EECE-4710 “IoT and TinyML”

TFL Micro “Micro Speech” Code Walkthrough

Cris Ababei

MARQUETTE UNIVERSITY

BE THE DIFFERENCE.

1

1

TensorFlow Lite Micro “Micro Speech” Model

Code Walkthrough!

micro_speech.ino

C:\Users\Cristinel Ababei\Documents\Arduino\libraries\Harvard_TinyMLx\examples\micro_speech

2

2

“Micro Speech” TFL-Micro Components

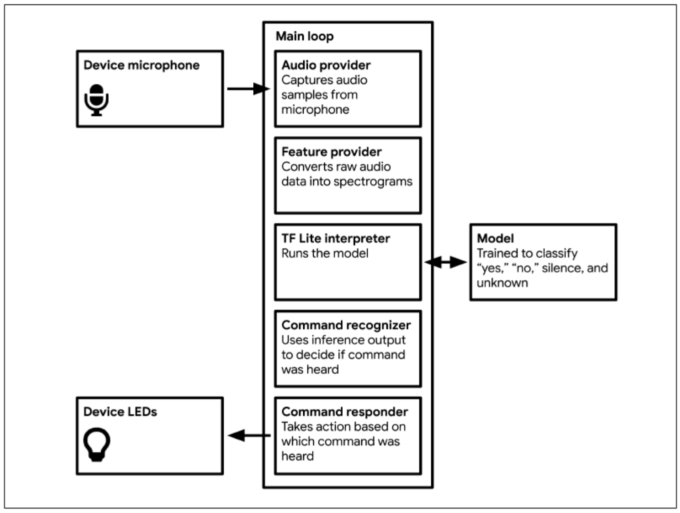
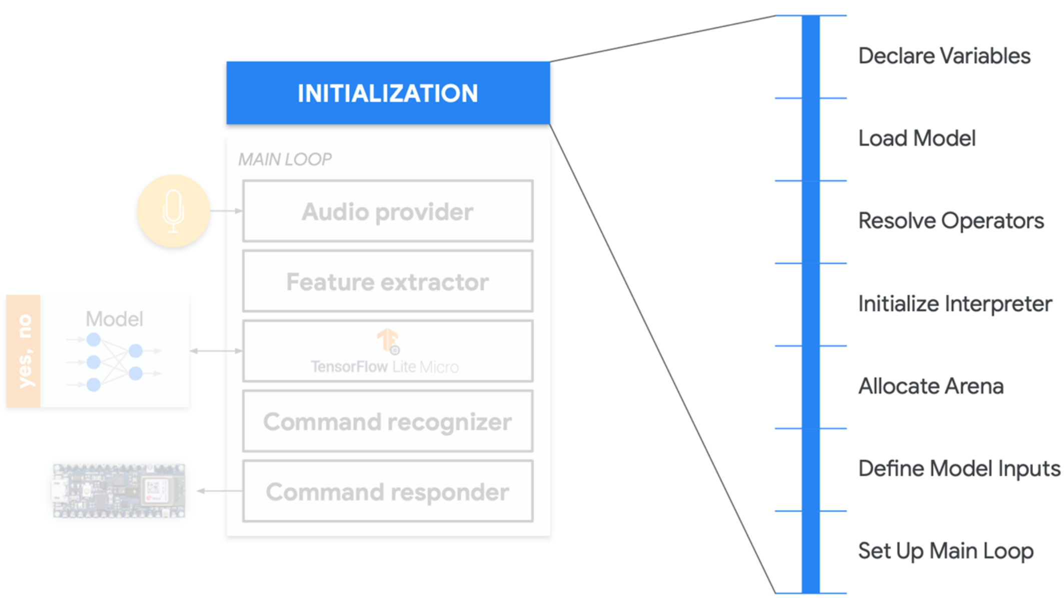


Figure 7-3. The components of our wake-word application

3

3



4


Declare Variables
Load Model
Resolve Operators
Initialize Interpreter
Allocate Arena
Define Model Inputs
Set Up Main Loop
// Globals, used for compatibility with Arduino-style sketches.
namespace {
tflite::ErrorReporter* error_reporter = nullptr;
const tflite::Model* model = nullptr;
tflite::MicroInterpreter* interpreter = nullptr;
TfLiteTensor* model_input = nullptr;
FeatureProvider* feature_provider = nullptr;
RecognizeCommands* recognizer = nullptr;
int32_t previous_time = 0;
// Create an area of memory to use for input, output, and intermediate arrays.
// The size of this will depend on the model you're using, and may need to be
// determined by experimentation.
constexpr int kTensorArenaSize = 10 * 1024;
uint8_t tensor_arena[kTensorArenaSize];
int8_t feature_buffer[kFeatureElementCount];
int8_t* model_input_buffer = nullptr;
} // namespace

5


Declare Variables
Load Model
Resolve Operators
Initialize Interpreter
Allocate Arena
Define Model Inputs
Set Up Main Loop
// Map the model into a usable data structure.
// This doesn't involve any copying or parsing,
// it's a very lightweight operation.
model = tflite::GetModel(g_model);
if (model->version() != TFLITE_SCHEMA_VERSION) {
TF_LITE_REPORT_ERROR(error_reporter,
"Model provided is schema version %d not equal to
supported version %d.", model->version(),
TFLITE_SCHEMA_VERSION);
return;
}

6

- Declare Variables
- Load Model
- **Resolve Operators**
- Initialize Interpreter
- Allocate Arena
- Define Model Inputs
- Set Up Main Loop

Load only the needed Ops

```
// Pull in only the operation implementations we need.
// This relies on a complete list of all the ops needed by this graph.
// An easier approach is to just use the AllOpsResolver, but this will
// incur some penalty in code space for op implementations that are not
// needed by this graph.
//
// tflite::AllOpsResolver resolver;
// NOLINTNEXTLINE(runtime-global-variables)
static tflite::MicroMutableOpResolver<4> micro_op_resolver(error_reporter);
if (micro_op_resolver.AddDepthwiseConv2D() != kTfLiteOk) {
  return;
}
if (micro_op_resolver.AddFullyConnected() != kTfLiteOk) {
  return;
}
if (micro_op_resolver.AddSoftmax() != kTfLiteOk) {
  return;
}
if (micro_op_resolver.AddReshape() != kTfLiteOk) {
  return;
}
```

Used if you have problem with memory

7

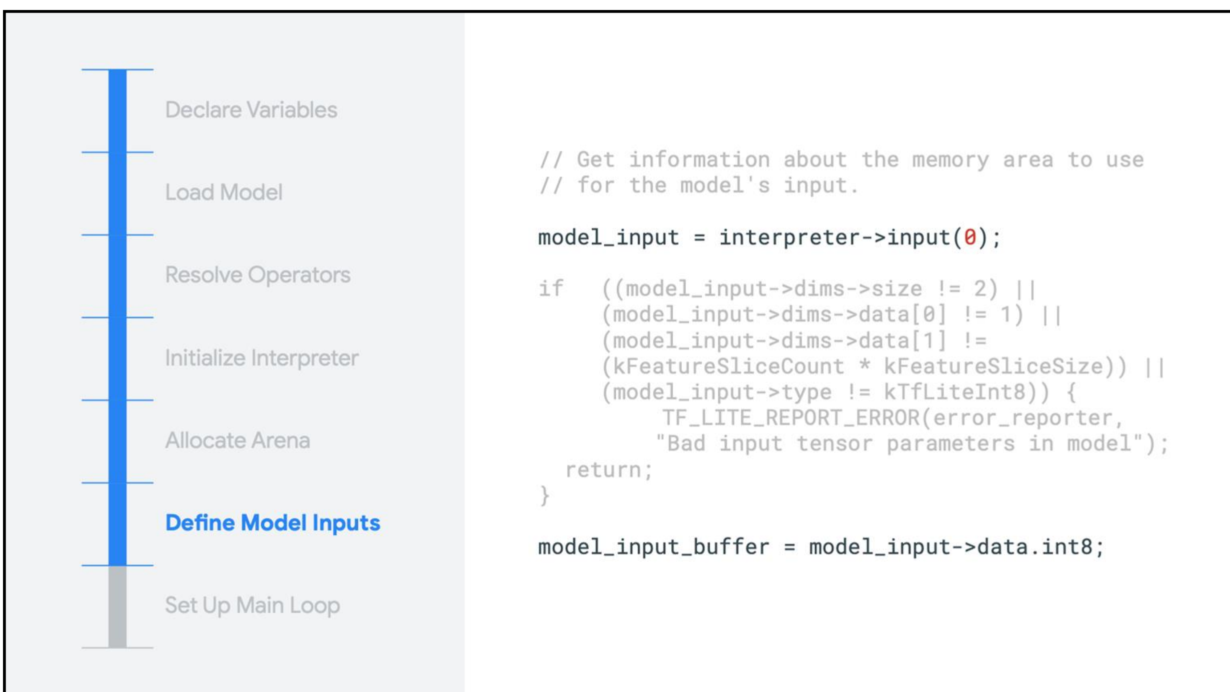
- Declare Variables
- Load Model
- Resolve Operators
- **Initialize Interpreter**
- Allocate Arena
- Define Model Inputs
- Set Up Main Loop

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

// Build an interpreter to run the model with.
static tflite::MicroInterpreter static_interpreter(
    model, micro_op_resolver, tensor_arena, kTensorArenaSize, error_reporter);
interpreter = &static_interpreter;

// Allocate memory from the tensor_arena for the model's tensors.
TfLiteStatus allocate_status = interpreter->AllocateTensors();
if (allocate_status != kTfLiteOk) {
  TF_LITE_REPORT_ERROR(error_reporter, "AllocateTensors() failed");
  return;
}
```

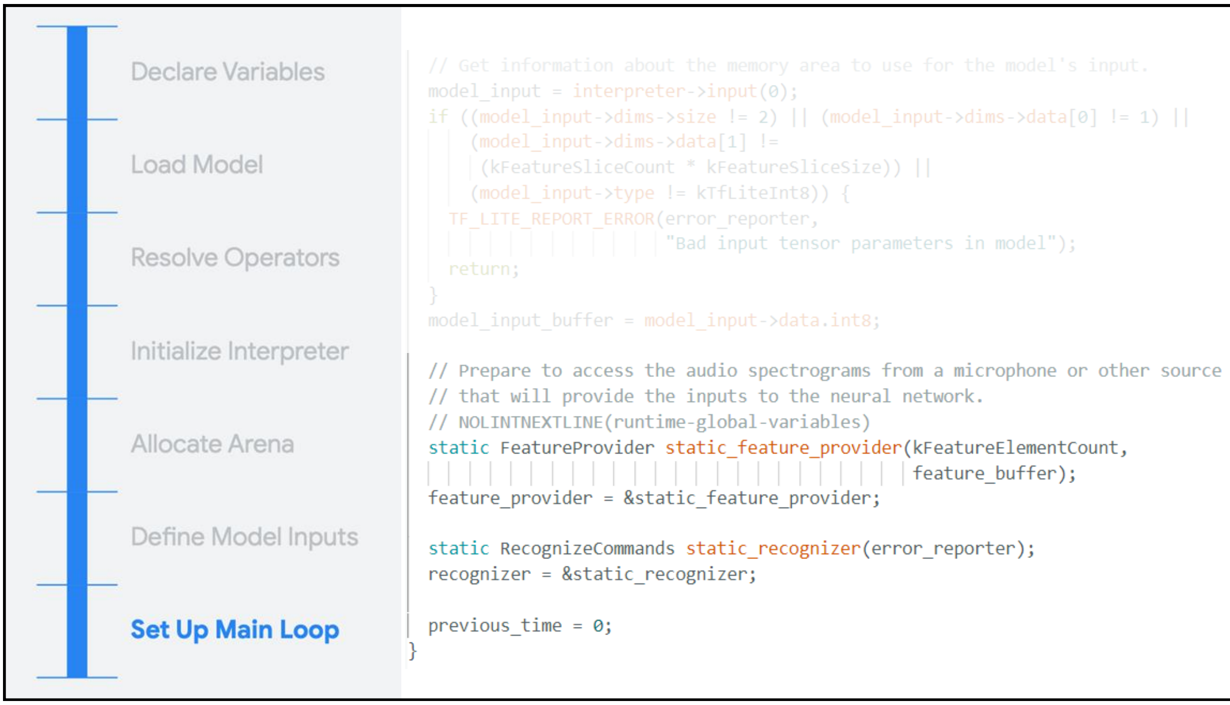
8


Declare Variables
Load Model
Resolve Operators
Initialize Interpreter
Allocate Arena
Define Model Inputs
Set Up Main Loop
// Build an interpreter to run the model with.
static tflite::MicroInterpreter static_interpreter(
model, micro_op_resolver, tensor_arena, kTensorArenaSize, error_reporter);
interpreter = &static_interpreter;
// Allocate memory from the tensor_arena for the model's tensors.
TfLiteStatus allocate_status = interpreter->AllocateTensors();
if (allocate_status != kTfLiteOk) {
TF_LITE_REPORT_ERROR(error_reporter, "AllocateTensors() failed");
return;
}
// Get information about the memory area to use for the model's input.
model_input = interpreter->input(0);
if ((model_input->dims->size != 2) || (model_input->dims->data[0] != 1) ||
(model_input->dims->data[1] !=
(kFeatureSliceCount * kFeatureSliceSize)) ||
(model_input->type != kTfLiteInt8)) {
TF_LITE_REPORT_ERROR(error_reporter,
"Bad input tensor parameters in model");
return;
}
model_input_buffer = model_input->data.int8;

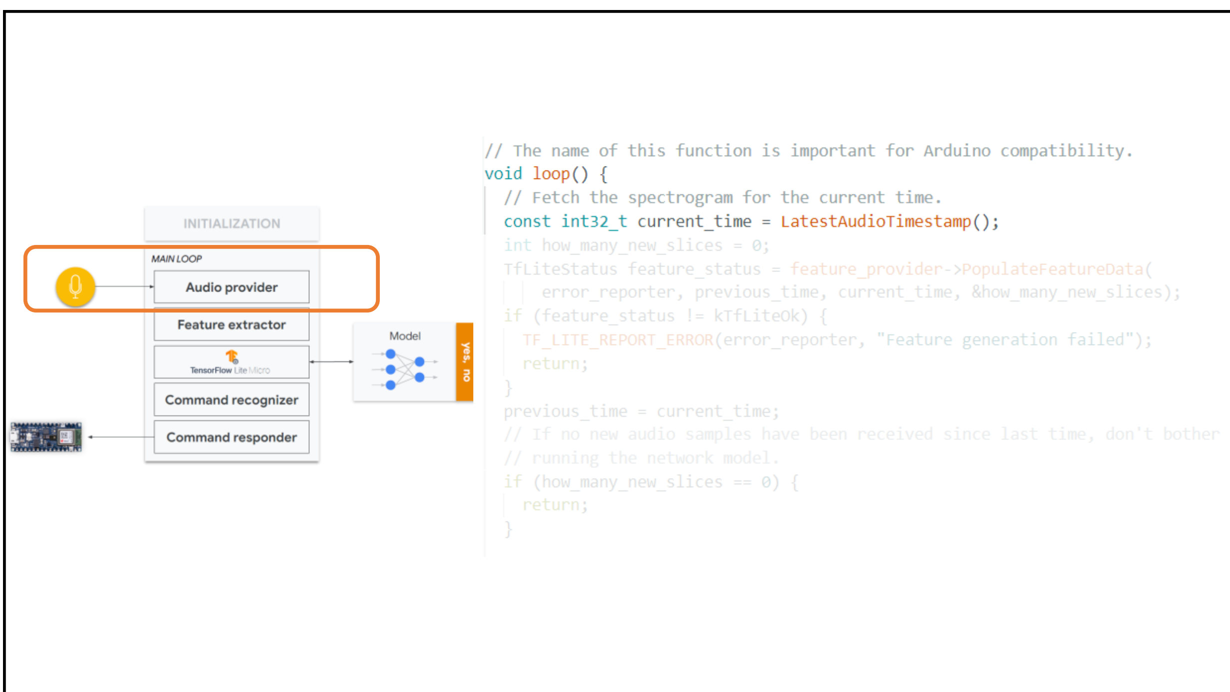
9


Declare Variables
Load Model
Resolve Operators
Initialize Interpreter
Allocate Arena
Define Model Inputs
Set Up Main Loop
// Get information about the memory area to use
// for the model's input.
model_input = interpreter->input(0);
if ((model_input->dims->size != 2) ||
(model_input->dims->data[0] != 1) ||
(model_input->dims->data[1] !=
(kFeatureSliceCount * kFeatureSliceSize)) ||
(model_input->type != kTfLiteInt8)) {
TF_LITE_REPORT_ERROR(error_reporter,
"Bad input tensor parameters in model");
return;
}
model_input_buffer = model_input->data.int8;

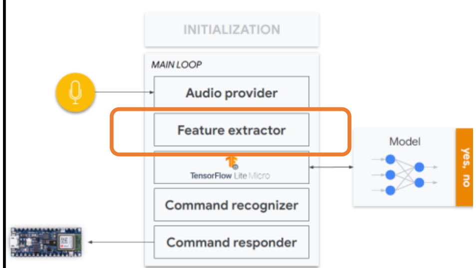
10


Declare Variables
Load Model
Resolve Operators
Initialize Interpreter
Allocate Arena
Define Model Inputs
Set Up Main Loop
// Get information about the memory area to use for the model's input.
model_input = interpreter->input(0);
if ((model_input->dims->size != 2) || (model_input->dims->data[0] != 1) ||
(model_input->dims->data[1] !=
(kFeatureSliceCount * kFeatureSliceSize)) ||
(model_input->type != kTfLiteInt8)) {
TF_LITE_REPORT_ERROR(error_reporter,
"Bad input tensor parameters in model");
return;
}
model_input_buffer = model_input->data.int8;
// Prepare to access the audio spectrograms from a microphone or other source
// that will provide the inputs to the neural network.
// NOLINTNEXTLINE(runtime-global-variables)
static FeatureProvider static_feature_provider(kFeatureElementCount,
feature_buffer);
feature_provider = &static_feature_provider;
static RecognizeCommands static_recognizer(error_reporter);
recognizer = &static_recognizer;
previous_time = 0;
}

11

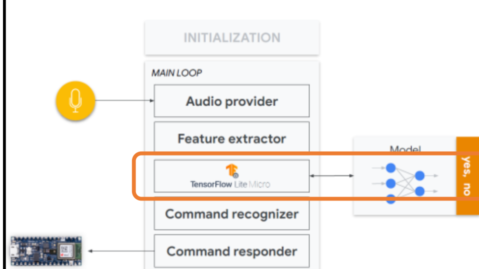

INITIALIZATION
MAIN LOOP
Audio provider
Feature extractor
TensorFlow Lite Micro
Command recognizer
Command responder
Model
yes, no
// The name of this function is important for Arduino compatibility.
void loop() {
// Fetch the spectrogram for the current time.
const int32_t current_time = LatestAudioTimestamp();
int how_many_new_slices = 0;
TfLiteStatus feature_status = feature_provider->PopulateFeatureData(
error_reporter, previous_time, current_time, &how_many_new_slices);
if (feature_status != kTfLiteOk) {
TF_LITE_REPORT_ERROR(error_reporter, "Feature generation failed");
return;
}
previous_time = current_time;
// If no new audio samples have been received since last time, don't bother
// running the network model.
if (how_many_new_slices == 0) {
return;
}

12



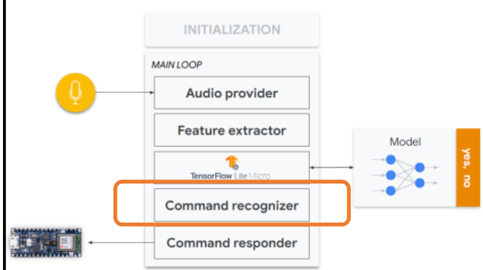

```
// The name of this function is important for Arduino compatibility.
void loop() {
  // Fetch the spectrogram for the current time.
  const int32_t current_time = LatestAudioTimestamp();
  int how_many_new_slices = 0;
  TfLiteStatus feature_status = feature_provider->PopulateFeatureData(
      error_reporter, previous_time, current_time, &how_many_new_slices);
  if (feature_status != kTfLiteOk) {
    TF_LITE_REPORT_ERROR(error_reporter, "Feature generation failed");
    return;
  }
  previous_time = current_time;
  // If no new audio samples have been received since last time, don't bother
  // running the network model.
  if (how_many_new_slices == 0) {
    return;
  }
```

13



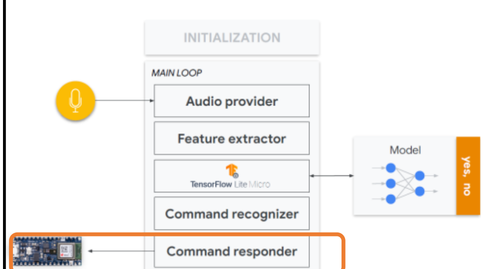

```
// Copy feature buffer to input tensor
for (int i = 0; i < kFeatureElementCount; i++) {
  model_input_buffer[i] = feature_buffer[i];
}

// Run the model on the spectrogram input and make sure it succeeds.
TfLiteStatus invoke_status = interpreter->Invoke();
if (invoke_status != kTfLiteOk) {
  TF_LITE_REPORT_ERROR(error_reporter, "Invoke failed");
  return;
}

// Obtain a pointer to the output tensor
TfLiteTensor* output = interpreter->output(0);
// Determine whether a command was recognized based on the output of inference
const char* found_command = nullptr;
uint8_t score = 0;
bool is_new_command = false;
```

14




```
// Obtain a pointer to the output tensor
TfLiteTensor* output = interpreter->output(0);
// Determine whether a command was recognized based on the output of inference
const char* found_command = nullptr;
uint8_t score = 0;
bool is_new_command = false;
TfLiteStatus process_status = recognizer->ProcessLatestResults(
    output, current_time, &found_command, &score, &is_new_command);
if (process_status != kTfLiteOk) {
  TF_LITE_REPORT_ERROR(error_reporter,
                       "RecognizeCommands::ProcessLatestResults() failed");
  return;
}
// Do something based on the recognized command. The default implementation
// just prints to the error console, but you should replace this with your
// own function for a real application.
RespondToCommand(error_reporter, current_time, found_command, score,
                 is_new_command);
}
```

15




```
// Obtain a pointer to the output tensor
TfLiteTensor* output = interpreter->output(0);
// Determine whether a command was recognized based on the output of inference
const char* found_command = nullptr;
uint8_t score = 0;
bool is_new_command = false;
TfLiteStatus process_status = recognizer->ProcessLatestResults(
    output, current_time, &found_command, &score, &is_new_command);
if (process_status != kTfLiteOk) {
  TF_LITE_REPORT_ERROR(error_reporter,
                       "RecognizeCommands::ProcessLatestResults() failed");
  return;
}
// Do something based on the recognized command. The default implementation
// just prints to the error console, but you should replace this with your
// own function for a real application.
RespondToCommand(error_reporter, current_time, found_command, score,
                 is_new_command);
}
```

16

Credits

- A previous edition of this course was developed in collaboration with Dr. Susan C. Schneider of Marquette University.
- We are very grateful and thank all the following professors, researchers, and practitioners for jump-starting courses on TinyML and for sharing their teaching materials:
- Prof. Marcelo Rovai - TinyML - Machine Learning for Embedding Devices, UNIFEI
  - https://github.com/Mjrovai/UNIFEI-IESTI01-TinyML-2022.1
- Prof. Vijay Janapa Reddi - CS249r: Tiny Machine Learning, Applied Machine Learning on Embedded IoT Devices, Harvard
  - https://sites.google.com/g.harvard.edu/tinyml/home
- Prof. Rahul Mangharam – ESE3600: Tiny Machine Learning, Univ. of Pennsylvania
  - https://tinyml.seas.upenn.edu/#
- Prof. Brian Plancher - Harvard CS249r: Tiny Machine Learning (TinyML), Barnard College, Columbia University
  - https://a2r-lab.org/courses/cs249r_tinyml/

17

17

References

- Additional references from where information and other teaching materials were gathered include:
- Applications & Deploy textbook: “TinyML” by Pete Warden, Daniel Situnayake
  - https://www.oreilly.com/library/view/tinyml/9781492052036/
- Deploy textbook “TinyML Cookbook” by Gian Marco Iodice
  - https://github.com/PacktPublishing/TinyML-Cookbook
- Jason Brownlee
  - https://machinelearningmastery.com/
- TinyMLedu
  - https://tinyml.seas.harvard.edu/
- Professional Certificate in Tiny Machine Learning (TinyML) – edX/Harvard
  - https://www.edx.org/professional-certificate/harvardx-tiny-machine-learning
- Introduction to Embedded Machine Learning - Coursera/Edge Impulse
  - https://www.coursera.org/learn/introduction-to-embedded-machine-learning
- Computer Vision with Embedded Machine Learning - Coursera/Edge Impulse
  - https://www.coursera.org/learn/computer-vision-with-embedded-machine-learning

18

18