Convert & Deploy Yolov4-tiny on Kria

本篇環境

1. Ubuntu 18.04
2. Tensorflow-gpu 2.6 (for Yolov4-tiny training)
3. Vitis AI 1.4.1
4. kv260 petalinux 2021.1

先備知識：須了解大致 AI training 與 DPU 設定的相關知識

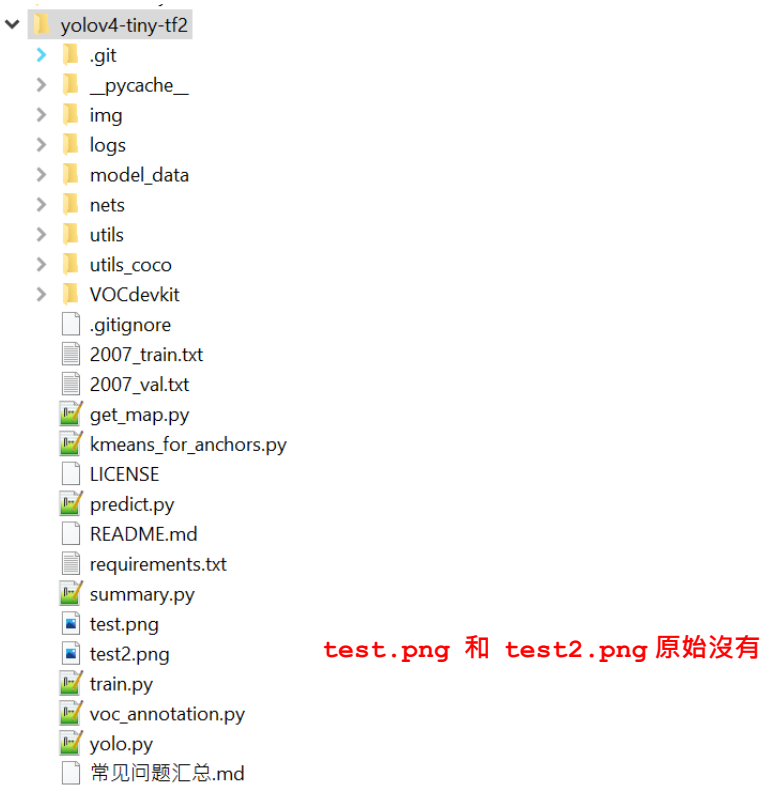
流程開始

◆ Step 1:

透過這篇的 github 來進行 Yolov4-tiny 的訓練

因為此時 Vitis AI 還未支援 tensorflow split 的 operation，因此需更改模型的架構

此包 github 的檔案結構



在 nets ---> CSPdarknet53_tiny.py 中將第 78 行：

```
x = Lambda(route_group,arguments={'groups':2, 'group_id':1})(x)
```

更改為

```
x = DarknetConv2D_BN_Leaky(int(num_filters/2), (1,1), weight_decay=weight_decay)(x)
```

此步驟是將透過 1x1 Conv 取代掉 tensorflow split

◆ Step 2：訓練前準備

1. 修改 voc_annotation.py 裡面的 annotation_mode=2，此舉會透過

   VOCdevkit/VOC2007/JPEGImages 生成訓練用的

   2007_train.txt 和 2007_val.txt

   修改 classes_path，指向自我訓練集的 label txt 檔案，一般在 model_data 中

2. 訓練結果預測設定需要更改

   yolo.py 中的 model_path 以及 classes_path

   model_path 指向訓練好的權重，一般在 logs 中

   classes_path 指向自己訓練及的 label txt 檔，一般在 model_data 中

3. 數據集的放置

   本篇使用 VOC 格式進行訓練

   訓練前將標籤文件放置到 VOCdevkit/VOC2007 文件夾下的 Annotation

   可透過 LabelImg 設定 label 路徑

   訓練前將圖片文件放置到 VOCdevkit/VOC2007 文件夾下的 JPEGImages

4. 數據集標籤

   請參考 LabelImg 的使用說明

5. 產生數據並執行訓練

   執行 voc_annotation.py 獲得訓練用的 2007_train.txt 和 2007_val.txt

   執行 train.py 進行訓練，訓練中或結束的結果會保存在 logs 中

   建議使用 pre-trained model weight 來提高與減少訓練的精度與時間

6. 其餘細節可參考 code 中的註釋或此包 github 的說明

◆ Step 3: 模型轉換

啟動 Vitis AI docker 1.41 環境，並啟動 vitis-ai-tensorflow2

可下載官方 Vitis AI Lab 中轉換 model 的 shell 來當作參考

1. Quantize(量化)

```
norris@ubuntu:~/Vitis-AI-2.0/vai_q_c_tf2_pt/lab$ ls -lh
total 100K
-rw-rw-r--  1 norris norris 3.7K Aug 28 20:22 1_tf2_quantize.sh
-rw-rw-r--  1 norris norris 3.4K Aug 28 20:22 2_tf2_eval_float_graph.sh
-rw-rw-r--  1 norris norris 3.5K Aug 28 20:22 3_tf2_eval_quantize_graph.sh
-rw-rw-r--  1 norris norris 3.4K Aug 28 20:22 4_tf2_compile_for_mpsoc.sh
-rw-rw-r--  1 norris norris 3.3K Aug 28 20:22 5_pytorch_quantize.sh
-rw-rw-r--  1 norris norris 3.4K Aug 28 20:22 6_pytorch_compile_for_mpsoc.sh
drwx------ 29 norris norris  36K Aug 28 20:22 images
drwx------  2 norris norris 4.0K Aug 28 20:22 pt_resnet50
-rw-rw-r--  1 norris norris  12K Aug 28 20:22 resnet50_quantize.py
drwx------  6 norris norris 4.0K Aug 28 20:22 tf2_resnet50_imagenet_224_224_7.76G_2.0
-rw-rw-r--  1 norris norris  17K Aug 28 20:22 val.txt
```

打開 1_tf2_quantize.sh 可以看到量化 code 是用 code/com 中的

train_eval_h5.py，可參考此檔案自行撰寫一 quantize 自己 model 的 python 檔

當中需要 import 原始訓練參考 github 中部分資料夾內文件，可以參考與此文件位置內

的範例 code

2. Compile

修改 4_tf2_compile_for_mpsoc.sh 中的設定

```
export TF2_NETWORK_PATH='tf2_resnet50_imagenet_224_224_7.76G_2.0'

vai_c_tensorflow2 -m ${TF2_NETWORK_PATH}/vai_q_output/quantized.h5 \
                  -a /opt/vitis_ai/compiler/arch/DPUCZDX8G/ZCU102/arch.json \
                  -o ${TF2_NETWORK_PATH}/vai_c_output \
                  -n resnet50_tf2 \
```

包含輸出路徑、開發版形式、arch.json 中的 DPU 形式以及輸出檔案名稱

接著執行 4_tf2_compile_for_mpsoc.sh，會發現以下錯誤

[UNILOG][FATAL][XIR_MULTI_DEFINED_OP][Multiple definition of OP!]

quant_max_pooling2d_1

此為官方 Vitis AI 的 Bug，等待下次更新後官方修正

解決方式可將上圖中紅色底線的檔案：

/tmp/yolov4_tf2_DPUCZDX8G_ISA1_B4096_org.xmodel

Copy 出來並進行 layer 名稱的變換

官方給出相關變換 layer 名稱的 code，可參考以下

```
import xir
g = xir.Graph.deserialize("quantize_model.xmodel")
ops = g.toposort()
for op in ops:
    if op.get_name() == "quant_max_pooling2d":
        replace_pool = g.create_op("quant_max_pooling2d_0", op.get_type(), op.get_attrs(), op.get_input_ops())
        [succ.replace_input_ops(op, replace_pool) for succ in op.get_fanout_ops()]
        g.remove_op(op)
g.serialize("renamed_quantize_model.xmodel")
```

此為針對 xmodel 進行 layer 變換的 code，因此執行此 code 之後，我們需要將環境變

更為 pytorch

```
conda deactivate
conda activate vitis-ai-pytorch
```

然後再進行 lab 中的 sh 範例

一樣先進行 shell 檔的修改

```
TARGET=MPSOC
NET_NAME=resnet50
DEPLOY_MODEL_PATH=vai_q_output

ARCH=/opt/vitis_ai/compiler/arch/DPUCZDX8G/ZCU102/arch.json

vai_c_xir -x pt_resnet50/vai_q_output/ResNet_int.xmodel \
          -o pt_resnet50/vai_c_output/${TARGET} \
          -n pt_resnet50 \
          -a ${ARCH}
```

包含輸出路徑、開發版形式、arch.json 中的 DPU 形式以及輸出檔案名稱

之後就可以將 compile 完成的 xmodel deploy 到開發版上了

*** notice:你必須確認你的 Vitsi AI 版本跟你開發版上的 meta-vitis-layer 是一致的，譬如說 petalinux 2021.1 中的 meta-vitis-layer 為 1.4，你的 Vitsi AI 就要為 1.4，否則會出現以下錯誤：***

```
terminate called after throwing an instance of 'std::bad_any_cast'
 what(): bad any_cast
Aborted
```

◆ Step 4: 部署訓練模型到開發版上

此篇使用 KV260，並採用 KV260 上的 smartcam app 作為 loading Yolov4-tiny model 的做法，因此可參考 smartcam 中讀取模型的流程以及需額外修改相關檔案

檔案包括 preprocess.json, drawdeault.json, label.json 等等

實際位置如下圖

```
xilinx-k26-starterkit-2021_1:~$ ls /opt/xilinx/share/ivas/smartcam/yolo4/
aiinference.json  drawresult.json   label.json        preprocess.json
```

```
xilinx-k26-starterkit-2021_1:~$ ls /opt/xilinx/share/vitis_ai_library/models/kv260-smartcam/yolo4-tiny/
label.json          md5sum.txt          yolo4-tiny.prototxt  yolo4-tiny.xmodel
```

檔案撰寫範例與此說明文件放置同一目錄，可供參考

◆ Step 5: 執行 smartcam 並觀察結果

```
sudo xmutil unloadapp
sudo xmutil loadapp kv260-smartcam
sudo smartcam --mipi -W 1920 -H 1080 --target dp -a <your task name>
```