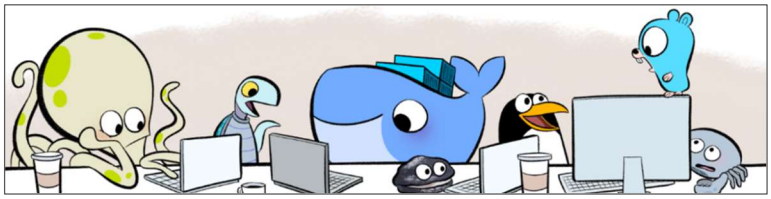
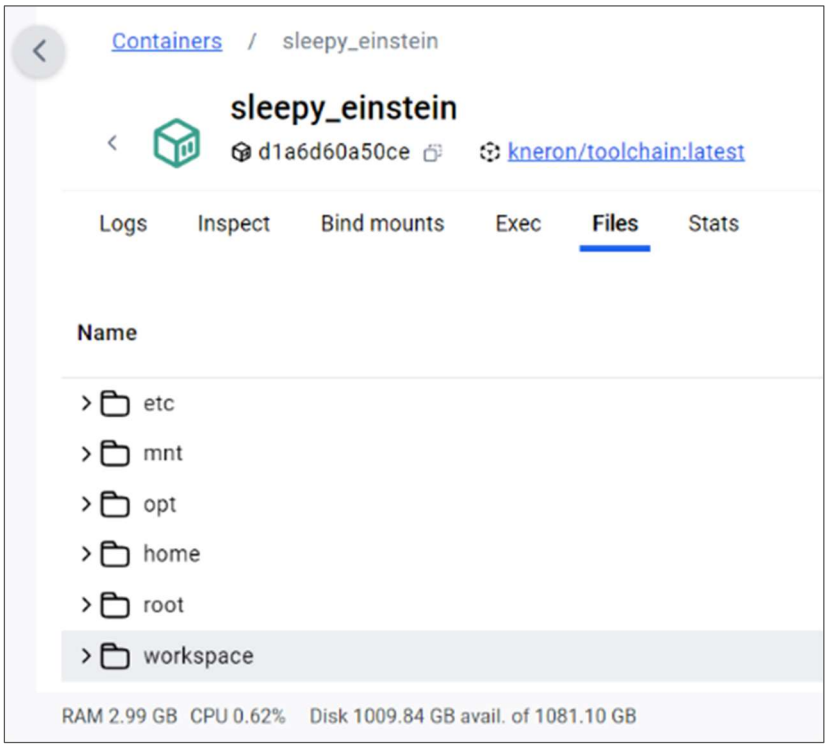
***新手建議從頭閱讀本文件，不要直接跳到後面去，並事先會使用板子 (本文件最後面有耐能官方手冊中「易混淆路徑」之說明)。**

耐能官方手冊網址

https://doc.kneron.com/docs/#toolchain/manual_1_overview/



(1.2.) Docker 是輕量化的虛擬機 (就是一台 Linux 主機的概念)，在這個虛擬機中，有一個名為/workspace 的目錄：



(1.3.) 每當在 WSL 執行「`docker run --rm -it -v /mnt/docker:/docker_mount kneron/toolchain:latest`」，Docker 就會從 `kneron/toolchain:latest` 這個映像檔，在 WSL 中啟動一個新的虛擬機 (就是 Container)，因此每次都會還原。

後面部分中「`/mnt/docker:/data1`」的意思是，把 Docker 中的 `/data1` 資料夾掛接到 WSL 的 `/mnt/docker`，其中的檔案不會被還原清除。用 VSCode 左下角的遠端連接，可以開啟 Container (Attach to ...)。

耐能手冊 Appendix 中的 Yolo Example (`/data1/keras_yolo3/`)

程式執行順序("dadin852"開頭)： `_onnx.py → _inf_onnx.py → _BIE_NEF.py → _inf_BIE.py → _inf_NEF.py`
範例：「`(base) root@79d94489376a:/data1/keras_yolo3# python dadin852_onnx.py`」

(Yolo Example Step 0) 不能什麼都用 wget 從網路上下載，因為檔案會有問題；可以下載到 WSL 後，再用指令，如「`sudo cp ~/Toolchain/yolov5s.onnx /mnt/docker/`」，複製到 Docker 中。

(Yolo Example Step 2) 我用.pt 模型轉換失敗，所以只好乖乖照著範例用.weights 模型 (2024/11/26)。轉換的時候，tiny.cfg 可以把原本的大模型轉換成 tiny 模型，但不確定轉出來的模型可否使用。 .pt 轉換成.onnx 的過程詳後面「部署 YOLOv5s 到 KL630 上」。

(Yolo Example Step 3) 關於 ktc.ModelConfig(...)的參數：model ID 可隨意輸入一個 int，但儘量用 32769 以上，因為 32768 以內的 models 是 Kneron 內部 ID；version 可隨意輸入。

(Yolo Example Step 4) 辨識的時候 ktc.kneron_inference()，記得改 num_classes，否則會報錯。

(dadin852_process.py) postprocess() 裡面用到的 anchors.txt，包含在 Step 0 轉換.onnx 時用的.cfg 當中。其輸出的 det_res[0]座標格式為"yxyx"。

(下頁還有另一部分)

部署 YOLOv5s 到 KL630 上

https://doc.kneron.com/docs/#model_training/

點擊此網頁中的「YoloV5s」，就可進到「Object Detection 手冊」

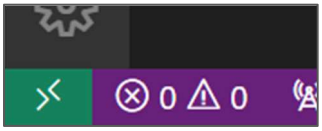
前置說明

- 因為耐能目前只支援 LeakyReLU (2024/11/29)，所以 yolov5 的模型得用 v2.0(含)以下的~~，所以 yolov5 我是 clone v2.0 的~~。把模型丟到 Netron 網站上，就能夠看到是 LeakyReLU 還是 SiLU。假如有 node 使用到不支援的 operator，也不一定表示無法使用，只是該節點的運算會轉由 CPU 負責而非 NPU，因此在速度上會有差。
- 耐能目前不支援 yolov5 官方 GitHub 下載的 v7.0 yolov5s.onnx (2024/11/29)，因為其 opset=17。

~~- 承上，v2.0 yolov5s.pt 若用 v7.0 的 export.py 轉換，會報錯，所以我用 v2.0 的 export.py。用 export.py 將.pt 轉成.onnx 時，要在 WSL 中執行，而非 Docker。~~

開啟 Docker

- 在 WSL 中「`docker run --rm -it -v /mnt/docker:/data1 kneron/toolchain:latest`」
- 到 VSCode 中打開遠端連結：



左下角此綠色按鈕

- 按 Attach to Running Container

.pt 轉成.onnx

- 先從 ultralytics/yolov5 官網取得 yolov5s.pt v2.0 (https://github.com/ultralytics/yolov5/releases/tag/v2.0 打開下方 Assets)。可下載到 WSL 後，再用指令，如「`sudo cp ~/Toolchain/your_model.pt /mnt/docker/`」，複製到 Docker 中。
- 開啟資料夾「`/workspace/ai_training/detection/yolov5/`」
- 在 Terminal 中「`cd yolov5/`」到「`… detection/yolov5/yolov5/`」

■ 啟動虛擬環境「`conda activate base`」

■ 參見「Object Detection 手冊」中的「Exporting ONNX …」小節，先修改所用 yaml 檔中的模型路徑，接著執行指令「`(base) root@b35cf7c7922d:/workspace/ai_training/detection/yolov5/yolov5# python ../exporting/yolov5_export.py --data /data1/yolov5/dadin852_paths_630.yaml`」

(noupsample 是給 KL520 用的，參見「Object Detection 手冊」最下面)

(■ 手冊中的下一節「Converting onnx …」我沒有用，但也能成功辨識)

~~■ 把 utils/utils.py 中所有的 cv2 直接註解掉，否則會報錯；接著額外 `pip install onnx`；然後 `cd yolov5/` 執行「`export PYTHONPATH="$PWD" && python models/export.py --weights your_model.pt --img-size 640`」~~

用.onnx 進行辨識(非必要)

■ (承上，在同一個目錄中) 執行「`python inference.py --data /data1/yolov5/dadin852_paths_630.yaml --img-path /data1/test_images/000000350003.jpg --save-path /data1/output.jpg --onnx`」

辨識結果會儲存在`--save-path`

(■ 根據我目前理解，「inference_e2e.py end-to-end 推論」是將辨識結果輸出成 json，不會有圖片；所以程式可以直接分析辨識結果，不需要倚賴人來「看」圖片)

■ yolov5 的 pre/post-process 和 yolov3 是不通用的

~~■ /yolov5/yolov5_postprocess.py 第 290 行應該改成「`bs, _, ny, nx, _ = x.shape # x(bs,3,20,20,85)`」~~

.onnx 轉成.nef

■ 在 Docker 中開啟資料夾「`/data1/`」

■ 啟動虛擬環境「`conda activate base`」

■ 修改此處.py 程式並執行「`(base) root@b35cf7c7922d:/data1# python yolov5/dadin852_BIE_NEF.py`」，此程式碼參考自：
https://www.kneron.com/forum/discussion/comment/1647/#Comment_1647

■ 轉出來的.nef 會位於 `/data1/kneron_flow/`

將.nef 放入板子(全科公司已經建置好的環境)

■ 板子開機會自動執行 `/mnt/flash/etc/rc.local`，執行 driver.sh 的指令包含於其中。可把 rc.local 中執行辨識的指令用「#」先註解掉，以便於手動執行。

- 在板子中執行「`cd /mnt/flash/plus/kp_firmware/kp_firmware_0/kp_firmware/bin/; export LD_LIBRARY_PATH=$(pwd)/lib;`」
- 先在主機啟動 TFTP 伺服器，再使用指令如「`tftp 192.168.3.100(主機 IP) -gr your_models.nef`」，把.nef 模型放進板子的 `./nef/` 中 (可以再 `chmod +x your_models.nef`)
- 用 vi 修改 ./ini/host_stream.ini 中「`[nnm] ModelPath ModelId`(見 dadin852_BIE_NEF.py 中的註解) `[voc] voc_enable=1`」
- 運行執行檔「`./kp_firmware_host_stream`」(不用 `rtsps` 也可從 HDMI 輸出辨識結果)

易混淆路徑之 Full Path

耐能官方手冊中所寫的路徑，常常只有一部分，容易讓人混淆。

在 Docker (kneron/toolchain:latest) 中

`/workspace/ai_training/detection/yolov5/yolov5/data/pretrained_paths_720.yaml`

`…/yolov5/yolov5/tutorial/tutorial.ipynb`

在 kneron_plus_vX.X.X.zip 中

`kneron_plus/python/example_model_zoo/KL720KnModelZooGenericImageInferenceYolov5.py`