Benchmarking the NXP i.MX8M+ neural processing unit: smart parking case study

Evaluando la unidad de procesamiento neuronal NXP i.MX8M+: el caso de estudio de parqueos inteligentes

Edgar Chaves-González[1], Luis G. León-Vega[2]

Chaves-González, E.; León-Vega, L.G. Benchmarking the NXP i.MX8M+ neural processing unit: smart parking case study. *Tecnología en Marcha*. Vol. 35, special issue. IEEE International Conference on Bioinspired Processing. December, 2022. Pág. 26-30.

https://doi.org/10.18845/tm.v35i9.6487

1 Embedded Software Engineer & Graduate Student Intern. RidgeRun LLC, Costa Rica. E-mail: edgar.chaves@ridgerun.com
https://orcid.org/0000-0002-0269-526X

2 Senior Embedded Software Engineer. RidgeRun LLC, Costa Rica. Master's in Electronics Student. Instituto Tecnológico de Costa Rica. Costa Rica. PhD fellow. Università degli Studi di Trieste. Italy. E-mail: luis.leon@ridgerun.com
https://orcid.org/0000-0002-3263-7853

Keywords

Computer vision; AI accelerator; embedded software; smart cameras; convolutional neural networks; TinyYOLO; Rosetta.

Abstract

Nowadays, deep learning has become one of the most popular solutions for computer vision, and it has also included the Edge. It has influenced the System-on-Chip (SoC) vendors to integrate accelerators for inference tasks into their SoCs, including NVIDIA, NXP, and Texas Instruments embedded systems. This work explores the performance of the NXP i.MX8M Plus Neural Processing Unit (NPU) as one of the solutions for inference tasks. For measuring the performance, we propose an experiment that uses a GStreamer pipeline for inferring license plates, which is composed of two stages: license plate detection and character inference. The benchmark takes execution time and CPU usage samples within the metrics when running the inference serially and parallel. The results show that the key benefit of using the NPU is the CPU freeing for other tasks. After offloading the license plate detection to NPU, we lowered the overall CPU consumption by 10x. The performance obtained has an inference rate of 1 Hz, limited by the character inference.

Palabras clave

Visión por computador; Acelerador de AI; software empotrado; cámaras inteligentes; convolutional neural networks; TinyYOLO; Rosetta.

Resumen

Actualmente, el aprendizaje profundo se ha convertido en una de las soluciones más populares para la visión por computador y ha incluido también el *Edge.* Esto ha influenciado a los productores de *System-on-Chip* (SoC) a integrar aceleradores para tareas de inferencia en sus SoC, incluyendo a NVIDIA, NXP y Texas Instruments. En este trabajo se explora el rendimiento de la unidad de procesamiento neuronal (NPU) de la NXP i.MX8M Plus como una de las soluciones de tareas de inferencia. Para las mediciones de desempeño, proponemos un experimento basado en un *pipeline* de GStreamer para la inferencia de placas de vehículos, el cual consta de dos etapas: la detección de placas y la inferencia de sus caracteres. Para ello, este *benchmark* toma muestras de tiempos de ejecución y uso de CPU dentro de sus muestras cuando se corre la ejecución de la inferencia de manera serial y paralela. Los resultados obtenidos demuestran que el beneficio en el uso del NPU radica en la liberación del CPU para otras tareas. Después de descargar la detección de placas al NPU, reducimos el consumo total de CPU 10 veces. El desempeño obtenido tiene una tasa de inferencia de 1 Hz, limitado por la inferencia de caracteres.

Introduction

Recently, there have been efforts for accelerating machine learning (ML) inference at the Edge. In 2020, NXP launched the i.MX8M Plus, an ARM-based SoC equipped with a Neural Processing Unit. It provides up to 2.3 TOP/s, supports 8-bit and 16-bit fixed-point arithmetic and implements the Winograd algorithm [1] for convolutions.

NVIDIA, another SoC vendor, proposes other similar solutions in the market. The NVIDIA Jetson Nano is the closest board in memory, CPU, and peripherals to the i.MX8M Plus. It is equipped with an NVIDIA Maxwell GPU that facilitates and accelerates algorithms using CUDA [2] and delivers up to 472 GOP/s (4.87x less than the theoretical of i.MX8M Plus) [3]. According to our previous measurements, we can run a TinyYoloV2, a deep neural network architecture for object detection on images, on a Jetson Nano, delivering up to 15 frames per second (fps) [4].

The application fields of these embedded systems are vast, going from home control and security to artificial intelligence, automated driving, healthcare, and others. They are preferred over cloud solutions because of their reliability in inference latency and power consumption, which does not reach more than 20W [1].

This work explores the performance benefit of offloading deep learning (DL) models to the NPU included in the i.MX8M Plus [1]. For this study, the system runs a license plate inference (LPI) pipeline implemented in the RidgeRun GstInference framework [5], utilizing TensorFlow Lite as DL backend and NNAPI as an accelerator-backend interface [6]. The key contribution of this work is a case study of the usage of the i.MX8M Plus in real-world applications, in particular, smart parking.

Benchmark architecture

Our work is based on a smart parking use case. It is an industry-coming sample to evaluate boards with an actual application environment. It consists of a camera recording the license plates of the cars coming into the parking lot. The camera is connected directly to the board, where an LPI task is executed on the captured images, and the results are uploaded to the Internet for data logging.

Our LPI architecture is two-fold. The first stage performs license plate detection (LPD) to recognise the license plate as an object within the image. It is based on an in-house trained TinyYOLOV3 model architecture [3]. The model delivers fixed bounding boxes quantifying their confidence and classifying it into several classes. The result is transferred to the license plate recognition (LPR) stage, based on Rosetta [7], to get the inference in a character string format.

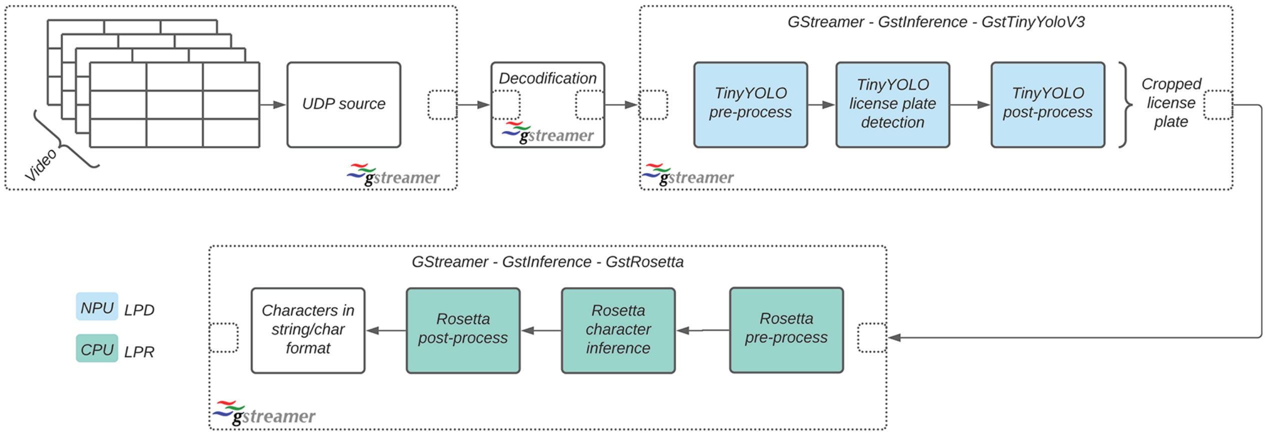
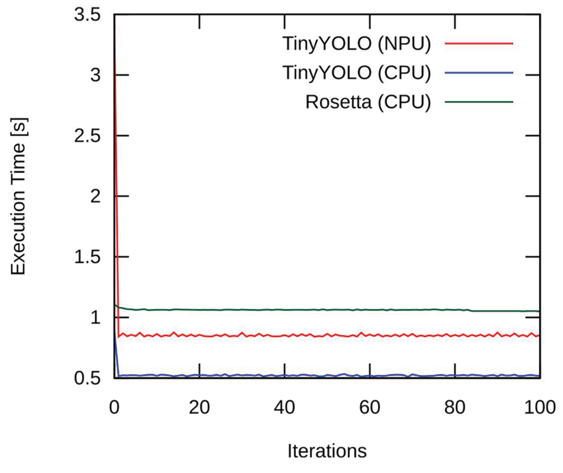


Figure 1. Dataflow of the LPD and LPR algorithms within a GStreamer pipeline based on GstInference

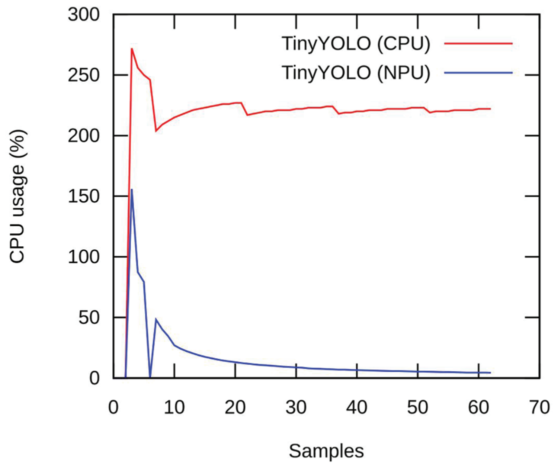
The inference data flow is implemented in a GStreamer pipeline, as presented in Figure 1. We have chosen GStreamer because of its support, portability and performance [8]. To add GStreamer support to the models, we wrapped TinyYOLOv3 and Rosetta into two GStreamer elements for isolating the LPD from LPR algorithms. Thus, we manage to place each algorithm in a separate thread using a RidgeRun's solution called GstInference, which makes our solution portable amongst several inference backends and devices (CPU, GPU, and TPU). According to Figure 1, our pipeline starts with a UDP source to receive the data from a network camera; then, the image is decoded and transformed into a suitable format for our GStreamer wrappers. Within our GStreamer elements (GstTinyYoloV3 and GstRosetta), we preprocess the data, perform the inference and post-process to place the results as metadata circulating within the pipeline.

Regarding the measurements, the benchmark takes samples of the CPU usage using the Linux PS at 2 Hz for 100 iterations to measure the results. Our setup is capable of running in two modes: i) individually, executing the two models isolated, and ii) in parallel, running both inference tasks concurrently. The time is estimated using the high-precision clock from the C++ Standard Template Library (STL).

Results



(a) Execution time for the inference time of each inference element.



(b) Combined CPU usage of using TinyYOLO and Rosetta together

Figure 2. Inference performance metrics. The NPU offloading benefits in freeing 10x the CPU usage. Rosetta was running on the CPU all the time.

After running our benchmark, we obtained the results presented in Figure 2. Figure 2.a shows the execution time of each algorithm running isolated (one at a time). Figure 2.b presents the results when running the whole system together using two configurations: i) TinyYOLOv3 on CPU, and ii) TinyYOLOv3 on NPU. In the first iterations, the inference is slow in the beginning because of a warmup process (context and resource allocation).

TinyYOLOv3 can run on both the NPU and the CPU regarding the individual runs. With NPU, the inference rate was 1.18 Hz, whereas the CPU was 1.69x faster, with 1.99 Hz. Rosetta could not be accelerated because the model does not support quantisation. It managed to run at approximately 1 Hz.

Figure 2.b presents the CPU consumption of running the LPD in the NPU and the CPU while executing the LPR parallel on the CPU. Running both algorithms on the CPU leads to consumption above 200%, which means that more than two cores are fully occupied during the inference tasks. On the other hand, when running the LPD on NPU and the LPR on the CPU, the CPU consumption drops below 20% (10x gained), freeing the CPU for other tasks. In both cases, the inference rate is 1 Hz. Hence, using the NPU does not benefit the inference rate, and it contributes to lowering the CPU utilization. According to our results for TinyYOLOv2 in the Jetson Nano, we expected to have around 15 Hz but the actual rate achieved was 2 Hz. It also highlights the resource underutilisation of the Rosetta model.

Conclusions

In this work, we have exposed a benchmark using a smart parking application for evaluating the NXP i.MX8M Plus for deep learning inference tasks, particularly for license plate detection and recognition. For this case study, we offloaded the LPD (TinyYOLOv3) to the NPU, freeing 10x of the CPU resources for other tasks, performing similarly and without impacting the final inference rate. We managed to get TensorFlow running at 1.99 Hz maximum and Rosetta at 1 Hz, posing the bottleneck of our application. In future work, we plan to extend this application to other platforms with similar power consumption and capabilities, such as NVIDIA Jetson Nano and Xilinx ZYNQ 7000.

Acknowledgements

This project was possible thanks to RidgeRun LLC and their Master's scholarship and internship programmes.

References

[1] "i.MX8M Plus", Nxp.com, 2021. [Online]. Available: https://www.nxp.com/products/processors-and-microcontrollers/arm-processors/i-mx-applications-processors/i-mx-8-processors/i-mx-8m-plus-arm-cortex-a53-machine-learning-vision-multimedia-and-industrial-iot:IMX8MPLUS. [Accessed: 22- Sep- 2021].

[2] NVIDIA. 2014. DATA SHEET NVIDIA Jetson Nano System-on-Module. [Online]. Available: https://developer.nvidia.com/embedded/jetson-nano [Accesed: 22- Mar- 2022]

[3] H. Bouzidi, H. Ouarnoughi, S. Niar, and S. Ait El Cadi. 2022. Performances Modeling of Computer Vision-based CNN on Edge GPUs. ACM Trans. Embed. Comput. Syst. DOI: https://doi.org/10.1145/3527169

[4] "GstInference Benchmarks", RidgeRun Embedded Solutions LLC. [Online]. Available: https://developer.ridgerun.com/wiki/index.php?title=GstInference/Benchmarks. [Accessed: 14- Sep- 2021].

[5] "Introduction to GstInference", developer.ridgerun.com. [Online]. Available: https://developer.ridgerun.com/wiki/index.php?title=GstInference/Introduction. [Accessed: 20- Oct- 2021].

[6] "Neural Networks API | Android NDK | Android Developers", Android Developers. [Online]. Available: https://developer.android.com/ndk/guides/neuralnetworks. [Accessed: 09- Sep- 2021].

[7] V. Sivakumar, A. Gordo and M. Paluri, "Rosetta: Understanding text in images and videos with machine learning", Facebook Engineering, 2021. [Online]. Available: https://engineering.fb.com/2018/09/11/ai-research/rosetta-understanding-text-in-images-and-videos-with-machine-learning/. [Accessed: 20- Sep- 2021].

[8] "GStreamer: open source multimedia framework", Gstreamer.freedesktop.org. [Online]. Available: https://gstreamer.freedesktop.org/. [Accessed: 14- Sep- 2021].