DOCUMENTACION RV1103
PRACTICA PROFESIONAL SUPERVISADA

Estudiante: Bautista Garcia
Supervision: Cesar Estrebou

Inferencia Optimizada en Microcontroladores con NPU: Caso RV1103

Universidad Nacional de La Plata
Facultad de Informatica e Ingenieria

Universidad Nacional de La Plata
Facultad de Informatica e Ingenieria

Contents

Contents

1 Configuracion del Hardware

La placa de desarrolo a utilizar es una **Luckfox Pico**, con su chip integrado **RV 1103G**. Esta placa no cuenta con una **memoria flash** (SPI Nand), por lo que se le debe proveer con un sistema operativo desde una tarjeta micro-SD.

1.1 Buildroot en Luckfox Pico RV1103

1) Configurar SDK:
Se recomienda hacerlo dentro de un docker ya que el SDK esta principalmente testeado con Ubuntu 22.04. Si se trabaja con Windows se recomienda WSL2.

```
docker pull luckfoxtech/luckfox_pico:1.0

docker run -it --name luckfox luckfoxtech/luckfox_pico:1.0 /bin/bash



cd /home
git clone https://github.com/LuckfoxTECH/luckfox-pico.git
cd luckfox-pico
```

2) Build buildroot:

```
# Exportar tool
source tools/linux/toolchain/arm-rockchip830-linux-uclibcgnueabihf/
env_install_toolchain.sh

# Opcional: Si ya se hicieron builds previamente
./build.sh clean

# Elegir configuraciones de placa (SD-CARD y RV-1103)
./build.sh lunch

# Ejecutar build
./build.sh
```

3) SD Card burning: Con los archivos **`.img`** creados en el build ejecutar la herramienta **soctoolkit** del siguiente link (SD card image burning). Ingresar la micro-SD a la placa y conectarse via Ethernet por usb-c.

NOTAS:

- El SoC toolkit no reconoce **micro-SD** conectadas a un lector interno de la computadora. Se requiere una interfaz intermedia USB ↔ micro-SD.

Figure 1: Interfaz requerida

- Recordar ejecutar SD card formatter y Soc toolkit en **modo administrador**. De lo contrario no va a reconocer la micro-SD.

4) Conexion via SSH:

```
# Alternativas de acceso
ssh root@172.32.0.93
ssh pico@172.32.0.70

# Password
password: luckfox
```

En ocasiones cuando la placa no se inicia hace tiempo, puede ser necesario reconectar la micro-SD a la misma, para que el booting funcione correctamente.

2 Inferencia usando NPU

1) Docker: Se recomienda usar la imagen de Docker preconfigurada por **rockchip**, que ya contiene los archivos de compilacion.

- Descargar imagen preconfigurada de Docker: Docker-toolkit-2.3.0

```
# Una vez descargada la imagen, en el directorio donde se va a montar el
container
docker load --input rknn-toolkit2-v2.3.0-cp38-docker.tar.gz

# En mi caso /Volumes/rknn es la carpeta donde tengo el GCC cross compiler
enviado a la placa previamente y el rknn_model_zoo donde se encuentran los .sh
y makefiles para la conversion y compilacion de los modelos.
docker run -it --rm ^
  -v C:\rknn:/toolchain ^
  -v C:\rknn:/workspace ^
  --platform linux/amd64 ^
  rknn-toolkit2:2.3.0-cp38 /bin/bash
```

2) Preparamos el modelo:

```
cd toolchain/Projects/rknn_model_zoo/examples/yolov5/model

# Si Docker no contiene wget instalado
apt update
apt install -y wget

# Script de descarga
./download_model.sh

# Posible error: ^M$ al final de la ruta en script de descarga (eliminarlo
usando sed -i 's/\r$//' download_model.sh) corrompe la descarga

# Conversion a .rknn
cd toolchain/Projects/rknn_model_zoo/examples/yolov5/python


# Usage: python convert.py model_path [rv1103|rv1103b|rv1106|rv1106b] [i8/fp]
```

```
[output_path]
python convert.py ../model/yolov5s_relu.onnx rv1103 i8 ../model/
yolov5s_relu.rknn
```

3) Compilacion ejecutable C: La herramienta de cross compilation se encuentra en cross-compilation tool (fetch code: rknn). Se recomienda colocar la misma en `Projects/toolchain`.

```
cd Projects/rknn_model_zoo

cd toolchain/Projects
vi build-linux.sh
# Agregar ruta de compilacion (/toolchain montado en docker) a build-linux.sh
#!/bin/bash
export GCC_COMPILER=/toolchain/Projects/arm-rockchip830-linux-uclibcgnueabihf/
bin/arm-rockchip830-linux-uclibcgnueabihf


# Si no tiene instalado cmake
apt update
apt install -y cmake

# Ejecutar script de compilacion
./build-linux.sh -t rv1103 -a armhf -d yolov5
```

Al ejecutar el script de **build-linux** el `TARGET_SOC` que aparece es el `rv1106`. Esto es normal ya que el compilador no distingue entre el `rv1103` y el `rv1106`

4) Enviar ejectuable a la placa:

```
scp -r "C:
\rknn\Projects\rknn_model_zoo\install\rv1106_linux_armhf\rknn_yolov5_demo"
root@172.32.0.93:/data
```

5) Ejecutar modelo:

```
cd /data/rknn_yolov5_demo/
```

```
# Set the library dependency environment
export LD_LIBRARY_PATH=/data/rknn_yolov5_demo/lib


# Ejecutar
./rknn_yolov5_demo model/yolov5s_relu.rknn model/bus.jpg

# Traer resultados
scp root@172.32.0.93:/data/rknn_yolov5_demo/out.png C:\rknn\
```

Si se ejecutaron todos los pasos correctamente se deberia obtener:

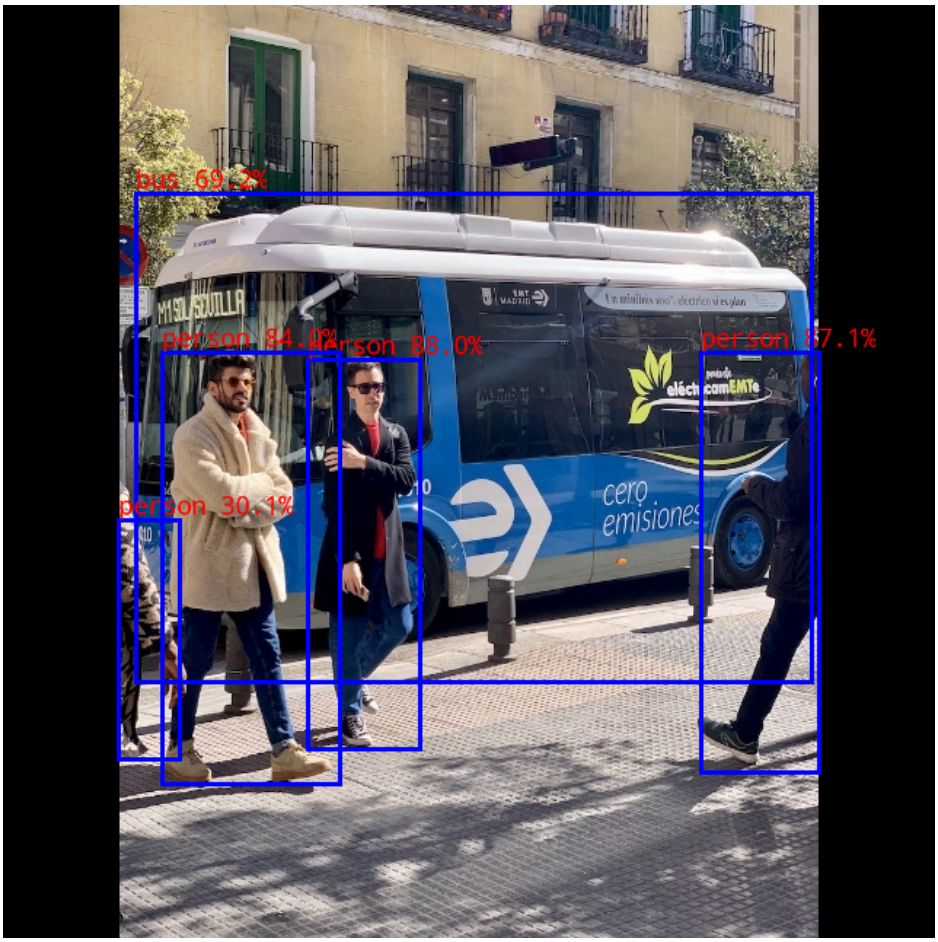

Figure 2: YOLOV5 Results

3 Inferencia usando CPU ARMv7 NEON Cortex

En caso de querer hacer inferencia usando exclusivamente la CPU, se debe usar **TFLITE**, sublibreria de Tensorflow adaptada para los microcontroladores. Para mas informacion sobre como usar TFLITE (tambien encontrado como LiteRT) ver **Build LiteRT**.

3.1 Build de Libreria

El primer paso es hacer el build de LiteRT usando el **cross-compiler** provisto por Rockchip (el mismo que usamos para compilar la inferencia de NPU). Ademas se debe tener en cuenta:

- Se debe deshabilitar **XNNPACK** ya que no funciona con el cross-compiler de luckfox.

1) Build usando CMake: Tambien se puede generar el build usando **Bazel** (se debe preinstalar usando algun sistema de paquetes):

```
# Se debe instalar FP16 ya que el cross-compiler de Rockchip no cuenta con la
misma
git clone --depth 1 https://github.com/Maratyszcza/FP16.git
# Copiar dentro del cross-compiler
cp -r FP16/include/fp16  $/toolchain/Projects/arm-rockchip830-linux-
uclibcgnueabihf/bin/../arm-rockchip830-linux-uclibcgnueabihf/sysroot/usr/
include/
```

```
# Clonar tensorflow --depth 1 (para no traer todo el commit history)
git clone --depth 1 https://github.com/tensorflow/tensorflow.git

# Comando de configuracion CMake (se puede hacer en un CMakeLists.txt de forma
mas ordenada usando variables de PATH)
cmake ../tensorflow/tensorflow/lite -DCMAKE_SYSTEM_NAME=Linux -
DCMAKE_SYSTEM_PROCESSOR=armv7-a -DCMAKE_C_COMPILER=/toolchain/Projects/arm-
rockchip830-linux-uclibcgnueabihf/bin/arm-rockchip830-linux-uclibcgnueabihf-gcc
-DCMAKE_CXX_COMPILER=/toolchain/Projects/arm-rockchip830-linux-uclibcgnueabihf/
bin/arm-rockchip830-linux-uclibcgnueabihf-g++ -DCMAKE_SYSROOT=/toolchain/
Projects/arm-rockchip830-linux-uclibcgnueabihf/bin/../arm-rockchip830-linux-
uclibcgnueabihf/sysroot -DCMAKE_BUILD_TYPE=Release -DTFLITE_ENABLE_XNNPACK=OFF
-DTFLITE_HOST_TOOLS_DIR=/usr/bin  -DFLATBUFFERS_LOCALE_INDEPENDENT=0 \
-DFP16_SOURCE_DIR=/toolchain/Projects/arm-rockchip830-linux-uclibcgnueabihf/
bin/../arm-rockchip830-linux-uclibcgnueabihf/sysroot/usr  -DCMAKE_C_FLAGS='-
```

```
DTF_MAJOR_VERSION=2 -DTF_MINOR_VERSION=16 -DTF_PATCH_VERSION=1 -
DTF_VERSION_SUFFIX=\"\" -DFLATBUFFERS_LOCALE_INDEPENDENT=0' -
DCMAKE_CXX_FLAGS='-DTF_MAJOR_VERSION=2 -DTF_MINOR_VERSION=16 -
DTF_PATCH_VERSION=1 -DTF_VERSION_SUFFIX=\"\" -
DFLATBUFFERS_LOCALE_INDEPENDENT=0'
```

- El comando **cmake** mostrado sirve para arquitecturas **ARMv7** con soporte de Neon y VFP4
- Los flags de ***Sysroot y Compiler*** dependen de las rutas especificas del compilador usado (estas corresponden al cross-compiler de Rockchip).
- Las C y Cxx flags agregadas se deben a que al deshabilitar XNNN, existen flags de versionado que no se incluyen en el branch de `#IF` determinado por el flag.

Para que esto funcione y no existan errores de XNNPACK, se tuvo que agregar al codigo fuente de LiteRT el siguiente fragmento (probablemente sea un bug):

```
# Error: cannot specify compile options for target "xnnpack-delegate" which is
not built by this project


# Solucion:
# tensorflow/tensorflow/lite/CMakeLists.txt
if(TFLITE_ENABLE_XNNPACK)
  target_compile_options(xnnpack-delegate
  PUBLIC ${TFLITE_TARGET_PUBLIC_OPTIONS}
  PRIVATE ${TFLITE_TARGET_PRIVATE_OPTIONS}
  )
endif()
```

2) Make de librerias estaticas:

```
# En carpeta donde se encuentra el build
make -j4 tensorflow-lite

# Error posible: CMake Error at /toolchain/Projects/tf-build/flatbuffers/
CMakeLists.txt:115 (check_symbol_exists):
Unknown CMake command "check_symbol_exists".

# Solucion (# tf-build/ es donde tengo el build de CMake):
sed -i '115i include(CheckSymbolExists)' tf-build/flatbuffers/CMakeLists.txt
```

3) Compilacion: Con todo instalado ya podemos armar un programa que haga inferencia de un modelo `.tflite` usando la api construida **`#include "tensorflow/lite/c/c_api.h"`**. Luego para compilarlo a un unico archivo estatico (que sera enviado a la placa), debemos incluir en la compilacion todas las librerias necesarias que se encuentran en **`/_deps`** dentro del build realizado:

```
# Compilacion de /tflite/main.c
/toolchain/Projects/arm-rockchip830-linux-uclibcgnueabihf/bin/arm-rockchip830-
linux-uclibcgnueabihf-g++ -I/toolchain/Projects/tensorflow -O3 -flto -
march=armv7-a -mfpu=neon-vfpv4 -mfloat-abi=hard -static /toolchain/Projects/
tflite/main.c /toolchain/Projects/tf-build/libtensorflow-lite.a -Wl,--start-
group /toolchain/Projects/tf-build/_deps/ruy-build/ruy/libruy_*.a "/toolchain/
Projects/tf-build/_deps/cpuinfo-build/libcpuinfo.a" "/toolchain/Projects/tf-
build/_deps/fft2d-build/libfft2d_fftsg.a" "/toolchain/Projects/tf-build/_deps/
fft2d-build/libfft2d_fftsg2d.a" "/toolchain/Projects/tf-build/_deps/
flatbuffers-build/libflatbuffers.a" "/toolchain/Projects/tf-build/_deps/
farmhash-build/libfarmhash.a" "/toolchain/Projects/tf-build/_deps/gemmlowp-
build/libeight_bit_int_gemm.a" /toolchain/Projects/tf-build/_deps/abseil-cpp-
build/absl/**/libabsl_*.a -Wl,--end-group -lpthread -ldl -lm -o /toolchain/
Projects/tflite/mobilenet_run
```

- El primer PATH es el cross-compiler de C++ mas alla de que la inferencia se haga con un `.c`.
- Entre los comandos **`Wl`** se incluyen todos los `.a` de las dependencias, para que en la compilacion se genere un unico archivo ejecutable (se copia el contenido de los .a al binario ejecutable) y no tener que enviar un `/lib` a la placa.

4 Bibliografia y Recursos

Todos los archivos mencionados o links para sus respectivas descargas, datasets, scripts de inferencia o conversion y archivos de configuracion se encuentran en **RV1103-Github**

- Documentacion oficial de Luckfox Pico (booteo e instalacion buildroot): Luckfox-Docs.
- RKNN Toolkit 2: Github Toolkit. *En* `/docs` *se encuentra documentacion muy valiosa sobre como usar sus distintas API y ejecutar modelos de prueba.*
- Inferencia usando LiteRT: LiteRT Docs.
- Informacion sobre Mobilenet V2: Keras Mobilenet

Para mas consultas contactarse a: **bautistagarciace@gmail.com**