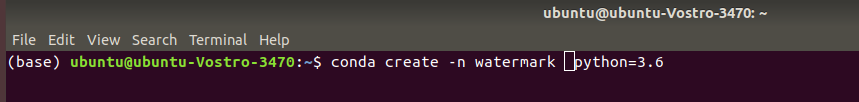
**Reference Documents for Mask RCNN model (watermark)**

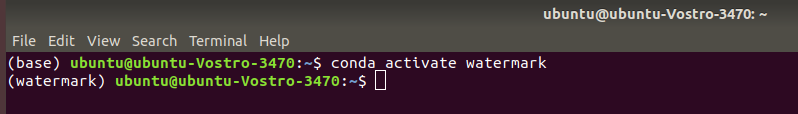
**Overview**

1. Installation
   1. Create a new Conda virtual environment (Optional)

conda create -n mask python=3.6

Screenshot:

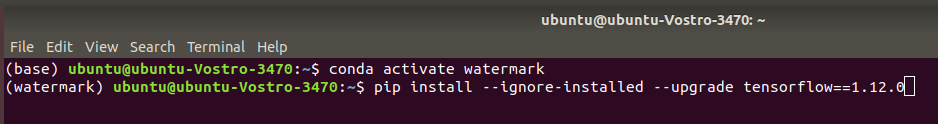




* 1. Install TensorFlow CPU for Python

pip install --ignore-installed --upgrade tensorflow==1.12.0

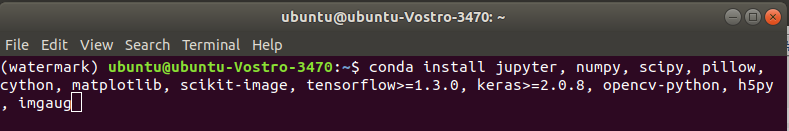
Screenshot:



* 1. Install Prerequisites

conda install jupyter, numpy, scipy, pillow, cython, matplotlib, scikit-image, tensorflow>=1.3.0, keras>=2.0.8, opencv-python, h5py, imgaug

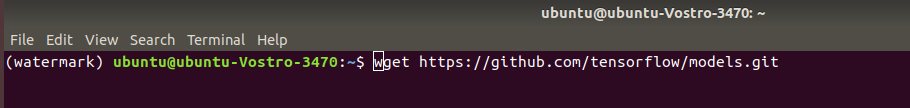
Screenshot:



1. Downloading the TensorFlow Models

wget <https://github.com/tensorflow/models.git>

Screenshot:



1. Protobuf Compilation

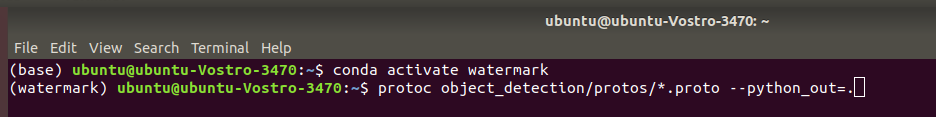
The Tensorflow Object Detection API uses Protobufs to configure model and training parameters. Before the framework can be used, the Protobuf libraries must be downloaded and compiled.

This should be done as follows:

* Head to the [protoc releases page](https://github.com/google/protobuf/releases)

protoc object\_detection/protos/\*.proto --python\_out=.

Screenshots:



1. Adding necessary Environment Variables

export PYTHONPATH=$PYTHONPATH:<PATH\_TO\_TF>/TensorFlow/models/research/

export PYTHONPATH=$PYTHONPATH:<PATH\_TO\_TF>/TensorFlow/models/research/object\_detection

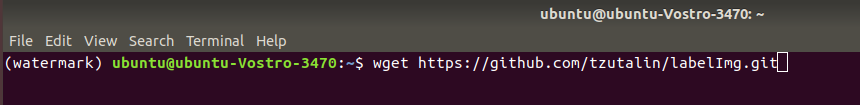
export PYTHONPATH=$PYTHONPATH:<PATH\_TO\_TF>/TensorFlow/models/research/slim

1. LabelImg Installation

* To annotate images we will be using the [labelImg](https://github.com/tzutalin/labelImg) package. If you haven’t installed the package yet, then download from below link.

wget <https://github.com/tzutalin/labelImg.git>

Screenshot:



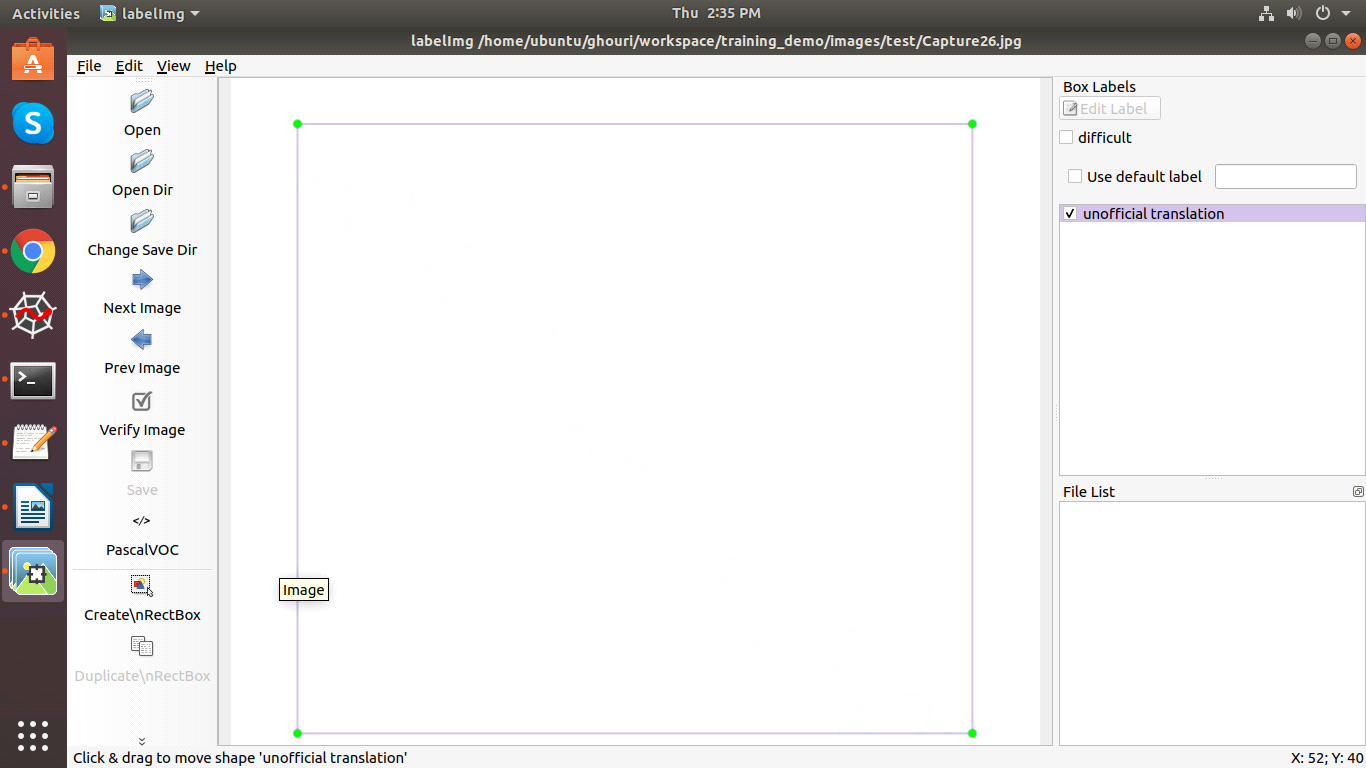
* Once you have collected all the images to be used to test your model (ideally more than 100 per class).
* Open a new Anaconda/Command Prompt window and cd into labelImg.
* Next go ahead and start labelImg, pointing it to your images folder.

python labelImg.py ..\..\images

Screenshot:

* A File Explorer Dialog windows should open, which points to the 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 images\train folder. Similarly, copy all testing images, with their \*.xml files, and paste them inside images\train.

Screenshots:



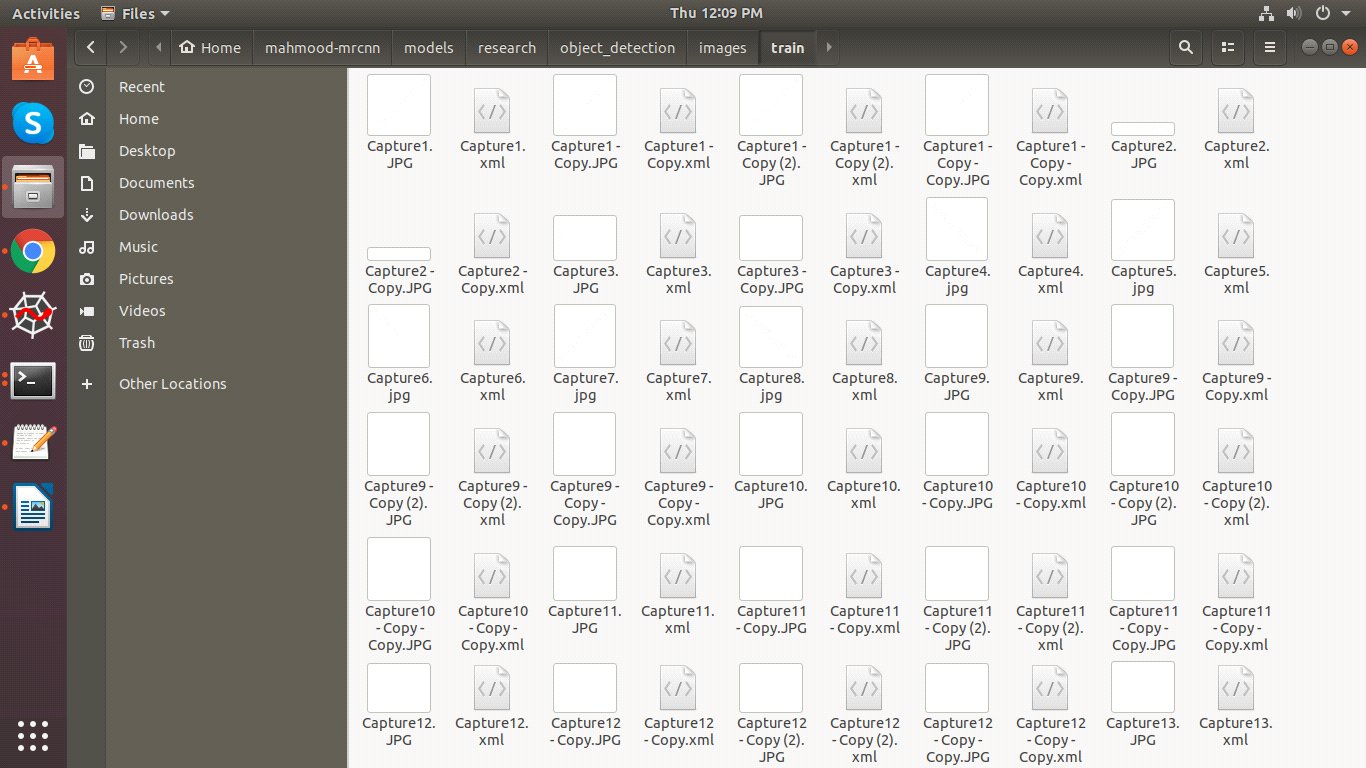
1. Gathering data

Before we can get started creating the object detector we need data, which we can use for training.

To train a robust classifier, we need a lot of pictures which should differ a lot from each other. So they should have different backgrounds, random object, and varying lighting conditions.

You can either take the pictures yourself or you can download them from the internet. For my microcontroller detector, I took about 25 pictures of each individual microcontroller and 25 pictures containing multiple microcontrollers.

Screenshots:



1. Generating TFRecords for training

With the images labeled, we need to create TFRecords that can be served as input data for training of the object detector. In order to create the TFRecords we will use two scripts from racoon dataset github. Namely the *xml\_to\_csv.py* and *generate\_tfrecord.py* files.

After downloading both scripts we can first of change the main method in the xml\_to\_csv file so we can transform the created xml files to csv correctly.

# Old:

def main():

image\_path = os.path.join(os.getcwd(), 'annotations')

xml\_df = xml\_to\_csv(image\_path)

xml\_df.to\_csv('raccoon\_labels.csv', index=None)

print('Successfully converted xml to csv.')

# New:

def main():

for folder in ['train', 'test']:

image\_path = os.path.join(os.getcwd(), ('images/' + folder))

xml\_df = xml\_to\_csv(image\_path)

xml\_df.to\_csv(('images/'+folder+'\_labels.csv'), index=None)

print('Successfully converted xml to csv.')

These creates two files in the images directory. One called *test\_labels.csv* and another one called *train\_labels.csv*.

Before we can transform the newly created files to TFRecords we need to change a few lines in the *generate\_tfrecords.py* file

From:

# TO-DO replace this with label map  
def class\_text\_to\_int(row\_label):  
 if row\_label == 'basketball':  
 return 1  
 elif row\_label == 'shirt':  
 return 2  
 elif row\_label == 'shoe':  
 return 3  
 else:  
 return None

To:

def class\_text\_to\_int(row\_label):  
 if row\_label == 'watermark\_label\_name':

return 1  
 else:  
 return None

If you are using a different dataset you need to replace the class-names with your own.

Now the TFRecords can be generated by typing:

python generate\_tfrecord.py --csv\_input=images\train\_labels.csv --image\_dir=images\train –output\_path=train.record

python generate\_tfrecord.py --csv\_input=images\test\_labels.csv --image\_dir=images\test --output\_path=test.record

These two commands generate a *train.record* and a *test.record* file which can be used to train our object detector.

1. Training model

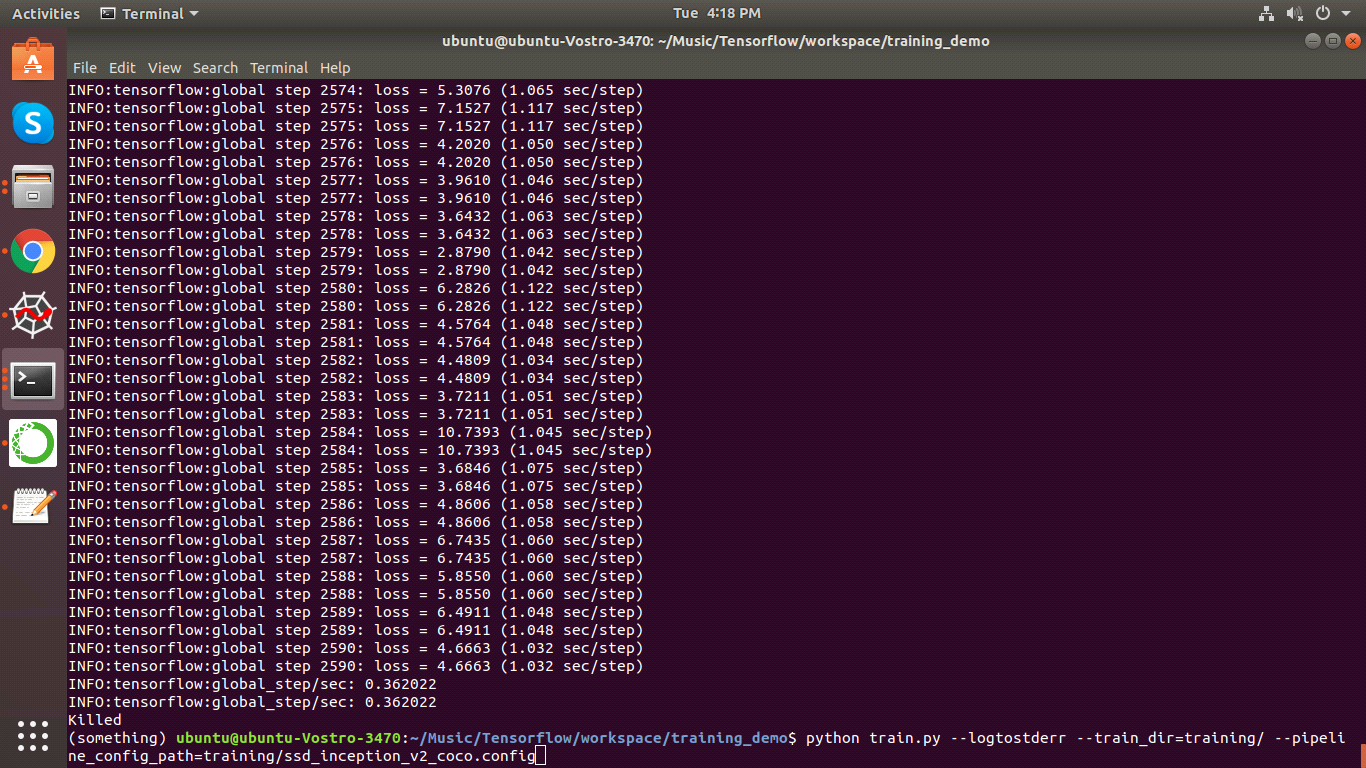
To train the model we will use the *train.py* file, which is located in the *object\_detection/legacy* folder. We will copy it into the *object\_detection* folder and then we will open a command line and type:

Update: Use the model\_main file in the object\_detection folder instead

python model\_main.py –logtostderr –train\_dir=training/ --pipeline\_config\_path=training/faster\_rcnn\_inception\_v2\_pets.config

If everything was setup correctly the training should begin shortly.

Screenshot:



1. Exporting inference graph

Now that we have a trained model we need to generate an inference graph, which can be used to run the model. For doing so we need to first of find out the highest saved step number. For this, we need to navigate to the training directory and look for the model.ckpt file with the biggest index.

Then we can create the inference graph by typing the following command in the command line.

python export\_inference\_graph.py --input\_type image\_tensor --pipeline\_config\_path training/mask\_rcnn\_inception\_v2\_pets.config --trained\_checkpoint\_prefix training/model.ckpt-XXXX --output\_directory inference\_graph

XXXX represents the highest number.

1. Testing object detector

In order to test our newly created object detector, we need to run our object detection code with our input image. Now we can run all the cells and we will see a new prediction.

Screenshots:

