【Intel OpenVINO 教学】GStreamer 串流视频智能分析不再慢吞吞—看 Intel OpenVINO DL Stream 如何加速视频推理推理

欧尼克斯实境互动工作室 许哲豪 2021/02/08

读取/储存串流图像文件 图像解码/编码/转换	读取/储存串流图像文件 图像解码/编码/转码
影像推理组件 影像分类、物件检测	图像推理组件 图像分类、物件检测
视频分析管道	视频分析管道

随着人工智能兴起，利用“深度学习”技术进行计算机视觉工作已是很普遍的应用，如图像分类、对象检测等。通常只需将单张图像送入训练好的模型中进行推理即可得到输出结果，但当遇到视频文件或串流视频时，逐格(by Frame)图像分析就变得很没效率，因为帧间时间差距过短（通常为 1/30 ~ 1/60 秒），场景中的对象位移量（变形量）可能过小，导致帧间得到几乎相同的计算结果（对象位置、尺寸及分类），浪费计算资源，也让系统看起来反应过慢。

为了使视频文件及串流视频在分析上能加快反应，Intel OpenVINO Toolkit（以下简称 **OpenVINO**）整合了 GStreamer、OpenCV 并提供 DL Streamer - GStreamer Video Analytics (GVA) Plugin，让用户可以更灵活地进行设置，使其在视频（连续图像）分析上更具效率。接下来就帮大家介绍“**什么是串流视频？**”、“**什么是 GStreamer？**”、“**如何安装 OpenVINO DL Streamer 环境**”、“**串流视频分析及优化流程**”，最后再以一个实时追踪车辆、行人的范例来介绍如何使用命令行操作，希望能帮大家对 DL Streamer 有更多认识。

*什么是串流视频？什么是 GStreamer？

相信大家都用过数字相机或摄像头，随便拍一张照片都是几百万甚至上千万像素，如果在不

压缩的情况下，一张超高画质(Full HD)图像分辨率为 1920 x 1080 像素（俗称 200 万或 2M），每个彩色像素红绿蓝各使用 1 Byte (8 bit)来表示，则要储存这张图像就要 6.22 MByte。若以这个分辨率拍摄每秒 30 帧(frame)的视频时，则 1 秒钟就要 186.6 Mbyte，1 分钟 11.2 GByte，一台 1 TByte (1,000 GByte)的硬盘也不过只能存不到 90 分钟视频，所以就有了利用人眼对纹理频率（平滑、杂乱纹理）、色彩浓度感知能力不同，发明了各种破坏性压缩方式来缩小图像及视频的文件大小，如 jpg，mp4 等格式，让图像、视频大小缩小数十到百倍，而人眼也很难分辨其差异。

虽然图像压缩已解决储存大小的问题，但如果想从网络上看一部视频却要等所有视频都下载完才能观看，这也太不方便了。于是就有人提出可以把取得图像、压缩、传送、解压缩、显示变成一个像水管(pipeline)依序分段工作的作法，视频（连续图像）内容就像水一样在水管中流动，使用者收到一小段资料就可以开始播放，不必等到视频全部下载完才能观看。而且视频内容只需暂存在缓冲区，不一定要存在硬盘中，可节省储存空间，因此透过此种方式传送的视频就称为串流视频(Stream Video)。目前像大家常看的 Youtube、各种直播或者是网络监控摄像头(IP Camera)都是采用这样的作法，当然我们也可以把存在硬盘中的文件当作串流视频读取，这样也可大幅增进播放效率。

目前市面上有很多串流视频的处理软件及开发工具，其中 GStreamer [1]就是最广为人知的开源工具。它主要在 Liunx 上运行，有很多硬件 (CPU, GPU, DSP, CODEC, ASIC 等) 厂商都对其进行了优化，使得影音内容在压缩（编码 Encode） / 解压缩（解码 Decode）效率大幅提升。如 Fig. 1 所示，GStreamer 采取功能方块作法，每个方块根据不同工作项目会有不同输入(sink)和输出信号源(src)，再按照需求将前后串联起来。以一个影音播放器(player)为例，首先开启影音文件，再将其分解(demuxer)成声音(src_01)和视频(src_02)两组信号源，再按照声音和视频格式给予对应的解码器(decoder)，最后再由对应的扬声器和显示器将声音和视频播出来，完成串流影音文件播放工作。当然这里是采取一边读取文件一边播放的串流型式存在。

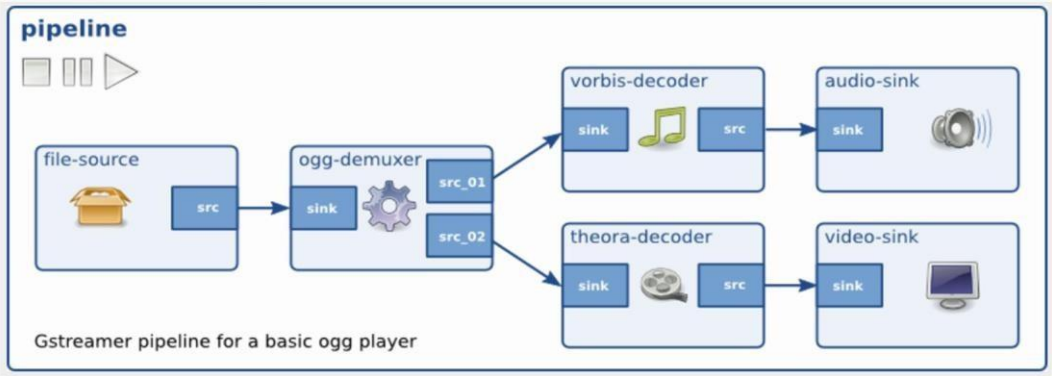


Fig. 1 GStreamer 串流影音播放工作流程。[2]

* 如何安装 OpenVINO DL Streamer 环境

OpenVINO 为了方便大家使用 DL Streamer，同样地在 Docker Hub 上提供了现成的映像文件(Image)给大家下载，不熟悉如何使用 Docker 安装的人，可以参考「【Intel OpenVINO™教学】如何利用 Docker 快速构建 OpenVINO 开发环境」[3]。不过上次文中

使用的 ubuntu18_dev 这个映像文件并不支持 DL Streamer, GStreamer, Speech Libraries 及 Intel Media SDK，所以要重新到 Docker Hub 下载 ubuntu18_data_dev 映像文件，执行下方指令即可获得映像文件（约 8.28 GByte）。不过这些文件有点大，下载时间从数分钟到数十分钟不等，具体视网络速度而定。

```
# 到 Docker Hub 下载映像文件
docker pull openvino/ubuntu18_data_dev
# 检查是否下载成功
docker images
```

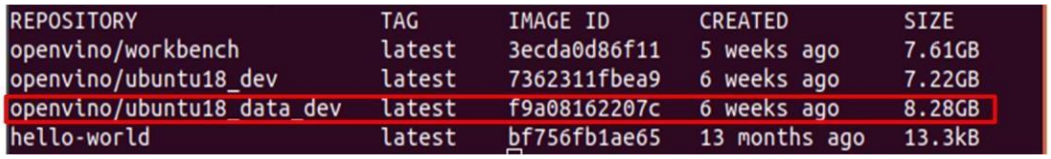


Fig. 2 Intel OpenVINO DL Streamer 映像文件下载成功画面。

另外补充一下，若非使用 Docker 安装，则要注意软件工作环境需满足下列条件。

- OpenVINO 2021.1（Inference Engine 2.1.0）或以上版本
- Linux Kernel 4.15 或以上版本
- GStreamer 1.16 或以上版本

更多 DL Streamer 完整说明可参考 Intel 官方 Github openvinotoolkit/dlstreamer_gst。 [4]

*串流视频分析及优化流程

OpenVINO 目前整合了 GStreamer 的串流视频工具（VAAPI, labav 等）、原有的推理引擎 (Inference Engine, IE)、OpenCV 等函数库，搭建 DL Streamer 串流视频分析插件 (GStreamer Video Analytics Pulgin, GVA)，方便让开发者创建的应用程序能轻松地发挥硬件加速计算组件的效能，如 Fig. 3 所示。

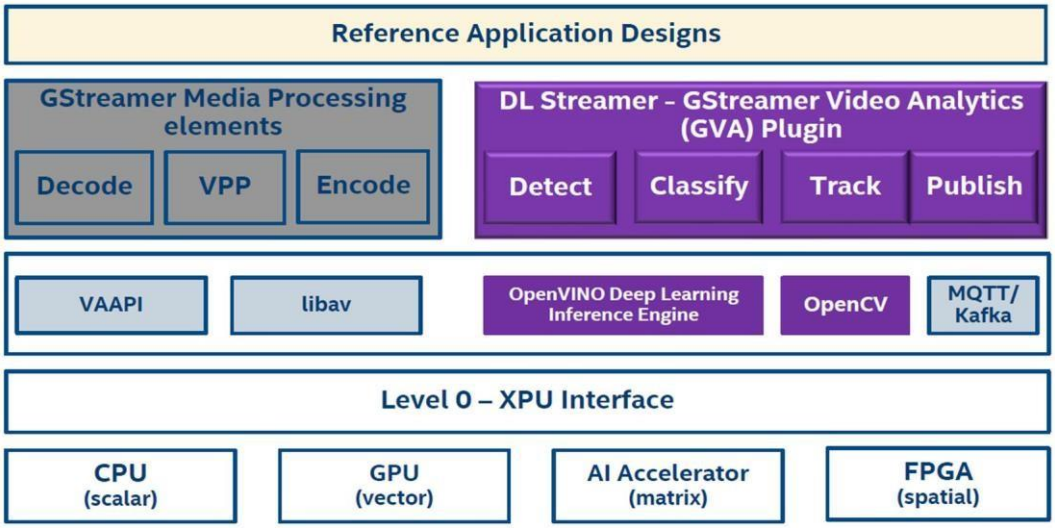


Fig. 3 OpenVINO DL Streamer Software Stack。（图片来源：Intel IOTG）

接下来就以一个常见的应用来举例什么是串流视频分析(Stream Video Analytics)。假设一个商店在门口处装了一组网络监控摄像头 (IP Camera)，想要利用拍到的图像来得知进门的顾客是否为重要客人(VIP)或者是黑名单(Blocklist)客户，此时就要对拍到的连续图像进行人脸识别，这就是串流视频分析。

一个完整的串流视频分析工作包括接收已压缩的视频、解码（解压缩）、前处理（如图像格式转换、亮度/色彩/对比调整等）、分析/推理（图像分类、对象检测等）、后处理（将结果以文字、绘图等方式迭回原图像上），最后视输出方式可选择直接输出到屏幕显示或着再重新编码（压缩）成文件储存回硬盘中，如 Fig. 4 中间列所示。

其中在推理部份可以是多种工作串接而成的，也可同时处理二种以上工作，在 OpenVINO GVA Plugin 中提供了多种函数（元素 Element）可供使用、包括检测 (gvadetect)、分类 (gvaclassify)、追踪(gvatrack)、水印(gvawatermark)、发布(gvapublish) 等，而更多完整的介绍及用法可参考官网 Github 说明[5][6]。如 Fig.4 上方列即表示使用了检测、识别、追踪及水印等元素来完成推理及显示工作。

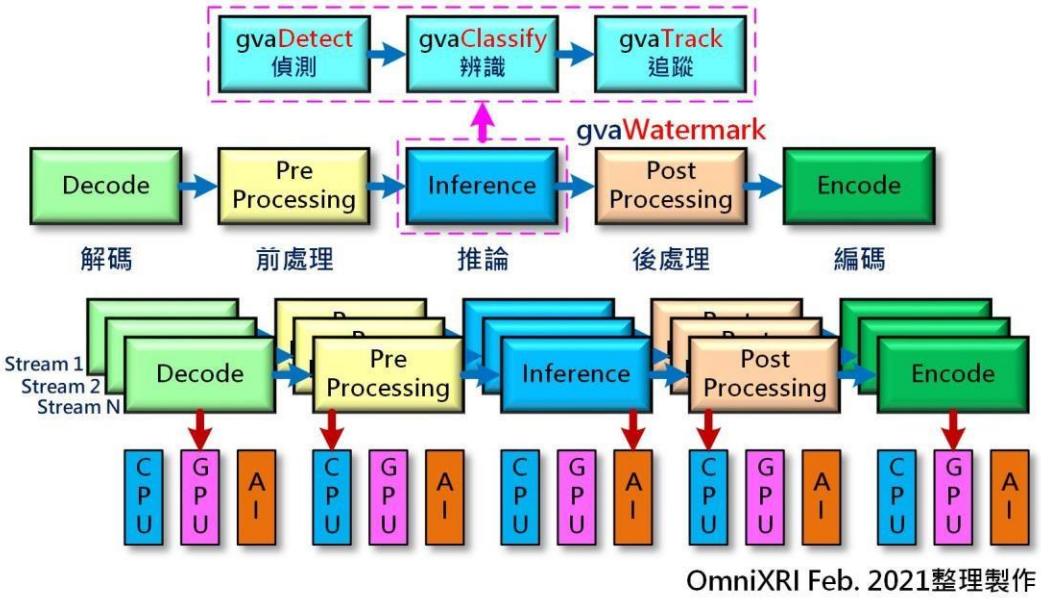


Fig. 4 DL Streamer 串流视频分析工作流程图。(OmniXRI Feb. 2021 整理制作)

在这一连串工作中，并不是所有工作都交给 CPU 来处理，而是将其分配给最适合、效率最高的硬件来执行才是最理想的。比方说文件的存取（硬盘和内存间数据搬移）、格式转换、绘图等非矩阵运算工作交给 CPU，解码/编码等大量数学运算交给 GPU，而神经网络推理所需的巨量矩阵运算工作则由 AI 芯片（也可以是 GPU, VPU, FPGA, NPU, ASIC 等）完成，如此就能得到最佳的工作效率。一般串流视频播放速度大约是 1/30 秒一个帧(Frame)，所以当传输及计算速度够快时，就能同时计算多个串流视频，如 Fig. 4 下方列所示。

接下来说明如何提升推理效率，如同本文一开始提到的，如果对串流视频的每一帧都进行推理，则系统执行效率会变得很差，很难达成每秒 30 帧的计算，如 Fig. 5 左上图所示（这里仅以检测为例，另可加入分类识别及其它工作项目）。为改善执行效率，可令对象检测及分类次数减少，因为帧间对象的位置、尺寸、外形可能变动不大，不需一直执行推理工作。如 Fig. 5 左下图所示，对象检测和图像分类的间隔数量可以设成不同（也可相同），而帧间隔数量则视视频中对象移动速度而定。但是采用这种方式有个缺点，有些帧可能不进行推理，所以会造成没有结果内容可以迭合回原图像中，画面会有闪烁情况发生。

如果想更精准表示对象的微小位移，则使用传统的对象追踪方式会比直接使用神经网络进行对象检测更快。只要在两个检测帧中间加入追踪动作，就可自动为中间帧找出对象的位置，改善对象检测位置偏差及画面闪烁问题，如 Fig. 5 右图所示。目前提供了两种追踪模式，zero-term 会给每个对象一个独立编号 (unique ID)方便在任一帧中查找，而 short-term 则会找出帧间最接近的内容位置。

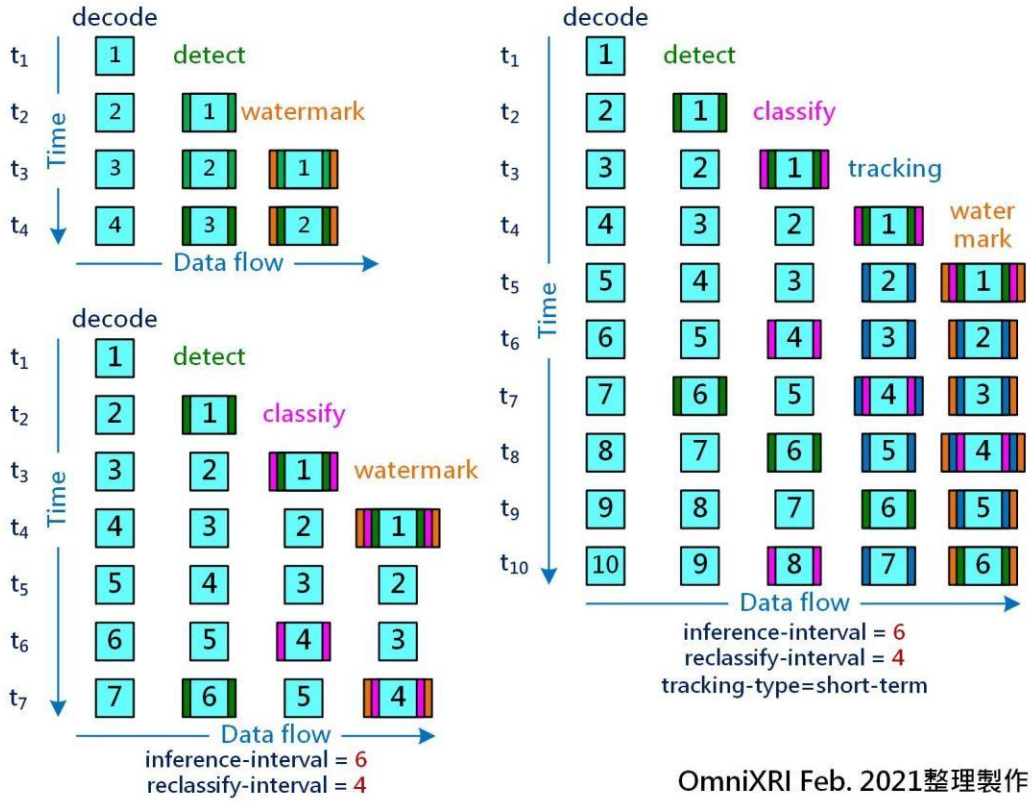


Fig. 5 串流视频分析优化方式比较图。左上图：每一帧全都进行对象检测，无优化。左下图：每 6 个帧检测一次，4 个帧分类一次。右图：间隔检测及分类并加入追踪补足。(OmniXRI Feb. 2021 整理制作)

更多 OpenVINO 中 GStreamer 及 DL Streamer 的原理及使用方式可参考官方提供的 Youtube 视频说明[7][8][9][10]。

＊DL Streamer 范例说明

目前 DL Streamer 提供多种执行范例，包括直接命令行指令(gst-launch)、C / C++ / Python 程序等，接下来就以直接命令行指令方式来进行介绍。启动 Docker 后这些范列预设会安装在 /opt/intel/openvino_2021.2.185/data_processing/dl_streamer/samples 路径下（openvino 后方数字代表对应版本）。执行范例前要先下载相关预训练模型及参数文件，其工作命令如下所示。

```
# 开放权限，允许本地连接，方便后面 Docker 中启用 X Window GUI 功能
xhost +local:root
```

```
# 以 Docker 方式运行 DL Streamer 范例之命令，其中另增加--user root 用户权力，方便在 Docker 中安装文本编辑器(nano, vim 等)
docker run -it --device /dev/dri:/dev/dri --d evi ce -cgroup-rule='c 189:* rmw' -v ~/.Xauthority:/root/.Xauthority -v /tmp/.X11-unix/:/tmp/.X11-unix/ -e DISPLAY=$DISPLAY -v /dev/bus/usb:/dev/bus/usb --rm --u ser root openvino/ubuntu18_data_dev:latest
# 启动 Docker 完成会出现下列提示字符，最后数字为 OpenVINO 安装版本
openvino@e3a20b255ff3:/opt/intel/openvino_2021.2.185$
# 接着更新 Docker 中系统套件及安装文本编辑器 nano 方便后续修改范例程序
apt update
apt install nano
# 下载范例程序所需模型及参数，相关范例程序也在此路径下
cd data_processing/dl_streamer/samples
./download_models.sh
# 完成下载后默认会将模型及参数存放在 /home/openvino/intel/dl_streamer/model 路径下
```

由于启动 Docker 的命令中有 --rm，所以每次离开后会清空所有新增内容，再重新启动时要重新执行上述安装及下载动作，以免造成范例无法运行问题。更多关于该映像文件不同启动方式及参数，可参考 Intel 官方 Docker Hub / openvino / ubuntu18_data_dev 说明[11]。

在 /data_processing/dl_streamer/samples/gst_launch/ 下主要是提供以 gst-launch 命令行方式执行串流视频分析的范例，有人脸检测及识别、语音事件检测、车辆行人追踪、中介数据发布等应用，这里会以车辆行人追踪分析为例进行说明。

首先进到/vehicle_pedestrian_tracking 路径下，执行 vehicle_pedestrian_tracking.sh 即可看到执行结果。

```
# 切换工作路径至范例程序所在路径
cd /data_processing/dl_streamer/samples/gst_launch/vehicle_pedestrian_tracking
# 执行车辆行人追踪分析范例，可选配输入三项参数，也可不输入使用默认值
./vehicle_pedestrian_tracking.sh [输入影片] [侦测间隔帧数] [推理精度]
```

为更清楚说明执行参数目的及运作指令方式，以下就将 vehicle_pedestrian_tracking.sh 内容完整显示于 Fig. 6 并补充说明如下：

- **输入视频(FILE)：**可支持本地端（硬盘中）影音文件（*.mp4 等）、网络摄像头(Webcam)、串流摄像头(IP Camera)网址(rtps://)或者一般网络串流文件 (http://)。本范

例默认值为 https://github.com/intel-iot-devkit/sample-videos/blob/master/person-bicycle-car-detection.mp4，也就是说，不输入此项参数时会自动上网读取这个串流视频，注意计算机必须连网才能正确执行。

- **检测间隔帧数(DETECTION_INTERVAL)**：为加速串流视频分析，可设定间隔帧数(Frame)减少计算，其概念如 Fig. 5 左下图所示。可另外搭配下方追踪项目使用，补齐中间帧的检测工作。默认为 10，以一般视频帧速度(FPS)为每秒 30 帧来算，相当于 1/3 秒检测一次。对于对象移动缓慢的场景已足够使用，可视需求调整长短。
- **推理精度(INFERENCE_PRICISION)**：可根据需求选择不同推理精度，预设为 FP32，即推理模型的权重值是以 32 位浮点数表示。若想加快计算可选择 FP16（16 位浮点数）或 INT8（8 位整数），当然这样可能会牺牲一点推理的正确性。

以上三组为可自由设定的参数，若只想使用默认值可不输入，若只想修改检测间隔帧数，此时第一项参数必须设成 ""（空字符串）令其使用默认值，否则将无法正确运作。

除了这些自由设定参数外，范例中还有许多参数可以调整，但需自行以文字编辑器（如 nano, vim 等）载入再修改。以下分别说明几个重要参数。

- **推理设备(INFERENCE_DEVICE)**：预设为 CPU，可自行修改成 GPU, MYRIAD, HDDL 等不同设备。但使用 GPU 前要确定是 INTEL 6 到 10 代绘图处理器 Iris 或 HD 系列，或者 11 代 X^e，OpenVINO 不支持 NVIDIA 或 AMD 的 GPU。另外模型大小不一，所以不保证所有模型一定可以在 CPU 以外设备上使用。另外提醒，如果在 Linux 上手动安装 OpenVINO，需另外安装 Intel OpenCL 驱动程序，否则将无法正确使用 GPU。
- **追踪型式(TRACKING_TYPE)**：预设为 short-term，另可选择 zero-term。主要用于协助补齐检测及分类未工作之帧的计算。
- **重新分类帧数间隔(RECLASSIFY_INTERVAL)**：默认为 10，这个数字可检测间隔帧数相同亦可不同，视需求而定。同样地也可搭配追踪型式补齐未分类中间帧数的分类结果。

在这个范例中主要使用四个 GVA Plugin 并搭配指定模型来完成工作，包括：

- **gvadetect**：负责车辆、脚踏车、行人检测，对应 person-vehicle-bike-detection-crossroad-0078。
- **gvainference**：负责人员及车辆属性推理，分别对应 person-attributes-recognition-crossroad-0230 及 vehicle-attributes-recognition-barrier-0039。
- **gvatrack**：负责补齐未进行对象检测及分类推理中间帧的分析结果。
- **gvawatermark**：负责将分析结果产生的图框和文字迭合回原图像中，方便用户了解运行速度及是否有正确检测及分类推理。
- **videoconvert**：视频格式（色彩空间）转换。
- **fpsdisplaysink**：渲染(Rendering)输出结果到屏幕速度。

更完整的参数说明可参考官网[12]。

```
#!/bin/bash
# ==============================================================================
# Copyright (C) 2020 Intel Corporation
#
# SPDX-License-Identifier: MIT
# ==============================================================================

set -e

# input parameters 外部輸入參數
FILE=${1:-https://github.com/intel-iot-devkit/sample-videos/raw/master/person-b$
DETECTION_INTERVAL=${2:-10}
INFERENCE_PRECISION=${3:-"FP32"}
INFERENCE_DEVICE=CPU

MODEL_1=person-vehicle-bike-detection-crossroad-0078
MODEL_2=person-attributes-recognition-crossroad-0230
MODEL_3=vehicle-attributes-recognition-barrier-0039

# Other tracking types available are short-term-imageless, zero-term, zero-term$
# For more information on tracking types and their difference, please turn to
# https://github.com/openvinotoolkit/dlstreamer_gst/wiki/Object-tracking.
TRACKING_TYPE="short-term"

RECLASSIFY_INTERVAL=10

if [[ $FILE == "/dev/video"* ]]; then
  SOURCE_ELEMENT="v4l2src device=${FILE}"
elif [[ $FILE == *"://"* ]]; then
  SOURCE_ELEMENT="urisourcebin buffer-size=4096 uri=${FILE}"
else
  SOURCE_ELEMENT="filesrc location=${FILE}"
fi

GET_MODEL_PATH() {
    model_name=$1
    precision=${INFERENCE_PRECISION}
    for models_dir in ${MODELS_PATH//:/ }; do
        paths=$(find $models_dir -type f -name "*$model_name.xml" -print)
        if [ ! -z "$paths" ];
        then
            considered_precision_paths=$(echo "$paths" | grep "/$precision/")
            if [ ! -z "$considered_precision_paths" ];
            then
                echo $(echo "$considered_precision_paths" | head -n 1)
                exit 0
            else
                echo $(echo "$paths" | head -n 1)
                exit 0
            fi
        fi
    done

    echo -e "\e[31mModel $model_name file was not found. Please set MODELS_PATH$
    exit 1
}

PROC_PATH() {
    echo $(dirname "$0")/model_proc/$1.json
}

DETECTION_MODEL=$(GET_MODEL_PATH $MODEL_1)
PERSON_CLASSIFICATION_MODEL=$(GET_MODEL_PATH $MODEL_2)
VEHICLE_CLASSIFICATION_MODEL=$(GET_MODEL_PATH $MODEL_3)

DETECTION_MODEL_PROC=$(PROC_PATH $MODEL_1)
PERSON_CLASSIFICATION_MODEL_PROC=$(PROC_PATH $MODEL_2)
VEHICLE_CLASSIFICATION_MODEL_PROC=$(PROC_PATH $MODEL_3)

PIPELINE="gst-launch-1.0 \
  ${SOURCE_ELEMENT} ! decodebin ! \
  gvadetect model=$DETECTION_MODEL \
            model-proc=$DETECTION_MODEL_PROC \
            inference-interval=${DETECTION_INTERVAL} \
            threshold=0.6 \
            device=${INFERENCE_DEVICE} ! \
  queue ! \
  gvatrack tracking-type=${TRACKING_TYPE} ! \
  queue ! \
  gvaclassify model=$PERSON_CLASSIFICATION_MODEL \
              model-proc=$PERSON_CLASSIFICATION_MODEL_PROC \
              reclassify-interval=${RECLASSIFY_INTERVAL} \
              device=${INFERENCE_DEVICE} object-class=person ! \
  queue ! \
  gvaclassify model=$VEHICLE_CLASSIFICATION_MODEL \
              model-proc=$VEHICLE_CLASSIFICATION_MODEL_PROC \
              reclassify-interval=${RECLASSIFY_INTERVAL} \
              device=${INFERENCE_DEVICE} object-class=vehicle ! \
  queue ! \
  gvawatermark ! videoconvert ! fpsdisplaysink video-sink=xvimagesink sync=true"

echo ${PIPELINE}
${PIPELINE}
```

串流影像來源：本地檔案、Webcam、遠端串流影像來源網址(rtsp:// http://)

偵測間隔幀數(預設10幀)

推論精度(預設FP32, 另可選FP16, INT8...)

推論裝置(預設CPU, 另可選GPU, MYRIAD...)

預訓練模型

追蹤型式(預設sort-term，另可選zero-term)

重新分類間隔幀數(預設10幀)

獲取模型路徑

顯示模型處理定義檔案路徑及名稱

物件偵測：行人、汽車、腳踏車

物件追蹤

物件分類：行人

物件分類：汽車

疊印文字、圖框 | 影片格式轉換 | 速度顯示 | 視訊同步顯示

OmniXRI Feb. 2021整理製作

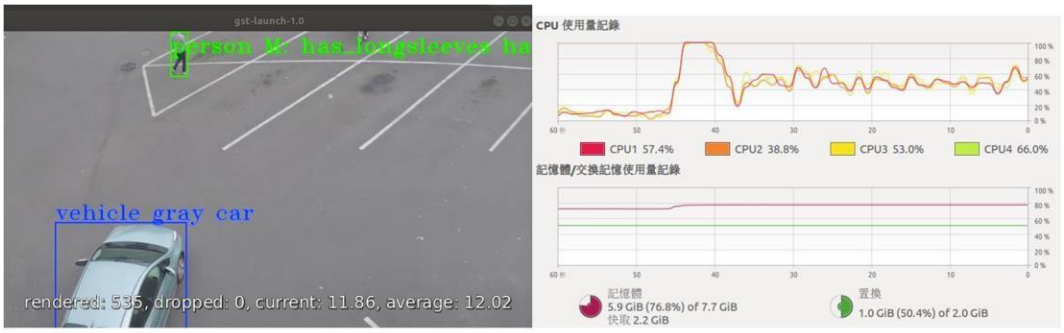
串流图像来源：本地文件 Webcam	串流图像来源：本地文件 Webcam
外部输入参数	外部输入参数
远程串流图像来源网址(rtsp:// http://)	远程串流图像来源网址(rtsp:// http://)
检测间隔帧数(预设 10 帧)	检测间隔帧数(预设 10 帧)
推理精度(预设 FP32，另可选 FP16，INT8…)	推理精度(预设 FP32，另可选 FP16，INT8…)
推理设备(默认 CPU，另可选 GPU，MYRIAD)	推理设备(默认 CPU，另可选 GPU，MYRIAD)
预训练模型	预训练模型
追踪型式(默认 sort-term, 另可选 zero-term)	追踪型式(默认 sort-term, 另可选 zero-term)
重新分类间隔帧数(预设 10 帧)	重新分类间隔帧数(预设 10 帧)
获取模型路径	获取模型路径
显示模型处理定义文件路径及名称	显示模型处理定义文件路径及名称
对象检测：行人、汽车、脚踏车	对象检测：行人、汽车、脚踏车
对象追踪	对象追踪
物件分类：行人	物件分类：行人
物件分类：汽车	物件分类：汽车
迭印文字、图框 | 视频格式转换 | 速度显示 | 视频同步显示	迭印文字、图框 | 视频格式转换 | 速度显示 | 视频同步显示
OmniXRI Feb. 2021 整理制作	OmniXRI Feb. 2021 整理制作

Fig. 6 以命令行方式进行串流视频车辆行人分析范例(vehicle_pedestrian_tracking.sh)说明。(OmniXRI Feb. 2021 整理制作)

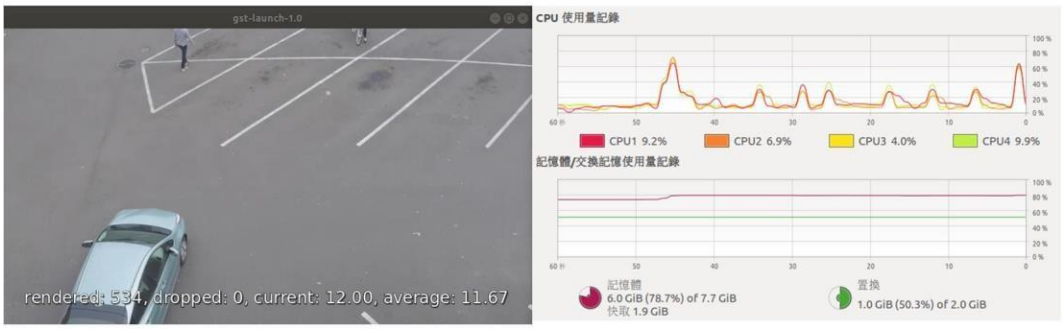
再来简单比较一下在不同参数下的执行结果。测试条件为：Intel Core i5-4400 @3.1GHz 4 核 4 线程，8 GByte 内存，OpenVINO Docker Image: ubuntu18_data_dev (OpenVINO 2021.2.185 for DL Streamer)。分别以下列三种方式进行比较。其中不启动追踪方式即是将 Fig. 6 中对象追踪（蓝色虚线框）那两行指令删除。实验结果如 Fig. 7 所示。

- **检测、分类间隔皆为 1 且不启动追踪：**执行时 CPU **四核平均使用率** 53.8%，推理负担极重，平均速度 12.02 FPS。每一个帧都会显示检测到的对象及分类结果。
- **检测、分类间隔皆为 10 且不启动追踪：**执行时 CPU 四核平均使用率 7.5%，推理负担极轻，平均速度 11.67 FPS。每十个帧会显示检测到的对象及分类结果，故画面会有点闪烁。
- **检测、分类间隔皆为 10 且启动追踪：执行时** CPU 四核平均使用率 21.65%，推理负担中等，平均速度 11.60 FPS。由于有启动追踪功能，所以每一个帧都会显示检测到的对象及分类结果。

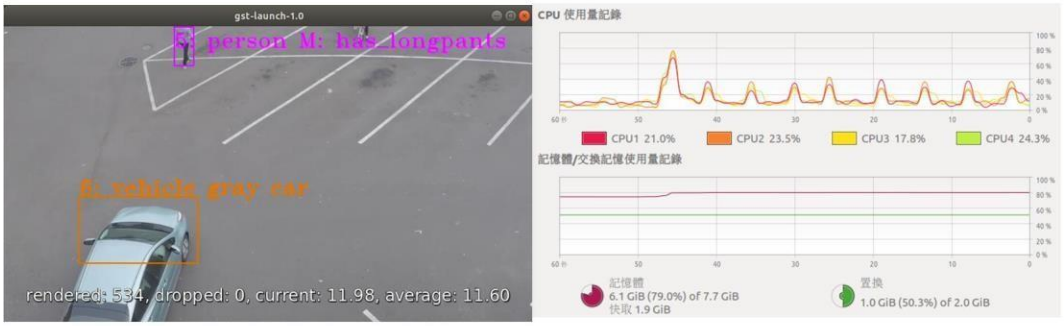
整体来说只测试一个串流视频还没让 CPU 耗尽所有资源，所以应该还可以同时接受多个视频（串流摄像头、网络摄像头）或更多分析项目。另外加长检测及分类帧数间隔明显会令系统计算量大幅减少。如果再加上追踪功能，除可满足逐格分析外又不会耗损太多计算资源，这样就可容纳更多串流视频一起工作。



(a) DETECTION_INTERVAL=1, RECLASSIFY_INTERVAL=1, Without Tracking



(b) DETECTION_INTERVAL=10, RECLASSIFY_INTERVAL=10, Without Tracking



(b) DETECTION_INTERVAL=10, RECLASSIFY_INTERVAL=10, TRACKING_TYPE=short-term

OmniXRI Feb. 2021整理製作

Fig. 7 不同测试条件实验结果，左行为输出图像，右行为 CPU 及内存使用情况。(OmniXRI Feb. 2021 整理制作)

* 小结

在当前网络发达的年代，不论是智能零售、智能安防、智慧城市等各种应用都离不开串流摄像头或网络视频的传送方式。有了像 Intel OpenVINO DL Streamer 这类智能分析工具后，就能更容易满足大量输入，快速推理的需求。本文篇幅有限，无法一一展现这项工具的所有细节，剩下部份请大家多多去探索。

* 参考文献

[1] GStreamer open source multimedia framework https://gstreamer.freedesktop.org/

[2] GStreamer pipeline https://gstreamer.freedesktop.org/documentation/application-development/introduction/basics.html

[3] 许哲豪，”【Intel OpenVINO™教学】如何利用 Docker 快速建置 OpenVINO 开发环境” http://omnixri.blogspot.com/2021/01/intel-openvinodockeropenvino.html

[4] Github – openvinotoolkit / dlstreamer_gst https://github.com/openvinotoolkit/dlstreamer_gst

[5] Github - openvinotoolkit / dlstreamer_gst / Elements https://github.com/openvinotoolkit/dlstreamer_gst/wiki/Elements

[6] GStreamer Video Analytics (GVA) Plugin https://openvinotoolkit.github.io/dlstreamer_gst/index.html

[7] Intel Software, "Full Pipeline Simulation Using GStreamer | OpenVINO™ toolkit | Ep. 47 | Intel Software" https://youtu.be/fWhPV_IqDy0

[8] Intel Software, "Full Pipeline Simulation Using GStreamer Samples | OpenVINO™ toolkit | Ep. 48 | Intel Software" https://youtu.be/EqHznsUR1sE

[9] Intel Software, "DL Streamer Tracking Element | OpenVINO™ toolkit | Ep. 64 | Intel Software" https://youtu.be/z4Heorhg3tM

[10] Intel Software, "DL-Streamer Python Custom Element | OpenVINO™ toolkit | Ep. 66 | Intel Software" https://youtu.be/SDGE9Vyd-bY

[11] Docker Hub openvino/ubuntu18_data_dev, “Intel® Distribution of OpenVINO™ toolkit Docker image for Ubuntu* 18.04 LTS” https://hub.docker.com/r/openvino/ubuntu18_data_dev

[12] Intel OpenVINO Toolktit, “Vehicle and Pedestrian Tracking Sample (gst-launch command line)” https://docs.openvinotoolkit.org/latest/gst_samples_gst_launch_vehicle_pedestrian_tracking_README.html