Jetson-Nano 国产套件使用手册

Jetson Nano Domestic Development Kit Guide

让世界更智慧 让生活更美好

图为信息科技(深圳)有限公司
TWOWIN TECHNOLOGY Co. Ltd

目录

文档修订目录

文档版本

Version2.0

文档版本号	修订日期	修订内容
V1.0	2022/1/6	初始发布
V2.0	2022/5/31	内容修改

前言

在使用本手册之前，请您认真阅读以下使用许可协议，只有在同意以下使用许可协议的情况下方能使用本手册中介绍的产品。

版权声明

免责条款

本文档依据现有信息制作，其内容如有更改，恕不另行通知。图为信息科技(深圳)有限公司在编写该文档的时候已尽最大努力保证其内容准确可靠，但图为信息科技(深圳)有限公司不对本文档中的遗漏、不准确、或错误导致的损失和损害承担责任。

技术支持与信息反馈

如果您在使用我们的产品时遇到问题,或者您认为我们的产品有某些功能缺陷,请访问我们的官网 www:https://twowin.com 联系我们的客服,我们将为您解决问题和反馈;或者需要技术支持指导以及有任何宝贵意见，也请您通过官网或者电话联系我们：

联系人 ：图为技术支持

手机：15920093612

电话：0755-82840481

网址：www.twowinit.com

地址：深圳市南山区科技南十二路长虹科技大厦 2512

安全警示及使用注意事项

- 安全说明

在使用本产品之前，必须先查阅本文档,对该产品有初步的认识与了解,且须遵守本产品使用手册中的安全说明以保证您的个人安全并避免损坏设备,若盲目操作造成损失或伤害,制造商对其错误操作造成的设备及个人生命财产安全的任何问题均不负责。

- 电源电压

Jetson nano 国产套件边缘计算平台输入端电源稳定可靠，功率最高 15W。

电源范围： 9 - 19 V DC；电流：5A(MAX)

- 环境要求：

工作温度： -20℃ - 65℃

通风要求： 计算平台安装的周边必须有良好通风的条件。

- 接地要求

电源适配器的供电源必须有良好的接地，特殊情况需安装计算平台上接地螺丝接地。

- 静电防护

电子元件和电路对静电放电很敏感，虽然本公司在设计电路板卡产品时会对板卡上的主要接口做防静电保护设计，但很难对所有元件及电路做到防静电安全防护。因此在处理任何电路板组件时，建议遵守防静电安全保护措施。防静电安全保护措施包括，但不限于以下几点：

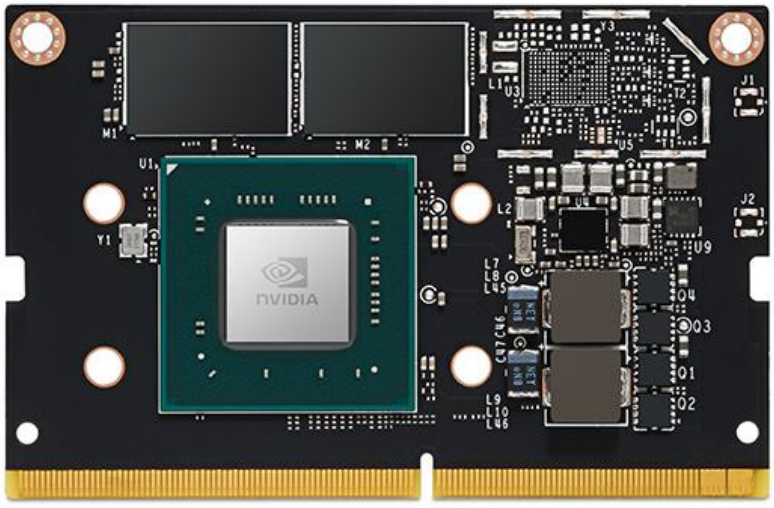
- 运输、存储过程中应将盒子放在防静电袋中，直至安装部署时再拿出板卡;
- 在身体接触盒子之前应将身体内寄存的静电释放掉：佩戴放电接地腕带;
- 仅在静电放点安全区域内操作盒子;
- 避免在铺有地毯的区域搬移盒子。

- 操作与维护

操作或维护人员需先经培训合格，方可参与操作或维护。

1. Jetson Nano 简介

Jetson Nano 模块的大小仅为 70 x 45mm，比一张信用卡还小。而在为多个行业（包括智慧城市、智慧工厂，以及农业和机器人）的边缘设备部署 AI 时，此支持量产的模组系统 (SOM) 可以提供强大支持。



Jetson Nano 可为运行现代 AI 算法提供 472 GFLOPS 的计算性能。它能并行运行多个神经网络并同时处理多个高分辨率的传感器，颇为适合在 NVR 和智能网关领域的应用程序。现在，凭借强大且高效的 AI、计算机视觉和高性能计算，您可在边缘端进行创新，而功耗仅为 5 至 10 瓦。

下面详细列举一些 Jetson Nano 的优势：

(1) 体型小巧，性能强大，价格实惠，整体采用类似树莓派的硬件设计，支持一系列流行的 AI 框架，并且英伟达投入了大量的研发精力为其打造了与之配套的 Jetpack SDK 开发包，通过该开发包可以使学习和开发 AI 产品变得更加简单和便捷。

(2) 专为 AI 而设计，性能相比树莓派更强大，搭载四核 Cortex-A57 处理器，128 核 Maxwell GPU 及 4GB LPDDR 内存，可为机器人终端、工业视觉终端带来足够的 AI 算力。

(3) 可提供 472 GFLOP，支持高分辨率传感器，可以并行处理多个传感器，并可在每个传感器流上运行多个现代神经网络。

(4) 支持英伟达的 NVIDIA JetPack 组件包，其中包括用于深度学习、计算机视觉、GPU 计算、多媒体处理等的板级支持包，CUDA，cuDNN 和 TensorRT 软件库。

支持一系列流行的 AI 框架和算法，比如 TensorFlow，PyTorch，Caffe / Caffe2，Keras，MXNet 等，使得开发人员能够简单快速的将 AI 模型和框架集成到产品中，轻松实现图像识别，目标检测，姿势估计，语义分割，视频增强和智能分析等强大功能。

目前人工智能风口开始逐步进入落地应用阶段，更多产品希望能够将人工智能算力运用于实际终端，即实现所谓的边缘计算需求。从根本上来说近几年推动人工智能的核心在于深度学习算法，但是深度学习的推理加速离不开高速 GPU 的支持，而一般桌面 PC 或服务器级别的显卡（如英伟达 1080Ti 等）价格非常昂贵，不适合边缘计算需求，而且体积也过于庞大。因此，英伟达推出的这款嵌入式人工智能开发板 Jetson Nano 非常契合当前行业需求.

以下为 nano 模块具体参数列表：

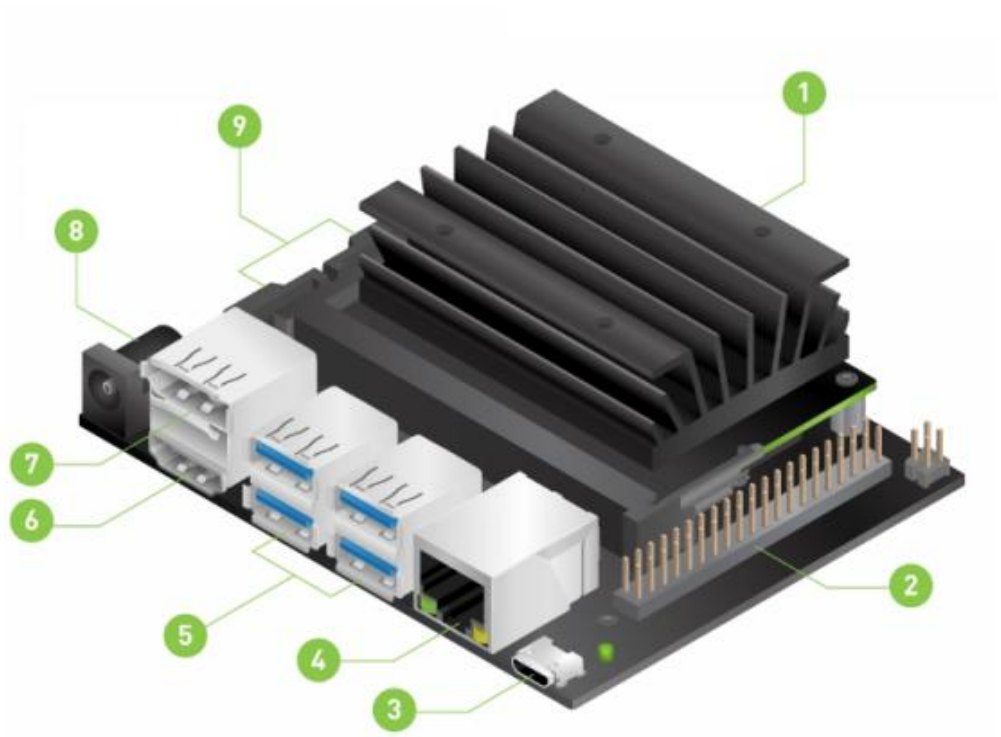
GPU	NVIDIA Maxwell™ 架构，配有 128 个 NVIDIA CUDA® 核心 0.5 TFLOPS (FP16)
CPU	四核 ARM® Cortex®-A57 MPCore 处理器
显存	4 GB 64 位 LPDDR4 1600 MHz – 25.6 GB/s
存储	16 GB eMMC 5.1 闪存
视频编码	250 MP/s 1x 4K @ 30 (HEVC) 2x 1080p @ 60 (HEVC) 4x 1080p @ 30 (HEVC)
视频解码	500 MP/s 1x 4K @ 60 (HEVC) 2x 4K @ 30 (HEVC) 4x 1080p @ 60 (HEVC) 8x 1080p @ 30 (HEVC)
摄像头	12 通道（3x4 或 4x2）MIPI CSI-2 D-PHY 1.1 (18 Gbps)
网络	10/100/1000 BASE-T 以太网
显示器	HDMI 2.0 或 DP1.2 | eDP 1.4 | DSI (1 x2) 2 同步
UPHY	1 个 x1/2/4 PCIE、1 个 USB 3.0、3 个 USB 2.0
I/O	3 个 UART、2 个 SPI、2 个 I2S、4 个 I2C、多个 GPIO
大小	69.6 mm x 45 mm
规格尺寸	260 引脚边缘连接器

2. Jetson Nano 环境配置

2.1 开箱配件介绍

直接购买的 Jetson Nano 包含一台裸机,micro-usb 数据线.两根杜邦线,如下图所示：

下图显示了 Jetson Nano 裸机的各个接口：



1：散热片

2:　40 个 pin 角的 GPIO 接口，主要用于连接外部设备，如温控器、水平仪等；NVIDIA 官方提供了了 JetsonGPIO 库（Python）可以方便的来控制 GPIO，Jetson.GPIO 库运用了跟树莓派 RPi.GPIO 库一样的 API。

GPIO 调用方法:(下面是以 GPIO216 为例说明)

sudo chmod 777 /sys/class/gpio/export

sudo echo 216 > /sys/class/gpio/export

sudo chmod 777 /sys/class/gpio/gpio216/value

sudo chmod 777 /sys/class/gpio/gpio216/direction

sudo　echo "out" > /sys/class/gpio/gpio216/direction(此设置是将 GPIO 设置为输出)

sudo　echo "in" > /sys/class/gpio/gpio216/direction(此设置是将 GPIO 设置为输入)

sudo echo 1 > /sys/class/gpio/gpio216/value(此设置是将 GPIO 设置为高电平)

sudo echo 0 > /sys/class/gpio/gpio216/value(此设置是将 GPIO 设置为低电平)

Jetson Nano J41 Header					
Sysfs GPIO	Name	Pin	Pin	Name	Sysfs GPIO
	3.3 VDC *Power*	1	2	**5.0 VDC** *Power*	
	I2C_2_SDA *I2C Bus 1*	3	4	**5.0 VDC** *Power*	
	I2C_2_SCL *I2C Bus 1*	5	6	**GND**	
gpio216	**AUDIO_MCLK**	7	8	**UART_2_TX** */dev/ttyTHS1*	
	GND	9	10	**UART_2_RX** */dev/ttyTHS1*	
gpio50	**UART_2_RTS**	11	12	**I2S_4_SCLK**	gpio79
gpio14	**SPI_2_SCK**	13	14	**GND**	
gpio194	**LCD_TE**	15	16	**SPI_2_CS1**	gpio232
	3.3 VDC *Power*	17	18	**SPI_2_CS0**	gpio15
gpio16	**SPI_1_MOSI**	19	20	**GND**	
gpio17	**SPI_1_MISO**	21	22	**SPI_2_MISO**	gpio13
gpio18	**SPI_1_SCK**	23	24	**SPI_1_CS0**	gpio19
	GND	25	26	**SPI_1_CS1**	gpio20
	I2C_1_SDA *I2C Bus 0*	27	28	**I2C_1_SCL** *I2C Bus 0*	
gpio149	**CAM_AF_EN**	29	30	**GND**	
gpio200	**GPIO_PZ0**	31	32	**LCD_BL_PWM**	gpio168
gpio38	**GPIO_PE6**	33	34	**GND**	
gpio76	**I2S_4_LRCK**	35	36	**UART_2_CTS**	gpio51
gpio12	**SPI_2_MOSI**	37	38	**I2S_4_SDIN**	gpio77
	GND	39	40	**I2S_4_SDOUT**	gpio78

3：micro-usb，默认使用该接口作为刷机接口；

4：有线网口；直接将网线接在该口上进行连网；

5：4 个 USB3.0 口，用于接入 USB 设备，例如 USB 型摄像头；

6：HDMI 输出端口，可以用来接显示屏；

7：DP 显示接口，可以用来接显示屏；（接口 6 和接口 7 都是用来接外设显示屏的，只是接口不同而已，实际使用时只采用一个即可，通常一般用 HDMI 接口）。

8： 直流 9-19V 输入电源；

9： MIPI CSI 摄像头，可以直接购买用于树莓派的摄像头即可使用；

除了裸机以外，还需自行购买键盘、鼠标、显示器、无线网卡、摄像头、风扇、外壳等配件才能更好的进行开发。

下面对部分配件选型和安装进行说明。

键盘鼠标可以直接使用支持 USB3.0 即可；

显示器：建议使用 HDMI 的显示器直接连接,不采用其他转接方式进行屏幕连接(使用 vga 或者其他转接会导致屏幕无显示)

无线网卡：联网可以采用有线和无线两种方式，本机有独立 WIFI 板载模块,在载板背面,可用于连接无线网络(为 2.4G/5G 双频段)

摄像头：摄像头可以采用两种方式，一种是直接使用树莓派的 CSI 摄像头，还有就是使用 USB 摄像头。如果使用树莓派摄像头，则按照下图所示方式接入即可（注意正反面）：

风扇：一般情况下，不需要使用风扇对 Jetson Nano 进行散热处理，但是如果用到深度学习技术，并且高频率的进行推理运算，那么最好需要在 GPU 组件上面加装一个散热风扇;如果购买的风扇带有 pwm 控制,就可以通过修改/sys/devices/pwm-fan/target_pwm 进行风扇速度控制,数值范围是 0-255,该控制在完全断电在开机后会重置,所以仅对当次设置有效,另外可通过安装 jtop 对设备风扇进行控制.

jtop 安装参考:https://www.jianshu.com/p/497a9f6e34fd

外壳：裸露的开发板容易短路，从安全性考虑，在安装完上述配件以后最好为整个 Jetson Nano 安装一个外壳。外壳种类多样，可以根据实际购买的外壳说明进行安装.

2.2 烧录系统

英伟达官方为 Jetson Nano 提供了 SD 卡版本的系统镜像，并且一直在更新和维护，但 sd 卡镜像不能用在我们这款国产 nano 开发套件上,因为我们这款国产套件使用的 nano 模块不带外置 sd 卡槽,它的刷机方式只能只用线刷的方式,通过 nvidia 官网提供的 sdk manager 工具或者我司提供的系统镜像两种方式进行刷机,以下进行分别介绍,两种方式任选一种即可.

2.2.1 SDK manager 介绍

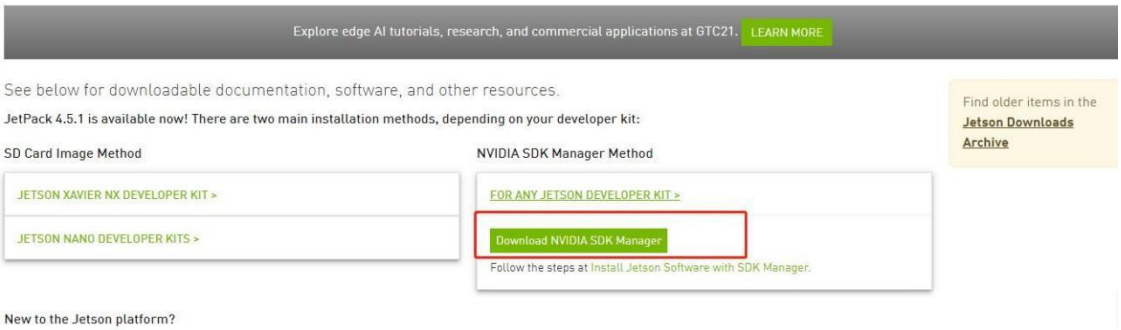
SDK manager 是 NVIDIA 官方提供的一个管理工具,用来进行 jetson 系列产品的系统及软件安装,是一个优缺点并存的工具.优点是支持目前 nvidia jetson 系列所有产品的系统及软件安装, 步骤简单,操作方便,整合了多个 jetpack 版本以及对应的 sdk 版本,无需再去官网下载 L4T 文件, 方便用户能更好安装例如 cuda,cudnn,opencv 以及 deepstream 这类 sdk;缺点是对于网络环境有要求,可能在使用过程中会出现登录不上以及安装报错等情况,这些均由网络环境造成,可能需要科学上网才可以正常使用.

2.2.2 安装与准备

首先说明 SDK manager 是需要安装在一台 ubuntu18.04(其他版本应该也可以,我没试过)的电脑上,而不是安装在任何一款 jetson 系列产品上.因为 jetson 系列产品为 arm64 位架构,SDK manager 并不支持.

SDK manager 官网下载链接：

https://developer.nvidia.com/embedded/downloads



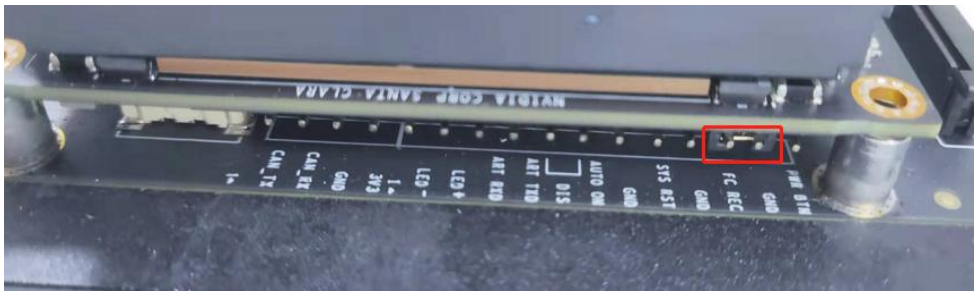
百度云盘下载链接:

链接：https://pan.baidu.com/s/1HOLORIpblrhpmJvzweYfOg

提取码：tw12

2.2.3 系统及软件安装

首先将 jetson nano 国产套件与 ubuntu 电脑通过 micro-usb 数据线相连,使用跳线帽短接 rec 和 gnd 针脚,如下图所示,连接完毕后，需要给 jetson 设备上电,不需要另外接显示器等其他外设.(具体操作查看另外附带的 SDK Manager 使用说明介绍文档)



2.2.4 使用镜像包刷机

镜像包下载地址

链接：https://pan.baidu.com/s/1DNRKm16jLxcxAcaC0pjBnQ
提取码：0zdv

具体使用方法请查看镜像包里的 readme.txt 文件.

2.3 开机和基本设置

完成烧录后拔掉电源重启进入设置页面,初次安装需要进行基本的配置，包括账户密码、Wifi 密码、输入法、时区等，具体根据提示完成即可，配置完成后会默认进行一次更新操作 Applying Changes，此更新需要联网，如果没有联网则先单击 cancel 取消等后面联网了再进

行手动更新。更新时间较长，等待即可。最后会进入桌面，如下图所示：

2.4 开发环境配置（以下内容非必须进行的操作，请按照个人需求进行相应操作）

2.4.1 更新源和软件

安装完系统后首先应该更新源，否则后续更新和升级会非常慢。但是由于 Jetson nano 采用的是 aarch64 架构的 Ubuntu 18.04.2 LTS 系统，与 AMD 架构的 Ubuntu 系统不同，因此需要替换成 aarch64 的源，这里一定要注意，不要替换成 x86-64 的源了。

我们这里选择清华的源进行更新。首先备份原本的源，更改 source.list 文件的名字，以备不时之需：

```
sudo cp /etc/apt/sources.list /etc/apt/sources.list.bak
sudo gedit /etc/apt/sources.list
```

然后删除所有内容，复制以下内容：

deb http://mirrors.tuna.tsinghua.edu.cn/ubuntu-ports/ bionic main multiverse restricted universe

deb http://mirrors.tuna.tsinghua.edu.cn/ubuntu-ports/ bionic-security main multiverse restricted universe

deb http://mirrors.tuna.tsinghua.edu.cn/ubuntu-ports/ bionic-updates main multiverse restricted universe

deb http://mirrors.tuna.tsinghua.edu.cn/ubuntu-ports/ bionic-backports main multiverse restricted universe

deb-src http://mirrors.tuna.tsinghua.edu.cn/ubuntu-ports/ bionic main multiverse restricted universe

deb-src http://mirrors.tuna.tsinghua.edu.cn/ubuntu-ports/ bionic-security main multiverse restricted universe

deb-src http://mirrors.tuna.tsinghua.edu.cn/ubuntu-ports/ bionic-updates main multiverse restricted universe

deb-src http://mirrors.tuna.tsinghua.edu.cn/ubuntu-ports/ bionic-backports main multiverse restricted universe

到这里换源就结束了。

打开终端，输入下述命令进行更新：

sudo apt-get update

sudo apt-get full-upgrade

上述更新时间较长，中间可能由于网速的关系会更新失败，此时不要关机重新执行命令即可，会自动断点续传的。

2.4.2 关屏时间设置

Jetson nano 默认 5 分钟内不操作 Jetson 即会关闭屏幕，重新开启屏幕需要重新输入开机密码。由于在开发 Jetson nano 的过程中经常需要等待，因此并不希望频繁的开启屏幕，而是希望屏幕一直打开。打开 System Settings 进入系统设置界面，如下图所所示：

单击 Brightness & Lock，然后将 Turn screen off when inactive for 改为 Never 即可，如下图所示：

2.4.3 安装中文输入法

由于在开发过程中经常需要使用中文搜索以及书写必要的中文注释，所以推荐为系统安装中文输入法。Jetson nano 自带 ibus 中文输入法，但是要简单的配置下才能进行中文的输入。在终端中直接输入命令 ibus 会出现下图所示界面，说明 Jetson nano 已经自带了 ibus 输入法环境了.

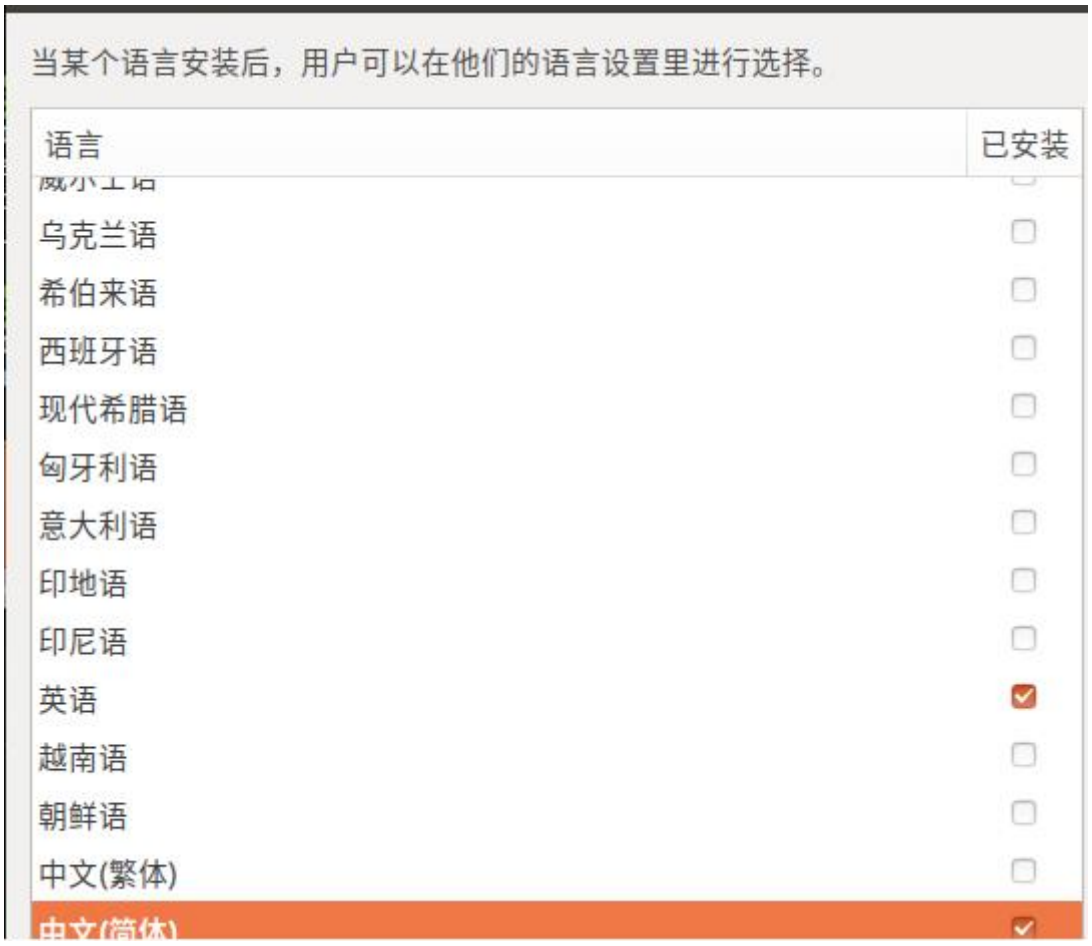
下面为 ibus 下载拼音输入法，输入命令：

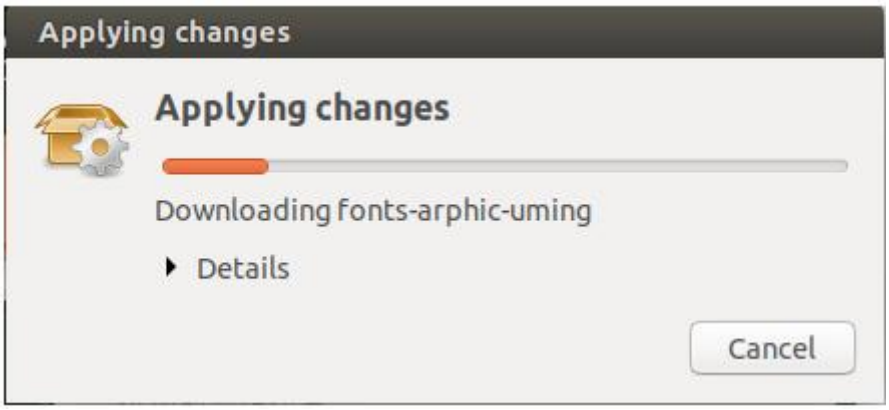
sudo apt-get install ibus-pinyin

上述下载和安装大概需要数十分钟的时间。安装完成后进入系统配置 System Settings 界面，选择语言支持选项 Language Support ，如下图所示:

然后选择“添加或删除语言”界面，会系统选择语言支持，如下图所示：

此处选择“中文简体”然后单击 Apply 即可。这个 Apply 过程会安装一系列中文语言包，如下图所示：

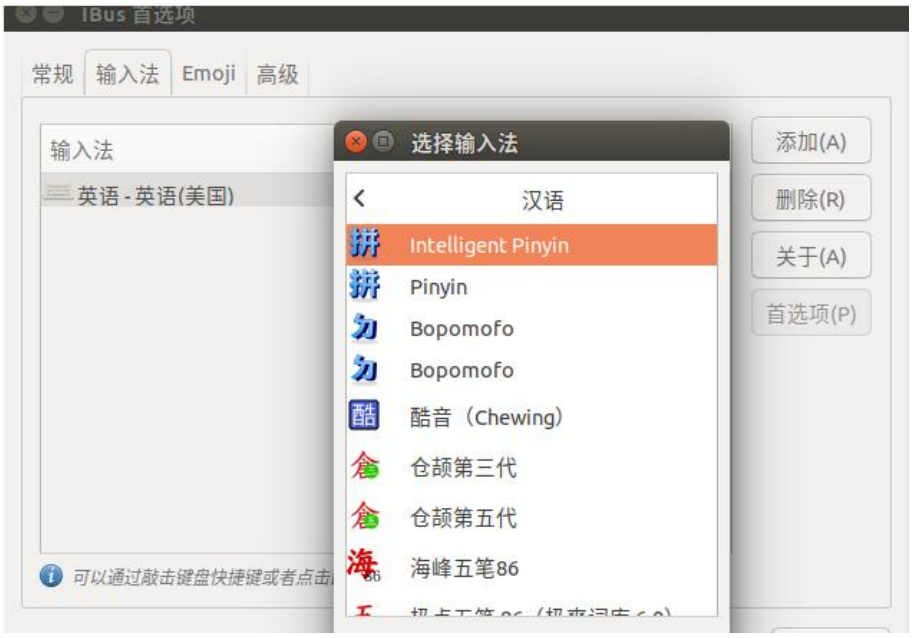




安装完成后在语言支持界面将汉语调整到最前面，如下图所示：

然后单击“应用到整个系统”。最后将“键盘输入法系统”改为 iBus 即可。

重新启动系统（必须），然后在终端中输入下述命令进入 ibus 配置界面：



添加完成后输入下面的命令重启 ibus 即可.

```
ibus restart
```

最后，将桌面顶任务栏中将输入法切成拼音输入法 Pi，如下图所示：

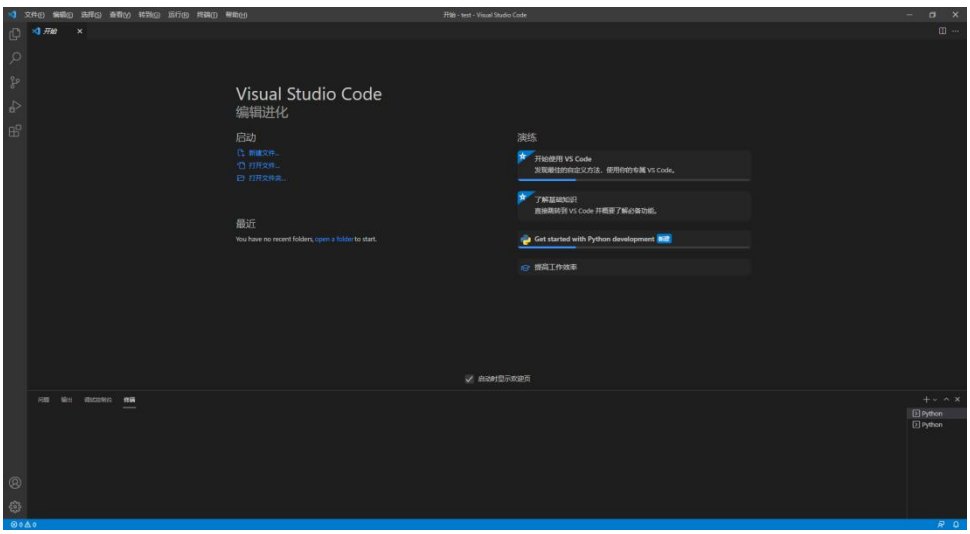
此时就可以打开浏览器输入中文了

2.4.4 安装 VScode

安装:

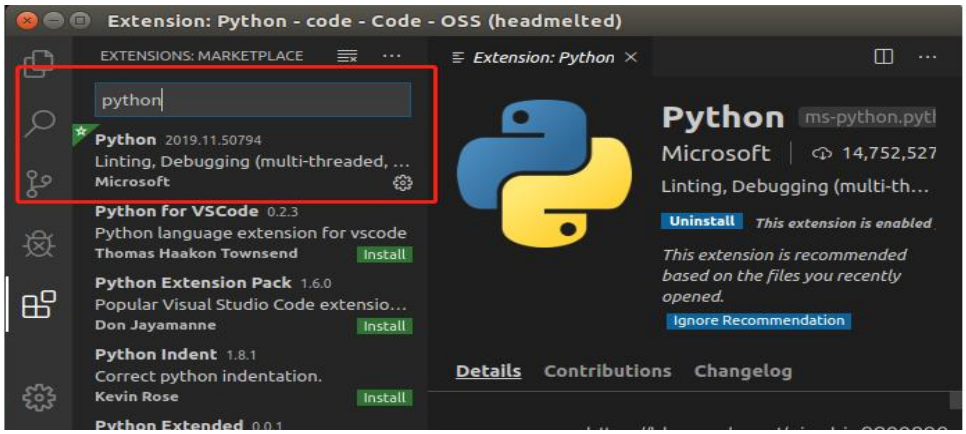
sudo dpkg -i code_1.63.2-1639561157_arm64.deb

安装完成后从可以在搜索中搜索 vscode，会弹出 vscode 应用程序，这个即为我们需要的 Python 编程 IDE。单击应用程序打开如下图所示：



下面简单演示下如何使用 vscode 执行 Python 脚本。

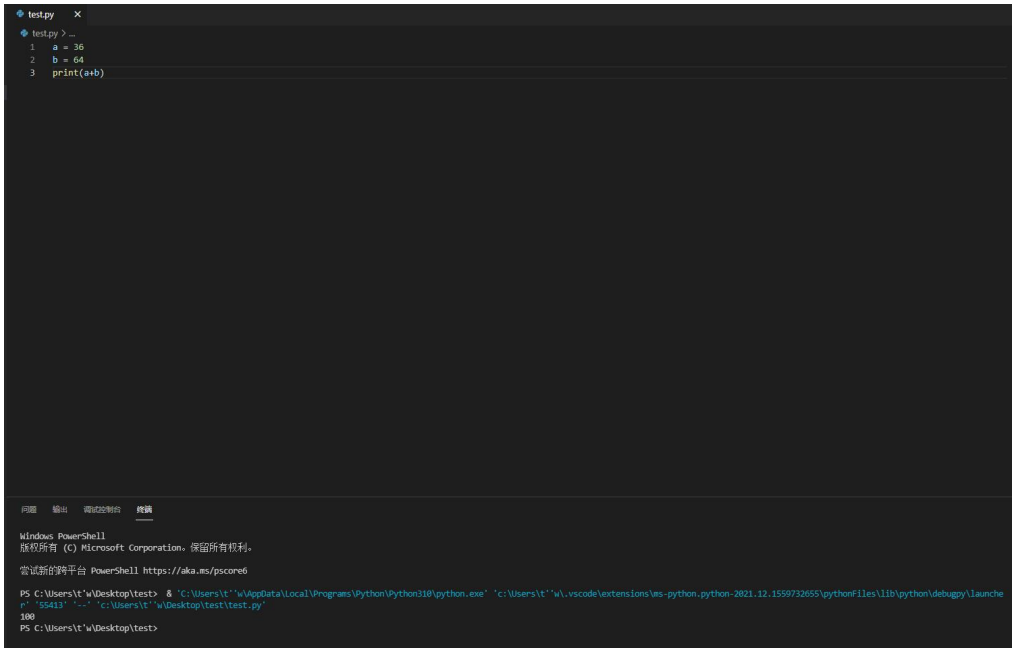
首先在 vscode 中安装 Python 插件，不熟悉 VS Code 的读者可以先在桌面 PC 上熟悉 VS Code 基本用法再切换到 Jetson nano 环境中来。插件安装如下图所示，在 Extensions 面板中搜索 python，选择第一个弹出的插件进行安装即可：



接下来在 home 目录下新建一个 code 文件夹，该文件夹用于存放 Python 代码脚本。然后在 vscode 中打开刚才创建的 code 文件夹，然后新建一个文件，按 ctrl+s 键保存文件，将文件命名为 main.py，然后输入下面的代码：

```
a = 36
b = 64
print(a+b)
```

然后按 ctrl+F5 键即可运行脚本，效果如下：



至此，已完成 Python 编辑器的安装和运行。

2.4.5 安装 Qt5

在实际的产品部署阶段，考虑到终端设备速度、稳定性、内存占用等因素，一般会采用 C++来开发最终的成品，而只有在产品模型设计阶段才会使用 python 进行算法开发。因此，需要一款能够在 Jetson nano 中开发 C++的编译器方便我们开发落地产品。VS Code 本身可以开发 C++应用，但是 Code-OSS 对于 C++的支持并不好，因此，需要另外安装一个优秀的 C++编译器来完成 C++开发任务。本文推荐使用 Qt。

Qt 是一个跨平台的 C++ 开发库，主要用来开发图形用户界面（Graphical User Interface，GUI）程序，当然也可以开发不带界面的命令行（Command User Interface，CUI）程序。Qt 是纯 C++开发的，所以用它来开发 C++应用具有天然的优势。Qt 支持的操作系统有很多，例如通用操作系统 Windows、Linux、Unix，智能手机系统 Android、iOS、WinPhone 以及嵌入式系统 Qnano、VxWorks 等等。当然，QT 也完全支持 Jetson nano 的 Ubuntu 环境。

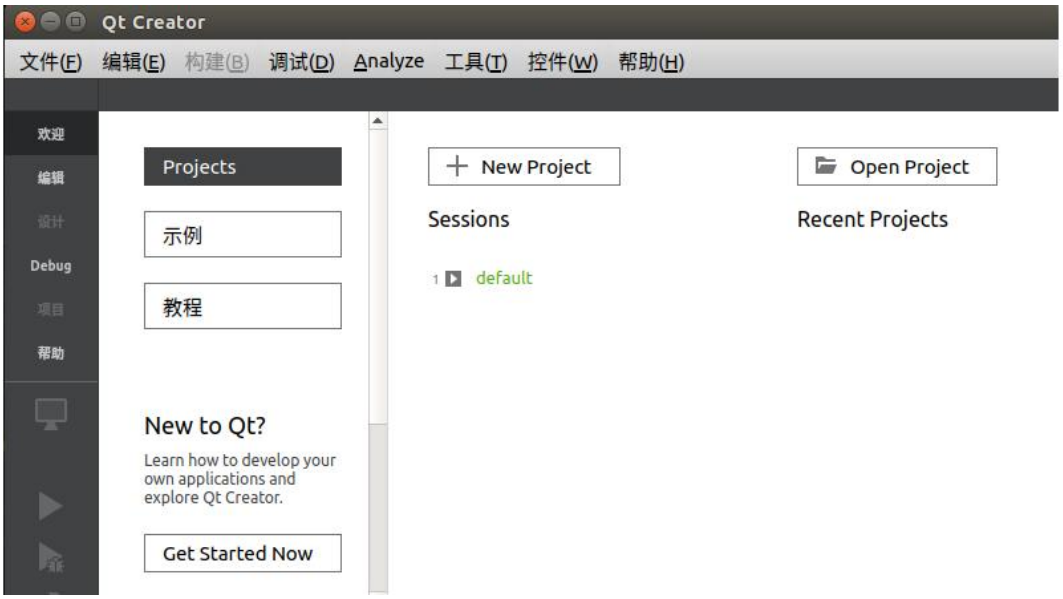
Jetson nano 下安装 QT 比较简单，只需要输入命令：

```
sudo apt-get install qt5-default qtcreator -y
```

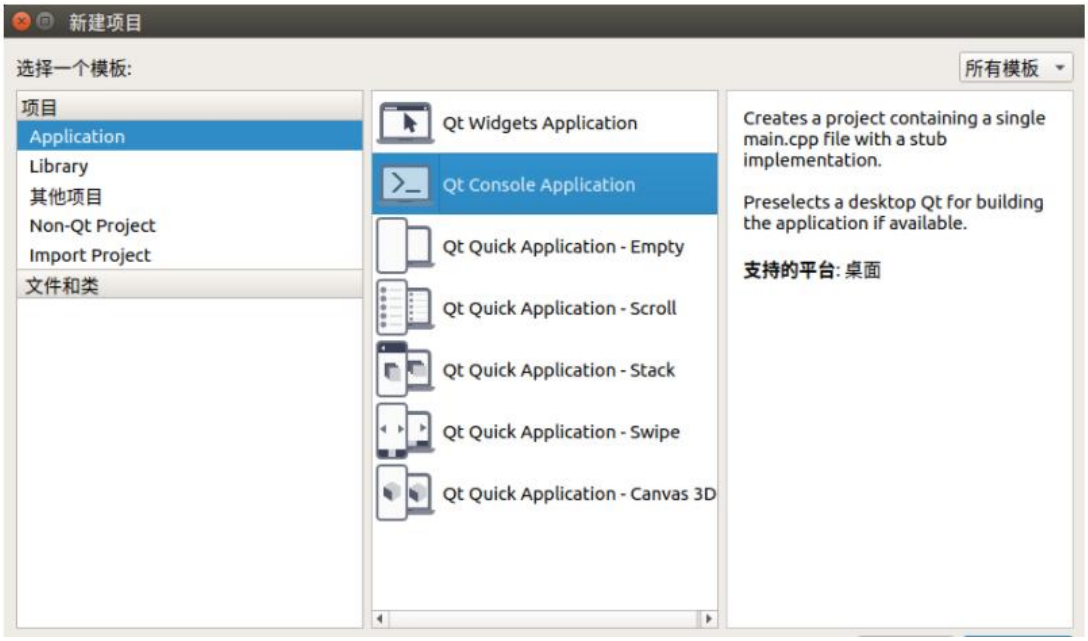
此时安装的是 Qt5.9.5 版本。

安装完成后，同样在搜索菜单中搜索 Qt，然后会出现 Qt Creator，这个即为 Qt 的 IDE，打开它。接下来简单演示如何创建一个简单的 C++控制台程序。

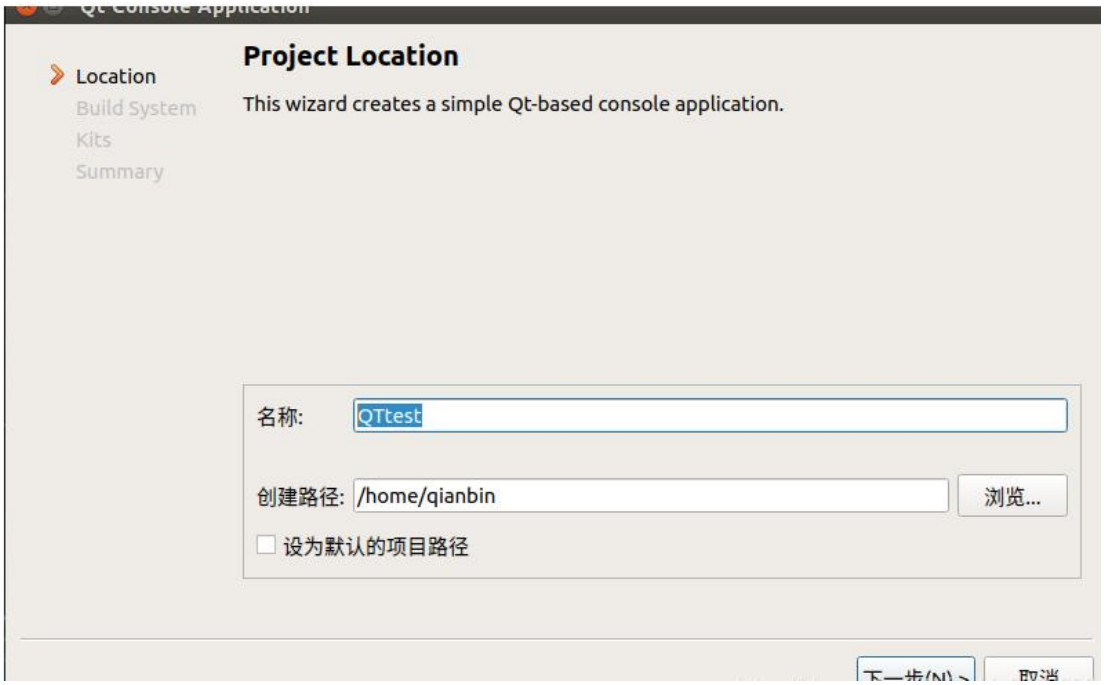
打开 Qt Creator，如下图所示：



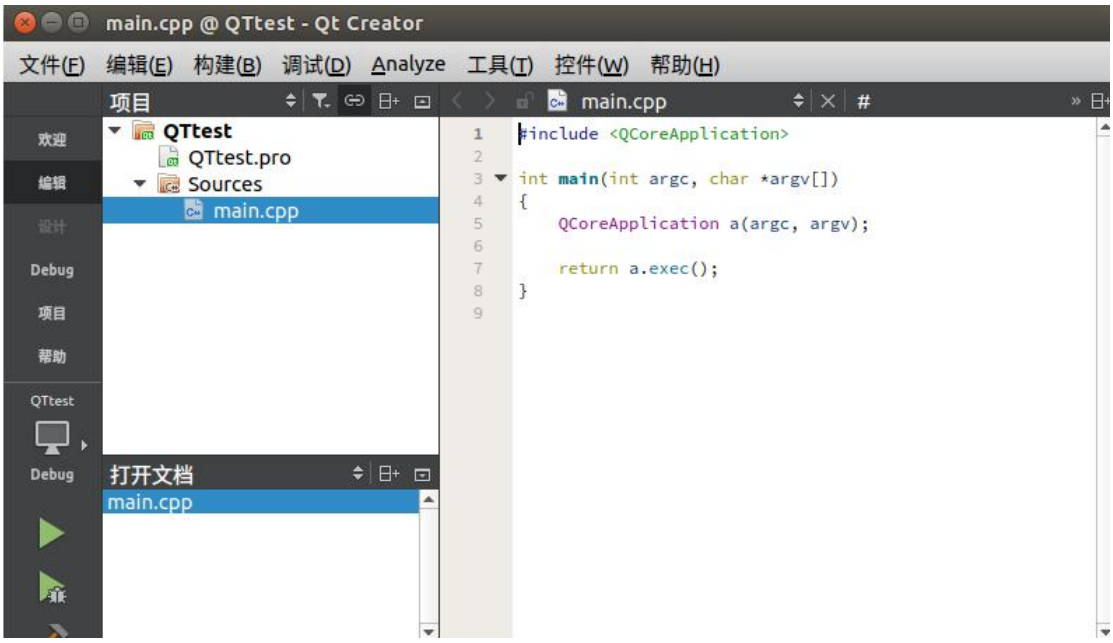
单击 New Project 创建一个新项目，这里选择 Application 下的 Qt COnsole Appliation 应用，即创建一个 Qt 版的 C++控制台程序：



然后工程命名为 QTtest：



然后一直默认单击下一步即可完成项目的创建。可以看到，Qt 已经为我们创建了一个 C++文件 main.cpp 用于编写 C++代码，并且还有一个 QTtest.pro 配置文件用于为整个项目进行配置，效果如下图所示：



此时可以直接按 ctrl+r 键运行项目，但是由于我们并没有任何输出代码，所以弹出的终端也没有输出任何值。我们修改一下 main.cpp 的代码，同样来执行两个整数的相加并输出其结果，完成代码如下：

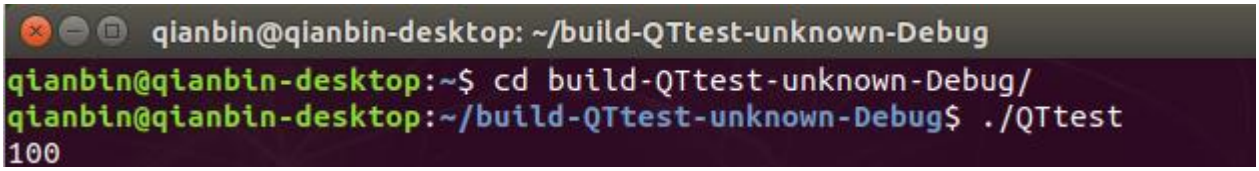
```
#include <QCoreApplication>
#include <QtDebug>

int main(int argc, char *argv[])
{
    QCoreApplication a(argc, argv);
    int c=64;
    int d=36;
    qDebug() << (c+d);
    return a.exec();
}
```

此时，再重新按 ctrl+r 键运行项目输出下图所示结果：

最后我们再看一下在主目录下生成了一个与 QTtest 对照的 debug 可执行项目 build-QTtest-unknown-Debug，在这个文件夹中生成来 debug 版本的 QTtest 可执行程序。通过终端 cd 命令进入到该文件夹，然后输入

./QTtest

会直接执行程序，如下图所示：



也就是说本质上我们已经成功的部署开发了一个应用，该应用功能很简单，仅仅实现了两个固定整数的相加。尽管简单，但是却梳理了我们正常开发人工智能产品的一种比较常见的形式，即先在 VS Code 中用 python 脚本进行算法验证，最后再用 QT 编写对应的 C++应用，最后生成二进制可执行程序，这个最终生成的二进制可执行程序就是我们的“产品”，这个可执行程序代码是封装起来的、不可见的、可以直接运行的。

至此，我们已经完成了 Jetson nano 的常规开发配置，接下来会进行几个小项目的演示，使读者可以更深入的学习 Jetson nano 的开发方法。

3. 项目案例

3.1 人脸检测

本节首先使用 Python 来完成人脸检测算法，其中会讲解 Python 配置和使用 Opencv 的基本方法以及一些常用 python 库的安装。

3.1.1 安装 pip

由于 Jetson nano 中已经预装了 Python3.6 版本，所以可以直接安装 pip。

在终端中输入下述命令进行安装：

```
sudo apt-get install python3-pip python3-dev
```

安装完成后此时的 pip 是 9.01 版本，需要对 pip 进行一下升级，否则后面在安装其它 Python 库的时候会出问题。升级命令如下：

```
python3 -m pip install --upgrade pip
```

升级后版本为 19.0.3。尽管完成了升级，但是这时候 pip3 有个小 bug 需要手动修复一下。

首先使用下面的命令打开 pip3 文件：

```
sudo vim /usr/bin/pip3
```

键盘输入字符 a 进入插入模式，然后可以开始编辑文件，将：

```
from pip import main
if __name__ == '__main__':
    sys.exit(main())
```

修改为：

```
from pip import __main__
if __name__ == '__main__':
    sys.exit(__main__._main())
```

然后按 Esc 键进入到命令模式。最后按英文的":"键进入末行模式，敲入 wq 按回车即可保存修改并退出编辑器。

3.1.2 安装 Python 常用机器学习包

```
sudo apt-get install python3-scipy
sudo apt-get install python3-pandas
sudo apt-get install python3-sklearn
```

3.1.3 配置用于 Python 的 Opencv

有两种方法安装 python 下的 opencv。一种是下载 Opencv 源码并且重新编译生成对应的 python 包，然后将该包拷贝到 python 的安装包路径中；另一种就是直接使用命令 sudo pip3 install python3-opencv。需要注意的是，第二种方式本质上安装的是已经编译好的 opencv 包，其 opencv 的版本是固定的，如果想要使用最新的 opencv，比如 opencv4，那么第二种方法就不合适。本小节先简单的采用第一种方式来安装。

原镜像中已经预装了 opencv4.1.1，可以使用下述命令来查看当前 Opencv 版本号:

```
opencv_version
```

输出结果如下图所示：

```
qianbin@qianbin-desktop:~$ sudo opencv_version
4.1.1
```

因此，我们也不需要重新进行编译，直接使用即可。

3.1.4 基于 Opencv 的人脸检测

（1）python 实现人脸检测

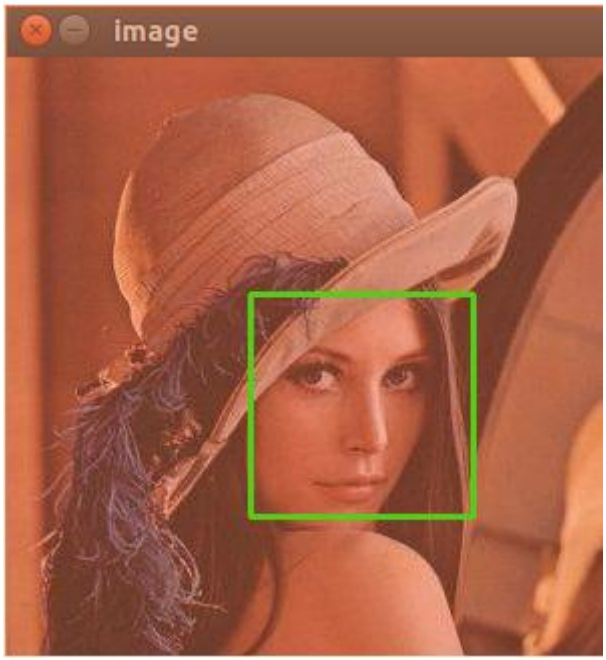
本小节首先编写一个 python 脚本用于检测图像中的人脸，使用 vscode 打开 2.4.4 节中创建的 code 文件夹，在该文件夹下新建一个 python 脚本，名为 face_detect_test.py,代码如下所示：

```
import cv2
filepath = "test.jpg"
img = cv2.imread(filepath) # 读取图片
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY) # 转换灰色
# OpenCV 人脸识别分类器
classifier = cv2.CascadeClassifier( "haarcascade_frontalface_default.xml" )
color = (0, 255, 0) # 定义绘制颜色
# 调用识别人脸
faceRects = classifier.detectMultiScale( gray, scaleFactor=1.2, minNeighbors=3, minSize=(32, 32))
if len(faceRects): # 大于 0 则检测到人脸
    for faceRect in faceRects: # 单独框出每一张人脸
        x, y, w, h = faceRect
        # 框出人脸
        cv2.rectangle(img, (x, y), (x + h, y + w), color, 2)
cv2.imshow("image", img) # 显示图像
c = cv2.waitKey(10)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

上述代码中 filepath 用于存放当前需要检测的图像路径，一般可以放在与源文件同目录下即可。在构造 opencv 人脸检测分类器时，需要对应的人脸检测配置文件，该文件存储了用于

人脸检测算法的相关参数，此文件可以从 opencv 的安装目录找到：/usr/share/opencv4/。找到后将其拷贝到源文件目录下即可。

按 ctrl+F5 运行，效果图如下所示：



（2）C++实现人脸检测

本小节编写一个 C++应用，用于检测图像中的人脸，使用 Qt5 进行开发。相关实现方法与 python 版相同。主要讲解如何在 QT 下集成 Opencv 进行 C++项目开发。

C++下开发 Opencv 需要进行一些额外的配置，先看一下 opencv 的位置。Jetson nano 预装的 Opencv4.1.1 的头文件位置如下图所示：

库文件放置在:

/usr/lib/aarch64-linux-gnu

因此，只需要在 Qt 的 pro 文件中将上述两个目录包含进来即可。

用 Qt Creator 重新打开 2.4.5 节创建的 QTtest 项目，编辑 QTtest.pro 文件如下：

```
QT -= gui

CONFIG += c++11 console
CONFIG -= app_bundle
CONFIG += C++11    #  添加对 C++11 的支持

# The following define makes your compiler emit warnings if you use
# any feature of Qt which as been marked deprecated (the exact warnings
# depend on your compiler). Please consult the documentation of the
# deprecated API in order to know how to port your code away from it.
DEFINES += QT_DEPRECATED_WARNINGS

# You can also make your code fail to compile if you use deprecated APIs.
# In order to do so, uncomment the following line.
# You can also select to disable deprecated APIs only up to a certain version of Qt.
#DEFINES += QT_DISABLE_DEPRECATED_BEFORE=0x060000        # disables all the APIs
deprecated before Qt 6.0.0

INCLUDEPATH += /usr/include/opencv4 #添加头文件路径

LIBS += -L/usr/lib/aarch64-linux-gnu -lopencv_core -lopencv_imgcodecs -lopencv_imgproc
-lopencv_highgui -lopencv_objdetect    #添加需要链接的库
```

SOURCES += main.cpp

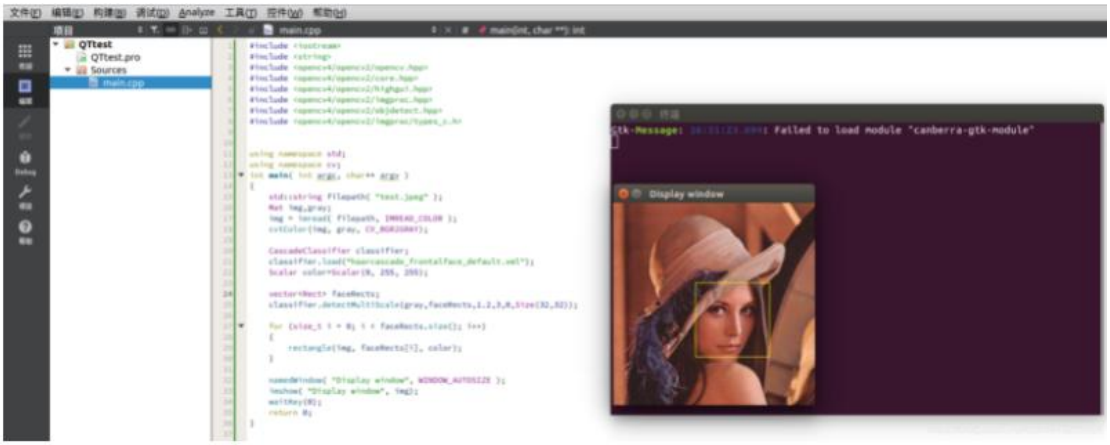
其中重点需要注意头文件和 lib 文件的添加方法。

接下来修改 main.cpp 文件，代码如下：

```
#include <iostream>
#include <string>
#include <opencv4/opencv2/opencv.hpp>
#include <opencv4/opencv2/core.hpp>
#include <opencv4/opencv2/highgui.hpp>
#include <opencv4/opencv2/imgproc.hpp>
#include <opencv4/opencv2/objdetect.hpp>
#include <opencv4/opencv2/imgproc/types_c.h>


using namespace std;
using namespace cv;
int main( int argc, char** argv )
{
    std::string filepath( "test.jpeg" );
    Mat img,gray;
    img = imread( filepath, IMREAD_COLOR );
    cvtColor(img, gray, CV_BGR2GRAY);

    CascadeClassifier classifier;
    classifier.load("haarcascade_frontalface_default.xml");
    Scalar color=Scalar(0, 255, 255);
```

```
    vector<Rect> faceRects;
    classifier.detectMultiScale(gray,faceRects,1.2,3,0,Size(32,32));

    for (size_t i = 0; i < faceRects.size(); i++)
    {
        rectangle(img, faceRects[i], color);
    }

    namedWindow( "Display window", WINDOW_AUTOSIZE );
    imshow( "Display window", img);
    waitKey(0);
    return 0;
}
```

重新生成整个项目，然后将 test.jpeg 和 haarcascade_frontalface_default.xml 文件放置在编译生成的 build-QTtest-unknown-Debug 文件夹中，运行项目效果图如下所示：



3.2 二维码检测（制作扫码枪）

现在支付宝和微信广泛使用二维码作为支付手段，在现实生活购物中我们经常会通过手机展示二维码给商家用于扫码，那么我们是否可以自行做一个扫码仪呢？有了 Jetson nano 这款嵌入式人工智能开发板，我们就可以自己制作一个扫码枪。

3.2.1 读取摄像头

本小节我们希望能够通过摄像头读取图像，并且对图像中的二维码进行实时解析，也就是实现一个扫码仪的功能。本小节实现摄像头读取功能。摄像头一般有两种可选，一种是相对价格比较便宜的 csi 摄像头（树莓派摄像头），还有一种是 USB 摄像头。值得注意的是，如果采用 USB 摄像头，那么图像的读取和渲染则是会用到 Jetson Nano 的 GPU，如果这个时候我们还在做一些深度学习的推理工作，那么很明显会占用掉一些 GPU 资源。相反，Jetson Nano 对于 csi 摄像头的读取和渲染则会采用 Gstreamer 管道来处理，会使用特定的硬件加速，整个处理效果会更好。

本小节我们将详细介绍两种摄像头的读取方式。无论哪种方式，我们均采用 Opencv 这个强大的图像处理开源库作为基础来执行相关操作。

（1）读取 CSI 摄像头

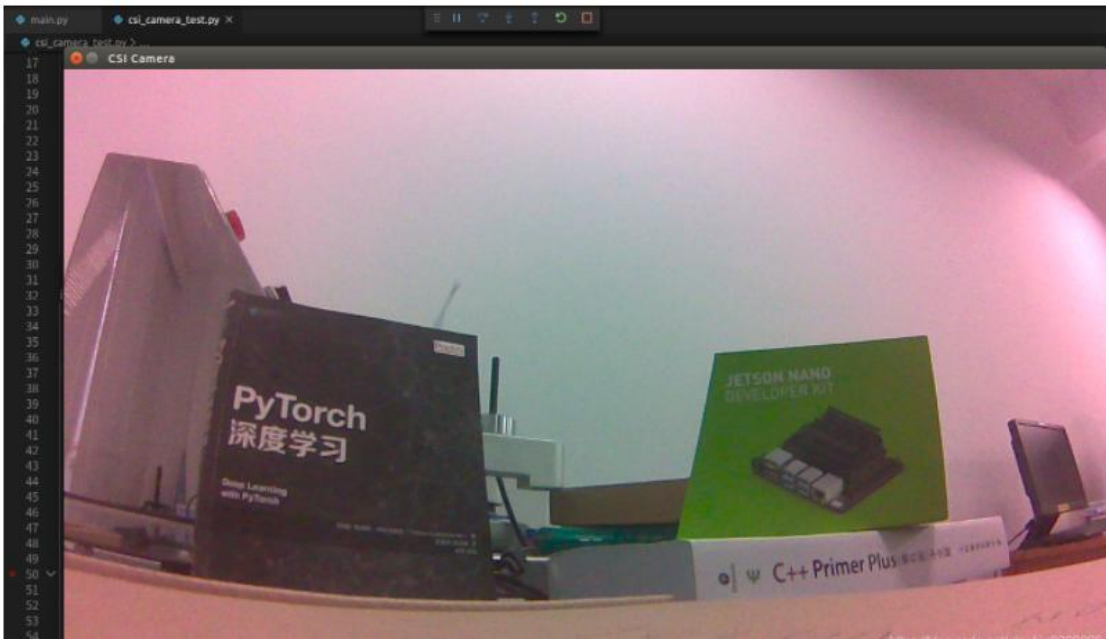
使用 Gstreamer 读取 CSI 摄像头主要分为 3 个步骤：创建 Gstreamer 管道；将管道绑定 opencv 的视频流；逐帧提取和显示。下面首先给出基于 Python 的详细代码：

```
import cv2
#  设置 gstreamer 管道参数
def gstreamer_pipeline(
    capture_width=1280, #摄像头预捕获的图像宽度
    capture_height=720, #摄像头预捕获的图像高度
    display_width=1280, #窗口显示的图像宽度
    display_height=720, #窗口显示的图像高度
```

```
    framerate=60,           #捕获帧率
    flip_method=0,          #是否旋转图像
):
    return (
        "nvarguscamerasrc ! "
        "video/x-raw(memory:NVMM), "
        "width=(int)%d, height=(int)%d, "
        "format=(string)NV12, framerate=(fraction)%d/1 ! "
        "nvvidconv flip-method=%d ! "
        "video/x-raw, width=(int)%d, height=(int)%d, format=(string)BGRx ! "
        "videoconvert ! "
        "video/x-raw, format=(string)BGR ! appsink"
        % (
            capture_width,
            capture_height,
            framerate,
            flip_method,
            display_width,
            display_height,
        )
    )


if __name__ == "__main__":
    capture_width = 1280
    capture_height = 720
    display_width = 1280
    display_height = 720
    framerate = 60
    flip_method = 0
```

```
    # 创建管道

print(gstreamer_pipeline(capture_width,capture_height,display_width,display_height,framerate,
flip_method))

    #管道与视频流绑定
    cap = cv2.VideoCapture(gstreamer_pipeline(flip_method=0), cv2.CAP_GSTREAMER)

    if cap.isOpened():
        window_handle = cv2.namedWindow("CSI Camera", cv2.WINDOW_AUTOSIZE)

        # 逐帧显示
        while cv2.getWindowProperty("CSI Camera", 0) >= 0:
            ret_val, img = cap.read()
            cv2.imshow("CSI Camera", img)

            keyCode = cv2.waitKey(30) & 0xFF
            if keyCode == 27:# ESC 键退出
                break

        cap.release()
        cv2.destroyAllWindows()
    else:
        print("打开摄像头失败")
```

紧接着 3.1.4 节中的第一部分内容，在 Code-OSS 中新建一个文件命名为 csi_camera_test.py，然后将上述代码复制到该文件中，保存然后按 ctrl+F5 运行脚本（前提：确保已经准确安装了 CSI 树莓派摄像头），运行效果如下所示：



可以看到已经可以正常的显示视频流图像了，但是由于树莓派摄像头本身的原因，其图像中还有很多的噪点，颜色也有些失真（真实工业场景中建议购买更好的摄像头）。下面我们同步的给出 C++版本。紧接着 3.1.4 节中第二部分内容，修改 main.cpp 文件如下：

```
#include <iostream>
#include <string>
#include <opencv4/opencv2/opencv.hpp>
#include <opencv4/opencv2/core.hpp>
#include <opencv4/opencv2/highgui.hpp>
#include <opencv4/opencv2/imgproc.hpp>
#include <opencv4/opencv2/objdetect.hpp>
#include <opencv4/opencv2/imgproc/types_c.h>
#include <opencv4/opencv2/videoio.hpp>

using namespace std;
using namespace cv;

string gstreamer_pipeline (int capture_width, int capture_height, int display_width, int
display_height, int framerate, int flip_method)
{
    return "nvarguscamerasrc ! video/x-raw(memory:NVMM), width=(int)" +
to_string(capture_width) + ", height=(int)" +
            to_string(capture_height) + ", format=(string)NV12, framerate=(fraction)" +
to_string(framerate) +
```

```
                "/1 ! nvvidconv flip-method=" + to_string(flip_method) + " ! video/x-raw,
width=(int)" + to_string(display_width) + ", height=(int)" +
                to_string(display_height) + ", format=(string)BGRx ! videoconvert ! video/x-raw,
format=(string)BGR ! appsink";
}

int main( int argc, char** argv )
{
    int capture_width = 1280 ;
    int capture_height = 720 ;
    int display_width = 1280 ;
    int display_height = 720 ;
    int framerate = 60 ;
    int flip_method = 0 ;

    //创建管道
    string pipeline = gstreamer_pipeline(capture_width,
    capture_height,
    display_width,
    display_height,
    framerate,
    flip_method);
    std::cout << "使用 gstreamer 管道: \n\t" << pipeline << "\n";

    //管道与视频流绑定
    VideoCapture cap(pipeline, CAP_GSTREAMER);
    if(!cap.isOpened())
    {
        std::cout<<"打开摄像头失败."<<std::endl;
        return (-1);
    }

    //创建显示窗口
    namedWindow("CSI Camera", WINDOW_AUTOSIZE);
    Mat img;

    //逐帧显示
    while(true)
    {
        if (!cap.read(img))
        {
            std::cout<<"捕获失败"<<std::endl;
            break;
        }
```

```
            imshow("CSI Camera",img);

            int keycode = cv::waitKey(30) & 0xff ; //ESC 键退出
                if (keycode == 27) break ;
        }

        cap.release();
        destroyAllWindows() ;
}
```

其中需要额外的添加 opencv 用于视频处理的头文件#include <opencv4/opencv2/videoio.hpp>。另外，还需要修改对 pro 文件，将视频处理对应的 opencv_videoio 库包含进来，完整的 pro 文件如下：

```
QT -= gui

CONFIG += c++11 console
CONFIG -= app_bundle
CONFIG += C++11   # 添加对 C++11 的支持

# The following define makes your compiler emit warnings if you use
# any feature of Qt which as been marked deprecated (the exact warnings
# depend on your compiler). Please consult the documentation of the
# deprecated API in order to know how to port your code away from it.
DEFINES += QT_DEPRECATED_WARNINGS

# You can also make your code fail to compile if you use deprecated APIs.
# In order to do so, uncomment the following line.
# You can also select to disable deprecated APIs only up to a certain version of Qt.
#DEFINES += QT_DISABLE_DEPRECATED_BEFORE=0x060000      # disables all the APIs
deprecated before Qt 6.0.0

INCLUDEPATH += /usr/include/opencv4 #添加头文件路径
```

LIBS += -L/usr/lib/aarch64-linux-gnu -lopencv_core -lopencv_imgcodecs -lopencv_imgproc -lopencv_highgui -lopencv_objdetect -lopencv_videoio #添加需要链接的库

SOURCES += main.cpp

保存所有修改后重新构建项目并运行可以得到相同的结果，视频可以正确显示。

（2）读取 USB 摄像头

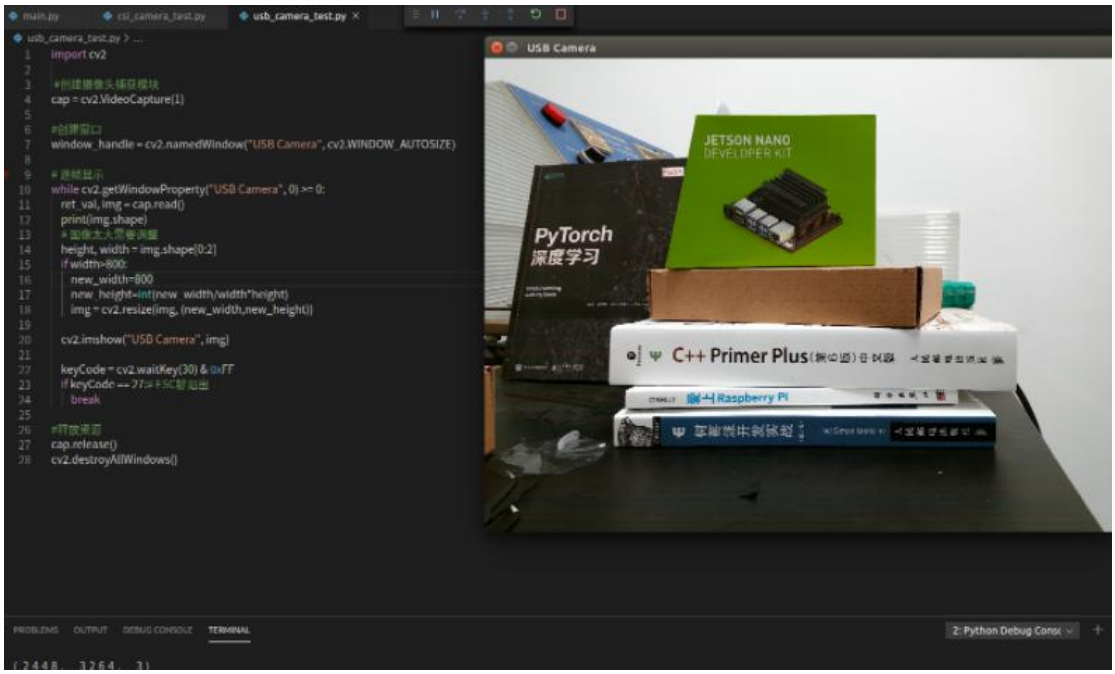
相比于读取 CSI 摄像头，读取 USB 摄像头更加简单，只需要两步：打开摄像头；逐帧提取。但是需要注意的是 Jetson Nano 并不是支持所有的 USB 摄像头，建议在采购的时候尽量选择 Linux 免驱的 USB 摄像头。本文采用的是一个 4K 高清摄像头。

下面给出 Python 版本的完整代码：

```
import cv2

 #创建摄像头捕获模块
cap = cv2.VideoCapture(1)

#创建窗口
window_handle = cv2.namedWindow("USB Camera", cv2.WINDOW_AUTOSIZE)

# 逐帧显示
while cv2.getWindowProperty("USB Camera", 0) >= 0:
    ret_val, img = cap.read()
    print(img.shape)

    # 图像太大需要调整
    height, width = img.shape[0:2]
    if width>800:
        new_width=800
        new_height=int(new_width/width*height)
        img = cv2.resize(img, (new_width,new_height))

    cv2.imshow("USB Camera", img)

    keyCode = cv2.waitKey(30) & 0xFF
```

```
    if keyCode == 27:# ESC 键退出
        break

#释放资源
cap.release()
cv2.destroyAllWindows()
```

上述代码在打开摄像头时使用了 cap = cv2.VideoCapture(1)，这里的参数 1 是因为当前的 Jetson nano 还连接了 CSI 摄像头，CSI 摄像头的标识为 0，因此这个 USB 摄像头的标识为 1，这个可以在实际使用时通过测试来得到。另外，上述代码中对图像的尺寸做了限制，如果宽度超过 800，则等比的缩放图像再显示。效果图如下所示：



可以看到这个 USB 4K 摄像头对于图像的显示效果还是不错的，颜色更加真实，噪点少。后面我们会继续使用这个摄像头进行二维码检测。

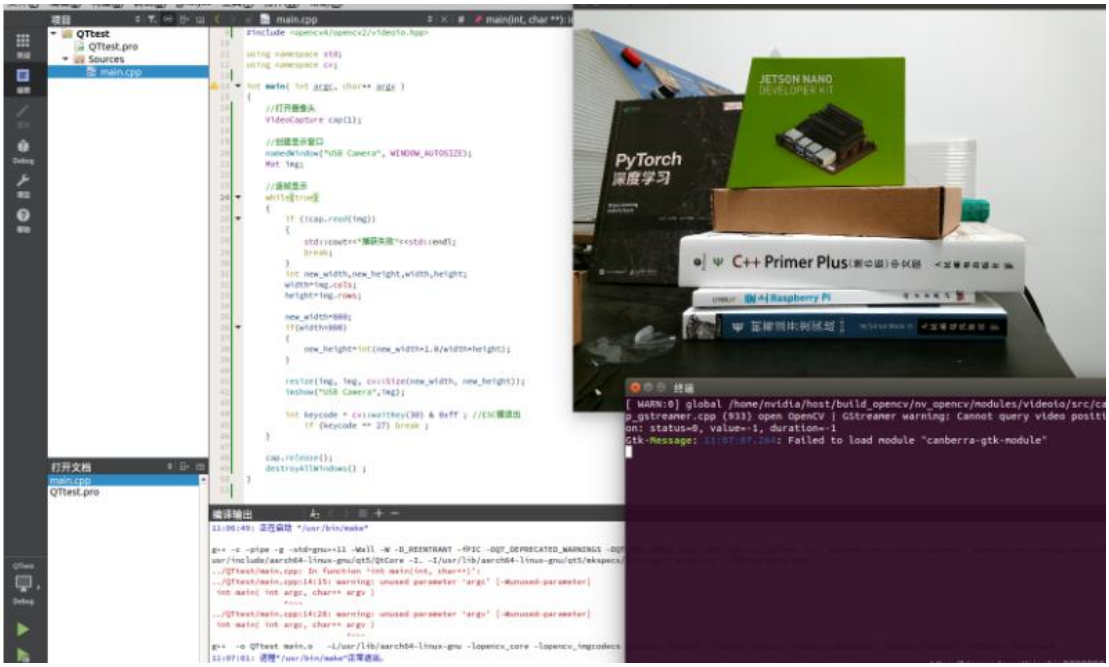
下面给出 C++版本代码，修改 main.cpp 文件，代码如下：

```
#include <iostream>
#include <string>
#include <opencv4/opencv2/opencv.hpp>
#include <opencv4/opencv2/core.hpp>
```

```
#include <opencv4/opencv2/highgui.hpp>
#include <opencv4/opencv2/imgproc.hpp>
#include <opencv4/opencv2/objdetect.hpp>
#include <opencv4/opencv2/imgproc/types_c.h>
#include <opencv4/opencv2/videoio.hpp>

using namespace std;
using namespace cv;

int main( int argc, char** argv )
{
    //打开摄像头
    VideoCapture cap(1);

    //创建显示窗口
    namedWindow("USB Camera", WINDOW_AUTOSIZE);
    Mat img;

    //逐帧显示
    while(true)
    {
        if (!cap.read(img))
        {
            std::cout<<"捕获失败"<<std::endl;
            break;
        }
        int new_width,new_height,width,height,channel;
        width=img.cols;
        height=img.rows;
        channel=img.channels();
        cout<<width<<"    "<<height<<"    "<<channel<<endl;

        new_width=800;
        if(width>800)
        {
            new_height=int(new_width*1.0/width*height);
        }

        resize(img, img, cv::Size(new_width, new_height));
        imshow("USB Camera",img);

        int keycode = cv::waitKey(30) & 0xff ; //ESC 键退出
            if (keycode == 27) break ;
    }
```

```
    cap.release();
    destroyAllWindows() ;
}
```

效果如下所示：



3.2.2 二维码检测和识读

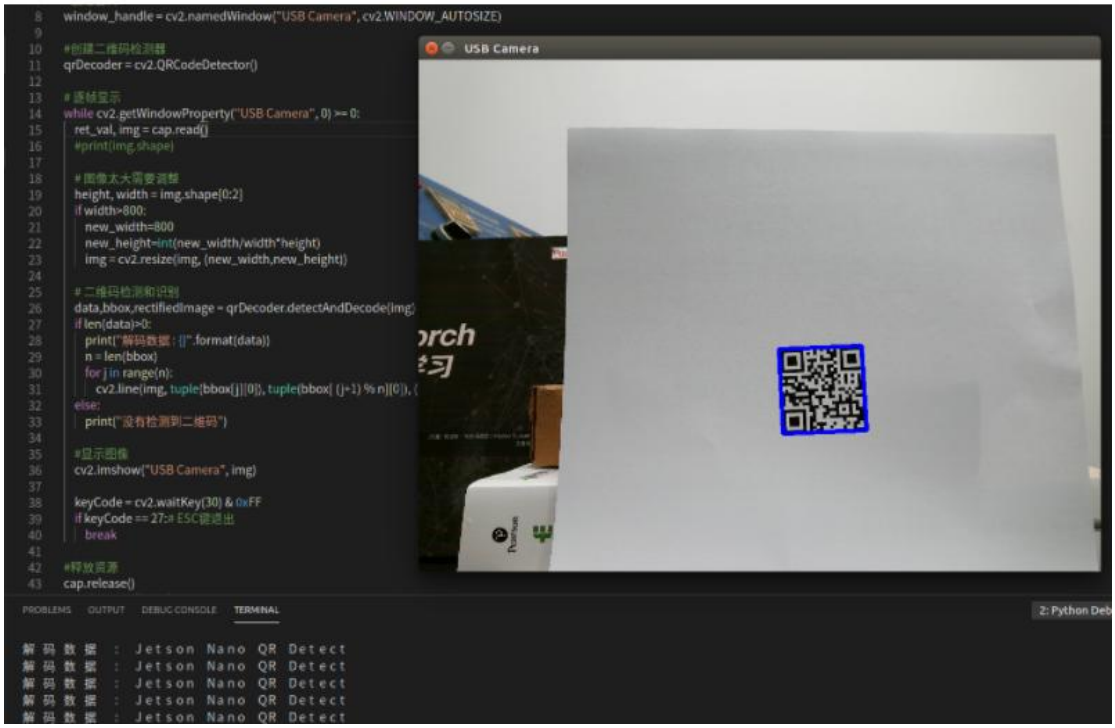
本小节将使用 Opencv 实现二维码检测和识读功能。在 opencv4.0 以后，已经集成了二维码识读模块，因此，我们可以采用最新的 opencv 来实现二维码检测和识读。二维码检测和识别主要分为 3 步：使用 QRCodeDetector()函数创建二维码检测器；使用 detectAndDecode 函数对图像进行二维码检测和识别；将检测结果输出。

这里主要是读取视频流的每帧图像然后对图像进行检测，为了方便，我们仅给出针对 USB 摄像头的完整实例，对于 CSI 摄像头可以根据 3.2.1 节内容将相关二维码检测代码迁移过去即可。结合 3.2.1 节中获取 USB 摄像头视频的代码，给出完整的 Python 版二维码检测和识读代码：

```
import cv2
```

```
import numpy as np

 #创建摄像头捕获模块
cap = cv2.VideoCapture(1)


#创建窗口
window_handle = cv2.namedWindow("USB Camera", cv2.WINDOW_AUTOSIZE)

#创建二维码检测器
qrDecoder = cv2.QRCodeDetector()

# 逐帧显示
while cv2.getWindowProperty("USB Camera", 0) >= 0:
    ret_val, img = cap.read()
    #print(img.shape)

# 图像太大需要调整
    height, width = img.shape[0:2]
    if width>800:
        new_width=800
        new_height=int(new_width/width*height)
        img = cv2.resize(img, (new_width,new_height))

# 二维码检测和识别
    data,bbox,rectifiedImage = qrDecoder.detectAndDecode(img)
    if len(data)>0:
        print("解码数据 : {}".format(data))
        n = len(bbox)
        for j in range(n):
            cv2.line(img, tuple(bbox[j][0]), tuple(bbox[ (j+1) % n][0]), (255,0,0), 3)
    else:
        print("没有检测到二维码")

#显示图像
    cv2.imshow("USB Camera", img)

    keyCode = cv2.waitKey(30) & 0xFF
    if keyCode == 27:# ESC 键退出
        break

#释放资源
cap.release()
cv2.destroyAllWindows()
```

效果图如下：



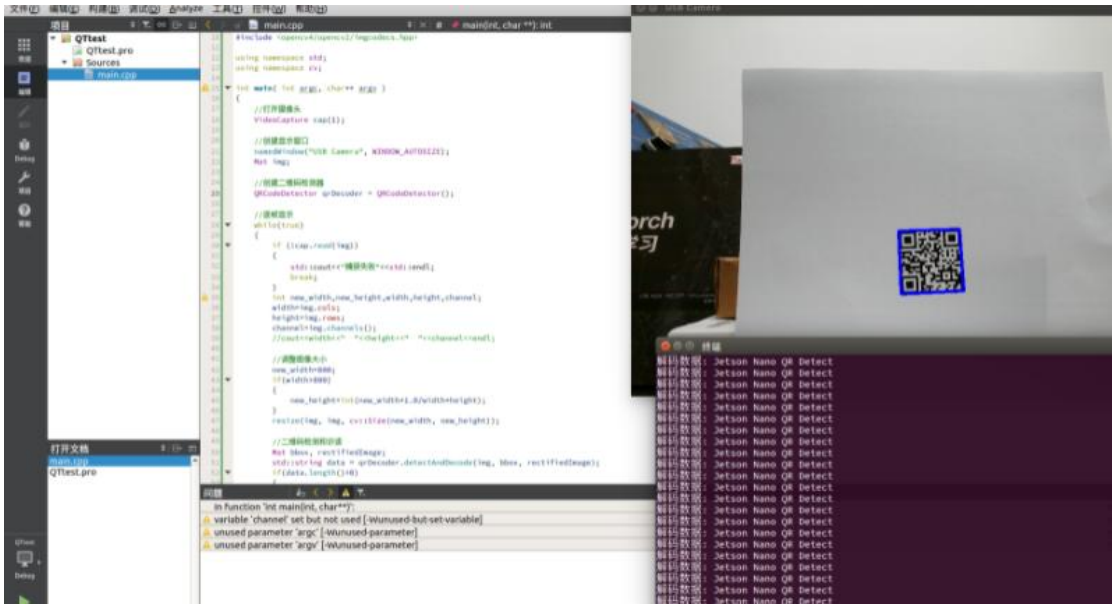
C++版本完整代码如下：

```
#include <iostream>
#include <string>
#include <opencv4/opencv2/opencv.hpp>
#include <opencv4/opencv2/core.hpp>
#include <opencv4/opencv2/highgui.hpp>
#include <opencv4/opencv2/imgproc.hpp>
#include <opencv4/opencv2/objdetect.hpp>
#include <opencv4/opencv2/imgproc/types_c.h>
#include <opencv4/opencv2/videoio.hpp>
#include <opencv4/opencv2/imgcodecs.hpp>

using namespace std;
using namespace cv;

int main( int argc, char** argv )
{
```

```
    //打开摄像头
    VideoCapture cap(1);

    //创建显示窗口
    namedWindow("USB Camera", WINDOW_AUTOSIZE);
    Mat img;

    //创建二维码检测器
    QRCodeDetector qrDecoder = QRCodeDetector();

    //逐帧显示
    while(true)
    {
        if (!cap.read(img))
        {
            std::cout<<"捕获失败"<<std::endl;
            break;
        }
        int new_width,new_height,width,height,channel;
        width=img.cols;
        height=img.rows;
        channel=img.channels();
        //cout<<width<<"    "<<height<<"    "<<channel<<endl;

        //调整图像大小
        new_width=800;
        if(width>800)
        {
            new_height=int(new_width*1.0/width*height);
        }
        resize(img, img, cv::Size(new_width, new_height));

        //二维码检测和识读
        Mat bbox, rectifiedImage;
        std::string data = qrDecoder.detectAndDecode(img, bbox, rectifiedImage);
        if(data.length()>0)
        {
            cout << "解码数据: " << data << endl;

            int n = bbox.rows;
            for(int i = 0 ; i < n ; i++)
            {
                line(img, Point2i(bbox.at<float>(i,0),bbox.at<float>(i,1)),
Point2i(bbox.at<float>((i+1) % n,0), bbox.at<float>((i+1) % n,1)), Scalar(255,0,0), 3);
```

```
            }
        }
        else
            cout << "没有检测到二维码" << endl;

        imshow("USB Camera",img);

        int keycode = cv::waitKey(30) & 0xff ; //ESC 键退出
            if (keycode == 27) break ;
    }

    cap.release();
    destroyAllWindows() ;
}
```

效果如下图所示：



4. 常用框架安装

4.1 pytorch 介绍

PyTorch 是一个开源的 Python 机器学习库，基于 Torch，用于自然语言处理等应用程序。2017 年 1 月，由 Facebook 人工智能研究院（FAIR）基于 Torch 推出了 PyTorch。它是一个基于 Python 的可续计算包，提供两个高级功能：

1、具有强大的 GPU 加速的张量计算（如 NumPy）；

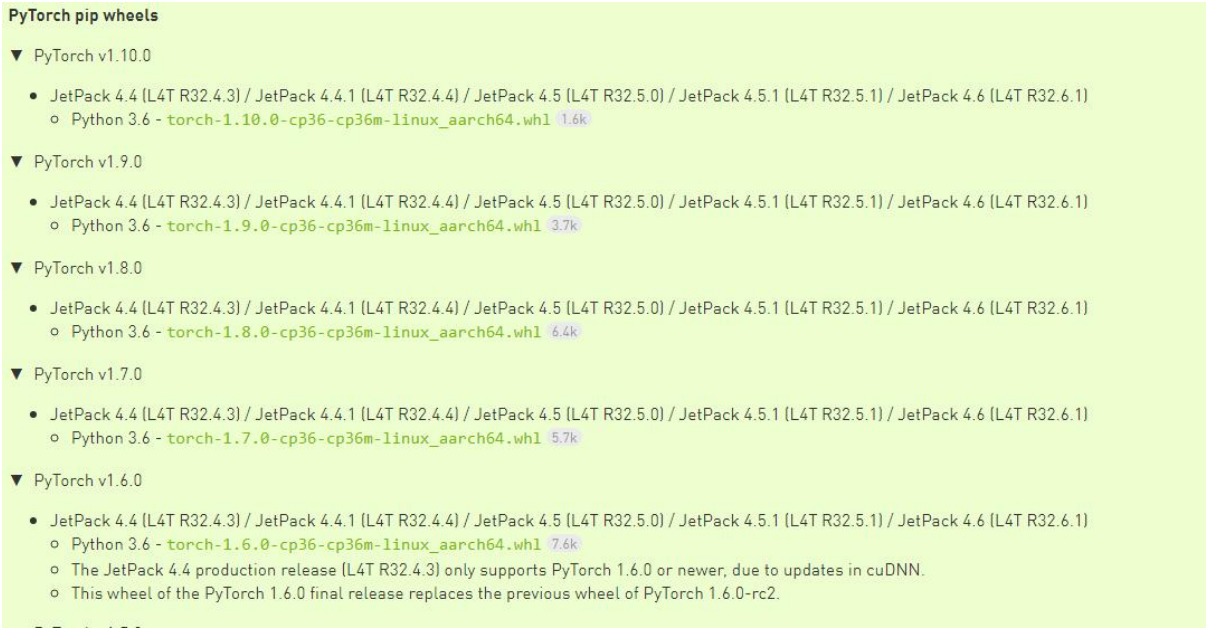
2、包含自动求导系统的深度神经网络。

发展:PyTorch 的前身是 Torch，其底层和 Torch 框架一样，但是使用 Python 重新写了很多内容，不仅更加灵活，支持动态图，而且提供了 Python 接口。它是由 Torch7 团队开发，是一个以 Python 优先的深度学习框架，不仅能够实现强大的 GPU 加速，同时还支持动态神经网络。PyTorch 既可以看作加入了 GPU 支持的 numpy，同时也可以看成一个拥有自动求导功能的强大的深度神经网络。除了 Facebook 外，它已经被 Twitter、CMU 和 Salesforce 等机构采用.

优点:PyTorch 是相当简洁且高效快速的框架，设计追求最少的封装，设计符合人类思维，它让用户尽可能地专注于实现自己的想法，与 google 的 Tensorflow 类似，FAIR 的支持足以确保 PyTorch 获得持续的开发更新，PyTorch 作者亲自维护的论坛 供用户交流和求教问题，入门简单.

4.2 安装

Pytorch 安装跟系统版本紧密联系，安装 pythoch 不同版本之前，先确定设备的 jetpack 版本号，根据下面图片来确定需要安装什么版本的 pythrch 才能正常使用.



PyTorch pip wheels

▼ PyTorch v1.10.0

- JetPack 4.4 (L4T R32.4.3) / JetPack 4.4.1 (L4T R32.4.4) / JetPack 4.5 (L4T R32.5.0) / JetPack 4.5.1 (L4T R32.5.1) / JetPack 4.6 (L4T R32.6.1)
  - Python 3.6 - torch-1.10.0-cp36-cp36m-linux_aarch64.whl 1.6k

▼ PyTorch v1.9.0

- JetPack 4.4 (L4T R32.4.3) / JetPack 4.4.1 (L4T R32.4.4) / JetPack 4.5 (L4T R32.5.0) / JetPack 4.5.1 (L4T R32.5.1) / JetPack 4.6 (L4T R32.6.1)
  - Python 3.6 - torch-1.9.0-cp36-cp36m-linux_aarch64.whl 3.7k

▼ PyTorch v1.8.0

- JetPack 4.4 (L4T R32.4.3) / JetPack 4.4.1 (L4T R32.4.4) / JetPack 4.5 (L4T R32.5.0) / JetPack 4.5.1 (L4T R32.5.1) / JetPack 4.6 (L4T R32.6.1)
  - Python 3.6 - torch-1.8.0-cp36-cp36m-linux_aarch64.whl 6.4k

▼ PyTorch v1.7.0

- JetPack 4.4 (L4T R32.4.3) / JetPack 4.4.1 (L4T R32.4.4) / JetPack 4.5 (L4T R32.5.0) / JetPack 4.5.1 (L4T R32.5.1) / JetPack 4.6 (L4T R32.6.1)
  - Python 3.6 - torch-1.7.0-cp36-cp36m-linux_aarch64.whl 5.7k

▼ PyTorch v1.6.0

- JetPack 4.4 (L4T R32.4.3) / JetPack 4.4.1 (L4T R32.4.4) / JetPack 4.5 (L4T R32.5.0) / JetPack 4.5.1 (L4T R32.5.1) / JetPack 4.6 (L4T R32.6.1)
  - Python 3.6 - torch-1.6.0-cp36-cp36m-linux_aarch64.whl 7.6k
  - The JetPack 4.4 production release (L4T R32.4.3) only supports PyTorch 1.6.0 or newer, due to updates in cuDNN.
  - This wheel of the PyTorch 1.6.0 final release replaces the previous wheel of PyTorch 1.6.0-rc2.

基于 python3.6 安装方式(以下方式为 1.8.0 为例):

wget https://nvidia.box.com/shared/static/p57jwntv436lfrd78inwl7iml6p13fzh.whl -O

torch-1.8.0-cp36-cp36m-linux_aarch64.whl

sudo apt-get install python3-pip libopenblas-base libopenmpi-dev

pip3 install Cython

pip3 install numpy torch-1.8.0-cp36-cp36m-linux_aarch64.whl

基于 python2.7 安装方式(以下方式为 1.4.0 为例):

wget https://nvidia.box.com/shared/static/1v2cc4ro6zvsbu0p8h6qcuaqco1qcsif.whl -O

torch-1.4.0-cp27-cp27mu-linux_aarch64.whl

sudo apt-get install libopenblas-base libopenmpi-dev

pip install future torch-1.4.0-cp27-cp27mu-linux_aarch64.whl

注意:PyTorch v1.4.0 for L4T R32.4.2 是最后一个支持 Python 2.7 的版本,所已建议在 python3.6 下安装.

torchvision 安装

```
sudo apt-get install libjpeg-dev zlib1g-dev libpython3-dev libavcodec-dev libavformat-dev
libswscale-dev
git clone --branch v0.x.0 https://github.com/pytorch/vision torchvision
cd torchvision
export BUILD_VERSION=0.x.0 # (填写 torchvision 安装的版本号)
python3 setup.py install
```

验证:

```
python3
>>> import torch
>>> print(torch.__version__)
>>> print('CUDA available: ' + str(torch.cuda.is_available()))
>>> print('cuDNN version: ' + str(torch.backends.cudnn.version()))
>>> a = torch.cuda.FloatTensor(2).zero_()
>>> print('Tensor a = ' + str(a))
>>> b = torch.randn(2).cuda()
>>> print('Tensor b = ' + str(b))
>>> c = a + b>>> print('Tensor c = ' + str(c))
>>> import torchvision
>>> print(torchvision.__version__)
```

Pytorch whl 包下载地址:

链接：https://pan.baidu.com/s/1J2DeI9HMt0MkH-ZgGfsFnA

提取码：wlf1

4.3 TensorFlow 介绍

TensorFlow™是一个基于数据流编程（dataflow programming）的符号数学系统，被广泛应用于各类机器学习（machine learning）算法的编程实现，其前身是谷歌的神经网络算法库 DistBelief。

Tensorflow 拥有多层级结构，可部署于各类服务器、PC 终端和网页并支持 GPU 和 TPU 高性能数值计算，被广泛应用于谷歌内部的产品开发和各领域的科学研究。

TensorFlow 由谷歌人工智能团队谷歌大脑（Google Brain）开发和维护，拥有包括 TensorFlow Hub、TensorFlow Lite、TensorFlow Research Cloud 在内的多个项目以及各类应用程序接口（Application Programming Interface, API)。自 2015 年 11 月 9 日起，TensorFlow 依据阿帕奇授权协议（Apache 2.0 open source license）开放源代码.

TensorFlow 是世界上最受欢迎的开源机器学习框架，它具有快速、灵活并适合产品级大规模应用等特点，让每个开发者和研究者都能方便地使用人工智能来解决多样化的挑战。

4.4 安装

以下是 Xaviernano 的官方 TensorFlow 版本信息。

Python 3.6+JetPack4.5

sudo apt-get install libhdf5-serial-dev hdf5-tools libhdf5-dev zlib1g-dev zip libjpeg8-dev

liblapack-dev libblas-dev gfortran

sudo apt-get install python3-pip

sudo pip3 install -U pip testresources setuptools==49.6.0

sudo pip3 install -U numpy==1.16.1 future==0.18.2 mock==3.0.5 h5py==2.10.0

keras_preprocessing==1.1.1 keras_applications==1.0.8 gast==0.2.2 futures protobuf pybind11#

TF-2.x

sudo pip3 install --pre --extra-index-url

https://developer.download.nvidia.com/compute/redist/jp/v45 tensorflow# TF-1.15

sudo pip3 install --pre --extra-index-url

https://developer.download.nvidia.com/compute/redist/jp/v45 'tensorflow<2'

Python 3.6+JetPack4.4

sudo apt-get update

sudo apt-get install libhdf5-serial-dev hdf5-tools libhdf5-dev zlib1g-dev zip libjpeg8-dev

liblapack-dev libblas-dev gfortran

sudo apt-get install python3-pip

sudo pip3 install -U pip testresources setuptools

sudo pip3 install -U numpy==1.16.1 future==0.17.1 mock==3.0.5 h5py==2.9.0

keras_preprocessing==1.0.5 keras_applications==1.0.8 gast==0.2.2 futures protobuf pybind11

TF-2.x

sudo pip3 install --pre --extra-index-url

https://developer.download.nvidia.com/compute/redist/jp/v44 tensorflow==2.3.1+nv20.12

TF-1.15

sudo pip3 install --pre --extra-index-url

https://developer.download.nvidia.com/compute/redist/jp/v44 ‘tensorflow<2’

TensorFlow 安装包下载地址:

链接：https://pan.baidu.com/s/1gLC-lZ5pC4OASezrpfY2Rw

提取码：6iyv

图为信息科技（深圳）有限公司保修条例

重要提示

图为信息科技（深圳）有限公司保证提供的每个嵌入式产品，就其所知在材料与工艺上均无任何缺陷，完全符合原厂正式发货之规格。

图为信息科技（深圳）有限公司保修范围包括全部原厂产品，由经销商配置的配件出现故障时请与经销商协商解决。图为科技提供的所有产品的保修期限均为一年（超出保修期限的提供终身维修服务），保修期限的起始时间自出厂之日起开始计算，对于保修期内维修好的产品，维修部分延长质保 12 个月。除非图为科技另行通知，否则您的原厂发货单日期即为出厂日期。

如何获得保修服务

如果您在保修期内产品不能正常运行，请与图为科技或经销商联系以获得保修服务，产品保修时请出示购货发票证明（这是您获得保修服务的权利证明）。

保修解决措施

当您要求保修服务时，您需要遵循图为科技规定的问题确定和解决程序。您需要接受技术人员通过电话或以电子邮件方式与您进行首次诊断，届时需要您配合详细填写我们所提供的报修单上所有问题，以确保我们准确判断故障原因及造成损毁位置（过保产品我们还会提供收费单，需要您确认）。图为科技有权对所报修产品进行“维修”或“更换”，如果 产品被“更换”或“维修”，被更换的“故障”产品或修理后更换后的“故障”零件将被返回图为科技。 因部分维修产品需发往原厂，为避免意外损失，图为科技提请您购买运输保险，如果用户放弃保险，那么所寄物品在运输途中损坏或遗失，图为科技不承担责任。 对于保修期限内的产品，用户承担维修产品返回厂家时的运费，图为科技承担维修后的产品返还用户的运费。

以下情况不在保修之列

1、 产品的不适当安装、使用不当、误用、滥用（如超出工作负荷等）

2、 不当的维护保管（如火灾、爆炸等）或自然灾害（如雷电、地震、台风等）所致产品故障或 陨坏。

3、 对产品的改动（如电路特性、机械特性、软件特性、三防处理等）。

4、 其它显然是由于使用不当造成的故障（如电压过高、电压过低、浮地电压过高、极性接反、 针脚弯曲或折断、接错总线、器件脱落、静电击穿、外力挤压、坠落受陨、温度过高、湿度 过大、运输不良等）。

5、 产品上的标志和部件号曾被删改或去除。

6、 产品超过保修期。

特别说明：

如多个产品出现同一故障或多次在同一设备出现相同故障或损坏时，为查找原因以确认责任。我司有权要求使用者提供周边设备实物或技术资料，例如：监视器，I/O 设备，电缆，电源，连接示意图，系统结构图等。否则，我们有权拒绝履行保修，维修时将按照市场价格收取费用，并收取维修保证金。