


Deploying the Pretrained Person Detection Model

In this reading you will deploy the pretrained person detection model onto your Arduino and walk through a little bit of the code to better understand how it works.



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

Deploying the Pretrained Person Detection 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. Open the person_detection.ino sketch, which you can find via the File drop-down menu. Navigate, as follows: [File](#) → [Examples](#) → [Harvard_TinyMLx](#) → [person detection](#). Our library version of the example is very similar to the one you'll find in the TensorFlow Lite for Microcontrollers library, but includes a couple of important tweaks to work better with the TinyML Shield and both the OV7675 and OV7670 so make sure you open the example out of our library.
3. Use the Tools drop down menu to select the appropriate Port and Board. This is important as it is telling the IDE which board files to use and on which serial connection it should send the code. In some cases, this may happen automatically, but if not, you'll want to select:
 - 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](#). Note that on different operating systems the exact

name of the board may vary but/and it should include the word Nano at a minimum. If you do not see that as an option then please go back to Setting up the Software and make sure you have installed the necessary board files.

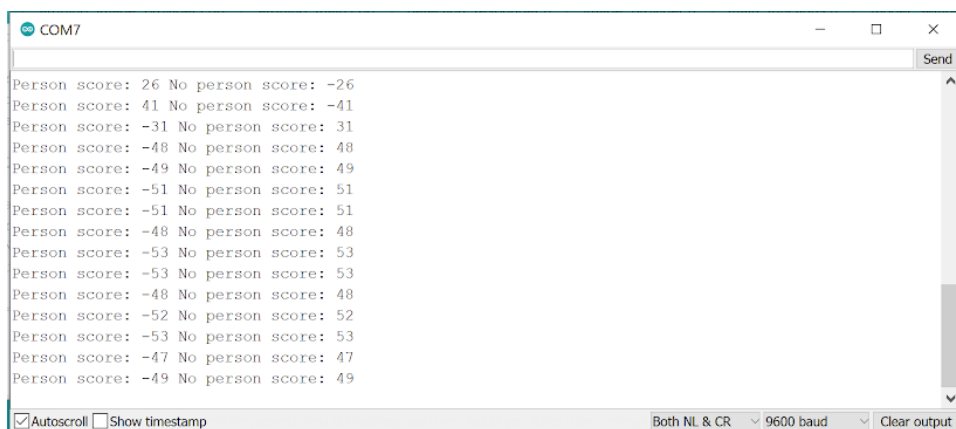
- b. Then select the USB Port associated with your board. This will appear differently on Windows, macOS, Linux but will likely indicate 'Arduino Nano 33 BLE' in parenthesis. You can select this by going to **Tools → Port: <Current Port (Board on Port)> → <TBD Based on OS> (Arduino Nano 33 BLE)**. Where <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 → **tttyUSB<#>** or **tttyACM<#>**
4. Use the rightward arrow to upload / flash the code. Do not be alarmed if you see a series of orange warnings appear in the console. This is expected as we are working with bleeding edge 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 have the OV7670 camera make sure to first make the changes indicated at the end of this document!

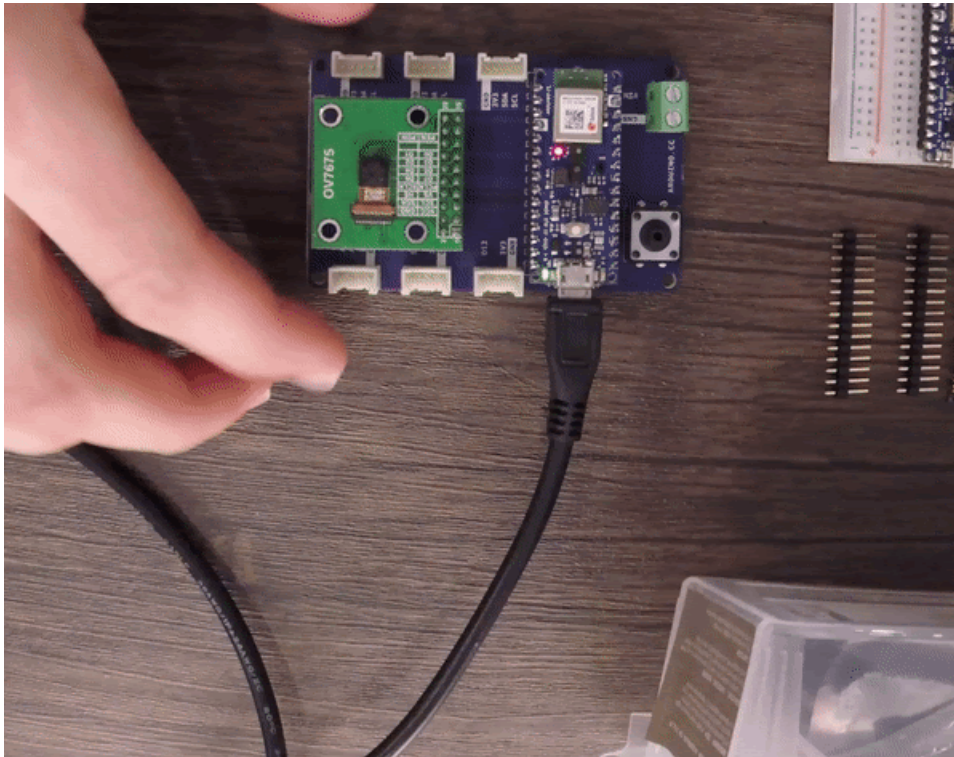
If you receive an error you will see an orange error bar appear and a red error message in the console. Don't worry -- there are many common reasons this may have occurred.

To help you debug other issues please check out our [FAQ appendix](#) with answers to the most common errors!

5. Open up the serial monitor. You will start to see it print a series of messages about the person score and the no person score. This is indicating the model's confidence of whether there is either a person or no person in the image!



At this point, you'll want to look at the board itself. The LED should be flashing blue when an image is being captured and then either red or green. Green indicates that the model detects that there is a person in the image and red indicates that it does not detect a person. Try holding the Arduino at arms length and point it toward your face. The LED should flash green. As you may The field of view is very narrow so if it's not working, try propping up the arduino and standing further back.



Understanding the Code in the Person Detection Example

Now that you have gotten the person detection application deployed to your microcontroller let's explore the code a little bit. The main [person_detection.ino](#) file consists of the standard arduino `setup()` and `loop()` functions. These are both quite similar to the `hello_world` and `micro_speech` example. To really start to understand the code let's take a look at a couple of the other files.

Open the [arduino_image_provider.cpp](#) file. Here we will find the `GetImage()` function which captures an image and loads it into the model's input.

First let's take a look at the camera initialization. Here we set the resolution of the image we receive from the camera to QCIF (176x144). In order to save space and complexity, the input to our model is a grayscale image, therefore we set the camera to transmit a grayscale image. We also set our camera model to the OV7675. Note: as shown in the changes to the OV7670 section below this is where you'll need to change the OV7675 to OV7670.

```
// Initialize camera if necessary
if (!g_is_camera_initialized) {
    if (!Camera.begin(QCIF, GRAYSCALE, 5, 0V7675)) {
        TF_LITE_REPORT_ERROR(error_reporter, "Failed to initialize camera!");
        return kTfLiteError;
    }
    g_is_camera_initialized = true;
}
```

Next we can capture an image with `Camera.readFrame()`. Since the input of our model is a 96x96 image we have to crop the 176x144 image we receive from the camera before passing it to the model. To do this we copy over the desired pixel values from the center of our original image into the input vector of our model.

```
// Read camera data
Camera.readFrame(data);

int min_x = (176 - 96) / 2;
int min_y = (144 - 96) / 2;
int index = 0;

// Crop 96x96 image. This lowers FOV, ideally we would downsample
// but this is simpler and more efficient on device.
for (int y = min_y; y < min_y + 96; y++) {
    for (int x = min_x; x < min_x + 96; x++) {
        // convert TF input image to signed 8-bit
        image_data[index++] = static_cast<int8_t>(data[(y * 176) + x] - 128);
    }
}
```

Now that we are familiar with capturing and loading the image, let's take a look at how the person detection application handles the output of the model. To do that open the [arduino_detection_responder.cpp](#) file.

The primary function of the detection responder is to control the LED output. In a real application this function would drive some other core function that is triggered by the presence of a person.

The blue LED will flash every inference, demonstrating how often an inference is run. Then we will compare the person no_person scores that are the outputs of the model. In this case we flash the green LED when the person_score is higher than the no_person_score (both of which are reported over serial). In some applications we might be more concerned with the false positives or false negatives, in which case we can adjust the criteria for detecting a person, potentially by using a threshold as we have discussed in Course 2.

```

//Flash the blue LED after every inference.
digitalWrite(LEDDB, LOW);
delay(100);
digitalWrite(LEDDB, HIGH);

// Switch on the green LED when a person is detected,
// the red when no person is detected
if (person_score > no_person_score) {
    digitalWrite(LEDG, LOW);
    digitalWrite(LEDRL, HIGH);
} else {
    digitalWrite(LEDG, HIGH);
    digitalWrite(LEDRL, LOW);
}

TF_LITE_REPORT_ERROR(error_reporter, "Person score: %d No person score: %d",
                        person_score, no_person_score);

```

And at a high level that is how the person_detection example works! We simply take and crop the input images, pass them to the neural network, and then report the higher scoring result!

Changes for the OV7670

To use the OV7670 instead of the OV7675, simply make the change you made in the camera test! Change the fourth argument of the call to `Camera.begin()` as highlighted below, from `OV7675` to `OV7670`. That's it! The library will handle the rest! You can find the camera initialization in

```

// Initialize the OV7675 camera
void if(!Camera.begin(QCIF, GRAYSCALE, 5, OV7675)) {
    Serial.println("Failed to initialize camera");
    While (1);
}

```