
```

```
GPIO.setmode(GPIO.BCM)                #引脚是 BCM 编号方式
```

```
GPIO.setup(18, GPIO.IN, pull_up_down=GPIO.PUD_UP)    #在 18 引脚处设置上拉电阻
```

```
GPIO.setup(21, GPIO.OUT)
```

```
GPIO.output(21, GPIO.LOW)    #初始化，21 号引脚设置为低电平
```

```
channel = GPIO.wait_for_edge(18,GPIO.FALLING,timeout =5000)
```

```
if channel is None:
```

```
    print("Timeout occurred")
```

```
else:
```

```
    GPIO.output(21, GPIO.HIGH)
```

```
    print("Edge detected on channel",channel)
```

```
    time.sleep(3)
```

```
GPIO.cleanup()
```

项目 9.3 配置 NVIDIA Jetson Nano 开发环境

NVIDIA Jetson Nano 开发板是一款功能强大的边缘计算设备，能够在图像分类、物体检测、分割和语音处理等应用程序中并行运行多个神经网络。所有这些基于一个简单易用的平台，且运行功率仅为 5 瓦。该开发板是做边学的理想工具，使用 Linux 开发环境、不仅有大量易于学习的教程，还有大量的由活跃开发者社区打造的开源项目。其他类似的开发板还有 Raspberry Pi 4，Intel Neural Compute Stick 2 和 Google Edge TPU Coral Dev Board 等。

Jetson NanoCPU 使用 64 位四核的 ARM Cortex-A57，树莓派 4 已经升级为 ARM Cortex-A72。4GB 的内存并不能完全使用，其中有一部分（1GB 左右）是和显存共享的。Jetson Nano 的最大优势还是在体积上，它采用核心板可拆的设计，核心板的大小只有 70 x 45 mm，可以很方便的集成在各种嵌入式应用中。

Jetson Nano 最大的特色就是包含了一块 128 核 NVIDIA Maxwell 架构的 GPU，虽然已经是几代前的架构，不过因为用于嵌入式设备，从功耗、体积、价格上也算一个平衡。Nano 的计算能力不高，勉强可以使用一些小规模、并且优化过的网络进行推理，训练的话还是不够用的。同时它的功耗也非常低，有两种模式：

- 5W（低功耗模式；可以使用 USB 口供电）

- 10W（必须使用 Power Jack 外接 5V 电源供电）

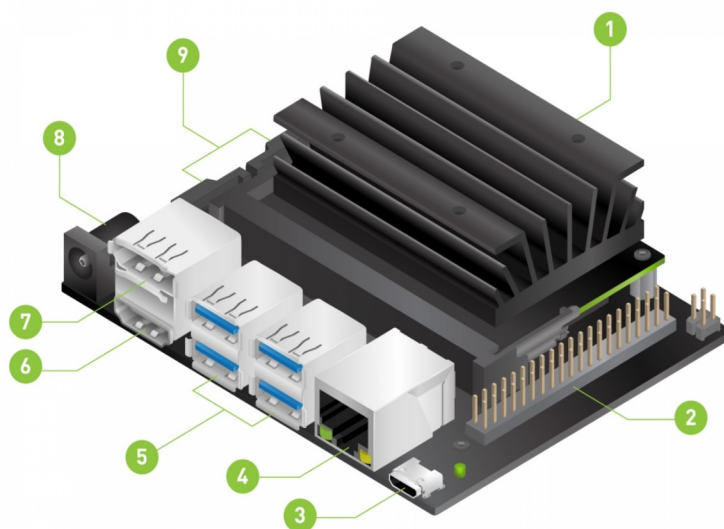


图 9-8 Jetson Nano 公版的实物图

Jetson Nano 公版的实物图如图 9-8 所示。其中 1 是用于主存储器的 microSD 卡插槽，可以进行系统镜像烧写。2 是 40 针 GPIO 扩展接口。3 是用来传输数据或使用电源供电的 Micro USB 接口。4 是千兆以太网端口。5 是 USB3.0 接口。6 是 HDMI 接口。7 是用来连接 DP 屏幕的 Display Port 接口。8 是用于 5V 电源输入的直流桶式插座。9 是 MIPI CSI-2 摄像头接口。端口和 GPIO 接头开箱即用，具有各种流行的外围设备，传感器和即用型项目，例如 NVIDIA 在 GitHub 上开源的 3D 可打印深度学习 JetBot。

Jetson Nano 可以运行各种深度学习模型，包括流行的深度学习框架的完整原生版本，如 TensorFlow, PyTorch, Caffe / Caffe2, Keras, MXNet 等。通过实现图像识别，对象检测和定位，姿势估计，语义分割，视频增强和智能分析等强大功能，这些模型可用于构建小型移动机器人、人脸签到打卡、口罩识别、智能门锁、智能音箱等复杂 AI 系统。

8.3.1 安装准备

Jetson Nano 开发板将 microSD 卡用作启动设备和主存储器。务必为项目准备一个大容量快速存储卡，后期还要安装 TensorFlow 等一些深度学习框架，还有可能要安装样本数据，建议最小采用 64 GB UHS-1 卡。需要使用能够在开发者套件的 Micro-USB 接口处提供 5V=2A 的高品质电源为开发者套件供电。并非每个宣称提供“5V=2A”的电源都能够真正做到这一点，电源不稳定可能造成系统不稳定。

Jetson 机身只有 Ethernet 有线网络，不包括无线网卡，使用的时候有时候不是很方便。官方推荐使用的 AC8265 这款 2.4G/5G 双模网卡，同时支持蓝牙 4.2。Jetson 包含 CSI 相机接口，B02 版本有两路，可以使用树莓派摄像头，IMX219 模组 800 万像素。

8.3.2 将镜像写入 microSD 卡

首先到英伟达官方下载官方镜像，也可以去开源社区下载配置好的镜像。把 microSD 卡插到读卡器上之后插到电脑，使用 SD Memory Card Formatter 格式化 microSD 卡，如图 9-9。

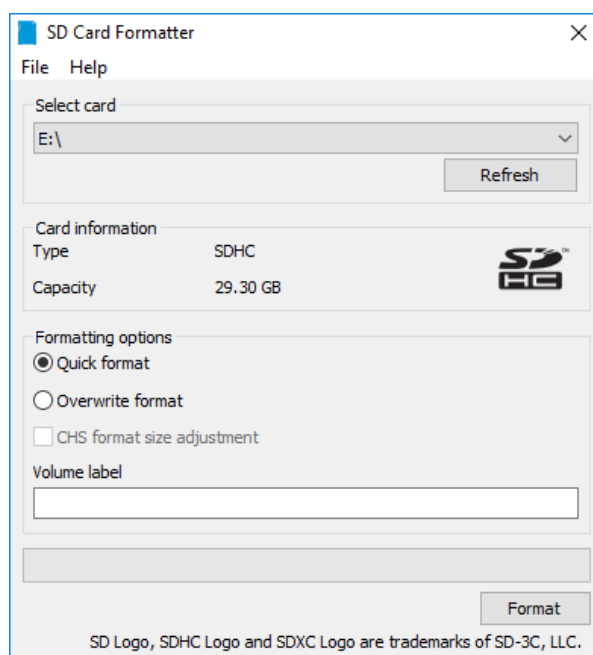


图 9-9 使用 SD Memory Card Formatter 格式化 microSD
然后使用 Etcher 将镜像写入 microSD 卡，如图 9-10。

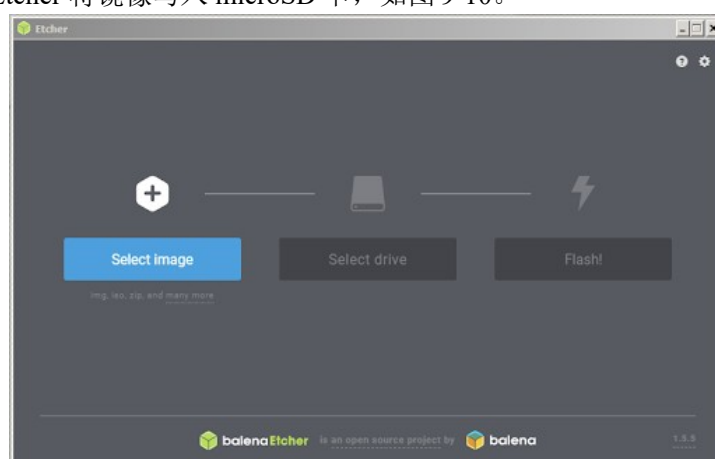


图 9-10 使用 Etcher 将镜像写入 microSD 卡

- 1) 下载、安装并启动 Etcher。
- 2) 单击“Select image”（选择镜像），然后选择先前下载的压缩镜像文件。
- 3) 插入 microSD 卡。
- 4) 如果 Windows 弹出如下提示对话框，则单击“Cancel”（取消）。
- 5) 单击“Select drive”（选择驱动器），并选择正确设备。
- 6) 单击“Flash!”（闪存！）。如果 microSD 卡通过 USB3 连接，Etcher 写入和验证镜像需要 10 分钟。

将已写入系统映像的 microSD 卡插入 Jetson Nano 模块底部的插槽中，如图 9-11。有两种方式可以与 Jetson Nano 开发板进行交互，一个是连接显示器、键盘和鼠标，二是通过 SSH 或 VNC 服务从另一台计算机远程访问。第一次请连接显示器，键盘和鼠标，然后连接的 Micro-USB 电源，开发板将自动开机并启动。

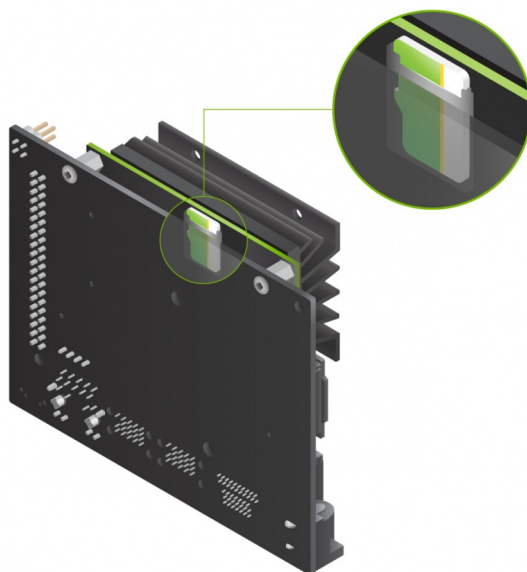


图 9-11 将 microSD 卡插入 Jetson Nano 模块底部的插槽

可以到 NVIDIA Jetson 开发者专区（Jetson Developer Kits），获取更多的 Jetson 平台信息，在 NVIDIA Jetson 论坛上提问或分享项目，可以在 Jetson 项目社区（Jetson Community Projects）获取一些非常有意思的项目，例如：

Hello AI World，该项目可以让你快速的启动并运行一组深度学习推理演示，体验 Jetson 的强大功能。演示使用计算机视觉相关的模型，包括实时摄像机的使用，使用带有 JetPack SDK 和 NVIDIA TensorRT 的 Jetson 开发工具包上的预训练模型进行实时图像分类和对象检测。还可以使用 C++ 编写自己的易于理解的识别程序。

JetBot 是面向有兴趣学习 AI 和构建有趣应用程序的创客、学生和爱好者。它易于设置和使用，并且与许多流行的配件兼容。通过几个交互式教程展示如何利用 AI 的力量来教 JetBot 跟随物体、避免碰撞等。

8.3.3 设置 VNC 服务器

Jetson Nano 开发板默认已经开启了 SSH 服务。如果觉得连接屏幕使用不方便的话，可以使用 VNC 实现 headless 远程桌面访问 Jetson Nano。VNC 可以从同一网络上的另一台计算机控制 Jetson Nano 开发板。需要通过一下设置开启 VNC 服务（配置方法可能会随着开发板镜像的不同而不同，请查阅官方文档）。

- 1) 安装 vino，可以用 `dpkg -l |grep vino` 查看是否已经安装。

```
# sudo apt update
# sudo apt install vino
```

- 2) 配置 VNC 服务

```
# gsettings set org.gnome.Vino prompt-enabled false
# gsettings set org.gnome.Vino require-encryption false
```

- 3) 设置 VNC 密码

```
# # Replace thepassword with your desired password
# gsettings set org.gnome.Vino authentication-methods "['vnc']"
# gsettings set org.gnome.Vino vnc-password $(echo -n 'thepassword'|base64)
thepassword 修改为你的密码。
```

- 4) VNC 服务器只有在本地登录到 Jetson 之后才可用。如果希望 VNC 自动可用，请使用系统设置应用程序来启用自动登录。

```
# mkdir -p ~/.config/autostart
# cp /usr/share/applications/vino-server.desktop ~/.config/autostart/.
```

- 5) 使用 VNC Viewer 软件进行 VNC 连接，首先需要查询 ip 地址，例如 192.169.1.195，输入 IP 地址后点击 OK，双击对应的 VNC 用户输入密码，最后进入到 VNC 界面。

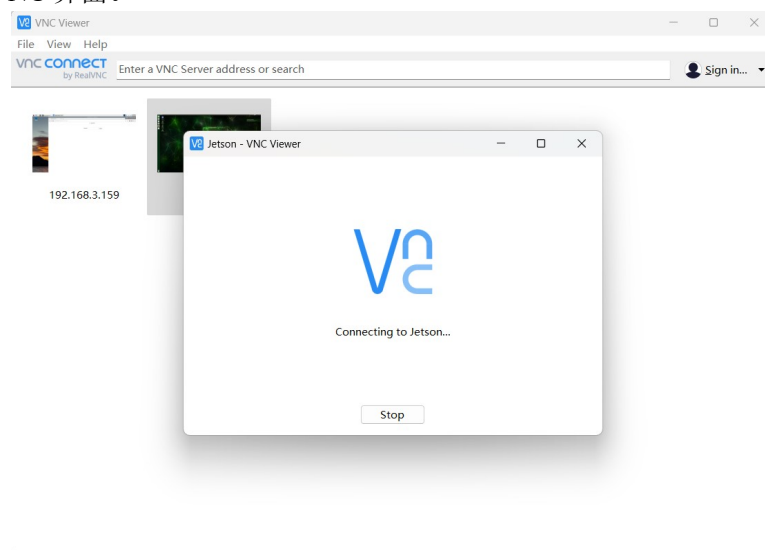


图 9-12 VNC Viewer 登录

8.3.4 Jetson Nano 安装 TensorFlow GPU

TensorFlow 是一个使用数据流图进行数值计算的开源软件库，这种灵活的架构可以将模型部署到桌面、服务器或移动设备中的 CPU 或 GPU 上。下面将在 Jetson Nano 安装 TensorFlow GPU 版本，安装 TensorFlow GPU 版本需要成功配置好 CUDA。不过在安装 TensorFlow GPU 之前，需要安装依赖项。

- 1) 安装 TensorFlow 所需的系统包

```
# sudo apt-get update
# sudo apt-get install libhdf5-serial-dev hdf5-tools libhdf5-dev zlib1g-dev
zip libjpeg9-dev liblapack-dev libblas-dev gfortran
```

- 2) 安装和升级 pip3

```
# sudo apt-get install python3-pip
# sudo python3 -m pip install --upgrade pip
# sudo pip3 install -U testresources setuptools==65.5.0
```

- 3) 安装 Python 包依赖项

```
# sudo pip3 install -U numpy==1.22 future==0.19.2 mock==3.0.5
keras_preprocessing==1.1.2 keras_applications==1.0.8 gast==0.4.0 protobuf
pybind11 cython pkgconfig packaging h5py==3.6.0
```

- 4) 安装 Python 包依赖项

```
# sudo pip3 install -U numpy==1.22 future==0.19.2 mock==3.0.5
keras_preprocessing==1.1.2 keras_applications==1.0.8 gast==0.4.0 protobuf
pybind11 cython pkgconfig packaging h5py==3.6.0
```

- 5) 确认 CUDA 已经被正常安装

```
# nvcc -V
nvcc: NVIDIA (R) Cuda compiler driver
Copyright (c) 2005-2019 NVIDIA Corporation
Built on Wed_Oct_23_21:14:42_PDT_2019
Cuda compilation tools, release 10.2, V10.2.89
```

- 6) 安装 TensorFlow

```
# sudo pip3 install --extra-index-url
https://developer.download.nvidia.com/compute/redist/jp/v51
tensorflow==2.11.0+nv23.01
```


安装的 TensorFlow 版本必须与正在使用的 JetPack 版本一致，版本信息请查看官网文档。

7) 验证安装

```
# python3
>>> import tensorflow
>>> import tensorflow as tf
2023-04-20 10:01:13.231340: I
tensorflow/stream_executor/platform/default/dso_loader.cc:49]
Successfully opened dynamic library libcudart.so.10.2
>>> print(tf.__version__)
2.11.0
```

8.3.5 Jetson Nano 安装 OpenCV

Jetson Nano 开发板上默认安装 JetPack 安装了对应的 OpenCV 不支持 CUDA 且版本是固定搭配的，可以使用 jtop 命令查看开发板系统信息，如图 9-13。目前系统 CUDA 版本是 10.2.89，OpenCV 版本是 4.1.1，不支持 CUDA。下面介绍如何在 Jetson Nano 开发板上手动编译与安装 OpenCV。

```
NVIDIA Jetson Nano (Developer Kit Version) - Jetpack 4.4 [L4T 32.4.3]

- Up Time:      0 days 2:54:19                      Version: 3.0.1
- Jetpack:      4.4 [L4T 32.4.3]                    Author: Raffaello Bonghi
- Board:                                     e-mail: raffaello@rnext.it
  * Type:      Nano (Developer Kit Version)
  * SOC Family: tegra210      ID: 33
  * Module:     P3448-0000    Board: P3449-0000
  * Code Name:  porg
  * Cuda ARCH:  5.3
  * Serial Number: 1423721002295
  * Board ids:  3448
- Libraries:
  * CUDA:      10.2.89
  * OpenCV:     4.1.1 compiled CUDA: NO
  * TensorRT:   7.1.3.0
  * VPI:        0.3.7
  * VisionWorks: 1.6.0.501
  * Vulkan:     1.2.70
  * cuDNN:      8.0.0.180
- Hostname:     nano-desktop
- Interfaces:
  * eth0:       192.168.1.103
```

图 9-13 Jetson Nano 开发板系统信息

在 Jetson Nano 开发板上安装 OpenCV 并不复杂。整个安装需要两个小时才能完成，可以创建了一个安装脚本。它以依赖项的安装开始，以 ldconfig 结束。

1) 安装依赖项

```
# echo "Installing OpenCV 4.7.0 on your Jetson Nano"
# echo "It will take 2.5 hours !"
#
# 查看 CUDA 安装路径
# cd ~
# sudo sh -c "echo '/usr/local/cuda/lib64' >> /etc/ld.so.conf.d/nvidia-
# tegra.conf"
# sudo ldconfig
#
# 安装依赖
# sudo apt-get install -y build-essential cmake git unzip pkg-config
# zlib1g-dev
# sudo apt-get install -y libjpeg-dev libjpeg9-dev libjpeg-turbo9-dev
# libpng-dev libtiff-dev
# sudo apt-get install -y libavcodec-dev libavformat-dev libswscale-dev
# libglew-dev
# sudo apt-get install -y libgtk2.0-dev libgtk-3-dev libcanberra-gtk*
# sudo apt-get install -y python-dev python-numpy python-pip
# sudo apt-get install -y python3-dev python3-numpy python3-pip
```

```
# sudo apt-get install -y libxvidcore-dev libx264-dev libgtk-3-dev
# sudo apt-get install -y libtbb2 libtbb-dev libdc1394-22-dev libxine2-dev
# sudo apt-get install -y gstreamer1.0-tools libv4l-dev v4l-utils qv4l2
# sudo apt-get install -y libgstreamer-plugins-base1.0-dev libgstreamer-
# plugins-good1.0-dev
# sudo apt-get install -y libavresample-dev libvorbis-dev libxine2-dev
# libtesseract-dev
# sudo apt-get install -y libfaac-dev libmp3lame-dev libtheora-dev
# libpostproc-dev
# sudo apt-get install -y libopencore-amrnb-dev libopencore-amrwb-dev
# sudo apt-get install -y libopenblas-dev libatlas-base-dev libblas-dev
# sudo apt-get install -y liblapack-dev liblapacke-dev libeigen3-dev
# gfortran
# sudo apt-get install -y libhdf5-dev protobuf-compiler
# sudo apt-get install -y libprotobuf-dev libgoogle-glog-dev libgflags-dev
```

2) 下载 OpenCV

```
# 删除旧版本 opencv
# cd ~
# sudo rm -rf opencv*
# 下载 opencv4.7.0
# wget -O opencv.zip https://github.com/opencv/opencv/archive/4.7.0.zip
# wget -O opencv_contrib.zip
# https://github.com/opencv/opencv_contrib/archive/4.7.0.zip
# unzip opencv.zip
# unzip opencv_contrib.zip
# 解压好后修改文件夹名
# mv opencv-4.7.0 opencv
# mv opencv_contrib-4.7.0 opencv_contrib
```

3) 编译 OpenCV

```
# cd ~/opencv
# mkdir build
# cd build

# run cmake
# cmake -D CMAKE_BUILD_TYPE=RELEASE \
-D CMAKE_INSTALL_PREFIX=/usr \
-D OPENCV_EXTRA_MODULES_PATH=~/opencv_contrib/modules \
-D EIGEN_INCLUDE_PATH=/usr/include/eigen3 \
-D WITH_OPENCL=OFF \
-D WITH_CUDA=ON \
-D CUDA_ARCH_BIN=5.3 \
-D CUDA_ARCH_PTX="" \
-D WITH_CUDNN=ON \
-D WITH_CUBLAS=ON \
-D ENABLE_FAST_MATH=ON \
-D CUDA_FAST_MATH=ON \
-D OPENCV_DNN_CUDA=ON \
-D ENABLE_NEON=ON \
-D WITH_QT=OFF \
-D WITH_OPENMP=ON \
-D BUILD_TIFF=ON \
-D WITH_FFMPEG=ON \
-D WITH_GSTREAMER=ON \
-D WITH_TBB=ON \
-D BUILD_TBB=ON \
-D BUILD_TESTS=OFF \
-D WITH_EIGEN=ON \
-D WITH_V4L=ON \
-D WITH_LIBV4L=ON \
-D OPENCV_ENABLE_NONFREE=ON \
-D INSTALL_C_EXAMPLES=OFF \
-D INSTALL_PYTHON_EXAMPLES=OFF \
-D PYTHON3_PACKAGES_PATH=/usr/lib/python3/dist-packages \
```

```
-D OPENCV_GENERATE_PKGCONFIG=ON \  
-D BUILD_EXAMPLES=OFF ..
```

```
# make -j4
```

编译之前需要设置 OpenCV 的内容、位置和方式，涉及许多内容，详细信息请参考 OpenCV 官网文档。例如 `-D WITH_QT=OFF`，这禁用了 Qt5 支持。运行配置后需要检查输出结果，很可能会出现错误。

准备好所有编译指令后，可以开始编译，将需要大约两个半小时。

4) 安装 OpenCV

```
# sudo rm -r /usr/include/opencv4/opencv2  
# sudo make install  
# sudo ldconfig  
#  
# make clean  
# sudo apt-get update  
#  
# echo "Congratulations!"  
# echo "You've successfully installed OpenCV 4.7.0 on your Jetson Nano"
```

项目 9.4 基于人脸识别的门禁系统

正如 Jetson Nano 简介文章中提到的，开发套件有一个移动行业处理器接口（MIPI）的相机串行接口（CSI）端口，MIPI 是 MIPI 联盟发起的为移动应用处理器制定的开放标准。支持 Raspberry Pi、Arducam 等常见的相机模块。这些相机很小，但适用于机器学习和计算机视觉应用，如物体检测、人脸识别、图像分割等视觉任务。

本项目将介绍如何访问 Jetson Nano 开发板上 CSI 摄像头模块，然后将使用 OpneCV 读取摄像头视频流并检测人脸，最后学习使用人脸识别库（`face_recognition`）来识别人脸。本项目中人脸识别任务使用的是 Raspberry Camera V2 相机模块，800 万像素、感光芯片为索尼 IMX219，静态图片分辨率为 3280×2464 、支持 1080p30, 720p60 以及 640×480 p90 视频录像。



图 9-14 将摄像头连接到 Jetson Nano

将摄像头连接到 Jetson Nano，如图 9-14。拔下 CSI 端口，将相机带状电缆插入端口。确保端口上的连接导线与带状电缆上的连接线对齐。

英伟达提供的 JetPack SDK 已经支持预装驱动程序的 RPi 相机，并且可以很容易地用作即插即用外围设备，不需要安装驱动程序。CSI 摄像头不支持即插即用，所以必须在开机前先装上去，系统才能识别 CSI 摄像头，如果开机之后再安装，会导致 Jetson Nano 识别不出摄像头，且有其他风险，因此请避免在开机状态下安装摄像头。

查看/dev/video0 检查摄像头信息。

```
# ls -l /dev/video0
crw-rw----+ 1 root video 81, 0 4月 20 21:24 /dev/video0
```

Linux 中所有设备文件或特殊文件的存储位置是/dev。如果命令的输出为空，则表明开发板上没有连接摄像头，或者连接有错。Ls 命令返回信息太少，通常无法判断到底哪个编号是哪个摄像头。要更进一步检测摄像头数量与详细规格，可以使用 v4l2-utils 工具。

```
# v4l2-ctl --list-devices
vi-output, imx219 9-0010 (platform:54080000.vi:4):
/dev/video0
```

Jetson 系列开发板系统使用 GStreamer 管道处理媒体应用程序，GStreamer 是一个多媒体框架，用于后端处理任务，如格式修改、显示驱动程序协调和数据处理。

检查摄像头连接的更麻烦的方法是使用 GStreamer 应用程序 gst 启动来启动一个显示窗口，并确认您可以从摄像头看到实时流，如果图 9-15 显示的是早上五点的无锡。

```
# gst-launch-1.0 nvarguscamerasrc ! 'video/x-raw(memory:NVMM),width=3820,
height=2464, framerate=21/1, format=NV12' ! nvvidconv flip-method=0 !
'video/x-raw,width=480, height=320' ! nvvidconv ! nvegltransform !
nveglglessink -e
```



图 9-15 GStreamer 打开的实时流

GStreamer 打开 1920 像素宽，1080 高@ 30 帧/秒的相机流，并在 960 像素宽 640 像素高的窗口中显示它。当您需要更改相机的方向时（翻转图片），flip-method 参数可以进行设置。上面的命令创建了一个 GStreamer 管道，其中的元素由! 分隔。它启动了一个 3820 像素，高 2464 像素，30 帧/秒的相机流。有关配置的详细信息，可以在 gst 发布的文档中找到。

8.4.1 使用 Haar 特征的 cascade 分类器检测人脸

Haar 特征的 cascade 分类器(classifiers) 是一种有效的物品检测方法。它是通过许多正负样例中训练得到 cascade 方程，然后将其应用于其他图片。

Haar 特征分类器就是一个 XML 文件，该文件中会描述人体各个部位的 Haar 特征值。OpenCV 有很多已经训练好的分类器，其中包括面部，眼睛，微笑等。这些 XML 文件只在

在/openopencv4/data/haarcascades/文件夹中。下面将使用 OpenCV 创建一个面部检测器。首先加载需要的 XML 分类器，然后以灰度格式加载输入图像或是视频。

```
import cv2
import numpy as np
from matplotlib import pyplot as plt

HAAR_CASCADE_XML_FILE_FACE =
"/usr/share/openopencv4/haarcascades/haarcascade_frontalface_default.xml"
HAAR_CASCADE_XML_FILE_EYE =
"/usr/share/openopencv4/haarcascades/haarcascade_eye.xml"

image = cv2.imread('sachin.jpg')
face_cascade = cv2.CascadeClassifier(HAAR_CASCADE_XML_FILE_FACE)
grayscale_image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
detected_faces = face_cascade.detectMultiScale(grayscale_image, scaleFactor = 1.3,
minNeighbors = 5)

for (x_pos, y_pos, width, height) in detected_faces:
    cv2.rectangle(image, (x_pos, y_pos),
        (x_pos + width, y_pos + height), (0, 255, 0), 2)

plt.imshow(image)
plt.show()
```

face_cascade.detectMultiScale 用于在图像中检测面部，如果检测到面部会返回面部所在的矩形区域 Rect(x,y,w,h)。其中 scaleFactor 参数表示在前后两次相继的扫描中，搜索窗口的比例系数，默认为 1.1 即每次搜索窗口依次扩大 10%。minNeighbors 参数表示构成检测目标的相邻矩形的最小个数(默认为 3 个)。运行结果如图 9-16 所示。

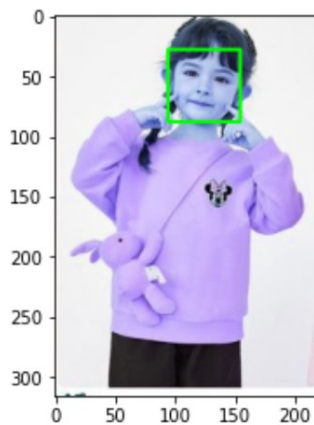


图 9-16 Haar 特征的 cascade 分类器检测结果

8.4.2 使用摄像头实时检测人脸

下面将介绍如何通过 OpenCV 调用 CSI 摄像头（IMX219）和 USB 摄像头。通常 video0 和 video1 是 CSI 摄像头，video2 是 USB 摄像头。调用 USB 摄像头非常简单，可以使用 cv2.VideoCapture(2) 直接打开 USB 摄像头。但是对与 CSI 摄像头，会提示错误：GStreamer: 管线没有被创建 摄像机没有打开。

```
[ WARN:0@0.215] global cap_gstreamer.cpp:2785 handleMessage OpenCV | GStreamer
warning: Embedded video playback halted; module v4l2src0 reported: Internal data stream error.
```

```
[ WARN:0@0.217] global cap_gstreamer.cpp:1679 open OpenCV | GStreamer warning: unable to start pipeline
[ WARN:0@0.217] global cap_gstreamer.cpp:1164 isPipelinePlaying OpenCV | GStreamer warning: GStreamer: pipeline have not been created
Cannot open Camera
```

CSI 摄像头需要使用 Gstreamer 读取视频流，步骤如下：创建 GStreamer 管道，将管道绑定 opencv 的视频流，逐帧提取和显示。

首先设置 GStreamer 管道参数，创建 GStreamer 管道。

```
GSTREAMER_PIPELINE = 'nvarguscamerasrc ! \
    video/x-raw(memory:NVMM), width=%d, height=%d, format=(string)NV12, \
    framerate=(fraction)%d/1 ! \
    nvvidconv flip-method=%d ! nvvidconv ! \
    video/x-raw, width=(int)%d, height=(int)%d, format=(string)BGRx ! \
    videoconvert ! appsink' % (3820, 2464, 21, 0, 640, 480)
```

其中(3820, 2464, 21, 0, 640, 480)分别表示：

摄像头预捕获的图像宽度，与高度分别是 3820, 2464。窗口显示的图像宽度与高度分别是 640, 480。捕获帧率（framerate）是 21 帧，是否旋转图像（flip_method）为否。

然后对视频流的每个图像帧进行处理并测试人脸检测。首先将彩色图像转换为灰度图像，因为颜色不决定面部特征，可以避免计算开销、提高性能。一旦确认图像中包含人脸会在边界周围绘制一个矩形。

```
HAAR_CASCADE_XML_FILE_FACE =
"/usr/share/opencv4/haarcascades/haarcascade_frontalface_default.xml"
def faceDetect():
    # Obtain face detection Haar cascade XML files from OpenCV
    face_cascade = cv2.CascadeClassifier(HAAR_CASCADE_XML_FILE_FACE)

    # Video Capturing class from OpenCV
    video_capture = cv2.VideoCapture(GSTREAMER_PIPELINE, cv2.CAP_GSTREAMER)
    if video_capture.isOpened():
        cv2.namedWindow("Face Detection Window", cv2.WINDOW_AUTOSIZE)

    while True:
        return_key, image = video_capture.read()
        if not return_key:
            break

        grayscale_image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
        detected_faces = face_cascade.detectMultiScale(grayscale_image, 1.3, 5)

        # Create rectangle around the face in the image canvas
        for (x_pos, y_pos, width, height) in detected_faces:
            cv2.rectangle(image, (x_pos, y_pos),
                          (x_pos + width, y_pos + height), (0, 0, 0), 2)

        cv2.imshow("Face Detection Window", image)

    key = cv2.waitKey(30) & 0xff
    # Stop the program on the ESC key
    if key == 27:
        break
```



```

video_capture.release()
cv2.destroyAllWindows()
else:
    print("Cannot open Camera")

if __name__ == "__main__":
    faceDetect()

```

程序运行结果如图 9-17。

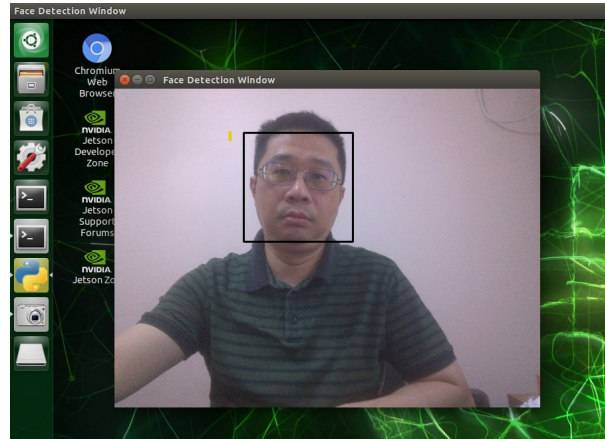


图 9-17 实时检测人脸

8.4.3 人脸识别

说到人脸识别的应用，已经越来越普及到人们的生活中，无论是餐厅商超，还是小区办公楼，都能看到不少地方使用人脸识别设备。例如公司预先将员工的照片录入系统，当员工访问系统时，可以由照相设备采集面孔，使用人脸识别技术，找到员工对应的身份信息，实现刷脸登录的功能，所有的身份信息和照片都在系统内，不使用互联网服务，确保数据安全。

Face Recognition 是一个强大、简单、易上手的人脸识别开源项目，它主要封装了 dlib 这一 C++ 图形库，通过 Python 语言将它封装为一个非常简单就可以实现人脸识别的 API 库，屏蔽了人脸识别的算法细节，大大降低了人脸识别功能的开发难度。Face Recognition 库进行人脸识别主要经过人脸检测、检测面部特征点、给脸部编码、从编码中找出人的名字这四个步骤。在 Face Recognition 库中对应的 API 接口如下：

1. 人脸检测
`face_locations(images, number_of_times_to_upsample=1, batch_size=128)`
2. 检测面部特征点
`face_landmarks(face_image, face_locations=None, model='large')`
3. 给脸部编码
`face_encodings(face_image, known_face_locations=None, num_jitters=1)`
4. 从编码中找出人的名字
`compare_faces(known_face_encodings, face_encoding_to_check, tolerance=0.6)`

Face Recognition 中，检测面部特征点 `face_landmarks` 已经在第 3 步给脸部编码 `face_encodings` 函数的开头执行过了，所以如果要进行人脸识别可以跳过第 2 步，只需要 1、3、4 步。此外 Face Recognition 库还提供了如 `load_image_file(file, mode='RGB')` 这一加

载面孔照片的函数等其他函数。

只需使用 `pip install face_recognition` 安装 `face_recognition`，当然必须先安装 `cmake`、`dlib`、`opencv`。

1) 在图片中定位人脸的位置

准备一张照片，定位结果如图 9-18。`face_recognition` 提供了加载图像的函数 `load_image_file()`，`face_locations` 用于定位图像中的人脸位置。其中 `number_of_times_to_upsample` 参数设置对图像进行多少次上采样以查找人脸。`model` "hog"则结果不太准确，但在 CPU 上运行更快。"cnn"是更准确的深度学习模型，需要 GPU 加速。在测试过程中会发现部分图像识别检测人脸失败的问题，`face_recognition` 更像是一个基础框架，帮助我们更加高效地去构建自己的人脸识别的相关应用。



图 9-18 `face_recognition` 人脸定位结果

```
import face_recognition
import cv2

image = face_recognition.load_image_file("graduation1.png")
face_locations = face_recognition.face_locations(image, number_of_times_to_upsample=0,
model="cnn")

print("I found {} face(s) in this photograph.".format(len(face_locations)))
face_num = len(face_locations)
for i in range(face_num):
    top, right, bottom, left = face_locations[i]
    print("A face is located at pixel location Top: {}, Left: {}, Bottom: {}, Right: {}".format(top,
left, bottom, right))
    start = (left, top)
    end = (right, bottom)
    color = (0, 255, 0)
    thickness = 3
    cv2.rectangle(image, start, end, color, thickness)
    cv2.imwrite("grad.png", image)

#显示识别结果
cv2.namedWindow("recognition", cv2.WINDOW_NORMAL)
cv2.imshow("recognition", image)
cv2.waitKey(0)
```

```
cv2.destroyAllWindows()
```

2) 从摄像头获取视频进行人脸识别

下面我们将从摄像头获取视频来进行人脸识别，输入自己和同学的人脸图像和一张未知人脸图像，然后进行人脸识别并在未知人脸图像标注各个人脸身份信息。`face_encodings`函数返回图像中每张人脸的128维人脸编码，返回结果保存在`me_face_encoding`中。

`compare_faces`函数将人脸编码列表与候选编码进行比较，以查看它们是否匹配。返回结果可能有多张人脸匹配成功，简单的处理是只以匹配的第一张人脸为结果。或者可以使用`face_distance`函数计算已知人脸和未知人脸特征向量的距离，距离越小表示两张人脸为同一个人的可能性越大。`face_distance`函数会将给定人脸编码列表，与已知的人脸编码进行比较，并得到每个比较人脸的欧氏距离。程序运行结果如图9-19。



图 9-19 从摄像头获取视频进行人脸识别

```
import face_recognition
import cv2
import numpy as np

GSTREAMER_PIPELINE = 'nvarguscamerasrc ! \
    video/x-raw(memory:NVMM), width=%d, height=%d, format=(string)NV12, \
    framerate=(fraction)%d/1 ! \
    nvvidconv flip-method=%d ! nvvidconv ! \
    video/x-raw, width=(int)%d, height=(int)%d, format=(string)BGRx ! \
    videoconvert ! appsink' % (3820, 2464, 21, 0, 640, 480)

me_image = face_recognition.load_image_file("me.png")
me_face_encoding = face_recognition.face_encodings(me_image)[0]

# Load a second sample picture and learn how to recognize it.
two_image = face_recognition.load_image_file("graduation1.png")
wang_face_encoding = face_recognition.face_encodings(two_image)[0]
li_face_encoding = face_recognition.face_encodings(two_image)[1]

# Create arrays of known face encodings and their names
known_face_encodings = [
    me_face_encoding,
```

```

    wang_face_encoding,
    li_face_encoding
]
known_face_names = [
    "Ping",
    "Wang",
    "Li"
]

face_locations = []
face_encodings = []
face_names = []
process_this_frame = True

video_capture = cv2.VideoCapture(GSTREAMER_PIPELINE, cv2.CAP_GSTREAMER)

while True:
    # Grab a single frame of video
    ret, frame = video_capture.read()

    # Only process every other frame of video to save time
    if process_this_frame:
        # Resize frame of video to 1/4 size for faster face recognition processing
        small_frame = cv2.resize(frame, (0, 0), fx=0.25, fy=0.25)

        # Convert the image from BGR color (which OpenCV uses) to RGB color (which
        face_recognition uses)
        rgb_small_frame = small_frame[:, :, :-1]

        # Find all the faces and face encodings in the current frame of video
        face_locations = face_recognition.face_locations(rgb_small_frame)
        face_encodings = face_recognition.face_encodings(
            rgb_small_frame, face_locations)

        face_names = []
        for face_encoding in face_encodings:
            # See if the face is a match for the known face(s)
            matches = face_recognition.compare_faces(
                known_face_encodings, face_encoding)
            name = "Unknown"

            # Or instead, use the known face with the smallest distance to the new face
            face_distances = face_recognition.face_distance(
                known_face_encodings, face_encoding)
            best_match_index = np.argmin(face_distances)
            if matches[best_match_index]:
                name = known_face_names[best_match_index]

            face_names.append(name)

        process_this_frame = not process_this_frame

    # Display the results
    for (top, right, bottom, left), name in zip(face_locations, face_names):

```

```

# Scale back up face locations since the frame we detected in was scaled to 1/4 size
top *= 4
right *= 4
bottom *= 4
left *= 4

# Draw a box around the face
cv2.rectangle(frame, (left, top), (right, bottom), (0, 0, 255), 2)

# Draw a label with a name below the face
cv2.rectangle(frame, (left, bottom - 35),
              (right, bottom), (0, 0, 255), cv2.FILLED)
font = cv2.FONT_HERSHEY_DUPLEX
cv2.putText(frame, name, (left + 6, bottom - 6),
            font, 1.0, (255, 255, 255), 1)

# Display the resulting image
cv2.imshow('Video', frame)

# Hit 'q' on the keyboard to quit!
if cv2.waitKey(1) & 0xFF == ord('q'):
    break

# Release handle to the webcam
video_capture.release()
cv2.destroyAllWindows()

```

项目 9.5 花卉识别

图像识别是指利用计算机对图像进行处理、分析和理解，以识别各种不同模式的目标和对象的技术，是应用深度学习算法的一种实践应用。现阶段图像识别技术一般分为人脸识别与商品识别，人脸识别主要运用在安全检查、身份核验与移动支付中，商品识别主要运用在商品流通过程中，特别是无人货架、智能零售柜等无人零售领域。随着嵌入式硬件资源和深度学习算法的突破，图像识别算法可直接运行与终端设备上，及所谓的边缘计算边缘计算，是指在靠近物或数据源头的一侧，采用网络、计算、存储、应用核心能力为一体的开放平台，就近提供最近端服务。其应用程序在边缘侧发起，产生更快的网络服务响应，满足行业在实时业务、应用智能、安全与隐私保护等方面的基本需求。

8.5.1 TensorFlow Lite 整体架构

TensorFlow Lite 是一组工具，可帮助开发者在移动设备、嵌入式设备和 IoT 设备上运行 TensorFlow 模型。它支持设备端机器学习推断，延迟较低，并且二进制文件很小。

TensorFlow Lite 包括两个主要组件：

- TensorFlow Lite 解释器(Interpreter)
- TensorFlow Lite 转换器(Converter)
- 算子库(Op kernels)
- 硬件加速代理(Hardware accelerator delegate)

TFLite 采用更小的模型格式，并提供了方便的模型转换器，可将 TensorFlow 模型转换

为方便解释器使用的格式，并可引入优化以减小二进制文件的大小和提高性能。比如 SavedModel 或 GraphDef 格式的 TensorFlow 模型，转换成 TFLite 专用的模型文件格式，在此过程中会进行算子融合和模型优化，以压缩模型，提高性能。

TensorFlow Lite 采用更小的解释器，可在手机、嵌入式 Linux 设备和微控制器等很多不同类型的硬件上运行经过专门优化的模型。安卓应用只需 1 兆左右的运行环境，在 MCU 上甚至可以小于 100KB。

TFLite 算子库目前有 130 个左右，它与 TensorFlow 的核心算子库略有不同，并做了移动设备相关的优化。

在硬件加速层面，对于 CPU 利用了 ARM 的 NEON 指令集做了大量的优化。同时，Lite 还可以利用手机上的加速器，比如 GPU 或者 DSP 等。另外，最新的安卓系统提供了 Android 神经网络 API（Android NN API），让硬件厂商可以扩展支持这样的接口。

图 9-1 展示了在 TensorFlow 2.0 中 TFLite 模型转换过程，用户在自己的工作台中使用 TensorFlow API 构造 TensorFlow 模型，然后使用 TFLite 模型转换器转换成 TFLite 文件格式（FlatBuffers 格式）。在设备端，TFLite 解释器接受 TFLite 模型，调用不同的硬件加速器比如 GPU 进行执行。

使用 TensorFlow Lite 的工作流程包括如下步骤，如图 9-20。

1) 选择模型

可以使用自己的 TensorFlow 模型、在线查找模型，或者从 TensorFlow 预训练模型中选择一个模型直接使用或重新训练。

2) 转换模型

如果使用的是自定义模型，请使用 TensorFlow Lite 转换器将模型转换为 TensorFlow Lite 格式。

3) 部署到设备

使用 TensorFlow Lite 解释器（提供多种语言的 API）在设备端运行模型。

4) 优化模型

使用模型优化工具包缩减模型的大小并提高其效率，同时最大限度地降低对准确率的影响。



图 9-20 TensorFlow Lite 的工作流程

8.5.2 训练花卉识别模型

使用常见卷积神经网络构建花卉识别模型，模型分为卷积层与全连接层两个部分，卷积层由 3 个 Conv2D 和 2 个 MaxPooling2D 层组成，在模型的最后把卷积后的输出张量传给多个全连接层来完成分类。

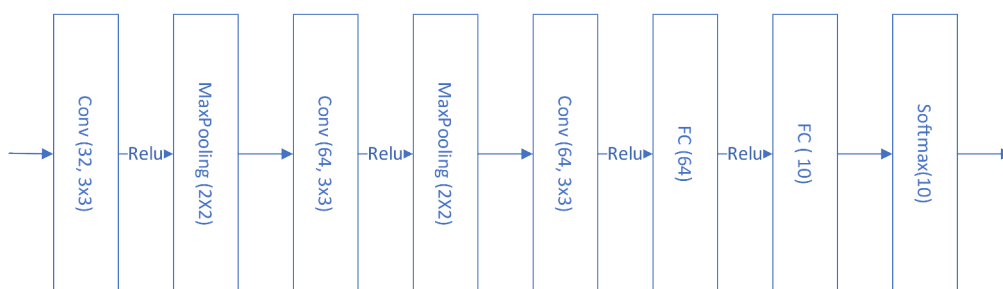


图 9-21 搭建卷积神经网络

卷积层的基本单位是卷积层后接上最大池化层，卷积层的作用是识别图像里的空间模式，例如线条和物体局部。最大池化层的作用是降低卷积层对位置的敏感。每个卷积层都使用 3×3 的卷积核，并在输出上使用 Relu 激活函数。第一个卷积层输出通道数为 32，第二、三卷积层的输出是 64。

卷积层输入是张量形状是 (image_height, image_width, color_channels)，包含了图像高度、宽度及颜色信息。花卉数据集中的图片形状是 (224,224, 3)，可以在声明第一层时将形状赋值给参数 input_shape。

下面将使用 TensorFlow Lite 实现花卉识别，在 Jetson Nano 设备上运行图像识别模型来识别花卉。本项目实施步骤如下：

1. 导入相关库

In[1]:

```
import tensorflow as tf
assert tf.__version__.startswith('2')
import os
import numpy as np
import matplotlib.pyplot as plt
```

2. 准备数据集

该数据集可以在 http://download.tensorflow.org/example_images/flower_photos.tgz 下载。每个子文件夹都存储了一种类别的花的图片，子文件夹的名称就是花的类别的名称。平均每一种花有 734 张图片，图片都是 RGB 色彩模式的。

In[2]:

```
data_root = tf.keras.utils.get_file(
    'flower_photos',
    'https://storage.googleapis.com/download.tensorflow.org/example_images/flower_photos.tgz',
    untar=True)
```

数据集解压后存放在.keras\datasets\flower_photos 目录下。

```
2016/02/11 04:52 <DIR>      daisy
2016/02/11 04:52 <DIR>      dandelion
2016/02/09 10:59      418,049 LICENSE.txt
2016/02/11 04:52 <DIR>      roses
2016/02/11 04:52 <DIR>      sunflowers
2016/02/11 04:52 <DIR>      tulips
```

将数据集划分为训练集和验证集。训练前需要手动加载图像数据，完成包括遍历数据集的目录结构、加载图像数据以及返回输入和输出。可以使用 Keras 提供的 ImageDataGenerator 类，它是 keras.preprocessing.image 模块中的图片生成器，负责生成一

个批次的图片，以生成器的形式给模型训练

`ImageDataGenerator` 的构造函数包含许多参数，用于指定加载后如何操作图像数据，包括像素缩放和数据增强。

接着需要一个迭代器来逐步加载单个数据集的图像。这需要调用 `flow_from_directory` () 函数并指定该数据集目录，如 `train`、`validation` 目录，函数还允许配置与加载图像相关的更多细节。`target_size` 参数允许将所有图像加载到一个模型需要的特定的大小，设置为大小为(224, 224)的正方形图像。

`batch_size` 默认的为 32，意思是训练时从数据集中的不同类中随机选出的 32 个图像，该值设置为 64。在评估模型时，可能还希望以确定性顺序返回批处理，这可以通过将 `shuffle` 参数设置为 `False`。

In[3]:

```
batch_size = 32
img_height = 224
img_width = 224

train_ds = tf.keras.utils.image_dataset_from_directory(
    str(data_root),
    validation_split=0.2,
    subset="training",
    seed=123,
    image_size=(img_height, img_width),
    batch_size=batch_size
)

val_ds = tf.keras.utils.image_dataset_from_directory(
    str(data_root),
    validation_split=0.2,
    subset="validation",
    seed=123,
    image_size=(img_height, img_width),
    batch_size=batch_size
)
```

Out[3]:

```
Found 3670 files belonging to 5 classes.
Using 2936 files for training.
Found 3670 files belonging to 5 classes.
Using 734 files for validation.
```

保存标签文件：

In[4]:

```
class_names = np.array(train_ds.class_names)
print(class_names)
```

Out[4]:

```
['daisy' 'dandelion' 'roses' 'sunflowers' 'tulips']
```

In[5]:

```

normalization_layer = tf.keras.layers.Rescaling(1./255)
train_ds = train_ds.map(lambda x, y: (normalization_layer(x), y)) # Where x—images, y—labels.
val_ds = val_ds.map(lambda x, y: (normalization_layer(x), y)) # Where x—images, y—labels.

AUTOTUNE = tf.data.AUTOTUNE

train_ds = train_ds.cache().shuffle(1000).prefetch(buffer_size=AUTOTUNE)
val_ds = val_ds.cache().prefetch(buffer_size=AUTOTUNE)

```

3. 搭建模型

卷积神经网络模型声明代码如下：

每个 Conv2D 和 MaxPooling2D 层的输出都是一个三维的张量，其形状描述了 (height, width, channels)。每一次层卷积和池化后输出宽度和高度都会收缩。每个 Conv2D 层输出的通道数量取决于声明层时的 filters 参数（如 32 或 64）。

Dense 层等同于全连接 (Full Connected) 层。在模型的最后将把卷积后的输出张量传给一个或多个 Dense 层来完成分类。Dense 层的输入为向量，但前面层的输出是 3 维的张量。因此需要使用 layers.Flatten() 将三维张量展开到 1 维，之后再传入一个或多个 Dense 层。数据集有 5 个类，因此最终的 Dense 层需要 5 个输出及一个 softmax 激活函数。

In[6]:

```

num_classes = len(class_names)
model = tf.keras.Sequential([
    layers.Conv2D(16, 3, padding='same', activation='relu'),
    layers.MaxPooling2D(),
    layers.Conv2D(32, 3, padding='same', activation='relu'),
    layers.MaxPooling2D(),
    layers.Conv2D(64, 3, padding='same', activation='relu'),
    layers.MaxPooling2D(),
    layers.Flatten(),
    layers.Dense(128, activation='relu'),
    layers.Dense(num_classes)
])

```

如果数据集没有采集到足够的图片，可以使用 RandomFlip、RandomRotation（旋转和水平翻转）对训练图像随机变换的方法来人为引入样这多样性。

4. 编译，训练模型

在训练之前先编译模型，损失函数使用类别交叉熵。

In[8]:

```

model.compile(
    optimizer=tf.keras.optimizers.Adam(),
    loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
    metrics=['acc'])

log_dir = "logs/fit/" + datetime.datetime.now().strftime("%Y%m%d-%H%M%S")
tensorboard_callback = tf.keras.callbacks.TensorBoard(
    log_dir=log_dir,

```

```
histogram_freq=1) # Enable histogram computation for every epoch.
```

训练模型，训练和验证准确性/损失的学习曲线如图 9-7 所示。

In[9]:

```
NUM_EPOCHS = 10
```

```
history = model.fit(train_ds,  
                    validation_data=val_ds,  
                    epochs=NUM_EPOCHS,  
                    callbacks=tensorboard_callback)
```



图 9-21 学习曲线

5. 转换为 TFLite 格式

使用 `tf.saved_model.save` 保存模型，然后将模型保存为 `tf lite` 兼容格式。

`SavedModel` 包含一个完整的 `TensorFlow` 程序——不仅包含权重值，还包含计算。它不需要原始模型构建代码就可以运行。

In[10]:

```
saved_model_dir = 'save/fine_tuning'  
tf.saved_model.save(model, saved_model_dir)  
  
converter = tf.lite.TFLiteConverter.from_saved_model(saved_model_dir)  
tflite_model = converter.convert()  
  
with open('save/fine_tuning/assets/model.tflite', 'wb') as f:  
    f.write(tflite_model)
```

模型文件保存在 save\fine_tuning\assets 目录下。

8.5.3 将 TensorFlow Lite 模型部署到 Jetson Nano 开发板

上一节已经使用 MobileNet V2 创建、训练和导出了自定义 TensorFlow Lite 模型，将训练的 TF Lite 模型文件和标签文件拷贝到 Jetson Nano 开发板。接下来将在 Jetson Nano 开发板上部署，使用该模型识别花卉图片。

1. 复制模型文件

将训练好的模型复制到 Jetson Nano 开发板上，可以创建一个花卉的类别名字用于显示 class_names：郁金香(tulips)、玫瑰(roses)、蒲公英(dandelion)、向日葵(sunflowers)、雏菊(daisy)。

```
model.tflite
class_names
flower.ipynb
```

不能在 Jetson Nano 开发板训练网络，回提示错误 OOM when allocating tensor with shape[64,96,112,112]。出现以上类似的错误，Jetson Nano 开发板资源有限，或者模型中的 batch_size 值设置过大，导致内存溢出，batch_size 是每次送入模型中的值受限于 GPU 内存的大小。

2. 加载图片

可以从网上自己下载一个图片用于预测，当然也可以调用开发板摄像头。或者直接从官网下载一个图片，如图 9-21。

读取处理图片的方法很多，可以使用 OpenCV 或者使用 TensorFlow 的 API 函数。

In[1]:

```
import tensorflow as tf
assert tf.__version__.startswith('2')
import os
import numpy as np
import matplotlib.pyplot as plt
import PIL

sunflower_url = "https://storage.googleapis.com/download.tensorflow.org/example_images/592px-Red_sunflower.jpg"
sunflower_path = tf.keras.utils.get_file('Red_sunflower', origin=sunflower_url)
PIL.Image.open(sunflower_path)
```

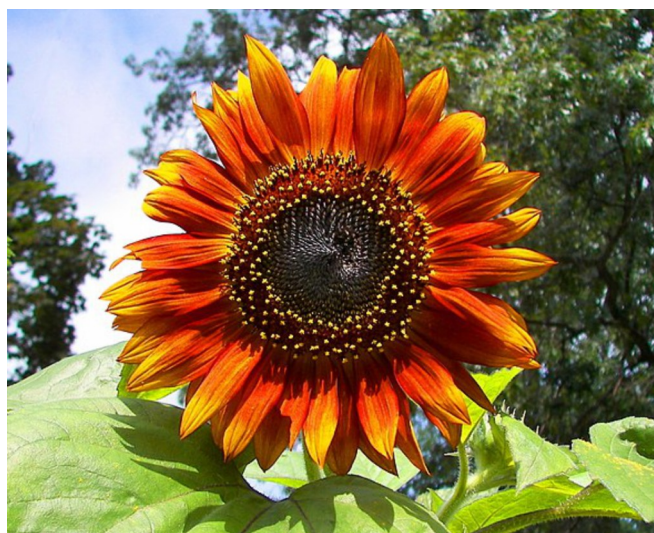


图 9-21 向日葵

3. 执行推理

TensorFlow Lite 解释器初始化后，开始编写代码以识别输入图像。TensorFlow Lite 无需使用 ByteBuffer 来处理图像，它提供了一个方便的支持库来简化图像预处理，同样还可以处理模型的输出，并使 TensorFlow Lite 解释器更易于使用。下面需要做的工作有：

- 1) 数据转换（Transforming Data）：将输入数据转换成模型接收的形式或排布，如 resize 原始图像到模型输入大小。

In[2]:

```
img = tf.io.read_file(sunflower_path)
img_tensor = tf.image.decode_image(img, channels = 3)
img_tensor = tf.image.resize(img_tensor, [224, 224])
img_tensor = tf.cast(img_tensor, tf.float32)
img_array = tf.expand_dims(img_tensor, 0) # Create a batch
```

- 2) 执行推理（Running Inference）：使用 TensorFlow Lite API 来执行模型。其中包括了创建解释器、分配张量等。

TensorFlow 推理 API 支持编程语言、支持常见的移动/嵌入式平台（例如 Android、iOS 和 Linux）。由于资源限制严重，必须在苛刻的功耗要求下使用资源有限的硬件，因此在移动和嵌入式设备上执行推理颇有难度。在 Android 上，可以使用 Java 或 C++ API 来执行 TensorFlow Lite 推理。在 iOS 上，TensorFlow Lite 适用于以 Swift 和 Objective-C 编写的原生 iOS 库，也可以直接在 Objective-C 代码中使用 C API。在 Linux 平台（包括 Raspberry Pi）上，可以使用以 C++ 和 Python 提供的 TensorFlow Lite API 运行推断。

运行 TensorFlow Lite 模型涉及几个简单步骤：

- 将模型加载到内存中。
- 基于现有模型构建 Interpreter。
- 设置输入张量值。（如果不需要预定义的大小，则可以选择调整输入张量的大小。）
- 执行推理。
- 读取输出张量值。

In[3]:

```
# Load the TFLite model and allocate tensors.
```



```

interpreter = tf.lite.Interpreter(model_path="model.tflite")
interpreter.allocate_tensors()

input_details = interpreter.get_input_details()
output_details = interpreter.get_output_details()

interpreter.set_tensor(input_details[0]['index'], img_array)
interpreter.invoke()

```

3) 解释输出（Interpreting Output）：用户取出模型推理的结果，并解读输出，如分类结果的概率。

In[4]:

```

# The function `get_tensor()` returns a copy of the tensor data.
# Use `tensor()` in order to get a pointer to the tensor.
output_data = interpreter.get_tensor(output_details[0]['index'])
print(output_data)
class_names = ['daisy', 'dandelion', 'roses', 'sunflowers', 'tulips']
score = tf.nn.softmax(predictions[0])
print(
    "This image most likely belongs to {} with a {:.2f} percent confidence."
    .format(class_names[np.argmax(score)], 100 * np.max(score))
)

```

Out[4]:

```

[[-3571.6514 -3989.4302  32.3793 2005.5446  717.6471]]
This image most likely belongs to sunflowers with a 100.00 percent confidence.

```

以上代码 TensorFlow 版本为 2.3.0，版本变化后 API 函数会改变，所以请注意版本。