疲劳驾驶检测系统

部署手册

一、 Jetson Nano 硬件外观

1.1 硬件规格

二、 系统安装

2.1 硬件材料准备

1) **电源线**：5V⎓2A 的 Micro-USB 电源线或者 5V=4A 电源，建议使用 5V=4A 电源，以避免供电不足导致系统运行问题，需要短接跳线帽。

2) **SD 卡**：Jetson Nano 要求最低配置 16G 的 SD 卡，推荐使用 64G 的 SD 卡。

3) **HDMI 线**：显示器需要支持 HDMI 输入

4) **网线**

2.2 系统下载

2.2.1 下载 JetPack SDK

GPU	NVIDIA Maxwell™ 架构，128 个 NVIDIA CUDA® 核心 0.5 TFLOPS (FP16)
CPU	Quad-core ARM® Cortex®-A57 处理器 @ 1.43 GHz
内存	4 GB 64 位 LPDDR4-1600 MHz–25.6 GB/s
存储	64 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	2x MIPI CSI-2 DPHY lanes
联网	千兆以太网，M.2 Key E 接口外扩 (可外接:AC8265 双模网卡)
显示	HDMI 2.0 或 DP1.2
USB	4x USB 3.0，USB 2.0 Micro-B
扩展接口	GPIO，I2C，I2S，SPI，UART
大小	69.6 mm x 45 mm
规格尺寸	260 引脚边缘连接器

下载 JetPack 4.4 - L4T 32.4.3 默认下载最新的，注意选择版本我下载的是4.4.1

JetPack 4.4 - L4T 32.4.3 包含以下深度神经网络库支持：

- TensorRT-7.1.3
- cuDNN-8.0
- CUDA-10.2

2.2.2 格式化 SD 卡

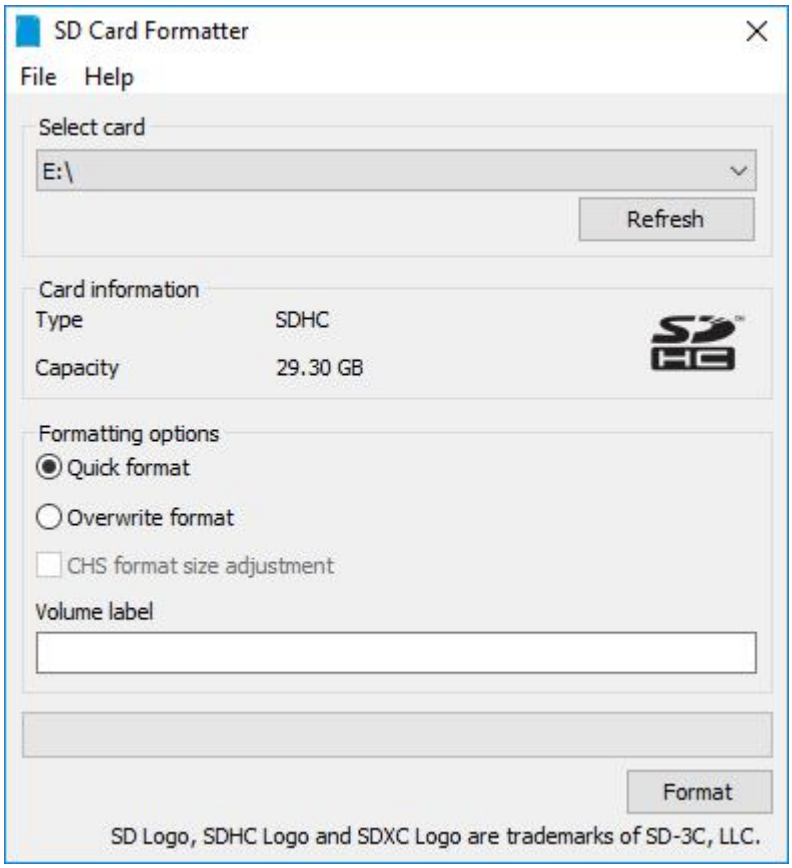
1、插入 SD 卡至计算机

2、下载安装并启动 SD Memory Card Formatter for Windows.

3、选择 SD 卡

4、选择 “Quick format”

5、点击 “Format” 开始格式化。



2.2.3 安装系统到 SD 卡

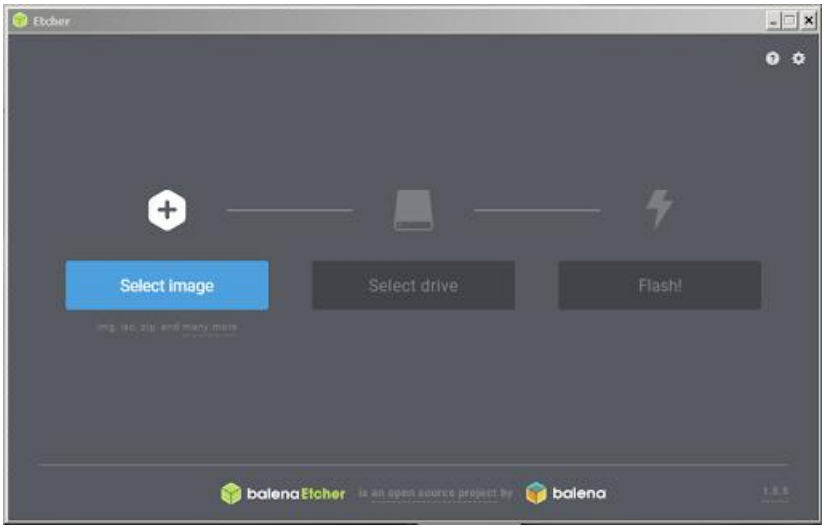
使用 Etcher 软件写入 JetPack 4.4 到 microSD 卡

1. 下载安装并启动 Etcher 软件

2. 点击“Select image”选择已经解压好的 JetPack 4.4

3. 点击“Select drive”选择正确的 microSD 卡

4. 点击“Flash!” 大约需要十分钟写入镜像，写入之后会自动验证是否写入成功



2.2.4 安装电源外设启动主机

将写入好镜像文件的 SD 卡插入 Jetson Nano Developer Kit ，连接电源，显示器，键盘与鼠标，网线。

1、打开电源后，Micro-USB 连接器旁边的绿色 LED 就会点亮。首次启动时，Jetson Nano

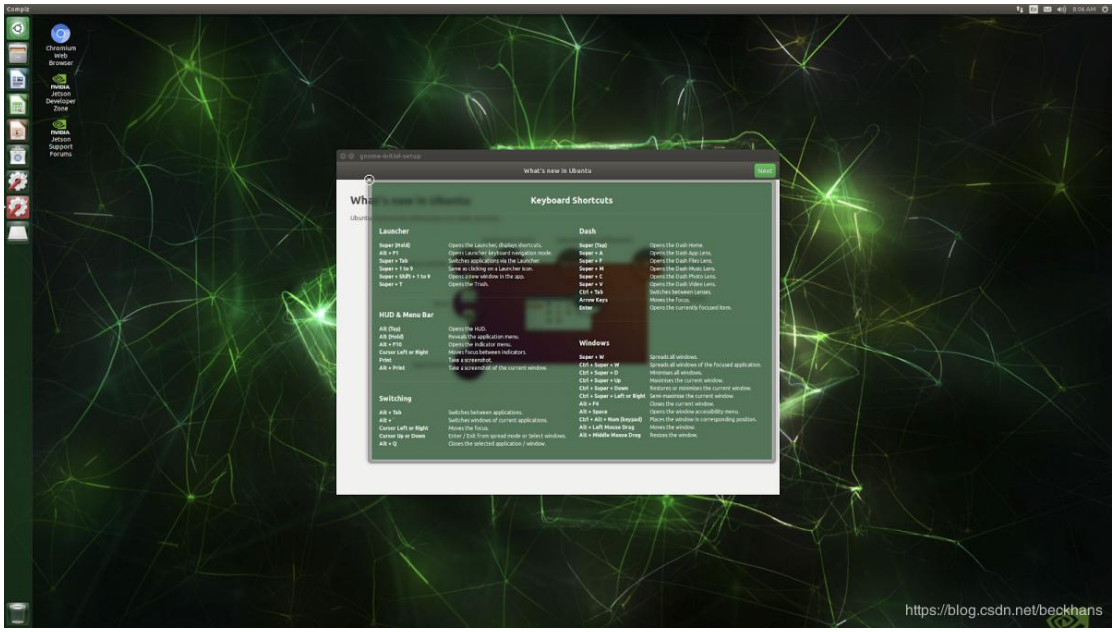
Developer Kit 将带您进行一些初始设置，包括：

2、 查看并接受 NVIDIA Jetson 软件 EULA

3、 选择系统语言，键盘布局和时区

4、 创建用户名，密码和计算机名

5、 登录



三、 环境配置

3.1 检查系统文件更新

sudo apt-get update 网络最好接网线

sudo apt-get full-upgrade

3.2 检查 CUDA

Jetson-nano 中已经安装了 CUDA10.2 版本，但是此时你如果运行 nvcc -V 是不会成功的，需要你把 CUDA 的路径写入环境变量中。

OS 中自带 Vim 工具 ，所以运行下面的命令编辑环境变量：

sudo vim ~/.bashrc

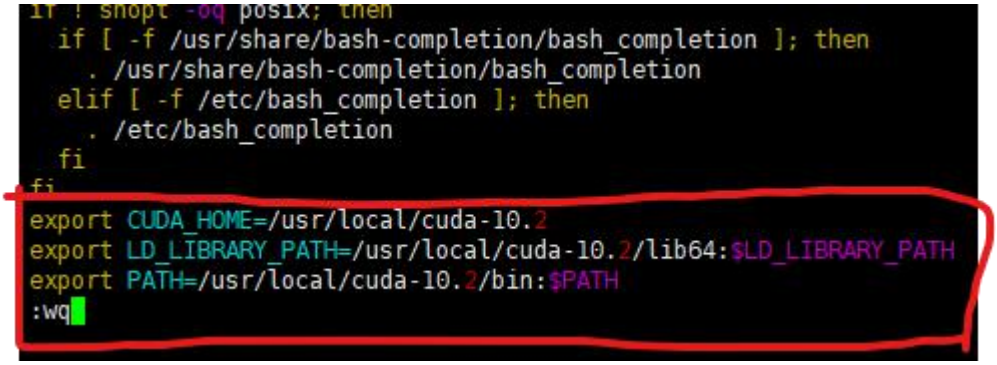
在最后添加

export CUDA_HOME=/usr/local/cuda-10.2

export LD_LIBRARY_PATH=/usr/local/cuda-10.2/lib64:$LD_LIBRARY_PATH 64后面不要填

export PATH=/usr/local/cuda-10.2/bin:$PATH

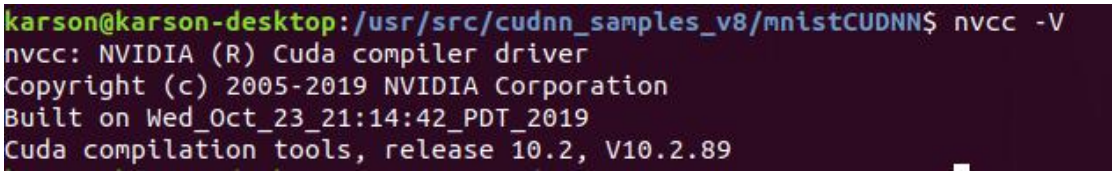
保存退出 保存退出执行：按 Esc 后，输 “：wq”回车



source 更新一下环境变量使刚才添加生成

source ~/.bashrc

source 后，此时再执行 nvcc -V 执行结果如下

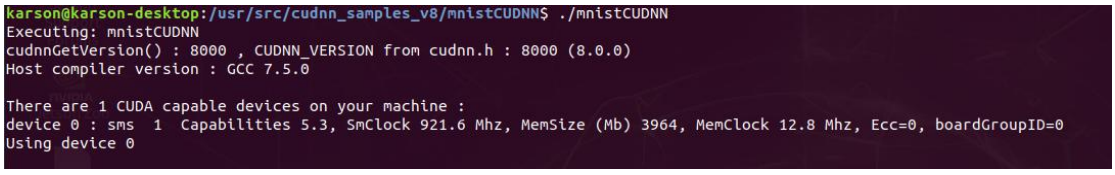


3.3 检查 CUDNN

cd /usr/src/cudnn_samples_v8/mnistCUDNN #进入例子目录

sudo make #编译一下例子

./mnistCUDNN



3.4 安装 PyTorch

下载 nvida 官方编译好的 Pytorch

Python 3.6 - torch-1.6.0rc2-cp36-cp36m-linux_aarch64.whl 这个连接是 PyTorch

1.6.0-rc2。JetPack 4.4 -L4T R32.4.3 由于升级 cuDNN10.2 原因，只支持 PyTorch 1.6.0 或者更新版本

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

pip3 install Cython

pip3 install numpy torch-1.4.0-cp36-cp36m-linux_aarch64.whl

```
skl@karson-desktop:~$ pip3 install torch-1.6.0rc2-cp36-cp36m-linux_aarch64.whl
Processing ./torch-1.6.0rc2-cp36-cp36m-linux_aarch64.whl
Collecting numpy (from torch==1.6.0rc2)
```

执行：python3 打印 python 版本

```
skl@karson-desktop:~$ python3
Python 3.6.9 (default, Jul 17 2020, 12:50:27)
[GCC 8.4.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> import torch
>>> print(torch.__version__)
1.6.0
>>> exit()
skl@karson-desktop:~$
```

3.5 安装对应版本 torchvision

PyTorch v1.0 - torchvision v0.2.2

PyTorch v1.1 - torchvision v0.3.0

PyTorch v1.2 - torchvision v0.4.0

PyTorch v1.3 - torchvision v0.4.2

PyTorch v1.4 - torchvision v0.5.0

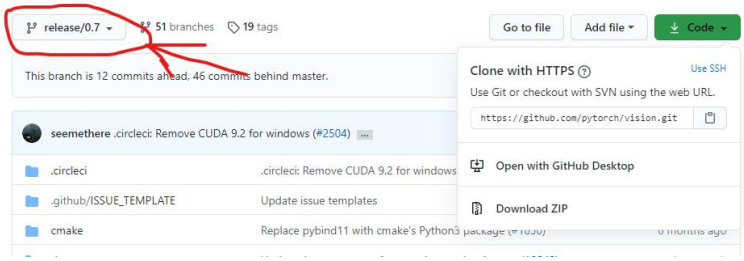
PyTorch v1.5 - torchvision v0.6.0

PyTorch v1.6 - torchvision v0.7.0-rc2

首先安装依赖包

sudo apt-get install libjpeg-dev zlib1g-dev　直接装装不上，可以先pip3 install torchvision 也可以不装因为后面用不到

使用迅雷下载 https://github.com/pytorch/vision 分支使用 release 0.7



解压后 cd torchvision，进入目录执行:sudo python3 setup.py install

检查是否安装正确

Python3

import torchvision

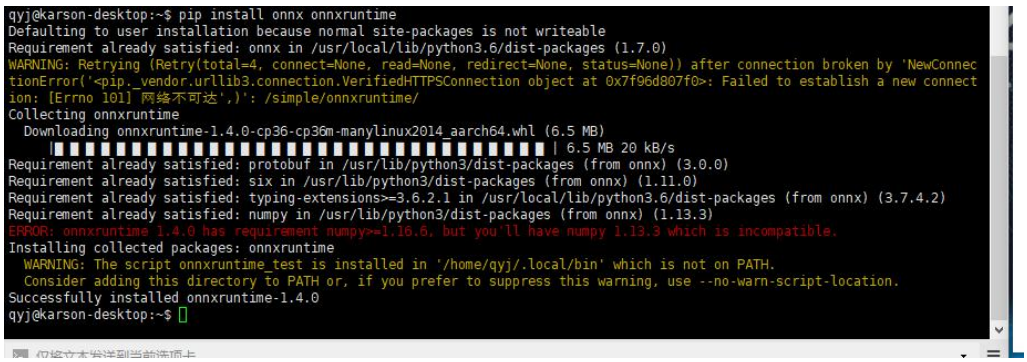
print(torchvision.__version__)

3.6 安装 onnx，onnxruntime

安装onnx之前要先把这些包下好，还要安装一些其他的包

```
pip3 install --upgrade pip
sudo apt-get install protobuf-compiler libprotoc-dev
```

sudo pip3 install onnx onnxruntime



检查是否安装正确

python3

import onnx

import onnxruntime

print(onnx.__version__)

print(onnxruntime.__version__)

```
还要装个pycuda
pip3 install pycuda --user -i
https://pypi.tuna.tsinghua.edu.cn/simple
注意pip3前面不要加sudo!
```

4.1 ONNX:

Open Neural Network Exchange（ONNX，开放神经网络交换）格式，是一个用于表示深度学习模型的标准，可使模型在不同框架之间进行转移。ONNX 是一种针对机器学习所设计的开放式的文件格式，用于存储训练好的模型。它使得不同的人工智能框架（如 Pytorch, MXNet）可以采用相同格式存储模型数据并交互。目前官方支持加载 ONNX 模型并进行推理的深度学习框架有： Caffe2, PyTorch, MXNet，ML.NET，TensorRT 和 Microsoft CNTK，并且 TensorFlow 也非官方的支持 ONNX。

4.2 TensorRT:

TensorRT 是英伟达公司出品的高性能的推断 C++库，专门应用于边缘设备的推断，TensorRT 可以将我们训练好的模型分解再进行融合，融合后的模型具有高度的集合度。例如卷积层和激活层进行融合后，计算速度可以就进行提升。当然，TensorRT 远远不止这个：我们平时所见到了深度学习落地技术：模型量化、动态内存优化以及其他的一些优化技术 TensorRT 都已经有实现，更主要的，其推断代码是直接利用 cuda 语言在显卡上运行的，所有的代码库仅仅包括 C++和 cuda，当然也有 python 的包装,我们在利用这个优化库运行我们训练好的代码后，运行速度和所占内存的大小都会大大缩减。

4.3 项目部署:

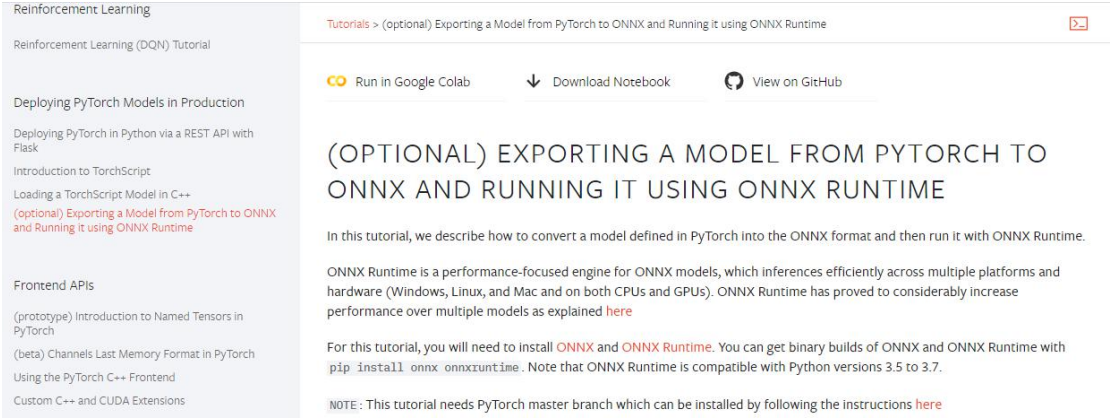
4.3.1 将网络权重导出成 onnx

将网络输出的权重 pth 文件转换成 onnx，利用 pytorch 自带的 onnx 库将权重文件转换成 onnx。

1. 安装 onnx 库，执行：pip install onnx

2. 参考 pytorch 官网 onnx 部署文档，链接：

https://pytorch.org/tutorials/advanced/super_resolution_with_onnxruntime.html



主要步骤：1.创建模型。2.加载训练 pth 权重。3.调用 pytorch.onnx 将权重导出成 onnx

实例代码：

```python
import numpy as np
from PIL import Image
import torch.onnx
from nets.yolo4_tiny import YoloBody
# 创建模型
model = YoloBody(len(anchors[0]), num_classes).eval()

# # 加载权重
model_path = './Epoch1-Total_Loss3.9009-Val_Loss5.2476.pth'
state_dict = torch.load(model_path)
model.load_state_dict(state_dict)

# # 打开图片
img = Image.open(r"F:\alldata/380.jpg")
# # 图片预处理
image_shape = np.array(np.shape(img)[0:2])
crop_img = np.array(letterbox_image(img, (model_image_size[0], model_image_size[1])))
photo = np.array(crop_img, dtype=np.float32)/255.0
photo = np.transpose(photo, (2, 0, 1))
photo = photo.astype(np.float32)
images = []
images.append(photo)
images = np.asarray(images)
images = torch.from_numpy(images)
images = images
outputs = model(images)

torch.onnx.export(model,  # model being run
                  images,  # model input (or a tuple for multiple inputs)
                  "yolov4_tiny.onnx",  # where to save the model (can be a file or file-like object)
                  export_params=True,  # store the trained parameter weights inside the model file
                  opset_version=11,  # the ONNX version to export the model to
                  do_constant_folding=True,  # whether to execute constant folding for optimization
                  input_names=['input'],  # the model's input names
                  output_names=['output'],  # the model's output names
                  dynamic_axes={'input': {0: 'batch_size'},  # variable lenght axes
                                'output': {0: 'batch_size'}}
                  )
```

4.3.2 将 onnx 转换成 TensorRT 引擎

转换方式有两种，使用 C++接口将 onnx 转换成 TensorRT 引擎并且侦测。或使用 Python 接口将 onnx 转换成 TensorRT 引擎。两种方式的区别在于 C++项目安全方面比较有优势,而 python 在数据预处理和后处理时比较方便。Nvidia 系列的板卡都自带 TensorRT，本次部署在 jetson_nano 板卡上。参考 TensorRT 官方部署文档(本次项目使用 python API)网址链接：https://docs.nvidia.com/deeplearning/tensorrt/developer-guide/index.html#python_topics

示例代码：参考 https://github.com/SeanAvery/yolov5-tensorrt.git

```python
import tensorrt as trt

EXPLICIT_BATCH = []
if trt.__version__[0] >= '7':
    EXPLICIT_BATCH.append(
        1 << (int)(trt.NetworkDefinitionCreationFlag.EXPLICIT_BATCH))

def build_engine(onnx_file_path, engine_file_path, verbose=False):
    """Takes an ONNX file and creates a TensorRT engine."""
    TRT_LOGGER = trt.Logger(trt.Logger.VERBOSE) if verbose else trt.Logger()
    with trt.Builder(TRT_LOGGER) as builder, builder.create_network(*EXPLICIT_BATCH) as network, trt.OnnxParser(network,
                                                                                                                 TRT_LOGGER) as parser:
        builder.max_workspace_size = 1 << 27  # 25#33554432#32M
        builder.max_batch_size = 4
        builder.fp16_mode = True
        # Parse model file
        print('Loading ONNX file from path {}...'.format(onnx_file_path))
        with open(onnx_file_path, 'rb') as model:
            print('Beginning ONNX file parsing')
            if not parser.parse(model.read()):
                print('ERROR: Failed to parse the ONNX file.')
                for error in range(parser.num_errors):
                    print(parser.get_error(error))
                return None
        if trt.__version__[0] >= '7':
            # The actual yolo*.onnx is generated with batch size 64.
            # Reshape input to batch size 1
            shape = list(network.get_input(0).shape)
            shape[0] = 1
            network.get_input(0).shape = shape
        print('Completed parsing of ONNX file')
        engine = builder.build_cuda_engine(network)
        print('Completed creating engine')
        with open(engine_file_path, 'wb') as f:
            f.write(engine.serialize())
        return engine

if __name__ == '__main__':
    onnx_file_path = 'yolov4-tiny.onnx'
    if not os.path.isfile(onnx_file_path):
        raise SystemExit(
            'ERROR: file (%s) not found!  You might want to run yolo_to_onnx.py first to generate it.' % onnx_file_path)
    engine_file_path = 'yolov4-tiny.engine'
    engine = build_engine(onnx_file_path, engine_file_path, False)
```

4.3.3 调用 tensorRT 引擎文件进行侦测

还有个pycuda要装

使用转换成的引擎文件根据网络的输出格式编写侦测业务。