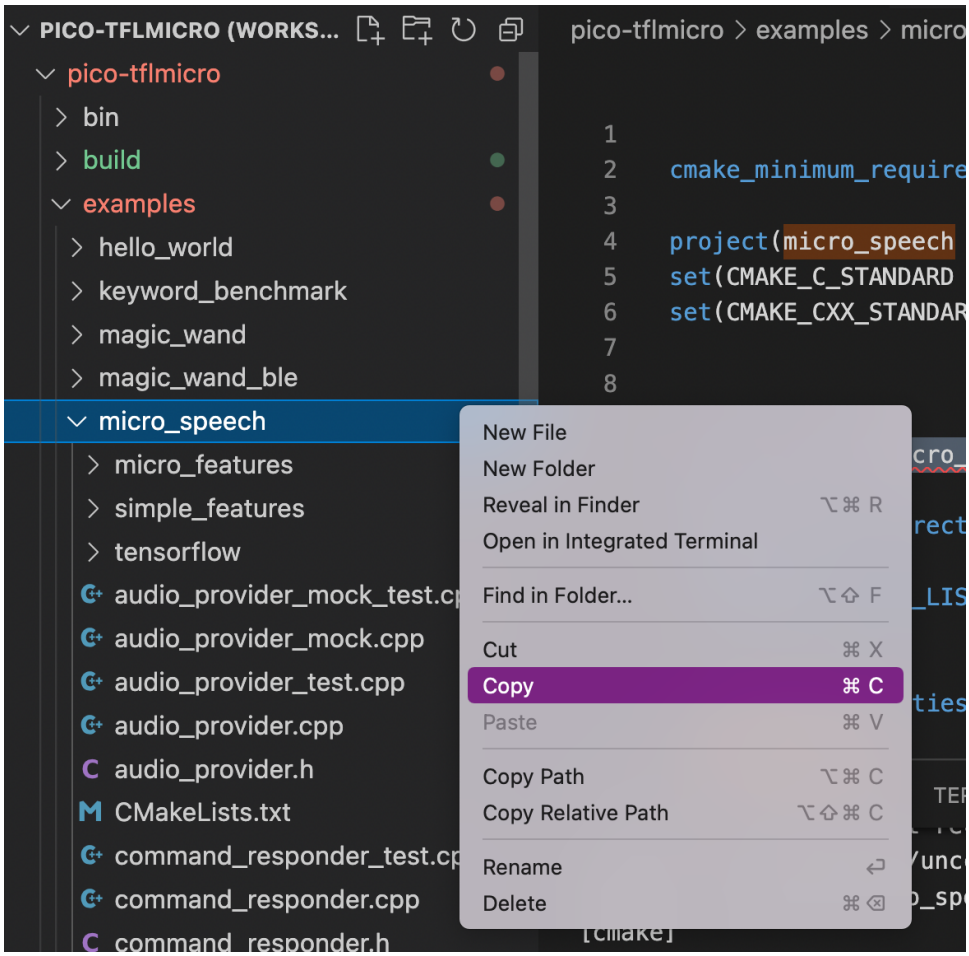
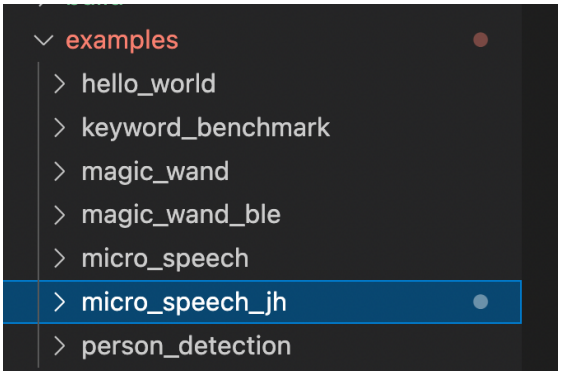
This work-through assumes that you already have the example repo (https://github.com/ArduCAM/pico-tflmicro) downloaded and you can build the unmodified micro_speech example. This document will go through how to copy it and modify it to run your specific model. If you don't want to build a duplicate, and you will just modify the original project in place, you can skip to step 8, but having a known-good backup is sometimes helpful.

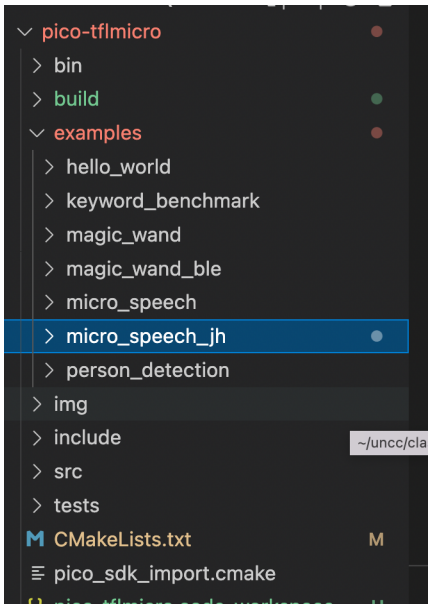
1. Right click on the micro_speech project in VSCode and 'copy'.

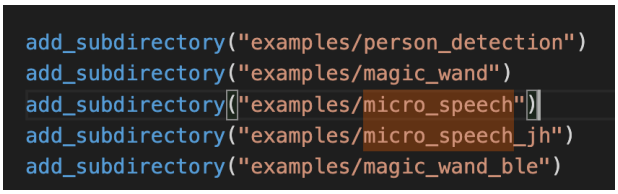


2. Now Right click and paste in the same area. You should have a new project 'micro_speech copy' at the same level as micro_speech. Rename it (also accessible via right-click) to whatever you want.

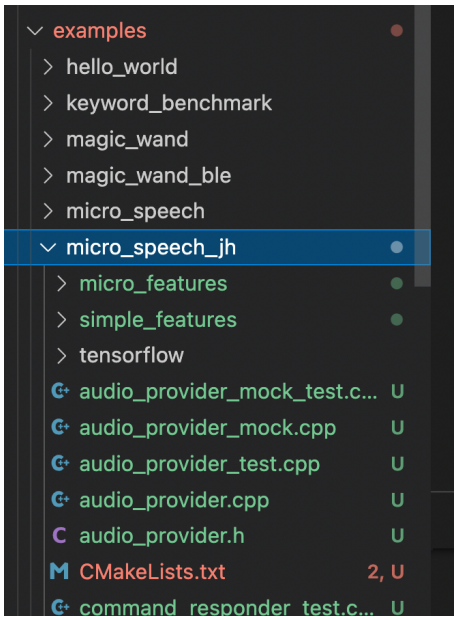


3. Modify examples/CMakeLists.txt. Where there is a line `add_subdirectory("examples/micro_speech")`, add a copy of that line with the name of your new project.



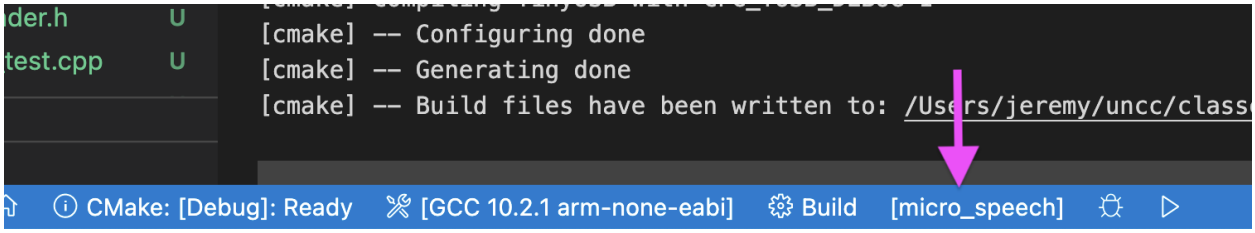


4. Under your new project there is another CMakeLists.txt file. Edit that as well. Replace every instance of “micro_speech” with your new project name. There should be 9 instances.

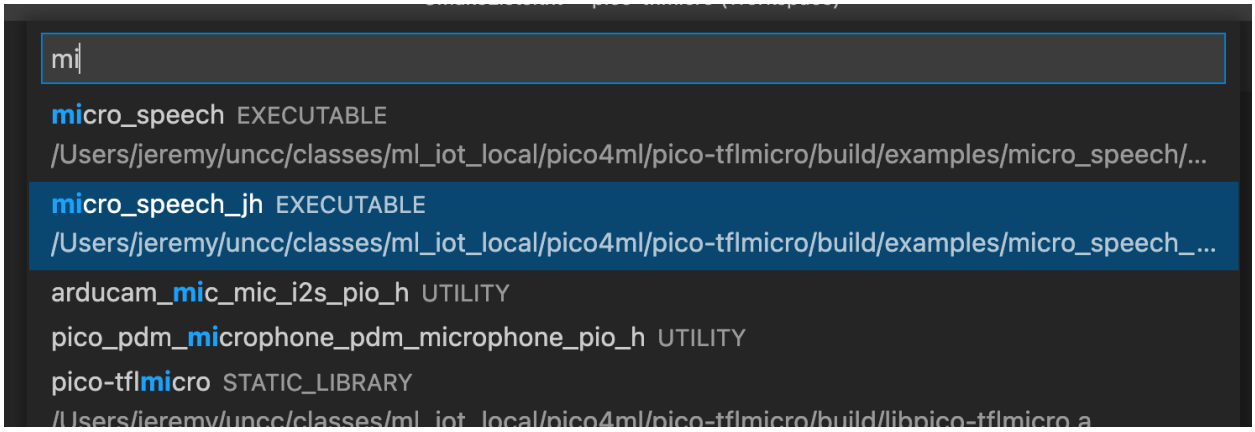


5. From the directory ‘.../pico-tflmicro/build’, run ‘`cmake ..`’ It should complete with no errors.

6. At the bottom of VSCode there is an indication of what project you are currently building.



Click that and at the top of VSCode a list of available build targets should show up. You may have to type a few letters into the search bar to find yours, but select your new project.



7. Click on 'Build' at the bottom (just to the left of [micro_speech] in the picture above). It should run and end with the following text:

```
[build] [100%] Built target micro_speech_jh
[build] Build finished with exit code 0
```

8. Now you have a duplicate of the original project that you can modify. If you chose to just modify the original project in place, you can join us here.
9. Convert your TFlite model into a C array of bytes:

```
!xxd -i model.tflite > model.c
```

Edit the file micro_features/model.cpp to include your model data. You want to keep the declaration and the types the same. The original file has some preprocessor code (`#ifdef` etc.) followed by:

```
const unsigned char g_model[] DATA_ALIGN_ATTRIBUTE = {
  0x20, 0x00, 0x00, 0x00, 0x54, 0x46, 0x4c, . . .
// several hundred lines of hex
0x03, 0x00, 0x00, 0x00};
const int g_model_len = 18712;
```

Original Leaving the declarations alone, replace the hex bytes in the array and the length value with those you just created with the xxd command. The **purple bolded text** above is what should be replaced.

10. In the file `micro_features/micro_model_settings.cpp`, change the array kCategoryLabels so that the labels correspond to the labels you used in training. You probably just need to replace "yes" and "no" with your chosen words. **Order matters**

here, because the numeric output of the model will index into this array to determine what's printed on the screen.

11. In the file `micro_features/micro_features_generator.cpp`, make sure that the feature extraction settings correspond to what you used in training.
    a. Some of the settings (the `kFeatureXXX` items) refer to variables defined in micro_model_settings.h.

Python/TensorFlow Element	**C Element**
window_step_ms	kFeatureSliceStrideMs
window_size_ms	kFeatureSliceDurationMs
num_filters OR model.input.shape[2]	kFeatureSliceSize
floor(((wave_length_ms - window_size_ms) / window_step_ms) + 1) OR model.input.shape[1]	kFeatureSliceCount
Next power of 2 above (window_size_ms * fsamp / 1000)	kMaxAudioSampleSize

12. Modify the OpResolver. This is done in main_functions.cpp in the section starting with this line:
    `static tflite::MicroMutableOpResolver<4> micro_op_resolver(error_reporter);`
    You have two options.
    a. Specify the layers you need **and the number of layers** (in the angle brackets, <4> in the original). The available layer resolvers are declared in `src/tensorflow/lite/micro/micro_mutable_op_resolver.h, but the most common ones are AddDepthwiseConv2D(), AddConv2D, AddFullyConnected, AddMaxPool2D, AddReshape, AddSoftmax.`
    There are already four blocks like this:

```
if (micro_op_resolver.AddDepthwiseConv2D() != kTfLiteOk) {
    return;
  }
```

    Add additional blocks with the specific function you need. Change the number in the angle brackets to reflect the number of different op resolvers you've added, so if you have the original for plus two new ones, your constructor line should look like this:

```
static tflite::MicroMutableOpResolver<6> micro_op_resolver(error_reporter);
```

13. Modify the code block including
    `if (found_command == "yes") {`
    to compare to your chosen keywords.

Debugging

```
Feature generation failed
Requested feature_data_ size 537140992 doesn't match 768
```

1. When I did not have the correct OpResolvers added, I saw the following error messages. The ones starting with ‘JHDBG’ are from printf() statements that I added; the others are pre-existing error messages.

   ```
   JHDBG: created model
   Didn't find op for builtin opcode 'CONV_2D' version '3'

   Failed to get registration from op code CONV_2D

   Failed starting model allocation.

   AllocateTensors() failed
   JHDBG: Just after setup
   ```

2. When I did not have kTensorArenaSize set to a sufficiently large value, I get the error below. This is the scratch space needed to run the neural network. It is mostly driven by the sum of all the tensors that need to be held in memory simultaneously. That can be estimated by the largest sequential pair of layer output tensors, but there’s a little bit of overhead. You can work to get this exactly, or you can just keep increasing the number until it stops complaining.

   ```
   JHDBG: created model
   Arena size is too small for all buffers. Needed 16512 but only 5648 was available.
   AllocateTensors() failed
   JHDBG: Just after setup
   ```

3. Here’s another one. This error is thrown inside main_functions.cpp, because the parameters of the model input do not match.

   ```
   JHDBG: created model
   Bad input tensor parameters in model
   JHDBG: Just after setup
   ```

   Fix it by changing the input shape testing code, e.g.

   ```
   if ((model_input->dims->size != 4)
       || (model_input->dims->data[0] != 1)
       || (model_input->dims->data[1] != kFeatureSliceCount)
       || (model_input->dims->data[2] != kFeatureSliceSize)
       || (model_input->dims->data[3] != 1)
       || (model_input->type != kTfLiteInt8)) {
   ```

4. Her

```
if (is_new_command) {
```