1 Introduction

This contribution focuses on experimenting with neural networks on resource-constrained devices. Specifically, the ArduCAM Pico4ML board with a 324x324 grayscale camera, a 160x80 RGB display, and an RP2040 chip commonly found on the $4 Raspberry Pi Pico board. We demonstrate that an MNIST classifier can be executed on the ArduCAM board after constructing and training the model in the TinyGrad deep learning framework.

Refer to the project repository for the complete code.

Fig 1: A demo of the ArduCAM board running the MNIST classifier. Top = real-time camera feed, left = 28x28 input to NN, right = predicted digit.

2 Why TinyGrad?

A machine learning framework is necessary to train deep learning models. It provides the building blocks for constructing a model and then training it with your training data and optimiser of choice. The two mainstream frameworks are PyTorch and TensorFlow. While these frameworks are well established, their sophistication makes it challenging to integrate them with ML accelerator hardware or hardware-constrained microcontrollers like the RP2040. TinyGrad has a similar API surface to PyTorch but it is implemented in <5000 lines of Python (vs >220k for PyTorch or >3.7M for TensorFlow) and models trained with it can be exported to C via the CLANG compiler front-end.

3 Creating and training a model in TinyGrad

```
1 from tinygrad import nn, Tensor
2 class TinyMnistModel:
3   def __init__(self): self.layers = [nn.Linear(784, 400), Tensor.relu, nn.Linear(400, 10)]
4   def __call__(self, x: Tensor) -> Tensor: return x.sequential(self.layers)
```

Above is the Python code to define a fully connected neural network with a 1x784 input, 400 hidden nodes and 10 output nodes. We are training a model for the MNIST dataset of handwritten digits and each image in the dataset is 28x28=784 greyscale. Higher accuracy would be achieved with more than 400 nodes in the hidden layer but the 2MB flash limit on the RP2040 constrains the number of weights we can use. The full code to train the model with a dataset can be found in the source repository.

4 Compiling a TinyGrad model to C

To export the model to C code, the following code is used:

```
1 from extra.export_model import export_model
2 prog, inp_sizes, out_sizes, state = export_model(<model>, "clang", Tensor.randn(28 * 28))
3 prog = prog.replace("unsigned char buf", "const unsigned char buf")
4 open("minst.c", "w").write(prog)
```

Line 3 is necessary because the RP2040 has 264KB of SRAM which is insufficient for loading the weight arrays into memory. Using `const` arrays allows the arrays to remain in flash memory during execution. The resulting C file only depends on the C standard library. For model inference, it produces a `net()` function

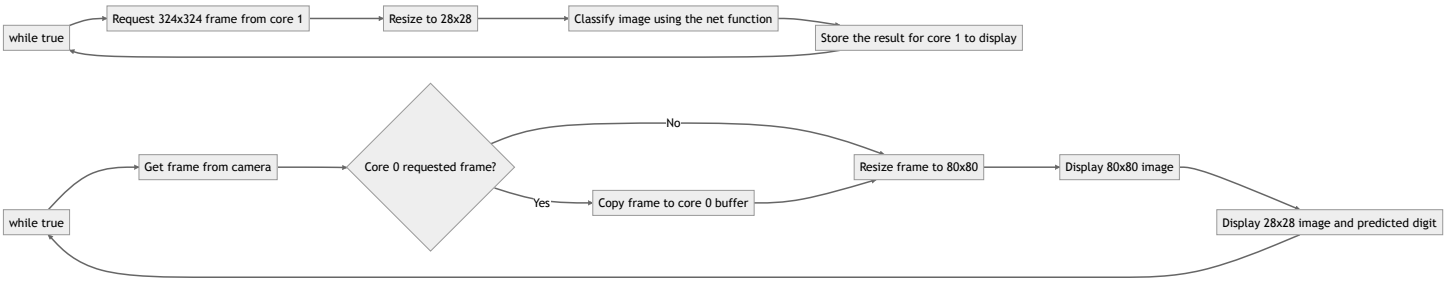

Fig 2: Flowcharts of the execution of core 0 (top) and core 1 (bottom).

with the signature: `void net(float *input0, float *output0)`. The dimensions of the input and output arrays are specified in `inp_sizes` and `out_sizes` from the above code snippet.

5 Using both cores to isolate model inference

The RP2040 has dual cores and the Raspberry Pi Pico SDK provides an API to execute code on both cores in parallel. Figure 2 outlines the execution of core 0 and core 1. Core 0 is responsible for performing model inference via the `net()` function generated by TinyGrad and core 1 is responsible for updating the display (see Figure 1). This separation allows the camera feed to update in real-time which creates the illusion that the processing is fast despite the limited processing power available.

6 Simplified development environment via Docker

A three-command Docker file has been produced to make it easier to reproduce this work. Furthermore, the repo contains a Devcontainer configuration that VSCode recognises so that VSCode development can be conducted within the Docker container.

7 Future Work

ML models can be compiled into ONNX, an intermediary format that bridges the gap between ML frameworks. An ONNX execution environment for bare metal devices is not available however, an approach to doing this is described in [1] and thus could be reproduced. TinyGrad supports ONNX so the conversion from ONNX $\rightarrow$ TinyGrad $\rightarrow$ C could be explored.

Training the MNIST model takes 3 minutes by default but takes over 10x longer when training with the CLANG flag required for exporting the model to C. Addressing this slowdown would be valuable.

TensorFlow Lite Micro and microTVM are alternative bare metal frameworks that could be benchmarked.

References

[1] Mark Deutel, Philipp Woller, Christopher Mutschler, and Juergen Teich. Energy-efficient deployment of deep learning applications on cortex-m based microcontrollers using deep compression. In *MBMV 2023; 26th Workshop*, pages 1–12, 2023.