IMPROVING MACHINE LEARNING EXPERIENCE WITH MULTIMEDIA TECHNIQUES

Marco A. Franchi

ABSTRACT

It has been very common face machine learning algorithms at the multimedia area: traffic count; real-time vigilance cameras; baggage tracker; face/expression recognition; and so on. However, it is perceptive the gap between machine learning and multimedia solutions, where even at simple embedded systems it is possible to reach 4k videos running at 60 frames per second, whereas the best neural network solution only handles 224x224 frames at 300 milliseconds in the same CPU system. Due to this gap, a vast number of solutions as being developed: dedicated hardware for inference process; model manipulation; accelerated pre-processing image solutions; and video manipulation techniques. This paper is based at the study of these videos manipulation techniques, exposing the most common algorithmics, showing the main difference between them and the expected performance for each one

1. INTRODUCTION

Aiming to diminish the gap between the machine learning inference process and the multimedia capability, some video manipulation solutions was purposed. Among them, the most common is the overlay solutions, which are able to create alpha layers over the video and insert information on it. These overlays are very common on object detections algorithmics, once they are responsible for drawing a square, select or color an object at the scene. The most common overlays are Scalable Vector Graphics (SVG), Cairo, and OpenCV. In addition to the overlays, it is important to use a great video framework able to handle all the elements involved at the inference and display solutions. One of the best and most useful ones is the GStreamer framework. GStreamer is able to handle plugins in pipelines, which is perfect to do quick tests. Apart from the video solutions, it is important to choose the best machine learning algorithmics as well. With a focus on object detection, the most common and valued ones are Single Shot Detection (SSD) and Tensorflow. Both have an incredible inference process capability and the TFlite version demonstrated a great tool for embedded systems. Thus, this paper intends to compare the combination of theses algorithmics for object detection solutions. This comparison aims to demonstrate how to increase the video frame rate with simple approaches and demonstrate the best scenarios to handle each neural network algorithmics.

2. MATERIALS AND METHODS

This section describes the material such as the video files, models, labels, and programming language used, and the adopted methodology.

2.1. Programming Language

This paper uses Python 3 language and all the required support for the object detection algorithm: the overlays libs, GStreamer plugins, and TFLite.

OpenCV: Open Source Computer Vision Library, this library is cross-platform and free for user under open-source BSD license.

SVG image: Scalable Vector Graphics, a Extensible Markup Language (XML)-based vector image format for two-dimensional graphics.

GStreamer: GStreamer is a library for constructiong graphics of media-handling components and is released under the LGPL license.

TFLite: Tensoflow is a library for dataflow and differenctiable programming across of tasks, free and open-source under the Apache License 2.0. TFLite is a special version for mobile development.

2.2. Models and Labels

As the focus of this paper is the video techniques, and as the SSD and TFlite already have a huge numbers of pre-processed models, this paper will not care about pre-processing or training models and will uses pre-processed models available at the TFLite official object detection support web page.

For this, the following pre-tested model and labels will be used at the tests:

Model: mobilenet ssd v2 coco quant postprocess.tflite

Labels: cocolabels.txt

2.3. Workflow

2.3.1. OpenCV V4L2 directly handle

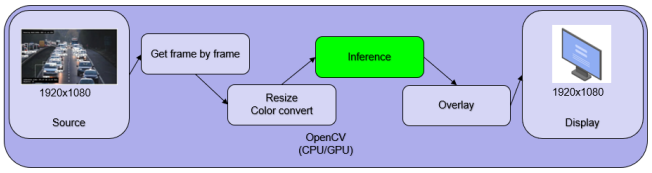

Figure 1: OpenCV V4L2 directly handle multimedia example.

For this approach, the OpenCV needs to handle the entire process: open the video, get the frame, resize and do a color convert on it, draw the informations at the frame, and display the results. This is easely to test, but not fast for ML purpose, once the entire process will be depent from OpenCV.

2.3.2. GStreamer appsink pipeline + OpenCV V4L2 output

This idea shows a better perfomance than OpenCV V4L2 due the GStremaer usage at the video openning process. It represents one less action to be processed by the OpenCV, and the GStreamer pipelines enables a vast number of

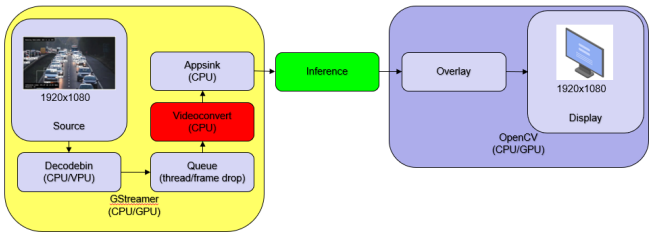

Figure 2: GStreamer appsink pipeline + OpenCV V4L2 output

plugins and its properties, which can guarantee some aditional chacateristics, such as dropping frame property. It means that the displayed frame rate will not be impacted for the inference process time. However, the videoconvert usage, required for the appsink to be able to display the results at screen, is a disavantage, once its only supported by CPU, and resize/color convert by CPU has a high processing coast.

2.3.3. GStreamer appsink + appsrc pipelines

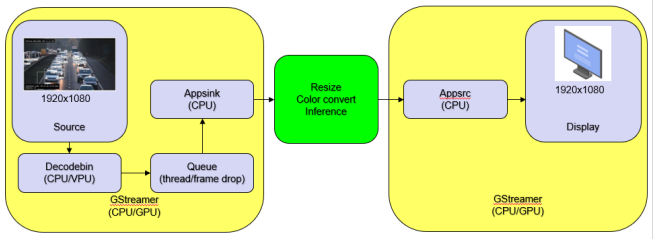

Figure 3: GStreamer appsink + appsrc pipelines

This combination shows very promissor, once we can use the appsink dropping frame property, but do not requires videoconvert, once its results will not be displayed yet. Actually, the resulted data will be processed by the inference process, which can include resizing and color convert, and the results will be send again to GStreamer, only to handle it and display at the screen.

2.3.4. GStreamer overlay plugins support

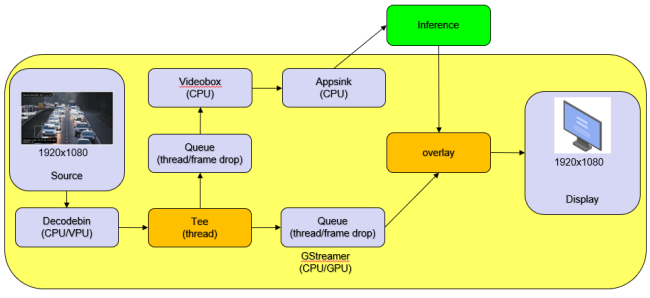

Figure 4: GStreamer overlay plugins support

This is the best approach, but the most dificult to use. Here, the tee usage create two threads: one to be processed by the inference process; other to be displayed. The videobox keeps the frame and is able to return it ot the overlay plugin, so what we see is the combination of two frames, one is the original video, without be touched, other is an alpha image with all the required resizing process being displayed over the original video.

3. RESULTS

The examples were made to return the FPS and inference time average collected during the video playback process. With this values, two graphs were generated: one showing the FPS per each example, other showing the Inference time per each example.

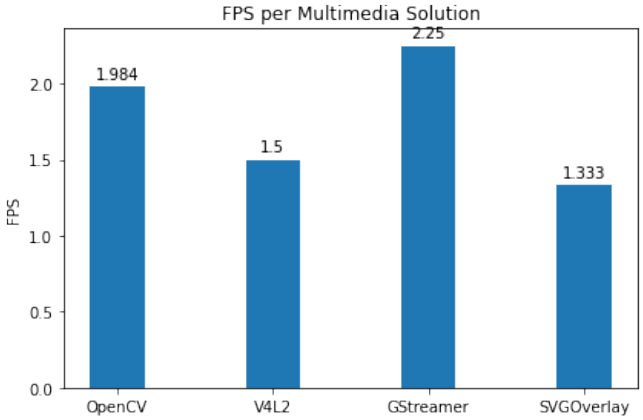

Figure 5: FPS per Multimedia Solution Results

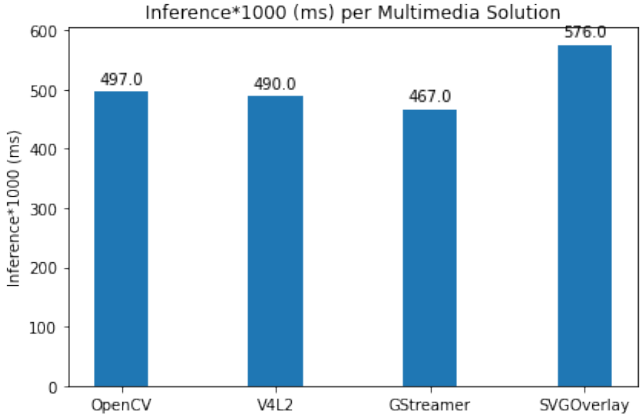

Figure 6: Inference*1000 (ms) per Multimedia Solution Results

4. CONCLUSION

Unfortunately, the object detection algorithms consume a lot of process resources, and the notebooks have no performance enough to handle it. However, it was still able to show the difference at the inference time and the framerate, both running together and consuming the same CPU process. According to the values obtained, noticed that the number of about 0.5 milliseconds is far from the expected for a video object detection solution. With some simples calculations, it is easy to note that 0.5 ms of inference time represents 2 frames being calculated per second. It is very slow and it is far from the multimedia capabilities, which can achieve 60 frames per second.

5. REPRODUCIBILITY

The main concern of this paper is the reproducibility. For those interested in this study that want to give a try on it,

check/ the source code, data, and variable, please, access the following page:

`https://github.com/marcofrk/ia369_final_project`

6. REFERENCES

Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. 2016. TensorFlow: A System for Large-Scale Machine Learning.

Ganesh Ananthanarayanan, Paramvir Bahl, Peter Bodík, Krishna Chintalapudi, Matthai Philipose, Lenin Ravindranath, and Sudipta Sinha. 2017. Real-Time Video Analytics: The Killer App for Edge Computing.

Google. 2019. Coral Edge TPU. Retrieved June 16, 2020, from https://coral.ai/

Nvidia. 2020. NVIDIA DeepStream SDK. Retrieved June 16, 2020 from https://developer.nvidia.com/deepstream-sdk

GStreamer. 2019. GStreamer: open source multimedia framework. Retrieved June 16, 2020 from https://gstreamer.freedesktop.org

Intel. 2020.03. Intel's Deep Learning Inference Engine Developer Guide. Retrieved June 16, 2020 from https://docs.openvinotoolkit.org/latest/