Training Custom Object Detector

Overview

1. Preparing the workspace
2. have a folder Tensorflow, placed under <PATH\_TO\_TF> (e.g. C:\Users\sglvladi\Documents), with the following directory tree:



1. create a new folder under TensorFlow and call it workspace. It is within the workspace that we will store all our training set-ups. Now let’s go under workspace and create another folder named training\_demo. Now our directory structure should be as so:



1. The training\_demo folder shall be our training folder, which will contain all files related to our model training. It is advisable to create a separate training folder each time we wish to train a different model. The typical structure for training folders is shown below:



1. Annotating images:

To annotate images we will be using the [labelImg](https://github.com/tzutalin/labelImg) package. If you haven’t installed the package yet, then have a look at [LabelImg Installation](https://tensorflow-object-detection-api-tutorial.readthedocs.io/en/latest/install.html" \l "labelimg-install).

* Once you have collected all the images to be used to test your model (ideally more than 100 per class), place them inside the folder training\_demo\images.
* Open a new Anaconda/Command Prompt window and cd into Tensorflow\addons\labelImg.

If (as suggested in [LabelImg Installation](https://tensorflow-object-detection-api-tutorial.readthedocs.io/en/latest/install.html" \l "labelimg-install)) you created a separate Conda environment for labelImg then go ahead and activate it by running: activate labelImg

* Next go ahead and start labelImg, pointing it to your training\_demo\images folder.

python labelImg.py ..\..\workspace\training\_demo\images

* A File Explorer Dialog windows should open, which points to the training\_demo\images folder.
* Press the “Select Folder” button, to start annotating your images.
* After splitting your dataset, copy all training images, together with their corresponding \*.xml files, and place them inside the training\_demo\images\train folder. Similarly, copy all testing images, with their \*.xml files, and paste them inside training\_demo\images\train.

1. Creating Label Map:

TensorFlow requires a label map, which namely maps each of the used labels to an integer values. This label map is used both by the training and detection processes.

item {

id: 1

name: 'signature'

}

Label map files have the extention .pbtxt and should be placed inside the training\_demo\annotations folder.

1. Creating TensorFlow Records:

We have generated our annotations and split our dataset into the desired training and testing subsets, it is time to convert our annotations into the so called TFRecord format.

There are two steps in doing so:

* Converting the individual \*.xml files to a unified \*.csv file for each dataset.
* Converting the \*.csv files of each dataset to \*.record files (TFRecord format).
* create a directory where we can store some scripts. Under the TensorFlow folder, create a new folder TensorFlow\scripts, which we can use to store some useful scripts. To make things even tidier, let’s create a new folder TensorFlow\scripts\preprocessing, where we shall store scripts that we can use to preprocess our training inputs. Below is out TensorFlow directory tree structure, up to now



1. Converting .xml to .csv:

* write a script that iterates through all \*.xml files in the training\_demo\images\train and training\_demo\images\test folders, and generates a \*.csv for each of the two.
* Create a new file with name xml\_to\_csv.py under TensorFlow\scripts\preprocessing, open it, paste the above code inside it and save.
* Install the pandas package.
* Finally, cd into TensorFlow\scripts\preprocessing and run:

*# Create train data:*

python xml\_to\_csv.py -i [PATH\_TO\_IMAGES\_FOLDER]/train -o [PATH\_TO\_ANNOTATIONS\_FOLDER]/train\_labels.csv

*# Create test data:*

python xml\_to\_csv.py -i [PATH\_TO\_IMAGES\_FOLDER]/test -o [PATH\_TO\_ANNOTATIONS\_FOLDER]/test\_labels.csv

1. Converting .csv to .records:

* we have obtained our \*.csv annotation files, we will need to convert them into TFRecords.

1. Configuring the Training pipeline:

The model we shall be using in our examples is the ssd\_inception\_v2\_coco model, since it provides a relatively good trade-off between performance and speed.

1. Training the Model:

To initiate a new training job, cd inside the training\_demo folder and type the following: python train.py --logtostderr --train\_dir=training/ -- pipeline\_config\_path=training/ssd\_inception\_v2\_coco.config

1. Observations for SSD Model











