SyNAP Heterogeneous Inference

PN: 511-000xxx-01 Rev 1.0.0

Contents

List of Figures

List of Tables

1. Introduction

Starting from version 3.0 SyNAP toolkit and runtime provide extensive support for heterogeneous inference. This means that:

- a model can be compiled to take advantage of all the computational resources available on the destination hardware (NPU, CPU, GPU)
- different parts of the model can be executed on different hardware units, also in parallel whenever possible
- the same workflow can be used to compile a model for different hardware targets
- the same API is available at runtime to run the model independently of the hardware unit(s) that will actually execute it

2. Use Case

In order to show how SyNAP heterogeneous inference can be useful in practice, we show here the use-case of converting a standard object-detection model. Even if we present a specific use-case, the same techniques and ideas are valid in general.

We would like to perform object-detection on images on a VS680. Instead of developing our own model, we decide to use a standard model from Google model zoo. In particular, we choose the *ssd_mobilenet_v1_coco* model, which is a Single Shot MultiBox Detector (SSD).

A pretrained, quantized tflite version of this model is available at the following link:

https://www.kaggle.com/models/tensorflow/ssd-mobilenet-v1/frameworks/tfLite/variations/default/versions/1

Note: For detailed information and description on the commands and options used in this document please refer to the SyNAP user manual: *SyNAP.pdf*.

3. Analysis

This model contains a MobileNetV1 backbone and a SSD head. The backbone is a convolutional neural network which performs feature extraction on the input image. The SSD head takes the features extracted by the backbone and performs the actual object detection. The detected boxes and the corresponding per-class scores are sent to a post-processing stage which performs non-maximum suppression (NMS) and returns the final detections (`TFLite_Detection_PostProcess` layer).

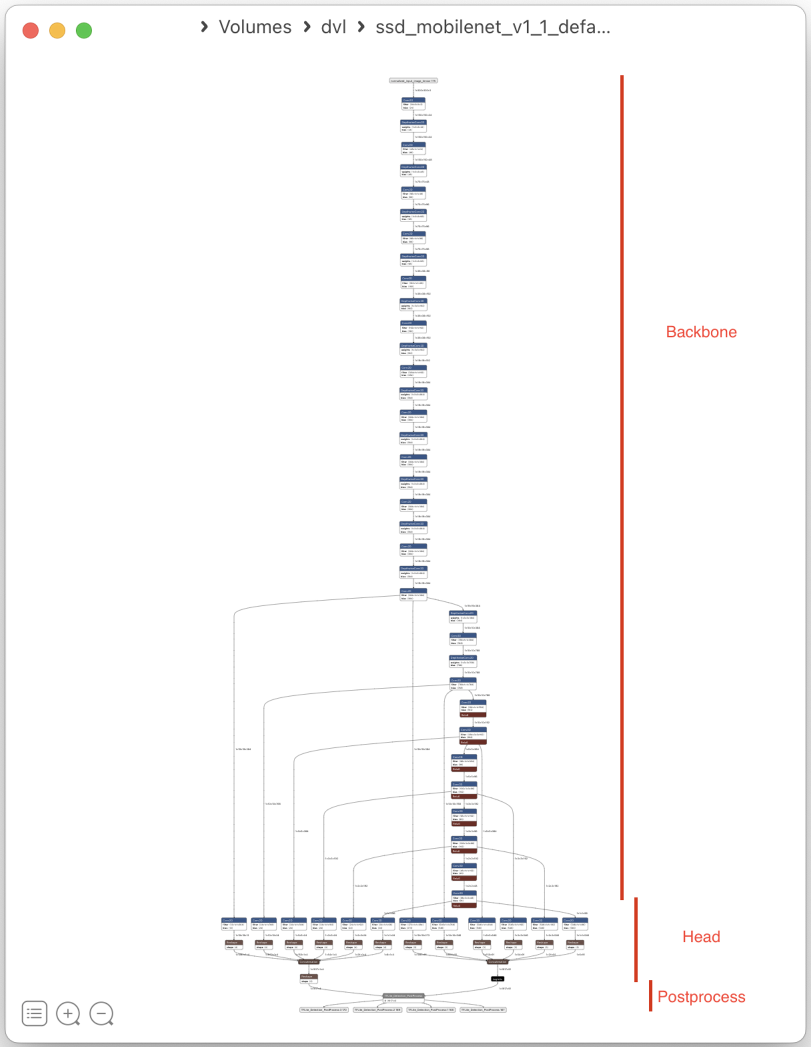


Figure 1. MobileNet SSD architecture

Since most of the model is composed by convolutional layers, the ideal hardware unit to execute it is the NPU. However, the TFLite_Detection_PostProcess layer is mostly sequential and it is not supported by the NPU, so compiling the entire model for execution on the NPU is not possible.

If we try to compile the entire model for execution on the NPU, the toolkit will detect this and generate a warning message:

```
$ synap convert --model ssd_mobilenet_v1_1_default_1.tflite --target VS680 --out-dir compiled
Warning: skipping unimplemented custom layer: TFLite_Detection_PostProcess
```

The toolkit has converted the model anyway without including the final unsupported layer in the compiled model. This allows us to experiment with the execution of the partial model. If the unsupported layer were in the middle of the network instead of being at the end, the conversion would have failed.

In any case in order to be able to completely execute this model, we have to split it in two parts, and execute each of them on a different hardware unit:

- the backbone and SSD head can be executed on the NPU
- the `TFLite_Detection_PostProcess` layer will have to be executed on the CPU

SyNAP provides two ways to achieve this:

1. convert only the backbone and SSD head of the model for execution on the NPU by *cutting away* the `TFLite_Detection_PostProcess` layer, and then implement the postprocessing in software
2. convert the entire model to an heterogeneous network, where the backbone and SSD head are executed on the NPU and the `TFLite_Detection_PostProcess` layer is executed on the CPU

The first method has the advantage that the postprocessing part can be implemented in any way we want, so we can experiment with different algorithms and optimizations. However, it requires to write some code for which it is necessary to understand in detail how the postprocessing layer works. SyNAP runtime library already provides support for the most common postprocessing layers, including `TFLite_Detection_PostProcess`.

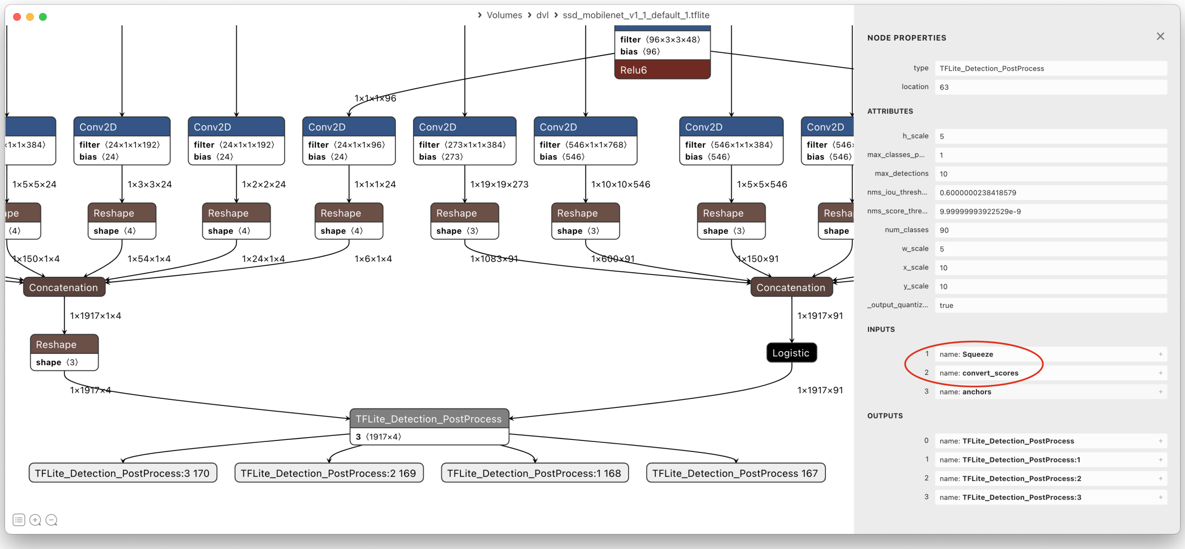
The second method is easier to use (no extra code to be written), and is actually the only possibility in cases where the unsupported layers are not at the end but in the middle of the network.

Let’s see how to use both methods.

4. Model Cutting

In order to perform the postprocessing explicity in SW, we need to cut away the `TFLite_Detection_PostProcess` layer. This doesn't require any modification to the *tflite* model itself, the layer can be removed easily when the model is converted, by providing a conversion metafile that specifies where to cut the model, that is which tensors should be the actual outputs of the converted model.

Let's see in detail the final part of the model.



NODE PROPERTIES

type	TFLite_Detection_PostProcess
location	63

ATTRIBUTES

h_scale	5
max_classes_p...	1
max_detections	10
nms_iou_thresh...	0.6000000238418579
nms_score_thre...	9.99999993922529e-9
num_classes	90
w_scale	5
x_scale	10
y_scale	10
_output_quantiz...	true

INPUTS

1	name: **Squeeze**
2	name: **convert_scores**
3	name: **anchors**

OUTPUTS

0	name: **TFLite_Detection_PostProcess**
1	name: **TFLite_Detection_PostProcess:1**
2	name: **TFLite_Detection_PostProcess:2**
3	name: **TFLite_Detection_PostProcess:3**

Figure 2. MobileNet SSD final part

We want the outputs of the converted model to be the input tensors of the `TFLite_Detection_PostProcess` layer. We can see that these are the tensors named `Squeeze` and `convert_scores`. There is a third input tensor named `anchors` but it is a constant tensor so we don't have to specify it as a model output.

To do this we create a conversion metafile `meta_cut.yaml` that specifies these tensors as the actual outputs of the converted model. Their content will then be fed at runtime to the SyNAP library to perform the postprocessing. Since there are several possible postprocessing algorithms, the metafile must also specify the format and attributes of these tensors so the SyNAP library can apply the right processing to them.

```yaml
outputs:
  - name: Squeeze
    format: tflite_detection_boxes anchors=${ANCHORS}
  - name: convert_scores
```

A few notes:

- `tflite_detection_boxes` indicate the format of the corresponding tensor
- the `${ANCHORS}` variable is replaced by the anchors extracted from the corresponding tensor in the tflite model at conversion time. This information is needed by the runtime library to convert the network output to the coordinates of the bounding boxes
- we don't explicitly specify any delegate, so the model will be compiled for the default delegate; on VS680 this is the NPU.

Important: For detailed information on the syntax and options of this metafile, please refer to the *Conversion Metafile* section in the `SyNAP.pdf` user manual.

Model conversion is performed as before, but with an additional argument to specify the conversion metafile. The warning message is now gone, since the unsupported layer has been excluded explicitly:

```
$ synap convert --model ssd_mobilenet_v1_1_default_1.tflite --meta meta_cut.yaml --target
↪VS680 --out-dir compiled
```

We can now push the compiled model to the target and run it with the `synap_cli_od` application:

Important: On Android the sample models can be found in `/vendor/firmware/models/` while on Yocto Linux they are in `/usr/share/synap/models/`. In this document we will refer to this directory as `$MODELS`.

```
$ adb shell mkdir /data/local/tmp/test
$ adb shell cp $MODELS/object_detection/coco/info.json /data/local/tmp/test
$ adb push compiled/model.synap /data/local/tmp/test
$ adb shell
dolphin:/ $ TEST_IMG=$MODELS/object_detection/coco/sample/sample001_640x480.jpg
dolphin:/ $ cd /data/local/tmp/test
dolphin:/data/local/tmp/test $ synap_cli_od $TEST_IMG

Loading network: model.synap

Input image: /vendor/firmware/models/object_detection/coco/sample/sample001_640x480.jpg (w =
↪640, h = 480, c = 3)
Detection time: 24.03 ms (pre:17802, inf:5846, post:374)
#   Score  Class   Position        Size  Description     Landmarks
0    0.76      1   184,  38    307, 433  bicycle
1    0.71      2   157,  94     72,  44  car
2    0.60      2   479,  41    160, 133  car
3    0.57      2   388, 107     72,  36  car
4    0.57      2    97, 101     18,  18  car
5    0.55     27   308,  51    102,  61  tie
6    0.51      2   421,  97     52,  42  car
```

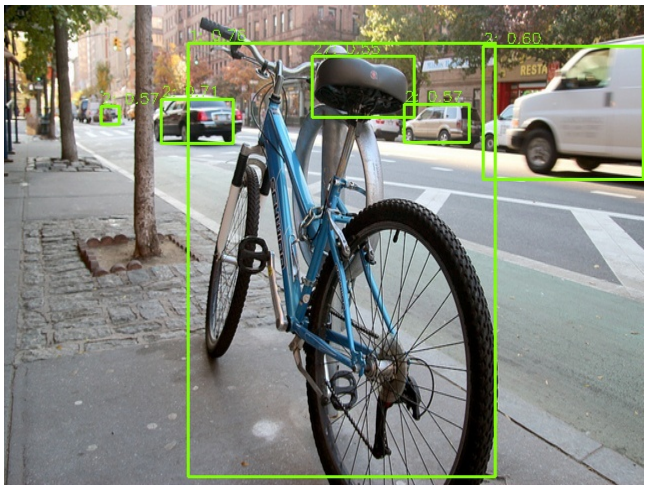


Figure 3. Object detection result

5. Heterogeneous Inference

Heterogeneous inference allows to execute different parts of the model on different hardware units. In this case we want to execute the entire model on the NPU, except the `TFLite_Detection_PostProcess` layer which we want to execute on the CPU.

As before we do this with a conversion metafile `meta_hi.yaml`.

```
delegate:
    '*': npu
    '63': cpu

outputs:
  - format: tflite_detection w_scale=640 h_scale=480
```

A few notes:

- we specify the `npu` delegate for all layers except layer 63 (`TFLite_Detection_PostProcess`) for which we specify the `cpu` delegate
- we want to convert the entire model so no need to indicate the output tensor names
- the format of the output tensor is not the same as before, since this is now the output of the TFLite_Detection_PostProcess layer: `tflite_detection`
- `w_scale` and `h_scale` indicate the width and height of the network input. This is needed by the runtime library to rescale the coordinates of the generated bounding boxes to the actual size of the input image

Let's recompile the model with the new metafile:

```
$ synap convert --model ssd_mobilenet_v1_1_default_1.tflite --meta meta_hi.yaml --target
↪VS680 --out-dir compiled
```

We can now push the compiled model to the target and execute it as before:

```
dolphin:/data/local/tmp/test $ synap_cli_od $TEST_IMG
Loading network: model.synap
INFO: Initialized TensorFlow Lite runtime.
INFO: Created TensorFlow Lite XNNPACK delegate for CPU.

Input image: /vendor/firmware/models/object_detection/coco/sample/sample001_640x480.jpg (w =
↪640, h = 480, c = 3)
Detection time: 26.52 ms (pre:17484, inf:8963, post:66)
#   Score  Class   Position        Size  Description     Landmarks
0    0.76      1   184,  38    307, 433  bicycle
1    0.71      2   157,  94     72,  44  car
2    0.60      2   479,  41    160, 133  car
3    0.57      2    97, 101     18,  18  car
4    0.57      2   400,  99     65,  38  car
5    0.55     27   308,  51    102,  61  tie
```

As we can see the results are almost exactly the same as before. Some minor differences are possible due to the fact that the postprocessing code is not identical. Regarding the execution time, even if one single inference is not enough to get accurate measurements, we can observe that the preprocessing time hasn't changed, the inference time is slightly higher, and external postprocessing time is reduced. This is what we expected since now most of the postprocessing is done during model inference.

In this case since CPU execution can only start after the NPU has generated its results, SyNAP runtime has no choice but to serialize the execution. However in case of more complex models with multiple heterogeneous parts, the runtime will try to execute them in parallel whenever possible, thus taking advantage as much as possible of the available hardware resources.

6. Automatic Heterogeneous Inference

Starting from SyNAP 3.1 heterogeneous inference will be the default, with automatic selection of the most suited hardware unit for each layer.

In this specific case the conversion toolkit will detect that the `TFLite_Detection_PostProcess` node is not available on the NPU and automatically select the CPU delegate for it.

This means that the conversion metafile is no longer needed, except if the user wants to override the default behavior or specify some additional attributes. For example if we know that the model has been trained with *RGB* images, we can specify this information explicitely in the metafile as an attribute of the input tensor. This information will be used at runtime by SyNAP preprocessing library to perform any necessary image format conversion.

Here below the corresponding meta file:

```yaml
delegate: auto

inputs:
  - format: rgb
```

Contact Us

Visit our website at www.synaptics.com to locate the Synaptics office nearest you.