Deploying a KWS Model with Your Favorite Keyword(s)

In this reading we are going to first generate a binary file representing the KWS model with our favorite keywords from Course 2 and then deploy that model to our Arduino using the Arduino IDE (which will include some code changes).

Converting the TFLite Model File into a Binary Array

Again we are going to first convert the `.tflite` file into a `.cc` file in Colab for use with the Arduino IDE. This time let's use a model with different keywords (and a different number of keywords), so we can explore more of the changes you will need to make to deploy custom models. We've provided you with one option for a model in the Colab below. That said, we also invite you to use your favorite KWS model you created in Course 2 if you happen to still have the `.tflite` file! If you'd like to go back and train a new `.tflite` file you can find the training Colab at this link.

As with the pretrained model we will be using the resulting `.cc` file, so make sure to download it or leave the tab open with the printout!

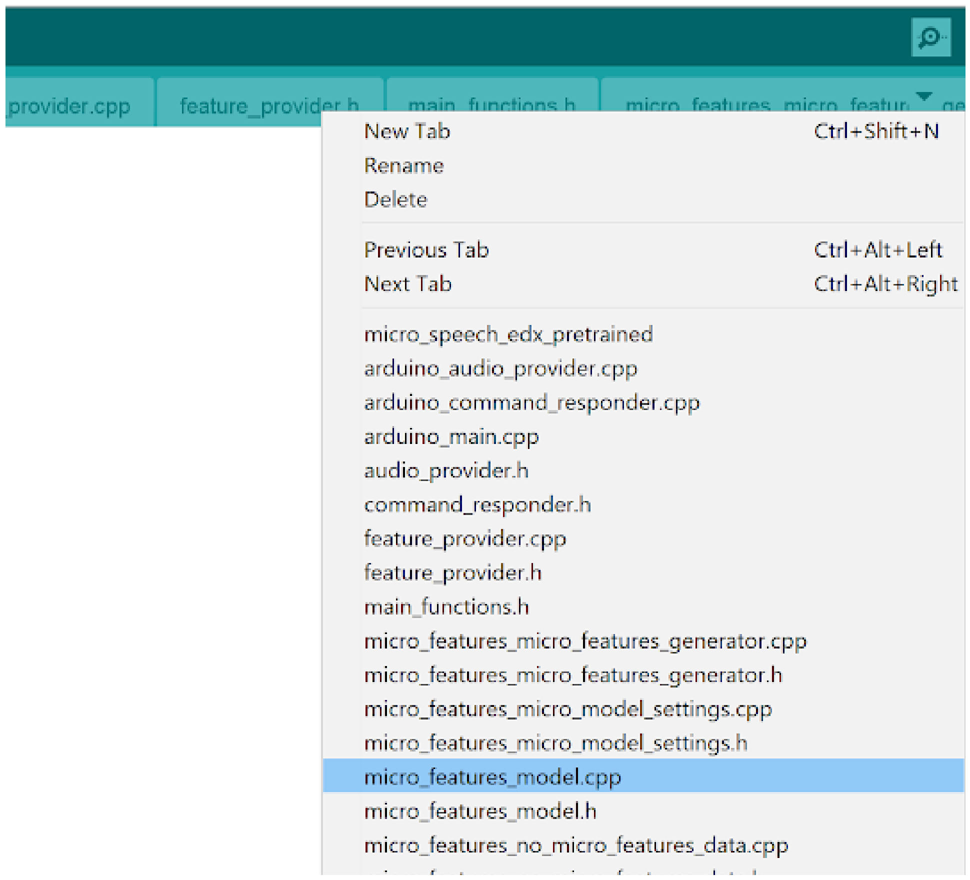
https://colab.research.google.com/github/tinyMLx/colabs/blob/master/4-5-18-KWS-FavoriteKeywords.ipynb

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

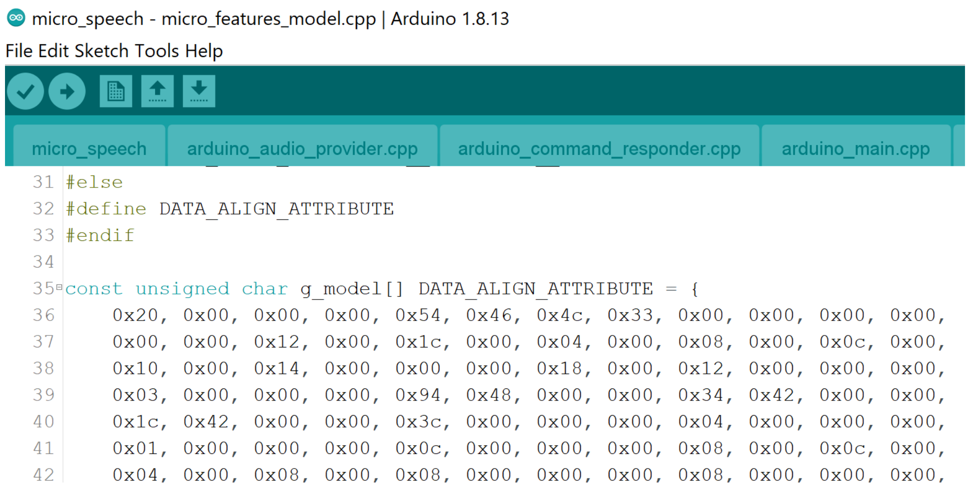
Updating the Arduino Code

1. Open the micro_speech.ino sketch, which you can find via the File drop-down menu. Navigate, as follows: `File → Examples → Harvard_TinyMLx → micro_speech`.

2. As before, navigate to the `micro_features_model.cpp` file and update the model. You can find that file by selecting it from tabs across the top of the Arduino IDE. If that file is not visible you can navigate to it (or other additional files) by clicking on the downward facing triangle at the end of the tabs which will open up a dropdown showing all of the files.

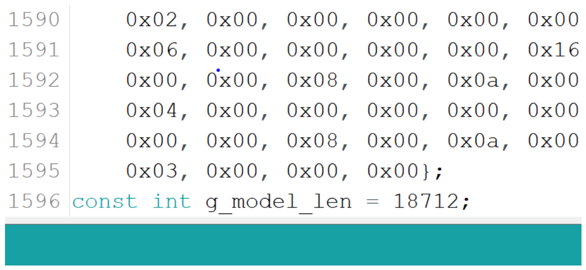

provider.cpp
feature_provider.h
main_functions.h
micro_features_micro_featur
New Tab Ctrl+Shift+N
Rename
Delete
Previous Tab Ctrl+Alt+Left
Next Tab Ctrl+Alt+Right
micro_speech_edx_pretrained
arduino_audio_provider.cpp
arduino_command_responder.cpp
arduino_main.cpp
audio_provider.h
command_responder.h
feature_provider.cpp
feature_provider.h
main_functions.h
micro_features_micro_features_generator.cpp
micro_features_micro_features_generator.h
micro_features_micro_model_settings.cpp
micro_features_micro_model_settings.h
micro_features_model.cpp
micro_features_model.h
micro_features_no_micro_features_data.cpp

   a. Copy the binary model file contents from the `KWS_favorite.cc` file into the `micro_features_model.cpp` file. **Make sure to only copy the binary data inside the `{ }` as the variable type is different in the downloaded or printed `model.cc` file** (it is of type `unsigned char` in the `.cc` file but it needs to be of type `const unsigned char` with the `DATA_ALIGN_ATTRIBUTE` in the `.cpp` file). If you lost your `KWS_favorite.cc` file, don't worry, you can use the staff's copy!


micro_speech - micro_features_model.cpp | Arduino 1.8.13
File Edit Sketch Tools Help
micro_speech
arduino_audio_provider.cpp
arduino_command_responder.cpp
arduino_main.cpp

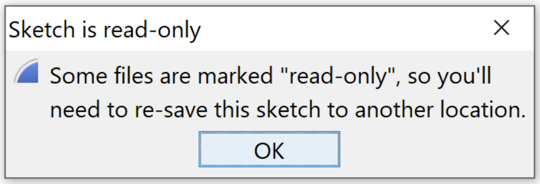
```
#else
#define DATA_ALIGN_ATTRIBUTE
#endif

const unsigned char g_model[] DATA_ALIGN_ATTRIBUTE = {
    0x20, 0x00, 0x00, 0x00, 0x54, 0x46, 0x4c, 0x33, 0x00, 0x00, 0x00, 0x00,
    0x00, 0x00, 0x12, 0x00, 0x1c, 0x00, 0x04, 0x00, 0x08, 0x00, 0x0c, 0x00,
    0x10, 0x00, 0x14, 0x00, 0x00, 0x00, 0x18, 0x00, 0x12, 0x00, 0x00, 0x00,
    0x03, 0x00, 0x00, 0x00, 0x94, 0x48, 0x00, 0x00, 0x34, 0x42, 0x00, 0x00,
    0x1c, 0x42, 0x00, 0x00, 0x3c, 0x00, 0x00, 0x00, 0x04, 0x00, 0x00, 0x00,
    0x01, 0x00, 0x00, 0x00, 0x0c, 0x00, 0x00, 0x00, 0x08, 0x00, 0x0c, 0x00,
    0x04, 0x00, 0x08, 0x00, 0x08, 0x00, 0x00, 0x00, 0x08, 0x00, 0x00, 0x00,
```

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.



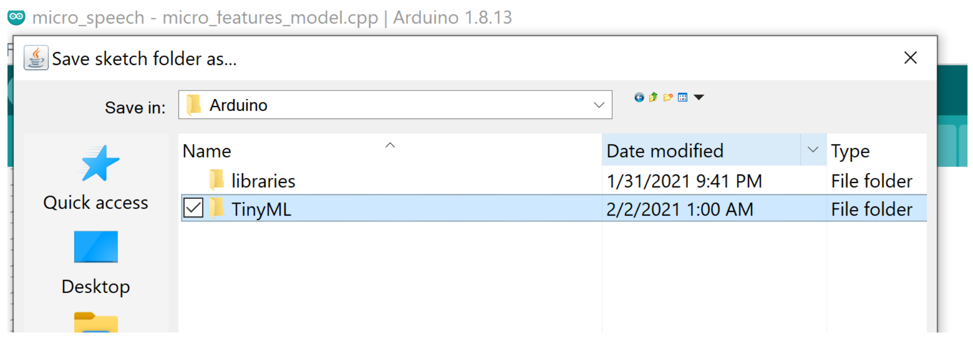
```
    0x02, 0x00, 0x00, 0x00, 0x00, 0x00
    0x06, 0x00, 0x00, 0x00, 0x00, 0x16
    0x00, 0x00, 0x08, 0x00, 0x0a, 0x00
    0x04, 0x00, 0x00, 0x00, 0x00, 0x00
    0x00, 0x00, 0x08, 0x00, 0x0a, 0x00
    0x03, 0x00, 0x00, 0x00};
const int g_model_len = 18712;
```

3. Next save your changes. You will most likely see a popup as shown below asking you to save a copy of the example as all examples are treated as “read-only” by default.


Sketch is read-only

Some files are marked "read-only", so you'll need to re-save this sketch to another location.

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_favorite`. That said, you can save it wherever you like with whatever name you want!



Name	Date modified	Type
libraries	1/31/2021 9:41 PM	File folder
TinyML	2/2/2021 1:00 AM	File folder

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 “up,down,go.” If you chose 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",
    "up",
    "down",
    "go",
};
```

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 up -- 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[1] == 'p') {
   last_command_time = current_time;
   digitalWrite(LEDR, LOW);
}

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

// Blue for go
if(found_command[0] == 'g') {
   last_command_time = current_time;
   digitalWrite(LEDB, LOW);
}

// All three for unknown
if(found_command[1] == 'n') {
   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 for either your favorite keywords from Course 2 or our model for "up,down,go" 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!