**Object Detection with SSD Mobile Net Model**

Object detection is a computer vision technique for locating instances of objects in images or videos. Object detection algorithms typically leverage machine learning or deep learning to produce meaningful results. The goal of object detection is to replicate this intelligence using a computer.

To train a custom object detection model with the Tensorflow Object Detection API, you need to go through the following steps:

1. **Image Labeling**
2. **Creation of tfrecord files**
3. **Environment setup for object detection**
4. **Configuring files**
5. **Train and export interference graph**
6. **Model Testing**
7. **Tflite file creation**
8. **Adding metadata to Tflite file**
9. **Errors and Solutions**
10. **Image Labeling**

Dataset (images) labeling is required for the training purpose. Install labelimg for the same. Image Labeling is the process of recognizing different entities in an image. You can recognize various entities like animals, plants, food, activities, colors, things, fictional characters, drinks etc. with Image Labeling

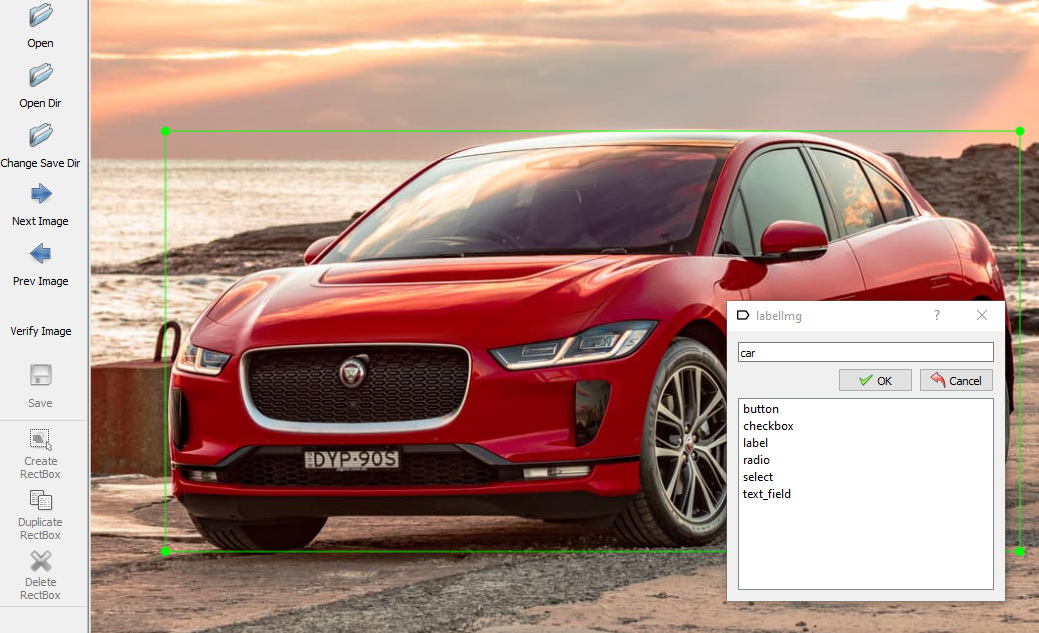
* Clone the labelimg from GitHub

$git clone https://github.com/qaprosoft/labelImg.git

$pyrcc5 -o resources.py resources.qrc

Refer <https://github.com/qaprosoft/labelImg>

* Open cmd and go to labelImg-master directory run the labelimg.py
* Create folders like images/train, images/test and Upload training images and testing images to images/train and images/test respectively.
* Open train and test directory and start labeling, Xml file will be created in respective directory.



1. **Creation of tfrecord files**

Now we need to convert these XML files to singular CSV files that can be then converted to the TFRecord files. To do this, I am going to make use of some of the code from <https://github.com/datitran/raccoon_dataset>, with some minor changes.

* First we have to convert xml to csv. Within the xml\_to\_csv script, I changed:

def main ():

image\_path = os.path.join(os.getcwd(), 'annotations')

xml\_df = xml\_to\_csv(image\_path)

xml\_df.to\_csv('raccoon\_labels.csv', index=None)

print ('Successfully converted xml to csv.')

**To:**

def main():

for directory in ['train','test']:

image\_path = os.path.join(os.getcwd(), 'images/{}'.format(directory))

xml\_df = xml\_to\_csv(image\_path)

xml\_df.to\_csv('data/{}\_labels.csv'.format(directory), index=None)

print('Successfully converted xml to csv.')

**Note:** Make sure images/\* and xml\_to\_csv.py in same directory**.**

* Run the xml\_to\_csv.py it will create test\_labels.csv and train\_lables.csv in data directory. Please verify the images path mention in the csv files are correct and absolute path is mentioned.
* Now, grab [**generate\_tfrecord.py**](https://github.com/datitran/raccoon_dataset/blob/master/generate_tfrecord.py)**.** The only modification that you will need to make here is in the class\_text\_to\_int function.

# TO-DO replace this with label map

def class\_text\_to\_int(row\_label):

if row\_label == 'car':

return 1

elif row\_label == 'bike':

return 2

else:

None

* Run generate\_tfrecord.py with the below command

$python generate\_tfrecord.py --csv\_input=data/train\_labels.csv -- output\_path=data/train.record

$python generate\_tfrecord.py --csv\_input=data/test\_labels.csv -- output\_path=data/test.record

**Note:** Given path in create\_tf\_example function should correct for test and train cases.

1. **Environment setup for object detection**

* Object detection is a [computer vision](https://heartbeat.fritz.ai/the-5-computer-vision-techniques-that-will-change-how-you-see-the-world-1ee19334354b) technique whose aim is to detect objects such as cars, buildings, and human beings, just to mention a few. The objects can generally be identified from [either pictures or video feeds](https://heartbeat.fritz.ai/detecting-objects-in-videos-and-camera-feeds-using-keras-opencv-and-imageai-c869fe1ebcdb).
* Create environment, for this create a directory “computer\_vision”

$mkdir computer\_vision

$cd computer\_vision

* Upgrade your pip. pip version must be greater than version 19.0

$pip install --upgrade pip

* The next step is to make sure that we have all the libraries and modules that we need to run the object detector on our machine.
* Here is a list of libraries that the project depends on. (Most of the dependencies comes with Tensorflow by default)
* Pycocotools
* pillow
* Cython
* Contextlib2
* Scipy
* pyyaml
* livs
* tf\_slim
* tf-models-official
* In case if you find any of the module missing just execute pip install in your environment to install.

**Note:** Tensorflow package will be installed with tf-model-official

* Clone the models from GitHub

$git clone <https://github.com/tensorflow/models.git>

$cd XXX//models/research

* Execute the following command from the research/ directory to compile the Protocol Buffer.

$protoc object\_detection/protos/\*.proto --python\_out=.

* PYTHONPATH is an environment variable which you can set to add additional directories where python will look for modules and packages. For most installations, you should not set these variables since they are not needed for Python to run. Python knows where to find its standard library.

$export PYTHONPATH=$PYTHONPATH:'pwd':'pwd'/slim

* Create setup.py file in research directory with below code.

"""Setup script for object\_detection."""

from setuptools import find\_packages

from setuptools import setup

REQUIRED\_PACKAGES = ['Pillow>=1.0', 'Matplotlib>=2.1', 'Cython>=0.28.1']

setup(

name='object\_detection',

version='0.1',

install\_requires=REQUIRED\_PACKAGES,

include\_package\_data=True,

packages=[p for p in find\_packages() if p.startswith('object\_detection')],

description='Tensorflow Object Detection Library',

)

$ python setup.py build

$ python setup.py install

* Now we have all the dependencies installed, let’s use implement custom Object Detection with ssd models.

<https://github.com/tensorflow/models/blob/master/research/object_detection/g3doc/tf2_detection_zoo.md>

Above link will give the list of object detection model, Hear I took [SSD MobileNet V2 FPNLite 640x640](http://download.tensorflow.org/models/object_detection/tf2/20200711/ssd_mobilenet_v2_fpnlite_640x640_coco17_tpu-8.tar.gz) because it is very less size and speed (ms) 39 and mean average precision (mAP) 28.2 with coco data set.

Best model should have high mAP.

$cd XXX/computer\_vision/models/research/object\_detection

$wget “paste model URL and run”

$ tar –xvzf path/ssd\_mobilenet\_v2\_fpnlite\_640x640\_coco17\_tpu-8.tar.gz

* After extracting ssd\_mobilenet\_v2\_fpnlite\_640x640\_coco17\_tpu-8.tar.gz, it gives two directories (checkpoint, saved model) and one file (Pipeline.config)

1. **Configuring files**

* Create pbtxt file, it links labels to some integer values. The Tensor Flow **Object Detection** API needs this **file** for training and **detection** purposes

$ mkdir training

$ cd training

$ touch object-detection.pbtxt

Open pbtxt file and write below

item {

id: 1

name: 'car'

}

item {

id: 2

name: 'bike'

}

* For custom object detection eight changes have to make in pipline.config

1. num\_classes: 2 line no 3

Change the number of classes to number of objects you want to detect.

My case I am detecting two and this changes can done in line no 3

1. batch\_size: 10

Change the batch size as you wish, if we give high batch size we get memory error, this changes can be done in line no 135

1. fine\_tune\_checkpoint: “ssd\_mobilenet\_v2\_fpnlite\_640x640\_coco17\_ tpu-8/checkpoint/ckpt-0"

Change fine\_tune\_checkpoint to the path of the model.ckpt file, this changes done in line no 165

1. fine\_tune\_checkpoint\_type: "detection"

Change fine\_tune\_checkpoint\_type to detection line no 171

1. label\_map\_path: "training/object-detection.pbtxt" line no 175
2. input\_path: "Path/data/train.record”

Change input\_path of the train\_input\_reader to the path of the train.record, this changes done in line no 177

1. label\_map\_path: " training/object-detection.pbtxt " line in 185
2. input\_path: "Path/data/test.record"

Input\_path of the eval\_input\_reader to the path of the test.record, this

Changes done in line no 189

* **Note:** Line numbers will change in different models

1. **Train and Export inference graph**

* Run model\_main\_tf2.py which is there in research/object\_detection
* $python model\_main\_tf2.py --pipeline\_config\_path=path/pipeline.config

--model\_dir=path/training/training\_check\_point -- num\_train\_steps=10000 --sample\_1\_of\_n\_eval\_example=1 --num\_eval\_steps=1

**Note:** Model\_dir indicates check points saving path and num\_train\_steps indicates number of epochs.

* To make it easier to use and deploy your model, I recommend converting it to a frozen graph file. This can be done with exporter\_main\_v2.py script.

$python exporter\_main\_v2.py --input\_type image\_tensor --pipeline\_config\_path path/pipeline.config --trained\_checkpoint\_dir path/training/training\_check\_point --output\_directory path/object\_detection/exported-mobile-model

Note: Results will be saved in exported-mobile-model directory

* To convert object detection model to tensorflowlite format, first export with export\_tflite\_graph\_tf2.py script.

$python export\_tflite\_graph\_tf2.py --pipeline\_config\_path path/pipeline.config --trained\_checkpoint\_dir path/training/training\_check\_point --output\_directory path/object\_detection/model\_dir

Note: At this time only SSD model can support tflite format, refer this

https://github.com/tensorflow/models/blob/master/research/object\_detection/g3doc/running\_on\_mobile\_tensorflowlite.md

1. **Model Testing**

* For testing the model we should follow object\_detection\_tutorial.ipynb file, this file is use to test the predefine model. But here we are testing our custom model with some changes.

**Link:**

<https://github.com/tensorflow/models/blob/master/research/object_detection/colab_tutorials/object_detection_tutorial.ipynb>

* Install object detection api

$pip install tensorflow-object-detection-api

* Open above link and import the given libraries and change the model\_dir in load\_model function with your model directory i.e. exported-mobile-model/saved\_model
* Change PATH\_TO\_LABELS and PATH\_TO\_TEST\_IMAGES\_DIR and run, you will see the resulted images.

1. **Tflite file creation**

* TensorFlow Lite is Tensor Flow’s lightweight solution for mobile and embedded devices. It lets you run machine-learned models on mobile devices with low latency, so you can take advantage of them to do classification, regression and computer vision models. This Tflite model we can directly integrate with mobile application.
* Below python code is you to convert export\_tflite\_graph\_tf2 to Tflite format.

import tensorflow as tf

model = tf.saved\_model.load("path/model\_dir/saved\_model")

model.signatures[tf.saved\_model.DEFAULT\_SERVING\_SIGNATURE\_DEF\_KEY]

.inputs[0].set\_shape([1, 640, 640, 3])  # height and width in pipline.config file

tf.saved\_model.save(model, "saved\_model\_updated", signatures=model .signatures[tf.saved\_model.DEFAULT\_SERVING\_SIGNATURE\_DEF\_KEY])

converter = tf.lite.TFLiteConverter.from\_saved\_model(saved\_model\_dir=

'path/model\_dir/saved\_model', signature\_keys=['serving\_default'])

converter.optimizations = [tf.lite.Optimize.DEFAULT]

converter.target\_spec.supported\_ops = [tf.lite.OpsSet.TFLITE\_BUILTINS,

tf.lite.OpsSet.SELECT\_TF\_OPS]

tflite\_model = converter.convert()

open("path/to/save/ssd\_mobile\_640.tflite", "wb").write(tflite\_model)

1. **Adding metadata to Tflite file**

* TensorFlow Lite metadata provides a standard for model descriptions. The metadata is an important source of knowledge about what the model does and its input / output information.

Follow this link to fill metadata:

https://www.tensorflow.org/lite/convert/metadata

* There are three parts to the model metadata in the [schema](https://github.com/tensorflow/tflite-support/blob/master/tensorflow_lite_support/metadata/metadata_schema.fbs):

1. **Model information** - Overall description of the model as well as items such as license terms.
2. **Input information** - Description of the inputs and pre-processing required such as normalization.
3. **Output information** - Description of the output and post-processing required such as mapping to labels.

* Before adding metadata to your model, you will need to tflite-support

$pip install tflite-support

$ touch path/labelmap.txt

* Write the labels in labelmap.txt, for my case I am using two class labels

car

bike

**Note:** Whatever label names we written in labelmap.txt that will display when object is detected. Tflite model and labelmap.txt both required to integrate with mobile application.

1. **Errors and Solutions**

* [AttributeError: module 'tensorflow.python.keras.utils.generic\_utils' has no attribute 'populate\_dict\_with\_module\_objects'](https://stackoverflow.com/questions/61137954/attributeerror-module-tensorflow-python-keras-utils-generic-utils-has-no-attr)

Follow this link:

<https://stackoverflow.com/questions/61137954/attributeerror-module-tensorflow-python-keras-utils-generic-utils-has-no-attr>

pip install tensorflow==2.1.0

* <https://stackoverflow.com/questions/63899220/attributeerror-module-tensorflow-core-keras-layers-experimental-preprocessing>

Follow this link:

<https://stackoverflow.com/questions/63899220/attributeerror-module-tensorflow-core-keras-layers-experimental-preprocessing>

pip install tensorflow –upgrade

* For memory reduce the batch size in pipeline.config