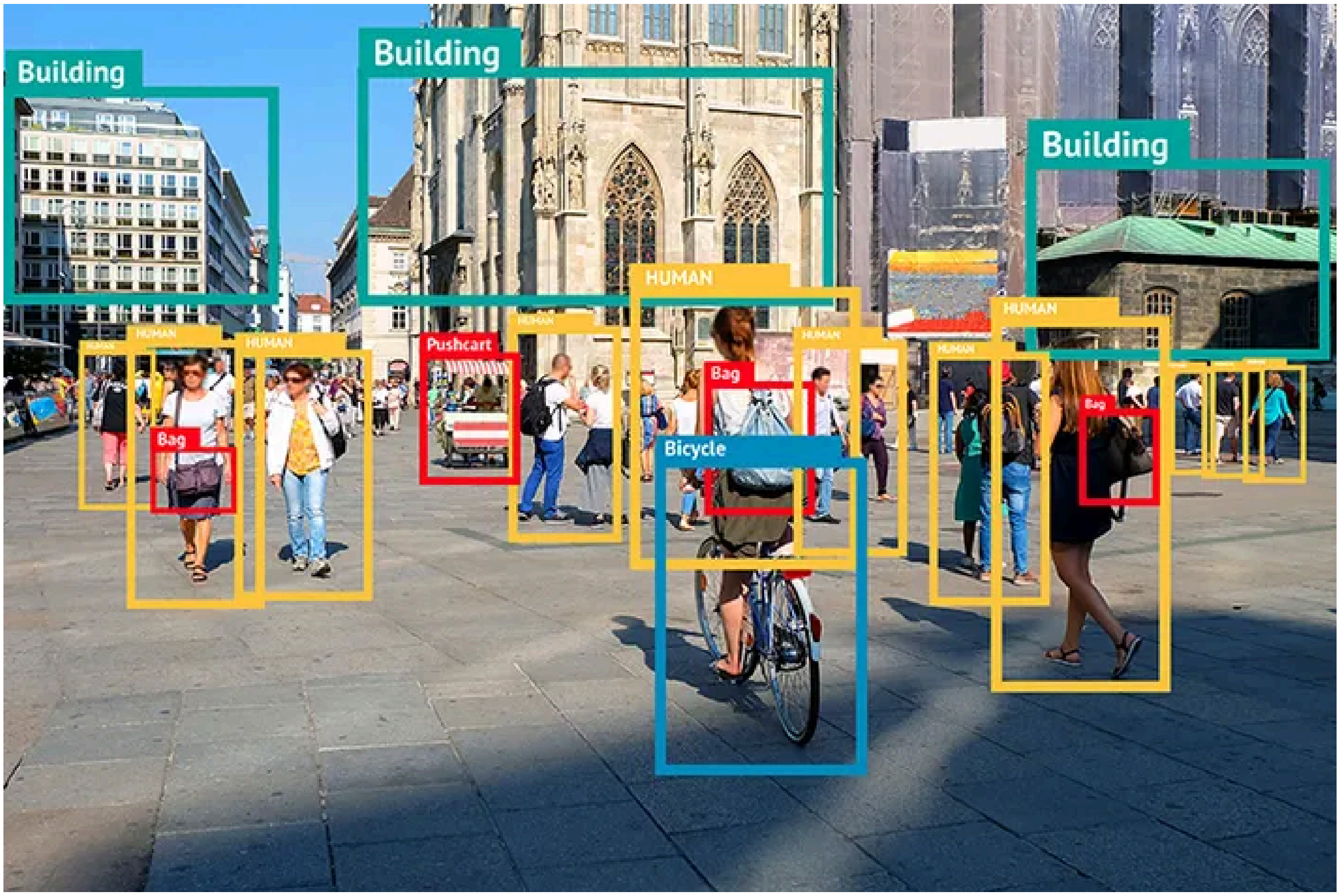
Object Detection using mobilenet SSD

Tauseef Ahmad · Follow
4 min read · Aug 24, 2022

In this article, I am sharing a step-by-step methodology to build a simple object detector using mobilenet SSD model and a webcam feed from your laptop to identify a specific object.



What is mobilenet?

Mobilenet is a type of convolutional neural network designed for mobile and embedded vision applications. Instead of using standard convolution layers, they are based on a streamlined architecture that uses depthwise separable convolutions. Using this architecture, we can build lightweight deep neural networks that have low latency for mobile and embedded devices (example: jetson nano). You can read more about the network architecture in the original paper by Google researchers in 2017. For a more in-depth explanation of depthwise separable convolutions, I also found this blog by Chi-Feng Wang quite helpful.

As mentioned above, ideal applications of mobilenet would include mobile and embedded devices. However, for demonstrating object detection using this network, we will be using a laptop and the in-built camera.

Initial requirements

For this project, we will NOT be training a mobilenet model from scratch but instead, use pre-trained model weights and model definition. In order to do so, we will need the following files to be downloaded first:

- Caffe prototxt file: Model definition is stored in this file
- Caffe model file: Pre-trained model weights

What is Caffe?

> *Caffe (Convolutional Architecture for Fast Feature Embedding) is a deep learning framework for creating image classification and image segmentation models. Initially, users can create and save their models as plain text PROTOTXT files. After the model is trained and refined using Caffe, the program saves the trained model as a CAFFEMODEL file.*

```python
# import libraries
import numpy as np
import cv2

# download the model as plain text as a PROTOTXT file and the trained model as a CAFFEMODEL file

# path to the prototxt file with text description of the network architecture
prototxt = "MobileNetSSD_deploy.prototxt"
# path to the .caffemodel file with learned network
caffe_model = "MobileNetSSD_deploy.caffemodel"

# read a network model (pre-trained) stored in Caffe framework's format
net = cv2.dnn.readNetFromCaffe(prototxt, caffe_model)

# dictionary with the object class id and names on which the model is trained
classNames = { 0: 'background',
    1: 'aeroplane', 2: 'bicycle', 3: 'bird', 4: 'boat',
    5: 'bottle', 6: 'bus', 7: 'car', 8: 'cat', 9: 'chair',
    10: 'cow', 11: 'diningtable', 12: 'dog', 13: 'horse',
    14: 'motorbike', 15: 'person', 16: 'pottedplant',
    17: 'sheep', 18: 'sofa', 19: 'train', 20: 'tvmonitor'}
```

initial_setup_mobilenet.py hosted with ❤ by GitHub view raw

Initial set up

After downloading the above files to our working directory, we need to load the Caffe model using the OpenCV DNN function `cv2.dnn.readNetFromCaffe` . Then, we define the class labels on which the network was trained (i.e. COCO labels). Our model was trained on 21 object classes which are passed as a dictionary where each key represents the class ID and the respective value is the name of the label.

Set up the camera

Since in this example we are using a camera feed for object detection, we instantiate an object of the `VideoCapture` class from the OpenCV library. As an input, the `VideoCapture` class receives an index of the device we want to use. If we have a single camera connected to the computer, we pass a value of 0.

Format the input

Then, we get the height and width of the image from the camera frame that will be used later to draw bounding boxes around the detected object. Now, we need to transform the image into a blob (which is a 4D NumPy array object — images, channels, width, height) using `cv2.dnn.blobFromImage` function. This is required to prepare the input image in the required format for the model intake. To learn more about what is blob and how the `cv2.dnn.blobFromImage` function works, refer to this blog. The input parameters of this function depend on the model that is being used. For mobilenet input parameters to the `cv2.dnn.blobFromImage` function, refer to this link from OpenVINO (an open-source toolkit for deploying AI inference).

```
# capture the webcam feed
cap = cv2.VideoCapture(0)

while True:
    ret, frame = cap.read()

    # size of image
    width = frame.shape[1]
    height = frame.shape[0]
    # construct a blob from the image
    blob = cv2.dnn.blobFromImage(frame, scalefactor = 1/127.5, size = (300, 300), mean = (127.5,
    # blob object is passed as input to the object
    net.setInput(blob)
    # network prediction
    detections = net.forward()
```


Format input

Object detection and visualization

The blob object is then set as input to the `net` network followed by a forward pass through the mobilenet network. Now, we loop over the detections — the detection summary is an array in the format 1, 1, N, 7 where N is the number of detected bounding boxes. Each detection has the format [`image_id`, `label`, `conf`, `x_min`, `y_min`, `x_max`, `y_max`]

```
image_id: ID of the image in the batch
label: predicted class ID
conf: confidence score of the predicted class
x_min, y_min: coordinates of the top left bounding box corner
x_max, y_max: coordinates of the bottom right bounding box corner

Note: Coordinates are in normalized format, in range [0, 1])
```

Next, we extract the confidence score of the detected object(s) from the third element `detections[0, 0, i, 2]` in the detection array. If the confidence score of the detected class is greater than the threshold confidence level (to filter out weak predictions), we get the class id of the detected class from the second element `detections[0, 0, i, 1]` in the detection array.

```python
    # detections array is in the format 1,1,N,7, where N is the #detected bounding boxes
    # for each detection, the description (7) contains : [image_id, label, conf, x_min, y_min, x_
    for i in range(detections.shape[2]):
        # confidence of prediction
        confidence = detections[0, 0, i, 2]
        # set confidence level threshold to filter weak predictions
        if confidence > 0.5:
            # get class id
            class_id = int(detections[0, 0, i, 1])
            # scale to the frame
            x_top_left = int(detections[0, 0, i, 3] * width)
            y_top_left = int(detections[0, 0, i, 4] * height)
            x_bottom_right   = int(detections[0, 0, i, 5] * width)
            y_bottom_right   = int(detections[0, 0, i, 6] * height)

            # draw bounding box around the detected object
            cv2.rectangle(frame, (x_top_left, y_top_left), (x_bottom_right, y_bottom_right),
                              (0, 255, 0))

            if class_id in classNames:
                # get class label
                label = classNames[class_id] + ": " + str(confidence)
                # get width and text of the label string
                (w, h),t = cv2.getTextSize(label, cv2.FONT_HERSHEY_SIMPLEX, 0.5, 1)
                y_top_left = max(y_top_left, h)
                # draw bounding box around the text
                cv2.rectangle(frame, (x_top_left, y_top_left - h),
                                   (x_top_left + w, y_top_left + t), (0, 0, 0), cv2.FILLED)
                cv2.putText(frame, label, (x_top_left, y_top_left),
                                cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0, 255, 0))
```

mobilenet_obj_detect.py hosted with ❤ by GitHub view raw

Object detection and visualization

Once the object is detected, we now try to visualize it by drawing a bounding box and adding the label of that object. The detection array returns the normalized top left and bottom right corner coordinates which are scaled to the frame dimension by multiplying with the width and height of the captured frame from the camera. Then, we draw a bounding box around the detected object using the `cv2.rectangle` function. If the detected class id matches with one of the 21 labels mentioned in the `classNames` dictionary, we add a text with the name of the label and add a box around the text.

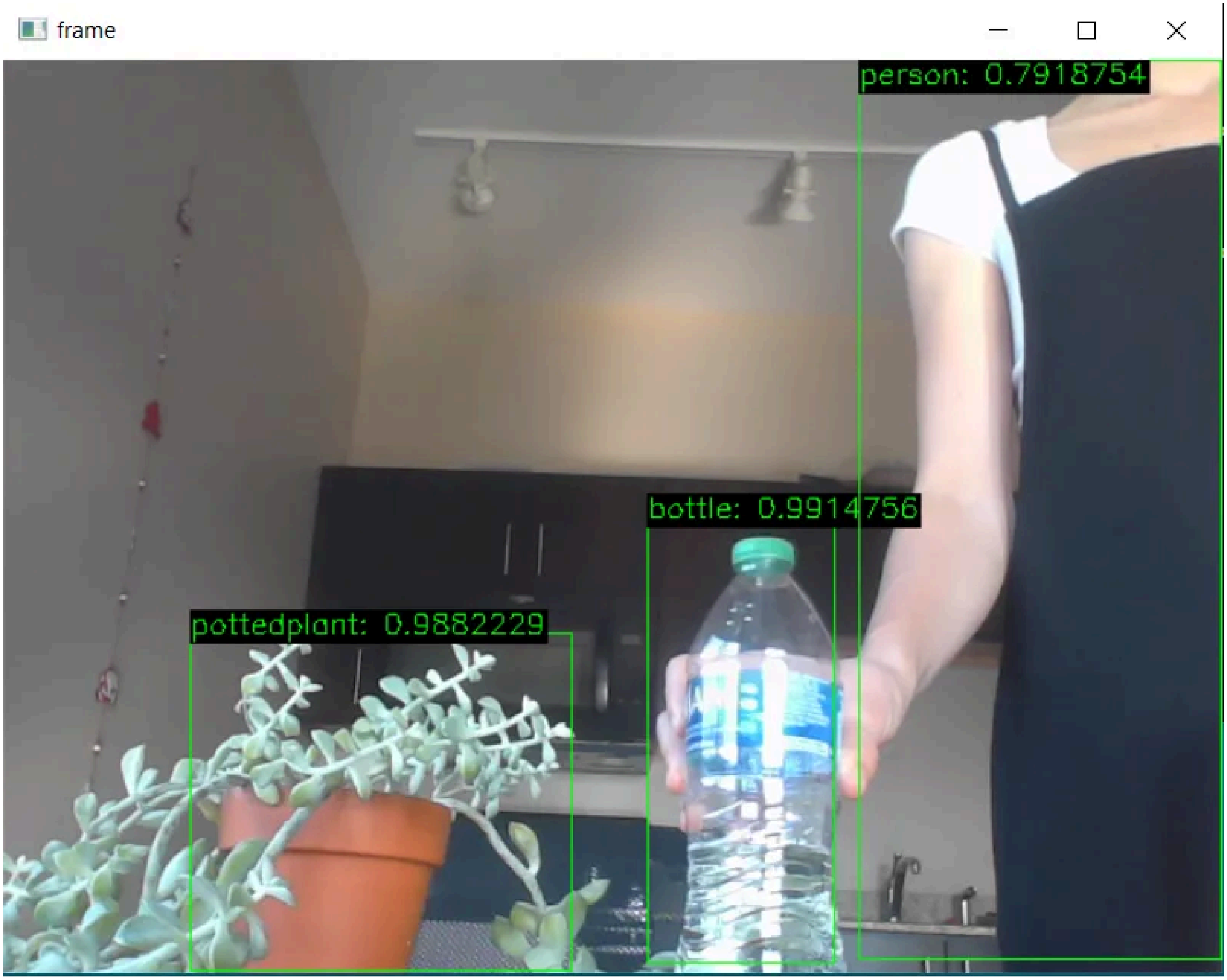
```
    cv2.namedWindow("frame", cv2.WINDOW_NORMAL)
    cv2.imshow("frame", frame)
    if cv2.waitKey(1) >= 0:  # Break with ESC
        break

cap.release()
cv2.destroyAllWindows()
```

close_camera.py hosted with ❤ by GitHub view raw

Display image and close camera feed

Now, we can go run our code from the python terminal and it will open a camera feed with a bounding box and text label of the detected object along with the confidence score in the frame.



Camera feed

If we want to close our camera feed, simply press any key from the keyboard.

References:

1. https://pyimagesearch.com/2018/05/14/a-gentle-guide-to-deep-learning-object-detection/
2. https://opencv-tutorial.readthedocs.io/en/latest/index.html
3. https://ebenezertechs.com/mobilenet-ssd-using-opencv-3-4-1-deep-learning-module-python/
4. https://github.com/djmv/MobilNet_SSD_opencv

Mlearning.ai Submission Suggestions

How to become a writer on Mlearning.ai

medium.com

Machine Learning | Artificial Intelligence | Object Detection | Computer Vision | Ml So Good

Written by Tauseef Ahmad

ML and computer vision enthusiast

More from Tauseef Ahmad