#include "ocr.h"

#include <iostream>

#include <string.h>

using namespace std;

using namespace cv;

//模型载入，模型初始化

void FuncOnnx\_ocr::loadFuncOnnx\_ocr(string path)

{

int i;

input\_node\_dims.clear();

input\_node\_names.clear();

output\_node\_names.clear();

m\_env1 = new Ort::Env(ORT\_LOGGING\_LEVEL\_ERROR, "onnx1");

session\_options1.SetIntraOpNumThreads(1);

session\_options1.SetGraphOptimizationLevel(GraphOptimizationLevel::ORT\_ENABLE\_EXTENDED);

const ORTCHAR\_T\* modePath;// = nullptr;

//#ifdef \_WIN32

// modePath = reinterpret\_cast<const wchar\_t \*>(path.c\_str());

//#else

// //modePath = path.constData();

//#endif

// stringtemp;

//const char\* modePath = path.c\_str();

//cout << path.c\_str(); //模型路径

wstring widstr = wstring(path.begin(), path.end());

modePath = widstr.c\_str(); //模型路径地址

m\_session1 = new Ort::Session(\*m\_env1, modePath, session\_options1); //模型初始化

Ort::AllocatorWithDefaultOptions allocator;

num\_input\_nodes = m\_session1->GetInputCount(); //模型输入节点

num\_output\_nodes = m\_session1->GetOutputCount(); //模型输出节点

input\_tensor\_size = 0;

for (i = 0; i < num\_output\_nodes; i++)

{

char\* output\_name = m\_session1->GetOutputName(i, allocator);

output\_node\_names.push\_back(output\_name);

}

//模型输入初始化

for (i = 0; i < num\_input\_nodes; i++)

{

char\* input\_name = m\_session1->GetInputName(i, allocator);

input\_node\_names.push\_back(input\_name);

Ort::TypeInfo type\_info = m\_session1->GetInputTypeInfo(i);

auto tensor\_info = type\_info.GetTensorTypeAndShapeInfo();

ONNXTensorElementDataType type = tensor\_info.GetElementType();

input\_node\_dims = tensor\_info.GetShape();

input\_tensor\_size = 1;

for (int j = 0; j < input\_node\_dims.size(); j++)

{

if (input\_node\_dims[j] > 0) {

input\_tensor\_size \*= input\_node\_dims[j];

}

if (j == 2) m\_nImageHeight = input\_node\_dims[j];

if (j == 3) m\_nImageWidth = input\_node\_dims[j];

}

}

}

//图片预测

void FuncOnnx\_ocr::ocr\_predict(Mat image, string & predict\_date)

{

vector<float> input\_tensor\_values(32 \* 280);

//图片像素归一化，整理成向量形式

unsigned int volChl = 32 \* 280;

for (unsigned j = 0; j < volChl; ++j)

{

/\*input\_tensor\_values[c\*volChl + j] = (((image.data[j \* 3 + c]) - 127.5)\*0.0078125);\*/

input\_tensor\_values[j] = ((image.data[j]) / 255.);

}

//进行图片预测

auto memory\_info = Ort::MemoryInfo::CreateCpu(OrtArenaAllocator, OrtMemTypeDefault); //100ms

Ort::Value input\_tensor = Ort::Value::CreateTensor<float>(memory\_info, input\_tensor\_values.data(), input\_tensor\_size, input\_node\_dims.data(), input\_node\_dims.size());

assert(input\_tensor.IsTensor());

auto outTensor = m\_session1->Run(Ort::RunOptions{ nullptr }, input\_node\_names.data(), &input\_tensor, num\_input\_nodes, output\_node\_names.data(), num\_output\_nodes);

assert(outTensor.front().IsTensor());

//预测结果概率处理

float findf = 0.00;

int findi = -1;

int findj = 0;

int findi\_bak = 0;

int nRet = 0;

//std::vector<Ort::Value> outTensor;

char sDict[] = " 1234567890qwerTYUIOPASDFGHtyuiopasdfghjklzxcvbnmQWERJKLZXCVBNM/-!.";

//if (doDetectOnnx(src, outTensor) != 1) return nRet;

assert(outTensor.front().IsTensor());

for (int i = 0; i<outTensor.size(); i++)

{

auto tensorOutPut\_info = outTensor[i].GetTensorTypeAndShapeInfo();

float\* floatarr = outTensor[i].GetTensorMutableData<float>();

std::vector<int64\_t> output\_tensor\_dims;

output\_tensor\_dims = tensorOutPut\_info.GetShape();

if (output\_tensor\_dims.size() == 3)

{

findj = 0;

for (int j = 0; j < output\_tensor\_dims[0]; j++)

{

for (int k = 0; k < output\_tensor\_dims[1]; k++)

{

findf = 0.00;

findi = -1;

for (int l = 0; l < output\_tensor\_dims[2]; l++)

{

if (floatarr[j \* output\_tensor\_dims[1] \* output\_tensor\_dims[2] + l] > findf)

{

findf = floatarr[j \* output\_tensor\_dims[1] \* output\_tensor\_dims[2] + l];

findi = l;

}

}

if (findj == j - 1 && findi == findi\_bak)

{

//

}

else

{

if (findi > 0)

{

//\*pcOutOcr++ = sDict[findi];

predict\_date += sDict[findi];

//cout << sDict[findi] ;

nRet = 1;

}

}

findi\_bak = findi;

findj = j;

}

}

}

}

//return nRet;

}

//void main() {

// FuncOnnx\_ocr tmp;

// tmp.loadFuncOnnx\_ocr("C://Users//pc//Desktop//img//torch\_model\_train\_Acc100.000\_test\_Acc99.567\_2\_3.onnx");

// Mat img; //临时文件

// string img\_path = R"(C:\Users\pc\Desktop\img\1.jpg)";

// img = imread(img\_path, 0);

// string predict\_date;

// tmp.ocr\_predict(img, predict\_date);//图片预测

// cout << predict\_date << endl;

// system("pause");

// return;

//}