DEPLOYING AN OBJECT DETECTION MODEL

In this lab session, you'll use transfer learning in AI/ML to build an object detection system using a small 640x480 pixel camera (OV7675) provided by the Arduino TinyML microcontroller kit. You'll learn to collect image data from the camera sensor and use existing image labeling (i.e., YOLO or any other label mechanism) to label your image, train a pre-trained network, and deploy your model to a device. At the end of this tutorial, you'll fully understand the characteristics of transfer learning and how to utilize them in complex data pre-processing, such as images in embedded devices using Edge Impulse and Arduino Nano BLE 33.

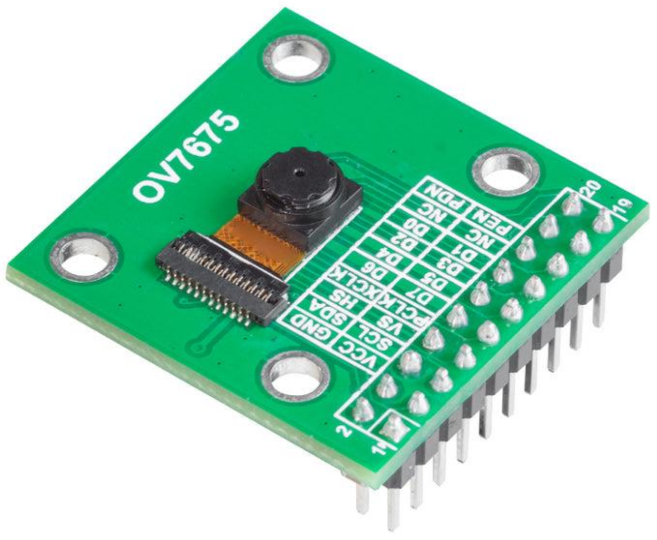
STATIONARY OBJECT DETECTION

This example shows how to re-train a pre-trained model using transfer learning and identify two objects, "scissors" and "pencils," through an Arduino Nano BLE 33 camera sensor (OV7675)

- ✓ Collect data from the device's camera
- ✓ Create training and testing dataset
- ✓ Through Edge Impulse, execute data labeling through a manual process or using YOLO to auto-label
- ✓ Train an object-detection based mobilenetV2 model (FOMO)
- ✓ Test model with live data and document device performance.

OV7675 CAMERA MODULE

The Arducam OV7675 camera supports resolutions of up to 640x480 pixels, ensuring clear and sharp images are captured. Its versatile range of features enhances the functionality of your projects. Whether you're developing a home automation system, a smart monitoring device, or any other Arduino-based project that requires visual input, the Arducam OV7675 camera offers a reliable and flexible solution. In this lab session, we will utilize the camera module to execute object detection using the ML model.



The OV7675 Camera Module

The Camera Specifications

Image Sensor	OV7576	Image Sensor	OV7576
Active Array Pixels	640x480	Resolution & Frame Rate	640x480/320x240/160x120@15fps
Pixel Size	2.5μmx2.5μm	Lens Optical Size	1/9 inch
S/N Ratio	38 dB	EFL	1.75mm
Dynamic Range	71 dB	F.NO	2.8
Interface	20-pin DVP	Field of View angle	63.9º
Shutter type	Electronic Rolling Shutter	Lens Distortion	<1.0%
Color Filter Array	Quad-Bayer RGB	Focus distance	0.12M - ∞(AT=0.23M)
Output Format	RAW/YUV/RGB	Power Supply	AVDD: 2.7VA ~ 3.0V DOVDD: 1.7V ~ 3.0V DVDD: 1.5V
Operational Temperature	-30° C to 70° C	Board Size	30.5mm x 30.5mm

Setting Up Your Project

1. Create a new project at Edge Impulse
2. Connect your Arduino Nano BLE 33 device through your terminal
3. You would need to clean your device connection from the previous project directory and reconnect it to your new project repository by typing `edge-impulse-daemon –clean`
4. Enter your email address and password
5. Connect your device to the correct project repository
6. After your device is successfully connected to the correct project, you will go to the edge impulse tab and do the following

(Troubleshooting: if your Arduino board is not connecting to edge impulse, go to your Arduino-nano-33-ble-sense folder and run the file according to your OS. Sometimes, the Arduino CLI influences the device connection with Edge Impulse since it is still in the developer mode, so you might need to flash the device)

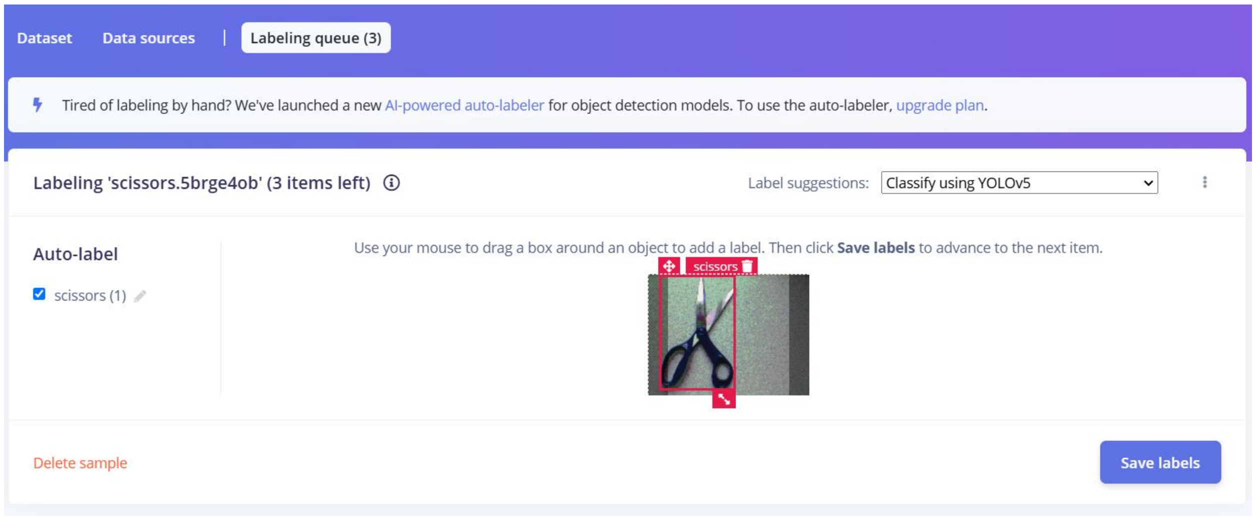
Collecting Data

1. With your device connected, we can collect some image data. In the studio, we go to the Data Acquisition tab. This is where all your raw data is stored, and you can start sampling new data.
2. Under "Collect Data," select your device and set the label name "pen or pencil."
3. Select the "Camera 160x120" option in the Sensor section
4. Position your camera so that it is steady, the image is not distorted, and the image can capture the whole object.
5. Select "start sampling"; it will take a snapshot of the image you positioned your camera on. Try taking the same object from multiple angles and positions for the robustness of the dataset.
6. After taking multiple snapshots of your object, go to the labeling queue option and start labeling either using manual tracing or "classify using YOLOv5".
7. Once all the images are labeled correctly, you can start the same process for objects as "scissors" and follow the same steps.

Classify using YOLOv5

In object detection ML projects, labeling defines regions of interest in the frame. Manually labeling images can become tedious and time-consuming, especially when dealing with massive datasets. This is why Edge Impulse Studio provides an AI-assisted labeling tool to help you in your labeling workflows. Using an existing library of pre-trained object detection models from YOLOv5 (trained with the COCO dataset), common objects in your images can quickly be identified and labeled in seconds without needing to write any code!

If your object is more specific than what is auto-labeled by YOLOv5, e.g. "coffee" instead of the generic "cup" class, you can modify the auto-labels to the left of your image. These modifications will automatically apply to future images in your labeling queue.



Dataset | Data sources | Labeling queue (3)

Tired of labeling by hand? We've launched a new AI-powered auto-labeler for object detection models. To use the auto-labeler, upgrade plan.

Labeling 'scissors.5brge4ob' (3 items left)

Label suggestions: Classify using YOLOv5

Auto-label

☑ scissors (1)

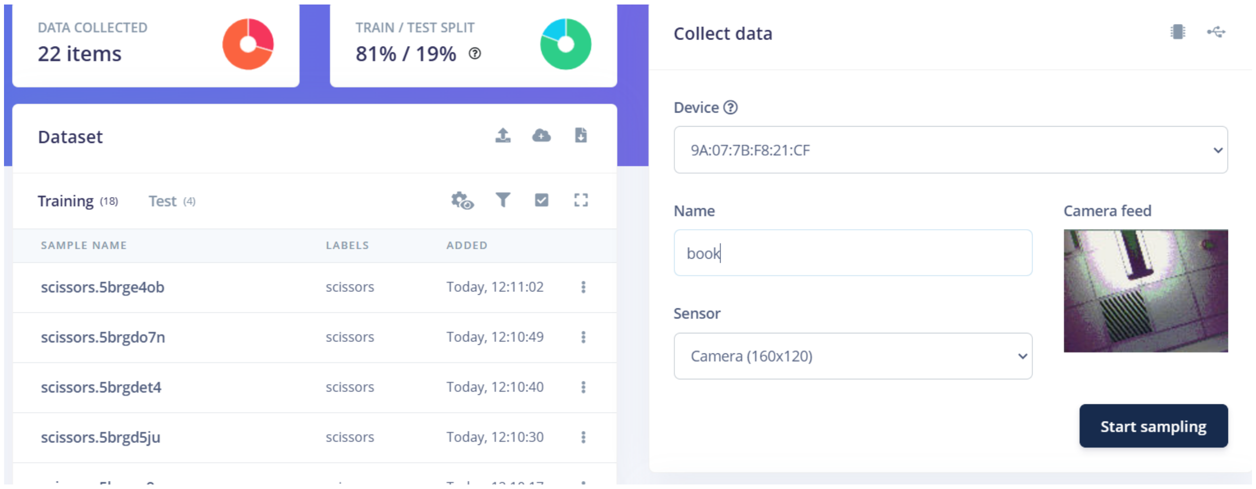
Use your mouse to drag a box around an object to add a label. Then click **Save labels** to advance to the next item.

Delete sample

Save labels

Labeling queue tab and auto labeling using YOLOv5

8. Split your dataset into at least 70%/30% range or 80%/20% (train/test split)



DATA COLLECTED 22 items

TRAIN / TEST SPLIT 81% / 19%

Dataset

Training (18) Test (4)

SAMPLE NAME	LABELS	ADDED
scissors.5brge4ob	scissors	Today, 12:11:02
scissors.5brgdo7n	scissors	Today, 12:10:49
scissors.5brgdet4	scissors	Today, 12:10:40
scissors.5brgd5ju	scissors	Today, 12:10:30

Collect data

Device: 9A:07:7B:F8:21:CF

Name: book

Sensor: Camera (160x120)

Camera feed

Start sampling

Data sampling section

EDGE IMPULSE

1. After successfully collecting your data, go to the "create impulse" tab and keep time series data as is
2. It will automatically identify "Image" in its processing block, select the recommended classifier in the learning block and select "save impulse."

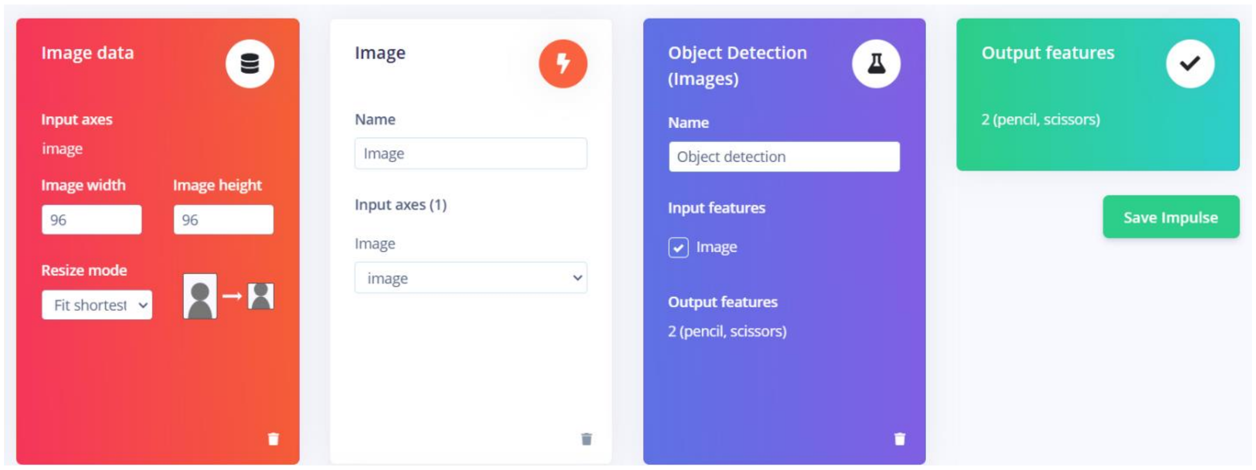


Image data

Input axes
image

Image width: 96
Image height: 96

Resize mode: Fit shortest

Image

Name: Image

Input axes (1)
Image: image

Object Detection (Images)

Name: Object detection

Input features
[✓] Image

Output features
2 (pencil, scissors)

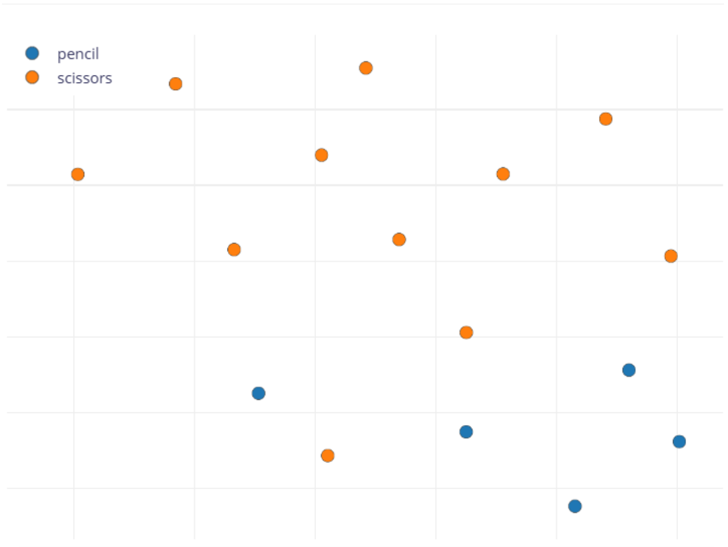
Output features
2 (pencil, scissors)

Edge Impulse "Create Impulse" Dashboard after selecting necessary blocks.

NOTE: Sometimes labeling your data incorrectly can mistreat it as another output feature.

IMAGE FEATURE GENERATE

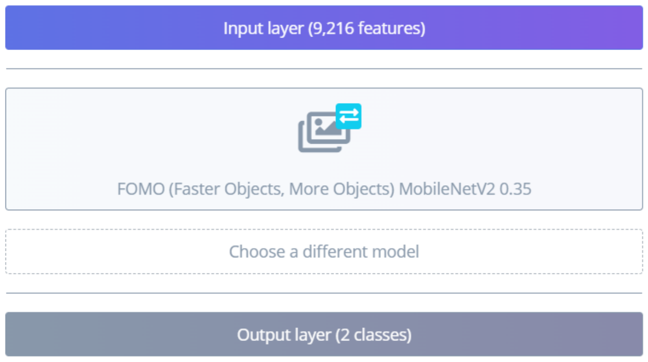
1. Go to **Impulse Design > Image**
2. Select "**Greyscale**" in the parameter section as **color depth**.
3. Then click on the "**save parameters**" button
4. It will prompt you to the "**Generate Feature**" tab; click on the "**Generate Feature**" button
5. The generate feature tab will take in your image datasets and generate greyscale images.
6. After the proper generation of data, it will prompt you to the "Feature Explorer" tab, which will give you a UMAP of your dataset samples.
7. This will make it easy to identify whether your distribution is grouped and distributed correctly.



The actual features will be generated. Using UMAP, a dimension reduction technique, the Feature explorer shows how the features are distributed on a two-dimensional plot.

CLASSIFIER

1. Go to the Classifier tab
2. You can mark your Neural Network training either in block level or expert mode
3. In this lab session, we will utilize transfer learning and training a pre-trained FOMO (faster objects, more objects)-base mobilenetV2 model.


Input layer (9,216 features)

FOMO (Faster Objects, More Objects) MobileNetV2 0.35

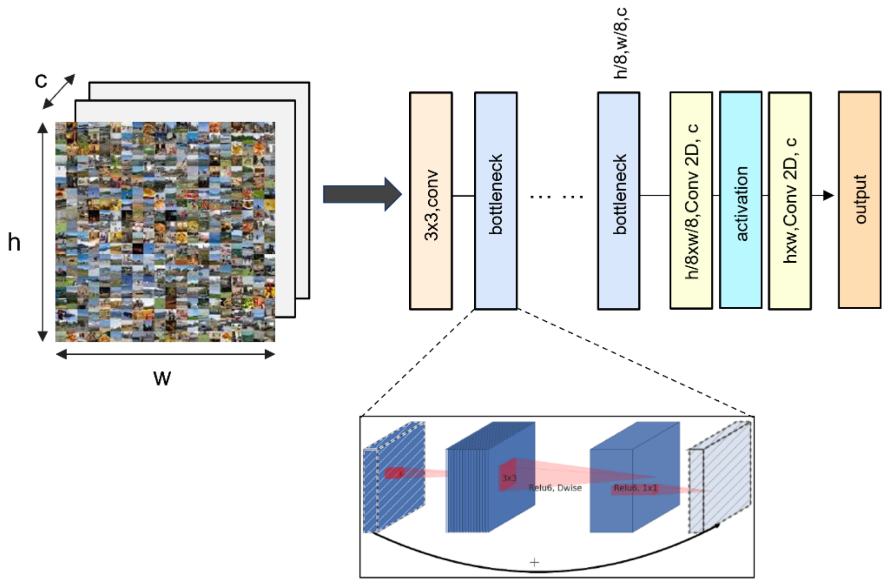
Choose a different model

Output layer (2 classes)

Neural Network Setting Block

FOMO MOBILENETV2

FOMO is an algorithm that brings real-time object detection, tracking, and counting to microcontrollers for the first time. FOMO is 30x faster than MobileNet SSD and runs in <200K of RAM. The FOMO model provides a variant in between; a simplified version of object detection suitable for many use cases where the position of the objects in the image is needed but when a large or complex model cannot be used due to resource constraints on the device. By default, FOMO uses MobileNetV2 as a base model for its trunk and does a spatial reduction of 1/8th from input to output (e.g., a 96x96 input results in a 12x12 output). This is implemented by cutting MobileNet off at the intermediate layer block_6_expand_relu. Logically, FOMO can be thought of as using the first section of MobileNetV2 followed by a standard classifier, where the classifier is applied fully convolutional fashion.



FOMO architecture

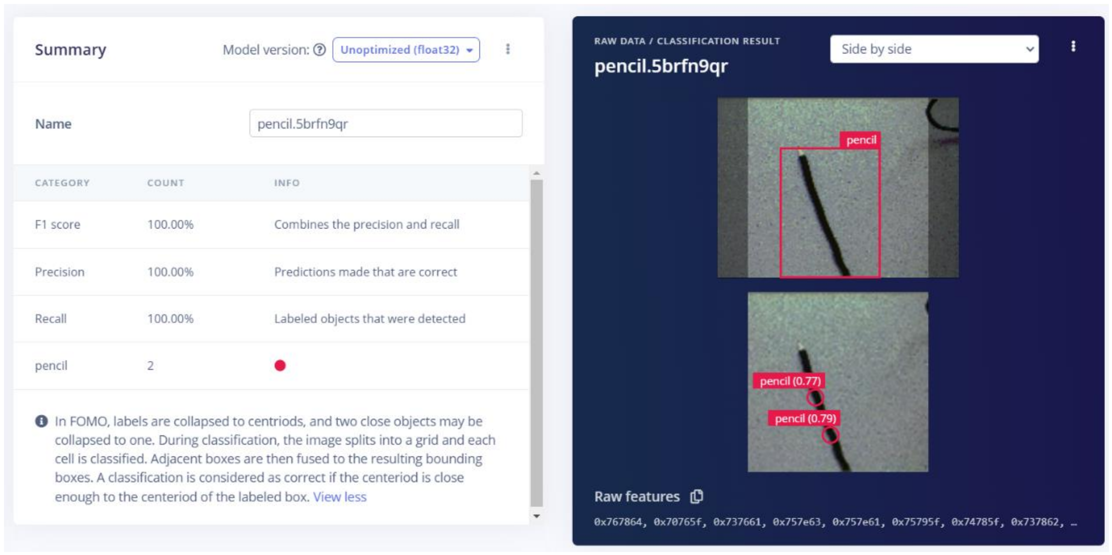
In the default configuration this FOMO classifier is equivalent to a single dense layer with 32 nodes followed by a classifier with num_classes outputs.

More about FOMO can be found: here

4. Click on the Save & Train button
5. In the training output you are able to see your loss, accuracy and val_accuracy in the "training output" tab over each epoch.
6. After training you can check your confusion matrix in the "Model" tab.

Live Classification

1. Go to the **Live classification** tab
2. In the classify new data section, upload real-time image data to test your model
3. Click on the "Start Sampling" button
4. The classification result tab will generate your testing data, and your trained FOMO model will try to classify it based on the three output features it was trained on.


Summary

Model version: Unoptimized (float32)

Name: pencil.5brfn9qr

CATEGORY	COUNT	INFO
F1 score	100.00%	Combines the precision and recall
Precision	100.00%	Predictions made that are correct
Recall	100.00%	Labeled objects that were detected
pencil	2	🔴

In FOMO, labels are collapsed to centriods, and two close objects may be collapsed to one. During classification, the image splits into a grid and each cell is classified. Adjacent boxes are then fused to the resulting bounding boxes. A classification is considered as correct if the centeriod is close enough to the centeriod of the labeled box. View less

RAW DATA / CLASSIFICATION RESULT

pencil.5brfn9qr

Side by side

Raw features

```
0x767864, 0x70765f, 0x737661, 0x757e63, 0x757e61, 0x75795f, 0x74785f, 0x737862, …
```

Raw data and features collected from a live demonstration

Report

1. Document your model architecture **(classifier)** and its result by using your own collected data from the OV7675 camera sensor and generate a minimum of two output features. Try to achieve an accuracy of over 82% and document the results generated by the classifier.
2. Document the feature explorer section and include the generated graph after testing.
3. Execute a live demonstration and include your confidence point from the "**live classification/Classification Result Tab.**"
4. Document the "**on device**" section after training to see your inference time, memory utilization, and peak performance.

Happy Inference!