TECHZIZOU

Home » Blog » Training a model for custom object detection (TF 2.x) on Google Colab

Artificial Intelligence Machine Learning Object Detection Tensorflow

Training a model for custom object detection (TF 2.x) on Google Colab

18 min read

Using TensorFlow Object Detection API

English

In this tutorial, I will be training a Deep Learning model for custom object detection using **TensorFlow 2.x** on Google Colab. Following is the roadmap for it.

Roadmap

- Collect the dataset of images and label them to get their XML files.
- Install the TensorFlow Object Detection API.
- Generate the TFRecord files required for training. (need generate_tfrecord.py script and CSV files for this)
- Edit the model pipeline config file and download the pre-trained model checkpoint.
- Train and evaluate the model.

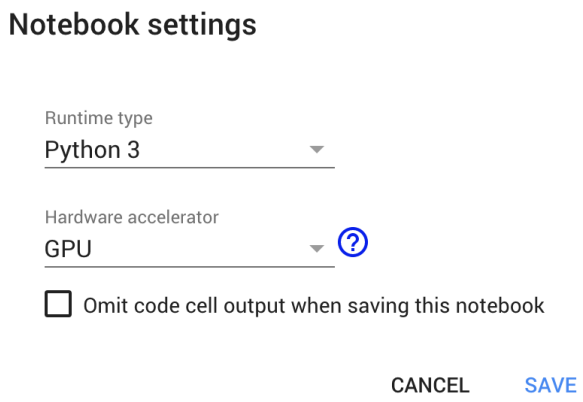
Here, I am training a model for custom object detection (mask). This is done in 16 steps mentioned below:

(But first ✅Subscribe to my YouTube channel 👉 https://bit.ly/3Ap3sdi 😄😜)

1. Import Libraries
2. Create customTF2, training, and data folders in your google drive
3. Create and upload your image files and XML files
4. Upload the generate_tfrecord.py file to the customTF2 folder on your drive
5. Mount drive and link your folder
6. Clone the TensorFlow models git repository & Install TensorFlow Object Detection API
7. Test the model builder
8. Navigate to */mydrive/customTF2/data/* and Unzip the images.zip and annotations.zip files into the data folder
9. Create test_labels & train_labels
10. Create_CSV and "label_map.pbtxt" files
11. Create 'train.record' & 'test.record' files
12. Download pre-trained model checkpoint
13. Get the model pipeline config file, make changes to it, and put it inside the data folder
14. Load Tensorboard
15. Train the model
16. Test your trained model

HOW TO BEGIN?

- Open my Colab notebook on your browser.
- Click on **File** in the menu bar and click on **Save a copy in drive**. This will open a copy of my Colab notebook on your browser which you can now use.
- Next, once you have opened the copy of my notebook and are connected to the Google Colab VM, click on **Runtime** in the menu bar and click on **Change runtime type**. Select **GPU** and click on save.



LET'S BEGIN !!

---

1) Import Libraries

```
import os
import glob
import xml.etree.ElementTree as ET
import pandas as pd
import tensorflow as tf
```

---

2) Create customTF2, training, and data folders in your google drive

Create a folder named *customTF2* in your google drive.

Create another folder named *training* inside the *customTF2* folder (*training* folder is where the checkpoints will be saved during training).

Create another folder named *data* inside the *customTF2* folder.

---

3) Create and upload your image files and their corresponding labeled XML files.

Create a folder named *images* for your custom dataset images and create another folder named *annotations* for its corresponding PASCAL_VOC format labeled XML files.

Next, create their zip files and upload them to the *customTF2* folder in your drive.

> **Make sure all the image files have their extension as ".jpg" only.**
>
> Other formats like **<.png> , <.jpeg>** will give errors since the *generate_tfrecord* and *xml_to_csv* scripts here have only **<.jpg>** in them. If you have other format images, you can make changes in the scripts accordingly.

For datasets, you can check out my Dataset Sources at the bottom of this article in the credits section.

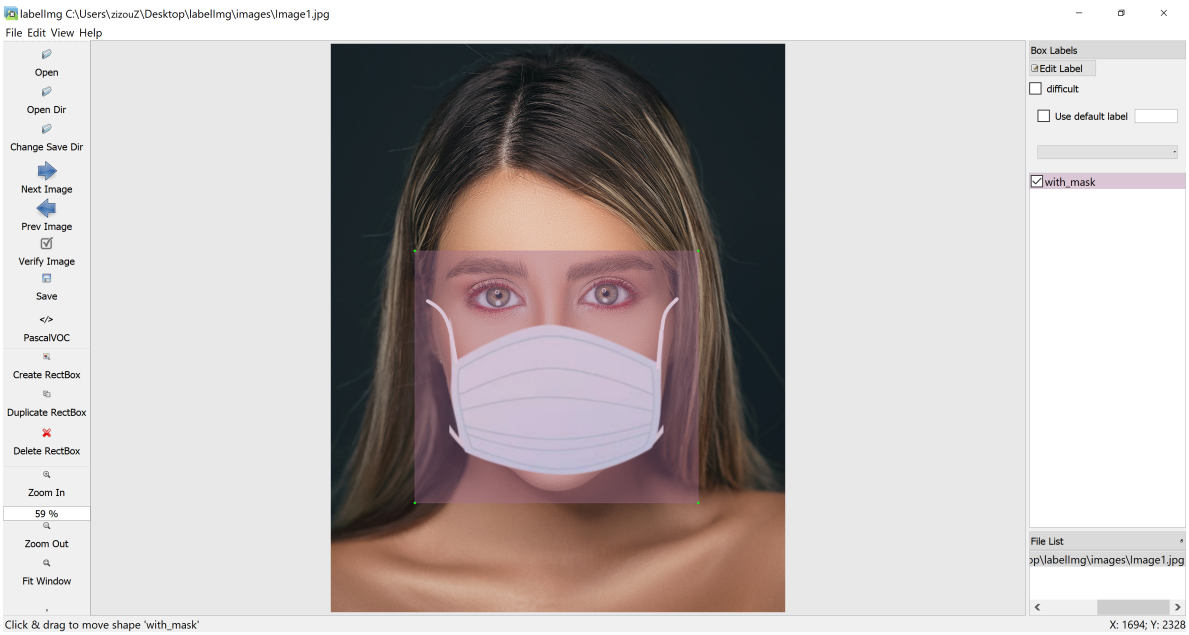
Collecting Images Dataset and labeling them to get their PASCAL_VOC XML annotations.

Labeling your Dataset

Input image example (**Image1.jpg**)

Original Photo by Ali Pazani from Pexels

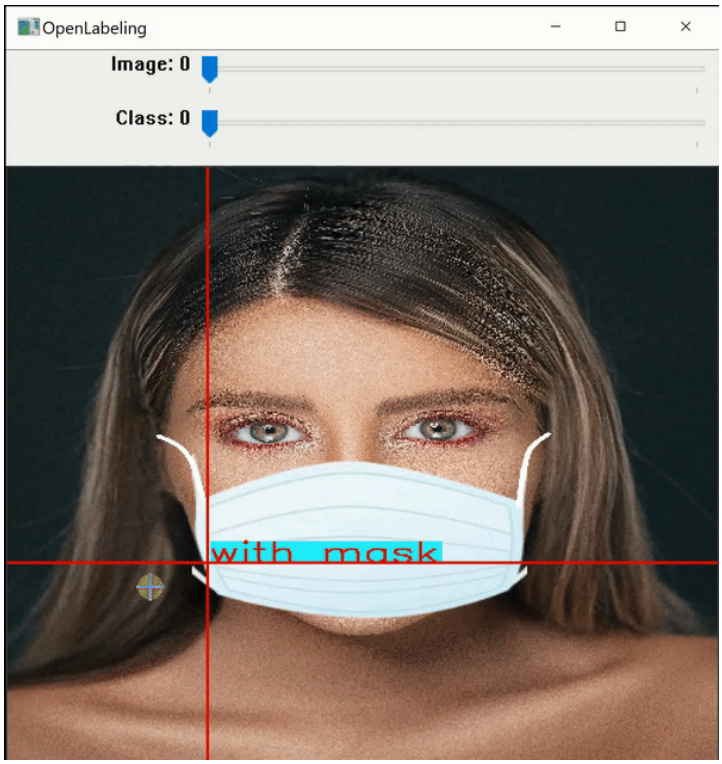
You can use any software for labeling like the labelImg tool.



labelImg GUI for Image1.jpg

I use an open-source labeling tool called **OpenLabeling** with a very simple UI.

English



OpenLabeling Tool GUI

Click on the link below to know more about the labeling process and other software for it:

- Image Dataset Labeling article

NOTE : Garbage In = Garbage Out. Choosing and labeling images is the most important part. Try to find good quality images. The quality of the data goes a long way towards determining the quality of the result.

The output PASCAL_VOC labeled XML file looks like as shown below:

```
<?xml version="1.0"?>
- <annotation>
    <folder>input</folder>
    <filename>Image1.jpg</filename>
    <path>C:\Users\zizou\Desktop\annotation-tool\input\Image1.jpg</path>
  - <source>
      <database>Unknown</database>
    </source>
  + <size>
    <segmented>0</segmented>
  - <object>
      <name>with_mask</name>
      <pose>Unspecified</pose>
      <truncated>0</truncated>
      <difficult>0</difficult>
    - <bndbox>
        <xmin>243</xmin>
        <ymin>124</ymin>
        <xmax>354</xmax>
        <ymax>204</ymax>
      </bndbox>
    </object>
</annotation>
```

Image1.xml

---

4) Upload the generate_tfrecord.py file to the customTF2 folder in your drive.

You can find the generate_tfrecord.py file here

```
from __future__ import division
from __future__ import print_function
from __future__ import absolute_import

import os
import io
import pandas as pd
import tensorflow as tf
import argparse

from PIL import Image
from tqdm import tqdm
from object_detection.utils import dataset_util
from collections import namedtuple, OrderedDict
```

```
15
16
17  def __split(df, group):
18      data = namedtuple('data', ['filename', 'object'])
19      gb = df.groupby(group)
20      return [data(filename, gb.get_group(x)) for filename, x in zip(gb.groups.keys(), gb.group
21
22
23  def create_tf_example(group, path, class_dict):
24      with tf.io.gfile.GFile(os.path.join(path, '{}'.format(group.filename)), 'rb') as fid:
25          encoded_jpg = fid.read()
26      encoded_jpg_io = io.BytesIO(encoded_jpg)
27      image = Image.open(encoded_jpg_io)
28      width, height = image.size
29
30      filename = group.filename.encode('utf8')
31      image_format = b'jpg'
32      xmins = []
33      xmaxs = []
34      ymins = []
35      ymaxs = []
36      classes_text = []
37      classes = []
38
39      for index, row in group.object.iterrows():
40          if set(['xmin_rel', 'xmax_rel', 'ymin_rel', 'ymax_rel']).issubset(set(row.index)):
41              xmin = row['xmin_rel']
42              xmax = row['xmax_rel']
43              ymin = row['ymin_rel']
44              ymax = row['ymax_rel']
45
46          elif set(['xmin', 'xmax', 'ymin', 'ymax']).issubset(set(row.index)):
47              xmin = row['xmin'] / width
48              xmax = row['xmax'] / width
49              ymin = row['ymin'] / height
50              ymax = row['ymax'] / height
51
52          xmins.append(xmin)
53          xmaxs.append(xmax)
54          ymins.append(ymin)
55          ymaxs.append(ymax)
56          classes_text.append(str(row['class']).encode('utf8'))
57          classes.append(class_dict[str(row['class'])])
58
59      tf_example = tf.train.Example(features=tf.train.Feat
```

```
        feature={
            'image/height': dataset_util.int64_feature(height),
            'image/width': dataset_util.int64_feature(width),
            'image/filename': dataset_util.bytes_feature(filename),
            'image/source_id': dataset_util.bytes_feature(filename),
            'image/encoded': dataset_util.bytes_feature(encoded_jpg),
            'image/format': dataset_util.bytes_feature(image_format),
            'image/object/bbox/xmin': dataset_util.float_list_feature(xmins),
            'image/object/bbox/xmax': dataset_util.float_list_feature(xmaxs),
            'image/object/bbox/ymin': dataset_util.float_list_feature(ymins),
            'image/object/bbox/ymax': dataset_util.float_list_feature(ymaxs),
            'image/object/class/text': dataset_util.bytes_list_feature(classes_text),
            'image/object/class/label': dataset_util.int64_list_feature(classes), }))
    return tf_example


def class_dict_from_pbtxt(pbtxt_path):
    # open file, strip \n, trim lines and keep only
    # lines beginning with id or display_name

    with open(pbtxt_path, 'r', encoding='utf-8-sig') as f:
        data = f.readlines()

    name_key = None
    if any('display_name:' in s for s in data):
        name_key = 'display_name:'
    elif any('name:' in s for s in data):
        name_key = 'name:'

    if name_key is None:
        raise ValueError(
            "label map does not have class names, provided by values with the 'display_name' o
        )

    data = [l.rstrip('\n').strip() for l in data if 'id:' in l or name_key in l]

    ids = [int(l.replace('id:', '')) for l in data if l.startswith('id')]
    names = [
        l.replace(name_key, '').replace('"', '').replace("'", '').strip() for l in data
        if l.startswith(name_key)]

    # join ids and display_names into a single dictionary
    class_dict = {}
    for i in range(len(ids)):
        class_dict[names[i]] = ids[i]
```

```
105
106     return class_dict
107
108
109 if __name__ == '__main__':
110     parser = argparse.ArgumentParser(
111         description='Create a TFRecord file for use with the TensorFlow Object Detection API.
112         formatter_class=argparse.RawDescriptionHelpFormatter)
113     parser.add_argument('csv_input', metavar='csv_input', type=str, help='Path to the CSV inp
114     parser.add_argument('pbtxt_input',
115                         metavar='pbtxt_input',
116                         type=str,
117                         help='Path to a pbtxt file containing class ids and display names')
118     parser.add_argument('image_dir',
119                         metavar='image_dir',
120                         type=str,
121                         help='Path to the directory containing all images')
122     parser.add_argument('output_path',
123                         metavar='output_path',
124                         type=str,
125                         help='Path to output TFRecord')
126
127     args = parser.parse_args()
128
129     class_dict = class_dict_from_pbtxt(args.pbtxt_input)
130
131     writer = tf.compat.v1.python_io.TFRecordWriter(args.output_path)
132     path = os.path.join(args.image_dir)
133     examples = pd.read_csv(args.csv_input)
134     grouped = __split(examples, 'filename')
135
136     for group in tqdm(grouped, desc='groups'):
137        tf_example = create_tf_example(group, path, class_dict)
138        writer.write(tf_example.SerializeToString())
139
140     writer.close()
141     output_path = os.path.join(os.getcwd(), args.output_path)
```

generate_tfrecord.py hosted with ❤ by GitHub view raw

5) Mount drive and link your folder

```
#mount drive

from google.colab import drive
drive.mount('/content/gdrive')

# this creates a symbolic link so that now the path /content/gdrive/My Drive/ is equal to /mydrive

!ln -s /content/gdrive/My Drive/ /mydrive
!ls /mydrive
```

6) Clone the TensorFlow models git repository & Install TensorFlow Object Detection API

```
# clone the tensorflow models on the colab cloud vm

!git clone --q https://github.com/tensorflow/models.git

# navigate to /models/research folder to compile protos

%cd models/research

# Compile protos.

!protoc object_detection/protos/*.proto --python_out=.

# Install TensorFlow Object Detection API.

!cp object_detection/packages/tf2/setup.py .
!python -m pip install
```

7) Test the model builder

```
!python object_detection/builders/model_builder_tf2_test.py
```

8) Navigate to */mydrive/customTF2/data/* and Unzip the images.zip and annotations.zip files into the data folder

```
%cd /mydrive/customTF2/data/

# unzip the datasets and their contents so that they are now in
/mydrive/customTF2/data/ folder

!unzip /mydrive/customTF2/images.zip -d .
!unzip /mydrive/customTF2/annotations.zip -d .
```

---

9) Create test_labels & train_labels

Current working directory is */mydrive/customTF2/data/*

Divide annotations into test_labels(20%) and train_labels(80%).

```
#creating two dir for training and testing
!mkdir test_labels train_labels

# lists the files inside 'annotations' in a random order (not really random, by their hash va
# Moves the first 274/1370 labels (20% of the labels) to the testing dir: `test_labels`
!ls annotations/* | sort -R | head -274 | xargs -I{} mv {} test_labels/


# Moves the rest of the labels ( 1096 labels ) to the training dir: `train_labels`
!ls annotations/* | xargs -I{} mv {} train_labels/
```

divide_labels.py hosted with ❤ by GitHub view raw

English

The working directory at this point:

```
mydrive➔customTF2 /
        └── data (cwd)
            ├── images
            │   ├── image_1.jpg
            │   └── ...
            │
            ├── annotations
            │   ├──
            │   └── ...
            │
            ├── train_labels //contains the labels only
            │   ├── image_1.xml
            │   └── ...
            │
            └── test_labels //contains the labels only
                ├── image_50.xml
                └── ...
```

---

10) Create the CSV files and the "label_map.pbtxt" file

Current working directory is */mydrive/customTF2/data/*

Run xml_to_csv script below to create *test_labels.csv* and *train_labels.csv*

This script also creates the *label_map.pbtxt* file using the classes mentioned in the xml files.

```
#adjusted from: https://github.com/datitran/raccoon_dataset
def xml_to_csv(path):
  classes_names = []
  xml_list = []

  for xml_file in glob.glob(path + '/*.xml'):
    tree = ET.parse(xml_file)
    root = tree.getroot()
    for member in root.findall('object'):
      classes_names.append(member[0].text)
      value = (root.find('filename').text  ,
               int(root.find('size')[0].text),
               int(root.find('size')[1].text),
               member[0].text,
               int(member[4][0].text),
```

```
                int(member[4][1].text),
                int(member[4][2].text),
                int(member[4][3].text))
        xml_list.append(value)
    column_name = ['filename', 'width', 'height', 'class', 'xmin', 'ymin', 'xmax', 'ymax']
    xml_df = pd.DataFrame(xml_list, columns=column_name)
    classes_names = list(set(classes_names))
    classes_names.sort()
    return xml_df, classes_names

for label_path in ['train_labels', 'test_labels']:
    image_path = os.path.join(os.getcwd(), label_path)
    xml_df, classes = xml_to_csv(label_path)
    xml_df.to_csv(f'{label_path}.csv', index=None)
    print(f'Successfully converted {label_path} xml to csv.')

label_map_path = os.path.join("label_map.pbtxt")
pbtxt_content = ""

for i, class_name in enumerate(classes):
    pbtxt_content = (
        pbtxt_content
        + "item {{\n    id: {0}\n    name: '{1}'\n}}\n\n".format(i + 1, class_name)
    )
pbtxt_content = pbtxt_content.strip()
with open(label_map_path, "w") as f:
    f.write(pbtxt_content)
    print('Successfully created label_map.pbtxt ')
```

xml_to_csv.py hosted with ❤ by GitHub view raw

The working directory at this point:

```
mydrive ➔ customTF2 /
          └── data/ (cwd)
              ├── images/
              │   └── ...
              ├── annotations/
              │   └── ...
              ├── train_labels/
              │   └── ...
              ├── test_labels/
              │   └── ...
              │
              ├── label_map.pbtxt
              │
              ├── test_labels.csv
              │
              └── train_labels.csv
```

The 3 files that are created i.e. **train_labels.csv**, **test_labels.csv**, and **label_map.pbtxt** look like as shown below:

train_labels.csv

91 to 100 of 1320 entries Filter

filename	width	height	class	xmin	ymin	xmax	ymax
masked (1664).jpg	1088	725	with_mask	418	98	584	260
193-with-mask.jpg	320	428	with_mask	58	80	273	371
masked (1533).jpg	1199	800	with_mask	383	354	624	636
masked (1533).jpg	1199	800	with_mask	627	453	833	710
unmasked (1452).jpg	179	281	without_mask	51	69	134	206
201.jpg	320	433	without_mask	74	98	278	372
347-with-mask.jpg	183	275	with_mask	30	54	150	209
masked (1865).jpg	284	177	with_mask	120	26	156	69
masked (1865).jpg	284	177	with_mask	179	33	228	77
296-with-mask.jpg	320	433	with_mask	41	147	286	405

Show 10 per page 1 2 3 4 5 6 7 8 9 10 11 20 100 130 132

train_labels.csv

test_labels.csv

1 to 10 of 323 entries Filter

filename	width	height	class	xmin	ymin	xmax	ymax
masked (1558).jpg	218	218	with_mask	30	64	132	190
masked (1889).jpg	265	190	with_mask	177	57	227	107
masked (1554).jpg	1024	576	with_mask	15	105	390	571
masked (1554).jpg	1024	576	with_mask	415	151	785	442
0.jpg	180	270	without_mask	48	12	117	111
masked (1380).jpg	1280	720	with_mask	660	186	1151	707
256-with-mask.jpg	193	261	with_mask	21	72	160	230
unmasked (1432).jpg	320	428	without_mask	72	95	266	362
masked (1418).jpg	860	571	with_mask	270	223	398	358
unmasked (1415).jpg	327	433	without_mask	76	84	284	380

Show 10 per page 1 2 10 30 33

test_labels.csv

label_map.pbtxt

```
item {
    id: 1
    name: 'with_mask'
}

item {
    id: 2
    name: 'without_mask'
}
```

label_map.pbtxt

The train_labels.csv contains the name of all the train images, the classes in those images, and their annotations.

The **test_labels.csv** contains the name of all the test images, the classes in those images, and their annotations.

The **label_map.pbtxt** file contains the names of the classes from your labeled XML files.

NOTE: I have 2 classes i.e. “with_mask” and “without_mask”.

Label map id 0 is reserved for the bo

English

11) Create train.record & test.record files

Current working directory is */mydrive/customTF2/data/*

Run the *generate_tfrecord.py* script to create *train.record* and *test.record* files

```
#Usage:
#!python generate_tfrecord.py output.csv output_pb.txt /path/to/images
output.tfrecords

#For train.record
!python /mydrive/customTF2/generate_tfrecord.py train_labels.csv
label_map.pbtxt images/ train.record

#For test.record
!python /mydrive/customTF2/generate_tfrecord.py test_labels.csv
label_map.pbtxt images/ test.record
```

The total number of image files is 1370. Since we divided the labels into two categories viz. train_labels(80%) and test_labels(20%), the number of files for "**train.record**" is 1096, and the number of files for "**test.record**" is 274.

12) Download pre-trained model checkpoint

Current working directory is */mydrive/customTF2/data/*

You can choose any model for training depending upon your data and requirement. Read this blog for more info on this. The official list of detection model checkpoints for TensorFlow 2.x can be found here.

In this tutorial, I will use the **ssd_mobilenet_v2_fpnlite_320x320_coco17_tpu-8** model.

```
# Download the pre-trained model
ssd_mobilenet_v2_fpnlite_320x320_coco17_tpu-8.tar.gz into the data folder &
unzip it

!wget
http://download.tensorflow.org/models/object_detection/tf2/20200711/ssd_mobi
lenet_v2_fpnlite_320x320_coco17_tpu-8.tar.gz

!tar -xzvf ssd_mobilenet_v2_fpnlite_320x320_coco17_tpu-8.tar.gz
```

---

13) Get the model pipeline config file, make changes to it and put it inside the data folder

Current working directory is */mydrive/customTF2/data/*

Download **ssd_mobilenet_v2_fpnlite_320x320_coco17_tpu-8.config** from */content/models/research/object_detection/configs/tf2*. Make the required changes to it and upload it to the */mydrive/custom/data* folder.

OR

Edit the config file from */content/models/research/object_detection/configs/tf2* in colab vm and copy the edited config file to the */mydrive/customTF2/data* folder.

You can also find the pipeline config file inside the model checkpoint folder we just downloaded in the previous step.

You need to make the following changes:

- change *num_classes* to the number of your classes.
- change *test.record* path, *train.record* path & *labelmap* path to the paths where you have created these files (paths should be relative to your current working directory while training).
- change *fine_tune_checkpoint* to the path of the directory where the downloaded checkpoint from step 12 is.
- change *fine_tune_checkpoint_type* with value **classification** or **detection** depending on the type.

- change ***batch_size*** to any multiple of 8 depending upon the capability of your GPU. (eg:- 24,128,...,512). The better the GPU capability, the higher you can go. Mine is set to 64.
- change ***num_steps*** to the number of steps you want the detector to train.

> *Max batch size= available GPU memory bytes / 4 / (size of tensors + trainable parameters)*

Next, copy the edited config file.

```
# copy the edited config file from the configs/tf2 directory to the data/ folder in your drive

!cp /content/models/research/object_detection/configs/tf2/ssd_mobilenet_v2_fpnlite_320x320_coco17_tpu-8.config /mydrive/customTF2/data
```

The workspace at this point:

```
/mydrive/
  |
  |
  └── customTF2/
       |
       ├------ data/ (cwd)
       |         ├------ images/
       |         |          ├── mask (1).jpg
       |         |          ├── mask (2).jpg
       |         |          └── ...
       |         ├------ annotations/
       |         |          ├──
       |         |          ├──
       |         |          └── ...
       |         ├------ train_labels/
       |         |          ├── mask (1).xml
       |         |          ├── mask (2).xml
       |         |          └── ...
       |         ├------- test_labels/
       |         |         ├── mask (10).xml
       |         |         ├── mask (20).xml
       |         |         └── ...
       |         ├------- label_map.pbtxt
       |         ├------- test_labels.csv
       |         ├------- train_labels.csv
       |         ├------- test.record
       |         ├------- train.record
       |         ├------- edited pipeline config
       |         └------- pre-trained model checkpoint
       |
       |
       |
       |
       ├------ training
       |           |
       |           └------ training checkpoints
       |
       |
       |
       ├------ generate_tfrecord.py
       |
       |
       ├------ images.zip
       |
       |
       └------ annotations.zip
```

There are many data augmentation options that you can add. Check the full list here. For beginners, the above changes are sufficient.

Data Augmentation Suggestions (optional)

First, you should train the model using the sample config file with the above basic changes and see how well it does. If you are overfitting, then you might want to do some more image augmentations.

In the sample config file: **`random_horizontal_flip`** & **`ssd_random_crop`** are added by default. You could try adding the following as well:

(**Note**: Each image augmentation will increase the training time drastically)

1. from **train_config {}:**

```
data_augmentation_options {
    random_adjust_contrast {
    }
  }
  data_augmentation_options {
    random_rgb_to_gray {
    }
  }
  data_augmentation_options {
    random_vertical_flip {
    }
  }
  data_augmentation_options {
    random_rotation90 {
    }
  }
  data_augmentation_options {
    random_patch_gaussian {
    }
  }
```

2. In **model {} > ssd {} > box_predictor {}:** set **`use_dropout`** to `true` This will help you to counter overfitting.

3. In **eval_config : {}** set the number of **testing** images you have in **`num_examples`** and remove **`max_eval`** to evaluate indefinitely

```
eval_config: {
  num_examples: 274 # set this to the number of test images we divided
earlier
  num_visualizations: 20 # the number of visualization to see in tensorboard
}
```

14) Load Tensorboard

```
%load_ext tensorboard
%tensorboard --logdir '/content/gdrive/MyDrive/customTF2/training'
```

---

15) Train the model

Navigate to the *object_detection* folder in Colab VM

```
%cd /content/models/research/object_detection
```

15 (a) Training using model_main_tf2.py

Here {PIPELINE_CONFIG_PATH} points to the pipeline config and {MODEL_DIR} points to the directory in which training checkpoints and events will be written.

```
# Run the command below from the content/models/research/object_detection
directory

"""
PIPELINE_CONFIG_PATH=path/to/pipeline.config
MODEL_DIR=path to training checkpoints directory
NUM_TRAIN_STEPS=50000
SAMPLE_1_OF_N_EVAL_EXAMPLES=1

python model_main_tf2.py -- \
--model_dir=$MODEL_DIR --num_train_steps=$NUM_TRAIN_STEPS \
--sample_1_of_n_eval_examples=$SAMPLE_1_OF_N_EVAL_EXAMPLES \
--pipeline_config_path=$PIPELINE_CONFIG_PATH \
--alsologtostderr
"""

!python model_main_tf2.py --
pipeline_config_path=/mydrive/customTF2/data/ssd_mobilenet_v2_fpnlite_320x32
0_coco17_tpu-8.config --model_dir=/mydrive/customTF2/training --
alsologtostderr
```

NOTE :

For best results, you should stop the training when the loss is less than 0.1 if possible, else train the model until the loss does not show any significant change for a while. The ideal loss should be below 0.05 (Try to get the loss as low as possible without overfitting the model. Don't go too high on training steps to try and lower the loss if the model has already converged viz. if it does not reduce loss significantly any further and takes a while to go down.)

Ideally, we want the loss to be as low as possible but we should be careful so that the model does not over-fit. You can set the number of steps to 50000 and check if the loss goes below 0.1 and if not, then you can retrain the model with a higher number of steps.

The output will normally look like it has "frozen", but DO NOT rush to cancel the process. The training outputs logs only every 100 steps by default, therefore if you wait for a while, you should see a log for the loss at step 100. The time you should wait can vary greatly, depending on whether you are using a GPU and the chosen value for `batch_size` in the config file, so be patient.

---

15 (b) Evaluation using model_main_tf2.py (Optional)

You can run this in parallel by opening another colab notebook and running this command simultaneously along with the training command above (don't forget to mount the drive, clone the TF git repo and install the TF2 object detection API there as well). This will give you validation loss, etc so you have a better idea of how your model is performing.

Here **{CHECKPOINT_DIR}** points to the directory with checkpoints produced by the training job. Evaluation events are written to **{MODEL_DIR/eval}**

```
# Run the command below from the content/models/research/object_detection
directory

"""
PIPELINE_CONFIG_PATH=path/to/pipeline.config
MODEL_DIR=path to training checkpoints directory
CHECKPOINT_DIR=${MODEL_DIR}
NUM_TRAIN_STEPS=50000
SAMPLE_1_OF_N_EVAL_EXAMPLES=1

python model_main_tf2.py -- \
--model_dir=$MODEL_DIR --num_train_steps=$NUM_TRAIN_STEPS \
--checkpoint_dir=${CHECKPOINT_DIR} \
--sample_1_of_n_eval_examples=$SAMPLE_1_OF_N_EVAL_EXAMPLES \
--pipeline_config_path=$PIPELINE_CONFIG_PATH \
--alsologtostderr
"""

!python model_main_tf2.py --
pipeline_config_path=/mydrive/customTF2/data/ssd_mobilenet_v2_fpnlite_320x32
0_coco17_tpu-8.config --model_dir=/mydrive/customTF2/training/ --
checkpoint_dir=/mydrive/customTF2/training/ --alsologtostderr
```

RETRAINING THE MODEL (in case you get disconnected)

If you get disconnected or lose your session on Colab VM, you can start your training where you left off as the checkpoint is saved on your drive inside the ***training*** folder. To restart the training simply run **steps 1, 5, 6, 7, 14, and 15.**

Note that since we have all the files required for training like the record files, our edited pipeline config file, the label_map file, and the model checkpoint folder, therefore we do not need to create these again.

The model_main_tf2.py script saves the checkpoint every 1000 steps. The training automatically restarts from the last saved checkpoint itself.

However, if you see that it doesn't restart training from the last checkpoint you can make 1 change in the pipeline config file. Change **fine_tune_checkpoint** to where your latest trained checkpoints have been written and have it point t
shown below:

```
fine_tune_checkpoint: "/mydrive/customTF2/training/ckpt-X" (where ckp
```

Read this TensorFlow Object Detection API tutorial to know more about the training process for TF2.

---

16) Test your trained custom object detection model

Export inference graph

Current working directory is */content/models/research/object_detection*

```
!python exporter_main_v2.py --
trained_checkpoint_dir=/mydrive/customTF2/training --
pipeline_config_path=/content/gdrive/MyDrive/customTF2/data/ssd_mobilenet_v2
_fpnlite_320x320_coco17_tpu-8.config --output_directory
/mydrive/customTF2/data/inference_graph
```

> Note: The *trained_checkpoint_dir* parameter in the above command needs the path to the training directory. There is a file called “checkpoint” which has all the model paths and the latest model checkpoint path saved in it. So it automatically uses the latest checkpoint. In my case, the checkpoint file had ckpt-36 written in it for the latest model_checkpoint_path.

> For *pipeline_config_path* give the path to the edited config file we used to train the model above.

Test your trained custom object detection model on images

Current working directory is */content/models/research/object_detection*

This step is optional.

```
# Different font-type and font-size for labels text
```

```
!wget https://freefontsdownload.net/download/160187/arial.zip
!unzip arial.zip -d .

%cd utils/
!sed -i "s/font = ImageFont.truetype('arial.ttf', 24)/font =
ImageFont.truetype('arial.ttf', 50)/" visualization_utils.py
%cd ..
```

Test your trained model

```
#Loading the saved_model
import tensorflow as tf
import time
import numpy as np
import warnings
warnings.filterwarnings('ignore')
from PIL import Image
from google.colab.patches import cv2_imshow
from object_detection.utils import label_map_util
from object_detection.utils import visualization_utils as viz_utils

IMAGE_SIZE = (12, 8) # Output display size as you want
import matplotlib.pyplot as plt
PATH_TO_SAVED_MODEL="/mydrive/customTF2/data/inference_graph/saved_model"
print('Loading model...', end='')

# Load saved model and build the detection function
detect_fn=tf.saved_model.load(PATH_TO_SAVED_MODEL)
print('Done!')

#Loading the label_map
category_index=label_map_util.create_category_index_from_labelmap("/mydrive/customTF2/data/la
#category_index=label_map_util.create_category_index_from_labelmap([path_to_label_map],use_di

def load_image_into_numpy_array(path):

    return np.array(Image.open(path))

image_path = "/mydrive/mask_test_images/image1.jpg"
#print('Running inference for {}... '.format(image_path), end='')

image_np = load_image_into_numpy_array(image_path)

# The input needs to be a tensor, convert it using `tf.c
```

```
input_tensor = tf.convert_to_tensor(image_np)
# The model expects a batch of images, so add an axis with `tf.newaxis`.
input_tensor = input_tensor[tf.newaxis, ...]

detections = detect_fn(input_tensor)

# All outputs are batches tensors.
# Convert to numpy arrays, and take index [0] to remove the batch dimension.
# We're only interested in the first num_detections.
num_detections = int(detections.pop('num_detections'))
detections = {key: value[0, :num_detections].numpy()
               for key, value in detections.items()}
detections['num_detections'] = num_detections

# detection_classes should be ints.
detections['detection_classes'] = detections['detection_classes'].astype(np.int64)

image_np_with_detections = image_np.copy()

viz_utils.visualize_boxes_and_labels_on_image_array(
      image_np_with_detections,
      detections['detection_boxes'],
      detections['detection_classes'],
      detections['detection_scores'],
      category_index,
      use_normalized_coordinates=True,
      max_boxes_to_draw=200,
      min_score_thresh=.4, # Adjust this value to set the minimum probability boxes to be cla
      agnostic_mode=False)
%matplotlib inline
plt.figure(figsize=IMAGE_SIZE, dpi=200)
plt.axis("off")
plt.imshow(image_np_with_detections)
```

object_detection_image.py hosted with ❤ by GitHub view raw

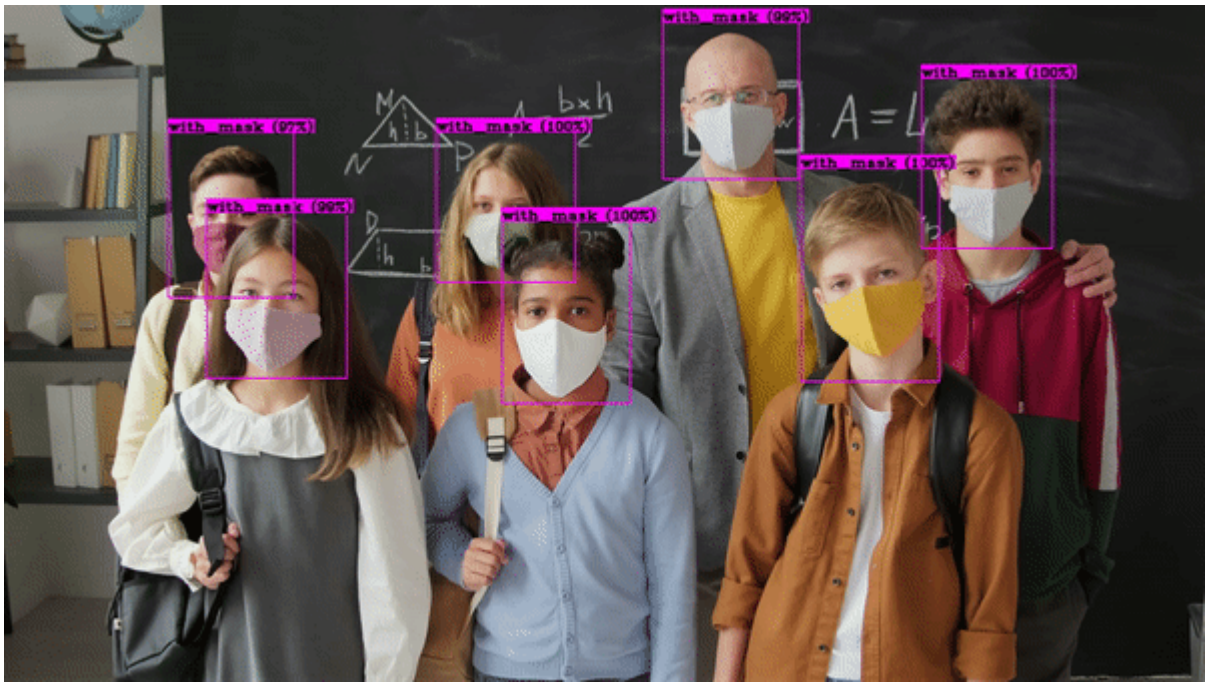
For testing on webcam capture or videos, use this colab notebook.

NOTE:

The dataset I have collected for mask detection contains mostly close-up images. For more long-shot images you can search online. There are many sites where you can download labeled and unlabeled datasets. I have given a few links at the bottom under Dataset Sources. I have also given a few links for mask datasets. Some of them have more than 10,000 images.

Though there are certain tweaks and changes we can make to our training config file or add more images to the dataset for every type of object class through augmentation, we have to be careful so that it does not cause overfitting which affects the accuracy of the model.

For beginners, you can start simply by using the config file I have uploaded on my GitHub. I have also uploaded my mask images dataset along with the PASCAL_VOC format labeled text files, which although might not be the best but will give you a good start on how to train your own custom object detector using an SSD model. You can find a labeled dataset of better quality or an unlabeled dataset and label it yourself later.



Original Video by Max Fischer from Pexels

My GitHub

Files for training

Train Custom Object Detection model using TF 2

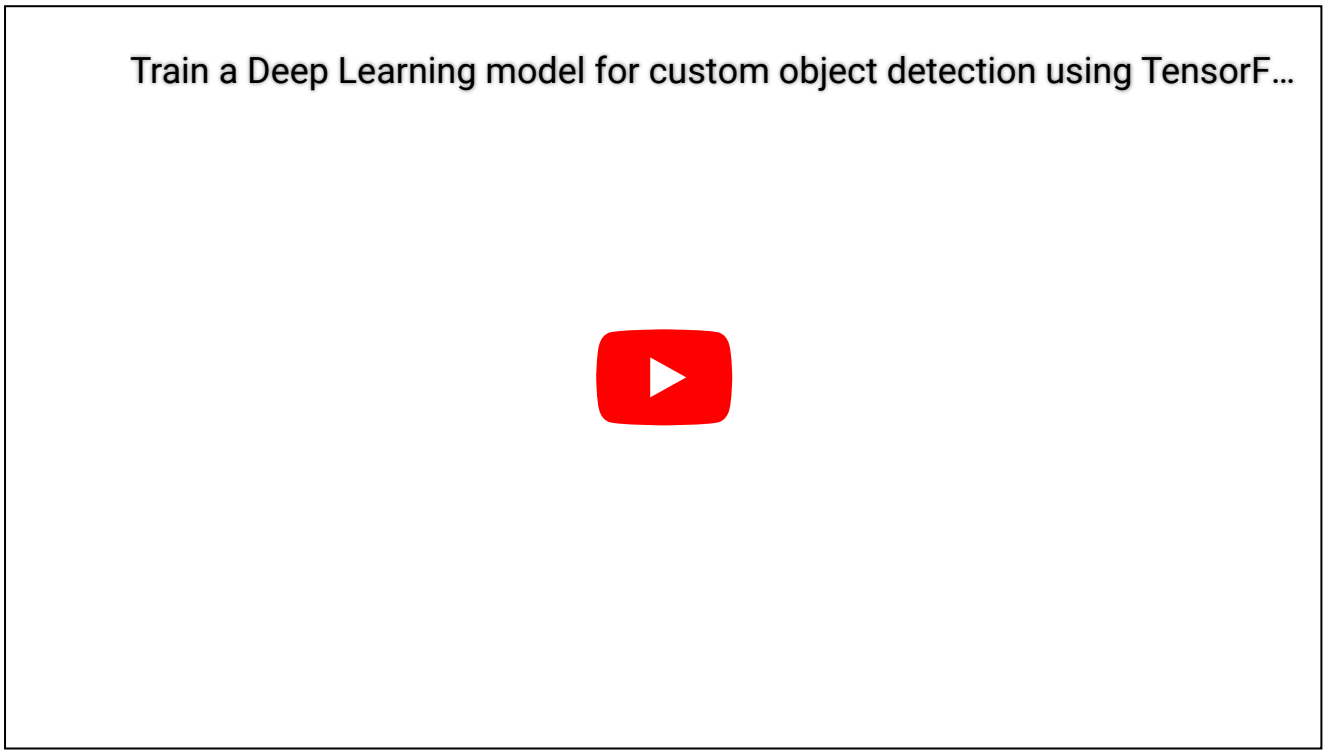
My mask dataset

https://www.kaggle.com/techzizou/labeled-mask-dataset-pascal-voc-format

My Colab notebook for this

Custom Object Detection TensorFlow 2.x

My Youtube Video on this!



CREDITS

Documentation / References

- Tensorflow Introduction
- Tensorflow Models Git Repository
- TensorFlow Object Detection API Repository
- TensorFlow Object Detection API tutorial
- TF Object Detection Documentation
- TF2 installation guide
- TensorFlow 2 Detection Model Zoo
- TensorFlow 2 Classification Model Zoo
- Training and Evaluation with TensorFlow 2
- Tensorflow tutorials
- Tensorflow Hub

- TensorFlow Hub Object Detection Colab
- Object detector tutorial

Dataset Sources

You can download datasets for many objects from the sites mentioned below. These sites also contain images of many classes of objects along with their annotations/labels in multiple formats such as the YOLO_DARKNET txt files and the PASCAL_VOC xml files.

- Open Images Dataset by Google
- Kaggle Datasets
- Roboflow Public Datasets
- VisualData Datasets

Mask Dataset Sources

- Prajnasb Github
- Andrewmvd Kaggle
- X-zhangyang Github

More Mask Datasets

- Prasoonkottarathil Kaggle (20000 images)
- Ashishjangra27 Kaggle (12000 images)

TROUBLESHOOTING:

OpenCV Error:

If you get an error for _registerMatType cv2 above, this might be because of OpenCV version mismatches in Colab. Run `!pip list|grep opencv` to see the versions of OpenCV packages installed i.e. `opencv-python`, `opencv-contrib-python` & `opencv-python-headless`. The versions will be different which is causing this error. This error will go away when colab updates its supported versions. For now, you can fix this by simply uninstalling and installing OpenCV packages.

Check versions:

```
!pip list|grep opencv
```

Use the following 2 commands if only the opencv-python-headless is of different version

```
!pip uninstall opencv-python-headless –y

!pip install opencv-python-headless==4.1.2.30
```

Or use the following commands if other opencv packages are of different versions. Uninstall and install all with the same version.

```
!pip uninstall opencv-python –y

!pip uninstall opencv-contrib-python –y

!pip uninstall opencv-python-headless –y

!pip install opencv-python==4.5.4.60

!pip install opencv-contrib-python==4.5.4.60

!pip install opencv-python-headless==4.5.4.60
```

DNN Error

DNN library is not found

This error is due to the version mismatches in the Google Colab environment. Here this can be due to 2 reasons. One, as of now since the default TensorFlow version in Google Colab is 2.8 but the default TensorFlow version for the Object Detection API which we install in step 6 is 2.9.0, this causes an error.

Second, the default cuDNN version for Google Colab as of now is 8.0.5 but for TF 2.8 and above it should have 8.1.0. This also causes version mismatches.

This error will go away when Colab updates its packages. But for temporary fixes, after searching on many forums online and looking at responses from members of the Google Colab team, the following are the 2 possible solutions I can recommend:

SOLUTION 1)

This is the easiest fix, however as per the comment of a Google Colab team member on a forum, this is not the best practice and is not safe. This can also cause mismatches with other packages or libraries. But as a temporary workaround here, this will work.

Run the following command before the training step. This will update the cudnn version and you will have no errors after that.

```
!apt install --allow-change-held-packages libcudnn8=8.1.0.77-1+cuda11
```

SOLUTION 2)

In this method, you can edit the package versions to be installed in the TensorFlow Object Detection API so that it is the same as the default version for Colab.

We divide step 6 into 2 sections.

Section 1:

```
# clone the tensorflow models on the colab cloud vm
!git clone --q https://github.com/tensorflow/models.git

#navigate to /models/research folder to compile protos
%cd models/research

# Compile protos.
!protoc object_detection/protos/*.proto --python_out=.
```

The above section 1 will clone the TF models git repository.

After that, you can edit the file at *object_detection/packages/tf2/setup.py* .
Change the code in the REQUIRED PACKAGES to include the following 4 lines after the pandas package line:

```
'tensorflow==2.8.0',
'tf-models-official==2.8.0',
'tensorflow_io==0.23.1',
'keras==2.8.0'
```

Next, after that, you can run section 2 of step 6 shown below to install the TF2 OD API with the updated *setup.py* file.

Section 2:

```
# Install TensorFlow Object Detection API.
!cp object_detection/packages/tf2/setup.py .
!python -m pip install .
```

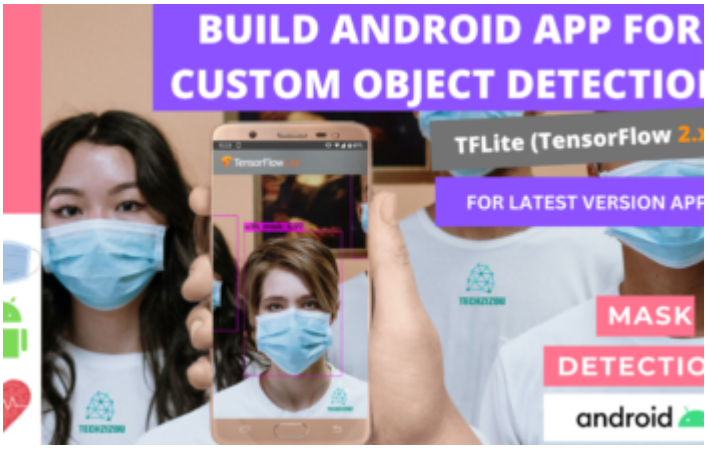
This will install the TensorFlow Object Detection API with TensorFlow 2.8 0 and other required packages with the updated versions we specified in the *setup.py* file.

Now you will be able to run the training step without any errors.

« Build a custom Image classification Android app using Teachable Machine (Old Version TF App)

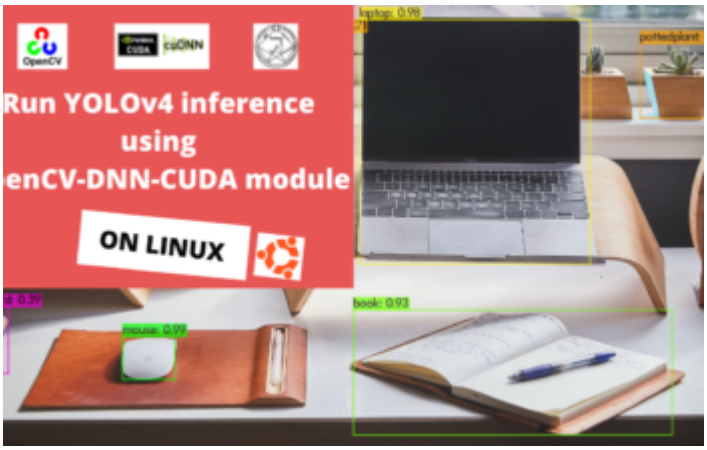
Build Android app for object detection (TensorFlow 1.x) »

Related Posts



Build Android app for custom object detection

Build a custom Image classification Android app using Teachable Machine (Latest Version TensorFlow App)



YOLOv4 inference using OpenCV-DNN-CUDA on Linux

Leave a Reply

Write a Comment...

Reply

Search …

Search

English