STEM STUDENT OUTREACH USING ROBOTICS AND FACIAL RECOGNITION AND OBJECT DETECTION

A Master's Project Report
Submitted to
the Temple University Graduate Board

In Partial Fulfillment of the
Requirements for the Degree of
Master of Electrical Engineering

---

by
Richard Sand
Diploma Date December 2023

Examining Committee Members:

Dr. Li Bai, Advisory Chair, Department of Electrical and Computer Engineering

Dr. Saroj Biswas, Department of Electrical and Computer Engineering

ABSTRACT

Object detection and facial recognition are two key capabilities in modern robotics. These allow robots to perform a myriad of autonomous tasks. Autonomous robots include smart robots that can recognize and react to their surroundings, such as with self-driving vehicles.

This project demonstrates simple object detection and facial recognition using a small commercial-grade robot called "Mini Pupper". This robot is significant in that it resembles larger projects such as the Stanford Quadruped and has many of the same capabilities, but in a footprint and price point aimed at consumers and hobbyists.

We are developing a simple toolkit for educator to do the STEM outreach using a Jupyter notebook. At this moment, we have developed two scripts to perform face recognition and object detection with a visual output that can be easily demonstrated and presented.

TABLE OF CONTENTS

Page

LIST OF FIGURES

CHAPTER 1
INTRODUCTION

The purpose of STEM education is to inspire and enable students with STEM abilities. Students need STEM and analytic problem-solving skills by combining integrated learning on real-world problems with hands-on learning. [1]

However, student enrollment in STEM education is declining for a variety of reasons, including the shutdown during COVID. [2]. Therefore, increased effort is needed with STEM outreach to ensure the next generation of scientists and engineers. [3]

Robotics is a natural tool for such outreach because it is highly visible and naturally engaging. Robotics help foster student interest in STEM and helps teach problem-solving skills. [4]

There are several classes of robot platforms that can be used for education outreach. These include 2 wheeled robots such as the *SparkFun JetBot AI Kit*, *4 wheeled robots like the Yahboom Raspbot AI Vision Robot Car*, quadruped and hexaped robots such as the *Mini-Pupper*, Lego's *Mindstorm*, drones, and others.

In this project we will implement a robot with AI vision-based capabilities to be used for STEM outreach at the high-school level.

CHAPTER 2

PROPOSAL

Several types of robots were considered for this project, including wheeled, quadruped, and arms. With the availability of consumer-grade (e.g. priced) robots, we were free to select a type of robot for maximum engagement for an outreach program. In 2015, the Google-owned Massachusetts firm Boston Dynamics introduced "Spot", a nimble quadruped robot that became extremely popular on social media. [5] We proposed to use a small quadruped robot due to the popular success of "Spot".

The Mini-Pupper (v1) is a small quadruped robot based upon a Raspberry PI microcontroller. It retails for under $600. It features 12 degrees of freedom, 3 per leg. Each hip can articulate front-to-back and side-to-side, and each knee front-to-back.

The pupper is expandable with a variety of accessories such as camera, microphone, LIDAR sensors, accelerometers, gyroscopes, inertial sensors, and GPS sensors, just to list a few. Note that the next generation of the Mini-Pupper - version 2 - comes with feedback sensors on each actuator so that the robot knows exactly what position each actuator (servo) is in.

For this project, we use the *Arducam 4K 8MP IMX219 Autofocus USB Camera Module with Metal Case*. https://www.amazon.com/dp/B09BR1RNSN

CHAPTER 3
IMPLEMENTATION

Both demos run directly from Jupyter Notebook.

If you followed the instructions to auto-start the Jupyter service then it should be running. Otherwise start the service and navigate to *http://<ip of your pupper>:8888*. To run the facial-recognition training, establish a secure shell (SSH) to the console of the Raspberry PI. Additional face images can be added via secure FTP (SFTP) as described below.

3.1 Facial Recognition

The facial recognition notebook is based upon Paul McWorter's AI tutorial [6], which in turn leverages Adam Geitgey's $face\_recognition$ library for python [7]. This library implements *Histograms of Oriented Gradient* (HOG) to create a feature set for each face learned. According to Dalal and Bill Triggs, locally normalized HOG descriptors provide excellent performance relative to other existing feature sets. [8]

Launch the Jupyter notebook *faceRecognizer.ipynb*.

The face images are now in the folder $\sim$ */docs*. If you want to add your own image, place here as well. The file name is used as the label.

Now we run the training:

```
cd
python train.py
```

You will see output similar to below:

```
Encoding Barack Obama
Encoding Hillary Clinton
Encoding Richard Sand
Encoding Nancy Pelosi
Encoding Gal Gadot
Encoding Bill Clinton
Encoding Taylor Swift
Encoding Ivanka Trump
Encoding George Clooney
Encoding George W. Bush
Encoding Matt Damon
```

3.2 Object Detection

Launch the Jupyter notebook *objectRecognizer.ipynb*. The object detection notebook is based upon Evan Juras's instructions for TensorFlow on the Raspberry Pi [9].

The Raspberry Pi used by the minipupper 1 has a relatively weak CPU, so we need a model that takes less processing power. We use the SSD Lite model from *TensorFlow Detection Model Zoo*. The zoo is Google's collection of pre-trained object detection models. The models vary their levels of speed and accuracy. The SSD Lite model is faster, but with lower accuracy. For this tutorial, we'll use SSDLite-MobileNet, which is the fastest model available.

CHAPTER 4

RESULTS

4.1 Facial Recognition Demo

Below is an example screenshot of the output window. The program correctly recognizes the one face in the picture and draws a box around it with the name as the label. Note that the program can detect multiple faces in a single frame.

Figure 4.1. Facial Recognition Output

Facial recognition is widely adopted today. For example, Facebook can recognize people in your photos after only a few tags [10]. Facial recognition is performed in several steps. First is to locate all faces in the frame. Then those faces are analyzed for features, and those features compared to the learned feature sets.

4.1.1 Imports

The program uses OpenCV (*cv*2) for running the camera capture, *IPython.display* and *ipywidgets* to display the output inside Jupyter, and *pickle* for loading the training data.

```
import cv2
from IPython.display import display,Image
import ipywidgets.widgets as widgets
import face_recognition
import os
import pickle
```

4.1.2 Starting the Video Capture

The next block of code sets two constants. It selects the font to use when labelling the frame, and the scaling constant for reducing the image size for analysis. Scaling down the image allows the recognition algorithm to process much quicker by having less data to analyse. So long as the amount of data is sufficient to maintain accuracy, this value should be kept as small as possible.

```
font = cv2.FONT_HERSHEY_SIMPLEX
scale = 0.333
```

The next line opens the camera for capture. On the Mini Pupper 1, there was an odd behavior where the USB camera could not be used by OpenCV after booting the device. Some process in the boot sequence was grabbing control of the camera and not releasing it. It was necessary to plug the camera in after boot (e.g. unplug and plug in again). Also note that the arugment "1" indicates the video device number. This may vary in different environments.

```
# Start the video capture
cam=cv2.VideoCapture(1)
```

4.1.3 Reading the Training Data

The training data is encoded using *pickle*. The data consists of labels and data for each face trained.

```
# Open the training data
with open('/home/ubuntu/docs/train.pkl', 'rb') as tdf:
    names = pickle.load(tdf)
    encodings = pickle.load(tdf)
```

```
# Start the IPython display
display_handle=display(None, display_id=True)
```

4.1.4 Main Loop

The main loop runs continuously until interrupted. It performs the following tasks:

- Read the next frame from the camera
- Resize the frame to the scale factor
- Convert the frame to RGB
- Invoke the facial recognition library to find all face positions
- Encode the found faces via HOG
- Loop through all face positions and encoding data
- Attempt to map each face encoding with the training data
- If a match is found, set the label
- Draw and label a box around the current face
- Update the frame display

```
try:
    while True:
        # Read the next frame
        _, frame = cam.read()
        frame = cv2.flip(frame, 1) # if your camera reverses your image
        # Convert to RGB
        frameSmall = cv2.resize(frame,(0,0), fx=scale, fy=scale)
        frameRGB = cv2.cvtColor(frameSmall, cv2.COLOR_BGR2RGB)
        # Find faces
        facePositions = face_recognition.face_locations(frameRGB)
        allEncodings = face_recognition.face_encodings(frameRGB, facePositions)
        for (top,right,bottom,left),face_encoding in zip(facePositions,allEncodings):
            name = 'Unknown'
            matches = face_recognition.compare_faces(encodings, face_encoding)
            if True in matches:
                matchIdx = matches.index(True)
                name = names[matchIdx]
                print('Recognized', name)
            top = int(top / scale)
            right = int(right / scale)
            bottom = int(bottom / scale)
            left = int(left / scale)
            cv2.rectangle(frame, (left,top), (right,bottom), (255,0,255), 2)
            cv2.putText(frame, name, (left,top - 6), font, 0.75, (255,0,255), 2)
        # Display as jpeg image
        _, frame = cv2.imencode('.jpeg', frame)
        display_handle.update(Image(data=frame.tobytes()))
except KeyboardInterrupt:
    pass
finally:
    cam.release()
    display_handle.update(None)
```

4.2 Object Detection Demo

Below is an example screenshot of the output window.

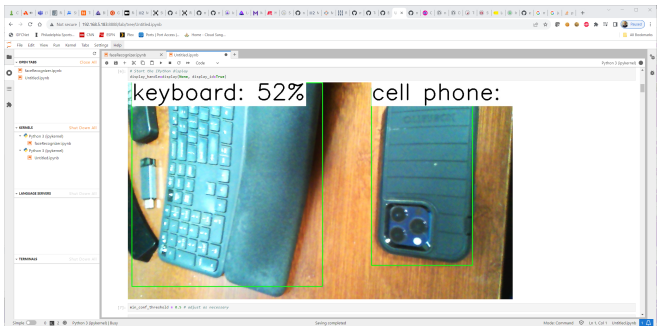

Figure 4.2. Object Detection Output

4.2.1 Imports

The program uses OpenCV (*cv*2) for running the camera capture, *IPython.display* and *ipywidgets* to display the output inside Jupyter, and *tflite* for performing the object recognition.

```
from tflite_runtime.interpreter import Interpreter
import numpy as np
import cv2
from matplotlib import pyplot as plt
from IPython.display import display,Image
import tflite_runtime
print(tflite_runtime.__version__)
```

4.2.2 Load the Model

This block loads the SSD-Lite-MobileNet model, which was selected for its speed. There is additional logic to support both Tensorflow 1 and Tensorflow 2, which have a slight difference in model persistence.

```
# Load model
interpreter = Interpreter(model_path="coco_ssd_mobilenet_v1_1.0_quant_2018_06_29/detect.tflite")
interpreter.allocate_tensors()
input_details = interpreter.get_input_details()
output_details = interpreter.get_output_details()
height = input_details[0]['shape'][1]
width = input_details[0]['shape'][2]
floating_model = (input_details[0]['dtype'] == np.float32)
input_mean = 127.5
input_std = 127.5
# Check output layer name to determine if this model was created with TF2 or TF1,
# because outputs are ordered differently for TF2 and TF1 models
outname = output_details[0]['name']
if ('StatefulPartitionedCall' in outname): # This is a TF2 model
    boxes_idx, classes_idx, scores_idx = 1, 3, 0
else: # This is a TF1 model
    boxes_idx, classes_idx, scores_idx = 0, 1, 2
# un-comment to see the details
print(input_details, output_details, height, width)
```

4.2.3 Load the Model Labels

Here we load the labels corresponding with each learned object in the model.

```
# Load the label map
with open("coco_ssd_mobilenet_v1_1.0_quant_2018_06_29/labelmap.txt", "r") as f:
    labels = [line.strip() for line in f.readlines()]
# Have to do a weird fix for label map if using the COCO "starter model" from
# https://www.tensorflow.org/lite/models/object_detection/overview
# First label is '???', which has to be removed.
if labels[0] == '???':
    del(labels[0])
```

4.2.4 Starting the Video Capture

The next block of code sets the *confidence threshold* used for object recognition. The higher the number, the higher the match confidence must be to accept a match. Setting the number higher improves accuracy but can eliminate matches from being recognized. Empirically, we determined that 0.5 is a good threshold for demonstration purposes.

```
min_conf_threshold = 0.5 # adjust as necessary
```

The next line opens the camera for capture. On the Mini Pupper 1, there was an odd behavior where the USB camera could not be used by OpenCV after booting the device. Some process in the boot sequence was grabbing control of the camera and not releasing it. It was necessary to plug the camera in after boot (e.g. unplug and plug in again). Also note that the arugment "1" indicates the video device number. This may vary in different environments.

```
# Start the video capture
cam=cv2.VideoCapture(1)
dispW = int(cam.get(cv2.CAP_PROP_FRAME_WIDTH))
dispH = int(cam.get(cv2.CAP_PROP_FRAME_HEIGHT))
# Start the IPython display
display_handle=display(None, display_id=True)
```

4.2.5 Main Loop

The main loop runs continuously until interrupted. It performs the following tasks:

- Read the next frame from the camera
- Resize the frame to the scale factor
- Convert the frame to RGB

- Loads and normalizes the frame data into a *Numpy* array
- Invoke the TensorFlow interpreter to identify objects in the frame
- Loop through the top 5 detections
- If the object confidence exceeds the threshold, set the label
- Draw and label a box around the recognized object
- Update the frame display

```
try:
    while True:
        # Read the next frame
        _, frame = cam.read()
        frame = cv2.flip(frame, 1) # if your camera reverses your image
        # Convert to RGB
        #frameSmall = cv2.resize(frame,(0,0), fx=scale, fy=scale)
        frameSmall = cv2.resize(frame, (height, width))
        frameRGB = cv2.cvtColor(frameSmall, cv2.COLOR_BGR2RGB)
        input_data = np.expand_dims(frameRGB, axis=0)
        # Normalize pixel values if using a floating model (i.e. if model is non-quantized)
        if floating_model:
            input_data = (np.float32(input_data) - input_mean) / input_std
        # Perform the actual detection by running the model with the image as input
        interpreter.set_tensor(input_details[0]['index'], input_data)
        interpreter.invoke()
        # Retrieve detection results
        boxes = interpreter.get_tensor(output_details[boxes_idx]['index'])[0] # Bounding box coordinates of detected ob
        classes = interpreter.get_tensor(output_details[classes_idx]['index'])[0]+1 # Class index of detected objects
        scores = interpreter.get_tensor(output_details[scores_idx]['index'])[0] # Confidence of detected objects
        # Loop over for first 5 detections and draw detection box if confidence is above minimum threshold
        if sum(x>min_conf_threshold for x in scores) > 5:
            print("too many objects!")
            continue
        for i in range(len(scores)):
            if ((scores[i] > min_conf_threshold) and (scores[i] <= 1.0)):
                # Get bounding box coordinates and draw box
                # Interpreter can return coordinates that are outside of image dimensions, need to force them to be wit
                ymin = int(max(1,(boxes[i][0] * dispH)))
                xmin = int(max(1,(boxes[i][1] * dispW)))
                ymax = int(min(dispH,(boxes[i][2] * dispH)))
                xmax = int(min(dispW,(boxes[i][3] * dispW)))
                cv2.rectangle(frame, (xmin,ymin), (xmax,ymax), (255, 0, 255), 2)
                # Draw label
                object_name = labels[int(classes[i])-1] # Look up object name from "labels" array using class index
```

```
                label = '%s: %d%%' % (object_name, int(scores[i]*100)) # Example: 'person: 72%'
                labelSize, baseLine = cv2.getTextSize(label, cv2.FONT_HERSHEY_SIMPLEX, 1.5, 2) # Get font size
                label_ymin = max(ymin, labelSize[1] + 15) # Make sure not to draw label too close to top of window
                cv2.rectangle(frame, (xmin, label_ymin-labelSize[1]-15), (xmin+labelSize[0], label_ymin+baseLine-15), (
                cv2.putText(frame, label, (xmin, label_ymin-10), cv2.FONT_HERSHEY_SIMPLEX, 1.5, (255, 0, 255), 2) # Dra
                xcenter = (xmax+xmin)/2
                ycenter = (ymax+ymin)/2
                print("Label, (X, Y), max(X, Y) = {}, ({}, {}), ({}, {})".format(object_name, xcenter, ycenter, dispW,
        # Display as jpeg image
        _, frame = cv2.imencode('.jpeg', frame)
        display_handle.update(Image(data=frame.tobytes()))
except KeyboardInterrupt:
    pass
finally:
    cam.release()
    display_handle.update(None)
```

CHAPTER 5
CONCLUSION

The Mini-Pupper is a good platform for teaching several topics, including robotics, ROS2, and machine learning. This project successfully demonstrates facial recognition and object detection on the Mini-Pupper using python *OpenCV*, *face_recognition* and *Tensorflow* libraries.

The Jupyter notebooks make demonstrating and presenting easy to perform and replicate. The technique for piping the display window to Juypter Notebook was key to enabling this.

Future work would include adding the necessary ROS2 service clients to enable the scripts to invoke the ROS2 services controlling the motion of the robot, so that the robot can orient itself to keep the tracked objects/faces centered in its field-of-view. Building upon that, the programmatic capabilities can be expanded to follow the tracked subject. LIDAR can also be used to augment the camera view for more precise tracking and expanded field-of-view.

REFERENCES CITED

[1] M. Ali, C. Talib, J. Surif, N. Ibrahim, and A. Abdullah. *2018 IEEE 10th Int. Conf. on Engineering Education (ICEED)*, pages 111–116, 2018.

[2] Emma R Wester, Lisa L Walsh, Sandra Arango-Caro, and Kristine L Callis-Duehl. Student engagement declines in stem undergraduates during covid-19-driven remote learning. *Journal of microbiology and biology education, 2021, Vol.22 (1)*, 2021.

[3] Katie Stringer, Kevi Mace, Tedra Clark, and Tara Donahue. Stem focused extracurricular programs: who's in them and do they change stem identity and motivation? *Research in science and technological education, 2020, Vol.38 (4)*, pages 507–522, 2020.

[4] Yiching Chen and Chi-Cheng Chang. The impact of an integrated robotics stem course with a sailboat topic on high school students' perceptions of integrative stem, interest, and career orientation. *Eurasia Journal of Mathematics, Science and Technology Education, 2018, Vol.14 (12)*, page em1614, 2018.

[5] Jeremy Moses and Geoffrey Ford. See spot save lives: fear, humanitarianism, and war in the development of robot quadrupeds. *Digital War, 2021, Vol.2 (1-3)*, pages 64–76, 2021.

[6] Paul McWhorter. Learning Artificial Intelligence on the Jetson Nano, 2020. https://www.youtube.com/playlist?list=PLGs0VKk2DiYxP-E1Z7-QXIERFFPkOuP4_.

[7] Adam Geitgey. Face recognition, 2018. https://github.com/ageitgey/face_recognition.

[8] N. Dilal and B. Triggs. Histograms of oriented gradients for human detection. *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2005.

[9] Evan Juras. Tutorial to set up TensorFlow Object Detection API on the Raspberry Pi, 2019. https://github.com/EdjeElectronics/TensorFlow-Object-Detection-on-the-Raspberry-Pi.

[10] Adam Geitgey. Machine learning is fun! part 4: Modern face recognition with deep learning, 2016. https://medium.com/@ageitgey/machine-learning-is-fun-part-4-modern-face-recognition-with-deep-learning-c3c

APPENDICES

APPENDIX A
INSTALLATION

This chapter walks through the setup on the Mini Pupper 1 to prepare it to run the Jupyter Notebook demos. NOTE: there is a script in the git repo called *setup.sh* that will perform all of the below tasks (except updating ROS2 and the DLib update for NVidia CUDA).

A.1 Flash the Drive

A 32 GB micro-SSD flash drive was plenty sufficient to run the pupper and demos. The flash image used can be found at: https://drive.google.com/drive/folders/175PWfA3oIgC565D_2thNbvp9qQf6Y6-V

Create the file *wireless.txt* and place into *system_boot* on the USB drive

or

after boot, using a local screen and keyboard, modify the file $/etc/netplan/50-cloud-init.yaml$. Then execute:

```
sudo netplan apply
reboot
```

Install some utility packages:

```
sudo apt-get install v4l-utils unzip
```

A.2 Update the Pupper Code - OPTIONAL

If you want to update, the latest pupper code can be downloaded from branch "ros2" from here:

https://github.com/mangdangroboticsclub/mini_pupper_ros

https://github.com/mangdangroboticsclub/mini_pupper_bsp

To clone latest code:

```
git clone https://github.com/mangdangroboticsclub/mini_pupper_ros.git -b ros2
cd ~/ros2_ws
rosdep install --from-paths src --ignore-src -r --skip-keys=gazebo_plugins --skip-keys=velodyne_gazebo_plugins -y
colcon build --symlink-install
```

A.3 Install Jupyter

Install Jupyter Notebook as follows:

```
sudo pip3 install pillow lxml jupyter matplotlib cython
```

To launch jupyter-notebook as a system service:

```
sudo cp ~/jupyter.service /etc/systemd/system/
sudo systemctl daemon-reload
sudo systemctl enable jupyter.service
sudo systemctl start jupyter.service
```

The file *jupyter.service* is a *systemd* service descriptor file. Its contents are (beware the line wrapping):

```
[Unit]
Description=Jupyter-Notebook Daemon

[Service]
Type=simple
ExecStart=/bin/bash -c
    "/usr/local/bin/jupyter-notebook --no-browser --notebook-dir=/home/ubuntu --ip 0.0.0.0 --NotebookApp.token=''"
```

```
WorkingDirectory=/home/ubuntu
User=ubuntu
Group=ubuntu
PIDFile=/run/jupyter-notebook.pid
Restart=on-failure
RestartSec=60s

[Install]
WantedBy=multi-user.target
```

To support displaying the camera video inside Jupyter Notebook:

```
pip3 install ipywidgets
```

A.4 Deploy the Demo Files

Clone and deploy the demo files and face images

```
git clone https://github.com/rsand-temple/minipupperstuff
cd minipupperstuff
cp *.ipynb ~
cp *.py ~
unzip faces.zip -d ~
```

APPENDIX B
SETUP FOR FACE RECOGNITION

This section is to set up the prerequisites for performing facial recognition in python with *OpenCV*.

B.1 Prerequisites

Install some packages we're going to need:

```
sudo apt-get install libopenblas-dev liblapack-dev libjpeg-dev
```

B.2 Install Face Recognition for Python

For all controllers OTHER THAN NVidia Jetson boards, install DLib and *face_recognition* via pip3. This can take around 30 minutes to install. For NVidia Jetson boards ONLY, we compile and install from source (see next subsection below).

```
pip3 install dlib face_recognition
```

B.2.1 Fix Necessary for NVidia CUDA support only

If you are following this tutorial on a robot other than the Mini Pupper 1 and are using an NVidia controller such as the Jetson Nano, follow these instructions to fix a minor problem in the CUDA support in DLib.

```
wget http://dlib.net/files/dlib-19.17.tar.bz2
tar jxvf dlib-19.17.tar.bz2
cd dlib-19.17/dlib/cuda/
```

Edit the file *cudnn_dlibapi.cpp* with the editor of your choice. Find the line below - it should be line 854 - and comment out:

```
//forward_algo = forward_best_algo;
```

Return back to top dlib-19.17 folder and continue the install. This can take a LONG time (hours) to complete.

```
cd ~/dlib-19.17
sudo python3 setup.py install
```

Finally, install the face recognition python package:

```
pip3 install face_recognition
```

B.3 Training

The face images are now in the folder $\sim$ */docs*. If you want to add your own image, place here as well. The file name is used as the label.

Now we run the training:

```
python train.py
```

APPENDIX C
SETUP FOR OBJECT DETECTION

These instructions are adapted for the latest Mini Pupper 1 from the following tutorial: [9]

https://github.com/EdjeElectronics/TensorFlow-Object-Detection-on-the-Raspberry-Pi

C.1 Install tensorflow and tflite

```
pip3 install tensorflow
pip3 install tflite-runtime
```

C.2 Install protobuf

```
sudo apt-get install protobuf-compiler
```

C.3 Install tensorflow

Now that we've installed all the packages, we need to set up the TensorFlow directory. Download the tensorflow repository from GitHub by issuing:

```
cd
mkdir tensorflow1
cd tensorflow1
git clone --depth 1 https://github.com/tensorflow/models.git
```

Edit .bashrc and append the following at the bottom.

```
export PYTHONPATH=$PYTHONPATH:~/tensorflow1/models/research:~/tensorflow1/models/research/slim
```

After, close and reopen your shell.

C.4 Compile the proto files

We need to use *protoc* to compile the Protocol Buffer (.proto) files used by the Object Detection API:

```
cd ~/tensorflow1/models/research
protoc object_detection/protos/*.proto --python_out=.
```

C.5 Download the SSD Lite model

Now, download the SSD Lite model from the *TensorFlow Detection Model Zoo* (https://github.com/tensorflow/models/blob/master/research/object_detection/g3doc/detection_model_zoo.md).

```
cd
mkdir coco_ssd_mobilenet_v1_1.0_quant_2018_06_29
cd coco_ssd_mobilenet_v1_1.0_quant_2018_06_29/
wget http://storage.googleapis.com/download.tensorflow.org/models/tflite/coco_ssd_mobilenet_v1_1.0_quant_2018_06_29.zip
unzip coco_ssd_mobilenet_v1_1.0_quant_2018_06_29.zip
```