# C++部署__fastdeploy

FastDeploy面向AI模型产业落地，帮助开发者简单几步即可完成AI模型在对应硬件上的部署，降低部署难度、压缩部署时间成本。支持40多个主流的AI模型在8大类常见硬件上的部署能力。

参考文档：

```
https://www.paddlepaddle.org.cn/
https://www.paddlepaddle.org.cn/fastdeploy
```

参考代码：

```
https://github.com/PaddlePaddle/FastDeploy
```

# 一、工控机基本环境安装

可参考《ubuntu装机》

**nvidia驱动安装：**

注意：系统重启后可能会出现nvidia驱动出现问题，可能是由于系统内核升级造成的。开发快速解决办法：重新安装驱动。`sudo ./NVIDIA-Linux-x86_64-525.116.04.run`；部署解决办法：避免系统内核升级，锁定内核版本。

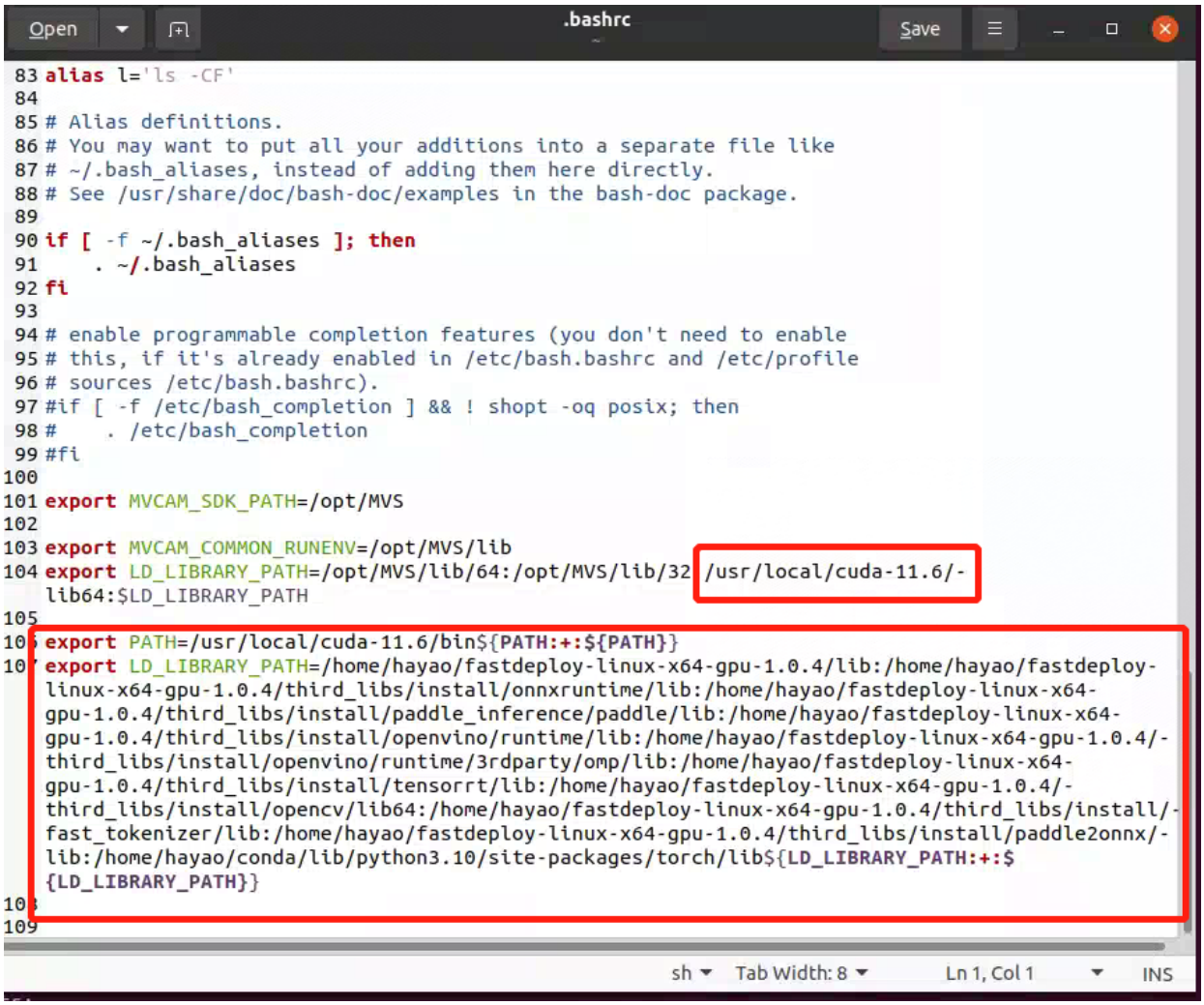
```
cat /proc/version #20.04
# 显卡：3060
# 安装nvidia 驱动 https://blog.csdn.net/weixin_48506823/article/details/123127186
报错解决办法：
https://www.cnblogs.com/xiaohuidi/p/14326784.html
```

**cuda 安装：**

https://blog.csdn.net/weixin_44857882/article/details/124108458

https://blog.csdn.net/weixin_37926734/article/details/123033286

```
wget
https://developer.download.nvidia.com/compute/cuda/11.6.0/local_installers/cuda_11.6
.0_510.39.01_linux.run
sudo sh cuda_11.6.0_510.39.01_linux.run
# continue->accept->取消driver安装后install
#配置环境变量 gedit ~/.bashrc
```




**cudnn安装：**

https://developer.nvidia.com/rdp/cudnn-download 官网注册账号下载对应版本（cuda11.*）

解压安装：

```
cp lib/* /usr/local/cuda-11.6/lib64/
cp include/* /usr/local/cuda-11.6/include/
```

**python 环境安装：**

```
apt update

apt install build-essential zlib1g-dev libncurses5-dev libgdbm-dev libnss3-dev
libssl-dev libreadline-dev libffi-dev libsqlite3-dev wget libbz2-dev

wget https://www.python.org/ftp/python/3.10.8/Python-3.10.8.tgz
tar -xf Python-3.10.*.tgz
cd Python-3.10.*/

./configure --enable-optimizations --prefix=/usr/local/python

make -j8
make install
```

**PyTorch安装:**

```
apt install python3-pip
# CUDA 11.6 python_base:3.8
pip3 install torch==1.12.1+cu116 torchvision==0.13.1+cu116 torchaudio==0.12.1 --
extra-index-url https://download.pytorch.org/whl/cu116
```

**yolov5 测试显卡调用:**

```
git clone https://github.com/ultralytics/yolov5

# -i https://pypi.tuna.tsinghua.edu.cn/simple
pip install -r requirements.txt

python3 detect.py --weights yolov5s.pt --source data/images/bus.jpg --device 0
```

**Cmake 安装:**

```
# https://cmake.org/download/
wget https://github.com/Kitware/CMake/releases/download/v3.25.3/cmake-3.25.3.tar.gz

tar -zxvf cmake-3.25.3.tar.gz

apt install g++
apt install build-essential
apt install libssl-dev

cd cmake-3.25.3
./bootstrap

make -j8
make install
```

**opencv安装配置：**

```
apt-get install build-essential

# libgtk2.0-dev 失败
apt-get install libgtk2.0-dev pkg-config libavcodec-dev libavformat-dev libswscale-dev

apt-get install python-dev python-numpy libtbb2 libtbb-dev libjpeg-dev libpng-dev libtiff-dev libdc1394-22-dev

wget -O opencv-4.6.0.zip https://github.com/opencv/opencv/archive/refs/tags/4.6.0.zip

cd opencv-4.6.0

mkdir -p build && cd build
cmake ..
make -j8
make install
```

**spdlog 和 nlohmann:**

```
git clone https://github.com/gabime/spdlog.git
cd spdlog && mkdir build && cd build
cmake ..
make
make install

git clone https://github.com/nlohmann/json.git
mkdir build && cd build
cmake ..
make
make install
```

**fastdeploy:**

```
wget https://bj.bcebos.com/fastdeploy/release/cpp/fastdeploy-linux-x64-gpu-1.0.4.tgz
tar xvf fastdeploy-linux-x64-gpu-1.0.4.tgz
# 不需要编译
# c++ 推理代码：cpp_info
mkdir build
cd build
cmake ..
make

./infer_demo /home/hayao/hayao_algo/config_source /home/hayao/hayao_algo/images # 报错： error while loading shared libraries: libonnxruntime.so.1.12.0: cannot open shared object file: No such file or directory
```

```
source /root/fastdeploy-linux-x64-gpu-1.0.4/fastdeploy_init.sh 再执行，报错
libcudart.so.11.0: cannot open shared object file: No such file or directory

# 解决 在~/.bashrc中添加环境变量 再source
export LD_LIBRARY_PATH=/home/hayao/fastdeploy-linux-x64-gpu-
1.0.4/lib:/home/hayao/fastdeploy-linux-x64-gpu-
1.0.4/third_libs/install/onnxruntime/lib:/home/hayao/fastdeploy-linux-x64-gpu-
1.0.4/third_libs/install/paddle_inference/paddle/lib:/home/hayao/fastdeploy-linux-
x64-gpu-1.0.4/third_libs/install/openvino/runtime/lib:/home/hayao/fastdeploy-linux-
x64-gpu-
1.0.4/third_libs/install/openvino/runtime/3rdparty/omp/lib:/home/hayao/fastdeploy-
linux-x64-gpu-1.0.4/third_libs/install/tensorrt/lib:/home/hayao/fastdeploy-linux-
x64-gpu-1.0.4/third_libs/install/opencv/lib64:/home/hayao/fastdeploy-linux-x64-gpu-
1.0.4/third_libs/install/fast_tokenizer/lib:/home/hayao/fastdeploy-linux-x64-gpu-
1.0.4/third_libs/install/paddle2onnx/lib:/home/hayao/conda/lib/python3.10/site-
packages/torch/lib

# 测试:
./infer_demo /home/hayao/hayao_algo/config_source /home/hayao/hayao_algo/images--
[ok]
```

# 二、yolov5服务器环境配置【容器】

服务器yolov5开发容器环境 https://hub.docker.com/r/ultralytics/yolov5/tags

```
docker pull ultralytics/yolov5:latest

docker run -dit --name hayao -p 5322:22 -p 5330-5399:5330-5399 -v
/etc/localtime:/etc/localtime -v /etc/machine-id:/etc/machine-id -v
/data01/hayao:/tmp/project -v /dev/shm:/dev/shm --gpus all --privileged
ultralytics/yolov5:latest /bin/bash

# 测试 python 3.10
python detect.py --weights yolov5s.pt --source data/images/bus.jpg --device 0
找不到有效gpu

pip uninstall torch*

# https://pytorch.org/get-started/previous-versions/
#如果速度慢，考虑换源
conda config --add channels https://mirrors.ustc.edu.cn/anaconda/pkgs/free/
conda config --add channels https://mirrors.ustc.edu.cn/anaconda/pkgs/main/
#下面两个非常重要，从清华源下载速度非一般
conda config --add channels
https://mirrors.tuna.tsinghua.edu.cn/anaconda/cloud/pytorch/
conda config --add channels
https://mirrors.tuna.tsinghua.edu.cn/anaconda/cloud/conda-forge/
#查看是否添加成功
conda config --show-sources
```

```
conda install pytorch==1.12.1 torchvision==0.13.1 torchaudio==0.12.1
cudatoolkit=10.2 -c pytorch # 失败 docker run 加上 --gpus all 重试 OK

# 安装yolov5
git clone https://github.com/ultralytics/yolov5

# 环境包安装
pip install -r requirements.txt

# 测试成功
python detect.py --weights yolov5s.pt --source data/images/bus.jpg --device 0


#使用vscode远程连接docker开发

# 宿主机与docker数据传输：创建docker时挂载目录，同步 /data01/hayao:/tmp/project

# 数据集配置 wood_test.yaml
# 模型配置 wood_yolov5s.yaml
# 训练  https://github.com/wudashuo/yolov5
python train.py --data data/wood_test.yaml --cfg models/wood_yolov5s.yaml --weight
pretrained/yolov5s.pt --epochs 50 --batch-size 16 --device 2,3
# 训练结果保存目录 ./runs/train/exp
python train.py --data data/hayao.yaml --cfg models/hayao_yolov5s.yaml --weight
pretrained/yolov5s.pt --epochs 100 --batch-size 16 --device 2,3
# 训练结果保存目录 ./runs/train/exp2


# 训练结果测试
python detect.py --weights /root/yolov5/runs/train/exp/weights/best.pt --source
/tmp/project/getech/yolov5_test/data/images/test/00342.jpg --device 0
# 测试结果保存目录：./runs/detect/exp2
python detect.py --weights /root/yolov5/runs/train/exp2/weights/best.pt --source
/tmp/project/demo_data/dataset/images/val/Image_20230310173745647.bmp --device 0
# 测试结果保存目录：./runs/detect/exp3

# 导出onnx 参数opset改为12
python export.py --data data/wood_test.yaml --weights runs/train/exp/weights/best.pt
--include onnx --device 0

python export.py --data data/hayao.yaml --weights runs/train/exp2/weights/best.pt --
include onnx --device 0

/root/yolov5/runs/train/exp2/weights/best.onnx
```

# 三、服务器Fastdeploy部署环境【容器】

https://github.com/PaddlePaddle/FastDeploy/tree/develop/examples/vision/detection/yolov5

https://www.paddlepaddle.org.cn/fastdeploy-api-doc/python/html/vision_results_cn.html

```
docker run -dit --name yolov5_deploy -p 8322:22 -p 8330-8399:8330-8399 -v
/etc/localtime:/etc/localtime -v /etc/machine-id:/etc/machine-id -v
/data01/hayao:/tmp/project -v /dev/shm:/dev/shm --gpus all --privileged
ultralytics/yolov5:latest /bin/bash

# 训练;  指标误检较高
python train.py --data data/hayao.yaml --cfg models/hayao_yolov5s.yaml --weight
pretrained/yolov5s.pt --epochs 100 --batch-size 16 --device 2,3

# 转onnx
python export.py --data data/hayao.yaml --weights runs/train/exp/weights/best.pt --
include onnx --device 0

# 测试 Image_20230310171432581
python detect.py --weights /root/yolov5/runs/train/exp/weights/best.pt --source
/tmp/project/demo_data/dataset/images/val/Image_20230310171432581.bmp --device 0
```

- YOLOv5 v7.0部署模型实现来自 YOLOv5 和基于COCO的预训练模型
  - （1）官方库提供的*.onnx可直接进行部署；
  - （2）开发者基于自己数据训练的YOLOv5 v7.0模型，可使用YOLOv5中的`export.py`导出ONNX文件后，完成部署。

## 1、C++ GPU部署环境：

环境要求：

- CUDA >= 11.2
- cuDNN >= 8.0
- python >= 3.6
- OS: Linux(x64)/Windows 10(x64)

**在工控机（30系显卡）测试（也可以在2080服务器测试（yolov5_deploy））：**

```
docker run -dit --name yolov5_deploy -p 8322:22 -p 8330-8399:8330-8399 -v
/etc/localtime:/etc/localtime -v /etc/machine-id:/etc/machine-id -v
/home/wood:/tmp/project -v /dev/shm:/dev/shm --gpus all --privileged
ultralytics/yolov5:latest /bin/bash

# 直接使用原有环境：CUDA11.3 PyTorch 1.13.1 torchvision 0.14.1
# 安装yolov5
git clone https://github.com/ultralytics/yolov5

# 环境包安装
pip install -r requirements.txt

# 测试成功
python detect.py --weights yolov5s.pt --source data/images/bus.jpg --device 0
```

```
# 训练
python train.py --data data/hayao.yaml --cfg models/hayao_yolov5s.yaml --weight
pretrained/yolov5s.pt --epochs 100 --batch-size 16 --device 2,3

# 测试
python detect.py --weights /root/yolov5/runs/train/exp/weights/best.pt --source
/tmp/project/demo_data/dataset/images/val/Image_20230310171432581.bmp --device 0

# 转onnx
python export.py --data data/hayao.yaml --weights runs/train/exp/weights/best.pt --
include onnx --device 0
```

**C++ SDK 安装：**

```
wget https://bj.bcebos.com/fastdeploy/release/cpp/fastdeploy-linux-x64-gpu-1.0.4.tgz
tar xvf fastdeploy-linux-x64-gpu-1.0.4.tgz
# 先安装Cmake
# c++ 推理代码：cpp_info
mkdir build
cd build
cmake ..
make # 报错 fatal error: cuda_runtime_api.h: No such file or directory 常规/usr/local
下找不到，这是镜像的问题，docker inspect ultralytics/yolov5:latest 查看
# 尝试重新安装 conda install pytorch==1.12.1 torchvision==0.13.1 torchaudio==0.12.1
cudatoolkit=11.6 -c pytorch -c conda-forge 还是不行
find -name cuda_runtime_api.h # ./opt/conda/include/cuda_runtime_api.h
# CMakeLists.txt 添加 include_directories(/opt/conda/include)
link_directories(/opt/conda/lib)
link_directories(/opt/conda/lib/python3.10/site-packages/torch/lib)

./infer_demo best.onnx Image_20230310171432581.bmp 1 # 报错： error while loading
shared libraries: libonnxruntime.so.1.12.0: cannot open shared object file: No such
file or directory
source /root/fastdeploy-linux-x64-gpu-1.0.4/fastdeploy_init.sh 再执行，还是报错
libcudart.so.11.0: cannot open shared object file: No such file or directory

# 尝试解决 在~/.bashrc中添加环境变量 再source
export LD_LIBRARY_PATH=/opt/conda/lib:/root/fastdeploy-linux-x64-gpu-
1.0.4/lib:/root/fastdeploy-linux-x64-gpu-
1.0.4/third_libs/install/onnxruntime/lib:/root/fastdeploy-linux-x64-gpu-
1.0.4/third_libs/install/paddle_inference/paddle/lib:/root/fastdeploy-linux-x64-gpu-
1.0.4/third_libs/install/openvino/runtime/lib:/root/fastdeploy-linux-x64-gpu-
1.0.4/third_libs/install/openvino/runtime/3rdparty/omp/lib:/root/fastdeploy-linux-
x64-gpu-1.0.4/third_libs/install/tensorrt/lib:/root/fastdeploy-linux-x64-gpu-
1.0.4/third_libs/install/opencv/lib64:/root/fastdeploy-linux-x64-gpu-
1.0.4/third_libs/install/fast_tokenizer/lib:/root/fastdeploy-linux-x64-gpu-
1.0.4/third_libs/install/paddle2onnx/lib:/opt/conda/lib/python3.10/site-
packages/torch/lib

# 测试:
./infer_demo best.onnx Image_20230310171432581.bmp 1 OK品
./infer_demo best.onnx Image_20230310170404244.bmp 1 NG品
```

**Cmake 安装：**

```
# https://cmake.org/download/
wget https://github.com/Kitware/CMake/releases/download/v3.25.3/cmake-3.25.3.tar.gz

tar -zxvf cmake-3.25.3.tar.gz

apt install g++
apt install build-essential
apt install libssl-dev

cd cmake-3.24.1
./bootstrap

make -j8
make install
```

**Opencv配置：**

参考 https://blog.csdn.net/qq_51022848/article/details/128095476

```
apt-get install build-essential
apt-get install libgtk2.0-dev pkg-config libavcodec-dev libavformat-dev libswscale-
dev
apt-get install python-dev python-numpy libtbb2 libtbb-dev libjpeg-dev libpng-dev
libtiff-dev libdc1394-22-dev

wget -O opencv-4.6.0.zip
https://github.com/opencv/opencv/archive/refs/tags/4.6.0.zip

wget -O opencv_contrib-4.6.0.zip
https://github.com/opencv/opencv_contrib/archive/refs/tags/4.6.0.zip

cp -r opencv_contrib-4.6.0 ./opencv-4.6.0

cd opencv-4.6.0
mkdir -p build && cd build

cmake -DCMAKE_BUILD_TYPE=Release -DOPENCV_GENERATE_PKGCONFIG=ON -
DCMAKE_INSTALL_PREFIX=/usr/local OPENCV_EXTRA_MODULES_PATH=../opencv_contrib-
4.6.0/modules ..   # 编译失败

# 先只编译opencv
mkdir -p build && cd build
cmake ..
# 报错:
    runtime library [libpng16.so.16] in /usr/lib/x86_64-linux-gnu may be hidden by
files in:
      /opt/conda/lib
```

```
    runtime library [libz.so.1] in /usr/lib/x86_64-linux-gnu may be hidden by files
in:
      /opt/conda/lib
    runtime library [libtiff.so.5] in /usr/lib/x86_64-linux-gnu may be hidden by
files in:
      /opt/conda/lib
# mv /opt/conda/lib 冲突

make -j32
make install
```

**spdlog/spdlog.h 配置:**

```
git clone https://github.com/gabime/spdlog.git
cd spdlog && mkdir build && cd build
cmake ..
make
make install

# nlohmann json
# 用法: https://github.com/nlohmann/json
# https://blog.csdn.net/gdizcm/article/details/121280329
git clone https://github.com/nlohmann/json.git
mkdir build && cd build
cmake ..
make
make install
```

**java环境配置:**

```
# jdk存放目录
/usr/local/java/jdk1.8.0_202
# 环境变量
vim /etc/profile 或者 vim ~/.bashrc
export JAVA_HOME=/usr/local/java/jdk1.8.0_202
export JRE_HOME=${JAVA_HOME}/jre
export CLASSPATH=.:${JAVA_HOME}/lib:${JRE_HOME}/lib
export PATH=${JAVA_HOME}/bin:$PATH

# 测试 java -version
```

常见国内镜像源:

```
清华：https://pypi.tuna.tsinghua.edu.cn/simple

阿里云：http://mirrors.aliyun.com/pypi/simple/

中国科技大学 https://pypi.mirrors.ustc.edu.cn/simple/

华中理工大学：http://pypi.hustunique.com/

山东理工大学：http://pypi.sdutlinux.org/

豆瓣：http://pypi.douban.com/simple/
```

### 2、C++ CPU部署环境：

环境要求：

- python >= 3.6
- OS: Linux(x64/aarch64)/Windows 10 x64/Mac OSX(x86/aarm64)

（可在T4服务器 yolov5_deploy容器测试）

# 四、接口调用方式

示例：jni 调用 c++ 接口调试 bull_vic->common_demo->paddle fastdeploy部署方式

```
# jbytearray 转 cvMat: https://www.saoniuhuo.com/question/detail-2440173.html
```

## 1、c++ -> jni调用方式

算法设计完成后，需要生成动态链接库提供给系统使用，接口设计参考如下：

### 算法需要提供：

- 推理使用的模型；
- 推理使用的配置文件；
- 推理额外参数；

推理类提供模型测试和推理接口，系统端根据算法人员提供的配置文件信息，实例化推理类。推理只有一个方法，

### 推理类说明：

**c++ 头文件(inference.h)如下：**

```
#ifndef _Included_inference_h
#define _Included_inference_h

#include "tracv_infer.h"
#include "det_infer.h"
#include "cls_infer.h"
#include "reg_infer.h"


class Inference
{
public:
    Inference(const string &config_dir);
    json run(std::vector<cv::Mat> & src_imgs, string &extra_info);
    ~Inference();

private:
    TracvInfer * tracv_inf = nullptr;
    DetInfer * det_inf = nullptr;
    ClsInfer * cls_inf = nullptr;
    RegInfer * reg_inf = nullptr;
};


#endif
```

**构造函数参数说明：**

```
Inference(const string &config_dir);
```

config_dir: 推理使用的配置文件目录。-- 算法人员提供配置文件的时候需要指明配置文件在config_dir目录下如何存放，是否有特殊子目录。配置文件可能是json或者yaml文件，里面包含如模型路径、过滤参数、产品规格信息等。例如以下路径：

```
/** 构造函数：json/yaml文件目录
- config_dir
--- traditioncv
----- inference.json
--- classification
----- inference.yaml
--- detection
----- inference.yaml
...
*/
```

**启动推理函数参数说明：**

```
json run(std::vector<cv::Mat> & src_imgs, string &extra_info);
```

src_imgs：图片数组。

extra_info：推理参数字典，如果需要特殊参数需要和系统商量，让系统将参数填入extra_info中，否则系统传一个空字符串。

**算法返回值：**

算法返回值需要按照json格式返回，按照传输的图片顺序返回图片信息。

coord: 推理结果位置信息，不同算法（如分类/目标检测/分割/传统cv测量不一致，此处需要项目中算法与系统对接）。

conf：置信度信息。

label: 返回标签信息，比如具体缺陷名称。

status: 状态信息，正常推理算法返回为 “OK"; 如果推理异常算法返回为”NG"。

```
[
    [""], //第一张图结果，表示为无缺陷。
    [{
        "conf": 0.9462416768074036,
        "coord": {
            "x_max": 1009,
            "x_min": 595,
            "y_max": 894,
            "y_min": 537
        },
        "label": "stomium",
        "status": "OK"
    },{
        "conf": 0.85,
        "coord": {
            "x_max": 2005,
            "x_min": 564,
            "y_max": 869,
            "y_min": 452
        },
        "label": "abnormal_ob",
        "status": "OK"
    },{
    ...
    }
    ], //第二张图结果，{}里面包含图里面缺陷名信息label,得分信息conf，位置信息coord，状态信息status，默认为OK
    ...
]
```

## 2、jni C++代码封装

```
#include "cn_getech_poros_algorithm_NativeFunction.h"
#include "inference.h"
#include "logger.h"
#include <algorithm>
#include <string>
#include <iostream>

//Inference类的实例化：完成模型初始化
extern "C" JNIEXPORT jlong JNICALL
Java_cn_getech_poros_algorithm_NativeFunction_init(
    JNIEnv *env, jobject thiz, jstring j_model_dir) {
    //...
    //...
  return reinterpret_cast<jlong>(infer);
}

//调用Inference类的run()方法对图片推理
extern "C" JNIEXPORT jstring JNICALL
Java_cn_getech_poros_algorithm_NativeFunction_forward(
    JNIEnv *env, jobject thiz, jlong java_pointer, jbyteArray data, jint jw, jint
jh, jint jc) {
    //...
    //...
  return cpp_string_to_jstring(env, str);
}

//内存释放
extern "C" JNIEXPORT void JNICALL
Java_cn_getech_poros_algorithm_NativeFunction_release(
    JNIEnv *env, jobject thiz, jlong java_pointer) {
  if (java_pointer == 0) {
    return;
  }
  Inference *ppredictor =
      (Inference *)java_pointer;
  delete ppredictor;
}
```

## 3、镜像与调试代码

镜像位置：服务器

镜像名称：common_algo

容器启动example(**以下为示例，注意名字，端口，挂载路径**)：

```
docker run -dit --name commom_algo_inf -p 5322:22 -p 5330-5399:5330-5399 -v
/etc/localtime:/etc/localtime -v /etc/machine-id:/etc/machine-id -v
/home/wood:/tmp/project -v /dev/shm:/dev/shm --gpus all --privileged
common_algo:latest /bin/bash
```

可自己启动一个新容器，也可直接在 commom_algo_inf 容器中测试代码：

```
# 例子参考：fastdeploy方式部署yolov5 v7.0
/root/cpp_inf/common_algo
```