Contents

Part I:
Introduction

Neural Network training and inference in Embedded Environments. NN Optimisations, Tiny ML, Embedded Environments, i.MX6

Why perform training and inference on ECUs?
Federated Learning

1. Background

Describe ECU systems, Tiny ML

Hypothesis: Training and inference of (small) neural networks in embedded systems can be considerably improved compared to general purpose neural networks frameworks

1.1 Anomally Detection

Explain the problem. Introduce terminlogy that will get explained in the next chapter

1.2 Scania Embedded Systems

ECU systems, the kind of data they generate, and the potential of federated learning

1.2.1 i.MX6 Target Processor

i.MX6 specifications, constraints

1.3 Development for Embedded Linux

Short introduction to writing applications for embedded linux

1.3.1 Yocto Project

Outline the Yocto Project based Build environment

2. Theory

Theoretical foundations

2.1 Expectations for the Hardware

Layout the Architecture of the i.MX6 - ISA and specifications. Optimisation possibilities through using the SIMD etc

2.1.1 Training vs Inference

Requirements for Training and uploading the wieghts vs Inference

2.2 Neural Network Performance Optimisations Techniques

Contrast traditional implementations in resource rich environments and the constraints of embedded environment. Layout general strategies to acquire performance improvements with little losses to accuracy e.g Purning, Quantisation

2.3 Tiny Machine Learning

Approaches to doing Machine Learning in Embedded Environments

2.4 ARM's CMSIS-NN

Introduction to ARM CMSIS-NN Kernels

Part II:
Implementation

Approach to writing Tiny Neural Networks and the nature of their target environments

3. Design

Describe the system design of the Real-Time Linux environment, support provided for the neural network execution, performance engineering based on the hardware, and design and optimisation of the neural network that is executed

3.1 Embedded Linux Environment

Information regarding configuration and other details

3.1.1 Neural Network Support

Leveraging hardware support for the application from the OS layer

3.1.2 CMSIS-NN Kernels

Utilising ARM's CMSIS-NN Kernels

3.1.3 Hooks for the Applications

Application design

3.2 Tiny Neural Network

The architecture of the Anomally Detection Neural Network

4. Development

Details of how different neural networks were implemented

4.1 General C Implementation

4.2 CMSIS-NN Implementation

4.3 Testing on Device

Process for testing on device

4.3.1 Flashing the application

Where the device comes in the development loop

4.3.2 Performance Evaluation

Perf tools and profiling techniques

Part III:
Analysis

Results from the implementation

5. Results

The Performance of the different neural networks on i.MX6

5.1 Neural Network Performance

Performance measure considered. Execution times vs Neural Network Accuracy etc...

5.1.1 ADNN on Tensorflow-Lite

Performance of a CNN application performing Anomally Detection running on tensorflow lite

5.1.2 ADNN written in C

Performance of a similar network written in C

5.1.3 ADNN written using CMSIS-NN

Performance of a similar network utilising CMSIS-NN

5.2 Further Neural Network Optmisations

Further breakdown of the performance achieved from different optimisation techniques

5.2.1 Pruning the Network

6. Discussion

Which optimisation approaches gave the most in improvement?

7. Conclusion and Future Work

What does it all mean? Where do we go from here?

References

[1] Patrik W. Daly Helmut Kopka. *A Guide to LaTeX 2ε*. Addison-Wesley Professional, 4th edition, 2003.