Jetson nano 部署 YoloV8

1 前言

需要的物品：

- Jetson nano 板子
- SD 卡、读取器
- 无线网卡
- 电源

注意事项：

- python3
- pip3

2 准备

2.1 烧录系统

下载 Jetson nano 系统镜像：https://pan.baidu.com/s/178A568iL4usDGrbkvxckzA?pwd=nano (提取码: nano)。根据 Jetson nano 型号选择 4GB 或者 2GB，我这里选择的是 4GB，JetPack4.6.1，对应压缩包 jetson-nano-jp461-sd-card-image.zip，下载完成后，进行解压，使用 balenaEtcher 软件将解压后的文件烧录到 SD 卡中

调整板子上的跳线帽（请注意，板子上原本的跳线帽只插了一个针脚，如果你用 5V4A 的 DC 电源，需要把跳线帽轻轻拔起，把两个针脚都插上，才可以正常使用 DC 电源哟！

将 SD 卡插入到 Jetson nano 板子上，连接好鼠标、键盘、无线网卡和显示器（板子上有一个绿色指示灯）

2.2 常用命令

查看用户名：

```
whoami
```

查看ip地址：

```
ifconfig
```

切换高低功率：Jetson nano 有两种供电方式，10W和5W

```
sudo nvpmodel -q  # 查看当前是那个模式

sudo nvpmodel -m 1 # 将当前模式切换到5W模式，将会自动关掉两个cpu,只使用cpu1,2
sudo nvpmodel -m 0 # 切换到高功率模式
```

两种模式，0 是高功率10w，1是低功率5w，默认状态是高功率。

```
sudo jetson_clocks  # 固定 CPU 频率
```

Jetson nano 有两种常用供电方式，一种是 5V 2.5A(12.5W) 的 microUSB 供电；但如果你有很多外设在（如键盘、鼠标、wifi、显示器等）在使用，最好用 5V 4A(20W) 的供电方式，来保证 Jetson nano 的正常工作。

2.3 移除无用软件

移除 libreoffice 会为系统省很多空间，这个软件对做深度学习和计算机视觉算法也没有太多用

```
sudo apt-get purge libreoffice*
sudo apt-get clean
```

2.4 更换国内安装源

备份原先 source.list

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

修改 source.list

```
sudo vim  /etc/apt/sources.list
```

用以下内容替换原内容

```
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
```

更新软件列表

```
sudo apt-get update
```

打开 ./bashrc，配置 cuda 的环境变量

```
vim ~/.bashrc
```

向 ./bashrc 加入以下内容

```
export PATH=/usr/local/cuda/bin${PATH:+:${PATH}}
export LD_LIBRARY_PATH=/usr/local/cuda/lib64${LD_LIBRARY_PATH:+:${LD_LIBRARY_PATH}}
export CUDA_ROOT=/usr/local/cuda
```

使修改生效

```
source ~/.bashrc
```

查看cuda版本

```
nvcc -V
```

2.5 配置 pip3

了解 Jetson nano 平台自带的 Python 版本

```
ls /usr/bin/python*
```

安装 pip3

```
sudo apt update
sudo apt install python3-pip
```

查看 pip 版本及安装路径

```
pip3 --version
# pip 9.0.1 from /usr/Lub/python3/dist-packages (python 3.6)
```

版本太低，更新版本

```
python3 -m pip install --upgrade pip setuptools wheel
```

2.6 安装工具

安装 jtop，查看 CPU 和 GPU 资源

```
sudo pip3 install -U jetson-stats
```

使用 jtop

```
sudo jtop
```

如果不能使用，请重启，或者自行百度

3 模型部署

以下内容参考：https://blog.csdn.net/qq_40672115/article/details/129640372

流程：`xx.pt` -> `xx.onnx` -> `xx.engine`

其中 `xx.pt` -> `xx.onnx` 在电脑上完成，`xx.onnx` -> `xx.engine` 在板子上实现

3.1 导出 xxx.onnx 文件 (本地电脑)

下载 YoloV8 代码

```
git clone https://github.com/ultralytics/ultralytics.git
```

下载将 `.onnx` 转换为 `xx.engine` 代码

```
git clone https://github.com/shouxieai/infer.git
```

在 ultralytics 中创建一个 weights 文件夹，将训练好的模型文件 `best.pt` 放进去

在 ultralytics 中创建 `detect.py` 文件，检测模型的正确性

```python
# @time   : 2023/7/12 10:23
# @author : Xiang Lei
# opencv自己有一个缓存，每次会顺序从自己的缓存中读取，而不是直接读取最新帧
# 单独用一个线程实时捕获视频帧，主线程在需要时从子线程拷贝最近的帧使用即可

import time
import cv2
import threading
import numpy as np
from copy import deepcopy

from ultralytics import YOLO

thread_lock = threading.Lock()  # 线程锁
thread_exit = False  # 线程退出标志


class MyThread(threading.Thread):
    def __init__(self, camera_id, img_height, img_width):
        super().__init__()
        self.camera_id = camera_id
        self.img_height = img_height
        self.img_width = img_width
        self.frame = np.zeros((img_height, img_width, 3), dtype=np.uint8)

    def get_frame(self):
        return deepcopy(self.frame)

    def run(self):
        global thread_exit
        cap = cv2.VideoCapture(self.camera_id)
        while not thread_exit:
            ret, frame = cap.read()
            if ret:
                frame = cv2.resize(frame, (self.img_width, self.img_height))
                thread_lock.acquire()
                self.frame = frame
                thread_lock.release()
            else:
                thread_exit = True
```

```
        cap.release()


def main():
    global thread_exit
    model = YOLO('weights/best.pt')
    video_file = "try.mp4"

    # model = YOLO("yolov8n.pt")
    # video_file = 0
    camera_id = video_file
    img_height = 640
    img_width = 640

    thread = MyThread(camera_id, img_height, img_width)  # 创建线程
    thread.start()  # 启动线程

    while not thread_exit:
        thread_lock.acquire()
        frame = thread.get_frame()
        thread_lock.release()

        results = model(frame)
        annotated_frame = results[0].plot()

        cv2.imshow("frame", annotated_frame)
        if cv2.waitKey(1) & 0xFF == ord("q"):
            thread_exit = True


if __name__ == "__main__":
    main()

```

在 ultralytics 中创建 `export.py` 文件，将 `best.py` 转换成 `best.onnx`

```
# @time   : 2023/7/13 20:38
# @author : Xiang Lei

from ultralytics import YOLO

model = YOLO("weights/best.pt")

success = model.export(format="onnx", batch=1)
```

将 `best.onnx` 放到 `infer/workspace` 中

模型需要完成修改才能正确被infer框架使用，正常模型导出的输出为[1,6,8400]，其中1代表batch，6分别代表cx,cy,w,h,以及have_mask、no_mask两个类别分数，8400代表框的个数。首先infer框架的输出只支持[1,8400,6]这种形式的输出，因此我们需要在原始onnx的输出之前添加一个Transpose节点，infer仓库workspace/v8trans.py就是帮我们做这么一件事情，v8trans.py具体内容如下：

```
import onnx
import onnx.helper as helper
import sys
import os

def main():

    if len(sys.argv) < 2:
        print("Usage:\n python v8trans.py yolov8n.onnx")
        return 1

    file = sys.argv[1]
    if not os.path.exists(file):
        print(f"Not exist path: {file}")
        return 1

    prefix, suffix = os.path.splitext(file)
    dst = prefix + ".transd" + suffix

    model = onnx.load(file)
    node  = model.graph.node[-1]

    old_output = node.output[0]
    node.output[0] = "pre_transpose"

    for specout in model.graph.output:
        if specout.name == old_output:
            shape0 = specout.type.tensor_type.shape.dim[0]
            shape1 = specout.type.tensor_type.shape.dim[1]
            shape2 = specout.type.tensor_type.shape.dim[2]
            new_out = helper.make_tensor_value_info(
                specout.name,
                specout.type.tensor_type.elem_type,
                [0, 0, 0]
            )
```

```
 new_out.type.tensor_type.shape.dim[0].CopyFrom(shape0)

 new_out.type.tensor_type.shape.dim[2].CopyFrom(shape1)

 new_out.type.tensor_type.shape.dim[1].CopyFrom(shape2)
            specout.CopyFrom(new_out)

    model.graph.node.append(
        helper.make_node("Transpose", ["pre_transpose"], [old_output], perm=[0, 2, 1])
    )

    print(f"Model save to {dst}")
    onnx.save(model, dst)
    return 0

if __name__ == "__main__":
    sys.exit(main())
```

在命令行终端输入如下指令即可添加Transpose节点，执行完成之后在当前目录下生成 `best.transd.onnx` 模型，该模型添加了Transpose节点。

```
python3 v8trans.py best.onnx
```

（将ONNX模型 `best.transd.onnx` 放入到infer/workspace文件夹下）

将 `infer` 文件夹的所有内容放到 `jetson nano` 板子上

3.2 配置 trtexec 环境变量 (Jetson nano)

打开bashrc文件

```
vim ~/.bashrc
```

按 i 进入输入模式，在最后一行添加如下语句

```
export PATH=/usr/src/tensorrt/bin:$PATH
```

3.按下 esc 键，输入 `:wq!` 保存退出即可，最后刷新下环境变量

```
source ~/.bashrc
```

3.3 导出 `xxx.engine` 文件 (Jetson nano)

在 infer 目录执行

```
trtexec --onnx=workspace/best.transd.onnx --
saveEngine=workspace/best.transd.engine
```

开始编译，需要一段时间

3.4 修改推理代码 (Jetson nano)

为了使用上面得到的 `best.transd.engine` ，需要对部分源码进行修改

yolo 模型的推理代码主要在 src/main.cpp 文件中，需要推理的图片放在 workspace/inference 文件夹中，源码修改较简单主要有以下几点：

1.main.cpp 134，135 行注释，只进行单张图片的推理

2.main.cpp 104 行 修改加载的模型为 best_transd.sim.engine 且类型为 V8

3.main.cpp 10 行 新增 mylabels 数组，添加自训练模型的类别名称

4.mian.cpp 115 行 cocolabels 修改为 mylabels

具体修改如下

```
int main() {
  // perf();                    //修改1 134 135行注释
  // batch_inference();
  single_inference();
  return 0;
}

auto yolo = yolo::load("best_transd.engine", yolo::Type::V8);
//  修改2

static const char *mylabels[] = {"box", "cola"};     // 修改3
新增mylabels数组

auto name = mylabels[obj.class_label]        // 修改4
cocolabels修改为mylabels
```

3.5 编译运行

编译用到的 Makefile 文件需要修改，修改后的 Makefile 文件如下

```
cc        := g++
nvcc      = /usr/local/cuda-10.2/bin/nvcc

cpp_srcs  := $(shell find src -name "*.cpp")
cpp_objs  := $(cpp_srcs:.cpp=.cpp.o)
cpp_objs  := $(cpp_objs:src/%=objs/%)
cpp_mk    := $(cpp_objs:.cpp.o=.cpp.mk)

cu_srcs   := $(shell find src -name "*.cu")
cu_objs   := $(cu_srcs:.cu=.cu.o)
cu_objs   := $(cu_objs:src/%=objs/%)
cu_mk     := $(cu_objs:.cu.o=.cu.mk)

include_paths := src        \
            /usr/include/opencv4 \
            /usr/include/aarch64-linux-gnu \
            /usr/local/cuda-10.2/include

library_paths := /usr/lib/aarch64-linux-gnu \
            /usr/local/cuda-10.2/lib64

link_librarys := opencv_core opencv_highgui opencv_imgproc opencv_videoio opencv_imgcodecs \
            nvinfer nvinfer_plugin nvonnxparser \
            cuda cublas cudart cudnn \
            stdc++ dl

empty         :=
export_path   := $(subst $(empty) $(empty),:,$(library_paths))

run_paths     := $(foreach item,$(library_paths),-Wl,-rpath=$(item))
include_paths := $(foreach item,$(include_paths),-I$(item))
library_paths := $(foreach item,$(library_paths),-L$(item))
link_librarys := $(foreach item,$(link_librarys),-l$(item))

cpp_compile_flags := -std=c++11 -fPIC -w -g -pthread -fopenmp -O0
cu_compile_flags  := -std=c++11 -g -w -O0 -Xcompiler "$(cpp_compile_flags)"
```

```
link_flags        := -pthread -fopenmp -Wl,-rpath='$$ORIGIN'

cpp_compile_flags += $(include_paths)
cu_compile_flags  += $(include_paths)
link_flags        += $(library_paths) $(link_librarys) $(run_paths)

ifneq ($(MAKECMDGOALS), clean)
-include $(cpp_mk) $(cu_mk)
endif

pro    := workspace/pro
expath := library_path.txt

library_path.txt :
    @echo LD_LIBRARY_PATH=$(export_path):"$$"LD_LIBRARY_PATH > $@

workspace/pro : $(cpp_objs) $(cu_objs)
        @echo Link $@
        @mkdir -p $(dir $@)
        @$(cc) $^ -o $@ $(link_flags)

objs/%.cpp.o : src/%.cpp
    @echo Compile CXX $<
    @mkdir -p $(dir $@)
    @$(cc) -c $< -o $@ $(cpp_compile_flags)

objs/%.cu.o : src/%.cu
    @echo Compile CUDA $<
    @mkdir -p $(dir $@)
    @$(nvcc) -c $< -o $@ $(cu_compile_flags)

objs/%.cpp.mk : src/%.cpp
    @echo Compile depends CXX $<
    @mkdir -p $(dir $@)
    @$(cc) -M $< -MF $@ -MT $(@:.cpp.mk=.cpp.o) $(cpp_compile_flags)

objs/%.cu.mk : src/%.cu
    @echo Compile depends CUDA $<
    @mkdir -p $(dir $@)
    @$(nvcc) -M $< -MF $@ -MT $(@:.cu.mk=.cu.o) $(cu_compile_flags)
```

```
77
78  run    : workspace/pro
79           @cd workspace && ./pro
80
81  clean :
82      @rm -rf objs workspace/pro
83      @rm -rf library_path.txt
84      @rm -rf workspace/Result.jpg
85
86  # 导出符号，使得运行时能够链接上
87  export LD_LIBRARY_PATH:=$(export_path):$(LD_LIBRARY_PATH)
```

编译运行

```
1  make run
```

出现错误： make: Warning: File "xxx" has modification time yyy s in the future

参考： https://blog.csdn.net/u012814856/article/details/99873057

4 摄像头检测

简单写了一个摄像头检测的 demo，主要修改以下几点：

1.main.cpp 新增 yolo_video_demo() 函数，具体内容参考下面

2.main.cpp 新增调用 yolo_video_demo() 函数代码，具体内容参考下面

```
1  static void yolo_video_demo(const string& engine_file){
   // 修改1 新增函数
2    auto yolo = yolo::load(engine_file, yolo::Type::V8);
3    if (yolo == nullptr)  return;
4
5    // auto remote_show = create_zmq_remote_show();
6
7    cv::Mat frame;
8    cv::VideoCapture cap(0);
9    if (!cap.isOpened()){
10     printf("Engine is nullptr");
11     return;
12   }
13
14   while(true){
15     cap.read(frame);
16     auto objs = yolo->forward(cvimg(frame));
17
```

```
    for(auto &obj : objs) {
      uint8_t b, g, r;
      tie(b, g, r) = yolo::random_color(obj.class_label);
      cv::rectangle(frame, cv::Point(obj.left, obj.top), cv::Point(obj.right, obj.bottom),
                    cv::Scalar(b, g, r), 5);

      auto name = mylabels[obj.class_label];
      auto caption = cv::format("%s %.2f", name, obj.confidence);
      int width = cv::getTextSize(caption, 0, 1, 2, nullptr).width + 10;
      cv::rectangle(frame, cv::Point(obj.left - 3, obj.top - 33),
                    cv::Point(obj.left + width, obj.top), cv::Scalar(b, g, r), -1);
      cv::putText(frame, caption, cv::Point(obj.left, obj.top - 5), 0, 1, cv::Scalar::all(0), 2, 16);
    }
      imshow("frame", frame);
      // remote_show->post(frame);
      int key = cv::waitKey(1);
      if (key == 27)
          break;
  }

  cap.release();
  cv::destroyAllWindows();
  return;
}

int main() {     // 修改2 调用该函数
  // perf();
  // batch_inference();
  // single_inference();
  yolo_video_demo("best.transd.sim.engine");
  return 0;
}
```

5 串口通信

先在 Jetson nano 上查看启用的串口

```
ls -l /dev/tty*
```

参考：https://blog.csdn.net/qq_25662827/article/details/122581819

6　户外使用

使用微雪 UPS Power Module (B) 电源

安装教程：https://www.bilibili.com/video/BV1Be4y1Q7id

参考：https://blog.csdn.net/qq_40672115/article/details/129640372