# Train and Deploy Your Custom Dataset KWS Model

In this reading we are going to walk through the steps for training a custom KWS model based on the dataset you just collected and then deploy that model onto your Arduino (including the necessary code changes in the Arduino IDE).

Screencast of Brian walking through this section goes here on the edX course

## Training with your Custom Dataset

The first thing you'll need to do is to upload your custom dataset into Colab and train a KWS model. Then in that Colab we'll need to convert that model first into a quantized `.tflite` file and then into a `.cc` file for use with the Arduino IDE.

As with the pretrained model and favorite keyword model we will be using the resulting `.cc` file, so make sure to download it or leave the tab open with the printout!
https://colab.research.google.com/github/tinyMLx/colabs/blob/master/4-6-8-CustomDatasetKWSModel.ipynb

## Updating the Arduino Code

1. Again open the micro_speech.ino sketch, which you can find via the File drop-down menu. Navigate, as follows: `File → Examples → Arduino_TensorFlowLite → micro_speech`.
2. As before, navigate to the `micro_features_model.cpp` file and update the model.

   a. Copy the binary model file contents from the KWS_custom.cc file into the micro_features_model.cpp file. As always make sure to only copy the binary data inside the { } as the variable type is different in the downloaded or printed KWS_custom.cc file.

   b. Next scroll all the way down to the bottom of the file and replace the model length. Again note that the .cpp file needs the variable to be of type const int while the .cc file will show unsigned int. Our suggestion again is to simply copy the numerical value.

3. Next save your changes. Again, we suggest that you make a folder called e.g., TinyML inside of your Arduino folder. You can find your main Arduino folder either inside of your Documents folder or in your Home folder, and save it in that folder with a descriptive name like micro_speech_edx_custom_keywords. That said, you can save it wherever you like with whatever name you want!

4. The two other things we need to change are the count and list of keywords in the model settings files and the output response file. As we do that in the rest of this document, we are going to assume that you used our model with the keywords “brian,” “vijay,” “pete.” If you choose to use different keywords, make sure that you update the following steps accordingly.

5. Navigate to the micro_features_micro_model_settings.h file and scroll down to line 40 and change the value of the kCategoryCount variable to be two more than the number of keywords you selected (one extra for silence and one extra for unknown). For example if you had 3 keywords (as we do in our example model) we need to update the variable to be 3 + 2 = 5 as shown below:

```
constexpr int kCategoryCount = 5;
```

6. Then, navigate to the micro_features_micro_model_settings.cpp file. You’ll find that it includes the .h file and otherwise has a single array. Update the values of that array to match your keywords. **Make sure to leave “silence” and “unknown” in the list as well, and make sure to list your key words in the order that you listed them on the training script**. So in the case of our example model we would update the array to be:

```
const char* kCategoryLabels[kCategoryCount] = {
    "silence",
    "unknown",
    "brian",
    "vijay",
    "pete",
};
```

7. Finally, navigate to the `arduino_command_responder.cpp` file. Scroll down to line 54. There you will see a series of `if` statements which control how the arduino responds to each command. They are each structured to look at the value of the `found_command` variable. This variable will be the string of the keyword (or "silence" or "unknown") you entered into the `kCategoryLabels` array in the previous step. So by comparing the `[0]` value in that string, the if statements are simply looking at the first letter! In each `if` statement you'll also see a `digitalWrite` to either the `LEDG`, `LEDR`, or `LEDB` variable which is used to turn on the appropriate color LED. So if we wanted to adapt this to our three keywords and, for example, only turn on the single colors for our keywords and turn all of the colors on for unknown, which will come out look white, we could update those three `if` statements to the following.

   ```
   // Red for Vijay
   if(found_command[0] == 'v') {
      last_command_time = current_time;
      digitalWrite(LEDR, LOW);
   }

   // Green for Pete
   if(found_command[0] == 'p') {
      last_command_time = current_time;
      digitalWrite(LEDG, LOW);
   }

   // Blue for Brian -- note here that you do not need to index
   //                   into the first letter only, just a unique
   //                   letter combination in the keyword! That
   //                   said make sure you do not index beyond the
   //                   end of ANY keyword or you will get an error!
   if(found_command[0] == 'b' && found_command[1] == 'r') {
      last_command_time = current_time;
      digitalWrite(LEDB, LOW);
   }

   // All three for unknown
   if(found_command[0] == 'u') {
      last_command_time = current_time;
      digitalWrite(LEDR, LOW);
      digitalWrite(LEDG, LOW);
      digitalWrite(LEDB, LOW);
   }
   ```

   **Remember, if you used different keywords make sure to update the `if` statements accordingly.** And that's it! You're Arduino code should be all set to handle your model!

## Deploying the New Model

1. Use a USB cable to connect the Arduino Nano 33 BLE Sense to your machine. You should see the green LED power indicator come on when the board first receives power.

2. As always, use the Tools drop down menu to select appropriate Port and Board.

    a. Select the Arduino Nano 33 BLE as the board by going to `Tools → Board: <Current Board Name> → Arduino Mbed OS Boards (nRF52840) → Arduino Nano 33 BLE.`

    b. Then select the USB Port associated with your board. This will appear differently on Windows, macOS, Linux but will likely indicate 'Arduino Nano 33 BLE" in parenthesis. You can select this by going to `Tools → Port: <Current Port (Board on Port)> → <TBD Based on OS> (Arduino Nano 33 BLE).` Where <TBD Based on OS> is most likely to come from the list below where <#> indicates some integer number

        i. Windows → `COM<#>`
        ii. macOS → `/dev/cu.usbmodem<#>`
        iii. Linux → `ttyUSB<#>` or `ttyACM<#>`

3. Use the rightward arrow next to the 'upload' / flash the code. You'll know the upload is complete when you red text in the console at the bottom of the IDE that shows 100% upload of the code and a statement that says something like "Done in <#.#> seconds."

    If you receive an error you will see an orange error bar appear and a red error message in the console (as shown below). Don't worry -- there are many common reasons this may have occurred. To help you debug please check out our FAQ appendix with answers to the most common errors!

4. You should now have a working Keyword Spotting model trained on your own custom dataset deployed to your Arduino! To test it out, open the Serial Monitor. You should start to see the result of the model begin to display in the Serial Monitor and the LED's light up according to how you instructed them in the previous section!