

Weed Detection Vision System Documentation

The aim of this tutorial is to get the TensorFlow (TF) object detection API working for detecting and classifying crops and weeds. This API can be used to detect, with bounding boxes, objects in images and/or video using either some of the pre-trained models made available or through models we can train on our own. Hopefully you can walk into this as cold as I did and get a working version up and running a lot quicker.

Key Reading List:

<https://towardsdatascience.com/how-to-train-your-own-object-detector-with-tensorflows-object-detector-api-bec72ecfe1d9>

<https://towardsdatascience.com/building-a-toy-detector-with-tensorflow-object-detection-api-63c0fdf2ac95>

<https://towardsdatascience.com/deep-learning-specialization-by-andrew-ng-21-lessons-learned-15ffaaef627c>

<https://cloud.google.com/blog/big-data/2016/08/how-a-japanese-cucumber-farmer-is-using-deep-learning-and-tensorflow>

Assumptions:

It is assumed that you have a windows 10, 64 bit PC with python version 3.5.2 and community edition pycharm installed. Also, please make use of the windows CMD throughout the documentation for entering the commands.

Python download link:

<https://www.python.org/downloads/windows/>

Pycharm download link:

<https://www.jetbrains.com/pycharm/download/#section=windows>

My System information

- 1.) OS Platform and Distribution: Windows 10 64 Bit
- 2.) TensorFlow installed from (source or binary): binary
- 3.) TensorFlow version :1.4.0
- 4.) Python version 3.5.2(v3.5.2:4def2a2901a5, Jun 25 2016, 22:18:55)
- 5.) GPU:nVidia GeForce 755M 2GB CPU: Intel x64-64 Intel Core i5-4200M CPU @2.50Ghz, 8GB memory

Step 1: Setting up the environment

Step 1.1: Download the object detection repository:

```
https://github.com/tensorflow/models.git
```

Rename the main folder from models-master to 'models'.

Step 1.2: Installing TensorFlow and dependencies

The following is amended steps which were inherited from the official instruction guide:

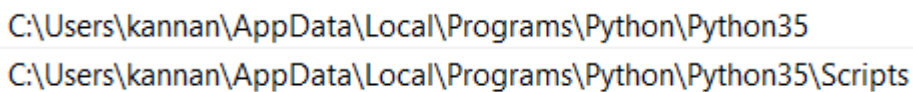
https://github.com/tensorflow/models/blob/master/research/object_detection/g3doc/installation.md

Tensorflow Object Detection API depends on the following libraries:

- Protobuf 2.6
- Pillow 1.0
- lxml
- tf Slim (which is included in the "tensorflow/models/research/" checkout)
- Jupyter notebook
- Matplotlib
- Tensorflow
- Pandas

To make use of the keyword pip you need to add it to the environment variables. This link explains it properly for python 2.7: <https://github.com/Langoor2/PokemonGo-Map-FAQ/wiki/%27python---pip%27-is-not-recognized-as-an-internal-or-external-command,-operable-program-or-batch-file>

For python 3.5.2 the only difference is the directory of Python35 and Scripts. Use the image below as a reference:



```
C:\Users\kannan\AppData\Local\Programs\Python\Python35  
C:\Users\kannan\AppData\Local\Programs\Python\Python35\Scripts
```

Official Tensorflow installation instructions:

https://www.tensorflow.org/install/install_windows

You will have two options to install TF. I went with native pip.

Installing Tensorflow

```
# For CPU  
pip install tensorflow  
# For GPU  
pip install tensorflow-gpu
```

Installing Dependencies:

```

pip install pillow
pip install lxml
pip install jupyter
pip install matplotlib
pip install pandas
# or in one
pip install pillow lxml jupyter matplotlib pandas

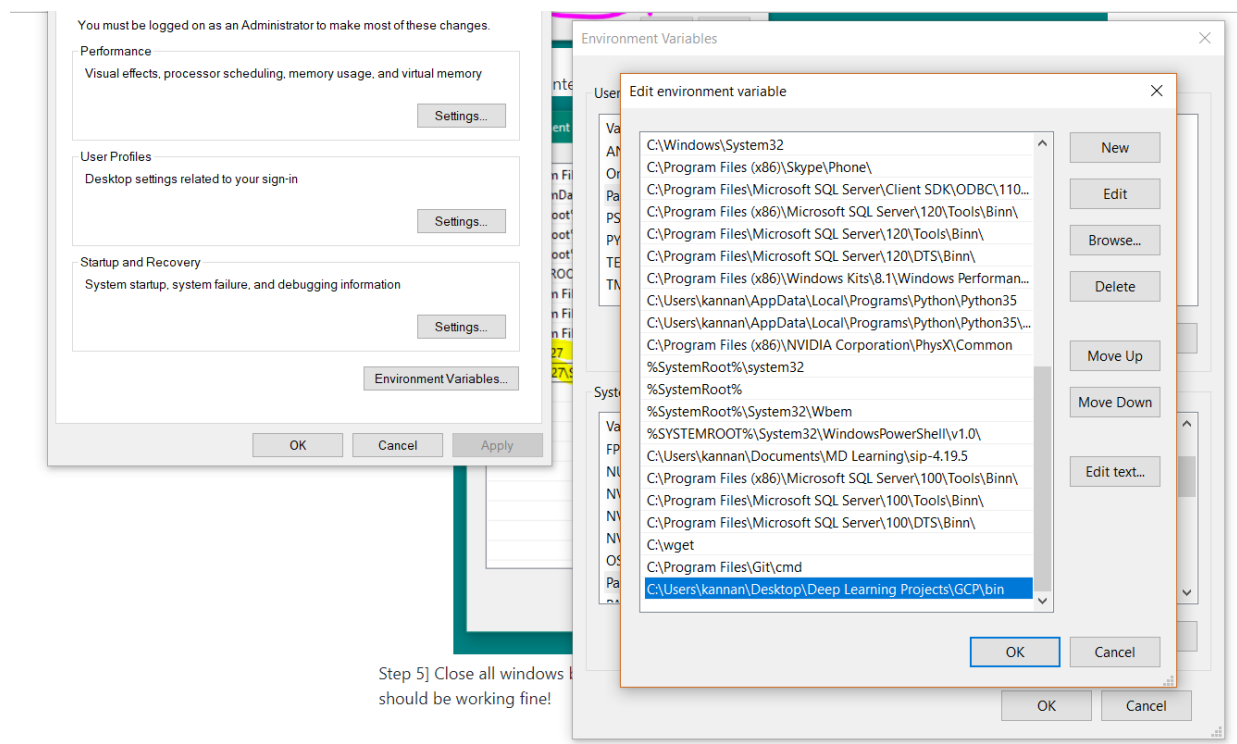
```

Step 1.3: Protobuf compilation and testing installation

Head to the protoc releases page and download the protoc-3.4.0-win32.zip, extract it, and you will find protoc.exe in the bin directory.

<https://github.com/google/protobuf/releases>

You can move this to something more appropriate if you like, or leave it in downloads. I eventually put mine in a random folder called GCP and added it to my system environment variable as shown below:



For protoc to be recognised as an internal command on CMD, it needs to be added to the system environment variable similar to how we did for 'pip'.

Now, from within the models/research directory, you can use the protoc command like so:

```

"C:\Users\kannan\Desktop\Deep Learning Projects\GCP\bin\protoc"
object_detection\protos*.proto --python_out=.

```

OR

If you have added protoc into system environment variable you can directly use the following command:

#From models/research directory

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

Note: If it does not work, try putting the protoc.exe file into models/research directory and running the above command again.

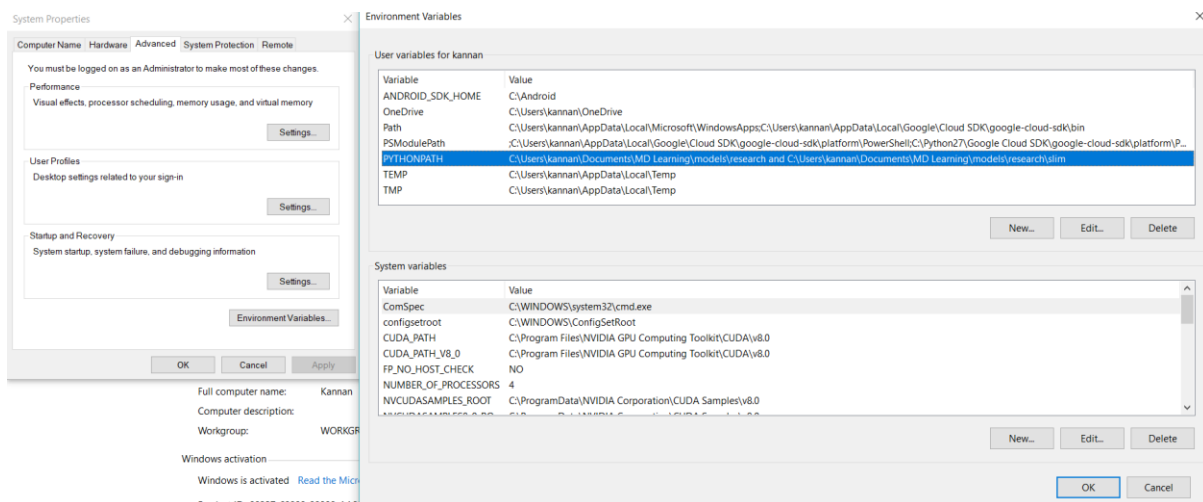
After executing, now navigate to models\research\object_detection\protos and verify the .py files were created successfully as a result of the compilation. (Only the .proto files were there to begin with)

Testing Installation

Make sure before testing installation, add two paths to PYTHONPATH system variable:

location of the models folder — Eg. *C://location_to_models_folder/models*

location of the models/slim folder — Eg. *C://location_to_models_folder/models/slim*



You can test that you have correctly installed the Tensorflow Object Detection API by running the following command:

#From models/research directory

```
python object_detection/builders/model_builder_test.py
```

There might be two errors when you run it using a python version greater than 3 — due to iteritems() function being used, which has been deprecated in python 3. Don't worry about the two errors, these are expected.

Step 2: Labelling the images

Try to collect few hundred images of the object of your interest. The bare minimum would be about 100, ideally more like 500+, but, the more the images you have, the more tedious labelling will become. So, try to find a middle ground to not compensate on accuracy as well.

We are going to be using labellmg to annotate the images. This step is basically to draw bounding boxes around the object in the image. The label program will automatically create an XML file that describes the object in the picture.

Labellmg is a very handy tool and annotations are created in the Pascal VOC format. It is written in Python and uses Qt for interface. Essentially we identify xmin, ymin, xmax and ymax for the object and pass that to the model along with the image for training.

In my case, a total of 282 test labels and 2192 labels for training were created on 187 images. Each image is 960 by 720 pixels.

Using the following link to download labellmg:

<https://github.com/tzutalin/labellmg>

You can directly download the prebuilt binary version for windows. Do not waste time trying to build it from source.

When running the .exe file, you should get a GUI window. From here, choose to open dir and pick the directory that you saved all of your images to. Now, you can begin to annotate with the create rectbox button. Draw your box, add the name in, and hit ok. Save, hit next image, and repeat! You can press the w key to draw the box and do ctrl+s to save faster.

Note: You can have as many labels as you want for one image. Save all the images in the same **Images** directory. Also, please remember, only bounding boxes can be drawn for tensorflow object detection API.

Link to my labelled images along with TFRecord files:

<https://drive.google.com/open?id=1GYZaxpg9MJiKwBxtzA32SVw9eArTzu79>

Step 2: Creating TF Records

Next up, I separated them into training and testing groups. To do this, I just copied about 10% of the images and their annotation XML files to a new dir called **Test** and then copied the remaining ones to a new dir called **Train** which has the remaining 90%.

We are going to be using helper code from the racoon dataset repo:

https://github.com/datitran/raccoon_dataset

We are going to inherit two codes – 1.) xml_to_csv.py and 2.) generate_tfrecord.py

Open the `xml_to_csv.py` and copy the raw code. It would be a good idea to make a new directory on desktop called **Object Detection** and drag the **Images** folder into it as shown below:

Card	Organize	New	Open	Select
> This PC > Desktop > Deep Learning Projects > Crop Weed > Object Detection > Images				
Name	Date modified	Type	Size	
Test	14/11/2017 5:06 PM	File folder		
Train	14/11/2017 5:06 PM	File folder		
Image1.jpg	10/11/2017 10:26 ...	JPEG image	220 KB	
Image1.xml	10/11/2017 8:40 PM	XML Document	2 KB	
Image2.jpg	31/10/2017 10:24 ...	JPEG image	274 KB	
Image2.xml	14/11/2017 12:09 ...	XML Document	4 KB	
Image3.jpg	31/10/2017 10:24 ...	JPEG image	278 KB	
Image3.xml	14/11/2017 11:13 ...	XML Document	3 KB	
Image4.jpg	31/10/2017 10:24 ...	JPEG image	279 KB	
Image4.xml	14/11/2017 11:27 ...	XML Document	4 KB	
Image5.jpg	31/10/2017 10:24 ...	JPEG image	276 KB	
Image5.xml	14/11/2017 12:15 ...	XML Document	4 KB	
Image6.jpg	31/10/2017 10:24 ...	JPEG image	280 KB	
Image6.xml	14/11/2017 2:49 PM	XML Document	5 KB	
Image7.jpg	31/10/2017 10:24 ...	JPEG image	276 KB	
Image7.xml	14/11/2017 3:09 PM	XML Document	5 KB	

The **Images** folder basically has all the images with their matching annotation files and two directories called **Test** and **Train**. As mentioned before copy 90% of the images and their matching annotation files into **Train** directory and likewise copy the remaining 10% of the images and their matching annotation files into the **Test** directory.

Within the **Object Detection** directory create a new `.py` using pycharm or any other IDE to paste the copied code. Also, make sure you create a new directory called **data** within **Object Detection** folder.

In the main loop we need to make some changes.

```
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.')

main()
```

Save the `xml_to_csv.py` file and open CMD. Change directory to **Object Detection** and run:

```
python xml_to_csv.py
```

You should have the csv files created in the **data** directory after running the above command. You can have a look at the csv files using excel.

Next up, go back to the racoon dataset to copy the generate_tfrecord raw code. Create a new .py file within the **Object Detection** folder with the raw code pasted in it. We need to make some changes in the file.

In line 29, change the labels to whatever you want them to be. In my case:

```
# TO-DO replace this with label map
def class_text_to_int(row_label):
    if row_label == 'crop':
        return 1
    if row_label == 'weed':
        return 2
    else:
        return None
```

Next up, Install the object_detection library formally by running the following from within the models/research directory:

```
python setup.py install
```

Now, we can run the generate_tfrecord.py script. We will run it twice, once for the train TFRecord and once for the test TFRecord.

```
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
```

Now, in your **data** directory, you should have train.record and test.record.

The following is how the folder tree should look like after completing the above steps:

```
Object Detection
-data/
--test_labels.csv
--train.record
--test_labels.csv
--test.record
-images/
--test/
---testingimages.jpg + matching xml files
--train/
---testingimages.jpg + matching xml files
--...yourimages.jpg + matching xml files
-xml_to_csv.py
-generate_tfrecord.py
```

Step 3: Setup configuration file and creating a new GCP account

Now keep in mind we will be working with **models** folder. Keep the **Object Detection** folder as well from which we will transfer some files later. Next up, we need to setup a configuration file and then either train a new model or start from a checkpoint with a pre-trained model.

Here, we have two options. We can use a pre-trained model, and then use transfer learning to learn a new object, or we could learn new objects entirely from scratch. The benefit of transfer learning is that training can be much quicker, and the required data that you might need is much less. For this reason, we're going to be doing transfer learning here.

TensorFlow has quite a few pre-trained models with checkpoint files available, along with configuration files. You can do all of this yourself if you like by checking out their configuring jobs documentation.

https://github.com/tensorflow/models/blob/master/research/object_detection/g3doc/configuring_jobs.md

The object API also provides some sample configurations to choose from.

https://github.com/tensorflow/models/blob/master/research/object_detection/g3doc/detection_model_zoo.md

I am going to go with mobilenet, using the following checkpoint and configuration file

```
wget https://raw.githubusercontent.com/tensorflow/models/master/research/object_detection/samples/configs/ssd_mobilenet_v1_pets.config
```

While training on GCP, we do not need to download the config file as it can be found under **.models\research\object_detection\samples\configs**

```
wget http://download.tensorflow.org/models/object_detection/ssd_mobilenet_v1_coco_11_06_2017.tar.gz
```

Extract the above file to **...models/research/object_detection** folder.

To use wget as a keyword on cmd follow the link:

<https://builtvisible.com/download-your-website-with-wget/>

Or, you could just directly download the file without using wget.

At this point, it would be a good idea to create a new GCP account and project.

Use the following link to set up the GCP:

https://github.com/tensorflow/models/blob/master/research/object_detection/g3doc/running_on_cloud.md

Follow this link up until you run a simple TensorFlow python program:

<https://cloud.google.com/ml-engine/docs/getting-started-training-prediction>

Next, make sure to install cloud sdk:

<https://cloud.google.com/sdk/downloads>

And also create a new Google cloud bucket.

<https://cloud.google.com/storage/docs/creating-buckets>

Make sure to select US as the country.

Now, create a new file on pycharm called object-detection.pbtxt and paste your labels in them. It should be saved in the **.../models/object_detection/data** directory

It basically contains just the label in the following format:

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

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

Also, create a new folder called **training** within the **...models/research/object_detection** folder. Copy the object-detection.pbtxt file into the **training** folder as well.

Checkpoint

Now go to your GCS Bucket and create a new folder called **data**. You should have the following files in the GCP storage bucket, under the data folder:

```
+ ${YOUR_GCS_BUCKET}/
+ data/
  - ssd_mobilenet_v1_pets.config
  - model.ckpt.index
  - model.ckpt.meta
  - model.ckpt.data-00000-of-00001
  - object-detection.pbtxt
  - train.record
  - test.record
```

The model files can be found from the

...models\research\object_detection\ssd_mobilenet_v1_coco_11_06_2017 folder locally. The `ssd_mobilenet_v1_pets.config` file needs to be edited further up in the documentation. So wait until you make the edits and then drag and drop it in the data folder later. The `.config` file can be found locally in **...models\research\object_detection\samples\configs**

The `train.record` and `test.record` can be found locally under the folder **Object Detection/data**

`object-detection.pbtxt` can also be found locally under the folder **...models\research\object_detection\training**

Configuration file:

In the configuration file, you need to search for all of the `PATH_TO_BE_CONFIGURED` points and change them. You may also want to modify batch size. Currently, it is set to 24 in my configuration file. Other models may have different batch sizes. If you get a memory error, you can try to decrease the batch size to get the model to fit in your VRAM. Finally, you also need to change the checkpoint name and path.

You could also configure the object detection pipeline for training on GCP by doing the following:

- In the `object_detection/samples/configs` folder, open the `ssd_mobilenet_v1_pets.config`
- Replace all instances of `PATH_TO_BE_CONFIGURED` with `gs://{YOUR_GCP_BUCKET_NAME}/data/`

It's a few edits, so here is my full configuration file:

```
# SSD with Mobilenet v1, configured for the Raccoon dataset.
# Users should configure the fine_tune_checkpoint field in the train config as
# well as the label_map_path and input_path fields in the train_input_reader and
# eval_input_reader. Search for "${YOUR_GCS_BUCKET}" to find the fields that
# should be configured.

model {
  ssd {
    num_classes: 2
    box_coder {
      faster_rcnn_box_coder {
        y_scale: 10.0
        x_scale: 10.0
        height_scale: 5.0
        width_scale: 5.0
      }
    }
  }
  matcher {
    argmax_matcher {
      matched_threshold: 0.5
      unmatched_threshold: 0.5
    }
  }
}
```

```

        ignore_thresholds: false
        negatives_lower_than_unmatched: true
        force_match_for_each_row: true
    }
}
similarity_calculator {
    iou_similarity {
    }
}
}
anchor_generator {
    ssd_anchor_generator {
        num_layers: 6
        min_scale: 0.2
        max_scale: 0.95
        aspect_ratios: 1.0
        aspect_ratios: 2.0
        aspect_ratios: 0.5
        aspect_ratios: 3.0
        aspect_ratios: 0.3333
    }
}
image_resizer {
    fixed_shape_resizer {
        height: 300
        width: 300
    }
}
}
box_predictor {
    convolutional_box_predictor {
        min_depth: 0
        max_depth: 0
        num_layers_before_predictor: 0
        use_dropout: false
        dropout_keep_probability: 0.8
        kernel_size: 1
        box_code_size: 4
        apply_sigmoid_to_scores: false
        conv_hyperparams {
            activation: RELU_6,
            regularizer {
                l2_regularizer {
                    weight: 0.00004
                }
            }
        }
        initializer {
            truncated_normal_initializer {
                stddev: 0.03
                mean: 0.0
            }
        }
        batch_norm {
            train: true,
            scale: true,
            center: true,
            decay: 0.9997,
            epsilon: 0.001,
        }
    }
}
}
}
feature_extractor {
    type: 'ssd_mobilenet_v1'
    min_depth: 16
    depth_multiplier: 1.0
    conv_hyperparams {
        activation: RELU_6,
        regularizer {
            l2_regularizer {

```

```

        weight: 0.00004
    }
}
initializer {
    truncated_normal_initializer {
        stddev: 0.03
        mean: 0.0
    }
}
batch_norm {
    train: true,
    scale: true,
    center: true,
    decay: 0.9997,
    epsilon: 0.001,
}
}
}
loss {
    classification_loss {
        weighted_sigmoid {
            anchorwise_output: true
        }
    }
    localization_loss {
        weighted_smooth_l1 {
            anchorwise_output: true
        }
    }
    hard_example_miner {
        num_hard_examples: 3000
        iou_threshold: 0.99
        loss_type: CLASSIFICATION
        max_negatives_per_positive: 3
        min_negatives_per_image: 0
    }
    classification_weight: 1.0
    localization_weight: 1.0
}
normalize_loss_by_num_matches: true
post_processing {
    batch_non_max_suppression {
        score_threshold: 1e-8
        iou_threshold: 0.6
        max_detections_per_class: 100
        max_total_detections: 100
    }
    score_converter: SIGMOID
}
}
}
train_config: {
    batch_size: 24
    optimizer {
        rms_prop_optimizer {
            learning_rate {
                exponential_decay_learning_rate {
                    initial_learning_rate: 0.004
                    decay_steps: 800720
                    decay_factor: 0.95
                }
            }
            momentum_optimizer_value: 0.9
            decay: 0.9
            epsilon: 1.0
        }
    }
}
}

```

```

fine_tune_checkpoint: "gs://farmweed/data/model.ckpt"
from_detection_checkpoint: true
data_augmentation_options {
  random_horizontal_flip {
  }
}
data_augmentation_options {
  ssd_random_crop {
  }
}
}

train_input_reader: {
  tf_record_input_reader {
    input_path: "gs://farmweed/data/train.record"
  }
  label_map_path: "gs://farmweed/data/object-detection.pbtxt"
}

eval_config: {
  num_examples: 40
}

eval_input_reader: {
  tf_record_input_reader {
    input_path: "gs://farmweed/data/test.record"
  }
  label_map_path: "gs://farmweed/data/object-detection.pbtxt"
  shuffle: false
  num_readers: 1
}

```

Modify cloud parameters settings

Edit the following file: *models/research/object_detection/samples/cloud/cloud.yml*

```

trainingInput:
  runtimeVersion: "1.4"
  scaleTier: CUSTOM
  masterType: standard_gpu
  workerCount: 5
  workerType: standard_gpu
  parameterServerCount: 3
  parameterServerType: standard

```

Modify the following files in case if you run into any errors related to Matplotlib later on:

Change setup.py to the following:

```

"""Setup script for object_detection."""

import logging
import subprocess
from setuptools import find_packages
from setuptools import setup
from setuptools.command.install import install

class CustomCommands(install):

    def RunCustomCommand(self, command_list):
        p = subprocess.Popen(
            command_list,
            stdin=subprocess.PIPE,
            stdout=subprocess.PIPE,

```

```

        stderr=subprocess.STDOUT)
        stdout_data, _ = p.communicate()
        logging.info('Log command output: %s', stdout_data)
        if p.returncode != 0:
            raise RuntimeError('Command %s failed: exit code: %s' %
                               (command_list, p.returncode))

    def run(self):
        self.RunCustomCommand(['apt-get', 'update'])
        self.RunCustomCommand(
            ['apt-get', 'install', '-y', 'python-tk'])
        install.run(self)

REQUIRED_PACKAGES = ['Pillow>=1.0', 'protobuf>=3.3.0', 'Matplotlib>=2.1']

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',
    cmdclass={
        'install': CustomCommands,
    }
)

```

The setup.py file can be found in **models/research** folder.

In **models/research/object_detection/utils/visualization_utils.py**, line 24 (before import matplotlib.pyplot as plt) add:

```

import matplotlib
matplotlib.use('agg')

```

In line 184 of **models/research/object_detection/evaluator.py**, change

```
tf.train.get_or_create_global_step()
```

to

```
tf.contrib.framework.get_or_create_global_step()
```

Finally, in line 103 of **models/research/object_detection/builders/optimizer_builder.py**, change

```
tf.train.get_or_create_global_step()
```

to

```
tf.contrib.framework.get_or_create_global_step()
```

Step 4: Training and Evaluation Jobs

Having modified all the required parameter files, we can now build and ship it off to Google ML for processing:

Build the packages:

change directory to models/research and run the following:

```
python setup.py sdist
cd slim
python setup.py sdist
```

This will create python packages in **dist/object_detection-0.1.tar.gz** and **slim/dist/slim-0.1.tar.gz**. Make sure to run the above three lines individually.

Run the training job

Note: being in Aus, we can't use ML from our local australia-southeast1 data center hence we are using us-central1.

Make sure to cd back into the **models/research** directory

```
# From models directory
gcloud ml-engine jobs submit training object_detection_${version_unique_ID} \
  --job-dir=gs://${YOUR_GCS_BUCKET}/train \
  --packages dist/object_detection-0.1.tar.gz,slim/dist/slim-0.1.tar.gz \
  --module-name object_detection.train \
  --region us-centrall \
  --config object_detection/samples/cloud/cloud.yml \
  -- \
  --train_dir=gs://${YOUR_GCS_BUCKET}/train \
  --
pipeline_config_path=gs://${YOUR_GCS_BUCKET}/data/ssd_mobilenet_v1_pets.config
```

In my case it was:

```
gcloud ml-engine jobs submit training object_detection_188007 --job-dir=gs://farmweed/train --packages dist/object_detection-0.1.tar.gz,slim/dist/slim-0.1.tar.gz --module-name object_detection.train --region us-centrall --config object_detection/samples/cloud/cloud.yml -- --train_dir=gs://farmweed/train -- pipeline_config_path=gs://farmweed/data/ssd_mobilenet_v1_pets.config
```

If you had followed everything up until this point, you could just copy my code and change your GCS bucket name from farmweed to your GCS bucket name.

Run the Evaluation Job

The eval job can be run in conjunction with training. However you will most likely have to increase the amount of resources allocated to your ML platform to run them in tandem.

```
# From models directory
gcloud ml-engine jobs submit training object_detection_EVAL_${version_unique_ID} \
  --job-dir=gs://${YOUR_GCS_BUCKET}/train \
  --packages dist/object_detection-0.1.tar.gz,slim/dist/slim-0.1.tar.gz \
  --module-name object_detection.eval \
  --region us-central1 \
  --scale-tier BASIC_GPU \
  -- \
  --checkpoint_dir=gs://${YOUR_GCS_BUCKET}/train \
  --eval_dir=gs://${YOUR_GCS_BUCKET}/eval \
  --
pipeline_config_path=gs://${YOUR_GCS_BUCKET}/data/ssd_mobilenet_v1_pets.config
```

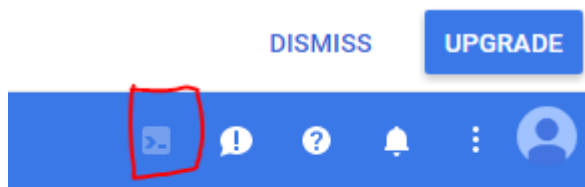
Note: the version unique ID can be anything of your choice.

In my case it was:

```
gcloud ml-engine jobs submit training object_detection_EVAL_188007 --job-dir=gs://farmweed/train --packages dist/object_detection-0.1.tar.gz,slim/dist/slim-0.1.tar.gz --module-name object_detection.eval --region us-central1 --scale-tier BASIC_GPU -- --checkpoint_dir=gs://farmweed/train --eval_dir=gs://farmweed/eval --pipeline_config_path=gs://farmweed/data/ssd_mobilenet_v1_pets.config
```

Monitor the jobs using tensorboard

Open the Google cloud SDK shell, top right hand corner of the GCP.



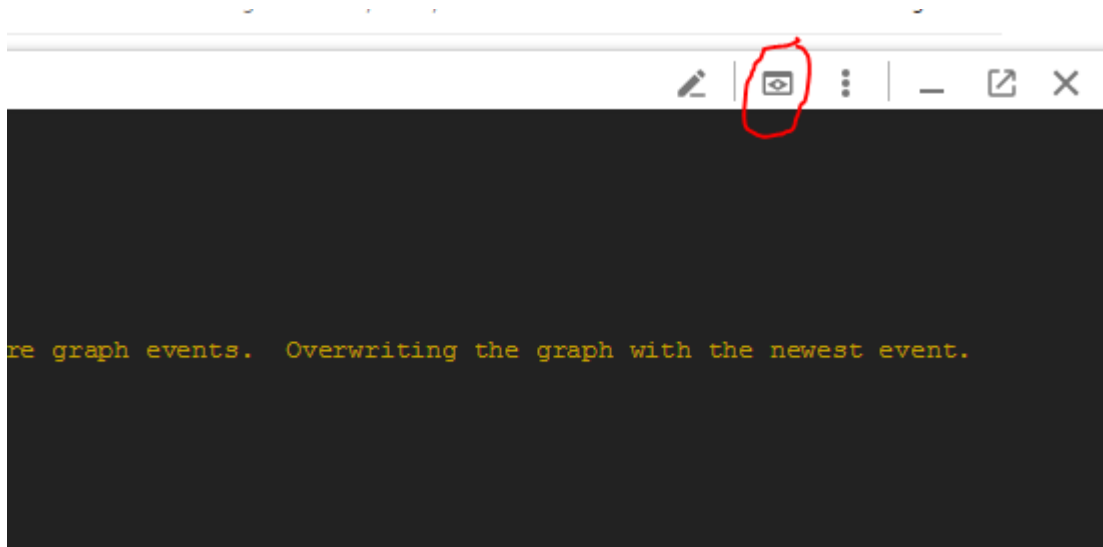
Run the following:

```
tensorboard --logdir=gs://${GCP_BUCKET_NAME} --port=8080
```

In my case it was:

```
tensorboard --logdir=gs://farmweed --port=8080
```

Open the tensorboard dashboard using the preview feature in the console



You should be able to check in on your training and eval models as they are being processed. Navigate on the tensorboard to check the accuracy, total loss and other parameters as they are being trained. There is a drop down list on the right top corner to navigate between scalars, images and other data.

If this is the first time you are running the model, it might take a while to populate, so be patient. Also, don't forget to stop your training job once you are satisfied with the results! Use Ctrl + C to stop the training.

Step 5: Exporting Inference Graph and Testing Results

We are going to test our model and see if it does what we had hoped. In order to do this, we need to export the inference graph.

Luckily for us, in the **models/object_detection** directory, there is a script that does this for us: `export_inference_graph.py`

To run this, you just need to pass in your checkpoint and your pipeline config, then wherever you want the inference graph to be placed. For example:

```
python export_inference_graph.py \
  --input_type image_tensor \
  --pipeline_config_path training/ssd_mobilenet_v1_pets.config \
  --trained_checkpoint_prefix training/model.ckpt-10856 \
  --output_directory inference_graph
```

You can download the checkpoint files from GCP under **train** folder. Move the checkpoint files into the **models/research/object_detection/training** directory. Make sure to rename all three files (index, data and meta) such that it starts with name model. Just look for the one with the largest step (the largest number after the dash), and that's the one you want to use. Next, make sure the `pipeline_config_path` is set to whatever config file you chose. Finally, choose the name for the output directory. It can be anything.

Run the above command from **models/research/object_detection** directory.

In my case it was:

```
python export_inference_graph.py --input_type image_tensor --pipeline_config_path
samples/configs/ssd_mobilenet_v1_pets.config --trained_checkpoint_prefix
training\model.ckpt-58658 --output_directory crop_weed_graph1
```

After running it, you should have a new directory, in my case, mine is `crop_weed_graph1`, inside it, I have new checkpoint data, a `saved_model` directory, and, most importantly, the `forzen_inference_graph.pb` file.

Now, we're just going to use the sample notebook, edit it, and see how our model does on some testing images. I copied some of my test images from **Object Detection/Images/Test** into the **models/research/object_detection/test_images** directory, and renamed them to be `image3.jpg`, `image4.jpg`...etc.

We can boot up jupyter notebook from **models/research/object_detection** folder by just typing 'jupyter notebook' on cmd. It should open up on your browser if successful.

Booting up jupyter notebook and opening the `object_detection_tutorial.ipynb`, let's make a few changes. First, head to the Variables section, and let's change the model name, and the paths to the checkpoint and the labels:

```
# What model to download.
MODEL_NAME = 'crop_weed_graph1'

# Path to frozen detection graph. This is the actual model that
# is used for the object detection.
PATH_TO_CKPT = MODEL_NAME + '/frozen_inference_graph.pb'

# List of the strings that is used to add correct label for each
# box.
PATH_TO_LABELS = os.path.join('training', 'object-
detection.pbtxt')

NUM_CLASSES = 2
```

Next, we can just delete the entire Download Model section, since we don't need to download anymore.

Finally, in the Detection section, change the `TEST_IMAGE_PATHS` var to:

```
TEST_IMAGE_PATHS = [ os.path.join(PATH_TO_TEST_IMAGES_DIR,
'image{}.jpg'.format(i)) for i in range(3, 8) ]
```

With that, you can go to the Cell menu option, and then "Run All."

After some time it should populate the results.