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

```
//本机环境
opencv-4.10.0
TensorRT-10.8.0.43
CUDA12.6
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

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

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

```
D:\TensorRT-10.8.0.43\lib

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

```
D:\opencv-4.10.0\build\x64\vc16\lib\opencv_world4100.lib
C:\Program Files\NVIDIA GPU Computing Toolkit\CUDA\v12.8\lib\x64\cudart_static.lib
D:\50GPU\TensorRT-10.8.0.43.Windows.win10.cuda-12.8\TensorRT-10.8.0.43\lib\nvinfer.lib
D:\50GPU\TensorRT-10.8.0.43.Windows.win10.cuda-12.8\TensorRT-10.8.0.43\lib\nvonnxparser.lib
D:\50GPU\TensorRT-10.8.0.43.Windows.win10.cuda-12.8\TensorRT-10.8.0.43\lib\nvinfer_plugin.lib
C:\Program Files\NVIDIA GPU Computing Toolkit\CUDA\v12.6\lib\x64\cudart_static.lib
kernel32.lib
user32.lib
gdi32.lib
winspool.lib
shell32.lib
ole32.lib
oleaut32.lib
uuid.lib
comdlg32.lib
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

```cpp
// 必须优先包含Windows头文件
#include <windows.h>
#include <iostream>
#include <vector>
#include <string>
#include <algorithm>
#include <cstring>
#include <fstream>
#include <direct.h>
#include <errno.h>
#include <iomanip>

// OpenCV头文件
#include <opencv2/opencv.hpp>
#include <opencv2/dnn.hpp>

// TensorRT 10.8头文件
#include <NvInfer.h>
#include <NvOnnxParser.h>
#include <NvInferRuntime.h>

// CUDA头文件
#include <cuda_runtime_api.h>

using namespace std;
using namespace cv;
using namespace nvinfer1;

// ==================== 全局常量（完全不变，和你正常识别时一致） ====================
static const int INPUT_W = 1280;
static const int INPUT_H = 640;
```

```cpp
static const int NUM_BOXES = 50;
static const int BOX_DIM = 6;
static const float CONF_THRESH = 0.3f;
static const float NMS_THRESH = 0.3f;
static const vector<string> CLASS_NAMES = { "hao", "huai", "3", "4", "5", "6", "7", "8", "9", "10" };

// ==================== TensorRT日志类 ====================
class Logger : public ILogger {
public:
    void log(Severity severity, const char* msg) noexcept override {
        if (severity <= Severity::kWARNING) {
            cout << "[TensorRT] " << msg << endl;
        }
    }
} gLogger;

// ==================== 工具函数 ====================
bool createDirectory(const string& path) {
    if (_mkdir(path.c_str()) == 0) return true;
    return errno == EEXIST;
}

vector<string> listImageFiles(const string& dir) {
    vector<string> img_paths;
    string search_path = dir + "\\*.*";
    WIN32_FIND_DATAA find_data;
    HANDLE hFind = FindFirstFileA(search_path.c_str(), &find_data);

    if (hFind == INVALID_HANDLE_VALUE) {
        cout << "[警告] 无法遍历目录: " << dir << endl;
        return img_paths;
    }

    do {
        if (find_data.dwFileAttributes & FILE_ATTRIBUTE_DIRECTORY) continue;
        string filename = find_data.cFileName;
        size_t dot_pos = filename.find_last_of(".");
        if (dot_pos == string::npos) continue;
        string ext = filename.substr(dot_pos + 1);
        transform(ext.begin(), ext.end(), ext.begin(), ::tolower);
        if (ext == "bmp" || ext == "jpg" || ext == "png" || ext == "jpeg") {
            img_paths.push_back(dir + "\\" + filename);
        }
    } while (FindNextFileA(hFind, &find_data));

    FindClose(hFind);
    return img_paths;
}

// ==================== 修复版预处理：正确识别 + 安全提速 ====================
// 1. 恢复INTER_LINEAR，和训练一致
// 2. 去掉错误的reshape+转置、static Mat
// 3. 优化convertTo归一化，保留正确NCHW排布，提速但不改动数据
void preprocessImage(const Mat& img, float* input_data) {
    Mat rgb, img_letterbox;
    // 1. BGR转RGB（和训练一致）
    cvtColor(img, rgb, COLOR_BGR2RGB);

    // 2. Letterbox缩放+填充（恢复INTER_LINEAR，模型训练用的插值）
    int img_h = rgb.rows, img_w = rgb.cols;
    float scale = min((float)INPUT_W / img_w, (float)INPUT_H / img_h);
    int new_w = cvRound(img_w * scale);
    int new_h = cvRound(img_h * scale);
    int pad_w = (INPUT_W - new_w) / 2;
    int pad_h = (INPUT_H - new_h) / 2;

    // 恢复双线性插值，保证和训练一致
    resize(rgb, img_letterbox, Size(new_w, new_h), INTER_LINEAR);
    copyMakeBorder(img_letterbox, img_letterbox, pad_h, INPUT_H - new_h - pad_h,
        pad_w, INPUT_W - new_w - pad_w, BORDER_CONSTANT, Sca
```

```
    // 3. 高效归一化，一步完成类型转换+缩放    Mat float_img;
    img_letterbox.convertTo(float_img, CV_32FC3, 1.0 / 255.0f);

    // 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
    // 逐通道批量拷贝，避免Python式低效循环，同时数据100%正确
    const float* src_r = float_img.ptr<float>(0) + 0;
    const float* src_g = float_img.ptr<float>(0) + 1;
    const float* src_b = float_img.ptr<float>(0) + 2;

    const int pixel_count = INPUT_W * INPUT_H;
    for (int i = 0; i < pixel_count; ++i) {
        input_data[i] = src_r[i * 3];
        input_data[i + pixel_count] = src_g[i * 3];
        input_data[i + pixel_count * 2] = src_b[i * 3];
    }
}

// ==================== 后处理 ====================
struct DetectResult {
    Rect box;
    float confidence;
    int cls_id;
    string cls_name;
};

vector<DetectResult> postprocessYolo26(float* output, int img_width, int img_height) {
    vector<DetectResult> valid_results;
    const float input_size_w = static_cast<float>(INPUT_W);
    const float input_size_h = static_cast<float>(INPUT_H);

    float scale = min(input_size_w / (float)img_width, input_size_h / (float)img_height);
    float pad_w = (input_size_w - (float)img_width * scale) / 2.0f;
    float pad_h = (input_size_h - (float)img_height * scale) / 2.0f;

    for (int i = 0; i < NUM_BOXES; ++i) {
        int idx = i * BOX_DIM;

        float x1_letter = output[idx + 0];
        float y1_letter = output[idx + 1];
        float x2_letter = output[idx + 2];
        float y2_letter = output[idx + 3];
        float confidence = output[idx + 4];
        int cls_id = static_cast<int>(output[idx + 5]);

        if (
            confidence < CONF_THRESH || confidence > 1.0f ||
            cls_id < 0 || cls_id >= CLASS_NAMES.size() ||
            x1_letter < 0 || x1_letter > input_size_w ||
            y1_letter < 0 || y1_letter > input_size_h ||
            x2_letter < 0 || x2_letter > input_size_w ||
            y2_letter < 0 || y2_letter > input_size_h ||
            x2_letter <= x1_letter || y2_letter <= y1_letter
            ) {
            continue;
        }

        x1_letter -= pad_w;
        y1_letter -= pad_h;
        x2_letter -= pad_w;
        y2_letter -= pad_h;

        float x1 = x1_letter / scale;
        float y1 = y1_letter / scale;
        float x2 = x2_letter / scale;
        float y2 = y2_letter / scale;

        x1 = max(0.0f, min(x1, (float)img_width - 1));
        y1 = max(0.0f, min(y1, (float)img_height - 1));
        x2 = max(x1 + 1.0f, min(x2, (float)img_width - 1));
        y2 = max(y1 + 1.0f, min(y2, (float)img_height - 1));

```

```cpp
        int x1_int = static_cast<int>(round(x1));
        int y1_int = static_cast<int>(round(y1));
        int x2_int = static_cast<int>(round(x2));
        int y2_int = static_cast<int>(round(y2));

        DetectResult res;
        res.box = Rect(x1_int, y1_int, x2_int - x1_int, y2_int - y1_int);
        res.confidence = confidence;
        res.cls_id = cls_id;
        res.cls_name = CLASS_NAMES[cls_id];
        valid_results.push_back(res);
    }

    if (!valid_results.empty()) {
        vector<Rect> boxes;
        vector<float> confidences;
        vector<int> cls_ids;
        for (const auto& res : valid_results) {
            boxes.push_back(res.box);
            confidences.push_back(res.confidence);
            cls_ids.push_back(res.cls_id);
        }
        vector<int> indices;
        cv::dnn::NMSBoxes(boxes, confidences, CONF_THRESH, NMS_THRESH, indices);
        vector<DetectResult> nms_results;
        for (int idx : indices) {
            nms_results.push_back(valid_results[idx]);
        }
        valid_results = nms_results;
    }

    return valid_results;
}

// ==================== 引擎构建 ====================
bool buildEngine(const string& onnx_path, const string& engine_path) {
    IBuilder* builder = createInferBuilder(gLogger);
    if (!builder) return false;

    uint32_t flag = 1U << static_cast<uint32_t>(NetworkDefinitionCreationFlag::kEXPLICIT_BATCH);
    INetworkDefinition* network = builder->createNetworkV2(flag);
    if (!network) {
        delete builder;
        return false;
    }

    nvonnxparser::IParser* parser = nvonnxparser::createParser(*network, gLogger);
    if (!parser->parseFromFile(onnx_path.c_str(), static_cast<int>(ILogger::Severity::kWARNING))) {
        cout << "[错误] ONNX模型解析失败！" << endl;
        delete parser;
        delete network;
        delete builder;
        return false;
    }

    IBuilderConfig* config = builder->createBuilderConfig();
#if NV_TENSORRT_MAJOR >= 9
    config->setMemoryPoolLimit(MemoryPoolType::kWORKSPACE, 1ULL << 30);
#else
    config->setMaxWorkspaceSize(1ULL << 30);
#endif

    IOptimizationProfile* profile = builder->createOptimizationProfile();
    ITensor* input_tensor = network->getInput(0);
    const char* input_name = input_tensor->getName();
    Dims4 input_dims(1, 3, INPUT_H, INPUT_W);
    profile->setDimensions(input_name, OptProfileSelector::kMIN, input_dims);
    profile->setDimensions(input_name, OptProfileSelector::kOPT, input_dims);
    profile->setDimensions(input_name, OptProfileSelector::kMAX, input_dims);
    config->addOptimizationProfile(profile);

```

```
    ICudaEngine* engine = builder->buildEngineWithConfig(*network, *config);
    if (!engine) {
        cout << "[错误] TensorRT引擎构建失败！" << endl;
        delete config;
        delete parser;
        delete network;
        delete builder;
        return false;
    }

    IHostMemory* serialized_engine = engine->serialize();
    ofstream ofs(engine_path, ios::binary);
    ofs.write(reinterpret_cast<const char*>(serialized_engine->data()), serialized_engine->size());
    ofs.close();

    delete serialized_engine;
    delete engine;
    delete config;
    delete parser;
    delete network;
    delete builder;

    cout << "[成功] 引擎保存至：" << engine_path << endl;
    return true;
}

// ==================== 引擎加载 ====================
ICudaEngine* loadEngine(const string& engine_path) {
    ifstream ifs(engine_path, ios::binary);
    if (!ifs) {
        cout << "[错误] 无法打开引擎文件：" << engine_path << endl;
        return nullptr;
    }
    ifs.seekg(0, ios::end);
    size_t size = ifs.tellg();
    ifs.seekg(0, ios::beg);
    char* buffer = new char[size];
    ifs.read(buffer, size);
    ifs.close();

    IRuntime* runtime = createInferRuntime(gLogger);
    ICudaEngine* engine = runtime->deserializeCudaEngine(buffer, size);
    delete[] buffer;
    delete runtime;

    if (!engine) {
        cout << "[错误] 引擎加载失败！" << endl;
        return nullptr;
    }

    cout << "[成功] 引擎加载完成！" << endl;
    return engine;
}

// ==================== 推理与计时 ====================
Mat inferSingleImage(ICudaEngine* engine, const Mat& img) {
    IExecutionContext* context = engine->createExecutionContext();
    if (!context) return img.clone();

    const int input_idx = 0;
    const int output_idx = 1;
    context->setOptimizationProfileAsync(0, 0);

    const int input_size = 3 * INPUT_W * INPUT_H;
    const int output_size = NUM_BOXES * BOX_DIM;

    float* host_input = new float[input_size];
    float* host_output = new float[output_size];

    float* device_input = nullptr;
    float* device_output = nullptr;
```

```
    cudaMalloc((void**)&device_input, input_size * sizeof(float));cudaMalloc((void**)&device_output, output_size * sizeof(float));
    // 预处理计时
    double preprocess_start = static_cast<double>(getTickCount());
    preprocessImage(img, host_input);
    double preprocess_end = static_cast<double>(getTickCount());
    double preprocess_time = (preprocess_end - preprocess_start) / getTickFrequency() * 1000;
    cout << "[耗时] 预处理: " << fixed << setprecision(2) << preprocess_time << " 毫秒" << endl;

    cudaMemcpy(device_input, host_input, input_size * sizeof(float), cudaMemcpyHostToDevice);

    void* bindings[2] = { nullptr };
    bindings[input_idx] = device_input;
    bindings[output_idx] = device_output;

    // 推理计时
    double infer_start = static_cast<double>(getTickCount());
    bool success = context->executeV2(bindings);
    double infer_end = static_cast<double>(getTickCount());
    double infer_time = (infer_end - infer_start) / getTickFrequency() * 1000;
    cout << "[耗时] 推理核心: " << fixed << setprecision(2) << infer_time << " 毫秒" << endl;

    if (!success) {
        cout << "[错误] 推理执行失败! " << endl;
        cudaFree(device_input);
        cudaFree(device_output);
        delete[] host_input;
        delete[] host_output;
        delete context;
        return img.clone();
    }

    cudaMemcpy(host_output, device_output, output_size * sizeof(float), cudaMemcpyDeviceToHost);
    vector<DetectResult> results = postprocessYolo26(host_output, img.cols, img.rows);

    Mat result = img.clone();
    for (const auto& res : results) {
        rectangle(result, res.box, Scalar(0, 0, 255), 3);
        string text = res.cls_name + ": " + to_string(res.confidence).substr(0, 4);
        putText(result, text, Point(res.box.x, res.box.y - 10),
            FONT_HERSHEY_SIMPLEX, 1.0f, Scalar(255, 255, 255), 2);
    }

    cudaFree(device_input);
    cudaFree(device_output);
    delete[] host_input;
    delete[] host_output;
    delete context;

    return result;
}

// ==================== 批量推理（不变） ====================
void batchInfer(const string& engine_path, const string& img_dir) {
    ICudaEngine* engine = loadEngine(engine_path);
    if (!engine) return;

    string output_dir = img_dir + "\\output";
    if (!createDirectory(output_dir)) {
        cout << "[错误] 无法创建输出目录: " << output_dir << endl;
        delete engine;
        return;
    }

    vector<string> img_paths = listImageFiles(img_dir);
    if (img_paths.empty()) {
        cout << "[警告] 未找到支持的图片文件: " << img_dir << endl;
        delete engine;
        return;
    }
```

```cpp
    int processed = 0;
    for (const string& path : img_paths) {
        cout << "\n[处理中] " << path << endl;

        Mat img = imread(path, IMREAD_UNCHANGED);
        if (img.empty()) {
            cout << "[警告] 无法读取图片：" << path << endl;
            continue;
        }

        Mat result = inferSingleImage(engine, img);

        size_t pos = path.find_last_of("\\");
        string filename = path.substr(pos + 1);
        string save_path = output_dir + "\\" + filename;
        if (imwrite(save_path, result)) {
            processed++;
            cout << "[成功] 结果保存至：" << save_path << endl;
        }
        else {
            cout << "[错误] 保存图片失败：" << save_path << endl;
        }
    }

    cout << "\n[批量处理完成] 总计：" << img_paths.size()
        << " | 成功：" << processed
        << " | 失败：" << img_paths.size() - processed << endl;
    cout << "[结果保存目录] " << output_dir << endl;

    delete engine;
}

// ==================== 主函数：只保留安全的多线程优化 ====================
int main(int argc, char** argv) {
    // 安全优化：开启OpenCV多线程和指令集，不改动数据，只提速
    cv::setUseOptimized(true);
    cv::setNumThreads(std::min(4, cv::getNumberOfCPUs()));

    setlocale(LC_ALL, "Chinese");
    cout << fixed << setprecision(2);

    if (argc != 4) {
        cout << "==================== YOLO26 TensorRT 推理工具 ====================" << endl;
        cout << "用法1（转换ONNX到引擎）：" << argv[0] << " -s onnx_path engine_path" << endl;
        cout << "用法2（批量推理图片）：" << argv[0] << " -d engine_path img_dir" << endl;
        cout << "==================================================================" << endl;
        return -1;
    }

    string cmd = argv[1];
    if (cmd == "-s") {
        cout << "[开始转换] ONNX → TensorRT引擎" << endl;
        if (!buildEngine(argv[2], argv[3])) {
            cout << "[失败] 引擎转换失败！" << endl;
            return -1;
        }
        cout << "[成功] 引擎转换完成！" << endl;
    }
    else if (cmd == "-d") {
        cout << "[开始推理] 批量处理图片" << endl;
        batchInfer(argv[2], argv[3]);
    }
    else {
        cout << "[错误] 无效命令！仅支持 -s 和 -d" << endl;
        return -1;
    }

    return 0;
}
```

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

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

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！