Homework 6

Submitted By: Javed Roshan

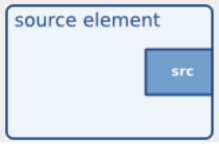
Part 1: GStreamer

1. In the lab, you used the Nvidia sink nv3dsink; Nvidia provides another sink, nveglglessink. Convert the following sink to use nveglglessink.

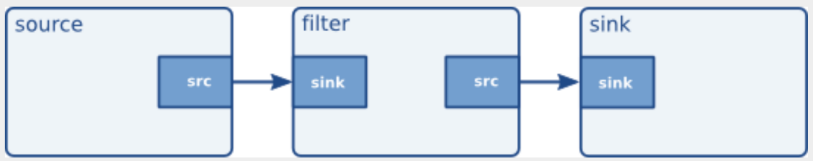
```
gst-launch-1.0 v4l2src device=/dev/video0 ! xvimagesink ! nveglglessink

gst-launch-1.0 v4l2src device=/dev/video0 ! video/x-raw, framerate=30/1 !
videoconvert ! video/x-raw, format=BGR ! nveglglessink
```

2. What is the difference between a property and a capability? How are they each expressed in a pipeline?

An Element is a basic building block for a media pipeline. A 'source element' as shown below generates data for use by a pipeline.



A *Pipeline* can be setup by linking a source element with zero or more *Filter*-like elements and finally a *Sink* element. The data will flow through the elements, and this is the basic concept of media handling in *GStreamer*.



Pads are the *Element's* interface to the outside world. Data streams from one element's source pad to another element's sink pad. ***Capability*** is a mechanism that describes the data that can flow through the pad. They are used for compatibility detection, auto-plugging, filtering etc. ***Properties*** are used to describe extra information of *capabilities*. A property consists of a key (a string) and a value.

Following is an example dump of the capabilities of the "***vorbisdec***" element.

```
Pad Templates:
  SRC template: 'src'
    Availability: Always
    Capabilities:
      audio/x-raw
                  format: F32LE
                    rate: [ 1, 2147483647 ]
                channels: [ 1, 256 ]

  SINK template: 'sink'
    Availability: Always
    Capabilities:
      audio/x-vorbis
```

As seen in the dump, there are two pads: a source and a sink pad. Both of these pads are *always* available, and both have capabilities attached to them. The sink pad will accept vorbis-encoded audio data, with the media type "audio/x-vorbis". The source pad will be used to send raw (decoded) audio samples to the next element, with a raw audio media type (in this case, "audio/x-raw"). The source pad will also contain properties for the audio sample rate (rate: 1, 2147483647) and the number of channels (1, 256).

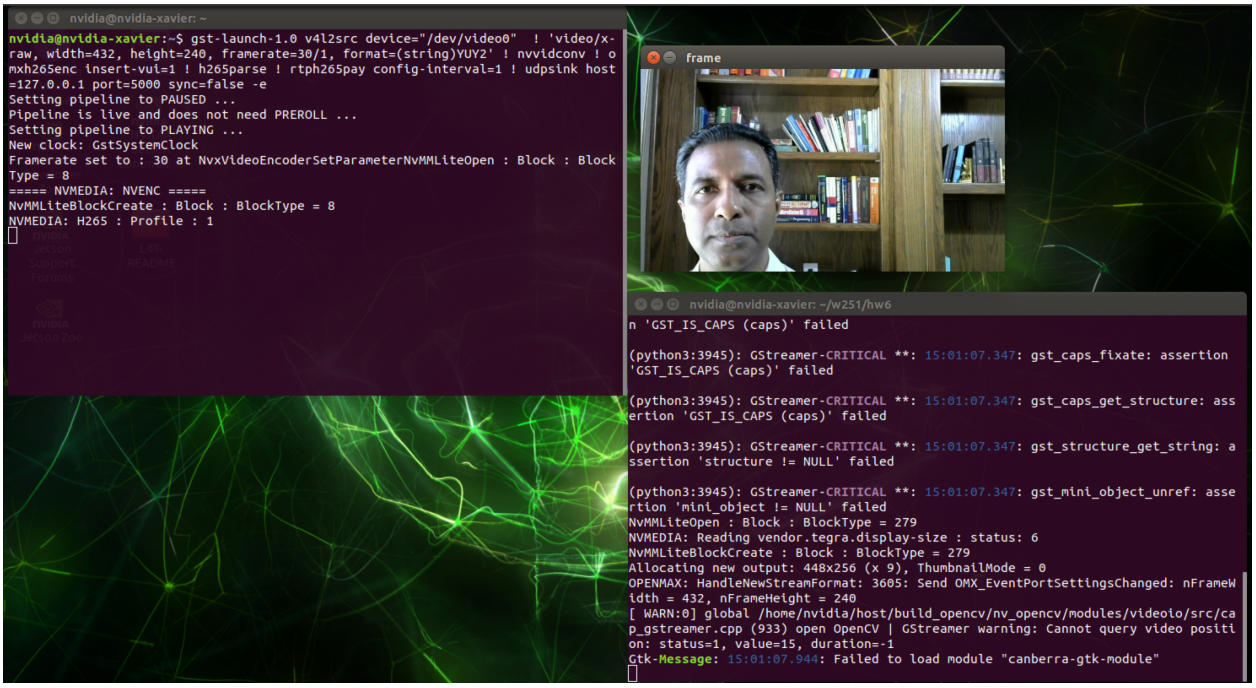
3. Explain the following pipeline, that is explain each piece of the pipeline, describing if it is an element (if so, what type), property, or capability. What does this pipeline do?

```
gst-launch-1.0 v4l2src device=/dev/video0 ! video/x-raw, framerate=30/1 !
videoconvert ! agingtv scratch-lines=10 ! videoconvert ! xvimagesink sync=false
```

The above pipeline can be de-constructed as follows:

v412src device=/dev/video0 shows that the source video data coming from device 0
video/x-raw defines the capability of the source that has the *framerate* of 30/1
videoconvert is a filter that convert video frames between a great variety of video formats
agingtv ages a video stream in real time, changes the colors and adds scratches and dust, the property *scratch-lines=10* defines the number of scratch lines for this video transformation
xvimagesink renders video frames to a drawable (XWindow) on a local display using the XVideo extension. *sync* is its property when enabled runs the XWindow in synchronous mode

4. GStreamer pipelines may also be used from Python and OpenCV. For example:



Publisher:

```
gst-launch-1.0 v4l2src device="/dev/video0"  ! 'video/x-raw, width=432, height=240, framerate=30/1, format=(string)YUY2' ! nvvidconv ! omxh265enc insert-vui=1 ! h265parse ! rtph265pay config-interval=1 ! udpsink host=127.0.0.1 port=5000 sync=false
```

Listener:

```
Camera Setting: udpsrc port=5000 ! application/x-rtp, media=video, width=640, height=480, payload=96, clock-rate=90000, encoding-name=H265 ! rtph265depay ! h265parse ! omxh265dec ! videoconvert ! appsink
```

```python
# cam.py; written by Javed Roshan
import numpy as np
import cv2

# the index depends on your camera setup and which one is your USB camera.
# you may need to change to 1 depending on your local config

camSet='udpsrc port=5000 ! application/x-rtp,media=video,width=640,height=480,payload=96,clock-rate=90000,encoding-name=H265 ! rtph265depay ! h265parse ! omxh265dec ! videoconvert ! appsink'
cap= cv2.VideoCapture(camSet)

#cap = cv2.VideoCapture(0)

while(True):
    # Capture frame-by-frame
    ret, img = cap.read()

    # Display the resulting frame
    cv2.imshow('frame', img)

    if cv2.waitKey(1) & 0xFF == ord('q'):
        break

# When everything done, release the capture
cap.release()
cv2.destroyAllWindows()
```

Part 2: Model optimization & quantization

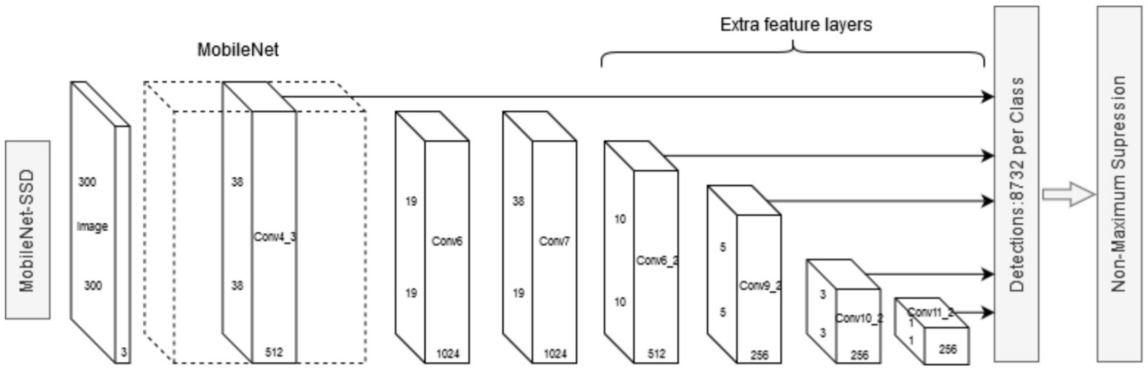
The base model was trained on a total of 2500 images downloaded via

```
python3 open_images_downloader.py --max-images=2500 --class-names
"Apple,Orange,Banana,Strawberry,Grape,Pear,Pineapple,Watermelon" --data=data3/fruit
```

The base model was downloaded from

```
wget https://nvidia.box.com/shared/static/djf5w54rjvpqocsiztzaandq1m3avr7c.pth -O
models/mobilenet-v1-ssd-mp-0_675.pth
```

Following is the model architecture:



It was run for 30 epochs with batch size as 4

```
python3 train_ssd.py --data=data3/fruit --model-dir=models/fruit --batch-size=4 --
epochs=30
```

The data format with some examples is as follows:

ImageID	0026df6bcaf93ac5	0026df6bcaf93ac5	003be557ebbb9e8c
Source	xclick	Xclick	activemil
LabelName	/m/0kpqd	/m/0kpqd	/m/0kpqd
Confidence	1	1	1
XMin	0.333125	0.488125	0.24875
XMax	0.57625	0.9725	0.326875
YMin	0.270833	0.360833	0.846516
YMax	0.474167	0.999167	0.935028
IsOccluded	1	0	-1
IsTruncated	0	1	-1
IsGroupOf	0	0	-1
IsDepiction	0	0	-1
IsInside	0	0	-1
id	/m/0kpqd	/m/0kpqd	/m/0kpqd
ClassName	Watermelon	Watermelon	Watermelon

The images are linked with the ImageIds and are split into train, validation, & test directories. “fruit” is the category of the images selected for this training.

I was able to convert the native pytorch model into ONNX based model. However, I was unable to convert it into a TF-RT based model. The code and models are available at https://github.com/jroshanucb/py-onnx-tfrt.git

The training performance is available in the repo in epochs.txt.