Opencv-Cuda dnn读模型

1. 下载zlib的DLL，参考官网<https://docs.nvidia.com/deeplearning/cudnn/install-guide/index.html#install-zlib-windows> ，点击ZLIB DLL下载，并将文件移动到opencv的bin目录下。
2. CMakeLists不需要特别修改
3. Yolov5源码

#include <fstream>

#include <opencv2/opencv.hpp>

std::vector<std::string> load\_class\_list()

{

std::vector<std::string> class\_list;

std::ifstream ifs("./classes.txt");

std::string line;

while (getline(ifs, line))

{

class\_list.push\_back(line);

}

return class\_list;

}

void load\_net(cv::dnn::Net &net, bool is\_cuda)

{

auto result = cv::dnn::readNet("./yolov5n.onnx");

if (is\_cuda)

{

std::cout << "Attempty to use CUDA\n";

result.setPreferableBackend(cv::dnn::DNN\_BACKEND\_CUDA);

result.setPreferableTarget(cv::dnn::DNN\_TARGET\_CUDA\_FP16);

}

else

{

std::cout << "Running on CPU\n";

result.setPreferableBackend(cv::dnn::DNN\_BACKEND\_OPENCV);

result.setPreferableTarget(cv::dnn::DNN\_TARGET\_CPU);

}

net = result;

}

const std::vector<cv::Scalar> colors = {cv::Scalar(255, 255, 0), cv::Scalar(0, 255, 0), cv::Scalar(0, 255, 255), cv::Scalar(255, 0, 0)};

const float INPUT\_WIDTH = 640.0;

const float INPUT\_HEIGHT = 640.0;

const float SCORE\_THRESHOLD = 0.2;

const float NMS\_THRESHOLD = 0.4;

const float CONFIDENCE\_THRESHOLD = 0.4;

struct Detection

{

int class\_id;

float confidence;

cv::Rect box;

};

cv::Mat format\_yolov5(const cv::Mat &source) {

int col = source.cols;

int row = source.rows;

int \_max = MAX(col, row);

cv::Mat result = cv::Mat::zeros(\_max, \_max, CV\_8UC3);

source.copyTo(result(cv::Rect(0, 0, col, row)));

return result;

}

void detect(cv::Mat &image, cv::dnn::Net &net, std::vector<Detection> &output, const std::vector<std::string> &className) {

cv::Mat blob;

auto input\_image = format\_yolov5(image);

cv::dnn::blobFromImage(input\_image, blob, 1./255., cv::Size(INPUT\_WIDTH, INPUT\_HEIGHT), cv::Scalar(), true, false);

net.setInput(blob);

std::vector<cv::Mat> outputs;

net.forward(outputs, net.getUnconnectedOutLayersNames());

float x\_factor = input\_image.cols / INPUT\_WIDTH;

float y\_factor = input\_image.rows / INPUT\_HEIGHT;

float \*data = (float \*)outputs[0].data;

const int dimensions = 85;

const int rows = 25200;

std::vector<int> class\_ids;

std::vector<float> confidences;

std::vector<cv::Rect> boxes;

for (int i = 0; i < rows; ++i) {

float confidence = data[4];

if (confidence >= CONFIDENCE\_THRESHOLD) {

float \* classes\_scores = data + 5;

cv::Mat scores(1, className.size(), CV\_32FC1, classes\_scores);

cv::Point class\_id;

double max\_class\_score;

minMaxLoc(scores, 0, &max\_class\_score, 0, &class\_id);

if (max\_class\_score > SCORE\_THRESHOLD) {

confidences.push\_back(confidence);

class\_ids.push\_back(class\_id.x);

float x = data[0];

float y = data[1];

float w = data[2];

float h = data[3];

int left = int((x - 0.5 \* w) \* x\_factor);

int top = int((y - 0.5 \* h) \* y\_factor);

int width = int(w \* x\_factor);

int height = int(h \* y\_factor);

boxes.push\_back(cv::Rect(left, top, width, height));

}

}

data += 85;

}

std::vector<int> nms\_result;

cv::dnn::NMSBoxes(boxes, confidences, SCORE\_THRESHOLD, NMS\_THRESHOLD, nms\_result);

for (int i = 0; i < nms\_result.size(); i++) {

int idx = nms\_result[i];

Detection result;

result.class\_id = class\_ids[idx];

result.confidence = confidences[idx];

result.box = boxes[idx];

output.push\_back(result);

}

}

int main()

{

std::vector<std::string> class\_list = load\_class\_list();

cv::Mat frame;

cv::VideoCapture capture(0);

if (!capture.isOpened())

{

std::cerr << "Error opening video file\n";

return -1;

}

bool is\_cuda = true;

cv::dnn::Net net;

load\_net(net, is\_cuda);

auto start = std::chrono::high\_resolution\_clock::now();

int frame\_count = 0;

float fps = -1;

int total\_frames = 0;

while (true)

{

capture.read(frame);

if (frame.empty())

{

std::cout << "End of stream\n";

break;

}

std::vector<Detection> output;

detect(frame, net, output, class\_list);

frame\_count++;

total\_frames++;

int detections = output.size();

for (int i = 0; i < detections; ++i)

{

auto detection = output[i];

auto box = detection.box;

auto classId = detection.class\_id;

const auto color = colors[classId % colors.size()];

cv::rectangle(frame, box, color, 3);

cv::rectangle(frame, cv::Point(box.x, box.y - 20), cv::Point(box.x + box.width, box.y), color, cv::FILLED);

cv::putText(frame, class\_list[classId].c\_str(), cv::Point(box.x, box.y - 5), cv::FONT\_HERSHEY\_SIMPLEX, 0.5, cv::Scalar(0, 0, 0));

}

if (frame\_count >= 30)

{

auto end = std::chrono::high\_resolution\_clock::now();

fps = frame\_count \* 1000.0 / std::chrono::duration\_cast<std::chrono::milliseconds>(end - start).count();

frame\_count = 0;

start = std::chrono::high\_resolution\_clock::now();

}

if (fps > 0)

{

std::ostringstream fps\_label;

fps\_label << std::fixed << std::setprecision(2);

fps\_label << "FPS: " << fps;

std::string fps\_label\_str = fps\_label.str();

cv::putText(frame, fps\_label\_str.c\_str(), cv::Point(10, 25), cv::FONT\_HERSHEY\_SIMPLEX, 1, cv::Scalar(0, 0, 255), 2);

}

cv::imshow("output", frame);

if (cv::waitKey(1) != -1)

{

capture.release();

std::cout << "finished by user\n";

break;

}

}

std::cout << "Total frames: " << total\_frames << "\n";

return 0;

}