Inference acceleration
on Nvidia Jetson AGX embedded systems
for detection of objects at roads surface

By FourthBrain-Viziochron

02.10.2022

Project Proposal

Given there is a Tensorflow 1.15 frozen graph model used by a startup that processes an input video or sequence of images for detecting objects, e.g. cigarette butts on roads with 2 fps rate, there is a need requested from the industry startup to improve performance of the model without changing the model also without retraining the model, or to achieve the same by changing the used pipeline in a way so that the processing performance will increase from 2fps to at least 6 fps. The objective needs to be achieved without significant increase of temperature of the unit, which would result in overheating otherwise during all seasons, but for winter.

However, among possibilities to change the existing implementation in order to achieve a better performance, were considered methods to leverage from Nvidia GPU acceleration by converting the model to TensorRT framework via onnx intermediary conversion [1]. But after the preliminary research it turned out that a model created with tensorflow 1.15 can highly likely only be easily used with TensorRT benefiting from Nvidia GPU acceleration support by[via] converting the frozen graph to uff for further integration with TensorRT[2]. Achieving ONNX TensorRT acceleration, as was pointed out by the vendor(Nvidia - AastaLLL, 2022) would only be supported for Tensorflow v2 models [3].

Project MVP

Research

Initial research has shown that the easiest way to achieve TensorRT acceleration on Nvidia GPU devices is to convert a model from Tensorflow into ONNX format [1], then execute with TensorRT engine:

```
/usr/src/tensorrt/bin/trtexec --onnx=[your/file]
```

However, further research has shown that it is only supported by Nvidia for models originated in Tensorflow v2. Moreover, for Tensorflow 1.15 as the method to achieve TensorRT support, the UFF format has been chosen for further attempts.

Testing hypotheses

TF-TRT

According to existing data, first attempts did show that the supplied model won't run or convert using TF-TRT method [5] listed below due to the mismatch of the current environment version of Tensorflow 1.15.5 with the original Tensorflow environment version which were used to export the model where Tensorflow 1.15.4 has been used.

```
import tensorflow as tf
from tensorflow.python.compiler.tensorrt import trt_convert as trt
with tf.Session() as sess:
    # First deserialize your frozen graph:
    with tf.gfile.GFile("frozen_cortexia_graph.pb", 'rb') as f:
        frozen_graph = tf.GraphDef()
        frozen_graph.ParseFromString(f.read())
    # Now you can create a TensorRT inference graph from your
    # frozen graph:
    converter = trt.TrtGraphConverter(
            input_graph_def=frozen_graph,
            nodes_blacklist=['logits', 'classes']) #output nodes
    trt_graph = converter.convert()
    # Import the TensorRT graph into a new graph and run:
    output_node = tf.import_graph_def(
        trt_graph,
        return_elements=['logits', 'classes'])
    sess.run(output_node)
```

Convert.txt template

TF-UFF

Conversion from TF frozen graph to UFF file has succeeded using the code listed below. However, as arguments which should have been specified as output_nodes was unknown, it has been omitted in the code below which might have reduced the model to a certain extent.

```
import tensorflow as tf
import uff
uff.from_tensorflow_frozen_model(frozen_file="frozen_file.pb", preprocessor=None,
output_filename="test.uff")
```

Next step that will require further investigation is to how convert the **UFF** to **TRT**.

TF-ONNX

Conversion of the model to ONNX model failed with the same error as TF-TRT code, which seems due to mismatch of versions of components of tensorflow. [6]

Tensorflow to TFlite

Did not work in initial attempts due to wrong TF version assumed

Google Cloud Vertex AI import

It turned out that importing the frozen graph into Google Cloud ML models requires converting the frozen graph to SavedModel format [4].

However, converting to TFlite by python or command line code requires the frozen graph to be converted into SavedModel format [7].

Testing results

Testing of several approaches above failed. However it has become possible to crowdsource the issue of the failures using the public nvidia developers forum , also to progress further with the implementation on a basic conversion from frozen graph to nvidia TensorRT .engine model via ONNX intermediary format. However, it turned out that the supplied graph was produced by Tensorflow 2.7, but not with Tensorflow 1.15 as it was pointed out. Moreover. The graph has had the performance already of approximately 5.4 FPS. So the performance gain after converting the model to .engine format was not that significant given that with FP32 it only reached 6.8 FPS rate.

Next section *Implementation* is based mostly on contributions from devtalk community member Naisy based on the public thread [3]

Implementation

Setting up the environment:

Given there was a supplied frozen graph[8] produced by Tensorflow 2.7, also with a test python code[8] It was considered to install Tensorflow 2.7 and related packages system-wide for a Proof of Concept [PoC] id est for a brief test.

The hardware selected for implementation is Nvidia Jetson AGX embedded system Devkit, with Operational System Tegra Linux Ubuntu based on Jetpack 5.0.2 distribution.

Installing Tensorflow GPU

Tensorflow installation has been performed according to the Nvidia guidelines [9]. It turned out that most recent TF2.0 version support the frozen graph

```
sudo apt-get update
sudo apt-get install libhdf5-serial-dev hdf5-tools libhdf5-dev zlib1g-dev zip libjpeg8-dev
liblapack-dev libblas-dev gfortran
sudo apt-get install python3-pip
 sudo pip3 install -U pip testresources setuptools==49.6.0
 sudo pip3 install -U --no-deps numpy==1.19.4 future==0.18.2 mock==3.0.5
keras_preprocessing==1.1.2 keras_applications==1.0.8 gast==0.4.0 protobuf pybind11
cython pkgconfig packaging
sudo env H5PY_SETUP_REQUIRES=0 pip3 install -U h5py==3.1.0

sudo pip3 install --pre --extra-index-url
https://developer.download.nvidia.com/compute/redist/jp/v50 tensorflow
```

Discovering parameters of supplied frozen_graph

I. Doesn’t require specific tensorflow version

```
python3 /usr/lib/python3.8/dist-packages/uff/bin/convert_to_uff.py frozen_file.pb -l
```

II. Second method shared by Naisy, which requires Tensorflow v 1.15.5

```
import tensorflow as tf

# Load frozen graph
graph_def = tf.GraphDef()
```

```
with tf.gfile.GFile("frozen_graph.pb", 'rb') as f:
    graph_def.ParseFromString(f.read())

    print('========== nodes ==========')
    nodes = [n.name + ' => ' +  n.op for n in graph_def.node]
    for node in nodes:
        print(node)

    print('========== inputs ==========')
    input_nodes = [n.name + ' => ' +  n.op for n in graph_def.node if n.op in ('Placeholder')]
    for node in input_nodes:
        print(node)

    print('========== outputs ==========')
    name_list = []
    input_list = []
    for n in graph_def.node:
        name_list.append(n.name)
        for name in n.input:
            input_list.append(name)

    outputs = set(name_list) - set(input_list)
    output_nodes = [n.name + ' => ' +  n.op for n in graph_def.node if n.name in outputs]
    for node in output_nodes:
        print(node)
```

Listing of checking_inputs_outputs.py

Running the conversion scripts:

From Tensorflow 2.0 frozen graph To ONNX

```
time python -m tf2onnx.convert --input frozen_graph.pb --output model.onnx --opset 12
--inputs x_in:0 --outputs decoder/mul_1:0,decoder/Softmax:0
```

Outputs:

```
Instructions for updating:
Use `tf.compat.v1.graph_util.extract_sub_graph`
2022-10-06 13:49:42,876 - INFO - Using tensorflow=2.7.0, onnx=1.11.0,
tf2onnx=1.12.1/b6d590
2022-10-06 13:49:42,876 - INFO - Using opset <onnx, 12>
2022-10-06 13:49:48,233 - INFO - Computed 0 values for constant folding
2022-10-06 13:49:52,516 - INFO - Optimizing ONNX model
2022-10-06 13:49:58,101 - INFO - After optimization: Cast -3 (3->0), Const -19 (146->127),
Identity -2 (2->0), Reshape -1 (3->2), Transpose -144 (146->2)
2022-10-06 13:49:58,373 - INFO -
```

```
2022-10-06 13:49:58,374 - INFO - Successfully converted TensorFlow model frozen_graph.pb to ONNX
2022-10-06 13:49:58,374 - INFO - Model inputs: ['x_in:0']
2022-10-06 13:49:58,377 - INFO - Model outputs: ['decoder/mul_1:0', 'decoder/Softmax:0']
2022-10-06 13:49:58,377 - INFO - ONNX model is saved at model.onnx
```

From ONNX to TRT

```
time /usr/src/tensorrt/bin/trtexec --onnx=model.onnx --saveEngine=model.engine
```

Outputs: model.engine file

```
[10/06/2022-13:25:28] [I] Total GPU Compute Time: 6.98522 s
[10/06/2022-13:25:28] [I] Explanations of the performance metrics are printed in the verbose logs.
[10/06/2022-13:25:28] [I]
&&&& PASSED TensorRT.trtexec [TensorRT v8201] # /usr/src/tensorrt/bin/trtexec
--onnx=model.onnx --saveEngine=model.engine
```

Running TRT Inference

```
python trt_classificator_av.py --model=model.engine --image=testimage.jpg
```

Outcomes: inference has shown ~ 6.8 fps on a larger input image 4107x2743 pixels.

```
load: 2.4099135398864746 sec
1st frame: 0.15322327613830566 sec
infer: 0.14696931838989258 sec
[array([[0.01609352, 0.01584916, 0.0152326 , ..., 0.01504145, 0.01473314,
        0.01656689],
       [0.01590273, 0.01591793, 0.01520135, ..., 0.01501078, 0.01453196,
        0.01645717],
       [0.01578423, 0.01565683, 0.01522273, ..., 0.01504697, 0.01458219,
        0.0160788 ],
       ...,
       [0.01620023, 0.01539864, 0.01554307, ..., 0.01535674, 0.01525709,
        0.01682294],
       [0.01588741, 0.0157625 , 0.01551292, ..., 0.01530198, 0.01531301,
        0.01665569],
       [0.01568917, 0.01575915, 0.01554907, ..., 0.01541131, 0.01540375,
        0.01633411]], dtype=float32), array([[-1.0533712 , -5.4133987 ,  2.8164186 ,
-0.19156438],
       [-0.10044891, -5.237487  ,  3.1390605 ,  0.03426504],
       [ 0.44452453, -4.618743  ,  2.3823073 ,  0.33535177],
       ...,
       [-1.1889708 , -3.897892  ,  3.5675566 ,  0.461129  ],
       [-0.9122926 , -3.2173848 ,  2.344221  ,  0.83963376],
       [-1.0568575 , -3.2151482 ,  1.8535609 ,  0.6914637 ]],
      dtype=float32)]
```

Comparing performance with the original inputs :

Running inference

```
python3 test_frozen_model_TF1-2.py
```

Outputs indicate approximately 5.4 FPS on 1920x1080 image

```
You may not need to update to CUDA 11.1; cherry-picking the ptxas binary is often sufficient.
9.352937936782837
0.19011902809143066
0.19014286994934082
0.18929290771484375
0.18883967399597168
0.18741393089294434
0.1872696876525879
0.1874561309814453
0.19055795669555664
0.19225692749023438
[[-2.2045977   0.03823908  0.5027579   0.8422655 ]
 [-2.523407    0.24567053  0.46661878  0.7061642 ]
 [-3.470491   -0.6949028   1.1846757   0.46239161]
 ...
 [-2.5493495  -0.92630035  1.6034929  -0.12209126]
 [-2.1062684  -0.00445377  1.3455819   0.30410582]
 [-1.6819496  -0.76968026  0.96385396  0.3319861 ]]
```

Performance of the converted model turned out to be approximately the same [6.8 fps] on an image with larger size compared with the original model 5.4fps on an image with less size.

Reducing precision in order to increase the performance

```
# FP32
time /usr/src/tensorrt/bin/trtexec --onnx=model.onnx --saveEngine=model_fp32.engine

# FP16
time /usr/src/tensorrt/bin/trtexec --onnx=model.onnx --saveEngine=model_fp16.engine --fp16
# infer
python trt_classificator_av.py --model=model_fp32.engine --image=testimage.jpg
python trt_classificator_av.py --model=model_fp16.engine --image=testimage.jpg
```

[Naisy, 2022]

```
# INT8
time /usr/src/tensorrt/bin/trtexec --onnx=model.onnx --saveEngine=model_int8.engine --int8
```

Implementing Jupyter pipeline interface on Nvidia Jetson embedded platform for a purpose of demo [by naisy]

Although all processing steps were done from the command line, in order to align with pipeline mood there was deployed a web interface on the target platform that allows the execution of the code from the web interface.

However, the Jupyter pipeline solution was crowdsourced so the steps were provided by the *Nais*y user from devtalk forum.

```
git clone https://github.com/naisy/docker
cd docker
sudo su
# Tensorflow 2.7.0
./run-jetson-jp461-base.sh
# Tensorflow 1.15.5
./run-jetson-jp461-donkeycar-overdrive3.sh

# JupyterLab
http://jetson_ip_address:8888
pass: jupyter
```

Further Research

The model supplier requested to investigate the option of batch processing with the existing model. However, it turned out that the supplied frozen graph had the batch information reduced. Moreover, it will require the supplier to export from SavedModel to frozen graph format preserving the batch information, in order to address the batch investigation concerns.

Conclusion:

Conducted research has shown that while converting a frozen graph to TRT model does't result in significant performance increase, reduction of precision of the TRT model from FP32 to FP16 or further to INT8 results in increase of the processing speed 2x times[Naisy, 2022]. Moreover, it was pointed out that the Nvidia Deepstream with INT8 model reaches up to 44 fps, and 27fps with FP16.Moreover, it will require further research and testing to study the applicability of use of the Deepstream due to its complexity.

References:

1. Github Tensorflow-ONNX. Retrieved from https://github.com/onnx/tensorflow-onnx
2. Github. Tensorflow-UFF Sample. Retrieved from https://github.com/NVIDIA/TensorRT/tree/main/samples/sampleUffMNIST
3. Nvidia Devtalk Forum. Retrieved from https://forums.developer.nvidia.com/t/tensorflow-model-acceleration-on-agx/229241/
4. StackOverflow. Saving Frozen Graph to Saved_Model format. Retrieved from https://stackoverflow.com/questions/44329185/convert-a-graph-proto-pb-pbtxt-to-a-savedmodel-for-use-in-tensorflow-serving-o/44329200#44329200
5. Converting Tensorflow Frozen Graph to TensorRT framework. Retrieved from https://docs.nvidia.com/deeplearning/frameworks/tf-trt-user-guide/index.html#using-frozengraph
6. Github. Tensorflow-onnx. Retrieved from https://github.com/onnx/tensorflow-onnx
7. Converting Tensorflow to TFlite. Retrieved from https://www.tensorflow.org/lite/models/convert/convert_models
8. Github. Capstone repository. Retrieved from https://github.com/AndreV84/capstone
9. Nvidia. Installing Tensorflow to Jetson devices. Retrieved from
10. https://docs.nvidia.com/deeplearning/frameworks/install-tf-jetson-platform/index.html