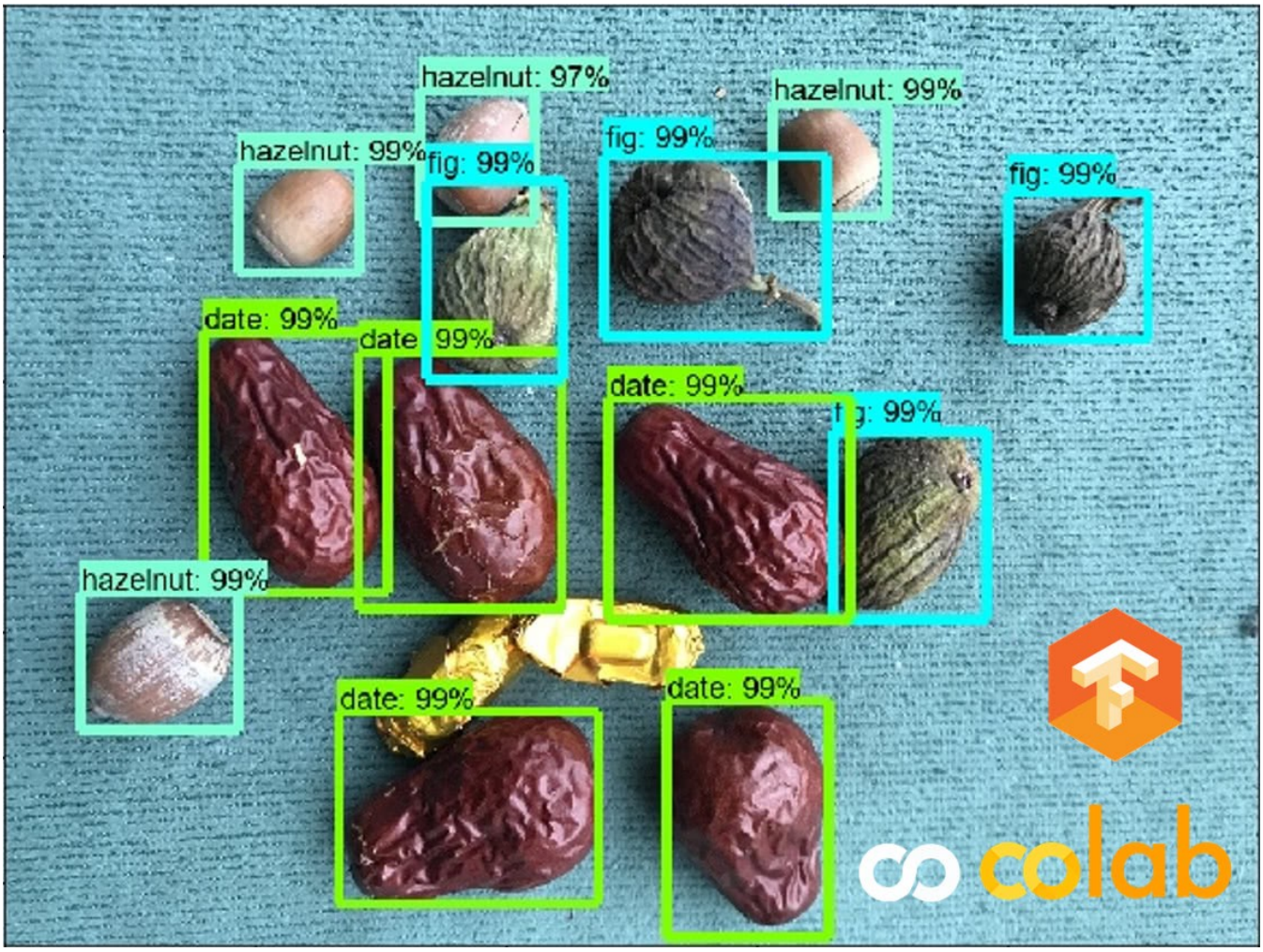
How to train an object detection model easy for free

Chengwei Zhang
Feb 12 · 5 min read



In this tutorial, you will learn how to train a custom object detection model easily with TensorFlow object detection API and Google Colab's free GPU.

Annotated images and source code to complete this tutorial are included.

TL: DR; Open the Colab notebook and start exploring.

Otherwise, let's start with creating the annotated datasets.

Step 1: Annotate some images

During this step, you will find/take pictures and annotate objects' bounding boxes. It is only necessary if you want to use your images instead of ones comes with **my repository**.

If your objects are simple ones like nuts and fruits in my example, 20 images can be enough with each image containing multiple objects.

In my case, I use my iPhone to take those photos, each come with 4032 x 3024 resolution, it will overwhelm the model if we use that as direct input to the model. Instead, resize those photos to uniformed size `(800, 600)` can make training and inference faster.

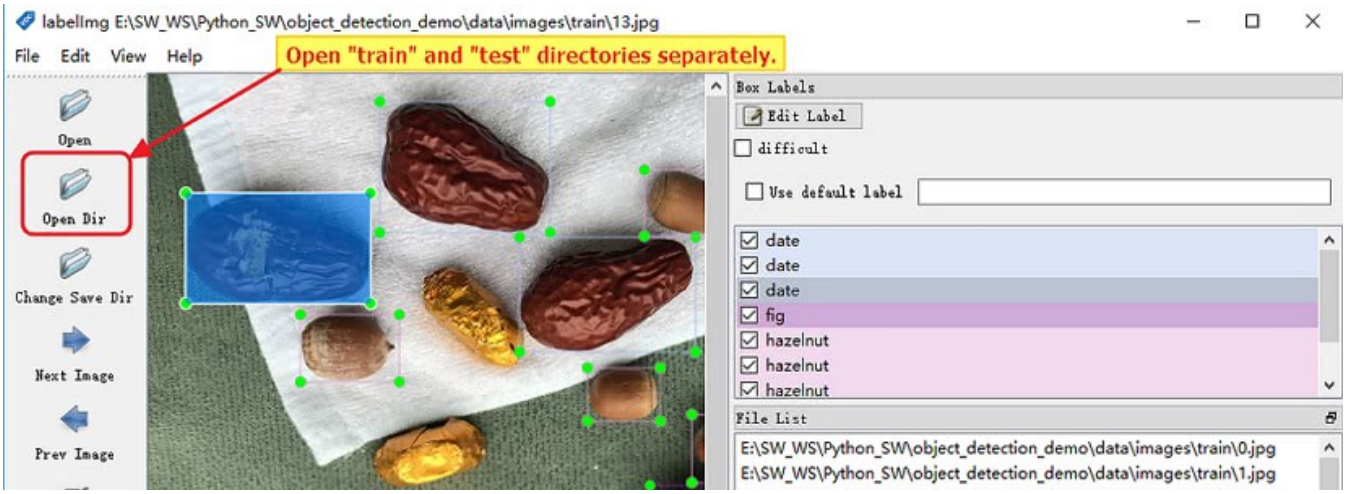
You can use the resize_images.py script in the repository to resize your images.

First, save your photos, ideally with `jpg` extension to `./data/raw` directory. Then run,

```
python resize_images.py --raw-dir ./data/raw --save-dir
./data/images --ext jpg --target-size "(800, 600)"
```

Resized images will locate in `./data/images/`

Next, we split those files into two directories, `./data/images/train` and `./data/images/test`. The model will only use images in the "**train**" directory for training and images in "**test**" directory serve as additional data to evaluate the performance of the model.



labelImg

Annotate resized images with **labelImg**, this annotation tool supports both Windows and Linux, it will generate `xml` files inside `./data/images/train` and `./data/images/test` directories.

Tips: use shortcuts (`w` *: draw box,* `d` *: next file,* `a` *: previous file, etc.) to accelerate the annotation.*

Step 2: prepare `tfrecord` files (source included in Colab notebook)

After running this step, you will have two files `train.record` and `test.record`, both are binary files with each one containing the encoded jpg and bounding box annotation information for the corresponding train/test set. The tfrecord file format is easier to use and faster to load during the training phase compared to storing each image and annotation separately.

There are two steps in doing so:

- Converting the individual `*.xml` files to a unified `*.csv` file for each set(train/test).
- Converting the annotation `*.csv` and image files of each set(train/test) to `*.record` files (TFRecord format).

Use the following scripts to generate the `tfrecord` files as well as the `label_map.pbtxt` file which maps every object class name to an integer.

```
# Convert train folder annotation xml files to a single csv file,
# generate the `label_map.pbtxt` file to `data/annotations` directory as well.
python xml_to_csv.py -i data/images/train -o data/annotations/train_labels.csv -l data/annotati

# Convert test folder annotation xml files to a single csv.
python xml_to_csv.py -i data/images/test -o data/annotations/test_labels.csv

# Generate `train.record`
python generate_tfrecord.py --csv_input=data/annotations/train_labels.csv --output_path=data/ar
```

```
python generate_tfrecord.py --csv_input=data/annotations/train_labels.csv --output_path=data/a

# Generate `test.record`
python generate_tfrecord.py --csv_input=data/annotations/test_labels.csv --output_path=data/anr
```

Step 3: Configuring a Training Pipeline

Instead of training the model from scratch, we will do transfer learning from a model pre-trained to detect everyday objects.

Transfer learning requires less training data compared to training from scratch.

But keep in mind transfer learning technique supposes your training data is somewhat similar to the ones used to train the base model. In our case, the base model is trained with coco dataset of common objects, the 3 target objects we want to train the model to detect are fruits and nuts, i.e. “date”, “fig” an “hazelnut”. They are similar to ones in coco datasets. On the other hand, if your target objects are lung nodules in CT images, transfer learning might not work so well since they are entirely different compared to coco dataset common objects, in that case, you probably need much more annotations and train the model from scratch.

To do the transfer learning training, we first will download the pre-trained model weights/checkpoints and then config the corresponding pipeline config file to tell the trainer about the following information.

- the pre-trained model checkpoint path(fine_tune_checkpoint),
- the path to those two tfrecord files,
- path to the **label_map.pbtxt** file(label_map_path),
- training batch size(batch_size)
- number of training steps(num_steps)
- number of classes of unique objects(num_classes)

Step 4: Train the model

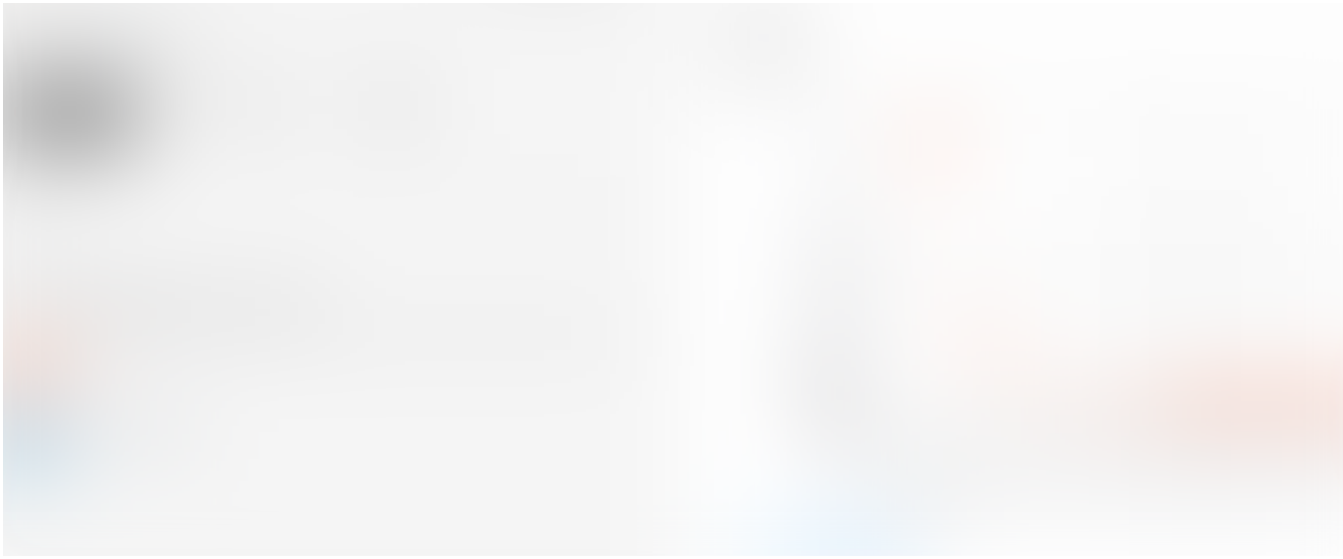
After that, we can start the training, where the **model_dir** is the path of a new directory to store our output model.

```
!python /content/models/research/object_detection/model_main.py \
    --pipeline_config_path={filename} \
    --model_dir={model_dir} \
    --alsologtostderr \
    --num_train_steps={num_steps} \
    --num_eval_steps={num_eval_steps}
```

train_cell.sh hosted with ♡ by GitHub view raw

Inside the colab notebook, TensorBoard is also configured to help you visualize the training progress and results. Here are two screenshots of TensorBoard show the prediction on test images and monitor of loss value.

TensorBoard Images



TensorBoard Scalars

Step 5: Exporting and download a Trained model

Once your training job is complete, you need to extract the newly trained model as an inference graph, which will be later used to perform the object detection. The conversion can be done as follows:

```
!python /content/models/research/object_detection/export_inference_graph.py \
    --input_type=image_tensor \
    --pipeline_config_path=/content/models/research/object_detection/samples/configs/faster_rcnn
    --output_directory=fine_tuned_model \
    --trained_checkpoint_prefix={last_model_path}
```

export_inference_graph.sh hosted with ♡ by GitHub view raw

You can find the model frozen graph file at path **fine_tuned_model/frozen_inference_graph.pb**. Download it either through Google Drive or directly as shown in the colab notebook.

The final section in the notebook shows you how to load the **.pb** file, the **label_map.pbtxt** file and make predictions on some test images. Here is a detection output example.

Conclusion and further thought

Training an object detection model can be resource intensive and time-consuming. This tutorial shows you it can be as simple as annotation 20 images and run a Jupyter notebook on Google Colab. In the future, we will look into deploying the trained model in different hardware and benchmark their performances. To name a few deployment options,

- Intel CPU/GPU accelerated with OpenVINO tool kit, with FP32 and FP16 quantized model.
- Movidius neural compute stick with OpenVINO tool kit.
- Nvidia GPU with Cuda Toolkit.
- SoCs with NPU like Rockchip RK3399Pro.

Stay tuned and don't forget to check out the GitHub repository and the Google Colab Notebook for this tutorial.

. . .

Originally published at www.dlology.com.

Machine Learning Deep Learning Colab TensorFlow Object Detection