//\*\* Image classification

#include <vector>

#include <memory>

#include <string>

#include <opencv2/opencv.hpp>

#include <inference\_engine.hpp>

using namespace std;

using namespace InferenceEngine;

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/test.xml");

network\_reader.ReadWeights("/home/intel/my\_model/test.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/my\_model/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(10, \*output, results);

cout << endl << "Top 10 results:" << endl << endl;

for (size\_t id = 0; id < 10; ++id) {

cout.precision(7);

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

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

}

return EXIT\_SUCCESS;

}

// \*\* Object Detection

#include <vector>

#include <memory>

#include <string>

#include <opencv2/opencv.hpp>

#include <inference\_engine.hpp>

using namespace std;

using namespace InferenceEngine;

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

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

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

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

// +++++++++++++++++++ Load layer extension for CPU ++++++++++++++++++++

auto extension\_ptr = make\_so\_pointer<InferenceEngine::IExtension>("/opt/intel/computer\_vision\_sdk/deployment\_tools/inference\_engine/lib/ubuntu\_16.04/intel64/libcpu\_extension.so");

plugin.AddExtension(extension\_ptr);

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

CNNNetReader network\_reader;

network\_reader.ReadNetwork("/home/intel/my\_model/ssd300.xml");

network\_reader.ReadWeights("/home/intel/my\_model/ssd300.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/my\_model/cars\_768x768.jpg");

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

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

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

//cv::imshow("infer", 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);

// +++++++++++++ check proposal count and objectsize of each proposal +++++++++++

const int maxProposalCount = output->dims()[1];

const int objectSize = output->dims()[0];

const Blob::Ptr output\_blob = output;

const float\* detection = static\_cast<PrecisionTrait<Precision::FP32>::value\_type\*>(output\_blob->buffer());

/\* Each detection has image\_id that denotes processed image \*/

for (int curProposal = 0; curProposal < maxProposalCount; curProposal++) {

float image\_id = detection[curProposal \* objectSize + 0];

float label = detection[curProposal \* objectSize + 1];

float confidence = detection[curProposal \* objectSize + 2];

/\* CPU and GPU plugins have difference in DetectionOutput layer, so we need both checks \*/

if (image\_id < 0 || confidence == 0) {

continue;

}

float xmin = detection[curProposal \* objectSize + 3] \* ori\_image.size().width;

float ymin = detection[curProposal \* objectSize + 4] \* ori\_image.size().height;

float xmax = detection[curProposal \* objectSize + 5] \* ori\_image.size().width;

float ymax = detection[curProposal \* objectSize + 6] \* ori\_image.size().height;

cout << "[" << curProposal << "," << label << "] element, prob = " << confidence << " (" << xmin << "," << ymin << ")-(" << xmax << "," << ymax << ")" << " batch id : " << image\_id;

if (confidence > 0.5) {

/\*\* Drawing only objects with >50% probability \*\*/

string stag;

ostringstream ostr;

ostr << label << ", " << fixed << setw(4) << setprecision(2) << confidence;

string header = ostr.str();

rectangle(ori\_image, cv::Point((int)xmin, (int)ymin), cv::Point((int)xmax, (int)ymax), cv::Scalar(0, 230, 0), 2, 4);

rectangle(ori\_image, cv::Point((int)xmin-1, (int)ymin-16), cv::Point((int)xmax+1, (int)ymin), cv::Scalar(0, 230, 0), CV\_FILLED, cv::LINE\_8, 0);

putText(ori\_image, header, cv::Point((int)xmin+4, (int)ymin), cv::FONT\_HERSHEY\_TRIPLEX, .5, cv::Scalar(70,70,70));

putText(ori\_image, header, cv::Point((int)xmin+3, (int)ymin-1), cv::FONT\_HERSHEY\_TRIPLEX, .5, cv::Scalar(0,0,0));

cout << " WILL BE PRINTED!";

}

cout << endl;

}

cv::imshow("result", ori\_image);

cv::waitKey(0);

return 0;

}

//\*\* Object Detection streaming version - CPU

#include <vector>

#include <memory>

#include <string>

#include <opencv2/opencv.hpp>

#include <inference\_engine.hpp>

using namespace std;

using namespace cv;

using namespace InferenceEngine;

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

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

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

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

// +++++++++++++++++++ Load layer extension for CPU ++++++++++++++++++++

auto extension\_ptr = make\_so\_pointer<InferenceEngine::IExtension>("/opt/intel/computer\_vision\_sdk/deployment\_tools/inference\_engine/lib/ubuntu\_16.04/intel64/libcpu\_extension.so");

plugin.AddExtension(extension\_ptr);

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

CNNNetReader network\_reader;

network\_reader.ReadNetwork("/home/intel/my\_model/ssd300.xml");

network\_reader.ReadWeights("/home/intel/my\_model/ssd300.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\*>();

VideoCapture cap("/dev/video0"); //<-- usb camera or can be any video stream

if(!cap.isOpened())

{

cout << "can't open input device" << endl;

return 1;

}

Mat ori\_image, infer\_image;

while(1)

{

// read one frame from input stream

cap.read(ori\_image);

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

//namedWindow("infer", WINDOW\_NORMAL);

//resizeWindow("infer", 600,600);

//imshow("infer", infer\_image);

//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<Vec3b>(pid)[ch];

}

}

/\* Running the request synchronously \*/

infer\_request.Infer();

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

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

// +++++++++++++ check proposal count and objectsize of each proposal +++++++++++

const int maxProposalCount = output->dims()[1];

const int objectSize = output->dims()[0];

const Blob::Ptr output\_blob = output;

const float\* detection = static\_cast<PrecisionTrait<Precision::FP32>::value\_type\*>(output\_blob->buffer());

/\* Each detection has image\_id that denotes processed image \*/

for (int curProposal = 0; curProposal < maxProposalCount; curProposal++) {

float image\_id = detection[curProposal \* objectSize + 0];

float label = detection[curProposal \* objectSize + 1];

float confidence = detection[curProposal \* objectSize + 2];

/\* CPU and GPU plugins have difference in DetectionOutput layer, so we need both checks \*/

if (image\_id < 0 || confidence == 0) {

continue;

}

float xmin = detection[curProposal \* objectSize + 3] \* ori\_image.size().width;

float ymin = detection[curProposal \* objectSize + 4] \* ori\_image.size().height;

float xmax = detection[curProposal \* objectSize + 5] \* ori\_image.size().width;

float ymax = detection[curProposal \* objectSize + 6] \* ori\_image.size().height;

cout << "[" << curProposal << "," << label << "] element, prob = " << confidence << " (" << xmin << "," << ymin << ")-(" << xmax << "," << ymax << ")" << " batch id : " << image\_id;

if (confidence > 0.5) {

/\*\* Drawing only objects with >50% probability \*\*/

string stag;

ostringstream ostr;

ostr << label << ", " << fixed << setw(4) << setprecision(2) << confidence;

string header = ostr.str();

rectangle(ori\_image, Point((int)xmin, (int)ymin), Point((int)xmax, (int)ymax), Scalar(0, 230, 0), 2, 4);

rectangle(ori\_image, Point((int)xmin-1, (int)ymin-16), Point((int)xmax+1, (int)ymin), Scalar(0, 230, 0), CV\_FILLED, LINE\_8, 0);

putText(ori\_image, header, Point((int)xmin+4, (int)ymin), FONT\_HERSHEY\_TRIPLEX, .5, Scalar(70,70,70));

putText(ori\_image, header, Point((int)xmin+3, (int)ymin-1), FONT\_HERSHEY\_TRIPLEX, .5, Scalar(0,0,0));

cout << " WILL BE PRINTED!";

}

cout << endl;

}

imshow("result", ori\_image);

if (waitKey(1) >= 13) // enterkey

break;

}

return 0;

}

//\*\* Object Detection streaming version – with GPU

#include <vector>

#include <memory>

#include <string>

#include <opencv2/opencv.hpp>

#include <inference\_engine.hpp>

using namespace std;

using namespace cv;

using namespace InferenceEngine;

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

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

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

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

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

CNNNetReader network\_reader;

network\_reader.ReadNetwork("/home/intel/my\_model/SSD\_GoogleNet\_v2\_fp16.xml");

network\_reader.ReadWeights("/home/intel/my\_model/SSD\_GoogleNet\_v2\_fp16.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\*>();

VideoCapture cap("/dev/video0"); //<-- usb camera or can be any video stream

if(!cap.isOpened())

{

cout << "can't open input device" << endl;

return 1;

}

Mat ori\_image, infer\_image;

while(1)

{

// read one frame from input stream

cap.read(ori\_image);

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

//namedWindow("infer", WINDOW\_NORMAL);

//resizeWindow("infer", 600,600);

//imshow("infer", infer\_image);

//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<Vec3b>(pid)[ch];

}

}

/\* Running the request synchronously \*/

infer\_request.Infer();

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

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

// +++++++++++++ check proposal count and objectsize of each proposal +++++++++++

const int maxProposalCount = output->dims()[1];

const int objectSize = output->dims()[0];

const Blob::Ptr output\_blob = output;

const float\* detection = static\_cast<PrecisionTrait<Precision::FP32>::value\_type\*>(output\_blob->buffer());

/\* Each detection has image\_id that denotes processed image \*/

for (int curProposal = 0; curProposal < maxProposalCount; curProposal++) {

float image\_id = detection[curProposal \* objectSize + 0];

float label = detection[curProposal \* objectSize + 1];

float confidence = detection[curProposal \* objectSize + 2];

/\* CPU and GPU plugins have difference in DetectionOutput layer, so we need both checks \*/

if (image\_id < 0 || confidence == 0) {

continue;

}

float xmin = detection[curProposal \* objectSize + 3] \* ori\_image.size().width;

float ymin = detection[curProposal \* objectSize + 4] \* ori\_image.size().height;

float xmax = detection[curProposal \* objectSize + 5] \* ori\_image.size().width;

float ymax = detection[curProposal \* objectSize + 6] \* ori\_image.size().height;

cout << "[" << curProposal << "," << label << "] element, prob = " << confidence << " (" << xmin << "," << ymin << ")-(" << xmax << "," << ymax << ")" << " batch id : " << image\_id;

if (confidence > 0.5) {

/\*\* Drawing only objects with >50% probability \*\*/

string stag;

ostringstream ostr;

ostr << label << ", " << fixed << setw(4) << setprecision(2) << confidence;

string header = ostr.str();

rectangle(ori\_image, Point((int)xmin, (int)ymin), Point((int)xmax, (int)ymax), Scalar(0, 230, 0), 2, 4);

rectangle(ori\_image, Point((int)xmin-1, (int)ymin-16), Point((int)xmax+1, (int)ymin), Scalar(0, 230, 0), CV\_FILLED, LINE\_8, 0);

putText(ori\_image, header, Point((int)xmin+4, (int)ymin), FONT\_HERSHEY\_TRIPLEX, .5, Scalar(70,70,70));

putText(ori\_image, header, Point((int)xmin+3, (int)ymin-1), FONT\_HERSHEY\_TRIPLEX, .5, Scalar(0,0,0));

cout << " WILL BE PRINTED!";

}

cout << endl;

}

imshow("result", ori\_image);

if (waitKey(1) >= 13) // enterkey

break;

}

return 0;

}

//\*\* Object Detection streaming version – with Movidius

//\*\* Everything is same as GPU code except “**eMYRIAD**”

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

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

InferencePlugin plugin(dispatcher.getSuitablePlugin(TargetDevice::**eMYRIAD**));