//\*\* Image classification

#include <vector>

#include <memory>

#include <string>

#include <opencv2/opencv.hpp>

#include <inference\_engine.hpp>

using namespace std;

using namespace InferenceEngine;

const char \*labels[13] = {"airplane", "automobile", "bird", "cat", "deer", "dog", "frog", "horse", "sheep", "truck", "rock", "scissors", "paper"};

int main(int argc, char \*argv[]) {

// ---------------------Load A Plugin for Inference Engine-----------------------------------------

InferenceEngine::PluginDispatcher dispatcher({""});

InferencePlugin plugin(dispatcher.getSuitablePlugin(TargetDevice::eCPU));

// --------------------Load IR Generated by ModelOptimizer (.xml and .bin files)------------------------

CNNNetReader network\_reader;

network\_reader.ReadNetwork("/home/intel/my\_model/cifar.xml");

network\_reader.ReadWeights("/home/intel/my\_model/cifar.bin");

network\_reader.getNetwork().setBatchSize(1);

CNNNetwork network = network\_reader.getNetwork();

// -----------------------------Prepare input blobs-----------------------------------------------------

auto input\_info = network.getInputsInfo().begin()->second;

auto input\_name = network.getInputsInfo().begin()->first;

input\_info->setPrecision(Precision::U8);

// ---------------------------Prepare output blobs------------------------------------------------------

auto output\_info = network.getOutputsInfo().begin()->second;

auto output\_name = network.getOutputsInfo().begin()->first;

output\_info->setPrecision(Precision::FP32);

// -------------------------Loading model to the plugin and then infer----------------------------------

auto executable\_network = plugin.LoadNetwork(network, {});

auto infer\_request = executable\_network.CreateInferRequest();

auto input = infer\_request.GetBlob(input\_name);

auto input\_data = input->buffer().as<PrecisionTrait<Precision::U8>::value\_type\*>();

/\* Copying data from image to the input blob \*/

cv::Mat ori\_image, infer\_image;

ori\_image = cv::imread("/home/intel/sample/input.jpg");

cv::resize(ori\_image, infer\_image, cv::Size(input\_info->getDims()[0], input\_info->getDims()[1]));

//cv::namedWindow("title", cv::WINDOW\_NORMAL);

//cv::resizeWindow("title", 600,600);

//cv::imshow("title", infer\_image);

//cv::waitKey(0);

size\_t channels\_number = input->dims()[2];

size\_t image\_size = input->dims()[1] \* input->dims()[0];

for (size\_t pid = 0; pid < image\_size; ++pid) {

for (size\_t ch = 0; ch < channels\_number; ++ch) {

input\_data[ch \* image\_size + pid] = infer\_image.at<cv::Vec3b>(pid)[ch];

}

}

/\* Running the request synchronously \*/

infer\_request.Infer();

// ---------------------------Postprocess output blobs--------------------------------------------------

auto output = infer\_request.GetBlob(output\_name);

auto output\_data = output->buffer().as<PrecisionTrait<Precision::FP32>::value\_type\*>();

vector<unsigned> results;

/\* This is to sort output probabilities and put it to results vector \*/

TopResults(13, \*output, results);

cout << endl << "Top 13 results:" << endl << endl;

for (size\_t id = 0; id < 13; ++id) {

cout.precision(7);

auto result = output\_data[results[id]];

cout << left << fixed << result << " label #" << results[id] << ", " << labels[results[id]] << endl;

}

return EXIT\_SUCCESS;

}