Model Deployment on KV260
iVS Lab User Guide

Hsien-Po,Meng 2023/10/26
Min-Hao,Huang 2023/10/26

Contents

0. Introduction

為了要部屬模型至 KV260 上，我們必須使用 Vitis AI 中所制定的流程完成模型量化(Quantization)、模型編譯(Compilation)，產生此流程所必需的特殊檔案，才能部署至 KV260 中並利用其 DPU 執行模型推論。
此任務所需要參考的文件主要是來自於官方所提供之 User Guide，以及我們在遇到 issues 時所參考的各種相關網站。
其中，官方所提供之 User Guide 如下所示:
User Guide 1414 (UG1414):
https://docs.xilinx.com/r/en-US/ug1414-vitis-ai
User Guide 1354 (UG1354):
https://docs.xilinx.com/r/en-US/ug1354-xilinx-ai-sdk/Introduction
Quick Start Guide for Zynq™ UltraScale+™:
https://xilinx.github.io/Vitis-AI/3.0/html/docs/quickstart/mpsoc.html#quickstart

For the following evaluation boards, refer to Vitis AI 3.0:

- AMD Zynq™ UltraScale+™ MPSoC ZCU102 Evaluation Board
- AMD Zynq™ UltraScale+™ MPSoC ZCU104 Evaluation Board
- AMD Versal™ VCK190 Evaluation Board
- AMD Kria™ KV260 Vision AI Starter Kit
- AMD Versal™ VCK5000 Development Card

!注意! 以上的 User Guide 若是要部屬模型至 KV260 上，必須參考 3.0 版本(目前最新版本為 3.5 版本)

1. 環境建置(host 端與 target 端)與操作範例

此段落之詳細步驟可以參考上述提及之 Quick Start Guide for Zynq™ UltraScale+™文件，其步驟描述詳細，這邊不再多贅述，以下解釋需要注意或是更改的部分:

➢ **Clone the Vitis AI Repository** 步驟中會需要 git clone 相關 repo 到 host 端，如圖所示:

Quickstart

Clone the Vitis AI Repository

```
[Host] $ git clone https://github.com/Xilinx/Vitis-AI
```

但如此會 clone 到最新版本(3.5 版本)之相關檔案，由於 KV260 必須使用 3.0 版本，因此必須把版本回溯至 3.0 版本，步驟如下:

A. 在 host 端仍先如同上圖之方式 git clone 最新版本之 repo。

B. Git clone 至 host 端後，於官方 Vitis AI repo 中找到其 history 頁面，複製 SHA 碼，如下圖所示:

C. 於 host terminal 端輸入以下指令: **git checkout c5d2bd4**，其版本便會回溯至 3.0。

➢ **嘗試 ResNet50 example 會遇到的問題:**

當嘗試以下指令時:

```
[Docker] $ cd examples/vai_runtime/resnet50_pt
[Docker] $ bash -x build.sh
```

可能會出現以下錯誤:

```
(vitis-ai-pytorch) vitis-ai-user@ivos02:/workspace/examples/vai_runtime/resnet50_pt$ bash build.sh
No LSB modules are available.
No LSB modules are available.
g++ (Ubuntu 10.3.0-1ubuntu1~20.04) 10.3.0
Copyright (C) 2020 Free Software Foundation, Inc.
This is free software; see the source for copying conditions.  There is NO
warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.

opencv4                 OpenCV - Open Source Computer Vision Library
```

因此必須把指令更改成:

[Docker] sh -x build.sh

便能成功解決，並繼續 Quick Start 文件中的步驟。

當操作到以下步驟時:

```
[Target] $ ./resnet50 /usr/share/vitis_ai_library/models/resnet50/resnet50.xmodel
```

可能會出現以下錯誤，表示無法打開 dpu.xclbin 檔案:

```
root@xilinx-kv260-starterkit-20222:~/Vitis-AI/examples/vai_runtime/resnet50# ./resnet50 /usr/share/vitis_ai_library/models/resnet50/resnet50.xmodel
WARNING: Logging before InitGoogleLogging() is written to STDERR
I0925 01:42:43.984153 216598 main.cc:292] create running for subgraph: subgraph_conv1
I0925 01:42:44.002641 216598 xrt_bin_stream.cpp:47] Please check your /etc/vart.conf
Its format should be :
    firmware: xx
Example:
    firmware: /run/media/mmcblk0p1/dpu.xclbin
F0925 01:42:44.002732 216598 xrt_bin_stream.cpp:51] [UNILOG][FATAL][VART_OPEN_DEVICE_FAIL][Cannot open device] open(/run/media/mmcblk0p1/dpu.xclbin) failed.
** Check failure stack trace: **
Aborted
```

解決方法是回到 etc repo 並修改 vart.conf:

[Docker] cd ~etc/

[Docker] vi vart.conf

將 firmware 所連結到的路徑修改成:

```
firmware: /lib/firmware/xilinx/kv260-benchmark-b4096/kv260-benchmark-b4096.xclbin
```

便能成功如同範例般顯示出結果與其圖片:

```
root@xilinx-kv260-starterkit-20222:~/Vitis-AI/examples/vai_runtime/resnet50# ./resnet50 /usr/share/vitis_ai_library/models/resnet50/resnet50.xmodel
WARNING: Logging before InitGoogleLogging() is written to STDERR
I0925 02:28:37.363512 219046 main.cc:292] create running for subgraph: subgraph_conv1

Image : 001.jpg
top[0] prob = 0.982662  name = brain coral
top[1] prob = 0.008502  name = coral reef
top[2] prob = 0.006621  name = jackfruit, jak, jack
top[3] prob = 0.000543  name = puffer, pufferfish, blowfish, globefish
top[4] prob = 0.000330  name = eel
```

2. 模型量化(Model Quantization)

大致而言，模型部署的流程如下：

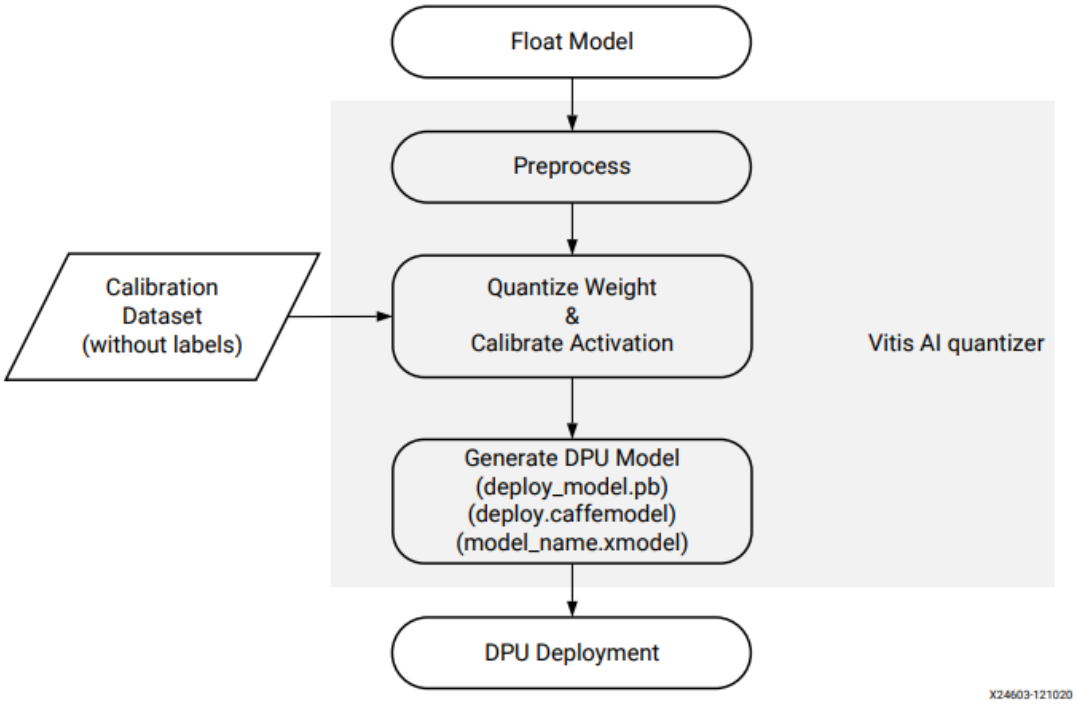
Model training → Model quantization → Model compilation → deployment on board

- **Quantization**

根據 Vitis AI user guide，在模型量化這個階段，能將 weight 與 activation 為 float32 表示的模型轉成以 int8 表示，並且最後會產出對應的.pt/.onnx/.xmodel 檔，供下一個階段 compilation 使用。

模型量化的流程如下圖：

Figure 19: **VAI Quantizer Workflow**



在模型量化之前，需要準備下列資料(以 PyTorch 為例)：

No.	Name	Description
1	model.pth	Pre-trained PyTorch model, generally pth file.
2	model.py	A Python script including float model definition.
3	calibration dataset	A subset of the training dataset containing 100 to 1000 images.

需要準備 model.pt，model.py(定義模型的 python code)以及 training 時用到的 dataset，用來做 calibration 使用。

準備好相關資料後，接下來就是執行 quantization 之步驟。流程與指令可以參考官方文件，這邊不再贅述。

- **Quantization 範例：以 yolov4-csp P85 為例**

參考文件：

Yolov4 環境設定 https://github.com/WongKinYiu/PyTorch_YOLOv4

Yolov4_csp Copyleft repo: https://github.com/Xilinx/Vitis-AI-Copyleft-Model-Zoo/tree/main/yolov4_csp

首先，參考 yolov4 的環境設置，git clone 下整個資料夾到本機端。

準備好 quantization 所需要的檔案之後，下一步是修改 models.py。因為 Vitis AI quantizer 看不懂模型後處理的運算(原因可能是因為那些 operation 不是 torch

tensor 的運算)，因此我們需要引入 Vitis AI 提供的 module partial quantization 功能。他可以將要輸入 quantizer 進行量化的部分框起來，其餘的 code 則是不進入 quantizer 做量化，簡單的範例說明如下所示:

Module Partial Quantization

You can use module partial quantization if not all the sub-modules in a model need to be quantized. Besides using general vai_q_pytorch APIs, the `QuantStub/DeQuantStub` operator pair can be used to realize it. The following example demonstrates how to quantize `subm0` and `subm2`, but not quantize `subm1`.

```
from pytorch_nndct.nn import QuantStub, DeQuantStub

class WholeModule(torch.nn.module):
    def __init__(self,...):
        self.subm0 = ...
        self.subm1 = ...
        self.subm2 = ...

        # define QuantStub/DeQuantStub submodules
        self.quant = QuantStub()
        self.dequant = DeQuantStub()

    def forward(self, input):
        input = self.quant(input) # begin of part to be quantized
        output0 = self.subm0(input)
        output0 = self.dequant(output0) # end of part to be quantized

        output1 = self.subm1(output0)

        output1 = self.quant(output1) # begin of part to be quantized
        output2 = self.subm2(output1)
        output2 = self.dequant(output2) # end of part to be quantized
```

簡易的 pseudo code 如下：

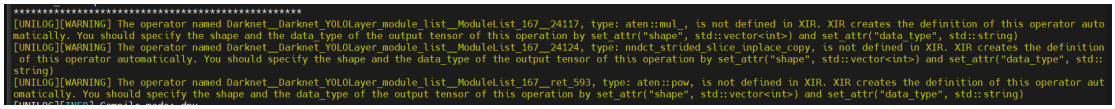
透過將後處理的程式移到 quantizer 之外，不讓這部分的運算進入 quantizer，就可以避免之後 compile 的錯誤。

```
class Darknet(nn.Module):
    def __init__(self, cfg, img_size=(416, 416), verbose=False):
        super(Darknet, self).__init__()
        from pytorch_nndct.nn import QuantStub, DeQuantStub
        self.quant = QuantStub()
        self.dequant = DeQuantStub()

    def forward(self, x, augment=False, verbose=False, quant=False, debug=False):
        x = self.quant(x)
        print("quantize start!")
        ...
        ...
        ...
        yolo_out = self.dequant(yolo_out)
        x = list(yolo_out)
        print("quantize end!")

        # post process
```

之前在 quantize 時遇到的 warning，原因為後處理的 code 也進到 quantizer，但是 quantizer 看不懂：



```python
def post_process(self, p):
    bs, _, _, ny, nx = p.shape
    # (bs, 3, 13, 13, 85) (bs, anchors, grid, grid, classes + xywh)
    if self.training:
        return p
    else:
        # from IPython import embed
        # embed()
        # print(p.shape,self.grid)
        io = p.sigmoid()
        io[..., :2] = io[..., :2] * 2.0 - 0.5 + self.grid
        io[..., 2:4] = (io[..., 2:4] * 2) ** 2 * self.anchor_wh
        io[..., :4] *= self.stride
        print("post process!")
        return (io.view(bs, -1, self.no), p)
```

在此沒有把後處理移到 quantizer 外的情況下，雖然 quantize 階段能夠順利完成，但進入 compile 階段時會出現下面的 warning 與 fatal error，因此必須使用上方所提及之 quant & dequant operation 處理。

```
[UNILOG][INFO] begin to compile...
[UNILOG][WARNING] xir::Op{name = Darknet__Darknet_Sequential_module_list__ModuleList_174__Conv2d_Conv2d__ret_643_sink_transpose_2, type = transpose} has been assigned to CPU.
[UNILOG][WARNING] xir::Op{name = Darknet__Darknet_YOLOLayer_module_list__ModuleList_175__ret_651, type = transpose} has been assigned to CPU.
[UNILOG][WARNING] xir::Op{name = Darknet__Darknet_YOLOLayer_module_list__ModuleList_175__ret_661_new, type = eltwise-fix} has been assigned to CPU: [DPUCZDX8G_ISA1_B4096 does not support eltwise SUB].
[UNILOG][WARNING] xir::Op{name = Darknet__Darknet_Sequential_module_list__ModuleList_170__Conv2d_Conv2d__ret_605_sink_transpose_1, type = transpose} has been assigned to CPU.
[UNILOG][WARNING] xir::Op{name = Darknet__Darknet_YOLOLayer_module_list__ModuleList_171__ret_613, type = transpose} has been assigned to CPU.
[UNILOG][WARNING] xir::Op{name = Darknet__Darknet_YOLOLayer_module_list__ModuleList_171__ret_623_new, type = eltwise-fix} has been assigned to CPU: [DPUCZDX8G_ISA1_B4096 does not support eltwise SUB].
[UNILOG][WARNING] xir::Op{name = Darknet__Darknet_Sequential_module_list__ModuleList_166__Conv2d_Conv2d__ret_567_sink_transpose_0, type = transpose} has been assigned to CPU.
[UNILOG][WARNING] xir::Op{name = Darknet__Darknet_YOLOLayer_module_list__ModuleList_167__ret_575, type = transpose} has been assigned to CPU.
[UNILOG][WARNING] xir::Op{name = Darknet__Darknet_YOLOLayer_module_list__ModuleList_167__ret_585_new, type = eltwise-fix} has been assigned to CPU: [DPUCZDX8G_ISA1_B4096 does not support eltwise SUB].
[UNILOG][WARNING] xir::Op{name = Darknet__24960, type = const-fix} has been assigned to CPU: [Has no fanout or at least one fanout is out of DPU subgraph.].
[UNILOG][WARNING] xir::Op{name = Darknet__24958, type = const-fix} has been assigned to CPU: [Has no fanout or at least one fanout is out of DPU subgraph.].
[UNILOG][WARNING] xir::Op{name = Darknet__24948, type = const-fix} has been assigned to CPU: [Has no fanout or at least one fanout is out of DPU subgraph.].
[UNILOG][WARNING] xir::Op{name = Darknet__Darknet_YOLOLayer_module_list__ModuleList_175__ret_649, type = reshape-fix} has been assigned to CPU: [xir::Op{name = Darknet__Darknet_YOLOLayer_module_list__ModuleList_175__ret_649, type = reshape-fix}'s input and output all from CPU.].
[UNILOG][WARNING] xir::Op{name = Darknet__Darknet_YOLOLayer_module_list__ModuleList_175__ret_673, type = strided_slice-fix} has been assigned to CPU: [it has one output is not from DPU.].
[UNILOG][WARNING] xir::Op{name = Darknet__Darknet_YOLOLayer_module_list__ModuleList_171__ret_611, type = reshape-fix} has been assigned to CPU: [xir::Op{name = Darknet__Darknet_YOLOLayer_module_list__ModuleList_171__ret_611, type = reshape-fix}'s input and output all from CPU.].
[UNILOG][WARNING] xir::Op{name = Darknet__Darknet_YOLOLayer_module_list__ModuleList_171__ret_635, type = strided_slice-fix} has been assigned to CPU: [it has one output is not from DPU.].
[UNILOG][WARNING] xir::Op{name = Darknet__Darknet_YOLOLayer_module_list__ModuleList_167__ret_573, type = reshape-fix} has been assigned to CPU: [xir::Op{name = Darknet__Darknet_YOLOLayer_module_list__ModuleList_167__ret_573, type = reshape-fix}'s input and output all from CPU.].
[UNILOG][WARNING] xir::Op{name = Darknet__Darknet_YOLOLayer_module_list__ModuleList_167__ret_597, type = strided_slice-fix} has been assigned to CPU: [it has one output is not from DPU.].
[UNILOG][FATAL][XCOM_ACGEN_UNSUPPORT_QUANTIZATION][unsupport quantization bit shift while assembly code generation.] Darknet_Darknet_YOLOLayer_module_list_ModuleList_175_ret_655_fix_FixShifterprecheck fail
the delta of fix_point between input and output is over bit width
fix_point_min of input is7, output fix point is -2
*** Check failure stack trace: ***
Aborted (core dumped)
```

修改完 models.py 之後就可以執行模型的 quantization:

[Docker] python3 test.py --cfg GM_E_shortcut_prune_ratio_0.9_P85_yolov4-CSP_Emirror.cfg --data data/coco_6cls.yaml --weight ./models/best.pt --quant_mode calib --output_path /workspace/yolov4_AMD/quantize_result --nndct_quant --img-size 224

其中 quant_mode 設定為 calib，這是為了做 calibration 與 model evaluation 使用。

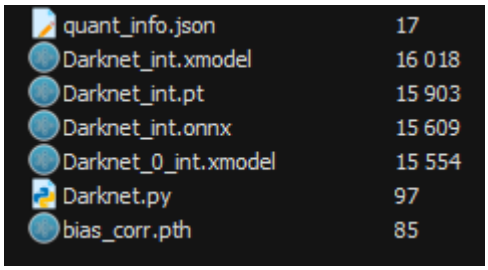
[Docker] python3 test.py --cfg GM_E_shortcut_prune_ratio_0.9_P85_yolov4-CSP_Emirror.cfg --data data/coco_6cls.yaml --weight ./models/best.pt --dump_xmodel --output_path /workspace/yolov4_AMD/quantize_result/ --nndct_quant --quant_mode test --batch-size 1 --deploy --img-size 224

再來就將 quant_mode 設為 test，產出 compile 需要的.xmodel 檔案。

上述指令所使用到的 test.py 來自以下參考網址，若是不同系列的 model 就必須得根據其 model 特性撰寫相關的 Python code。

https://github.com/Xilinx/Vitis-AI-Copyleft-Model-Zoo/tree/main/yolov4_csp/code

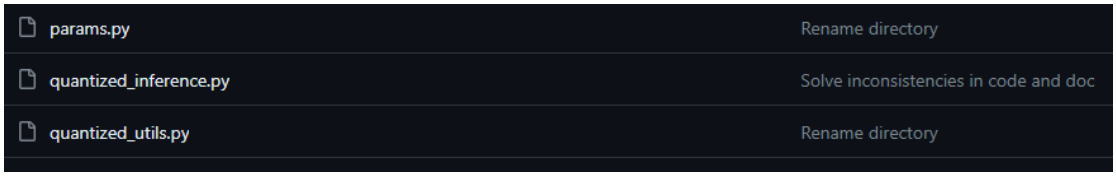
下圖是 quantization 後的產出檔案：



- **執行 quantized model 的 inference**

參考文件：yolov3 quantization compilation with PyTorch
https://github.com/LogicTronix/Vitis-AI-Reference-Tutorials/tree/main/Quantizing-Compiling-YOLOv3-Pytorch-with-DPU-Inference

inference code 參考了上面的 github，將 yolov3 inference 改成我們現在要用的 yolov4 inference。基本上整個 code 是大同小異。要注意的地方是，跑 inference 需要吃的檔案為 torch script (.pt) 而非.xmodel 檔。



將 params.py 修改成我們當前的 model 的設定(如下圖所舉例)，就可以跑 model inference 了。

```
TRAINING_PARAMS = {
    "model_params": {
        "backbone_name": "darknet_53",
        "backbone_pretrained": "",
    },
    "yolo": {
        "anchors": [
            [[3, 3], [5, 6], [10, 8]],
            [[7, 15], [18, 14], [16, 32]],
            [[38, 29], [42, 67], [100, 108]],
        ],
        "classes": 6,
    },
    "batch_size": 16,
    "confidence_threshold": 0.2,
    "images_path": "./images/",
    "classes_names_path": "./data/coco_6cls.names",
    "img_h": 224,
    "img_w": 224,
    "parallels": [0],
    "pretrain_snapshot": "./weights/best_singlescale.pt",
}
```

Inference 後的效果如下圖：

3. 模型編譯(Model Compilation)

根據 Vitis AI model deployment 的流程，在此階段可將量化完成並且成功轉檔的檔案(.xmodel 檔案)輸入至 Vitis AI Compiler 中，完成模型的編譯。其中編譯指令如同 Quick Start 中所示:

[Docker] $ vai_c_xir -x quantize_result/ResNet_int.xmodel
-a /opt/vitis_ai/compiler/arch/DPUCZDX8G/KV260/arch.json
-o resnet18_pt -n resnet18_pt

將.xmodel 檔案路徑輸入至-x 後，-a 後則是輸入 DPU 型號與其架構(須與 target 端所使用之型號架構相同，此處為 KV260 範例)，-o 則是輸出資料夾名稱，-n 為輸出檔案名稱。此處以 ResNet 為範例，須根據實際所應用之 model 而修改。

此外，在模型編譯完成後，必須自行撰寫.prototxt 檔案，描述模型架構等相關資訊，在 Vitis AI 範例中，不同種模型會具有不同種 prototxt 的撰寫方式，這邊建議下載相同種類模型之 prototxt 文件範例根據實際應用修改，詳細撰寫方式與參數介紹可以參考官方文件:

https://docs.xilinx.com/r/en-US/ug1354-xilinx-ai-sdk/Using-the-Configuration-File

若是 yolov4 之範例介紹可以參考:

https://xilinx.eetrend.com/blog/2023/100571337.html

或是以下 github 相關 issues 提問，以獲取關鍵訊息:

https://github.com/Xilinx/Vitis-AI/issues/346

https://github.com/Xilinx/Vitis-AI/issues/564

https://github.com/Xilinx/Vitis-AI/issues/703

撰寫此 prototxt 檔案會在之後的模型部署階段使用到。

4. 模型部署(Model Deployment)於 Target 端

在模型部署至 target 的階段，需要把成功經過編譯的模型相關輸出檔案，也就是編譯完成之資料夾檔案與自行撰寫之.prototxt(放在相同資料夾內)一同傳送至 target 端進行模型部署，如同 Quick Start 中所述，將所需資料傳送至以下路徑:

/usr/share/vitis_ai_library/models/

接下來便可以根據模型的種類去選取以下路徑之不同範例中的測試指令進行模型效能測試:

/Vitis-AI/examples/vai_library/samples/

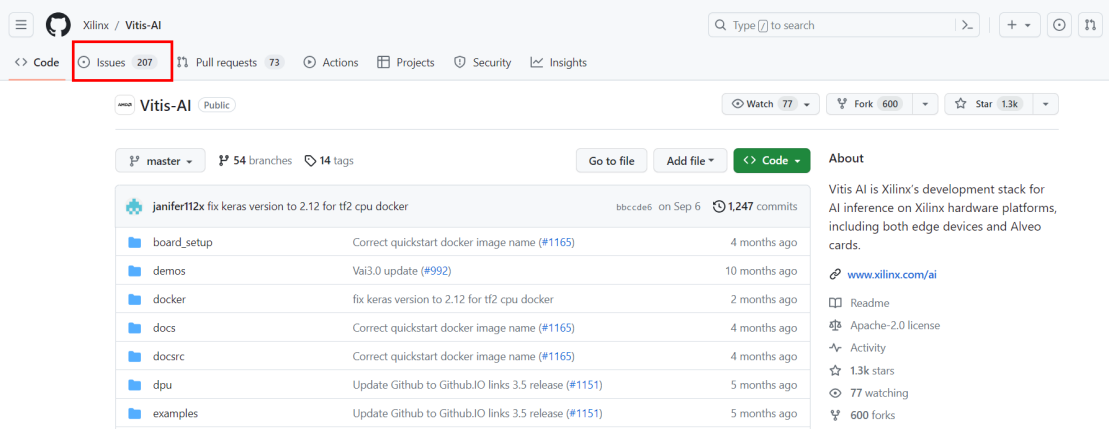
在「/usr/share/vitis_ai_library/models/」路徑中也有許多不同模型的相關檔案可以參考。

若使用範例中的 test_jpeg 指令測試模型效能，通常會自動產生具有結果之相關圖片，因在 target 端無法打開此結果圖片，所以若想要觀察此結果，必須將結果

圖片利用 scp 指令或是 rsync 指令再傳送回 host 端，才能將結果進行後續處理與觀察。

5. 尋求技術支援或解答

對於無法解決之技術問題，可以至 Vitis AI github 官方網站發表 issues 詢問，如下圖所示:



或是至 AMD 官方論壇中的 Vitis AI 專區進行詢問:

https://support.xilinx.com/s/topic/0TO2E000000YKY9WAO/vitis-ai-ai?language=en_US

根據經驗，AMD 官方論壇中能獲得解答的時間更短，且較常有人解答。