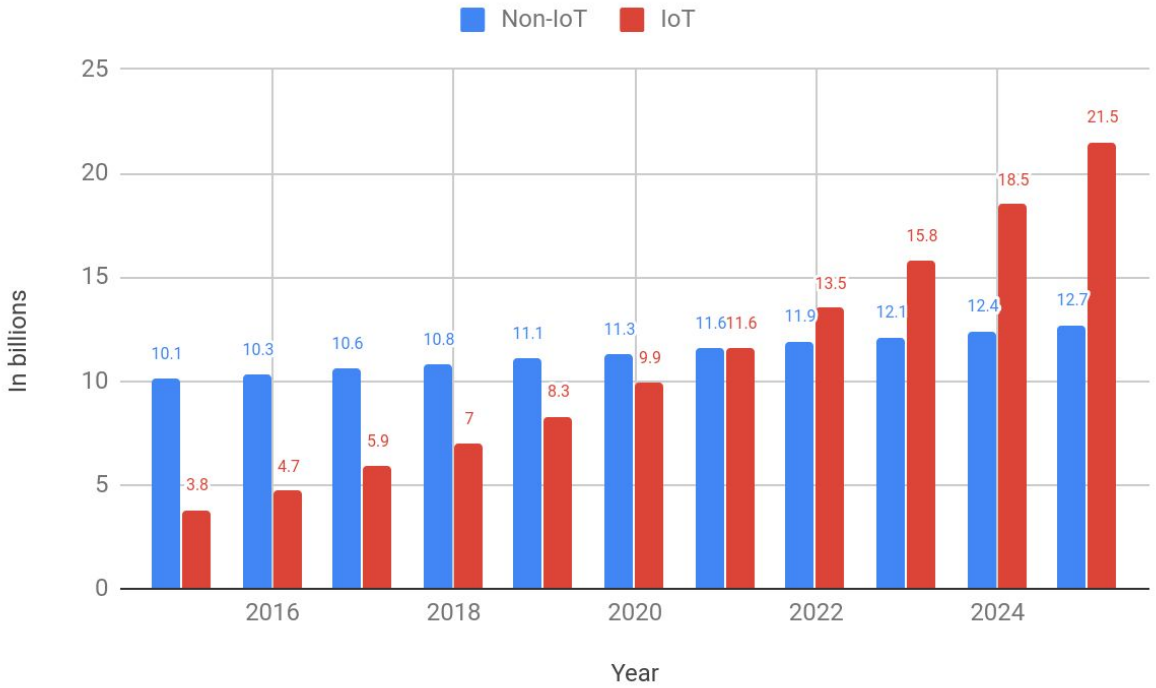
Copyright Notice

Trends in adoption of Smart devices



IoT Analytics - State of the IoT 2018

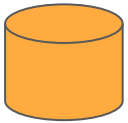
Factors driving this trend

- Smaller, faster models
- On device accelerators
- Demands move ML capability from cloud to on-device
- Smart device growth demands bring ML to the edge

Benefits of on-device ML

High-performance

Better privacy



Local data accessibility

Works offline

Coral

Coral Dev Board

Coral USB Accelerator

Image credits

Software for Coral

- Mendel OS
- Edge TPU Compiler
- Mendel Development Tool (`mdt`)
- Edge TPU models : https://coral.withgoogle.com/models/

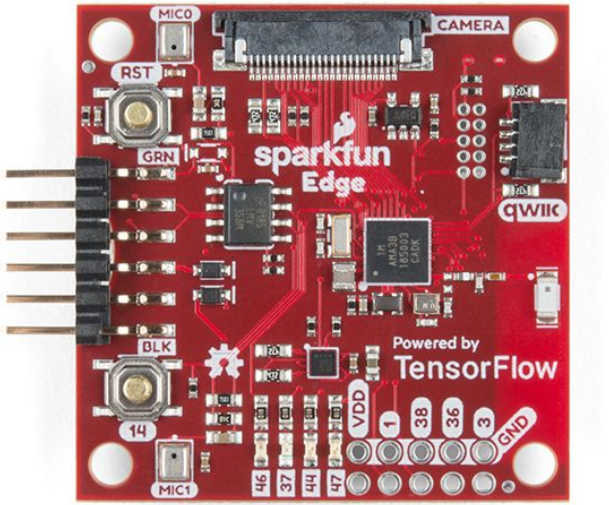
Raspberry Pi

- Small sized
- Low cost
- Just like a computer
- Accessibility
- Raspbian (OS)

Image credits

Micro-controllers

- Low-powered
- Small form-factor
- Some specially designed for ML tasks
- No reliance on network connectivity



SparkFun Edge

Image credits

Options

- Compile TensorFlow from Source
- Install TensorFlow with pip
- Use TensorFlow Lite Interpreter directly

Build From Source

https://www.tensorflow.org/install/source_rpi

```
curl -sSL https://get.docker.com | sh

sudo docker run hello-world
```

Build From Source

https://www.tensorflow.org/install/source_rpi

```
git clone https://github.com/tensorflow/tensorflow.git

cd tensorflow
```

Build From Source

https://www.tensorflow.org/install/source_rpi

```
sudo CI_DOCKER_EXTRA_PARAMS= \
        "-e CI_BUILD_PYTHON=python3 \
         -e CROSSTOOL_PYTHON_INCLUDE_PATH=/usr/include/python3.4" \
         tensorflow/tools/ci_build/ci_build.sh PI-PYTHON3 \
         tensorflow/tools/ci_build/pi/build_raspberry_pi.sh
```

Build From Source

https://www.tensorflow.org/install/source_rpi

```
pip install <your wheel name>
```

Use Pre-built Packages

https://www.tensorflow.org/install/pip

```
sudo apt update
sudo apt install python3-dev python3-pip
sudo apt install libatlas-base-dev
pip install --upgrade tensorflow
```

Use Interpreter Only

https://www.tensorflow.org/lite/guide/python

```
pip3 install tflite_runtime-1.14.0-cp37-cp37m-linux_armv7l.whl
```

dog	0.91
rabbit	0.07
hamster	0.02

Identify classes of different objects in the image

Model

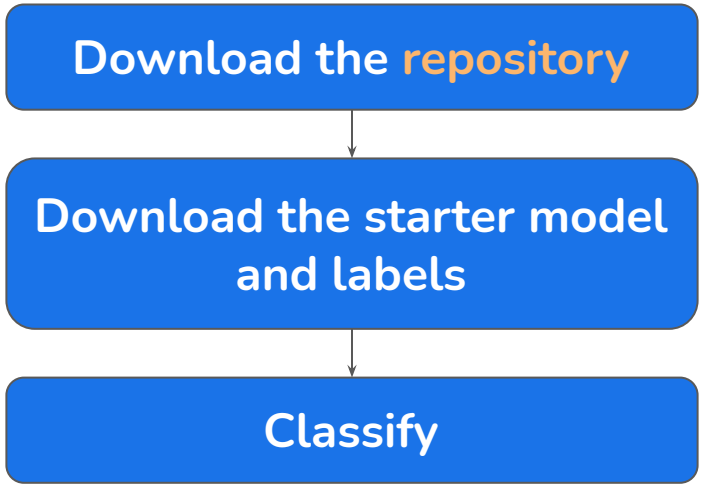
Pre-quantized MobileNet trained on ImageNet

1000 different classes of objects

More details on the model can be found at

https://www.tensorflow.org/lite/models/image_classification/overview

Quickstart


Download the repository
Download the starter model and labels
Classify

1

Initialize the Interpreter

Load the interpreter with the model and make it ready for inference

2

Preprocess input

Preprocess by resizing and normalizing the image data

3

Perform Inference

Pass input to the Interpreter and invoke it

4

Obtain results and map

Extract the resulting scores for each class and map them

classification.py

Initializing the Interpreter

```
# Load the model and allocate tensors
interpreter = tf.lite.Interpreter(model_path='mobilenet_v2_1.0_224.tflite')
interpreter.allocate_tensors()
```

classification.py

Get the model’s tensors

```
# Load the model and allocate tensors
interpreter = tf.lite.Interpreter(model_content=tflite_model)
interpreter.allocate_tensors()

# Get input and output tensors.
input_details = interpreter.get_input_details()
output_details = interpreter.get_output_details()
...
```

1

Initialize the Interpreter

Load the interpreter with the model and make it ready for inference

2

Preprocess input

Preprocess by resizing and normalizing the image data

3

Perform Inference

Pass input to the Interpreter and invoke it

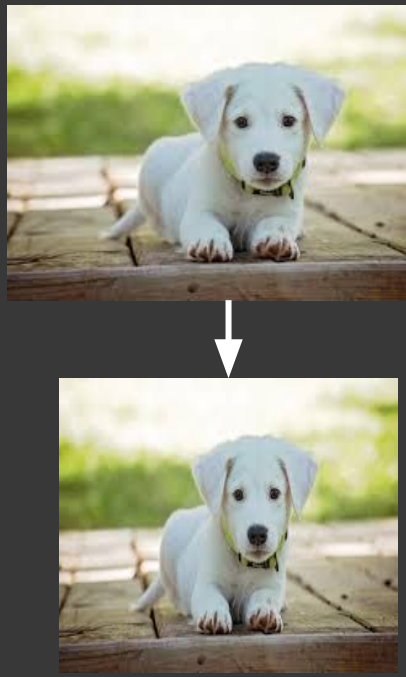
4

Obtain results and map

Extract the resulting scores for each class and map them

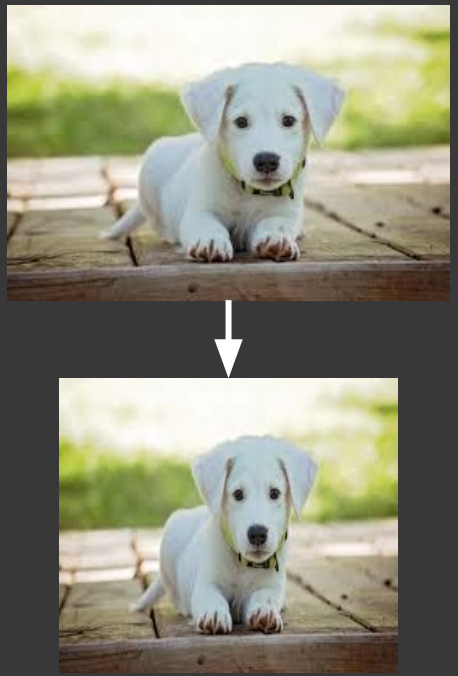
Preprocess the image

```
# Read image and decode
img = tf.io.read_file(filename)
img_tensor = tf.image.decode_image(img)

# Preprocess image
img_tensor = tf.image.resize(img_tensor, size)
img_tensor = tf.cast(img_tensor, tf.uint8)

# Add a batch dimension
input_data = tf.expand_dims(img_tensor, axis=0)
```



Preprocess the image

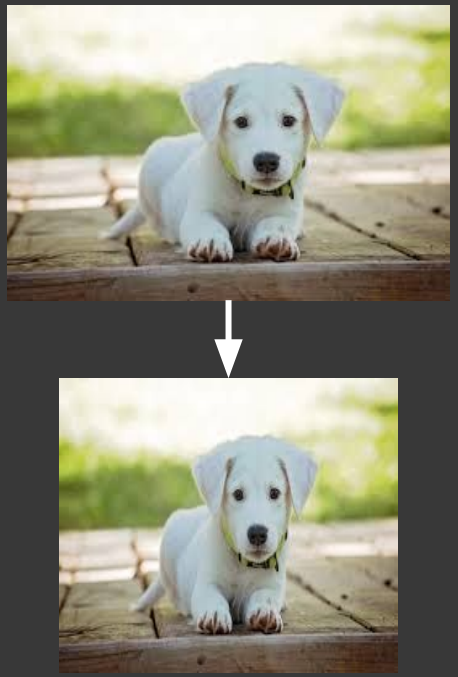
```
# Read image and decode
img = tf.io.read_file(filename)
img_tensor = tf.image.decode_image(img)

# Preprocess image
img_tensor = tf.image.resize(img_tensor, size)
img_tensor = tf.cast(img_tensor, tf.uint8)

# Add a batch dimension
input_data = tf.expand_dims(img_tensor, axis=0)
```



classification.py

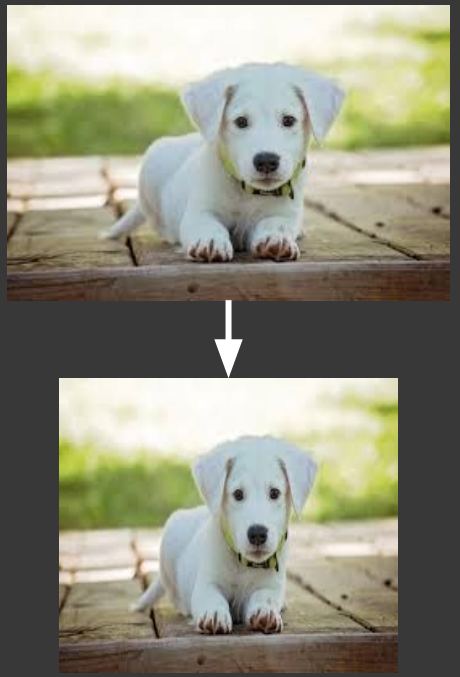
Preprocess the image

```
# Read image and decode
img = tf.io.read_file(filename)
img_tensor = tf.image.decode_image(img)

# Preprocess image
img_tensor = tf.image.resize(img_tensor, size)
img_tensor = tf.cast(img_tensor, tf.uint8)

# Add a batch dimension
input_data = tf.expand_dims(img_tensor, axis=0)
```



Preprocess the image

```
# Read image and decode
img = tf.io.read_file(filename)
img_tensor = tf.image.decode_image(img)

# Preprocess image
img_tensor = tf.image.resize(img_tensor, size)
img_tensor = tf.cast(img_tensor, tf.uint8)

# Add a batch dimension
input_data = tf.expand_dims(img_tensor, axis=0)
```



1

Initialize the Interpreter

Load the interpreter with the model and make it ready for inference

2

Preprocess input

Preprocess by resizing and normalizing the image data

3

Perform Inference

Pass input to the Interpreter and invoke it

4

Obtain results and map

Extract the resulting scores for each class and map them

classification.py

Perform inference

```
# Point data to be used for testing the interpreter and run it
interpreter.set_tensor(input_details[0]['index'], input_data)
interpreter.invoke()
```

1

Initialize the Interpreter

Load the interpreter with the model and make it ready for inference

2

Preprocess input

Preprocess by resizing and normalizing the image data

3

Perform Inference

Pass input to the Interpreter and invoke it

4

Obtain results and map

Extract the resulting scores for each class and map them

classification.py

Report only the top results

```
# Obtain results
predictions = interpreter.get_tensor(output_details[0]['index'])

# Get indices of the top k results
_, top_k_indices = tf.math.top_k(predictions, k=top_k_results)
top_k_indices = np.array(top_k_indices)[0]

for i in range(top_k_results):
    print(labels[top_k_indices[i]],
          predictions[top_k_indices[i]] / 255.0)
```

Label	Probability
dog	0.91
rabbit	0.07
hamster	0.02

classification.py

Report only the top results

```
# Obtain results
predictions = interpreter.get_tensor(output_details[0]['index'])

# Get indices of the top k results
_, top_k_indices = tf.math.top_k(predictions, k=top_k_results)
top_k_indices = np.array(top_k_indices)[0]

for i in range(top_k_results):
    print(labels[top_k_indices[i]],
          predictions[top_k_indices[i]] / 255.0)
```

Label	Probability
dog	0.91
rabbit	0.07
hamster	0.02

classification.py

Report only the top results

```
# Obtain results
predictions = interpreter.get_tensor(output_details[0]['index'])

# Get indices of the top k results
_, top_k_indices = tf.math.top_k(predictions, k=top_k_results)
top_k_indices = np.array(top_k_indices)[0]

for i in range(top_k_results):
    print(labels[top_k_indices[i]],
          predictions[top_k_indices[i]] / 255.0)
```

Label	Probability
dog	0.91
rabbit	0.07
hamster	0.02

classification.py

Report only the top results

```
# Obtain results
predictions = interpreter.get_tensor(output_details[0]['index'])

# Get indices of the top k results
_, top_k_indices = tf.math.top_k(predictions, k=top_k_results)
top_k_indices = np.array(top_k_indices)[0]

for i in range(top_k_results):
    print(labels[top_k_indices[i]],
          predictions[top_k_indices[i]] / 255.0)
```

Label	Probability
dog	0.91
rabbit	0.07
hamster	0.02

Detect multiple objects within an image

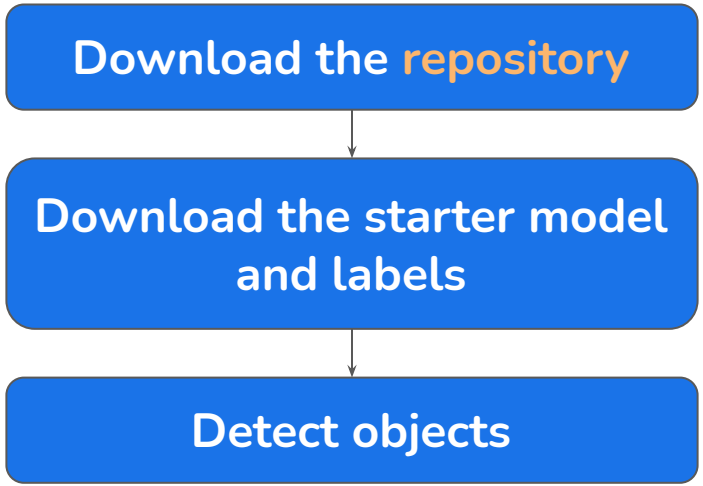
Recognize different classes of objects

Model

- Pre-optimized MobileNet SSD trained on COCO dataset
- COCO dataset has 80 common object categories
- A labels file to map the model’s outputs
- You can download the model (.tflite) and labels (.txt)

http://storage.googleapis.com/download.tensorflow.org/models/tflite/coco_ssd_mobilenet_v1_1.0_quant_2018_06_29.zip

Quickstart


Download the repository
Download the starter model and labels
Detect objects

1

Initialize the Interpreter

Load the interpreter with the model and make it ready for inference by getting the input and output tensors

2

Preprocess input

Center crop the input image so that the model can accept the input

3

Perform Inference

Pass the preprocessed input to the Interpreter and invoke it

4

Fetch the outputs

Extract the outputs as locations, classes, scores and number of detections

detector.py

Initializing the Interpreter

```
# Load the model and allocate tensors
interpreter = tf.lite.Interpreter(model_path='detect.tflite')
interpreter.allocate_tensors()

# Get input and output tensors.
input_details = interpreter.get_input_details()
output_details = interpreter.get_output_details()
```

1

Initialize the Interpreter

Load the interpreter with the model and make it ready for inference

2

Get cam feed and preprocess

Gather images from the camera feed and center crop the input image so that the model can accept the input

3

Perform Inference

Pass the preprocessed input to the Interpreter and invoke it

4

Fetch the outputs

Extract the outputs as locations, classes, scores and number of detections

main.py

Raspberry Pi Camera

```
with picamera.PiCamera() as camera:
    camera.resolution = (640, 480)
    while True:
        # Input image
        image = np.empty((480, 640, 3), dtype=np.uint8)
        camera.capture(image, 'rgb')
        # Use the frame captured from the stream
```

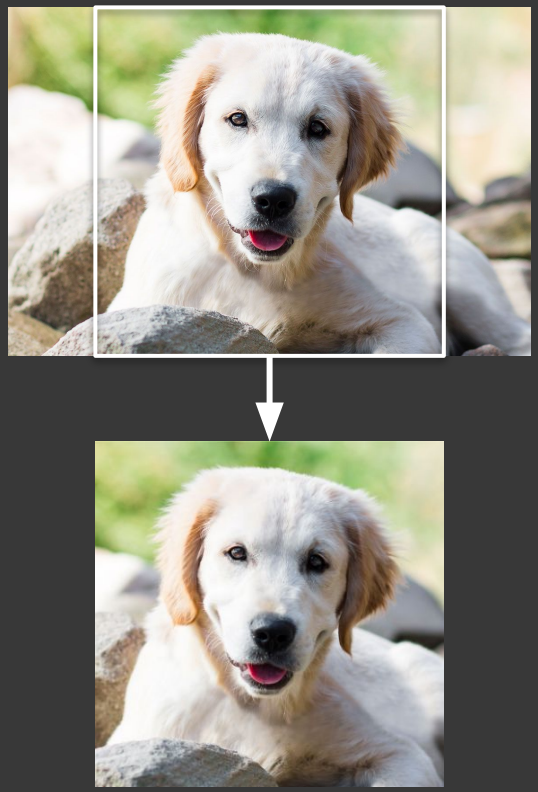
Preprocessing

Image credits

utils.py

Center and Crop

```
# Get the dimensions
height, width, _ = frame.shape # Image shape
new_width, new_height = (300, 300) # Target shape

# Calculate offsets between heights and widths
offset_height = (height - new_height) // 2
offset_width = (width - new_width) // 2

# Crop to the biggest square in the center
image = tf.image.crop_to_bounding_box(frame,
                                      offset_height,
                                      offset_width,
                                      new_height,
                                      new_width)
```



1

Initialize the Interpreter

Load the interpreter with the model and make it ready for inference

2

Get cam feed and preprocess

Gather images from the camera feed and center crop the input image so that the model can accept the input

3

Perform Inference

Pass the preprocessed input to the Interpreter and invoke it

4

Fetch the outputs

Extract the outputs as locations, classes, scores and number of detections

detector.py

Perform inference

```
def detect(self, image, threshold=0.1):
    # Add a batch dimension
    frame = np.expand_dims(image, axis=0)

    # run model
    self.interpreter.set_tensor(self.input_details[0]['index'], frame)
    self.interpreter.invoke()
```

1

Initialize the Interpreter

Load the interpreter with the model and make it ready for inference

2

Get cam feed and preprocess

Gather images from the camera feed and center crop the input image so that the model can accept the input

3

Perform Inference

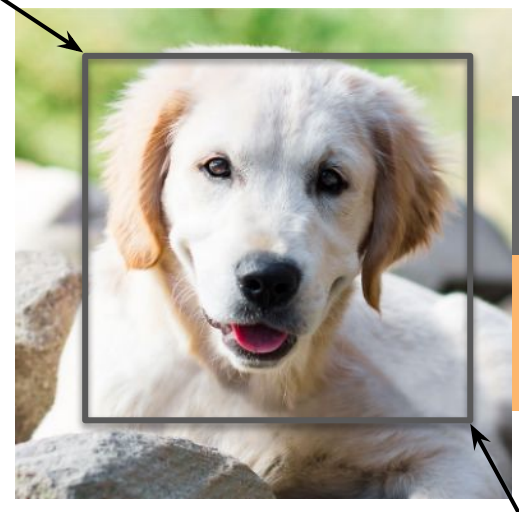
Pass the preprocessed input to the Interpreter and invoke it

4

Fetch the outputs

Extract the outputs as locations, classes, scores and number of detections

How a detected object is represented

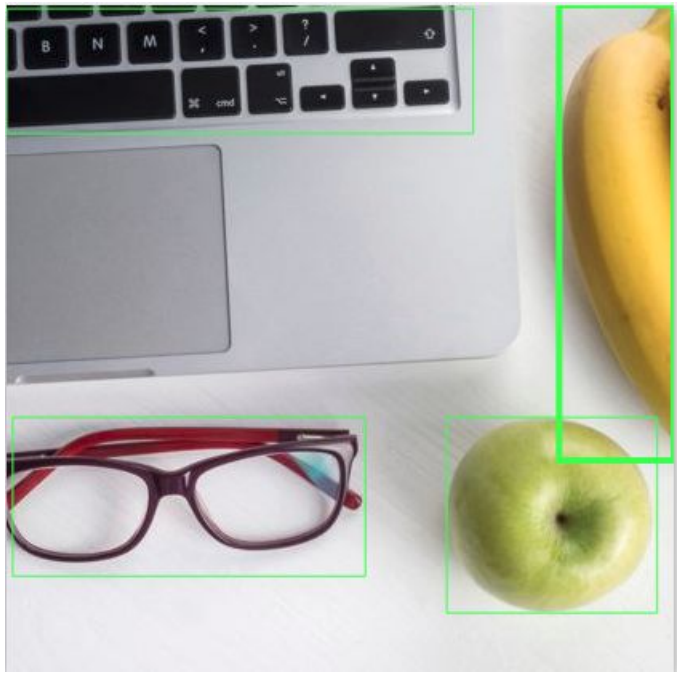


Class	Score	Location
Dog	0.92	top, left, bottom, right

How the COCO model sees outputs

index	name	
0	locations	A list of floats in [0, 1] representing *normalized bounding boxes* [top left bottom right]
1	classes	A list of integers (output as float) each indicating the *index of a class label* from the labels file
2	scores	Array of floats in [0, 1] representing *probability* that a class was detected
3	number of detections	Floating point value expressing *total number of results*

Interpreting the number of results



Class	Score	Location (top, left, bottom, right)
Apple	0.96	275, 257, 407, 379
Glasses	0.89	4, 257, 224, 356
Computer Keyboard	0.77	0, 2, 292, 80
Banana	0.67	345, 0, 417, 284

Image credits

detector.py

Fetching the outputs

```
# Normalized coordinates of the detected objects
boxes = interpreter.get_tensor(output_details[0]['index'])[0]

# Recognized classes of objects
classes = interpreter.get_tensor(output_details[1]['index'])[0]

# Probabilities of the detected classes
scores = interpreter.get_tensor(output_details[2]['index'])[0]

# Maximum number of results
num_detections = interpreter.get_tensor(output_details[3]['index'])[0]
```

detector.py

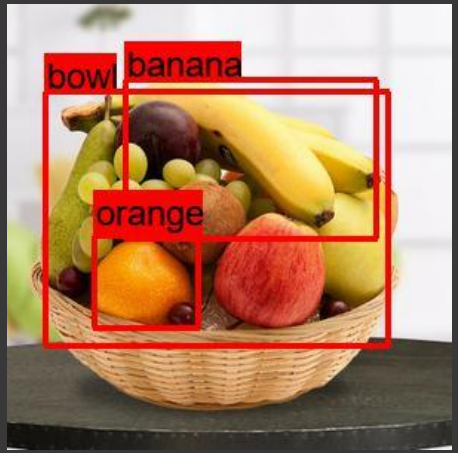
Discarding less relevant results

```
min_score_thresh = 0.6
number_boxes = boxes.shape[0]
detected_boxes = []
probabilities = []
categories = []

for i in range(number_boxes):
    if scores is None or scores[i] > min_score_thresh:
        box = tuple(boxes[i].tolist()) # [top, left, bottom, right]
        detected_boxes.append(box)
        probabilities.append(scores[i])
        categories.append(self.category_index[classes[i]])
```

visualization_utils.py

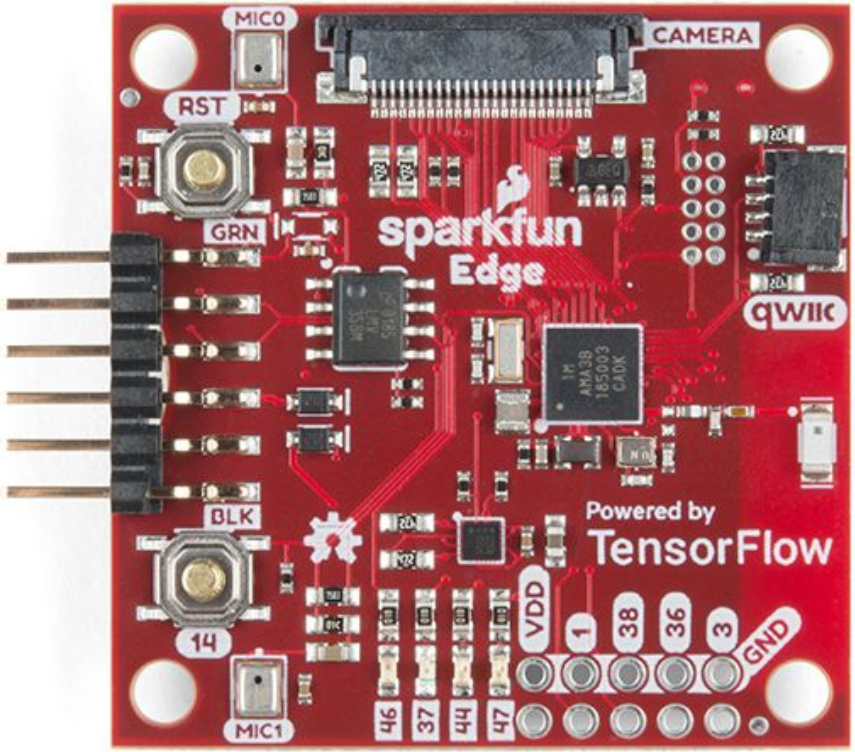
Reporting results

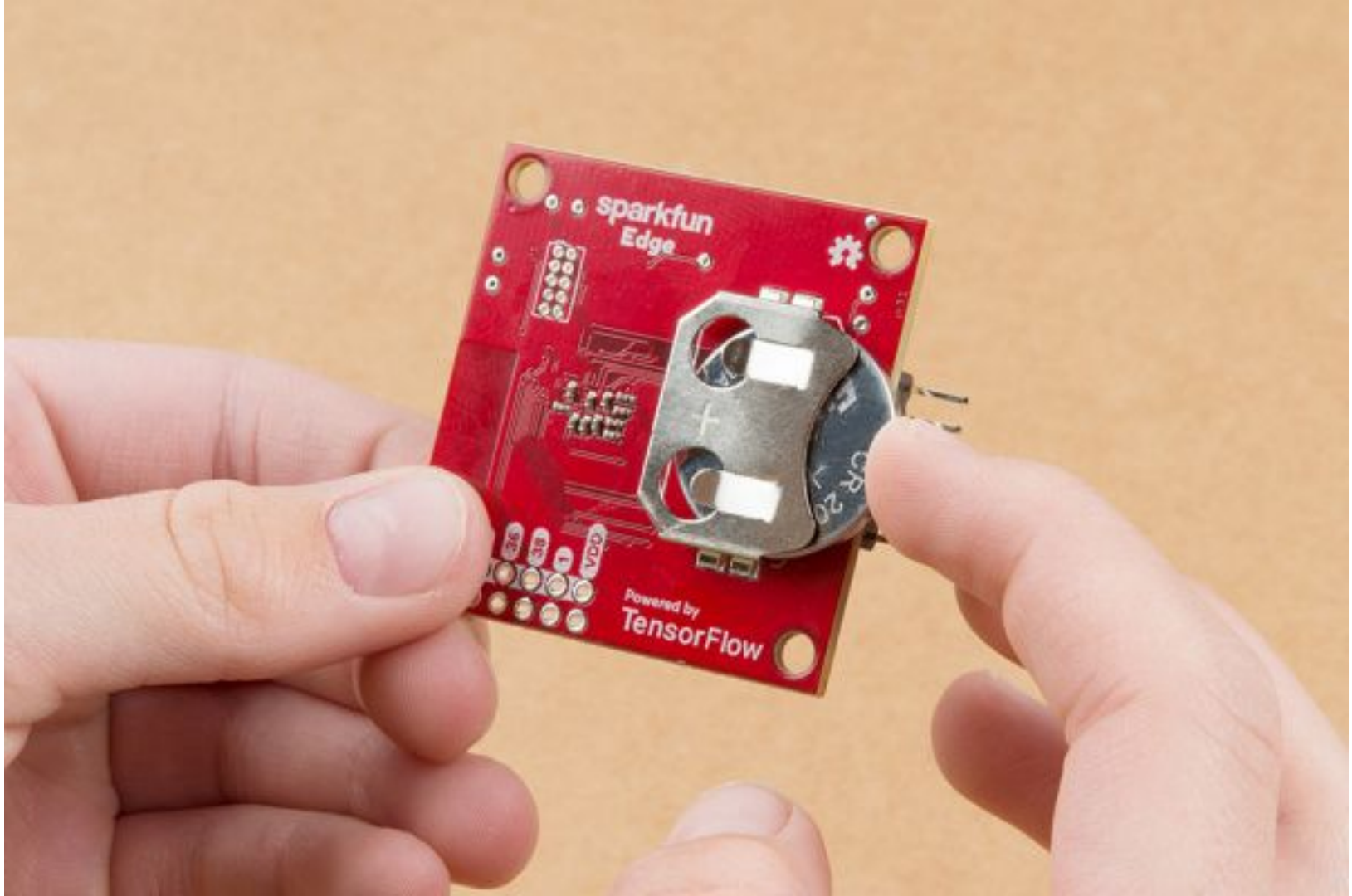
```
# Convert normalized boxes to regular bounding boxes
(top, bottom, left, right) = (top * im_height, bottom * im_height,
                              left * im_width, right * im_width)

# Draw lines for the detected boxes
draw.line([(left, top), (left, bottom), (right, bottom), (right, top),
           (left, top)], width=thickness, fill=color)

# Draw the display string (predicted class)
draw.text(
    # Calculate position of text to be placed at the top-left corner
    (left + margin, text_bottom - text_height - margin),
    display_str,
    fill='black',
    font=font)
```



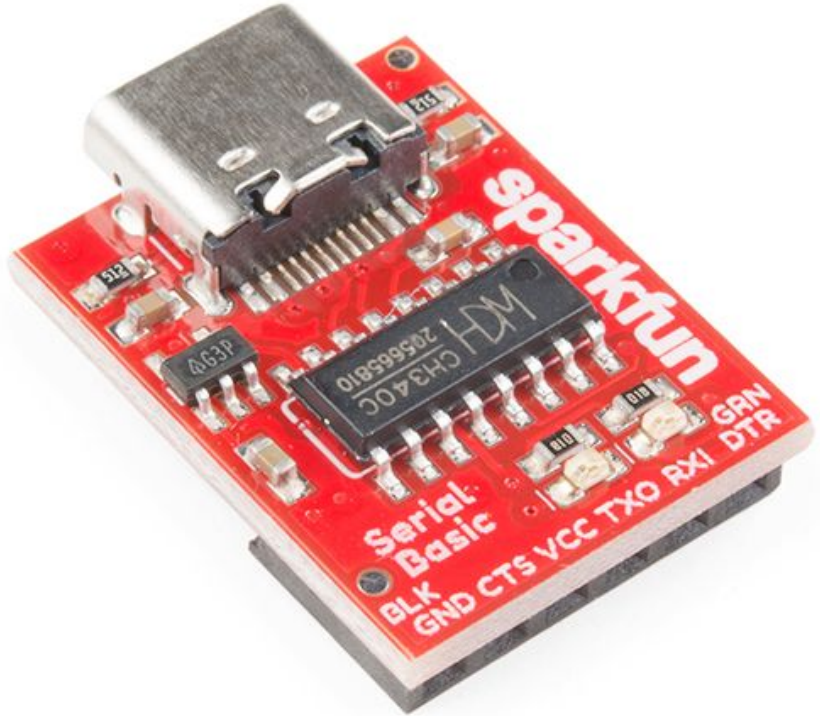
https://www.youtube.com/watch?v=e-Gu6Sp4OUQ

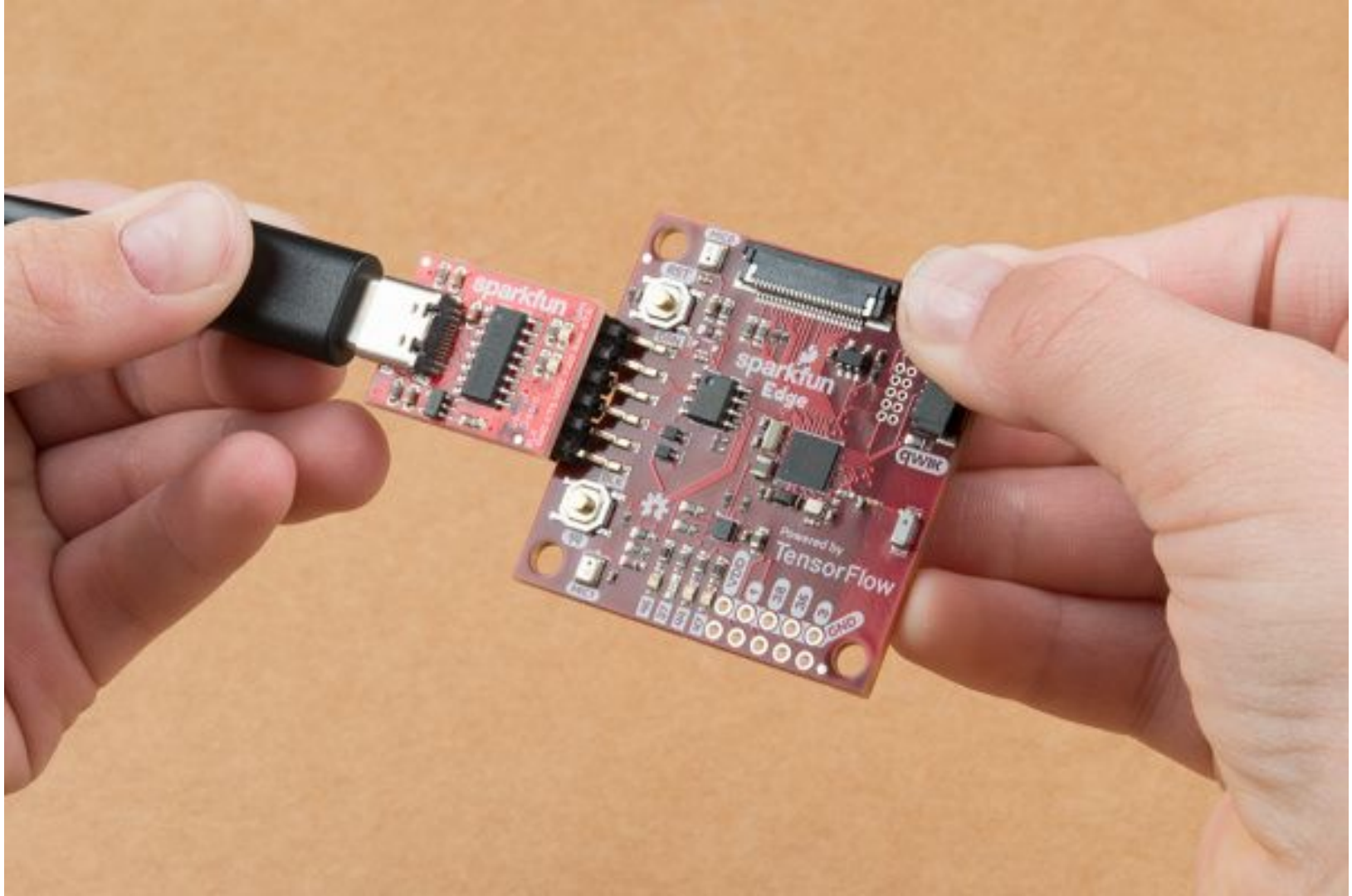
https://www.sparkfun.com/products/15170

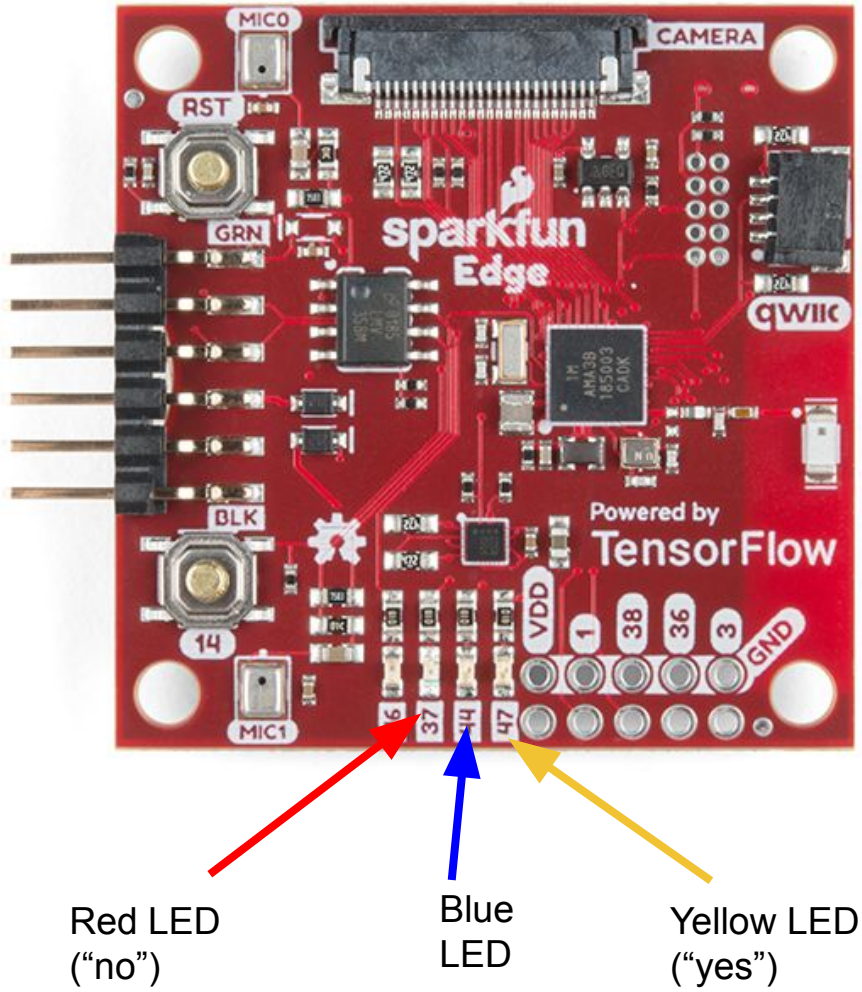



sparkfun
Edge
Powered by
TensorFlow

https://www.sparkfun.com/products/15096




sparkfun
sparkfun
Edge
Powered by
TensorFlow


MIC0
CAMERA
RST
GRN
sparkfun
Edge
qwiic
BLK
Powered by
TensorFlow
14
VDD
1
38
36
3
GND
MIC1
6
37
4
47
Red LED
(“no”)
Blue
LED
Yellow LED
(“yes”)

https://codelabs.developers.google.com/codelabs/sparkfun-tensorflow/

AI on a microcontroller with TensorFlow Lite and SparkFun Edge

20 mins remaining

Report a mistake

1. Introduction

Machine learning helps developers build software that can understand our world. We can use it to create intelligent tools that make users' lives easier, like the Google Assistant, and fun experiences that let users express their creativity, like Google Pixel's portrait mode.

But often, these experiences require a lot of computation. Machine learning often needs to run on a powerful cloud server, or at least a powerful mobile phone.

With TensorFlow Lite, it's possible to run machine learning inference on tiny, low-powered hardware, like microcontrollers. This means you can build amazing experiences that add intelligence to the smallest devices, bringing machine learning closer to the world around us.

In this codelab, we'll learn to deploy a machine learning model to the SparkFun Edge, a microcontroller designed by Google and SparkFun to help developers experiment with ML on tiny devices.

What is the SparkFun Edge?

The SparkFun Edge is a microcontroller-based platform: a tiny computer on a single circuit board. It has a processor, memory, and I/O hardware that allows it to send and receive digital signals to other devices. It also has four software-controllable LEDs, in your favorite Google colors.

Next

https://www.tensorflow.org/lite/microcontrollers/get_started