Introduction to Tiny Deep Learning

Nemanja Milošević

Virtual Tech Workshop

Data Science Conference 2022, Belgrade, Nov 15th

Hi!

- Nemanja Milošević
- Assistant Professor @ University of Novi Sad, Faculty of Sciences
- @nmilosev
- nmilosev.svbtle.com

Agenda

- Introduction to “Tiny” Machine/Deep Learning, Edge Computing
- Use cases & Applications
- Hardware Overview + Simulators
- Software Overview
- Projects/Demos

Introduction to TinyML

- One of exciting and promising trends in ML and embedded computing
- Means of running ML models (both for inference and training) on embedded (Edge, IoT) devices
- Embedded devices → low cost, low power, low-footprint, no operating system (e.g. Arduino, Raspberry Pi Pico, ESP32, …)
  - Note: We consider a SBC like NVIDIA Jetson, Raspberry Pi, or an Android phone more “Edge” and less “Tiny” :)

Introduction to TinyML

- **Benefits**
  - Small and power efficient
  - Low cost, accessible
  - Large-scale application
  - Low latency
  - Reduced bandwidth costs
  - Security
  - Reliability

Introduction to TinyML

- **Limitations**
  - Memory
  - Computing power
  - Available software
- **TinyML/DL is *mostly* used for inference, not for training**
- **Devices can still be used for data collection, even fine-tuning**

Applications

- **The idea: Machine/Deep Learning Everywhere! :)**
- **If there is power/size/network/security constraints → consider TinyDL**
- **Some example applications**
  - Predictive Maintenance (Industrial)
  - Smart Agriculture
  - Smart Cities
  - Real-time healthcare
  - …

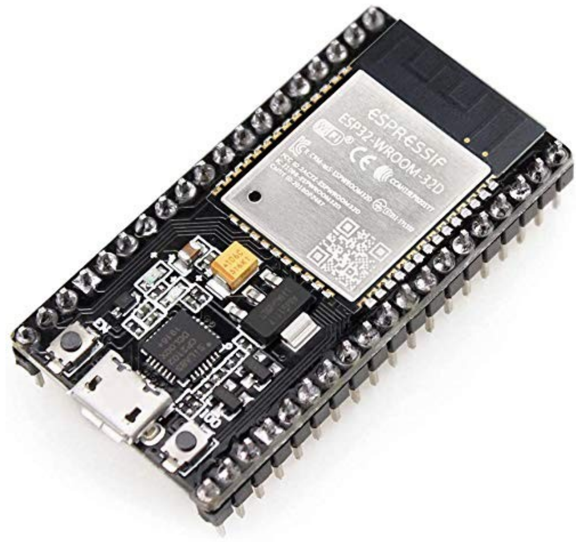
Hardware

- There is a lot of compatible hardware available
- Arduino Nano (and variants)
- Seeeduino XIAO (very small)
- Raspberry Pi Pico
- ESP32
- No hardware and still want to experiment? → https://wokwi.com/

ESP32

- Board we will use today
- Dual-core (240MHz, 32-bit, Xtensa CPU)
- 320 KiB RAM, 448 KiB + 4MB (external FlashROM) ROM
- WiFi, BT 4.2
- 22 GPIOs, 1 programmable LED
- Temperature sensor built-in, Hall effect sensor
- <$10

- Simulator → https://docs.wokwi.com/guides/esp32



Software overview

- Most boards include development tools
- There are also platform-agnostic SDK’s (e.g. PlatformIO)
- Many are compatible with Arduino IDE → Development in C
- Collect Data → Train model → (optional) Optimize Model for Edge → Flash model and communication code to the board
- There are also specific runtimes which can be used (e.g. CONNX)

Software overview - TensorFlow Lite for Microcontrollers

- TensorFlow “adapted” for microcontrollers
- Does not require operating system support, any additional libraries or Dynamic Memory Allocation (important for small devices)
- Many supported platforms (Espressif, Arduino, Adafruit, Wio, …)
- Build model → Convert model to TF Lite → Convert model to C byte array (xxd) → Done!
- Caveats: Limited number of supported operations, no on-device train

Software overview – MicroPython

- What if we don’t want to do all that stuff?
- TF Lite also supports MicroPython!
- MicroPython is a lean and efficient implementation of Python 3 with a subset of the standard library
- This means we can run “normal” pure Python code on the device itself
- Requires only 256kB of storage and 16kB of RAM
- Performance is limited (compared to C) but it is great for prototyping

Software overview – MicroPython

- MicroPython runs on bare-metal, with support for many boards
- It also has packages for hardware control (e.g. pin control, WiFi control, CPU frequency control, …)
- Has a package manager (upip), web server implementation, mqtt clients, numpy-like libraries, ...
- It also runs on Windows, Linux, Mac
- Run it in Docker:

```
docker run -it --entrypoint bash micropython/unix:v1.18
```

Software overview – MicroPython w/TF

- MicroPython does not have numpy → use ulab (similar/same API, limited functionality) – https://github.com/v923z/micropython-ulab
- MicroPython with ulab and TFLite builds are available here: https://github.com/mocleiri/tensorflow-micropython-examples
- Flash firmware → Copy TFLite model and code to run it → Done

Software overview – What about “normal” ML?

- For some use cases you may want to use non-deep Machine Learning algorithms
- Very good solution → https://github.com/BayesWitnesses/m2cgen
- m2cgen transforms ML models to pure code
- We can run pure Python code :)
- Support for linear models, SVM’s, Decision Trees, Random Forests, Boosted RFs
- Support for: scikit-learn, lightning, XGBoost, LightGBM

Software overview – IDEs

- **You can use your favorite Python IDE (Pycharm, VSCode, …)**
- **We just need support to flash the built firmware (once)**
- **Thonny IDE is nice for this (https://thonny.org/)**
    - Open-source Python IDE
    - Built with microcontroller/micropython development in mind
    - Serial port REPL
    - Built-in: filesystem management, flashing, firmware download/upload etc.

Last thing: What about model optimization?

- Models can be quantized after training or we can use QAT (Quantization Aware Training)
- We can also use operation fusion and similar techniques
- DL Compilers are also available for some platforms

Projects

- Time to build!

  1. Lets get familiar with the board and micropython (Blink LED project)
  2. Lets build a RandomForest model for the Iris dataset and deploy it to the device and run inference (m2cgen)
  3. Lets build a small TFLite neural net model, quantize it to 8 bits, and deploy it
  4. Lets use a custom neural net implementation and train Iris on the device
  5. IoT on-device anomaly detection (full example)

- You can try 1-4 in the simulator also! (you can also wire up various sensors)

Questions? :)