Hello and welcome to a miniseries and introduction to the [**TensorFlow Object Detection API**](https://github.com/tensorflow/models/tree/master/research/object_detection). 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 you can train on your own (which the API also makes easier).

To begin, you're going to want to make sure you have TensorFlow and all of the dependencies. For CPU TensorFlow, you can just do pip install tensorflow, but, of course, the GPU version of TensorFlow is much faster at processing so it is ideal. If you need to install GPU TensorFlow:

Installing GPU TensorFlow links:

* [**GPU TensorFlow on Ubuntu tutorial**](https://pythonprogramming.net/how-to-cuda-gpu-tensorflow-deep-learning-tutorial/)
* [**GPU TensorFlow on Windows tutorial**](https://www.youtube.com/watch?v=r7-WPbx8VuY)

If you do not have a powerful enough GPU to run the GPU version of TensorFlow, one option is to use [**PaperSpace**](https://goo.gl/h7SSkv). Using that link should give you $10 in credit to get started, giving you ~10-20 hours of use.

Beyond this, the other Python dependencies are covered with:

pip install pillow

pip install lxml

pip install jupyter

pip install matplotlib

Next, we need to clone the github. We can do this with git, or you can just download the repository to .zip:

git clone https://github.com/tensorflow/models.git OR click the green "clone or download" button on the [**https://github.com/tensorflow/models**](https://github.com/tensorflow/models) page, download the .zip, and extract it.

Once you have the models directory (or models-master if you downloaded and extracted the .zip), navigate to that directory in your terminal/cmd.exe. The next steps are slightly different on Ubuntu vs Windows.

**On Ubuntu:**

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

And...

export PYTHONPATH=$PYTHONPATH:`pwd`:`pwd`/slim

If you get an error on the protoc command on Ubuntu, check the version you are running with protoc --version, if it's not the latest version, you might want to update. As of my writing of this, we're using 3.4.0. In order to update or get protoc, head to the [**protoc releases page**](https://github.com/google/protobuf/releases). Download the python version, extract, navigate into the directory and then do:

sudo ./configure

sudo make check

sudo make install

After that, try the protoc command again (again, make sure you are issuing this from the models dir).

**On Windows:**

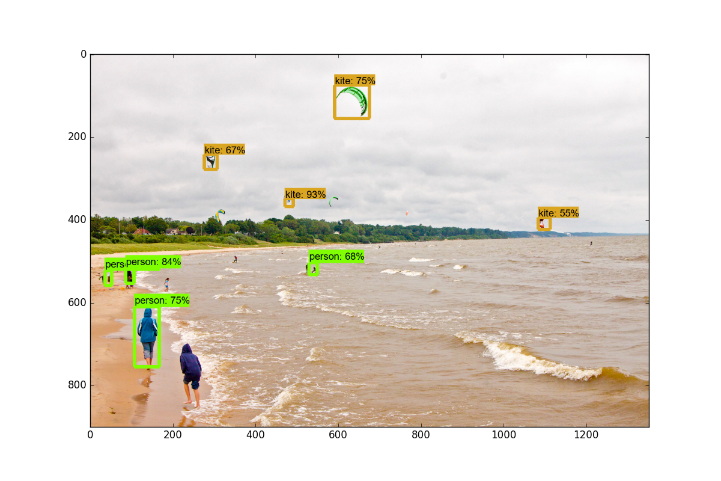
Head to the [**protoc releases page**](https://github.com/google/protobuf/releases) and download the protoc-3.4.0-win32.zip, extract it, and you will find protoc.exe in the bin directory.

You can move this to something more appropriate if you like, or leave it here. I eventually put mine in program files, making a "protoc" directory and dropping it in there.

Now, from within the models (or models-master) directory, you can use the protoc command like so:

"C:/Program Files/protoc/bin/protoc" object\_detection/protos/\*.proto --python\_out=.

Next, open terminal/cmd.exe from the models/object\_detection directory and open the Jupyter Notebook with jupyter notebook. From here, choose the object\_detection\_tutorial.ipynb. From here, you should be able to cell in the main menu, and choose run all

You should get the following results:

In the next tutorial, we'll cover how we can label data live from a webcam stream by modifying this sample code slightly.

Going from using the pre-built models to adding custom objects is a decent jump from my findings, and I could not locate any full step-by-step guides, so hopefully I can save you all from the struggle. Once solved, the ability to train for any custom object you can think of (and create data for) is an awesome skill to have.

Alright, so a brief overview of the steps needed to do this:

1. Collect a few hundred images that contain your object - The bare minimum would be about 100, ideally more like 500+, but, the more images you have, the more tedious step 2 is...
2. Annotate/label the images, ideally with a program. I personally used [**LabelImg**](https://github.com/tzutalin/labelImg). This process is basically drawing boxes around your object(s) in an image. The label program automatically will create an XML file that describes the object(s) in the pictures.
3. Split this data into train/test samples
4. Generate TF Records from these splits
5. Setup a .config file for the model of choice (you could train your own from scratch, but we'll be using transfer learning)
6. Train
7. Export graph from new trained model
8. Detect custom objects in real time!
9. ...
10. Profit!

So, for this tutorial, I needed an object. I wanted something useful, but that wasn't already done. Obviously, everyone needs to know where the macaroni and cheese is, so let's track that!

I used Google Images, Bing, and ImageNet to collect some images of Mac n Cheese. In general, pictures around the size of 800x600, not too large and not too small.

For this tutorial, you can track \*anything\* you want, you just need 100+ images. Once you have images, you need to annotate them. For this, I am going to use [**LabelImg**](https://github.com/tzutalin/labelImg), you can grab it with git clone https://github.com/tzutalin/labelImg, or just download and extract the zip.

Installation instructions are on the [**labelimg github**](https://github.com/tzutalin/labelImg), but for Python3 on Ubuntu:

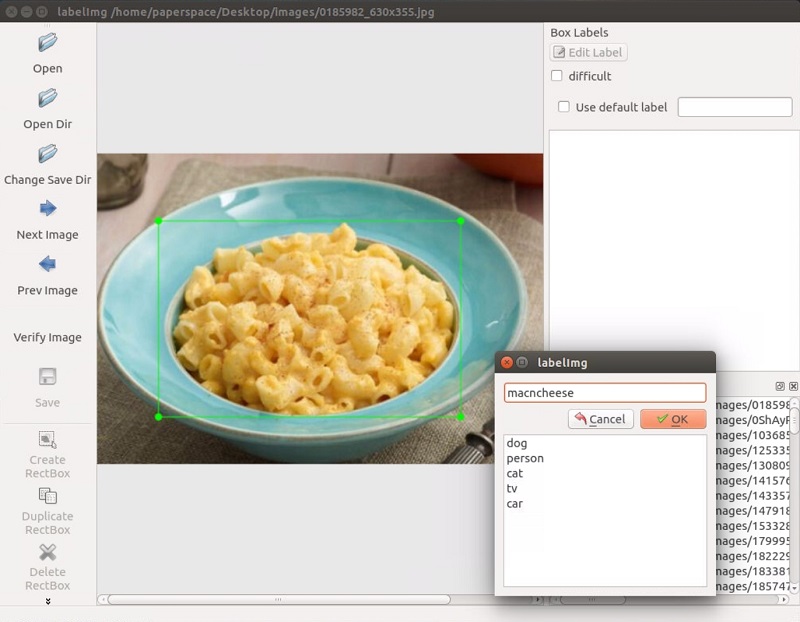
sudo apt-get install pyqt5-dev-tools

sudo pip3 install lxml

make qt5py3

python3 labelImg.py

When running this, 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. Not sure if there's a shortcut for the next image.



Once you have over 100 images labeled, we're going to separate them into training and testing groups. To do this, just copy about 10% of your images and their annotation XML files to a new dir called test and then copy the remaining ones to a new dir called train

Once you've done all of this, you're ready to go to the next tutorial, where we're going to cover how we can create the required TFRecord files from this data.

Alternatively, if you would like to just use my pre-made files, you can download my [**labeled macaroni and cheese**](https://pythonprogramming.net/static/downloads/machine-learning-data/object-detection-macaroni.zip).

Welcome to part 4 of the TensorFlow Object Detection API tutorial series. In this part of the tutorial, we're going to cover how to create the TFRecord files that we need to train an object detection model.

At this point, you should have an images directory, inside of that has all of your images, along with 2 more diretories: train and test. Inside the test directory should be a copy of ~10% of your images with their XML annotation data, and then the training directory should have a copy of the rest. If you do not have this, go to the previous tutorial.

Now we need to convert these XML files to singular CSV files that can be then converted to the TFRecord files. To do this, I am going to make use of some of the code from [**datitran's github**](https://github.com/datitran/raccoon_dataset), with some minor changes. To begin, we're going to use [**xml\_to\_csv.py**](https://github.com/datitran/raccoon_dataset/blob/master/xml_to_csv.py). You can either clone his entire directory or just grab the files, we'll be using two of them. Since his repository has changed multiple breaking times since I've been messing with it, I will note that the exact commit that I've been playing with is: [**here**](https://github.com/datitran/raccoon_dataset/commit/386a8f4f1064ea0fe90cfac8644e0dba48f0387b). If either of these two scripts aren't working for you, try pulling from the same commit as me. Definitely try his latest versions though. For example, at the time of my writing this, he has just updated for multiple box labels in images, which is obviously a very useful improvement.

Within the xml\_to\_csv script, I changed:

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.')

To:

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.')

This just handles for the train/test split and naming the files something useful. Go ahead and make a data directory, and run this to create the two files. Next, create a training directory from within the main Object-Detection dir. At this point, you should have the following structure, and it is on my Desktop:

Object-Detection

-data/

--test\_labels.csv

--train\_labels.csv

-images/

--test/

---testingimages.jpg

--train/

---testingimages.jpg

--...yourimages.jpg

-training

-xml\_to\_csv.py

Now, grab [**generate\_tfrecord.py**](https://github.com/datitran/raccoon_dataset/blob/master/generate_tfrecord.py). The only modification that you will need to make here is in the class\_text\_to\_int function. You need to change this to your specific class. In our case, we just have ONE class. If you had many classes, then you would need to keep building out this if statement.

# TO-DO replace this with label map

def class\_text\_to\_int(row\_label):

if row\_label == 'macncheese':

return 1

else:

None

Judging by that to-do, this function may change quite a bit in the future, so, again, use your intuition to modify the latest version, or go to the same commit that I am using.

Next, in order to use this, we need to either be running from within the models directory of the cloned models github, or we can more formally install the object detection API.

I am doing this tutorial on a fresh machine to be certain I don't miss any steps, so I will be fully setting up the Object API. If you've already cloned and setup, feel free to skip the initial steps and pick back up on the setup.py part!

First, I am cloning the repository to my desktop:

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

Then, following the installation instructions:

sudo apt-get install protobuf-compiler python-pil python-lxml

sudo pip install jupyter

sudo pip install matplotlib

And then:

# From tensorflow/models/

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

If you get an error on the protoc command on Ubuntu, check the version you are running with protoc --version, if it's not the latest version, you might want to update. As of my writing of this, we're using 3.4.0. In order to update or get protoc, head to the [**protoc releases page**](https://github.com/google/protobuf/releases). Download the python version, extract, navigate into the directory and then do:

sudo ./configure

sudo make check

sudo make install

After that, try the protoc command again (again, make sure you are issuing this from the models dir).

and

# From tensorflow/models/

export PYTHONPATH=$PYTHONPATH:`pwd`:`pwd`/slim

Finally, let's install the object\_dection library formally by doing the following from within the models directory:

sudo python3 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.

python3 generate\_tfrecord.py --csv\_input=data/train\_labels.csv --output\_path=data/train.record

python3 generate\_tfrecord.py --csv\_input=data/test\_labels.csv --output\_path=data/test.record

**Update:** As of Jan 12 2019, one of my viewers pointed out the above commands now require an additional flag: --image\_dir. So, instead, you should do:

python3 generate\_tfrecord.py --csv\_input=data/train\_labels.csv --output\_path=data/train.record --image\_dir=images/

python3 generate\_tfrecord.py --csv\_input=data/test\_labels.csv --output\_path=data/test.record --image\_dir=images/

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

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, which is what we'll be covering in the next tutorial.

Welcome to part 4 of the TensorFlow Object Detection API tutorial series. In this part of the tutorial, we're going to cover how to create the TFRecord files that we need to train an object detection model.

At this point, you should have an images directory, inside of that has all of your images, along with 2 more diretories: train and test. Inside the test directory should be a copy of ~10% of your images with their XML annotation data, and then the training directory should have a copy of the rest. If you do not have this, go to the previous tutorial.

Now we need to convert these XML files to singular CSV files that can be then converted to the TFRecord files. To do this, I am going to make use of some of the code from [**datitran's github**](https://github.com/datitran/raccoon_dataset), with some minor changes. To begin, we're going to use [**xml\_to\_csv.py**](https://github.com/datitran/raccoon_dataset/blob/master/xml_to_csv.py). You can either clone his entire directory or just grab the files, we'll be using two of them. Since his repository has changed multiple breaking times since I've been messing with it, I will note that the exact commit that I've been playing with is: [**here**](https://github.com/datitran/raccoon_dataset/commit/386a8f4f1064ea0fe90cfac8644e0dba48f0387b). If either of these two scripts aren't working for you, try pulling from the same commit as me. Definitely try his latest versions though. For example, at the time of my writing this, he has just updated for multiple box labels in images, which is obviously a very useful improvement.

Within the xml\_to\_csv script, I changed:

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.')

To:

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.')

This just handles for the train/test split and naming the files something useful. Go ahead and make a data directory, and run this to create the two files. Next, create a training directory from within the main Object-Detection dir. At this point, you should have the following structure, and it is on my Desktop:

Object-Detection

-data/

--test\_labels.csv

--train\_labels.csv

-images/

--test/

---testingimages.jpg

--train/

---testingimages.jpg

--...yourimages.jpg

-training

-xml\_to\_csv.py

Now, grab [**generate\_tfrecord.py**](https://github.com/datitran/raccoon_dataset/blob/master/generate_tfrecord.py). The only modification that you will need to make here is in the class\_text\_to\_int function. You need to change this to your specific class. In our case, we just have ONE class. If you had many classes, then you would need to keep building out this if statement.

# TO-DO replace this with label map

def class\_text\_to\_int(row\_label):

if row\_label == 'macncheese':

return 1

else:

None

Judging by that to-do, this function may change quite a bit in the future, so, again, use your intuition to modify the latest version, or go to the same commit that I am using.

Next, in order to use this, we need to either be running from within the models directory of the cloned models github, or we can more formally install the object detection API.

I am doing this tutorial on a fresh machine to be certain I don't miss any steps, so I will be fully setting up the Object API. If you've already cloned and setup, feel free to skip the initial steps and pick back up on the setup.py part!

First, I am cloning the repository to my desktop:

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

Then, following the installation instructions:

sudo apt-get install protobuf-compiler python-pil python-lxml

sudo pip install jupyter

sudo pip install matplotlib

And then:

# From tensorflow/models/

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

If you get an error on the protoc command on Ubuntu, check the version you are running with protoc --version, if it's not the latest version, you might want to update. As of my writing of this, we're using 3.4.0. In order to update or get protoc, head to the [**protoc releases page**](https://github.com/google/protobuf/releases). Download the python version, extract, navigate into the directory and then do:

sudo ./configure

sudo make check

sudo make install

After that, try the protoc command again (again, make sure you are issuing this from the models dir).

and

# From tensorflow/models/

export PYTHONPATH=$PYTHONPATH:`pwd`:`pwd`/slim

Finally, let's install the object\_dection library formally by doing the following from within the models directory:

sudo python3 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.

python3 generate\_tfrecord.py --csv\_input=data/train\_labels.csv --output\_path=data/train.record

python3 generate\_tfrecord.py --csv\_input=data/test\_labels.csv --output\_path=data/test.record

**Update:** As of Jan 12 2019, one of my viewers pointed out the above commands now require an additional flag: --image\_dir. So, instead, you should do:

python3 generate\_tfrecord.py --csv\_input=data/train\_labels.csv --output\_path=data/train.record --image\_dir=images/

python3 generate\_tfrecord.py --csv\_input=data/test\_labels.csv --output\_path=data/test.record --image\_dir=images/

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

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, which is what we'll be covering in the next tutorial.

Welcome to part 5 of the TensorFlow Object Detection API tutorial series. In this part of the tutorial, we will train our object detection model to detect our custom object. To do this, we need the Images, matching TFRecords for the training and testing data, and then we need to setup the configuration of the model, then we can train. For us, that means we need to setup a configuration file.

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**](https://github.com/tensorflow/models/blob/master/object_detection/g3doc/configuring_jobs.md) documentation. The object API also provides some [**sample configurations**](https://github.com/tensorflow/models/tree/master/research/object_detection/samples/configs) to choose from.

I am going to go with mobilenet, using the following [**checkpoint**](http://download.tensorflow.org/models/object_detection/ssd_mobilenet_v1_coco_11_06_2017.tar.gz) and [**configuration file**](https://github.com/tensorflow/models/blob/master/object_detection/samples/configs/ssd_mobilenet_v1_pets.config)

wget https://raw.githubusercontent.com/tensorflow/models/master/object\_detection/samples/configs/ssd\_mobilenet\_v1\_pets.config

wget http://download.tensorflow.org/models/object\_detection/ssd\_mobilenet\_v1\_coco\_11\_06\_2017.tar.gz

You can check out some of the other checkpoint options to start with [**here**](https://github.com/tensorflow/models/blob/master/research/object_detection/g3doc/detection_model_zoo.md).

Put the config in the training directory, and extract the ssd\_mobilenet\_v1 in the models/object\_detection directory

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/path, num\_classes to 1, num\_examples to 12, and label\_map\_path: "training/object-detect.pbtxt"

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

# SSD with Mobilenet v1, configured for the mac-n-cheese 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: 1

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: 10

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: "ssd\_mobilenet\_v1\_coco\_11\_06\_2017/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: "data/train.record"

}

label\_map\_path: "data/object-detection.pbtxt"

}

eval\_config: {

num\_examples: 40

}

eval\_input\_reader: {

tf\_record\_input\_reader {

input\_path: "data/test.record"

}

label\_map\_path: "training/object-detection.pbtxt"

shuffle: false

num\_readers: 1

}

Inside training dir, add object-detection.pbtxt:

item {

id: 1

name: 'macncheese'

}

And now, the moment of truth! From within models/object\_detection:

python3 train.py --logtostderr --train\_dir=training/ --pipeline\_config\_path=training/ssd\_mobilenet\_v1\_pets.config

Barring errors, you should see output like:

INFO:tensorflow:global step 11788: loss = 0.6717 (0.398 sec/step)

INFO:tensorflow:global step 11789: loss = 0.5310 (0.436 sec/step)

INFO:tensorflow:global step 11790: loss = 0.6614 (0.405 sec/step)

INFO:tensorflow:global step 11791: loss = 0.7758 (0.460 sec/step)

INFO:tensorflow:global step 11792: loss = 0.7164 (0.378 sec/step)

INFO:tensorflow:global step 11793: loss = 0.8096 (0.393 sec/step)

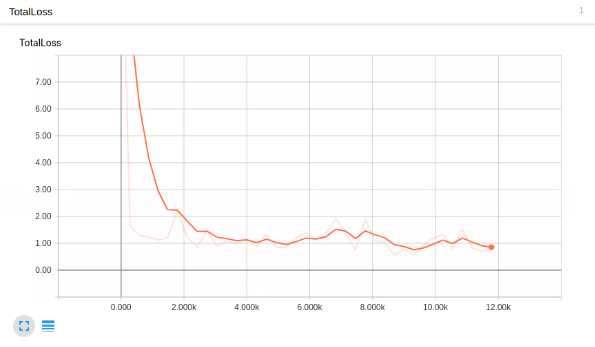
Your steps start at 1 and the loss will be much higher. Depending on your GPU and how much training data you have, this process will take varying amounts of time. On something like a 1080ti, it should take only about an hour or so. If you have a lot of training data, it might take much longer. You want to shoot for a loss of about ~1 on average (or lower). I wouldn't stop training until you are for sure under 2. You can check how the model is doing via TensorBoard. Your models/object\_detection/training directory will have new event files that can be viewed via TensorBoard.

From models/object\_detection, via terminal, you start TensorBoard with:

tensorboard --logdir='training'

This runs on 127.0.0.1:6006 (visit in your browser)

My total loss graph:



Looks good enough, but does it detect macaroni and cheese?!

In order to use the model to detect things, we need to export the graph, so, in the next tutorial, we're going to export the graph and then test the model.

Welcome to part 6 of the TensorFlow Object Detection API tutorial series. In this part of the tutorial, 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:

python3 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 mac\_n\_cheese\_inference\_graph

Your checkpoint files should be in the training directory. 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, and then finally choose the name for the output directory, I went with mac\_n\_cheese\_inference\_graph

Run the above command from models/object\_detection

If you get an error about no module named 'nets', then you need to re run:

# From tensorflow/models/

export PYTHONPATH=$PYTHONPATH:`pwd`:`pwd`/slim

# switch back to object\_detection after this and re run the above command

Otherwise, you should have a new directory, in my case, mine is mac\_n\_cheese\_inference\_graph, 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 models/object\_detection/images/test images into the models/object\_detection/test\_images directory, and renamed them to be image3.jpg, image4.jpg...etc.

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 = 'mac\_n\_cheese\_inference\_graph'

# 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 = 1

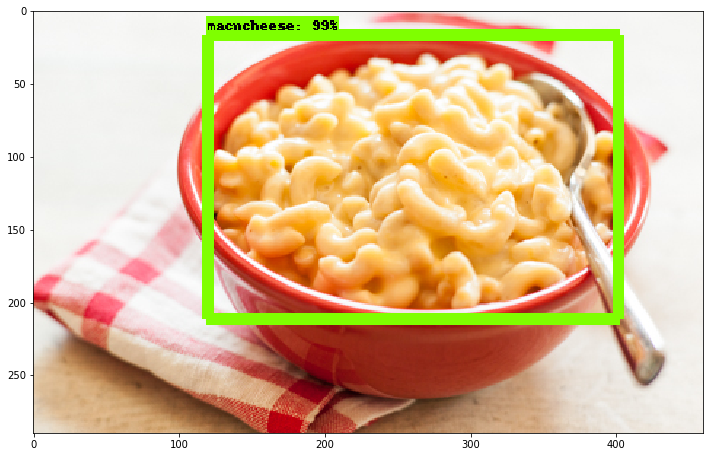
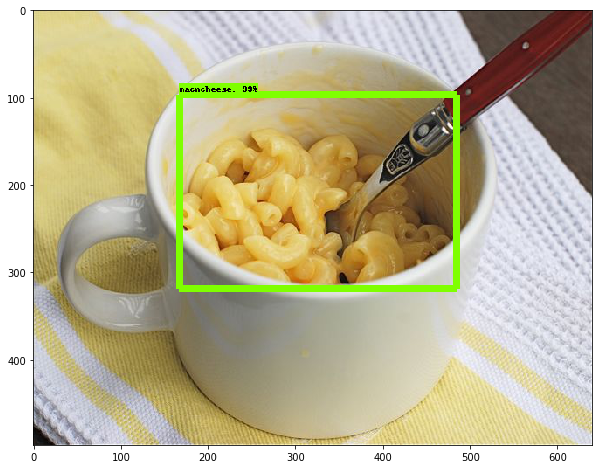
Next, we can just delete the entire Download Model section, since we don't need to download anymore.

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."

Here are a few of my results:



Overall, I am extremely pleased at how well this all works, and, even when you have a very small dataset, you can still have success, and only need to train a model for about an hour (on a decent GPU anyway) using transfer learning. Very cool!

CREDITS : <https://pythonprogramming.net/introduction-use-tensorflow-object-detection-api-tutorial/>