

# YOLO26 + TensorRT实现C++快速识别

原创 于 2026-01-30 15:04:50 发布 · 504 阅读 · 12 · 8 · CC 4.0 BY-SA 版权

文章标签: #c++ #开发语言 #YOLO #tensorRT #计算机视觉 #实时检测

## 1. 引言

**YOLO 26:** 最新一代 YOLO 系实时 **检测器** (性能最佳)

**TensorRT :** NVIDIA GPU 推理加速核心 (低精度优化、层融合、显存优化)

C++ 部署: 工业级需求 (低延迟、高并发、跨平台)

## 2. 环境搭建:

[一键获取完整项目代码](#)

```
1 //本机环境
2 opencv-4.10.0
3 TensorRT-10.8.0.43
4 CUDA12.6
5 VS2019
```

## 3. 项目 搭建:

### 3.1 VS2019 创建空 C++ 项目

打开 VS2019, 点击「创建新项目」→选择「空项目」→点击「下一步」;

项目命名: YOLO26\_TRT, 选择保存路径→取消「将解决方案和项目放在同一目录中」→点击「创建」;  
右键项目名称 (Vajra\_TRT10) →「属性」→先确认「配置」选「Release」, 「平台」选「x64」

### 3.2 VS2019项目属性核心配置

#### 1. 常规配置 (基础兼容)

路径: 配置属性 → 常规

平台工具集: 选择「Visual Studio 2019 (v142)」(VS2019 默认, 若没有需安装 VS2019 的 v142 工具集);

字符集: 选择「使用多字节字符集」(避免 OpenCV 中文路径报错);

C++ 语言标准: 配置属性 → C/C++ → 语言 → C++ 标准 → 选择「ISO C++ 17 标准 (/std:c++17)」

#### 2. 包含目录配置 (让 VS 找到头文件)

路径: 配置属性 → VC++ 目录 → 常规 → 包含目录

点击「下拉箭头→编辑」, 添加以下路径 (替换为你的实际路径):

[一键获取完整项目代码](#)

```
1 C:\Program Files\NVIDIA GPU Computing Toolkit\CUDA\v12.6\include
2 D:\TensorRT-10.8.0.43\include
3 D:\opencv-4.10.0\build\include
4 D:\opencv-4.10.0\build\include\opencv2
```

点击「确定」保存。

#### 3. 库目录配置 (让 VS 找到 lib 文件)

路径: 配置属性 → VC++ 目录 → 常规 → 库目录

点击「下拉箭头→编辑」, 添加以下路径:

```
1 D:\opencv-4.10.0\build\x64\vc16\lib
```



百事从欢。913

已关注

```

2 |   3 | D:\TensorRT-10.8.0.43\lib
4 |
5 | C:\Program Files\NVIDIA GPU Computing Toolkit\CUDA\v12.6\lib\x64

```

点击「确定」保存。

#### 4. 链接器输入配置 (指定要链接的 lib 库)

路径: 配置属性 → 链接器 → 输入 → 附加依赖项

点击「下拉箭头→编辑」, 添加以下 lib 文件名 (逐行粘贴, 确保版本匹配) [应该有多余的但不影响]:

[一键获取完整项目代码](#)

```

1 | D:\opencv-4.10.0\build\x64\vc16\lib\opencv_world4100.lib
2 | C:\Program Files\NVIDIA GPU Computing Toolkit\CUDA\v12.8\lib\x64\cudart_static.lib
3 | D:\50GPU\TensorRT-10.8.0.43.Windows.win10.cuda-12.8\TensorRT-10.8.0.43\lib\nvinfer.lib
4 | D:\50GPU\TensorRT-10.8.0.43.Windows.win10.cuda-12.8\TensorRT-10.8.0.43\lib\nvnnxparser.lib
5 | D:\50GPU\TensorRT-10.8.0.43.Windows.win10.cuda-12.8\TensorRT-10.8.0.43\lib\nvinfer_plugin.lib
6 | C:\Program Files\NVIDIA GPU Computing Toolkit\CUDA\v12.6\lib\x64\cudart_static.lib
7 | kernel32.lib
8 | user32.lib
9 | gdi32.lib
10 | winspool.lib
11 | shell32.lib
12 | ole32.lib
13 | oleaut32.lib
14 | uuid.lib
15 | comdlg32.lib
16 | advapi32.lib

```

## 4、添加推理代码文件

右键项目名称 (YOLO26\_TRT) → 「添加」→ 「新建项」→ 选择「C++ 文件 (.cpp)」→ 命名为main.cpp→ 点击「添加」;

源代码如下:

cpp

[一键获取完整项目代码](#)

```

1 // 必须优先包含Windows头文件
2 #include <windows.h>
3 #include <iostream>
4 #include <vector>
5 #include <string>
6 #include <algorithm>
7 #include <cstring>
8 #include <fstream>
9 #include <direct.h>
10 #include <errno.h>
11 #include <iomanip>
12
13 // OpenCV头文件
14 #include <opencv2/opencv.hpp>
15 #include <opencv2/dnn.hpp>
16
17 // TensorRT 10.8头文件
18 #include <NvInfer.h>
19 #include <NvOnnxParser.h>
20 #include <NvInferRuntime.h>
21
22 // CUDA头文件
23 #include <cuda_runtime_api.h>
24
25 using namespace std;
26 using namespace cv;
27 using namespace nvinfer1;
28
29 // ===== 全局常量 (完全不变, 和你正常识别时一致)
30 static const int INPUT_W = 1280;
31 static const int INPUT_H = 640;

```



百事从欢。913

已关注

```

32 | static const int NUM_BOXES = 50;      33 | static const int BOX_DIM = 6;
34 | static const float CONF_THRESH = 0.3f;
35 | static const float NMS_THRESH = 0.3f;
36 | static const vector<string> CLASS_NAMES = { "hao", "huai", "3", "4", "5", "6", "7", "8", "9", "10" };
37 |
38 // ===== TensorRT日志类 =====
39 class Logger : public ILogger {
40 public:
41     void log(Severity severity, const char* msg) noexcept override {
42         if (severity <= Severity::kWARNING) {
43             cout << "[TensorRT] " << msg << endl;
44         }
45     }
46 } gLogger;
47
48 // ===== 工具函数 =====
49 bool createDirectory(const string& path) {
50     if (_mkdir(path.c_str()) == 0) return true;
51     return errno == EEXIST;
52 }
53
54 vector<string> listImageFiles(const string& dir) {
55     vector<string> img_paths;
56     string search_path = dir + "\\*.*";
57     WIN32_FIND_DATAA find_data;
58     HANDLE hFind = FindFirstFileA(search_path.c_str(), &find_data);
59
60     if (hFind == INVALID_HANDLE_VALUE) {
61         cout << "[警告] 无法遍历目录: " << dir << endl;
62         return img_paths;
63     }
64
65     do {
66         if (find_data.dwFileAttributes & FILE_ATTRIBUTE_DIRECTORY) continue;
67         string filename = find_data.cFileName;
68         size_t dot_pos = filename.find_last_of(".");
69         if (dot_pos == string::npos) continue;
70         string ext = filename.substr(dot_pos + 1);
71         transform(ext.begin(), ext.end(), ext.begin(), ::tolower);
72         if (ext == "bmp" || ext == "jpg" || ext == "png" || ext == "jpeg") {
73             img_paths.push_back(dir + "\\" + filename);
74         }
75     } while (FindNextFileA(hFind, &find_data));
76
77     FindClose(hFind);
78     return img_paths;
79 }
80
81 // ===== 修复版预处理: 正确识别 + 安全提速 =====
82 // 1. 恢复INTER_LINEAR, 和训练一致
83 // 2. 去掉错误的reshape+转置、static Mat
84 // 3. 优化convertTo归一化, 保留正确NCHW排布, 提速但不改动数据
85 void preprocessImage(const Mat& img, float* input_data) {
86     Mat rgb, img_letterbox;
87     // 1. BGR转RGB (和训练一致)
88     cvtColor(img, rgb, COLOR_BGR2RGB);
89
90     // 2. Letterbox缩放+填充 (恢复INTER_LINEAR, 模型训练用的插值)
91     int img_h = img.rows, img_w = img.cols;
92     float scale = min((float)INPUT_W / img_w, (float)INPUT_H / img_h);
93     int new_w = cvRound(img_w * scale);
94     int new_h = cvRound(img_h * scale);
95     int pad_w = (INPUT_W - new_w) / 2;
96     int pad_h = (INPUT_H - new_h) / 2;
97
98     // 恢复双线性插值, 保证和训练一致
99     resize(rgb, img_letterbox, Size(new_w, new_h), INTER_LINEAR);
100    copyMakeBorder(img_letterbox, img_letterbox, pad_h, INPUT_H - new_h - pad_h,
101                  pad_w, INPUT_W - new_w - pad_w, BORDER_CONSTANT, Sca
102

```



百事从欢。913

已关注

```

103 // 3. 高效归一化，一步完成类型转换+缩放 | Mat float_img;
104 img_letterbox.convertTo(float_img, CV_32FC3, 1.0 / 255.0f);
105
106
107 // 4. 安全高效NCHW赋值：保证通道/行列完全正确，比原始三重循环更快
108 // 逐通道批量拷贝，避免Python式低效循环，同时数据100%正确
109 const float* src_r = float_img.ptr<float>(0) + 0;
110 const float* src_g = float_img.ptr<float>(0) + 1;
111 const float* src_b = float_img.ptr<float>(0) + 2;
112
113 const int pixel_count = INPUT_W * INPUT_H;
114 for (int i = 0; i < pixel_count; ++i) {
115     input_data[i] = src_r[i * 3];
116     input_data[i + pixel_count] = src_g[i * 3];
117     input_data[i + pixel_count * 2] = src_b[i * 3];
118 }
119 }
120
121 // ===== 后处理 =====
122 struct DetectResult {
123     Rect box;
124     float confidence;
125     int cls_id;
126     string cls_name;
127 };
128
129 vector<DetectResult> postprocessYolo26(float* output, int img_width, int img_height) {
130     vector<DetectResult> valid_results;
131     const float input_size_w = static_cast<float>(INPUT_W);
132     const float input_size_h = static_cast<float>(INPUT_H);
133
134     float scale = min(input_size_w / (float)img_width, input_size_h / (float)img_height);
135     float pad_w = (input_size_w - (float)img_width * scale) / 2.0f;
136     float pad_h = (input_size_h - (float)img_height * scale) / 2.0f;
137
138     for (int i = 0; i < NUM_BOXES; ++i) {
139         int idx = i * BOX_DIM;
140
141         float x1_letter = output[idx + 0];
142         float y1_letter = output[idx + 1];
143         float x2_letter = output[idx + 2];
144         float y2_letter = output[idx + 3];
145         float confidence = output[idx + 4];
146         int cls_id = static_cast<int>(output[idx + 5]);
147
148         if (
149             confidence < CONF_THRESH || confidence > 1.0f ||
150             cls_id < 0 || cls_id >= CLASS_NAMES.size() ||
151             x1_letter < 0 || x1_letter > input_size_w ||
152             y1_letter < 0 || y1_letter > input_size_h ||
153             x2_letter < 0 || x2_letter > input_size_w ||
154             y2_letter < 0 || y2_letter > input_size_h ||
155             x2_letter <= x1_letter || y2_letter <= y1_letter
156         ) {
157             continue;
158         }
159
160         x1_letter -= pad_w;
161         y1_letter -= pad_h;
162         x2_letter -= pad_w;
163         y2_letter -= pad_h;
164
165         float x1 = x1_letter / scale;
166         float y1 = y1_letter / scale;
167         float x2 = x2_letter / scale;
168         float y2 = y2_letter / scale;
169
170         x1 = max(0.0f, min(x1, (float)img_width - 1));
171         y1 = max(0.0f, min(y1, (float)img_height - 1));
172         x2 = max(x1 + 1.0f, min(x2, (float)img_width - 1));
173         y2 = max(y1 + 1.0f, min(y2, (float)img_height - 1));
174

```



百事从欢。913

已关注

```

175 |         int x1_int = static_cast<int>(round(x1));
176 |         int y1_int = static_cast<int>(round(y1));
177 |         int x2_int = static_cast<int>(round(x2));
178 |         int y2_int = static_cast<int>(round(y2));
179 |
180     DetectResult res;
181     res.box = Rect(x1_int, y1_int, x2_int - x1_int, y2_int - y1_int);
182     res.confidence = confidence;
183     res.cls_id = cls_id;
184     res.cls_name = CLASS_NAMES[cls_id];
185     valid_results.push_back(res);
186 }
187
188 if (!valid_results.empty()) {
189     vector<Rect> boxes;
190     vector<float> confidences;
191     vector<int> cls_ids;
192     for (const auto& res : valid_results) {
193         boxes.push_back(res.box);
194         confidences.push_back(res.confidence);
195         cls_ids.push_back(res.cls_id);
196     }
197     vector<int> indices;
198     cv::dnn::NMSBoxes(boxes, confidences, CONF_THRESH, NMS_THRESH, indices);
199     vector<DetectResult> nms_results;
200     for (int idx : indices) {
201         nms_results.push_back(valid_results[idx]);
202     }
203     valid_results = nms_results;
204 }
205
206 return valid_results;
207 }
208
209 // ====== 引擎构建 ======
210 bool buildEngine(const string& onnx_path, const string& engine_path) {
211     IBuilder* builder = createInferBuilder(gLogger);
212     if (!builder) return false;
213
214     uint32_t flag = 1U << static_cast<uint32_t>(NetworkDefinitionCreationFlag::kEXPLICIT_BATCH);
215     INetworkDefinition* network = builder->createNetworkV2(flag);
216     if (!network) {
217         delete builder;
218         return false;
219     }
220
221     nvonnxparser::IParser* parser = nvonnxparser::createParser(*network, gLogger);
222     if (!parser->parseFromFile(onnx_path.c_str(), static_cast<int>(ILogger::Severity::kWARNING))) {
223         cout << "[错误] ONNX模型解析失败!" << endl;
224         delete parser;
225         delete network;
226         delete builder;
227         return false;
228     }
229
230     IBuilderConfig* config = builder->createBuilderConfig();
231 #if NV_TENSORRT_MAJOR >= 9
232     config->setMemoryPoolLimit(MemoryPoolType::kWORKSPACE, 1ULL << 30);
233 #else
234     config->setMaxWorkspaceSize(1ULL << 30);
235 #endif
236
237     IOptimizationProfile* profile = builder->createOptimizationProfile();
238     ITensor* input_tensor = network->getInput(0);
239     const char* input_name = input_tensor->getName();
240     Dims4 input_dims(1, 3, INPUT_H, INPUT_W);
241     profile->setDimensions(input_name, OptProfileSelector::kMIN, input_dims);
242     profile->setDimensions(input_name, OptProfileSelector::kOPT, input_dims);
243     profile->setDimensions(input_name, OptProfileSelector::kMAX, input_dims);
244     config->addOptimizationProfile(profile);
245 }
```



百事从欢。913

已关注

```

246     ICudaEngine* engine = builder->buildEngineWithConfig(*network, *config); 247     if (!engine) {
248         cout << "[错误] TensorRT引擎构建失败! " << endl;
249         delete config;
250         delete parser;
251         delete network;
252         delete builder;
253         return false;
254     }
255
256     IHostMemory* serialized_engine = engine->serialize();
257     ofstream ofs(engine_path, ios::binary);
258     ofs.write(reinterpret_cast<const char*>(serialized_engine->data()), serialized_engine->size());
259     ofs.close();
260
261     delete serialized_engine;
262     delete engine;
263     delete config;
264     delete parser;
265     delete network;
266     delete builder;
267
268     cout << "[成功] 引擎保存至: " << engine_path << endl;
269     return true;
270 }
271
272 // ====== 引擎加载 ======
273 ICudaEngine* loadEngine(const string& engine_path) {
274     ifstream ifs(engine_path, ios::binary);
275     if (!ifs) {
276         cout << "[错误] 无法打开引擎文件: " << engine_path << endl;
277         return nullptr;
278     }
279     ifs.seekg(0, ios::end);
280     size_t size = ifs.tellg();
281     ifs.seekg(0, ios::beg);
282     char* buffer = new char[size];
283     ifs.read(buffer, size);
284     ifs.close();
285
286     IRuntime* runtime = createInferRuntime(gLogger);
287     ICudaEngine* engine = runtime->deserializeCudaEngine(buffer, size);
288     delete[] buffer;
289     delete runtime;
290
291     if (!engine) {
292         cout << "[错误] 引擎加载失败! " << endl;
293         return nullptr;
294     }
295
296     cout << "[成功] 引擎加载完成! " << endl;
297     return engine;
298 }
299
300 // ====== 推理与计时 ======
301 Mat inferSingleImage(ICudaEngine* engine, const Mat& img) {
302     IExecutionContext* context = engine->createExecutionContext();
303     if (!context) return img.clone();
304
305     const int input_idx = 0;
306     const int output_idx = 1;
307     context->setOptimizationProfileAsync(0, 0);
308
309     const int input_size = 3 * INPUT_W * INPUT_H;
310     const int output_size = NUM_BOXES * BOX_DIM;
311
312     float* host_input = new float[input_size];
313     float* host_output = new float[output_size];
314
315     float* device_input = nullptr;
316     float* device_output = nullptr;

```



百事从欢。913

已关注

```

317     cudaMalloc((void**)&device_input, input_size * sizeof(float));318 |
318     cudaMalloc((void**)&device_output, output_size * sizeof(float));319 |
320     // 预处理计时
321     double preprocess_start = static_cast<double>(getTickCount());
322     preprocessImage(img, host_input);
323     double preprocess_end = static_cast<double>(getTickCount());
324     double preprocess_time = (preprocess_end - preprocess_start) / getTickFrequency() * 1000;
325     cout << "[耗时] 预处理: " << fixed << setprecision(2) << preprocess_time << " 毫秒" << endl;
326
327     cudaMemcpy(device_input, host_input, input_size * sizeof(float), cudaMemcpyHostToDevice);
328
329     void* bindings[2] = { nullptr };
330     bindings[input_idx] = device_input;
331     bindings[output_idx] = device_output;
332
333     // 推理计时
334     double infer_start = static_cast<double>(getTickCount());
335     bool success = context->executeV2(bindings);
336     double infer_end = static_cast<double>(getTickCount());
337     double infer_time = (infer_end - infer_start) / getTickFrequency() * 1000;
338     cout << "[耗时] 推理核心: " << fixed << setprecision(2) << infer_time << " 毫秒" << endl;
339
340     if (!success) {
341         cout << "[错误] 推理执行失败! " << endl;
342         cudaFree(device_input);
343         cudaFree(device_output);
344         delete[] host_input;
345         delete[] host_output;
346         delete context;
347         return img.clone();
348     }
349
350     cudaMemcpy(host_output, device_output, output_size * sizeof(float), cudaMemcpyDeviceToHost);
351     vector<DetectResult> results = postprocessYolo26(host_output, img.cols, img.rows);
352
353     Mat result = img.clone();
354     for (const auto& res : results) {
355         rectangle(result, res.box, Scalar(0, 0, 255), 3);
356         string text = res.cls_name + ":" + to_string(res.confidence).substr(0, 4);
357         putText(result, text, Point(res.box.x, res.box.y - 10),
358                 FONT_HERSHEY_SIMPLEX, 1.0f, Scalar(255, 255, 255), 2);
359     }
360
361     cudaFree(device_input);
362     cudaFree(device_output);
363     delete[] host_input;
364     delete[] host_output;
365     delete context;
366
367     return result;
368 }
369
370 // ===== 批量推理 (不变) =====
371 void batchInfer(const string& engine_path, const string& img_dir) {
372     ICudaEngine* engine = loadEngine(engine_path);
373     if (!engine) return;
374
375     string output_dir = img_dir + "\\output";
376     if (!createDirectory(output_dir)) {
377         cout << "[错误] 无法创建输出目录: " << output_dir << endl;
378         delete engine;
379         return;
380     }
381
382     vector<string> img_paths = listImageFiles(img_dir);
383     if (img_paths.empty()) {
384         cout << "[警告] 未找到支持的图片文件: " << img_dir << endl;
385         delete engine;
386         return;
387     }
388 }
```



百事从欢。913

已关注

```

389     int processed = 0;
390         390 |     for (const string& path : img_paths) {
391             cout << "\n[处理中] " << path << endl;
392
393             Mat img = imread(path, IMREAD_UNCHANGED);
394             if (img.empty()) {
395                 cout << "[警告] 无法读取图片: " << path << endl;
396                 continue;
397             }
398
399             Mat result = inferSingleImage(engine, img);
400
401             size_t pos = path.find_last_of("\\");
402             string filename = path.substr(pos + 1);
403             string save_path = output_dir + "\\\" + filename;
404             if (imwrite(save_path, result)) {
405                 processed++;
406                 cout << "[成功] 结果保存至: " << save_path << endl;
407             }
408             else {
409                 cout << "[错误] 保存图片失败: " << save_path << endl;
410             }
411         }
412
413         cout << "\n[批量处理完成] 总计: " << img_paths.size()
414             << " | 成功: " << processed
415             << " | 失败: " << img_paths.size() - processed << endl;
416         cout << "[结果保存目录] " << output_dir << endl;
417
418         delete engine;
419     }
420
421 // ====== 主函数: 只保留安全的多线程优化 ======
422 int main(int argc, char** argv) {
423     // 安全优化: 开启OpenCV多线程和指令集, 不改动数据, 只提速
424     cv::setUseOptimized(true);
425     cv::setNumThreads(std::min(4, cv::getNumberOfCPUs()));
426
427     setlocale(LC_ALL, "Chinese");
428     cout << fixed << setprecision(2);
429
430     if (argc != 4) {
431         cout << "===== YOLO26 TensorRT 推理工具 =====" << endl;
432         cout << "用法1 (转换ONNX到引擎) : " << argv[0] << " -s onnx_path engine_path" << endl;
433         cout << "用法2 (批量推理图片) : " << argv[0] << " -d engine_path img_dir" << endl;
434         cout << "===== ===== =====" << endl;
435         return -1;
436     }
437
438     string cmd = argv[1];
439     if (cmd == "-s") {
440         cout << "[开始转换] ONNX → TensorRT引擎" << endl;
441         if (!buildEngine(argv[2], argv[3])) {
442             cout << "[失败] 引擎转换失败!" << endl;
443             return -1;
444         }
445         cout << "[成功] 引擎转换完成!" << endl;
446     }
447     else if (cmd == "-d") {
448         cout << "[开始推理] 批量处理图片" << endl;
449         batchInfer(argv[2], argv[3]);
450     }
451     else {
452         cout << "[错误] 无效命令! 仅支持 -s 和 -d" << endl;
453         return -1;
454     }
455
456     return 0;
457 }

```



百事从欢。913

已关注

代码有两种用法

-s的模型转换

-d的批量图像识别

具体用法如下：

cpp

一键获取完整项目代码

```
1 ===== YOLO26 TensorRT 推理工具 =====
2 用法1（转换ONNX到引擎）：D:\Code\YOLO26-TRT\x64\Release\YOLO26-TRT.exe -s onnx_path engine_path
3 用法2（批量推理图片）：D:\Code\YOLO26-TRT\x64\Release\YOLO26-TRT.exe -d engine_path img_dir
4 =====
```

## 五、结语

如果在实践中遇到问题，欢迎在评论区交流；后续也会持续更新更多 AI 模型部署的实战技巧，敬请关注！

### YOLOv11智能车辆目标检测资源重磅开源，完全免费，零基础可用，全流程落地

YOLOv11车辆检测全流程落地资源！完全免费，零基础可用。含全量数据集、完整代码库和详细指南，已验证防坑，助你快速复现！

20年老程序员亲测：YOLO+TensorRT推理加速实战（速度提3倍，附完整代码+避坑）

专注于Python爬虫开发，分享爬虫技巧、项目实操

ONNX导出报错“Unsupported operator”开，opset选12，关闭动态维度；用GPU导出（device=0），避免CPU算子；TensorRT转引擎失败检查ONNX模型是否简化，删除

YOLO26实时性优化:TensorRT加速部署教程

本教程不讲理论推导,不堆参数对比,只聚焦一件事:如何把官方YOLO26模型,通过TensorRT加速,实现NVIDIA GPU上的高吞吐、低延迟推理部署。全程基于已预置环境的

Yolo系列模型的TensorRT-C++推理实践

Yolo系列模型的TensorRT-C++推理实践 在边缘计算和实时视觉系统日益普及的今天,如何让YOLO这类高效目标检测模型真正“跑得快、稳得住”,成了从算法到落地的关键一

YOLO学习笔记 | YOLO11对象检测，实例分割，姿态评估的TensorRT部署c++

尘世冰封的

该方案需要根据实际模型结构进行调整，特别是输出维度和后处理逻辑需要与具体模型版本严格匹配。建议使用TensorRT的polygraphy工具进行逐层验证，确保模型转换的

[C++][cmake]使用C++部署yolov13目标检测的tensorrt模型支持图片视频推理windows测试通过

FL1623863129的

首先需要大家安装好VS2019或者VS2022，还有如下环境，由于安装包很多需要去官方搜索下载，需要自己安装，其中版本可以有区别，但是如果快速复现这个项目，最好

YOLO26无人机应用:自动跟随系统部署教程

YOLO26无人机自动跟随系统部署教程:从零启动到实时推理 你是否想过让无人机像影子一样精准锁定目标、自主保持距离、平滑跟随移动物体?这不是科幻电影的桥段,而是

YOLO26实战:从ONNX到TensorRT的全平台一键导出指南

在YOLO26工业部署场景中,将PyTorch模型转换为TensorRT引擎是实现GPU实时推理的核心环节,但手动转换流程繁琐、跨平台命令差异大、ONNX算子不兼容、动态Shape

YOLO26模型部署 (C++)

低调猫熊的

本文介绍了YOLOv26模型的部署与测试过程。作者首先将官方提供的yolo26s.pt模型转换为ONNX格式，解析了模型输出格式（300个检测框，每个框包含坐标、置信度和

yolo26的TensorRT部署详解 (C++版)

低调猫熊的

本文介绍了使用TensorRT部署YOLO26模型的C++实现过程。主要包括模型初始化、推理和后处理三个部分：1) 通过加载.engine文件初始化模型，检查输入输出维度；2)

YOLO26镜像性能优化:让推理速度提升3倍

这些版本组合完全支持TensorRT和FP16 加速,无需额外升级。提示:不要轻易升级 PyTorch 或 CUDA 版本,可能导致兼容性问题。2.2 激活正确 Conda 环境 镜像启动后默

YOLO26边缘计算部署:Jetson设备适配实战指南

这套YOLO26官方训练与推理镜像,并非通用x86环境的简单移植,而是专为NVIDIA Jetson系列(Orin NX、Orin AGX、Xavier NX等)深度定制的生产级环境。它跳过了繁琐的

用C++部署yolov5+deepsort+tensorrt实现目标跟踪 热门推荐

IT3

一、参考资料 Jetson 系列——基于yolov5和deepsort的多目标头部识别，跟踪，使用tensorrt和c++加速 二、相关介绍 2.1 重要说明 该项目能部署在Jetson系列的产品，也

Windows+ONNX+TensorRT+YOLOV8+C++环境搭建

yljxh的

跑通了Python环境下的Yolov8，但是考虑到性能，想试试C++环境下的优化效果。

YOLO26训练效率低?PyTorch 1.10算力适配优化教程

YOLO26训练慢,80%源于默认参数未适配当前硬件。以下参数不是“可选”,而是必须调整: model.train( data='data.yaml', imgsz=640, epochs=200, batch=128,# 关键!填满G

工业视觉必看! YOLOv8+TensorRT实现实时缺陷检测的GPU优化攻略

layneyao的

工业视觉必看! YOLOv8+TensorRT实现实时缺陷检测的GPU优化攻略

【推理加速】TensorRT C++ 部署YOLO11全系模型

关注微信公众号【OpenCV学

TensorRT YOLO11 C++ 加速推理



百事从欢。913

已关注