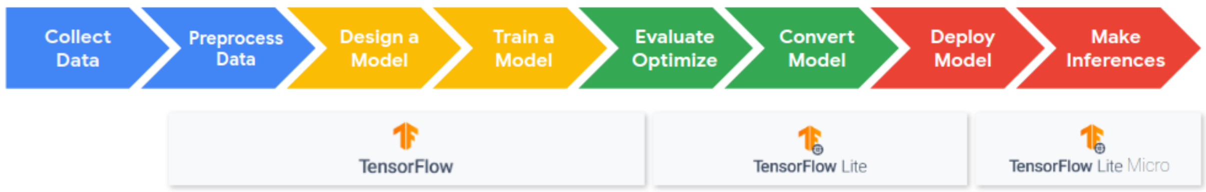
TinyML “Keyword Spotting” Workflow

In Course 2, you were introduced to the machine learning workflow. In this reading, we will expand on this, introducing the additional steps necessary to get our real-world deployment of the “Keyword Spotting” application running on our own microcontrollers. The reading gives you an overview of what is to come, while the videos that follow will help you understand the challenges of performing ML deployment effectively on-device.

The first stages of the workflow are essentially the same as a traditional machine learning workflow: we collect and process our data, and then design and train a model for a specific machine learning task using the TensorFlow framework. The model will likely require iterative design to meet performance goals. Following this, we move to TFLite to optimize our model (e.g., quantizing model weights into a suitable format) and then convert it into a suitable format, often a hexdump file (i.e., a binary file with its contents represented by hexadecimal values). Finally, we flash this hexdump file to the flash memory of the microcontroller, which is then able to perform inferences on the embedded device. The rest of this article will expand upon these stages with reference to our hello world application. The workflow is illustrated below.



Step 1: Data collection

As we explored in Course 2, collecting the dataset can often be the most challenging part of a tinyML application. Datasets must be large and specific in order to be applicable. For example, in the Keyword Spotting application we need a dataset **aligned** to individual words that includes thousands of examples which are representative of real world audio (e.g., including background noise). For our Keyword Spotting application, we will start off using the Speech Commands dataset from Pete Warden. Later on, we will show you how to collect your own data to train a new model.

Step 2 - Data Preprocessing

For efficient inference we may need to **extract features** from the input signal to use for classification with our NN. First we’ll need to take any analog input signals collected from our sensors and convert them into digital signals. Then we may need to apply additional transformations. For example, in the Keyword Spotting application we will convert the audio signals into images through the use of spectrograms, all on-device!

Step 3 - Model Design

In order to deploy a model onto our microcontroller we need it to be **very small**. We explored the tradeoffs of such models and just how small they need to be (hint: it's tiny)! Since our Keyword Spotting application uses the "tinyConv" model, we should not need to worry about the size of the model, but parsimony should always be exercised when designing these models for the sake of efficiency - always have in mind, what is the smallest number of parameters that can be used to achieve a highly accurate model?

Step 4 - Training

When we build our own dataset, we will train our model using standard training techniques explored in Courses 1 and 2. Remember, training requires highly accurate computation (close to machine precision) since we are often working with small gradients during the training process, which largely precludes us from performing training on our embedded device. Because of this, we train our models using TensorFlow, as done previously.

Step 5 - Evaluation

We then test the final accuracy. In most industrial settings, initial testing is often poor and requires iterative design in order to match performance goals and the accuracy required by end users. These performance goals are application-dependent, and range from inference speed to model accuracy. For our Keyword Spotting application, we are only hoping to see a flashing light as our output, so there are no real performance goals beyond having a reasonable accuracy to detect a keyword of interest properly.

Step 6 - Conversion

In this stage, we take our functioning model and aim to compress and port it to our microcontroller device. First, we quantize our model weights and activations appropriately so that they can be represented by 8-bit or fixed-point arithmetic, and subsequently convert the model to a hexdump file. This stage is relatively simple to implement but is an important step to get right to ensure our model continues to function correctly when deployed on the microcontroller. The Keyword Spotting application needs no special treatment in this stage, all application models must go through this.

Step 7 - Write Deployment Code

Note that once we have our model ready, we also need all of the preprocessing code in C++ to be able to deploy our model to the Arduino with TFMicro. This involves the TFMicro runtime, as well as code to tell the Arduino how to respond to the information from the machine learning model. For our Keyword Spotting application, we have to tell it to turn on our LED and to report results over Serial!

Step 8 - Deploy the Full Application

The last thing to do is deploy the model. This stage involves loading the binary file into the application and using the Arduino IDE to flash the file to the microcontroller. Voila!