RBOT 230 Robot Sensing and Perception

Assignment 2 (Week 6): Object Detection using CNNs

By Salinee Kingbaisomboon
Sage ID: 20801897

GOALS

The original goal of this week's assignment is to use a Realsense sensor located in a fixed position to detect the model vehicle (such as toy car) using CNNs from DLib.

However, I'm not able to detect such a toy car successfully with the **pre-trained models** from DLib (**mmod_front_and_rear_end_vehicle_detector** & **mmod_rear_end_vehicle_detector**) [1]. Both models didn't detect a toy car at all since it's not a real car.

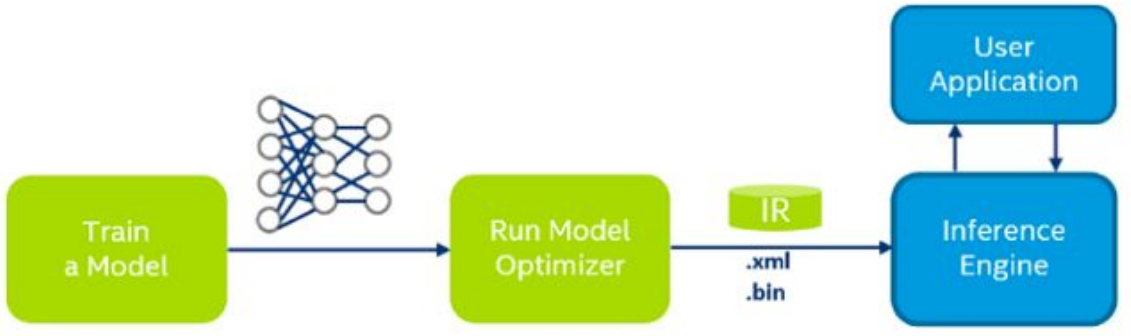
Therefore, I decided to do the **Face Detection** instead with both **DLib** and **OpenVINO** in order to compare the performance for both frameworks.

Why using OpenVINO

The main reason why I decided to use this framework is because my computer's display card is an Intel graphics card which doesn't support CUDA. So, it is a great idea to utilize OpenVINO.

I choose to install **OpenVINO™ toolkit version 2019_R3.1** [2] which is more stable and works well on my computer. Any newer version of OpenVINO doesn't work when I try to run the Visual Studio projects on the sample code directly.

Below is how **OpenVINO** uses pre-trained neural network models as the **Intermediate Representation** (in the form of .xml and .bin files) which OpenVINO will use this representation to utilize the *Inference Engine capabilities* out of the Intel graphic card.



Model Optimizer produces an Intermediate Representation (IR) of the network, which can be read, loaded, and inferred with the Inference Engine. The Inference Engine API offers a unified API across a number of supported Intel® platforms. The Intermediate Representation is a pair of files describing the model:

- `.xml` - Describes the network topology
- `.bin` - Contains the weights and biases binary data.

Image Credit: Intel

STEPS IN THE CODE (DLib implementation)

1. Implement Dlib Library in order to implement the deep learning model.
2. Implement the ***frontal face detector*** class for face detection.
3. Implement the ***shape predictor*** class for face landmark detection .
4. Using Realsense Color Sensor.

SPECIFICATIONS OF THE SUBMITTED PACKAGE

There are three main artifacts which will be described as following:

Assignment_3_Object_Detection.pdf

This is the documentation of this assignment.

SalineeAssignment3 solution (Visual Studio solution)

1. **DLibFaceDetection.cpp**

Below is the complete code on the file. The code will read the live frame from the Realsense camera and perform the face and its landmarks detection using the DLib library and pre-trained models.[1]

```
#include <iostream>

#include <opencv2/core.hpp>
#include <opencv2/video.hpp>
#include <opencv2/highgui.hpp>

#include <dlib/opencv.h>
#include <dlib/dnn.h>
#include <dlib/image_io.h>
#include <dlib/gui_widgets.h>
#include <dlib/image_processing.h>
#include <dlib/image_processing/frontal_face_detector.h>
#include <dlib/image_processing/render_face_detections.h>

#include <librealsense2/rsutil.h>
#include <librealsense2/rs.hpp>

using namespace cv;
using namespace std;
using namespace dlib;

// The human face detector network (source:
https://github.com/davisking/dlib-models/blob/master/mmod_human_face_
detector.dat.bz2)
// The shape predictor face landmark network (source:
https://github.com/davisking/dlib-models/blob/master/shape_predictor_
68_face_landmarks.dat.bz2)
template <long num_filters, typename SUBNET> using con5d =
con<num_filters, 5, 5, 2, 2, SUBNET>;
template <long num_filters, typename SUBNET> using con5 =
con<num_filters, 5, 5, 1, 1, SUBNET>;

template <typename SUBNET> using downsampler = relu<affine<con5d<32,
relu<affine<con5d<32, relu<affine<con5d<16, SUBNET>>>>>>>>>;
template <typename SUBNET> using rcon5 = relu<affine<con5<45,
SUBNET>>>;

using net_type = loss_mmod<con<1, 9, 9, 1, 1,
rcon5<rcon5<rcon5<downsampler<input_rgb_image_pyramid<pyramid_down<6>
>>>>>>>;

int main() try
{
#pragma region Realsense Setup

     rs2::log_to_console(RS2_LOG_SEVERITY_ERROR);

     // Declare RealSense pipeline, encapsulating the actual device
```

```
and sensors
      rs2::pipeline pipe;

      // Create a configuration for configuring the pipeline with a
non default profile
      rs2::config cfg;

      // Add desired streams to configuration
      cfg.enable_stream(RS2_STREAM_COLOR, 640, 480, RS2_FORMAT_BGR8,
30);

      // Start streaming with default recommended configuration
      pipe.start(cfg);

      // Camera warmup - dropping several first frames to let
auto-exposure stabilize
      for (int i = 0; i < 30; i++)
      {
            pipe.wait_for_frames();
      }

#pragma endregion

#pragma region Defined Initial Settings

      image_window win1;
      image_window win2;
      cv::TickMeter timer;

      // Load face detection and pose estimation models
      net_type net;
      deserialize("../mmod_human_face_detector.dat") >> net;

      // Load Load face detection and pose estimation models for
point landmarking model
      frontal_face_detector detector = get_frontal_face_detector();
      shape_predictor pose_model;
      deserialize("shape_predictor_68_face_landmarks.dat") >>
pose_model;

#pragma endregion

      // Grab and process frames until the main window is closed by
the user
      while (!win1.is_closed() && !win2.is_closed())
      {
            // Full Timer Resolution
            timer.reset();
```

```
            timer.start();

            // Wait for next set of frames from the camera
            rs2::frameset data = pipe.wait_for_frames();
            rs2::frame color = data.get_color_frame();

            // Query frame size (width and height)
            const int w = color.as<rs2::video_frame>().get_width();
            const int h = color.as<rs2::video_frame>().get_height();

            // Create OpenCV matrix of size (w,h) from the colorized
depth data
            Mat liveFrame(Size(w, h), CV_8UC3,
(void*)color.get_data(), Mat::AUTO_STEP);

            // Convert OpenCV image format to Dlib's image format
            cv_image<bgr_pixel> dlibIm(liveFrame);
            matrix<rgb_pixel> matrix;
            assign_image(matrix, dlibIm);

            // Run the detector on the image and show the output
            auto dets = net(matrix);
            win1.clear_overlay();
            win1.set_image(matrix);
            for (auto&& d : dets)
            {
                  // Measure performance
                  timer.stop();
                  cv::String strTime1 = cv::format("Processed Time:
%4.3f msec", timer.getTimeMilli());

                  win1.add_overlay(d, rgb_pixel(255, 0, 0), strTime1);
            }

            // Detect faces
            std::vector<dlib::rectangle> faces = detector(dlibIm);

            // Find the pose of each face.
            std::vector<full_object_detection> shapes;
            for (unsigned long i = 0; i < faces.size(); ++i)
                  shapes.push_back(pose_model(dlibIm, faces[i]));

            // Display it all on the screen
            win2.clear_overlay();
            win2.set_image(dlibIm);
            win2.add_overlay(render_face_detections(shapes));
      }
```

```
	return EXIT_SUCCESS;
}
catch (std::exception& e)
{
	cout << e.what() << endl;
}
```

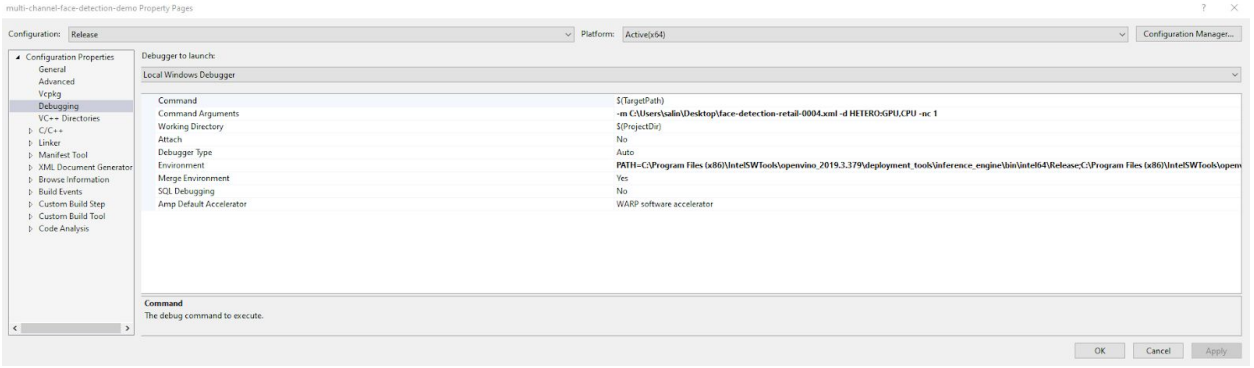
2. **OpenVINOFaceDetection.cpp**

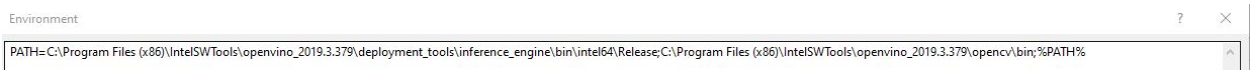
The code is this file was adapted from the **Multi-Channel Face Detection C++ Demo** from the OpenVINO™ Toolkit's website[3].

In order to run this code, first we need to run the following cmd command to set sets all necessary environment variables:

```
C:\Program Files (x86)\IntelSWTools\openvino\bin\setupvars.bat
```

To debug or run the code on Windows in Microsoft Visual Studio, make sure to properly configure **Debugging environment** settings for the Debug and Release configurations. Below is the screenshot of how to set up this path in Visual Studio.





After that, I need to set the following command arguments in the Visual Studio. For the model (-m), it is required. I downloaded the IR files for face detection from Intel's site[5]. For the device (-d), it is an optional but I do want to test the **GPU** and **CPU** to see how they're different.

```
-m C:\Users\salin\Desktop\face-detection-retail-0004.xml -d
HETERO:GPU,CPU -nc 1
```

Below is the list of support device types that the Inference Engine of OpenVINO can infer models in different formats and plugins:[4]

PLUGIN	DEVICE TYPES
GPU plugin	Intel® Processor Graphics, including Intel® HD Graphics and Intel® Iris® Graphics
CPU plugin	Intel® Xeon® with Intel® AVX2 and AVX512, Intel® Core™ Processors with Intel® AVX2, Intel® Atom® Processors with Intel® SSE
FPGA plugin (available in the Intel® Distribution of OpenVINO™ toolkit)	Intel® Vision Accelerator Design with an Intel® Arria 10 FPGA (Speed Grade 1), Intel® Vision Accelerator Design with an Intel® Arria 10 FPGA (Speed Grade 2), Intel® Programmable Acceleration Card with Intel® Arria® 10 GX FPGA
VPU plugins (available in the Intel® Distribution of OpenVINO™ toolkit)	Intel® Movidius™ Neural Compute Stick powered by the Intel® Movidius™ Myriad™ 2, Intel® Neural Compute Stick 2 powered by the Intel® Movidius™ Myriad™ X, Intel® Vision Accelerator Design with Intel® Movidius™ VPUs
GNA plugin (available in the Intel® Distribution of OpenVINO™ toolkit)	Intel® Speech Enabling Developer Kit, Amazon Alexa* Premium Far-Field Developer Kit, Intel® Pentium® Silver processor J5005, Intel® Celeron® processor J4005, Intel® Core™ i3-8121U processor
Multi-Device plugin	Multi-Device plugin enables simultaneous inference of the same network on several Intel® devices in parallel
Heterogeneous plugin	Heterogeneous plugin enables automatic inference splitting between several Intel® devices (for example if a device doesn't support certain layers).

Pre-Trained Model Files

Below are the list of the pre-trained models used in both **DLibFaceDetection.cpp** and **OpenVINOFaceDetection.cpp** respectively.

1. Mmod_human_face_detector.dat (for **DLibFaceDetection.cpp)**[1]
2. Shape_predictor_68_face_landmarks.dat (for **DLibFaceDetection.cpp)**[1]

3. Face-detection-retail-0004.bin (for **OpenVINOFaceDetection.cpp)**[5]
4. Face-detection-retail-0004.xml (for **OpenVINOFaceDetection.cpp)**[5]

Result Video Files

Below is a list of video files for DLib and OpenVINO's implementation. Due to their size, I need to zipped them in order to be able to upload to Github's repository.

1. DLib Face Detection.zip
2. OpenVINO Face Detection - CPU.zip
3. OpenVINO Face Detection - GPU.zip

Based on the quality and speed from each video file, I can say that using OpenVINO with the selected GPU as a device performs the best out of the three. Using DLib's face detection deep learning model will produce the very jitter lag video which makes it almost impossible to run this in the production. (**Maybe because the code didn't utilize a GPU like OpenVINO**).

You will find the source code and related video files on the following github.

GitHub: https://github.com/salineeKing/RBOT230

REFERENCES

[1] https://github.com/davisking/dlib-models

[2] https://docs.openvinotoolkit.org/2019_R3.1/_docs_install_guides_installing_openvino_windows.html

[3] https://docs.openvinotoolkit.org/2020.1/_demos_multi_channel_face_detection_demo_README.html

[4] https://docs.openvinotoolkit.org/2019_R3.1/_docs_IE_DG_supported_plugins_Supported_Devices.html

[5] https://download.01.org/opencv/2019/open_model_zoo/R3/20190905_163000_models_bin/face-detection-retail-0004/FP16/