**Object Detection User Guide**

By: Hannah Li

The purpose of this program is to automate object detection for road signs, using our own dataset. This program can detect multiple objects in one image with a certain percentage of accuracy. The steps below demonstrate how to create this model from scratch; however, you can also treat this as a transfer learning model and add to the model I pre-trained.

**Set Up**

Download the “Object Detection Final” model, and the images in “train”, and “val” folders from [this Github repository](https://github.com/hannahli2014/object_detection) or Hannah folder in E-Motion Inc. Google Drive (use this [link](https://drive.google.com/drive/folders/1bQyltZ0AqxJJSzRYuk9oXs_24PTazOQy?usp=sharing) to access.).

Download the LabelImg software [here](https://github.com/tzutalin/labelImg) (if you plan to add more images to your dataset).

Download Jupyter Notebook [here](https://jupyter.org/install).

To use pretrained model, download images and xml files in respective folders. Once you download models directory from TensorFlow GitHub (download can be done in the notebook), create 2 new folders with the path ./models/research/object\_detection/images/{}, with {} as “new train” and “new val” Create 2 more folders with the path ./models/research/{} with {} as “new train xml” and “new val xml”.

Store train images in a “new train” folder, store val images in “new val” folder.

Store train xml files in “new train xml” folder, store val xml files in “new val xml” folder.

Store test images in ./models/research/object\_detection/test\_images.

**Steps**

1. Gather and annotate images

2. Store images in project directory

3. Install libraries

4. Create labelmap.pbtxt

5. Update {model}.config file

6. Update %run command

7. Run code

8. Choose trained model

9. Upload test images

10. Update test image path, number of classes

**Dataset**

The dataset currently contains x training images, y test images, and z classes. You can add more images and classes to the dataset by using the LabelImg software to create bounding boxes and labels around these images. This software will create .xml files for each image that is needed for the model.

**How to Store New Images**

To create your own dataset or add new images to my current dataset, follow the steps below:

1. Gather new images and store in /images/train and /images/validation folders

2. Create .xml files using LabelImg software

3. Store .xml files in train/validation folders

./ refers to the base directory this project is stored. This is the directory where you will download the *models* file from TensorFlow GitHub.

Add new training images to the ./models/research/object\_detection/images/new train folder.

Add new validation images to the ./models/research/object\_detection/images/new val folder.

Add new training xml files to the ./models/research/new train xml folder.

Add new validation xml files to the ./models/research/new val xml folder.

**Installing Libraries**

Make sure to have these libraries installed to successfully run the code:

pip install pillow

pip install lxml

pip install Cython

pip install jupyter

pip install matplotlib

pip install pandas

pip install opencv-python

pip install tensorflow

**Create labelmap.pbtxt**

Labelmap.pbtxt is a file that maps an ID to a text label (we create an ID when we generate TFRecords in the code). Write the file in a new cell block, using the format shown below. Create an item map for each class in your dataset.

%writefile ./object\_detection/training/labelmap.pbtxt

item {  
 id: 1  
 name: 'stop'  
}item {  
 id: 2  
 name: 'pedestrianCrossing'  
}item {  
 id: 3  
 name: 'oneWay'  
}item {  
 id: 4  
 name: 'rightLaneMustTurn'  
}

...

**Update {model}.config file**

TensorFlow GitHub hosts a “model zoo” that is essentially the framework for a bunch of different models that can be used for object detection. Each model has a .config file that is code to run the model. Select the .config file for the model you want to use, copy the code in this file and paste into a new code cell/overwrite my old .config file. In your new .config file, regardless of the model you chose, there are 7 parts of the code you need to update. Write the file in a new cell block, using the format shown below, and updating these 7 parts in the file (replace {model} with your model name.

\*\*This file needs to be re-written only if you change the number of classes, number of steps, or paths to any of the files below. Update file each time you train/re-train the model. If you want to reduce the number of iterations after running file, you have to delete this file and rerun. If you want to add more iterations, update file and rerun to continue running from most recent iteration.

%writefile ./object\_detection/training/{model}.config

model {

ssd {

num\_classes: 39

box\_coder {

faster\_rcnn\_box\_coder {

y\_scale: 10.0

x\_scale: 10.0

height\_scale: 5.0

width\_scale: 5.0

}

}

Update num\_classes to how many image classes you have in your datset.

fine\_tune\_checkpoint: "/home/hannah/Software/python/models/research/object\_detection/ssd\_mobilenet\_v2\_coco\_2018\_03\_29/model.ckpt"

fine\_tune\_checkpoint\_type: "detection"

#change this to how many steps/iterations you want to train

num\_steps: 6050

data\_augmentation\_options {

random\_horizontal\_flip {

}

}

Update fine\_tune\_checkpoint to where your model.ckpt is stored. This path will look similar to above except with where your model name folder is stored.

Update num\_steps to how many steps you want to run your model. I find that a smaller or very clean dataset requires fewer steps.

train\_input\_reader: {

tf\_record\_input\_reader {

input\_path: "/home/hannah/Software/python/models/research/train.record"

}

label\_map\_path: "/home/hannah/Software/python/models/research/object\_detection/training/labelmap.pbtxt"

}

Update input\_path to where your train.record file is stored. Path should be under ./models/research/

Update label\_map\_path to where your labelmap.pbtxt file is stored. Path should be under ./models/research/object\_detection/training/

eval\_input\_reader: {

tf\_record\_input\_reader {

input\_path: "/content/models/research/val.record"

}

label\_map\_path: "/content/models/research/object\_detection/training/labelmap.pbtxt"

shuffle: false

num\_readers: 1

}

Update input\_path to where your val.record file is stored. Path should be under ./models/research/

Update label\_map\_path to where your labelmap.pbtxt file is stored. Path should be under ./models/research/object\_detection/training/ (same as above).

**Update %run command**%run legacy/train.py --logtostderr --train\_dir=training/ --pipeline\_config\_path=training/ssd\_mobilenet\_v2\_coco.config

Change this to your .config file name.

**Run Code**

After you update these files, run the code up to and including the %run line. Initial training of the model can take a while depending on the number of steps you set.

**Choose Trained Model**

After training the model, use the command ls training to view your saved models. Choose the most updated one and replace it in the code below.

python3 my\_inference\_graph.py \

--input\_type image\_tensor \

--pipeline\_config\_path training/ssd\_mobilenet\_v2\_coco.config \

--trained\_checkpoint\_prefix training/model.ckpt-6050 \

--output\_directory ./inference\_graph

Run this line of code to set as object detection model to use with test images (this is the model that will predict the location and label for an object in an image).

**Upload Test Images**

Test images should be stored in ./model/research/object\_detection/test\_images/ folder.

**Update Test Image Path, Number of Classes**

PATH\_TO\_IMAGE = os.path.join(os.getcwd(),'test\_images', 'stopoenway.jpg')

NUM\_CLASSES = 39

NUM\_CLASSES needs to be updated initially, and every time after you increase/reduce the number of classes in the dataset. Update PATH\_TO\_IMAGE for each new image you want to test (only need to update file name if stored in test\_images folder).

Run remaining code to successfully detect object(s) in test image.