**Source: Interdigital New York, Qualcomm inc.**

**Title: [FS\_AI4Media] pCR on Split inferencing for object detection scenario**

**Spec: 3GPP TR 26.847 v0.3.0**

**Agenda item: 9.6**

**Document for: Discussion and Agreement**

1. **Introduction**

We provided several contributions to the permanent document as well as scripts to 5G-MAG for the evaluation of object detection scenarios

This contribution updates the TR with a split inferencing for object detection scenario performed on *ssd\_resnet* and *retinanet* models including for each model:

* Simple and multi-branch split operations,
* ONNX models splitability evaluation,
* Intermediate data size evaluation from parsing ONNX model and at inference stage,
* Multi-branch scripts results on inference (added with the number of tensors)
* Intermediate data compression.

The contribution includes changes in the reference clause 2, a new scenario description clause 5.3 and scripts in annex B.3

The contribution also refines the multi-branch scripts results (clause 5.3.9.7) and examples of predictions with ssd\_resnet and retinanet (clause 5.3.9.8) from the text of the PD.

1. **Reason for Change**

Update the TR with the split inferencing for object detection scenario of the PD

1. **Proposal**

It is proposed to agree the following changes to 3GPP TR 26.847

\* \* \* Begin first changes \* \* \* \*

# 2 References

The following documents contain provisions which, through reference in this text, constitute provisions of the present document.

- References are either specific (identified by date of publication, edition number, version number, etc.) or non‑specific.

- For a specific reference, subsequent revisions do not apply.

- For a non-specific reference, the latest version applies. In the case of a reference to a 3GPP document (including a GSM document), a non-specific reference implicitly refers to the latest version of that document in the same Release as the present document.

1. [1] 3GPP TR 21.905: "Vocabulary for 3GPP Specifications".Ssd\_resnet model: <https://github.com/nikheelpandey/SSD_RESNET>
2. Retinanet model : [https://github.com/onnx/models?tab=readme-ov-file#object\_detection](https://github.com/onnx/models?tab=readme-ov-file" \l "object_detection)
3. 5G-MAG repository for AI4Media evaluations: <https://github.com/5G-MAG/rt-ai-ml-evaluation-framework>
4. VGG16 model: https://pytorch.org/vision/main/models/generated/torchvision.models.vgg16.html
5. ONNX/ <https://onnx.ai/onnx/api/utils.html>
6. SFU object repository <https://dash-large-files.akamaized.net/WAVE/3GPP/AIML/ReferenceDataSets/sfuhwobjects.tar.gz>
7. 5G-MAG scripts for ssd\_resnet: <https://github.com/5G-MAG/rt-ai-ml-evaluation-framework/blob/development/scripts/objectdetection/ssd300/>
8. 5G-Mag split multibranch repository: <https://github.com/5G-MAG/rt-ai-ml-evaluation-framework> in a branch called split\_multibranch.
9. NNC codec: <https://github.com/fraunhoferhhi/nncodec>
10. Netron: Visualization tool for neural network, deep learning and machine learning models. <https://netron.app/>

\* \* \* End first changes \* \* \* \*

\* \* \* Begin second changes \* \* \* \*

## 5.3 Scenario 2: Split inferencing for object detection and labelling

### 5.3.1 Motivation and use case relevance

Object detection and tracking find prevalent applications in today’s world. These applications range from surveillance, image-based gallery and web search, media annotation, autonomous driving and more.

TR 22.874 section 5.2 describes these scenarios where deep learning-based object detection and tracking is performed.

### 5.3.2 Description of the scenario

In this scenario, a pre-trained model is used to detect objects in a video sequence. The output of the inference may consist of the following:

* Detected object labels per image
* Bounding boxes for the detected objects
* Masks describing pixel-accurate location of the object

In this scenario, it is assumed that the end device is resource constrained and may not have sufficient memory/processing capabilities, or battery power to perform the object detection task.

It is proposed that by splitting the model into 2 parts, where one part is inferred in the end device and the other part is inferred in the network, the end device will be able to perform the inference within its capabilities.

Two configurations are possible, based on the exact use cases:

* The image/video is captured on the device and inference is run on the image/video to produce feature maps that are then sent to the network for further inference. This step may be performed to protect user privacy. The device will then receive the results once the inference is finalized on the network. An example of such a use case is image/video-based web search, where the user captures an image/video and receives web search results. Another such use case is where the user captures an image/video and attempts to remove a specific object from the image/video.
* The image/video is provided by a content provider and processed by the network to enable the user to perform different tasks. The video is processed by a deep network to produce distilled features, which are then used by the device to perform task-specific inference. Different users viewing the same image/video may run different tasks. An example of such a use case is a sports game streaming service, where different users may have different interests in the game. One user may configure their application to track and annotate the players of their favourite team. Another user may be interested in extracting statistics about the ball. The core of the network produces a set of features that can be used to perform both tasks, where each user will run the model head specific to their selected task.

### 5.3.3 Supporting companies and 3GPP members

* Qualcomm,
* Interdigital.

### 5.3.4 Anchor AI/ML DNN model(s) for the scenario

The evaluation using the PyTorch framework includes several DNN models belonging to the table below:

* retinanet [x2]
* ssd\_resnet: The SSD300 model from Nvidia [x1]

| **Model** | **Size (MB)** | **No. of parameters** |
| --- | --- | --- |
| retinanet | 146 MB | About 34 to 45 millions |
| ssd\_resnet SSD300 (ResNet-50) | 89 MB | About 23 millions |

### 5.3.5 Testbed architecture and anchors

The testbed architecture for this scenario is based on this figure from clause 4.2.



**Figure 5.3.5-1 Testbed architecture for the scenario**

The split configurations for the scenario are compared to three anchors:

1. Where the anchor model is fully inferenced on the device.
2. Where the anchor model is fully inferred on the network.
3. Where the anchor model is split between the device and the network for at least the first layers of the model to meet the privacy requirements as described in 10.X.1.

The anchor model used is shown in clause 10.3.4.

Test network latencies are not considered to ensure scenario reproducibility.

Multiple model split configurations are considered as described in clause 10.2.6.

### 5.3.6 Test configuration factors, constraints, and settings

Split configurations can include different computational capabilities (CPU/GPU), encoding/decoding functions (optimization and/or compression/decompression), as well as serialization/deserialization functions.

. 

**Figure 5.3.6-1 Testbed configuration**

### 5.3.7 Feasibility/performance evaluation metrics and requirements

We evaluate the performances according to the following metrics for each split point configuration: inference latency, output data size, resulting accuracy. The evaluation may include the impact of encoding/decoding functions and/or serialization/deserialization functions on the measured metrics. The delivery latency is estimated from the output data size according to the different bandwidths of the 5G network.

### 5.3.8 Test dataset(s) and scripts for the scenario

The SFU-HW-Objects and the SFU-HW-Tracking datasets are used for this evaluation scenario.

A set of scripts is made available under the 5G-MAG rt-ml-ai-evaluation-framework repository [x3]. Detailed scripts API is provided in Annex B.3

Two models were evaluated with different scripts adapted for each model (ssd\_resnet, retinanet)

### 5.3.9 Results for single and multi-branch split on ONNX models

#### Introduction

ONNX provides a function *extract\_model()* enabling the extraction of a sub-model from an ONNX model [x5] as shown below

**A screenshot of a computer

Description automatically generated**

**Figure 5.3.9.1-1 ONNX extract\_model function**

This function is already used by scripts in 5G-MAG repository [x3], such as split\_onnx.py and split\_retinanet.py

Single branch or multi-branches split script making use of extract-model function are described in clause 5.3.9.2 and 5.3.9.3. Specific issues regarding input tensors and output results applied to multi-branch split are also showed, when Part II needs model input tensor and, in §2.4, when Part I generates partial output result of the full outputs results.

#### Bottleneck/single branch split

For some models, or some parts of a model, a node is connected with only one input node and one output node. For example, it is the case for all nodes of the VGG16 model [x4], and for some nodes of ssd\_resnet [x1]. An overview of the beginning of VGG16 opened with Netron [x10] is provided below.

A diagram of a computer

Description automatically generated with medium confidence

**Figure 5.3.9.2-2 VGG16 layers visualisation with Netron**

In order to split just before the node 5, “vgg0\_conv2\_fwd”:

A diagram of a computer code

Description automatically generated with medium confidence

**Figure 5.3.9.2-3 Split VGG16 at node 5 “vgg0\_conv2\_fwd” split**

The intermediate data communicated between the two sub models will be the tensor “vgg0\_pool0\_fwd”.

To get the two sub models, you need to give the tensor name of the input and the tensor name of the output of each part to the *extract\_model* function.

For the first part, the following needs to be provided:

* input: [“data”] (model input)
* output: [“vgg0\_pool0\_fwd”] (split tensor name)

For the second part, the following needs to be provided:

* input: [“vgg0\_pool0\_fwd”] (split tensor name)
* output: ['vgg0\_dense2\_fwd'] (model output)

A single branch script ‘split\_onnx.py’ is used to split the model and detailed in clause B.3.3. The multi branch script ‘split\_onnx\_multi.py’ detailed in clause B.3.4 can be used as well

#### Multi branches split

For some models, or some parts of a model, a node is connected to several nodes for its input, and/or several nodes for its output. Below is an example for the resnet model [x1] split at node 6 with an overview of part I and Part II opened with Netron [x10].



**Figure 5.3.9.3-1 Split illustration of resnet model at node 6 with Netron**

The intermediate data communicated between the two submodels will be the tensors:[“/feature\_extractor/feature\_extractor/feature\_extractor.4/feature\_extractor.4.0/conv2/Conv\_output\_0”, “/feature\_extractor/feature\_extractor/feature\_extractor.3/MaxPool\_output\_0”].

To get the two sub models, you need to give the tensor name of the input and the tensor name of the output of each part to the *extract\_model* function.

For the first part, the following data needs to be provided

* input: [“input”] (model input)
* output: [“/feature\_extractor/feature\_extractor/feature\_extractor.4/feature\_extractor.4.0/conv2/Conv\_output\_0”, “/feature\_extractor/feature\_extractor/feature\_extractor.3/MaxPool\_output\_0”]

For the second part, the following needs to be provided:

* input: [“/feature\_extractor/feature\_extractor/feature\_extractor.4/feature\_extractor.4.0/conv2/Conv\_output\_0”, “/feature\_extractor/feature\_extractor/feature\_extractor.3/MaxPool\_output\_0”]
* output: [“output1”,” output2”] (model output)

The multi branch script ‘split\_onnx\_multi.py’ is used to split the model and detailed in clause B.3.4 can be used as well

#### AI/ML model splitability from ONNX model assessment

The next table gives a status of the current assessed models regarding the split operations with ONNX [x5]. Process inference was run on both models ssd-resnet and retinanet.

ONNX multi-branch split function APIs and scripts were used on different model as follows:

**Table 5.3.9-6-1: Split operations with ONNX model files**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Model name** | **task** | **Splitted** | **Verified with Inference** | **Multi-branch** | **Nb Maximum branches** |
| ssd\_resnet | Object detection | yes | yes | Yes | 13 |
| retinanet | Object detection | yes | yes | Yes | 85 |
| yolo-v2 | Object detection | yes | no | Yes | 3 |
| yolo-v4 | Object detection | yes | no | Yes | 7 |
| tinyyolov2 | Object detection | yes | no | no | 1 |
| mobilenetv3 | Image classification | yes | no | Yes | 3 |
| Resnet152 | Image classification | yes | no | Yes | 2 |
| Resnet18 | Image classification | yes | no | Yes | 2 |
| ResNext50FF | Image classification | yes | no | Yes | 2 |
| squeezenet | Image classification | yes | no | Yes | 2 |
| efficientnet | Image classification | yes | no | Yes | 4 |
| caffenet | Image classification | yes | no | no | 1 |
| inception | Image classification | yes | no | Yes | 5 |
| vgg16 | Image classification | yes | no | No | 1 |
| nerf | 3D model rendering | yes | no | Yes | 36 |

The table above demonstrates that a large set of ONNX files was successfully split.

#### Multi-branch split operation analysis

**SSD\_RESNET**

* Intermediate data contains between 1 and 13 tensors (number of branches)
* Input anchor and input of part I is [“input”]
* Output anchor is [“output1” ,“output2”]
* From nodes 1 to 153
  + Input of Part II is exactly the output of Part I
  + Part II is generating anchor outputs [“output1” ,“output2”]
* Node 154
  + Part I is generating one of the anchor outputs: “output1”.
  + Part I output is ['/Reshape\_1\_output\_0', '/Reshape\_3\_output\_0', '/Reshape\_5\_output\_0', '/Reshape\_7\_output\_0', '/Reshape\_9\_output\_0', '/Reshape\_11\_output\_0', 'output1']
  + Input of Part II is the output of Part I except ‘output1”  
    ['/Reshape\_1\_output\_0', '/Reshape\_3\_output\_0', '/Reshape\_5\_output\_0', '/Reshape\_7\_output\_0', '/Reshape\_9\_output\_0', '/Reshape\_11\_output\_0']
  + Part II is generating anchor outputs [“output2”]
  + A consolidation is needed to rebuild the output anchor [“output1” ,“output2”]

**RETINANET**

* Intermediate data contains between 1 and 85 tensors (number of branches)
* Input anchor and input of part I is [“input\_images”]
* Output anchor is [“2734”, “2712”,”2713”]
* From nodes 1 to 65:
  + Part I is generating intermediate data,
  + Part II needs to have the input “input\_images” in addition to output of part I,
  + Part II is generating is [“2734”, “2712”,”2713”].
* From nodes 66 to 2230:
  + Part I is generating only intermediate data, between 7 and 85 tensors
  + Input of Part II is exactly the output of Part I
* Node 2230
  + Part I is generating 8 tensors including anchor outputs “2712”:  
    ['/Concat\_27\_output\_0', '/Slice\_50\_output\_0', '/Gather\_output\_0', '/Cast\_4\_output\_0', '/Gather\_1\_output\_0', '/Cast\_3\_output\_0', '/Gather\_68\_output\_0', '2712']
  + Part II input is the part I output except the anchor output ‘2712’  
    ['/Concat\_27\_output\_0', '/Slice\_50\_output\_0', '/Gather\_output\_0', '/Cast\_4\_output\_0', '/Gather\_1\_output\_0', '/Cast\_3\_output\_0', '/Gather\_68\_output\_0']
  + Part II is generating anchor outputs [“2734”, “2713”].
  + A consolidation is needed to rebuild the output anchor [“2734”, “2712”,”2713”]
* Nodes 2231 to 2248
  + Part I is generating between 6 and 8 tensors including anchor outputs [“2712”,”2713”]
  + Part II input is the part I output except the anchor outputs [“2712”,”2713”]
  + Part II is generating anchor outputs [“2734”]
  + A consolidation is needed to rebuild the output anchor [“2734”, “2712”,”2713”]

#### Intermediate data size evaluation

#### Evaluation of the extraction of intermediate data size from ONNX model file

In this section, we tried to determine the size of the intermediate data by parsing ONNX model file, without any inferencing. This evaluation is made on models ssd\_resnet [x1] and retinanet [x2].

**SSD\_RESNET**

**First step:**

We parsed the ONNX file to extract the tensors, their types and their shape information and evaluated the tensor size for each tensor.

Example of size calculation for some tensors:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Tensor name | Tensor shape | Number of Data | Data type | Data size (bytes) | Tensor size  (bytes) |
| “input” | ['1', '3', '300', '300'] | 270000 | float32 | 4 | 1080000 |
| “Conv\_output\_0” | ['1', '64', '150', '150'] | 1440000 | float32 | 4 | 5760000 |
| “output1” | ['1', '4', '8732'] | 34928 | float32 | 4 | 139712 |
| “output2” | ['1', '81', '8732'] | 707292 | float32 | 4 | 2829168 |

**Second step:**

We extracted the list of the tensors that composes the intermediate data at each potential split point. We calculated the intermediate data size for each split points, by adding up the size of each tensor (first step).

The Figure 1 shows the intermediate data size in blue and, the number of tensors (i.e. number of branches) of the split points in green.

A graph with lines and lines

Description automatically generated with medium confidence

**Figure 5.3.9.6.1-1 Intermediate data size and number of branches per node**

**Third step**

We cross-checked these intermediate data sizes with the ones obtained by inference.

**RETINANET**

**First step:**

We parsed the ONNX file to extract the tensors, their types and their shape information and evaluated the tensor size for each tensor.

Example of tensors we obtained:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Tensor name | Tensor shape | Number of Data | Data type | Data size (bytes) | Tensor size  (bytes) |
| input\_images | ['bs', '3', 'h', 'w'] | bs\*3\*h\*w | float32 | 4 | 4\*bs\*3\*h\*w |
| /Split\_output\_0 | ['unk\_\_0', '3', 'h', 'w'] | Unk\_0\*3\*h\*w | float32 | 4 | 4\*Unk\_0\*3\*h\*w |
| 2712 | ['Gather2712\_dim\_0'] | Gather2712\_dim\_0 | float32 | 4 | 4\*Gather2712\_dim\_0 |
| 2713 | ['Gather2712\_dim\_0'] | Gather2712\_dim\_0 | int64 | 8 | 8\*Gather2712\_dim\_0 |
| 2734 | ['Concat2734\_dim\_0', '4'] | Concat2734\_dim\*4 | float32 | 4 | 4\*Concat2734\_dim\*4 |

However, unlike ssd\_resnet, the shape information contains variable parameters:

* Some of these are the size of the input data: height(h) and width(w).
* Others are unknown: “Unk\_0”, “Concat2734\_dim”, “Gather2712\_dim\_0”, “Gather2712\_dim\_0”.

Consequently, it is not possible to carry out the second step to calculate the intermediate size of all the split point configurations.

#### Evaluation of the correlation between the size of the input data size and the size of the intermediate data at inference time

In this section, we performed inference on images of different dimensions at several split points. This evaluation is made on models ssd\_resnet [x1] and retinanet [x2].

**SSD\_RESNET**

Figure below shows the evaluation of intermediate data size at inference stage.

A graph with different colored lines

Description automatically generated

**Figure 5.3.9.6.2-1 inference experimentation with ssd\_resnet with images having various dimensions**

We obtained the same values as in section 5.3.9.6.1 above because the image is resized at a fix dimension (300,300) at the pre-processing stage.

**RETINANET**

We calculated the size of intermediate data for a set of different image dimensions.

Figure 3 shows the size of the intermediate data obtained after inferencing images with different dimensions at different split points node.

Figure 4 also shows the size of the intermediate data obtained after inferencing images with different dimensions at different split points node with a bar graph representation.

A graph with different colored lines

Description automatically generated

**Figure 5.3.9.6.2-2 inference experimentation with retinanet with images having various dimensions**

A graph with different colored lines

Description automatically generated

**Figure 5.3.9.6.2-3 inference experimentation with *retinanet* with images having various dimensions (Bar graph)**

We observed a variation in the intermediate size with the variation of the input image dimension. However, we did not observe a direct link between the size of the input image in number of pixels and the size of the intermediate data.

By analyzing the *retinanet* tensors, we observed the presence of a tensor “'/transform/Resize\_output\_0'” at node 126, which explains some clipping effects after this node. For example, intermediate data size is the same for images having dimension (3840\*2160, 1920\*1080,1280\*720) for node 150 (so after 126). We still observed differences for other dimensions meaning that other processing is done beside the resizing.

#### Conclusion for evaluation of intermediate data size

From these experimentations, we observed that it is not always possible to obtain the intermediate data size before the inference phase, for two reasons:

1. Models can be designed to infer media data with variable size.
2. Models can produce intermediate tensor data of varying shape whose relationship to the characteristics of the input data is unknown.

Consequently, for some models, the size of the intermediate tensors can only be calculated at inference stage. In this case, the size of the tensors composing the intermediate data must be transmitted as metadata with the intermediate data.

#### Multi-branch scripts results

Intermediate data size for some split point of ssd\_resnet. Results are independent of the original image/frame size as the model requests an image resized at a fix dimension 300x300.

**Table 5.3.9.7-1: Multi-branch script results for ssd\_resnet**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Nodes | | Intermediate data size (Mbytes) | *Nb branches* | *Estimated network latency (ms)* | | |
| Split | Node name |  |  | *at 500 Mbps* | *at 1Gbps* | *at 5 Gbps* |
| Anchor (full network inference) |  | 1.08 (RGB) |  | *17,28* | *8,64* | *1,73* |
| split\_node\_1 | /feature\_extractor/feature\_extractor/feature\_extractor.2/Relu | 5.76 | *1* | *92,16* | *46,08* | *9,22* |
| split\_node\_3 | /feature\_extractor/feature\_extractor/feature\_extractor.4/feature\_extractor.4.0/conv1/Conv | 1.44 | *1* | *23,04* | *11,52* | *2,30* |
| split\_node\_10 | /feature\_extractor/feature\_extractor/feature\_extractor.4/feature\_extractor.4.0/relu\_2/Relu | 5.76 | *1* | *92,16* | *46,08* | *9,22* |
| split\_node\_17 | /feature\_extractor/feature\_extractor/feature\_extractor.4/feature\_extractor.4.1/relu\_2/Relu | 5.76 | *1* | *92,16* | *46,08* | *9,22* |
| split\_node\_24 | /feature\_extractor/feature\_extractor/feature\_extractor.4/feature\_extractor.4.2/relu\_2/Relu | 5.76 | *1* | *92,16* | *46,08* | *9,22* |
| split\_node\_25 | /feature\_extractor/feature\_extractor/feature\_extractor.5/feature\_extractor.5.0/conv1/Conv | 5.76 | *1* | *92,16* | *46,08* | *9,22* |
| split\_node\_39 | /feature\_extractor/feature\_extractor/feature\_extractor.5/feature\_extractor.5.1/relu\_2/Relu | 2.96 | *1* | *47,36* | *23,68* | *4,74* |
| split\_node\_46 | /feature\_extractor/feature\_extractor/feature\_extractor.5/feature\_extractor.5.2/relu\_2/Relu | 2.96 | *1* | *47,36* | *23,68* | *4,74* |
| split\_node\_54 | /feature\_extractor/feature\_extractor/feature\_extractor.6/feature\_extractor.6.0/conv1/Conv | 2.96 | *1* | *47,36* | *23,68* | *4,74* |
| split\_node\_62 | /feature\_extractor/feature\_extractor/feature\_extractor.6/feature\_extractor.6.1/conv1/Conv | 5.91 | *1* | *94,56* | *47,28* | *9,46* |
| split\_node\_68 | /feature\_extractor/feature\_extractor/feature\_extractor.6/feature\_extractor.6.1/relu\_2/Relu | 5.91 | *1* | *94,56* | *47,28* | *9,46* |
| split\_node\_69 | /feature\_extractor/feature\_extractor/feature\_extractor.6/feature\_extractor.6.2/conv1/Conv | 5.91 | *1* | *94,56* | *47,28* | *9,46* |
| split\_node\_75 | /feature\_extractor/feature\_extractor/feature\_extractor.6/feature\_extractor.6.2/relu\_2/Relu | 5.91 | *1* | *94,56* | *47,28* | *9,46* |
| split\_node\_76 | /feature\_extractor/feature\_extractor/feature\_extractor.6/feature\_extractor.6.3/conv1/Conv | 5.91 | *1* | *94,56* | *47,28* | *9,46* |
| split\_node\_82 | /feature\_extractor/feature\_extractor/feature\_extractor.6/feature\_extractor.6.3/relu\_2/Relu | 5.91 | *1* | *94,56* | *47,28* | *9,46* |

Intermediate data size for some split point of retinanet. For this model, results are dependent of the original image/frame size. Results of the below table are for an image of size 640x428

**Table 5.3.9.7-1: Multi-branch script results for retinanet – input image dimension 640x428**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Nodes | | Intermediate data size (Mbytes) | *Nb branches* | *Estimated network latency (ms)* | | |
| Split | Node name |  |  | *at 500 Mbps* | *at 1Gbps* | *at 5 Gbps* |
| Anchor (full network inference) |  | 0,82 |  | *13,15* | *6,57* | *1,31* |
| split\_node\_100 | /transform/Cast\_2 | 3,35 | *72* | *53,64* | *26,82* | *5,36* |
| split\_node\_200 | /transform/Cast\_19 | 11,55 | *74* | *184,76* | *92,38* | *18,48* |
| split\_node\_300 | /backbone/body/layer3/layer3.5/relu\_2/Relu | 58,43 | *75* | *934,94* | *467,47* | *93,49* |
| split\_node\_400 | /head/classification\_head/conv/conv.2/conv.2.1/Shape | 63,62 | *80* | *1017,95* | *508,98* | *101,80* |
| split\_node\_500 | /head/classification\_head/Transpose\_1 | 94,73 | *73* | *1515,65* | *757,83* | *151,57* |
| split\_node\_600 | /head/classification\_head/conv/conv.2/conv.2.1\_3/InstanceNormalization | 98,33 | *65* | *1573,35* | *786,68* | *157,34* |
| split\_node\_700 | /head/classification\_head/Reshape\_8 | 98,85 | *53* | *1581,67* | *790,84* | *158,17* |
| split\_node\_800 | /head/regression\_head/conv/conv.2/conv.2.1\_1/Constant\_1 | 108,82 | *46* | *1741,15* | *870,57* | *174,11* |
| split\_node\_900 | /head/regression\_head/Unsqueeze\_10 | 101,71 | *40* | *1627,38* | *813,69* | *162,74* |
| split\_node\_1000 | /head/regression\_head/conv/conv.2/conv.2.1\_4/Reshape | 101,81 | *25* | *1628,89* | *814,45* | *162,89* |
| split\_node\_1100 | /anchor\_generator/Cast\_7 | 69,31 | *25* | *1109,01* | *554,51* | *110,90* |
| split\_node\_1200 | /anchor\_generator/Reshape\_5 | 69,43 | *31* | *1110,96* | *555,48* | *111,10* |
| split\_node\_1300 | /anchor\_generator/Cast\_45 | 72,19 | *28* | *1154,98* | *577,49* | *115,50* |
| split\_node\_1400 | /Div | 72,23 | *18* | *1155,71* | *577,85* | *115,57* |
| split\_node\_1500 | /Reshape\_1 | 22,44 | *27* | *359,05* | *179,53* | *35,91* |
| split\_node\_1600 | /Slice\_21 | 18,08 | *28* | *289,22* | *144,61* | *28,92* |
| split\_node\_1700 | /Constant\_93 | 4,58 | *32* | *73,23* | *36,61* | *7,32* |
| split\_node\_1800 | /Gather\_37 | 1,45 | *26* | *23,28* | *11,64* | *2,33* |
| split\_node\_1900 | /Max\_4 | 1,21 | *26* | *19,31* | *9,65* | *1,93* |
| split\_node\_2000 | /Constant\_195 | 0,33 | *30* | *5,23* | *2,62* | *0,52* |
| split\_node\_2100 | /Constant\_225 | 0,09 | *27* | *1,47* | *0,73* | *0,15* |
| split\_node\_2200 | /Min\_9 | 0,08 | *21* | *1,21* | *0,61* | *0,12* |

Results shows that intermediate data size may be relatively large resulting in a relatively large network latency. Intermediate data compression can be considered to significantly reduce intermediate data size, in return for potential additional latency due to compression/decompression time.

#### Example of predictions with ssd\_resnet and retinanet

The table below shows example of predictions with ssd\_resnet and retinanet models applied on the first frame of the video FourPeople\_1280x720\_60.mp4 from SFU-HW-Objects.

**Table 5.3.9.8-2: scripts predictions for ssd\_resnet and retinanet**

|  |  |  |
| --- | --- | --- |
| Ground Truth | Predictions | |
| ssd\_resnet | retinanet |
| A group of people sitting at a table  Description automatically generated | A group of people sitting at a table  Description automatically generated | A group of people sitting at a table  Description automatically generated |
| person 427 226 704 720  person 717 200 988 485  person 933 163 1281 517  person 89 151 130 196  person 131 154 159 191  person 104 201 135 238  person 144 192 169 227  person 196 123 277 261  person 328 216 363 244  person 384 244 422 337  person 518 123 590 236  person 429 166 460 195  person 451 200 465 224  person 476 192 493 214  person 466 158 479 187  person 602 220 618 242  person 600 242 623 265  person 636 242 659 265  person 634 217 653 238  person 678 242 710 319  person 658 253 684 308  potted\_plant 1 216 175 566  cup 537 449 589 503  cup 682 443 730 501  cup 1051 446 1098 509  person 189 223 441 721  chair 921 310 1012 480  chair 658 325 733 481  cup 343 401 399 455 | bottle 502 438 36 62 0.51  dining table 32 481 674 214 0.52  dining table 7 464 1237 213 0.55  person 1052 169 219 345 0.67  person 180 202 255 313 0.80  person 741 195 245 288 0.82  person 434 162 275 | person 999 166 280 327 0.83  person 190 221 239 288 0.83  person 749 205 233 285 0.82  person 446 232 232 324 0.82  potted\_plant 0 198 249 331 0.74  cup 1053 451 33 51 0.66  cup 684 448 43 50 0.64  cup 723 428 36 62 0.60  cup 541 456 41 48 0.57  chair 931 318 70 149 0.57  laptop 95 148 140 135 0.52  chair 651 336 71 148 0.50  cup 500 442 36 64 0.50  chair 447 290 248 204 0.45 dining\_table 0 483 798 223 0.45 cup 347 463 40 73 0.40 |
|  | mAP Score | |
| Ssd\_resnet | retinanet |
| A blue rectangular bar graph with text  Description automatically generated | A graph of different types of plants  Description automatically generated with medium confidence |

Predictions and mAP score are the same for anchor predictions and for split predictions, whatever the split point when the intermediate data are not compressed with loss. When intermediate data are compressed with loss, predictions and mAP score may evolve according to the level of loss, as described in the next section.

#### Intermediate data compression for ssd\_resnet and retinanet

Intermediate data compression is performed with the same multi-branch scripts (Annex B.3.4) using *encode\_algo* parameter for numpy quantization or nnc [x9] with different qc parameters.

* Models
  + ssd\_resnet: ssd\_resnet.onnx generated by <https://github.com/5G-MAG/rt-ai-ml-evaluation-framework/blob/development/scripts/objectdetection/ssd300/convert_ssd300_to_onnx.py> (develop branch)
  + retinanet : retinanet.onnx available at <https://github.com/5G-MAG/rt-ai-ml-evaluation-framework/tree/main/models>
* Split points:
  + ssd\_resnet: This model has 154 nodes.   
    We split this model at index nodes [10, 30,50,70,90,110,130].
  + retinanet: This model has 2248 nodes.   
    We split this model at index nodes [100,400,700,1000,1300,1600,1900]
* Compression
  + Compressed data:
    - Intermediate data generated by the first part of the model are compressed using one of the following algorithms.
    - Compressed Intermediate data is then decompressed and provided for inference of the second part of the model.
  + Compressions algorithm:
    - Numpy quantization from float32 bits to float 16 bits  
      numpy.float16 : Half precision float: sign bit, 5 bits exponent, 10 bits mantissa
    - Neural Network Encoder Decoder by Fraunhofer HHI [x9]  
      This library includes a key parameter in the encoder function that controls the rate-performance trade-off, called Quantization Parameter (QP) that controls the quantization stepsize and thus the rate-performance trade-off for all weight parameters. A lower qp is related to a lower quantization stepsize, which yields a higher bitrate but also a lower model performance degradation. Conversely, increasing the qp value results in a lower bitrate but also in a higher model performance degradation. <https://github.com/fraunhoferhhi/nncodec/wiki/Usage>
* Metrics
  + The accuracy of the results is measured by computing the mAP metrics using the function calculate\_map() of the script [calc\_map.py](https://github.com/5G-MAG/rt-ai-ml-evaluation-framework/blob/main/scripts/objectdetection/calc_map.py) at [x3]

*Note: For practical reasons, we create a version calc\_map\_image\_dataset.py calling the function calculate\_map() on the whole dataset of images*

* Summary of the experiments process:

We decided to evaluate the effects of the quantization only (32 bits to 16 bits) and the effects of quantization and entropy coding (using various qp of Fraunhofer’s nncodec)

* + First step: test with one input image

We first evaluated on one image to design and set up the testbed pipeline. We implemented a first prototype integrating the compression library to quickly present results and get quick feedback on results. These first results were encouraging regarding the reasonable impact of the compression on the final task accuracy.

Outcome:

* + - We found that the studied models were resilient to loss of accuracy with intermediate data degradations.
    - More images should be used for greater statistical robustness on results
  + Second Step: tests have been extended to a full video.

We then evaluated on a full video to average the result on all the frames. We used the video FourPeople\_1280x720\_60.mp4 and ground-truth available at [x3]. This video contains 600 frames.

Outcome:

* + - We found that the processing takes a lot of time (several weeks) with very few differences to the results with one image regarding accuracy and compressed intermediate data size.
    - There was not enough disparity between the frames on the selected video (FourPeople).
  + Third Step: test with a selection of various images.

We decided to evaluate on a large dataset of images based on existing evaluation datasets. We used the coco dataset 2017 validation images with the 2017 Train/Val annotations.

The dataset contains 5000 images with a large diversity on the objects.

We checked the performance of ssd\_resnet on the first 200 images of the dataset.

Outcome:

* + - We noticed that for some of them, ssd\_resnet yields poor results. As the objective was to measure the impact of lossy compression, it was important to ensure good initial accuracy (e.g. If the model does not detect the objects on the images we start at a mAP of 0 and the impact of compression will not be measured.)
  + Fourth step: test with a selection of various images on which ssd\_resnet and retinanet have good accuracy  
    We decided to select the first 50 images of coco dataset where ssd\_resnet performs perfectly. As a result, the mAP score of ssd\_resnet on this dataset is 100.

We kept this same dataset for the evaluation with retinanet.

The score of retinanet on this dataset was very high, 94.21, and considered high enough for our experiments

* Experiments with a dataset of 50 selected images
  + Test set conditions:
    - 50 selected images from coco dataset 2017 where ssd\_resnet score is perfect
* List of 50 selected images:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 000000000285.jpg | 000000042889.jpg | 000000052462.jpg | 000000004395.jpg | 000000029187.jpg |
| 000000000802.jpg | 000000044652.jpg | 000000052507.jpg | 000000004495.jpg | 000000016598.jpg |
| 000000032081.jpg | 000000046804.jpg | 000000052565.jpg | 000000004765.jpg | 000000017031.jpg |
| 000000032285.jpg | 000000049259.jpg | 000000054164.jpg | 000000007784.jpg | 000000017115.jpg |
| 000000032570.jpg | 000000049269.jpg | 000000055072.jpg | 000000007888.jpg | 000000018519.jpg |
| 000000033005.jpg | 000000050844.jpg | 000000031749.jpg | 000000009448.jpg | 000000018737.jpg |
| 000000038070.jpg | 000000051314.jpg | 000000031050.jpg | 000000010764.jpg | 000000020059.jpg |
| 000000038825.jpg | 000000051326.jpg | 000000030675.jpg | 000000010995.jpg | 000000020107.jpg |
| 000000039480.jpg | 000000051712.jpg | 000000015278.jpg | 000000011122.jpg | 000000022396.jpg |
| 000000039670.jpg | 000000051961.jpg | 000000001490.jpg | 000000015746.jpg | 000000022623.jpg |

* + Compression Algorithms
    - No encoding
    - Numpy quantization float32 bits to float 16 bits
    - Lossy compression with nnc, with QP=-38
    - Lossy compression with nnc, with QP=-26
    - Lossy compression with nnc, with QP=-18
    - Lossy compression with nnc, with QP=-14
    - Lossy compression with nnc, with QP=-10
    - Lossy compression with nnc, with QP=-6
    - Lossy compression with nnc, with QP=-4
    - Lossy compression with nnc, with QP=0
  + Results
    - mAP score in function of split points and compression algorithm:

A screenshot of a graph

Description automatically generated

**Figure 5.3.9.9-2 ssd\_resnet map score prediction on dataset 50 selected images**

A screenshot of a computer

Description automatically generated

**Figure 5.3.9.9-3 retiananet map score prediction on dataset 50 selected images**

Outcome:

* + - We notice that mAP score is affected by the choice of the algorithm and the choice of the selected split point
    - We notice that according to the split point it is interesting to not always select the same algorithm to reach a given mAP score
    - Representation of mAP score according to the size of intermediate data:  
      The curve is built by plotting for each compression algorithm the mAP score as a function of the compressed intermediate size.

A screenshot of a computer

Description automatically generated

**Figure 5.3.9.9-4 ssd\_resnet map score prediction on dataset 50 selected images with split at node 10**

A graph with numbers and lines

Description automatically generated

**Figure 5.3.9.9-5 retinanet map score prediction on dataset 50 selected images with split at node 1000**

* Representation of mAP score according to the size of intermediate data and split point:  
  A curve per split point.  
  For each split point, the curve is built by plotting for each compression algorithm the point (compressed intermediate size, mAP score).

A screen shot of a computer

Description automatically generated

**Figure 5.3.9.9-6 ssd\_resnet map score prediction on dataset 50 selected images with 7 splits**

A screen shot of a graph

Description automatically generated

**Figure 5.3.9.9-7 ssd\_resnet map score prediction on dataset 50 selected images with 7 splits -zoom X-axis**

A screen shot of a graph

Description automatically generated

**Figure 5.3.9.9-8 retinanet map score prediction on dataset 50 selected images with 7 splits**

A screen shot of a graph

Description automatically generated

**Figure 5.3.9.9-9 retinanet map score prediction on dataset 50 selected images with 7 splits - zoom X-axis**

* Compression performance   
  Representation of mAP score according to ratio compressed intermediate size / uncompressed intermediate size

For each split point, and each compression algorithm we compute the ratio :

We then plot each point ( compressed ratio percentage, mAP score).

A screen shot of a computer

Description automatically generated

**Figure 5.3.9.9-10 Compression performance with ssd\_resnet**

A screenshot of a computer screen

Description automatically generated

**Figure 5.3.9.9-11 Compression performance with retinanet**

* Outcome:
* Quantization 32 bits to 16 bits offers a compression ratio at 50% with no impact on the accuracy
* Thanks to nnc it is possible to reach higher compression ratio up to 80% and same accuracy
* Above 80% of compression ratio, compression may impact differently the accuracy according to the model
* Analysis
* We notice that the mAP score is affected by both the choice of the algorithm and the choice of the selected split point
* We notice that the selection of the compression algorithm needs to be made according to the selected split point to maintain a given mAP score
* We consider that these results using on-the-shelf compression tools offer interesting compression rates with limited impact on the accuracy

NOTE: MPEG (MPEG-FCM: Feature Coding for Machines) are designing codecs for intermediate data in the context of split computer vision models. The expected compression ratio of the uncompressed feature size verses the compressed feature size in near lossless setting ranges from 6000:1 to 40000:1 on instance segmentation, object detection and object tracking. The obtained compression ratio of intermediate data while preserving near lossless accuracy is defined within a tolerance of 1% drop in task accuracy, relative to the performance achieved by the original task model operating directly on the input data.

NOTE: The MPEG-FCM results have not been evaluated in the context of 3GPP.

\* \* \* End second changes \* \* \* \*

\* \* \* Begin third changes \* \* \* \*

Annex B: Scripts and Datasets

## B.1 Introduction

[Editor’s note: Describe the general aspects regarding the AI/ML software framework, including use of docker container with scripts and datasets].

## B.2 Scripts for the evaluation of compressed AI/ML model transmission

## B.3 Scripts for the evaluation of split inferencing for object detection and labeling

### B.3.1 Introduction

The scripts corresponding to the evaluation of Split and Compression are available in [x8]

The code is available for both Linux and Windows platforms. Tests have been done on different machines with ubuntu 22.04.4 LTS and Git Bash for Windows 2.45.2.

Model sources:

* + ssd\_resnet.onnx is generated by [convert\_ssd300\_to\_onnx.py](https://github.com/5G-MAG/rt-ai-ml-evaluation-framework/blob/development/scripts/objectdetection/ssd300/convert_ssd300_to_onnx.py) and available at [x4].
  + retinanet.onnx is available at [x1] under “models” directory.

### B.3.2 repository and installation

**FPN/retinanet and single branch SSD300 scripts**

The command to clone the git repository is:

|  |
| --- |
| git clone https://github.com/5G-MAG/rt-ai-ml-evaluation-framework.git |

**Multibranch scripts**

The command to clone the git repository is:

|  |
| --- |
| git clone -b split\_multibranch https://github.com/5G-MAG/rt-ai-ml-evaluation-framework.git |

A readme file scripts/objectdetection/multibranch/Readme.md described how it works for splitting operation and inferencing with or without compression using quantization and/or NNCodec [x9]. Scripts may be used without installation of NNCodec.

The documented scripts are the following:

* split\_onnx\_multi.py
* split\_ssd\_resnet.sh
* split\_retinanet.sh
* infer\_onnx\_multi.py
* infer\_image.sh
* infer\_video.sh
* infer\_dataset\_image.sh
* calc\_map\_image.py
* calc\_map\_video.py
* calc\_map\_image\_dataset.py

*Note: to use NNCodec, a specific installation shall be done following* [x9].

Additional bash scripts located in scripts/objectdetection/tool\_for\_dataset are provided to download a set of images and associated annotations from the cocodataset repository, as well as a script to convert annotations to conform to calc\_map scripts. A readme file scripts/objectdetection/tool\_for\_dataset /Readme.md details the following scripts usage:

* get\_cocodataset\_annotations.sh
* get\_cocodataset\_selection.sh

### B.3.3 Single branch APIs and scripts applied to SSD300 model

The scripts are:

* **convert\_ssd300\_to\_onnx.py**

This script converts the PyTorch ssd\_300 model to ONNX.

|  |
| --- |
| **Usage:**  python convert\_ssd300model.py <output\_path\_to\_directory> Output: <output\_path\_to\_directory>/ssd\_resnet.onnx  **Example**: From rt-ml-ai-evaluation-framework directory  python scripts/objectdetection/ssd300/convert ssd300model.py ./models |

* **split\_onnx.py**

This script splits an ONNX file at identified bottlenecks points.

|  |
| --- |
| **Usage:**  python split\_onnx.py <path\_to\_onnx file> <split\_point\_name> <split\_flag>  split\_flag :’before’ to split before the split\_point\_name , ‘after’ to split after the split\_point\_name |

|  |
| --- |
| **Example:**  python split\_onnx.py ./models/ssd\_resnet.onnx /feature\_extractor/feature\_extractor/feature\_extractor.4/feature\_extractor.4.0/relu\_2/Relu before  Output : First and second part of the split in “./models”  Special character “/” in split\_point\_name is replaced with a “\_”.  Output example: /models/ssd\_resnet\_Part\_I\_\_feature\_extractor\_feature\_extractor\_feature\_extractor.4\_feature\_extractor.4.0\_relu\_2\_Relu.onnx  ./models/ssd\_resnet\_Part\_II\_\_feature\_extractor\_feature\_extractor\_feature\_extractor.4\_feature\_extractor.4.0\_relu\_2\_Relu.onnx |

* **infer\_onnx.py**

This script is used to run the inference of ssd300 model on an image or on a video. It infers the first part and the second part of the model sequentially in GPU or in CPU. The predictions are saved with the format [label top\_left\_x top\_left\_y bottom\_right\_x bottom\_right\_y confidence\_score], compatible with the scripts visualize.py and calc\_map.py. Intermediate data are saved in numpy binary format .npz. The visual prediction results, the image with the boxes, are saved with the .png format. For video, only the first visual prediction is saved.

|  |
| --- |
| **Usage**:  python infer\_onnx.py [-h] [-c PATH\_TO\_CONFIG] [-s INPUT\_SOURCE] [-loop LOOP] [-partI PARTI] [-partII PARTII] [-anchor ANCHOR] [-results\_filename RESULTS\_FILENAME] -results\_dir RESULTS\_DIR [-no\_CPU\_anchor] [-no\_GPU\_anchor] [-ref\_split REF\_SPLIT] [-no\_split]    **Help:**  infer\_onnx is a script that run the inference of a ssd resnet model, full model or split.  Options:  -h, --help show this help message and exit  -c PATH\_TO\_CONFIG, --path\_to\_config PATH\_TO\_CONFIG Path to config file  -s INPUT\_SOURCE, --input\_source INPUT\_SOURCE Path to input source  -loop LOOP loop inference  -partI PARTI Path to model part I  -partII PARTII Path to model part II  -anchor ANCHOR Path to model anchor  -results\_filename RESULTS\_FILENAME Path to results file -results\_dir RESULTS\_DIR Path to results directory hosting predictions  -no\_CPU\_anchor no inference with CPU on model anchor  -no\_GPU\_anchor no inference with GPU on model anchor  -ref\_split REF\_SPLIT reference split label  -no\_split no split (just anchor for instance) |

### B.3.4 Multi branch split APIs and scripts (ssd\_resnet and retinanet) with compression

* **split\_onnx\_multi.py**

This script splits an ONNX file at any node. Split point may be referenced by the node index/rank or by the node name.

|  |  |
| --- | --- |
| **Usage**  **Help**  split\_onnx\_multi is a script that split a ONNX model at a rank or at a node name | |
| optional arguments: |  |
| -h, --help | show this help message and exit |
| -a ANCHOR, --anchor ANCHOR | Path to model anchor |
| -r RANK, --rank RANK | Rank of the node where to split the model;  e.g., with '-r 7' Model I will contain nodes [0-6] and model II will contain nodes [7-48] |
| -n NAME, --name NAME | Name of the node where to split the model |
| -f FLAG, --flag FLAG | Split flag indicating if the split occurs 'before' or 'after' the given node (default is 'before') |

Script outputs are the two model subsets part I and part II, suffixed with “\_part\_I\_node\_#rank” (resp. \_part\_II\_node\_#rank), located in the same directory as the anchor.

|  |
| --- |
| **Usage:**  python split\_onnx\_multi.py -a ./models/ssd\_resnet.onnx -r 7  **Output:**  Model will be split at rank 7, node name /feature\_extractor/feature\_extractor/feature\_extractor.4/feature\_extractor.4.0/conv3/Conv (before)  Model Part I: Extraction at level 0 to 7 (excluded):  input=['input'] output=['/feature\_extractor/feature\_extractor/feature\_extractor.4/feature\_extractor.4.0/relu\_1/Relu\_output\_0', '/feature\_extractor/feature\_extractor/feature\_extractor.3/MaxPool\_output\_0']  Model Part II: Extraction at level 7 (included) to 154 : input=['/feature\_extractor/feature\_extractor/feature\_extractor.4/feature\_extractor.4.0/relu\_1/Relu\_output\_0', '/feature\_extractor/feature\_extractor/feature\_extractor.3/MaxPool\_output\_0'] output=['output1', 'output2']  Split done.  part I: ./models/ssd\_resnet\_part\_I\_node\_7.onnx  part II: ./models /ssd\_resnet\_part\_II\_node\_7.onnx  Intermediate data to transfer: ['/feature\_extractor/feature\_extractor/feature\_extractor.4/feature\_extractor.4.0/relu\_1/Relu\_output\_0', '/feature\_extractor/feature\_extractor/feature\_extractor.3/MaxPool\_output\_0'] |

* **infer\_onnx\_multi.py**

This script infers the first ONNX model subset (part I) from a first ONNX runtime, encodes the output of the inference, decodes the encoded output, and passes the decoded output as an input for inference of the second ONNX subset (part II) with a second ONNX runtime. It emulates the two split inferences from a first and a second endpoint. It supports models “ssd\_resnet” and “retinanet” or any other models having same pre-processing and same post-processing as these models.

|  |  |
| --- | --- |
| **Usage**  **Help**  infer\_onnx is a script that run the inference of either a ssd resnet model or retinanet model, full model or split | |
| optional arguments: |  |
| -h, --help | show this help message and exit |
| -s INPUT\_SOURCE, --input\_source INPUT\_SOURCE | Path to input source (Image, video, directory of images) |
| -loop LOOP | loop inference (for image) |
| -partI PARTI | Path to model part I |
| -partII PARTII | Path to model part II |
| -anchor ANCHOR | Path to model anchor |
| -f FAMILY, --family FAMILY | Model family i.e., 'ssd\_resnet' or 'retinanet' |
| -results\_filename RESULTS\_FILENAME | Path to .csv results file (time measurement) |
| -results\_dir RESULTS\_DIR | Path to results directory hosting predictions |
| -no\_CPU\_anchor | no inference with CPU on model anchor |
| -no\_GPU\_anchor | no inference with GPU on model anchor |
| -ref\_split REF\_SPLIT | reference split label |
| -no\_split | no split (just anchor for instance) |
| -no\_check\_model | do not check the model partI and partII |
| -save\_intermediate\_data | to save intermediate data |
| -nb\_frames NB\_FRAMES | Limit the inference to the nb\_frames first frames for a video |
| -encode\_algo ENCODE\_ALGO | algo for intermediate\_data encoding 0=no encoding; 1= convert to float16; 8xx for nnc with xx = abs(qp) (e.g., 838 for qp=-38) |
| -PU\_partI PU\_PARTI | Processing Unit for part I (CPU or GPU) |
| -PU\_partII PU\_PARTII | Processing Unit for part II (CPU or GPU) |
| --labels\_coco LABELS\_COCO | Path to labels file coco\_labels.csv |

According to the parameters the script is doing:

* Verification of the ONNX model by using checker.check\_model() function. This verification can be skipped especially for the splitted models that may sometimes raise an unfounded error (i.e., an error which does not prevent a correct inference with the ONNX runtime)
* inference of the anchor using GPU (except if flag -no\_GPU\_anchor is activated)
* inference of part I and part II using processing units indicating by flags (-PU\_partI and -PU\_partII)(except if flag -no\_split is activated). Intermediate data are encoded and decoded using the algorithm indicated by the identifier -encode\_algo.
* inference of the anchor using CPU (except if flag -no\_CPU\_anchor is activated)

Script outputs are:

* A .csv file containing information on time measurement, intermediate data size, predictions (described below)
* A folder containing the image (or frame) with the predictions on overlay (bounding boxes with label and prediction score)
* A folder containing the .txt predictions file compatible with format expected by calc\_map scripts

Results .csv file have following columns:

|  |  |  |  |
| --- | --- | --- | --- |
| Column name | Type | Unit | Description |
| Source | String | - | path to the input data source |
| Nodes | String | - | node reference (e.g., split\_node\_0010) |
| Inference\_loop | Int | - | number of inference run on the source |
| UE\_inference\_time\_CPU (avg; std) | Float | ms | inference time of the part I (average and standard deviation on all measures) when the processing unit of part I is CPU |
| UE\_inference\_time\_CPU\_steady\_state (avg;std) | Float | ms | inference time of the part I (average and standard deviation on all measures except the first one) when the processing unit of part I is CPU |
| UE\_inference\_time\_GPU (avg;std) | Float | ms | inference time of the part I (average and standard deviation on all measures) when the processing unit of part I is GPU |
| UE\_inference\_time\_GPU\_steady\_state (avg; std): | Float | ms | inference time of the part I (average and standard deviation on all measures except the first one) when the processing unit of part I is GPU |
| Server\_inference\_time\_CPU(avg; std): | Float | ms | inference time of the part II (average and standard deviation on all measures except the first one) when the processing unit of part II is CPU |
| Server\_inference\_time\_CPU\_steady\_state(avg; std): | Float | ms | inference time of the part II (average and standard deviation on all measures except the first one) when the processing unit of part I is CPU |
| Server\_inference\_time\_GPU(avg; std); | Float | ms | inference time of the part II (average and standard deviation on all measures except the first one) when the processing unit of part II is GPU |
| Server\_inference\_time\_GPU\_steady\_state (avg; std); | Float | ms | inference time of the part II (average and standard deviation on all measures except the first one) when the processing unit of part I is GPU |
| Total\_inference\_time\_steady\_state (avg; std); | Float | ms | Sum of “UE inference time steady state” + “Server inference time steady state “ (average and standard deviation) |
| Encoding\_time (min,avg,max); | Float | ms | Encoding time of the intermediate data |
| Decoding\_time (min,avg,max); | Float | ms | Decoding time of the intermediate data |
| nb\_inference; | Int | - | number of inference run on a video source (nb frames) |
| intermediate\_data\_size (min,avg,max); | Int | bytes | Intermediate data size (uncompressed) |
| encoded\_intermediate\_data\_size (min,avg,max); | Int | bytes | Intermediate data size (encoded/compressed) |
| encoding\_algo; | Int | - | Encoding Algorithm identifier |
| predictions\_size (min,avg,max); | Int | Bytes | Size of the post-processed predictions |
| predictions | String | - | Post-processed predictions  (label, boxes coordinates (top\_x, top\_y, bottom\_x, bottom\_y), confidence score)  (e.g., bear 13 28 573 620 0.49) |

* **calc\_map\_image.py**

This script calculates the mAP score for an image. This script is based on the existing calc\_map.py script, and is a simple adaptation to be able to compute the mAP score on a single image. It uses the same calculate\_map() function as calc\_map.py. It can compute the mAP score for several splits.

|  |  |
| --- | --- |
| **Usage**  **Help**  Calculate the mAP for the object detection prediction | |
| optional arguments: |  |
| -h, --help | show this help message and exit |
| -i IMAGE\_PATH,  --image\_path IMAGE\_PATH | Path to the image |
| -p PREDICTION\_PATH,  --prediction\_path PREDICTION\_PATH | Path to the prediction file or directory containing prediction files (for several splits) |
| -g GROUNDTRUTH\_PATH,  --groundtruth\_path GROUNDTRUTH\_PATH | Path to the ground-truth annotation file |
| -r RESULTS\_FILENAME,  --results\_filename RESULTS\_FILENAME | Path to .csv results file |
| --threshold THRESHOLD | The threshold for the prediction confidence to consider the prediction. |
| *--no\_plot* | *do not display the plot.* |

* **calc\_map\_video.py**

This script calculates the mAP score for a video. This script is based on the existing calc\_map.py script and is a simple adaptation to be able to compute the mAP score on a single video. It uses the same calculate\_map() function as calc\_map.py. It can compute the mAP score for several splits.

|  |  |
| --- | --- |
| **Usage**  **Help**  Calculate the mAP for the object detection prediction | |
| optional arguments: |  |
| -h, --help | show this help message and exit |
| -v VIDEO\_PATH, --video\_path VIDEO\_PATH | Path to the video |
| -p PREDICTION\_PATH,  --prediction\_path PREDICTION\_PATH | Path to the directory containing prediction files (one split) or Path to the directory containing directories of each split (several splits) |
| -g GROUNDTRUTH\_PATH,  --groundtruth\_path GROUNDTRUTH\_PATH | Path to the directory containing groundtruth annotation file |
| -r RESULTS\_FILENAME,  --results\_filename RESULTS\_FILENAME | Path to .csv results file |
| --labels\_imagenet LABELS\_IMAGENET | Path to labels file imagenet\_coco.csv |
| --labels\_coco LABELS\_COCO | Path to labels file coco\_labels.csv |
| -o IMAGE\_PATH, --image\_path IMAGE\_PATH | Path to the output file containg the mAP outplot plot |
| --threshold THRESHOLD | The threshold for the prediction confidence to consider the prediction. |

* **calc\_map\_image\_dataset.py**

This script calculates the mAP score for a set of images. This script is based on the existing calc\_map.py script and is a simple adaptation to be able to compute the mAP score on a set of images. It uses the same calculate\_map() function than calc\_map.py. It can compute the mAP score for several splits.

|  |  |
| --- | --- |
| **Usage**  **Help**  Calculate the mAP for the object detection prediction | |
| optional arguments: |  |
| -h, --help | show this help message and exit |
| -d IMAGE\_DATASET\_PATH,  --image\_dataset\_path IMAGE\_DATASET\_PATH | Path to the directory containing the images |
| -p PREDICTION\_PATH,  --prediction\_path PREDICTION\_PATH | Path to the directory containing prediction files (one split) or Path to the directory containing directories of each split (several splits) |
| --multisplit | indicates if prediction path contains several directories for multi split |
| -g GROUNDTRUTH\_PATH,  --groundtruth\_path GROUNDTRUTH\_PATH | Path to the directory containing groundtruth annotation file |
| -r RESULTS\_FILENAME,  --results\_filename RESULTS\_FILENAME | Path to .csv results file |
| --labels\_imagenet LABELS\_IMAGENET | Path to labels file imagenet\_coco.csv |
| --labels\_coco LABELS\_COCO | Path to labels file coco\_labels.csv |
| --threshold THRESHOLD | The threshold for the prediction confidence to consider the prediction. |

### B.3.5 FPN/RPN Retinanet scripts

The scripts are:

* **convert\_model.py**

a script to convert a pre-trained model into an ONNX model

* **inferonnx.py**:

this script is used to run an object detection inference model and produce predication results in the following format [label top\_left\_x top\_left\_y bottom\_right\_x bottom\_right\_y confidence\_score]. The model is used to produce results for the anchors, where the full model is run locally on the device or completely in the network.

|  |
| --- |
| **usage**:  inferonnx.py [-h] [--mask] dataset\_name model\_location  inferonnx.py: error: the following arguments are required: dataset\_name, model\_location |

* **split\_retinanet.py**:

this script is used to split the *retinanet* represented in the ONNX format. It takes the model at models/retinanet.onnx and splits at the four feature pyramid network (FPN) feature maps, as shown by the 4 nodes with red arrows pointed to in Figure 2.3-1, together with four other auxiliary operations (two of which are pointed to by the blue arrows in Figure 2.3-1 and there are two similar ones on the right side of the graph but not shown) that provide the input image shape information for later stages of the network. Note that the split needs 8 split points, rather than a single split point, due to branching and joining present in the structure of *retinanet*.

The splitting results in two partial models, called retinanet\_part1.onnx and retinanet\_part2.onnx, also in ONNX format. The input to part 1 is the input image. The feature maps in the output of part 1 is part of the input to part 2. The correct operation of part 2 needs additional input which is the shape of the input image. However, it makes no sense to feed the input image (together with the feature maps) as input to part 2. To resolve this problem, a dummy image of the same shape as the input image is used to generate the shape needed by part2. As a result, there is an overlap between part 1 and part 2. The overlap is chosen in such a way that only the portion of the graph directly contributing to generating the shape of the dummy image is included to minimize the additional processing. This is corroborated by the sizes of the models:

* + retinanet.onnx: 149.433MB
  + retinanet\_part1.onnx: 120.731MB
  + retinanet\_part2.onnx: 28.840MB

from which we see that the sum of the two partial models is only 0.14MB bigger than the size of the whole model, indicating that the overlap is negligible and so is the additional processing for generating the shape of the dummy image.

The two parts are fed into infer\_split.py for split inference.

* **infer\_split.py**:

this script is used to run split inference. It is passed the two parts of the model. It runs the first part of the model and saves the results in numpy binary format NPZ. Then it proceeds to run inference using the second part of the model, which loads the NPZ files as input and produces the object detection results. A flag SAVE\_FEATURES\_IN\_FILEs controls whether to write the FPN feature maps to the NPZ files, and it can be set to 0 to save storage, and in that case the feature maps out of the execution of part 1 are directly fed to part 2. This script also compares the performance between split inference and non-split inference in terms of normalized MSE.

|  |
| --- |
| **usage**:  splitinfer.py [-h] [--mask] dataset\_name model\_part1\_location model\_part2\_location  Run split inference using ONNX models  positional arguments:  dataset\_name Dataset name  model\_location Path to the unsplit ONNX Model  model\_part1\_location Path to 1st part of the ONNX Model  model\_part2\_location Path to 2nd part of the ONNX Model  optional arguments:  -h, --help show this help message and exit  --mask Indicates if output of model is a Mask and needs to be converted |

* **calc\_map.py**:

This script is used to calculate the mean Average Precision (mAP) score for the predictions. It compares the predicted labels and their bounding boxes to the ground truth annotations that are provided by the dataset.

|  |
| --- |
| **usage:**  calc\_map.py [-h] [--ds DATASET\_NAME] [--threshold THRESHOLD] video\_name  Calculate the mAP for the object detection prediction.  positional arguments:  video\_name The name of the video sequence, e.g. Kimono.  optional arguments:  -h, --help show this help message and exit  --ds DATASET\_NAME Name of the dataset. Defaults to SFU-HW-Objects.  --threshold THRESHOLD  The threshold for the prediction confidence to consider the prediction. |

* **visualize.py**:

The visualize script takes the ground truth annotations or the predictions and renders them on top of the video. This script is useful to inspect the prediction results.

|  |
| --- |
| **usage:**  visualize.py [-h] [--sleep\_time SLEEP\_TIME] video\_fn annotation\_path  Visualize Object Detection.  positional arguments:  video\_fn Path to the video file  annotation\_path Path to the folder with annotations/predictions  optional arguments:  -h, --help show this help message and exit  --sleep\_time SLEEP\_TIME  Specifies the interval between the display of 2 consecutive frames |

\* \* \* End third changes \* \* \* \*