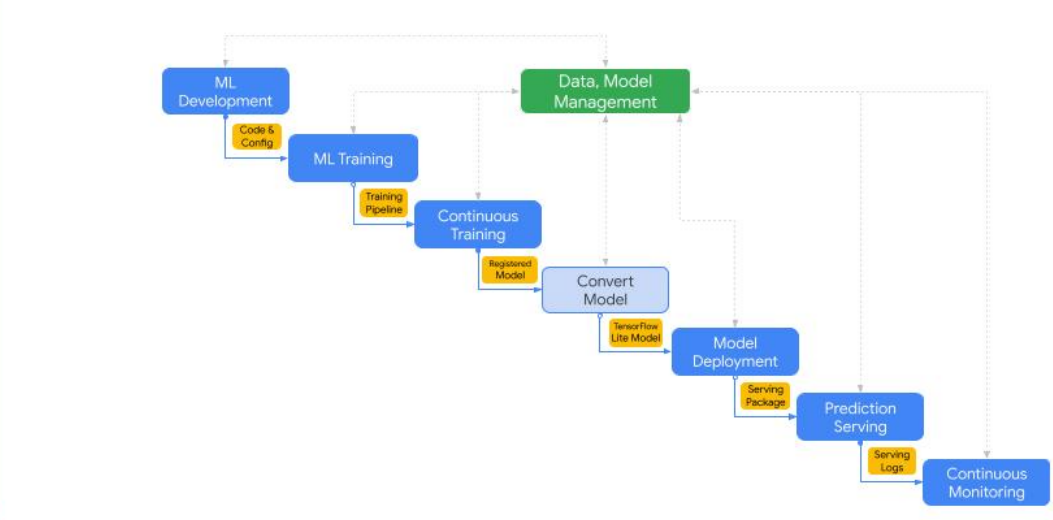
Overview of Model Conversion



Objectives

While running standard Tensorflow, Pytorch, or Keras models in Colaboratory is very useful on a desktop or laptop computer, they are not very useful when it comes to microcontrollers - the models are just way too big! These frameworks include many features and as such are very large. When running models only for inference, most components are non-essential and can be stripped out to save on memory and compute overhead. This is exactly what Google did with TensorFlow, culminating in TensorFlow Lite (TFLite) for Microcontrollers. It is important to understand what the differences are between various frameworks and optimizations schemes.

The model conversion procedure is not particularly complicated, but it is one of the most important aspects of TinyML. Certain inference libraries may only be able to work with certain file formats, or may perform inadequately due to poorly optimized operations.

In this section, we will delve into some of the particularities of the model conversion process:

- ML Frameworks
    - TensorFlow vs. TensorFlow Lite vs. TensorFlow Lite Micro and others
    - Understand how inference frameworks differ from one another and why that's important to keep in mind
- Common Optimizations
    - Pruning, Quantization, Knowledge Distillation, Joint optimization, etc.
- Role of Optimizations
    - Why these optimizations are so important for TinyML

Motivation

The core runtime for TFLite Micro is only around 16 kB, several orders of magnitude smaller than that of TensorFlow. However, despite the fact that TFLite Micro is derived from TensorFlow, it cannot natively run TensorFlow models. In order for us to run our model on-device, we must go through a model conversion process.

Model conversion occurs when transferring between representations of a neural network model. For example, we could convert between a Pytorch and TensorFlow model, or between a Keras model and one cast in the ONNX format. Each of these formats has been built independently and optimized for a specific purpose. TFLite Micro was built with resource-constrained microcontrollers in mind, and is optimized for a tiny memory footprint and highly efficient computation, but is stripped of all unnecessary functionality. This includes functionalities that we might be very used to using, such as debugging!

Conversion

To convert our model from a TensorFlow format to TFLite, we must use the TFLite Converter Python API. This API converts our model into a FlatBuffer, which is an efficient serialization format for performance-critical applications. In this data format, computations are a lot more efficient, and it is optimized for statically typed languages (whereas JSON is often the format of choice for dynamically typed languages such as Javascript), making it a good choice for our resource-constrained microcontrollers which often utilize C, which is a statically typed language.

OK, now we have our TFLite model. However, the model conversion does not stop here. Because most microcontrollers do not natively contain filesystem support, we cannot copy and paste our file onto our device, instead we must embed the model in the microcontroller. This is typically done by converting it to a C array (using the xxd command) and then compiling it into our program.

Optimizations

Are we done now? Not if you want the job done properly. We still have to include model quantization and model pruning! **Model quantization** involves taking the weights of our trained neural network and changing them into a different numeric representation. Typically, this is from 16-bit floating point (FP16) or 32-bit floating point (FP32) to an 8-bit representation. Some applications have even gone so far as to develop binary neural networks - where the neural weights are all represented by a single bit. Using a simplified numeric representation can have a profound effect on the model size. **Model pruning** is a similar method for reducing the size, but instead of altering the numeric representation, simply cuts out weights that are not that important to us. For example, all weights that are somewhat close to zero, it might just set to zero and ignore them, somewhat similar to how sparse matrix calculations simply ignore zero values and just focus their attention on the non-zero values.