

# 1. How to train an object detector classifier for multiple objects using tensorflow(GPU) on Windows 10

Once we have installed Tensorflow, Cuda and CuDNN, we can pass to the next level! The purpose of this post is to explain how to train your own convolutional neural network object detection classifier for multiple objects. In this post we do not develop a neural network of zero, we will only take this github:

[https://github.com/tensorflow/models/blob/master/research/object\\_detection/g3doc/detection\\_model\\_zoo.md](https://github.com/tensorflow/models/blob/master/research/object_detection/g3doc/detection_model_zoo.md)

It provides a collection of detection models pre-trained on datasets of different kinds of objects. They are also useful for initializing your models when training on novel datasets like our dataset of artworks. At the end of this post, you will have the idea to develop your own model that can identify and program that can draw boxes around specific items in that case 12 items objects of different artworks, like *Gioconda*, *Athens School*, *Las Meninas*, *The Last Supper*.



There are several good tutorials available for how to use TensorFlow's Object Detection API to train a classifier for a single object, like this one:

<https://codelabs.developers.google.com/codelabs/tensorflow-for-poets/#0>

The post is written for Windows 10. But without forgetting Linux, only emphasize that the development process is very similar and can also be used for Linux operating systems, but the file paths and package installation commands should be modified accordingly. Only tell you before start that TensorFlow-GPU allows your PC or VM to use the video card to provide extra processing power while training, so it will be used for this post, Regular Tensorflow not. In my experience doing this experiment, using TensorFlow-GPU instead of regular TensorFlow reduces training time by a factor of about 8hr using Fast-RCNN model, 3 hours to train instead of 8 hours with SSD-Mobilenet model for 3 objects, 21 hours for 12 objects. You can search different models to tensorflow here:

[https://github.com/tensorflow/models/blob/master/research/object\\_detection/g3doc/detection\\_model\\_zoo.md](https://github.com/tensorflow/models/blob/master/research/object_detection/g3doc/detection_model_zoo.md)

Regular TensorFlow can also be used for this post, I have tried it in my PC, but it will take longer. If you use regular TensorFlow, you do not need to install CUDA and cuDNN like in the first post **[Link to post]**.

## **1. Installing TensorFlow-GPU**

Install Tensorflow-GPU following the instructions of the first post. Download CUDA 9.0 and CuDNN 7.1.1, but you will likely need to continue updating the CUDA and CuDNN versions to the latest supported version.

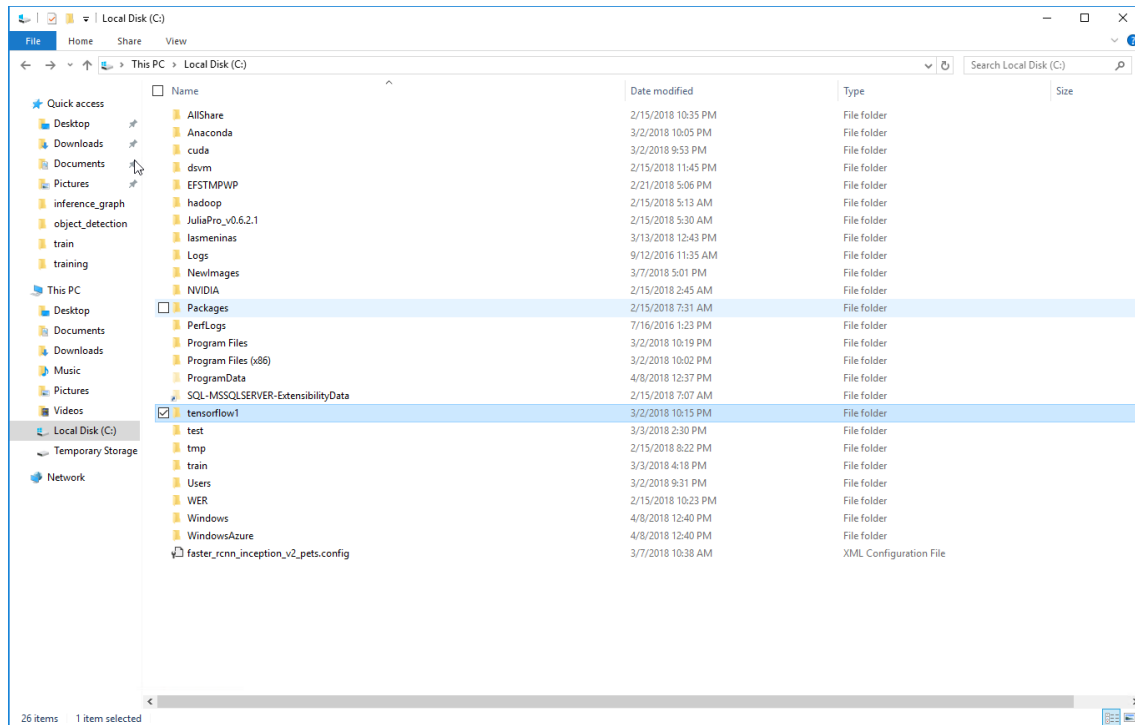
## **2. Setting up the Object Detection directory structure and Anaconda Virtual Environment**

The TensorFlow Object Detection API requires using the specific directory structure provided in its GitHub repository. It also requires several additional Python packages, specific additions to the PATH and PYTHONPATH variables, and a few extra setup commands to get everything set up to run or train an object detection model. Stay tuned at each step because the processes are very meticulous and the minimum failure can give us error.

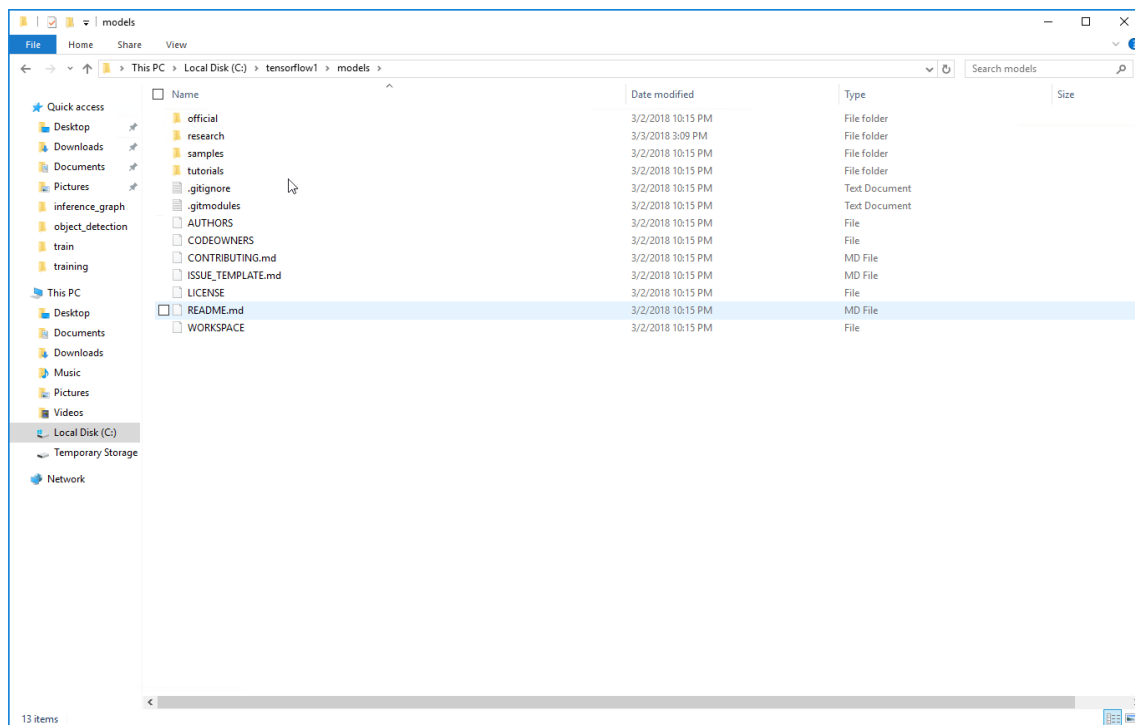
### **Download TensorFlow Object Detection API repository from GitHub**

Create a folder directly in C: and name it "tensorflow" or whatever you want. This working directory will contain the full TensorFlow object detection framework, as well as your training images, training data, trained classifier, configuration files, and everything else needed for the object detection classifier.

Download the full TensorFlow object detection repository located at <https://github.com/tensorflow/models> by clicking the “Clone or Download” button and downloading the zip file. Open the downloaded zip file and extract the “models-master” folder directly into your C:\DIRECTORY you just created. Rename “models-master” to just “models”.




*Image of the principal directory*



*Image of Tensorflow Object Detection API directory*



## COCO-trained models {#coco-models}



Model name	Speed (ms)	COCO mAP[^1]	Outputs
ssd_mobilenet_v1_coco	30	21	Boxes
ssd_mobilenet_v2_coco	31	22	Boxes
ssd_inception_v2_coco	42	24	Boxes
faster_rcnn_inception_v2_coco	58	28	Boxes
faster_rcnn_resnet50_coco	89	30	Boxes
faster_rcnn_resnet50_lowproposals_coco	64		Boxes
rfcn_resnet101_coco	92	30	Boxes
faster_rcnn_resnet101_coco	106	32	Boxes
faster_rcnn_resnet101_lowproposals_coco	82		Boxes
faster_rcnn_inception_resnet_v2_atrous_coco	620	37	Boxes
faster_rcnn_inception_resnet_v2_atrous_lowproposals_coco	241		Boxes
faster_rcnn_nas	1833	43	Boxes
faster_rcnn_nas_lowproposals_coco	540		Boxes
mask_rcnn_inception_resnet_v2_atrous_coco	771	36	Masks
mask_rcnn_inception_v2_coco	79	25	Masks
mask_rcnn_resnet101_atrous_coco	470	33	Masks
mask_rcnn_resnet50_atrous_coco	343	29	Masks

*\*Note: What is column mAP? <https://stackoverflow.com/questions/46094282/why-we-use-map-score-for-evaluate-object-detectors-in-deep-learning>*

As we can see in the table SSD-Mobilenet has much more speed than the other models and smaller mAP, we could conclude that it is the best but once my project has been realized I have noticed that for example in this case with art paintings, it has given good results in terms of detection but in terms of accuracy it give me low level of success. On the other hand faster rcnn has surprised me because according to the table it has a very slow detection speed with respect to SSD-Mobilenet but it has gone very well on my MS Surface webcam. At the end to have been analyzing both models decided to train with both and extract their inference graph to use it both on tablets, laptops and mobile or lot devices. You can choose which model to train your objection detection classifier on. If you are planning on using the object detector on a device with low computational like mobile, use the SDD-MobileNet model. If you will be running your object detector on a laptop or desktop PC, use one of the RCNN models.

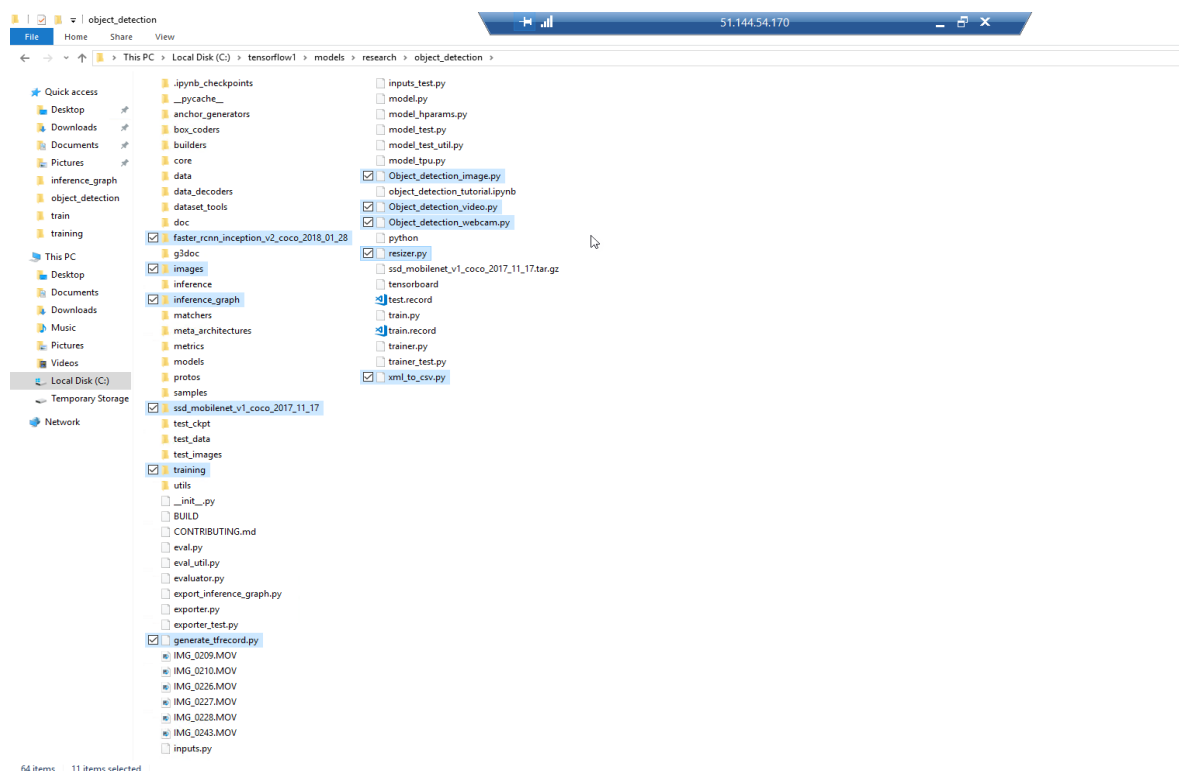
In this post will use the Faster-RCNN-Inception-V2 model and ssd\_mobilenet\_v1\_coco. Download the models and open the downloaded faster\_rcnn\_inception\_v2\_coco\_2018\_01\_28.tar.gz and ssd\_mobilenet\_v1\_coco\_2017\_11\_17.tar.gz file with a file archiver such as WinZip or 7-

Zip and extract the faster\_rcnn\_inception\_v2\_coco\_2018\_01\_28 and ssd\_mobilenet\_v1\_coco\_2017\_11\_17 folder to the C:\tensorflow1\models\research\object\_detection folder.

*\*Note: The models date and versions will likely change in the future, but it should still work with this tutorial.*

## Download this tutorial's repository from GitHub

Download the full repository located on this [page](#), scroll to the top and click Clone or Download and extract all the contents directly into the C:\tensorflow1\models\research\object\_detection directory. This establishes a specific directory structure that will be used for the resto of the post. At this point, your \object\_detection folder should look like:



### *Final directory with Object Detection API with 11 extra-files(selected)*

This repository contains the images, annotation data, .csv files, and TFRecords needed to train a "Artworks" objects detector. It also contains Python scripts that are used to generate the training data. It has scripts to test out the object detection classifier on images, videos, or a webcam feed.

If you want to practice training your own "Artwork" Detector, you can leave all the files as they are. You can follow along with this tutorial to see how each of the files were generated, and then run the training. You will still need to generate the TFRecord files train.record and test.record as described in nexts steps.

You can also use the frozen inference graph for my trained Artwork model detector from the I and extract the contents to \object\_detection\inference\_graph. This inference graph will work "out of the box". You can test it after all the setup instructions have been completed by running the Object\_detection\_image.py or video or webcam script.

If you want to train your own object detector, delete the following files (do not delete the folders):

- All files in \object\_detection\images\train and \object\_detection\images\test
- The "test\_labels.csv" and "train\_labels.csv" files in \object\_detection\images
- All files in \object\_detection\training
- All files in \object\_detection\inference\_graph

Now, you are ready to start from scratch in training your own object detector. This post will assume that all the files listed above were deleted, and will go on to explain how to generate the files for your own training dataset.

## Set up new Anaconda virtual environment

Next, we'll work on setting up a virtual environment in Anaconda for tensorflow-gpu. From the Start menu in Windows, search for the Anaconda Prompt utility, right click on it, and click "Run as Administrator".

Activate the environment of tensorflow by issuing:

```
C:\> activate tensorflow
```

Install the other necessary packages by issuing the following commands:

```
(tensorflow) C:\> conda install -c anaconda protobuf
(tensorflow) C:\> pip install pillow
(tensorflow) C:\> pip install lxml
(tensorflow) C:\> pip install jupyter
(tensorflow) C:\> pip install matplotlib
(tensorflow) C:\> pip install pandas
(tensorflow) C:\> pip install opencv-python
```

*\*Note: The 'pandas' and 'opencv-python' packages are not needed by TensorFlow, but they are used in the Python scripts to generate TFRecords and to work with images, videos, and webcam feeds.*

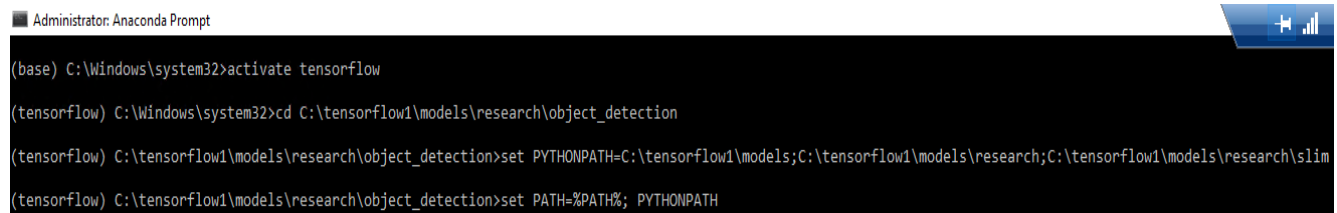
## Configure PATH and PYTHONPATH environment variables

The PATH variable must be configured to add the \models, \models\research, and \models\research\slim directories. For some reason, you need to create a PYTHONPATH variable with these directories, and then add PYTHONPATH to the PATH



variable. Otherwise, it will not work. Do this by issuing the following commands (from any directory):

```
(tensorflow) C:\> set PYTHONPATH=C:\tensorflow1\models;  
C:\tensorflow1\models\research;C:\tensorflow1\models\research\slim  
(tensorflow) C:\> set PATH=%PATH%; PYTHONPATH  
*Note: Every time the "tensorflow1" virtual environment is exited, the PATH and  
PYTHONPATH variables are reset and need to be set up again.
```



```
Administrator: Anaconda Prompt  
(base) C:\Windows\system32>activate tensorflow  
(tensorflow) C:\Windows\system32>cd C:\tensorflow1\models\research\object_detection  
(tensorflow) C:\tensorflow1\models\research\object_detection>set PYTHONPATH=C:\tensorflow1\models;C:\tensorflow1\models\research;C:\tensorflow1\models\research\slim  
(tensorflow) C:\tensorflow1\models\research\object_detection>set PATH=%PATH%; PYTHONPATH
```

## Compile Protobufs and run setup.py

Now we need to compile the Protobuf files, which are used by TensorFlow to configure model and training parameters. Unfortunately, the short protoc compilation command posted on TensorFlow's Object Detection API [installation page](#) does not work on Windows. Every .proto file in the \object\_detection\protos directory must be called out individually by the command.

To understand what the protobuf files are (*Protocol Buffer, Mechanism for serializing structured data by Google*):

- [https://www.tensorflow.org/extend/tool\\_developers/](https://www.tensorflow.org/extend/tool_developers/)
- <https://developers.google.com/protocol-buffers/?hl=en>

In the Anaconda Command Prompt, change directories to the \models\research directory and copy and paste the following command into the command line and press Enter:

```
protoc --python_out=. .\object_detection\protos\anchor_generator.proto  
.\object_detection\protos\argmax_matcher.proto  
.\object_detection\protos\bipartite_matcher.proto .\object_detection\protos\box_coder.proto  
.\object_detection\protos\box_predictor.proto .\object_detection\protos\eval.proto  
.\object_detection\protos\faster_rcnn.proto  
.\object_detection\protos\faster_rcnn_box_coder.proto  
.\object_detection\protos\grid_anchor_generator.proto  
.\object_detection\protos\hyperparams.proto .\object_detection\protos\image_resizer.proto  
.\object_detection\protos\input_reader.proto .\object_detection\protos\losses.proto  
.\object_detection\protos\matcher.proto  
.\object_detection\protos\mean_stddev_box_coder.proto .\object_detection\protos\model.proto  
.\object_detection\protos\optimizer.proto .\object_detection\protos\pipeline.proto  
.\object_detection\protos\post_processing.proto .\object_detection\protos\preprocessor.proto  
.\object_detection\protos\region_similarity_calculator.proto  
.\object_detection\protos\square_box_coder.proto .\object_detection\protos\ssd.proto  
.\object_detection\protos\ssd_anchor_generator.proto
```



```
. \object_detection\protos\string_int_label_map.proto . \object_detection\protos\train.proto
. \object_detection\protos\keypoint_box_coder.proto
. \object_detection\protos\multiscale_anchor_generator.proto
```

This creates a name\_pb2.py file from every name.proto file in the \object\_detection\protos folder.

*\*Note: TensorFlow occasionally adds new .proto files to the \protos folder. You may need to update the protoc command to include the new .proto files.*

```
(tensorflow) C:\tensorflow1\models\research>protoc --python_out=. . \object_detection\protos\anchor_generator.proto . \object_detection\protos\argmax_matcher.proto . \object_detection\protos\bipartite_matching.proto . \object_detection\protos\box_coder.proto . \object_detection\protos\box_predictor.proto . \object_detection\protos\eval.proto . \object_detection\protos\fast_rcnn.proto . \object_detection\protos\fast_rcnn_box_coder.proto . \object_detection\protos\fast_rcnn_evaluator.proto . \object_detection\protos\hyperparams.proto . \object_detection\protos\image_resizer.proto . \object_detection\protos\input_reader.proto . \object_detection\protos\losses.proto . \object_detection\protos\mean_stddev_box_coder.proto . \object_detection\protos\model.proto . \object_detection\protos\optimizer.proto . \object_detection\protos\pipeline.proto . \object_detection\protos\post_processing.proto . \object_detection\protos\preprocessor.proto . \object_detection\protos\region_similarity_calculator.proto . \object_detection\protos\square_box_coder.proto . \object_detection\protos\ssd.proto . \object_detection\protos\ssd_evaluator.proto . \object_detection\protos\string_int_label_map.proto . \object_detection\protos\train.proto . \object_detection\protos\keypoint_box_coder.proto . \object_detection\protos\multiscale_anchor_generator.proto

(tensorflow) C:\tensorflow1\models\research>.
```

Finally, run the following commands from the C:\tensorflow1\models\research directory:

```
(tensorflow) C:\tensorflow1\models\research> python setup.py build
(tensorflow) C:\tensorflow1\models\research> python setup.py install
```

### 3. Labelling pictures

Now that the TensorFlow Object Detection API is all set up and ready to go, we need to provide the images it will use to train a new detection classifier.

#### Gather Pictures

TensorFlow needs hundreds of images of an object to train a good detection classifier. To train a robust classifier, the training images should have random objects in the image along with the desired objects, and should have a variety of backgrounds and lighting conditions. There should be some images where the desired object is partially obscured, overlapped with something else, or only halfway in the picture.

For my Artwork Detection classifier, I have twelve different objects I want to detect (Diego Velazquez, Maria Agustina, Infanta Margarita, Isaben de Velasco and Mari barbola part of "Las Meninas", Platon, aristoteles and heraclito of "School of Athens", Gioconda of "Mona Lisa" and finally Jesus, Judas and Mateo of "The Last Supper").

I used Google Image Search to find about 80 pictures of each artwork. You can use your phone or download images of the objects from Google Image Search. I recommend having at least 200 pictures overall. I used 310 aprox, pictures to train my artworks detector. In addition to a better training has been cut the complete image of the art box focusing on the object we want to detect.



*Images of the different artworks cropped*

In the case of art paintings, we are lucky that they are static images that will always be presented to our model in the same way, they are not objects with movement or relevant changes of appearance such as an animal, cars, etc. With the images of the art pictures I played when choosing the images for the dataset with the main properties of computer vision, luminosity, size cause. They always appear in the same way, we can only extract real photos with people in front of the picture as another training case.

You can use the `resizer.py` script in this repository to reduce the size of the images cause they should be less than 200KB each, and their resolution shouldn't be more than 720x1280. The larger the images are, the longer it will take to train the classifier.

After you have all the pictures you need, move 20% of them to the `\object_detection\images\test` directory, and 80% of them to the `\object_detection\images\train` directory. Make sure there are a variety of pictures in both the `\test` and `\train` directories.

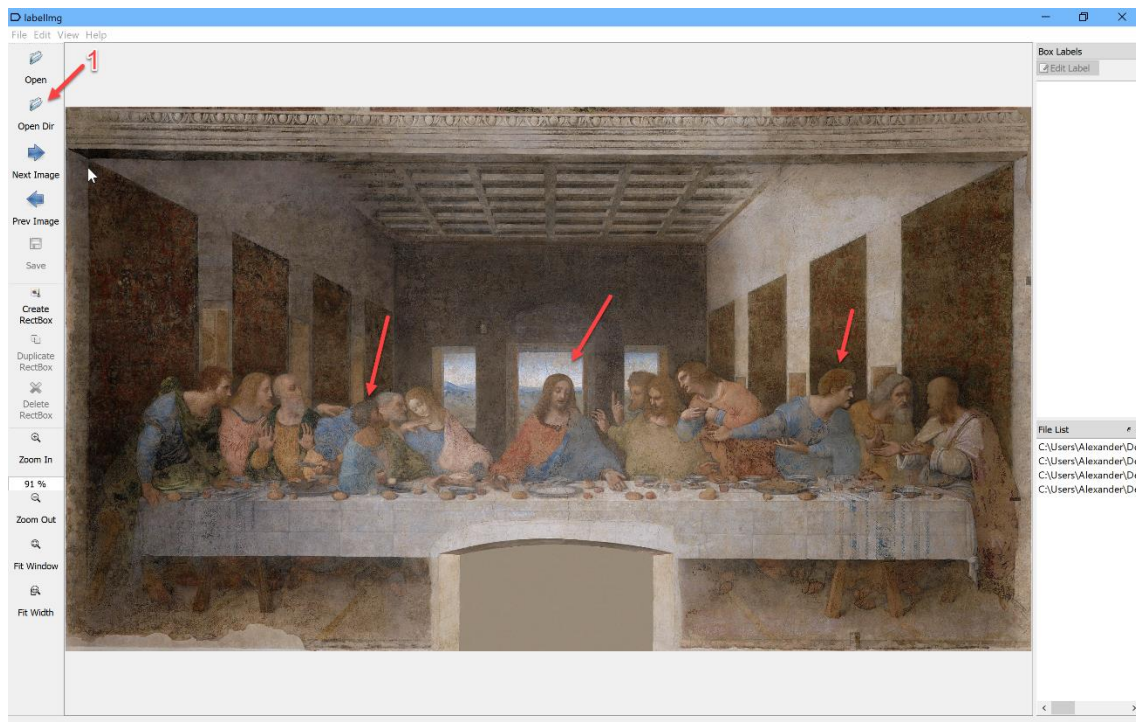
## **Label Pictures**

With all the pictures gathered, it's time to label the desired objects in every picture. Labelling is a great and new tool for me for labeling images, and its GitHub page has very clear instructions on how to install and use it.

[Labellmg GitHub link](#)

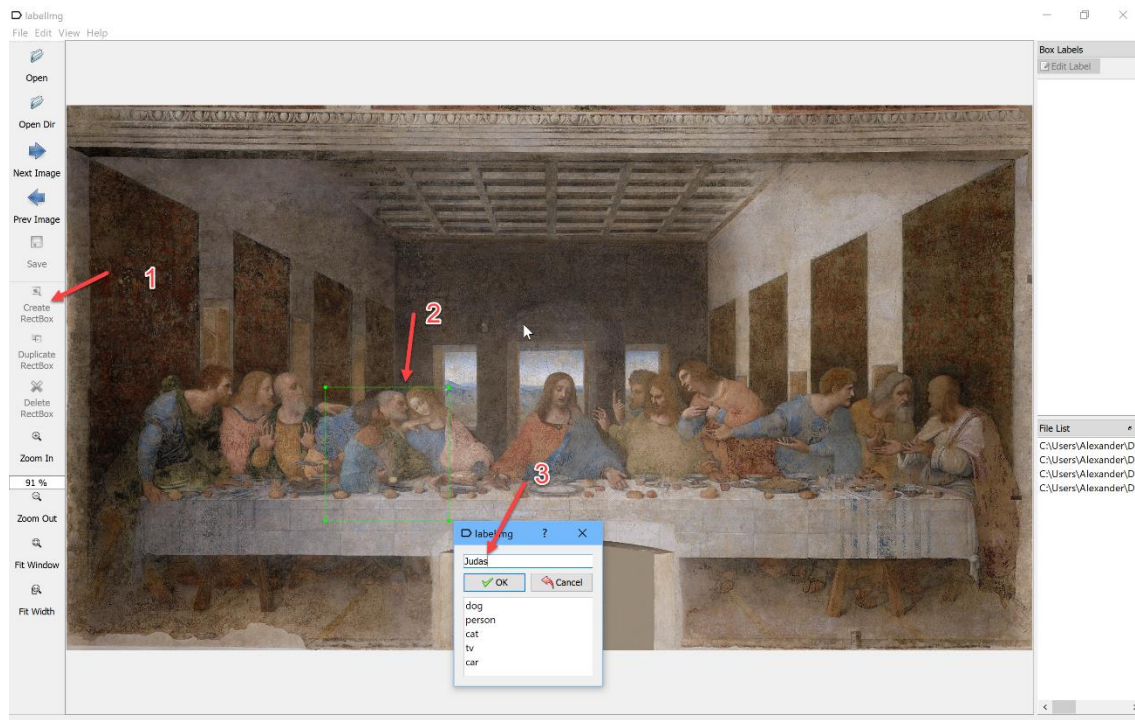
[Labellmg download link](#)

Download and install Labellmg, point it to your \images\train directory, and then draw a box around each object in image. Repeat the process for all the images in the \images\test directory. This will take a while and a lot of patience...



*Image open file directory and observe what objects we want*

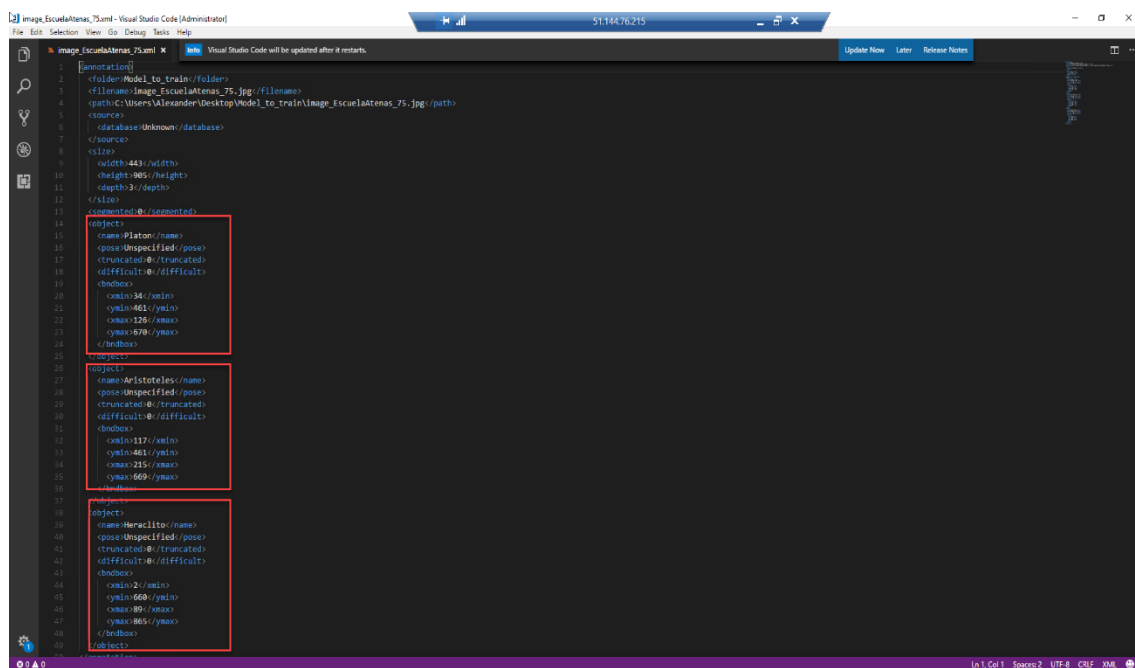




*Image about create RectBox around objects*

## 4. Generating training data

Labelling saves a .xml file containing the label data for each image. These .xml files will be used to generate TFRecords, which are one of the inputs to the TensorFlow trainer. Once you have labeled and saved each image, there will be one .xml file for each image in the \test and \train directories.



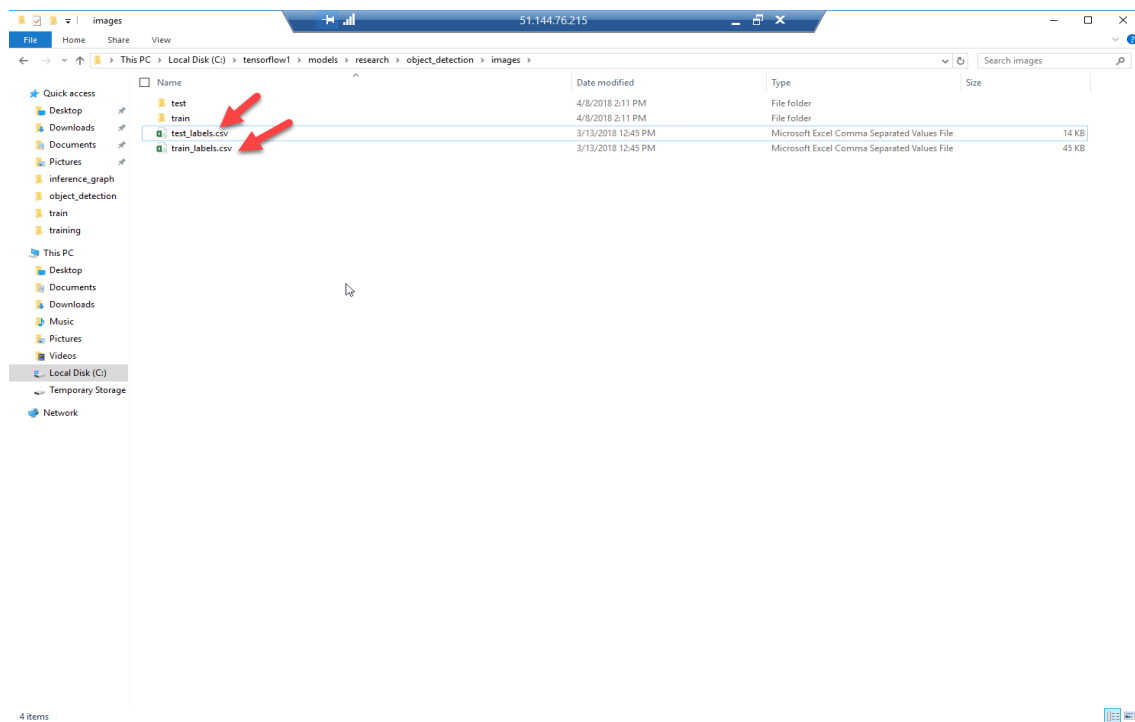
*Image of XML file generate by Labellmg with the objects with their respective coordinates.*

With the images labeled, it's time to generate the TFRecords that serve as input data to the TensorFlow training model. This tutorial uses the `xml_to_csv.py` and `generate_tfrecord.py` scripts from [Dat Tran's Raccoon Detector dataset](#), with some slight modifications to work with our directory structure.

First, the image .xml data will be used to create .csv files containing all the data for the train and test images. From the `\object_detection` folder, issue the following command in the Anaconda command prompt:

```
(tensorflow) C:\tensorflow1\models\research\object_detection> python xml_to_csv.py
```

This creates a `train_labels.csv` and `test_labels.csv` file in the `\object_detection\images` folder.



*Image of train and test csv files from xml file of Labellmg*

A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
1	filename	width	height	class	xmin	ymin	xmax	ymax						
2	image_Esc1	1280	912	Platon	568	467	658	667						
3	image_Esc1	1280	912	Aristoteles	658	467	734	672						
4	image_Esc1	1280	912	Heraclito	453	666	676	888						
5	image_Esc1	1280	688	Platon	568	465	658	670						
6	image_Esc1	1280	688	Aristoteles	648	466	737	669						
7	image_Esc1	1280	912	Platon	566	466	659	666						
8	image_Esc1	1280	912	Aristoteles	660	466	741	669						
9	image_Esc1	1280	912	Heraclito	457	667	654	888						
10	image_Esc1	1280	912	Platon	569	472	661	663						
11	image_Esc1	1280	912	Aristoteles	657	472	736	667						
12	image_Esc1	1280	912	Heraclito	464	675	674	887						
13	image_Esc1	1280	912	Platon	572	470	662	668						
14	image_Esc1	1280	912	Aristoteles	652	467	739	666						
15	image_Esc1	1280	912	Heraclito	465	676	671	890						
16	image_Esc1	1280	912	Platon	569	471	661	663						
17	image_Esc1	1280	912	Aristoteles	650	469	736	666						
18	image_Esc1	1280	912	Heraclito	469	671	665	895						
19	image_Esc1	1280	912	Platon	571	474	662	661						
20	image_Esc1	1280	912	Aristoteles	655	473	742	663						
21	image_Esc1	1280	912	Heraclito	463	672	657	881						
22	image_Esc1	1280	912	Platon	571	470	659	667						
23	image_Esc1	1280	912	Aristoteles	656	470	740	671						
24	image_Esc1	1280	912	Heraclito	471	670	657	887						
25	image_Esc1	1280	912	Platon	572	469	665	668						
26	image_Esc1	1280	912	Aristoteles	661	470	737	668						
27	image_Esc1	1280	912	Heraclito	443	668	667	892						
28	image_Esc1	335	562	Platon	112	135	200	331						
29	image_Esc1	335	562	Aristoteles	199	134	277	332						
30	image_Esc1	335	562	Heraclito	6	335	185	556						
31	image_Esc1	335	562	Platon	106	133	201	331						
32	image_Esc1	335	562	Aristoteles	195	132	276	332						
33	image_Esc1	335	562	Heraclito	11	335	173	557						
34	image_Esc1	335	562	Platon	111	135	202	331						
35	image_Esc1	335	562	Aristoteles	193	134	268	333						
36	image_Esc1	335	562	Heraclito	13	335	176	554						
37	image_Esc1	335	562	Platon	113	137	201	331						
38	image_Esc1	335	562	Aristoteles	192	135	273	331						
39	image_Esc1	335	562	Heraclito	11	335	173	557						
40	image_Esc1	335	562	Platon	111	135	202	331						
41	image_Esc1	335	562	Aristoteles	193	134	268	333						
42	image_Esc1	335	562	Heraclito	13	335	176	554						
43	image_Esc1	335	562	Platon	113	137	201	331						
44	image_Esc1	335	562	Aristoteles	192	135	273	331						
45	image_Esc1	335	562	Heraclito	11	335	173	557						
46	image_Esc1	335	562	Platon	111	135	202	331						
47	image_Esc1	335	562	Aristoteles	193	134	268	333						
48	image_Esc1	335	562	Heraclito	13	335	176	554						
49	image_Esc1	335	562	Platon	113	137	201	331						
50	image_Esc1	335	562	Aristoteles	192	135	273	331						
51	image_Esc1	335	562	Heraclito	11	335	173	557						
52	image_Esc1	335	562	Platon	111	135	202	331						
53	image_Esc1	335	562	Aristoteles	193	134	268	333						
54	image_Esc1	335	562	Heraclito	13	335	176	554						
55	image_Esc1	335	562	Platon	113	137	201	331						
56	image_Esc1	335	562	Aristoteles	192	135	273	331						
57	image_Esc1	335	562	Heraclito	11	335	173	557						
58	image_Esc1	335	562	Platon	111	135	202	331						
59	image_Esc1	335	562	Aristoteles	193	134	268	333						
60	image_Esc1	335	562	Heraclito	13	335	176	554						
61	image_Esc1	335	562	Platon	113	137	201	331						
62	image_Esc1	335	562	Aristoteles	192	135	273	331						
63	image_Esc1	335	562	Heraclito	11	335	173	557						
64	image_Esc1	335	562	Platon	111	135	202	331						
65	image_Esc1	335	562	Aristoteles	193	134	268	333						
66	image_Esc1	335	562	Heraclito	13	335	176	554						
67	image_Esc1	335	562	Platon	113	137	201	331						
68	image_Esc1	335	562	Aristoteles	192	135	273	331						
69	image_Esc1	335	562	Heraclito	11	335	173	557						
70	image_Esc1	335	562	Platon	111	135	202	331						
71	image_Esc1	335	562	Aristoteles	193	134	268	333						
72	image_Esc1	335	562	Heraclito	13	335	176	554						
73	image_Esc1	335	562	Platon	113	137	201	331						
74	image_Esc1	335	562	Aristoteles	192	135	273	331						
75	image_Esc1	335	562	Heraclito	11	335	173	557						
76	image_Esc1	335	562	Platon	111	135	202	331						
77	image_Esc1	335	562	Aristoteles	193	134	268	333						
78	image_Esc1	335	562	Heraclito	13	335	176	554						
79	image_Esc1	335	562	Platon	113	137	201	331						
80	image_Esc1	335	562	Aristoteles	192	135	273	331						
81	image_Esc1	335	562	Heraclito	11	335	173	557						
82	image_Esc1	335	562	Platon	111	135	202	331						
83	image_Esc1	335	562	Aristoteles	193	134	268	333						
84	image_Esc1	335	562	Heraclito	13	335	176	554						
85	image_Esc1	335	562	Platon	113	137	201	331						
86	image_Esc1	335	562	Aristoteles	192	135	273	331						
87	image_Esc1	335	562	Heraclito	11	335	173	557						
88	image_Esc1	335	562	Platon	111	135	202	331						
89	image_Esc1	335	562	Aristoteles	193	134	268	333						
90	image_Esc1	335	562	Heraclito	13	335	176	554						
91	image_Esc1	335	562	Platon	113	137	201	331						
92	image_Esc1	335	562	Aristoteles	192	135	273	331						
93	image_Esc1	335	562	Heraclito	11	335	173	557						
94	image_Esc1	335	562	Platon	111	135	202	331						
95	image_Esc1	335	562	Aristoteles	193	134	268	333						
96	image_Esc1	335	562	Heraclito	13	335	176	554						
97	image_Esc1	335	562	Platon	113	137	201	331						
98	image_Esc1	335	562	Aristoteles	192	135	273	331						
99	image_Esc1	335	562	Heraclito	11	335	173	557						
100	image_Esc1	335	562	Platon	111	135	202	331						
101	image_Esc1	335	562	Aristoteles	193	134	268	333						
102	image_Esc1	335	562	Heraclito	13	335	176	554						
103	image_Esc1	335	562	Platon	113	137	201	331						
104	image_Esc1	335	562	Aristoteles	192	135	273	331						
105	image_Esc1	335	562	Heraclito	11	335	173	557						
106	image_Esc1	335	562	Platon	111	135	202	331						
107	image_Esc1	335	562	Aristoteles	193	134	268	333						
108	image_Esc1	335	562	Heraclito	13	335	176	554						
109	image_Esc1	335	562	Platon	113	137	201	331						
110	image_Esc1	335	562	Aristoteles	192	135	273	331						
111	image_Esc1	335	562	Heraclito	11	335	173	557						
112	image_Esc1	335	562	Platon	111	135	202	331						
113	image_Esc1	335	562	Aristoteles	193	134	268	333						
114	image_Esc1	335	562	Heraclito	13	335	176	554						
115	image_Esc1	335	562	Platon	113	137	201	331						
116	image_Esc1	335	562	Aristoteles	192	135	273	331						
117	image_Esc1	335	562	Heraclito	11	335	173	557						
118	image_Esc1	335	562	Platon	111	135	202	331						
119	image_Esc1	335	562	Aristoteles	193	134	268	333						
120	image_Esc1	335	562	Heraclito	13	335	176	554						
121	image_Esc1	335	562	Platon	113	137	201	331						
122	image_Esc1	335	562	Aristoteles	192	135	273	331						
123	image_Esc1	335	562	Heraclito										

```
    return 9
elif row_label == 'Jesucristo':
    return 10
elif row_label == 'Judas':
    return 11
elif row_label == 'Mateo':
    return 12
else:
    None
```

Then, generate the TFRecord files by issuing these commands from the \object\_detection folder:

```
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 generate a train.record and a test.record file in \object\_detection. These will be used to train the new object detection classifier.

## 5. Creating a label map and configuring training

The last thing to do before training is to create a label map and edit the training configuration file.

### Label map

The label map tells the trainer what each object is by defining a mapping of class names to class ID numbers. Use a text editor to create a new file and save it as labelmap.pbtxt in the C:\tensorflow1\models\research\object\_detection\training folder. *Don't forget that the file type is .pbtxt, not .txt.* In the text editor, copy or type in the label map in the format below. The example below is the label map for my Artworks Detector:

```
item {
  id: 1
  name: 'Diego Velazquez'
}

item {
  id: 2
  name: 'Maria Agustina'
}

item {
  id: 3
  name: 'Infanta Margarita'
}

item {
```



```
id: 4
name: 'Isabel de Velasco'
}

item {
  id: 5
  name: 'Mari Barbola'
}

item {
  id: 6
  name: 'Platon'
}

item {
  id: 7
  name: 'Aristoteles'
}

item {
  id: 8
  name: 'Heraclito'
}

item {
  id: 9
  name: 'Gioconda'
}

item {
  id: 10
  name: 'Jesucristo'
}

item {
  id: 11
  name: 'Judas'
}

item {
  id: 12
  name: 'Mateo'
}
```

## Configure training

Finally, the object detection training pipeline must be configured. It defines which model and what parameters will be used for training. This is the last step before running training ☺

In this step we can do the same with SSD-Mobilenet and Faster-RCNN. First we will do the training with Faster-RCNN and then I will explain how it would be done with the Mobilenet model.

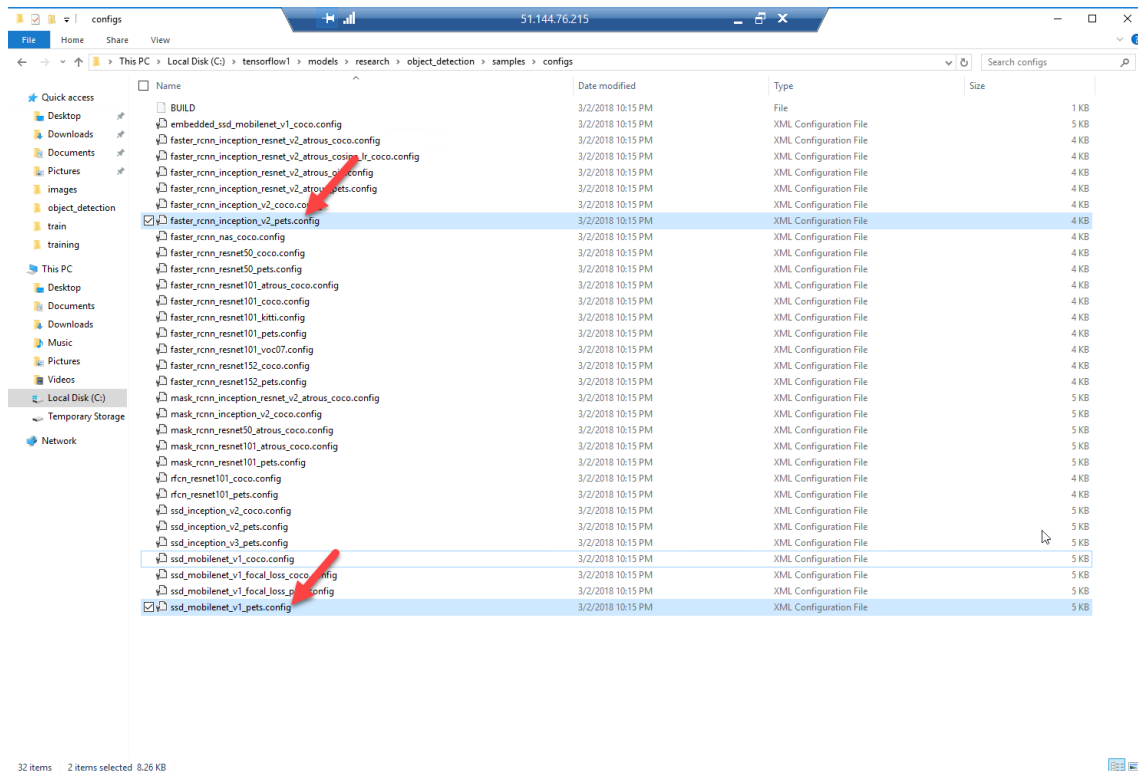
Navigate to C:\tensorflow1\models\research\object\_detection\samples\configs and copy the faster\_rcnn\_inception\_v2\_pets.config file into the \object\_detection\training directory. Then, open the file with a text editor. There are several changes to make to the .config file, mainly changing the number of classes and examples, and adding the file paths to the training data.

Make the following changes to the faster\_rcnn\_inception\_v2\_pets.config file. Note: The paths must be entered with single forward slashes (NOT backslashes), or TensorFlow will give a file path error when trying to train the model. Also, the paths must be in double quotation marks ( " ), not single quotation marks ( ' ).

- Line 9. Change num\_classes to the number of different objects you want the classifier to detect. For the above basketball, shirt, and shoe detector, it would be num\_classes : 3 .
- Line 110. Change fine\_tune\_checkpoint to:
  - fine\_tune\_checkpoint :  
"C:/tensorflow1/models/research/object\_detection/faster\_rcnn\_inception\_v2\_coco\_2018\_01\_28/model.ckpt"
- Lines 126 and 128. In the train\_input\_reader section, change input\_path and label\_map\_path to:
  - input\_path : "C:/tensorflow1/models/research/object\_detection/train.record"
  - label\_map\_path:  
"C:/tensorflow1/models/research/object\_detection/training/labelmap.pbtxt"
- Line 132. Change num\_examples to the number of images you have in the \images\test directory.
- Lines 140 and 142. In the eval\_input\_reader section, change input\_path and label\_map\_path to:
  - input\_path : "C:/tensorflow1/models/research/object\_detection/test.record"
  - label\_map\_path:  
"C:/tensorflow1/models/research/object\_detection/training/labelmap.pbtxt"

Save the file after the changes have been made. Great! Now we can start the training job!

For Mobilenet of other model we have to do the same process, get the config file of each model and change the parameters that I explained before. The model of SSD-Mobilenet you can find it here like Faster-RCNN:



*Image directory of Tensorflow pre-trained models(Coco or Pets datasets)*

## 6. Training

From the \object\_detection directory, issue the following command to begin training:

```
python train.py --logtostderr --train_dir=training/ --  
pipeline_config_path=training/faster_rcnn_inception_v2_pets.config
```

If we want to use SSD-Mobilenet to train your model, you only need to change the config\_path:

```
python train.py --logtostderr --train_dir=training/ --  
pipeline_config_path=training/ssd_mobilenet_v1_pets.config
```

If everything has been set up correctly, TensorFlow will initialize the training. When training begins, it will look like this:

```
Administrator Anaconda Prompt - python train.py --logtostderr --train_dir=training/ --pipeline_config_path=training/faster... 51.144.76.215
Instructions for updating:
Please switch to tf.train.create_global_step
INFO:tensorflow:Scale of 0 disables regularizer.
INFO:tensorflow:Scale of 0 disables regularizer.
INFO:tensorflow:depth of additional conv before box predictor: 0
WARNING:tensorflow:From C:\tensorflow\models\research\object_detection\core\box_predictor.py:391: calling reduce_mean (from tensorflow.python.ops.math_ops) with keep_dims is deprecated and will be removed in a future version.
Instructions for updating:
keep_dims is deprecated, use keepdims instead
WARNING:tensorflow:From C:\tensorflow\models\research\object_detection\core\losses.py:306: softmax_cross_entropy_with_logits (from tensorflow.python.ops.nn_ops) is deprecated and will be removed in a future version.
Instructions for updating:
Future major versions of TensorFlow will allow gradients to flow into the labels input on backprop by default.
See tf.nn.softmax_cross_entropy_with_logits_v2.

WARNING:tensorflow:From C:\tensorflow\models\research\object_detection\meta_architectures\ Faster_Rcnn_Meta_Arch.py:1952: get_or_create_global_step (from tensorflow.contrib.framework.python.ops.variables) is deprecated and will be removed in a future version.
Instructions for updating:
Please switch to tf.train.get_or_create_global_step
INFO:tensorflow:Summary name /clone_loss is illegal; using clone_loss instead.
C:\Anaconda\envs\tensorflow\lib\site-packages\tensorflow\python\ops\gradients_impl.py:98: UserWarning: Converting sparse IndexedSlices to a dense Tensor of unknown shape. This may consume a large amount of memory.
  "Converting sparse IndexedSlices to a dense Tensor of unknown shape."
WARNING:tensorflow:From C:\Anaconda\envs\tensorflow\lib\site-packages\tensorflow\python\ops\clip_ops.py:113: calling reduce_sum (from tensorflow.python.ops.math_ops) with keep_dims is deprecated and will be removed in a future version.
Instructions for updating:
keep_dims is deprecated, use keepdims instead
WARNING:tensorflow:From C:\Anaconda\envs\tensorflow\lib\site-packages\tensorflow\contrib\slim\python\slim\learning.py:736: Supervisor.__init__ (from tensorflow.python.training.supervisor) is deprecated and will be removed in a future version.
Instructions for updating:
Please switch to tf.train.MonitoredTrainingSession
2018-04-09 10:49:14.772624 I C:\tf-jenkins\workspace\rel-win\Windows-gpu\PV36\tensorflow\core\platform\cpu_feature_guard.cc:148] Your CPU supports instructions that this TensorFlow binary was not compiled to use: AVX2
2018-04-09 10:49:15.874721 I C:\tf-jenkins\workspace\rel-win\Windows-gpu\PV36\tensorflow\core\common_runtime\gpu\gpu_device.cc:1121] Found device 0 with properties:
  name: Tesla K80 major: 3 minor: 7 memoryClockRate(GHz): 0.8235
  pciBusID: b554:00:00.0
TotalMemory: 11.8616 freeMemory: 11.06616
2018-04-09 10:49:15.874831 I C:\tf-jenkins\workspace\rel-win\Windows-gpu\PV36\tensorflow\core\common_runtime\gpu\gpu_device.cc:1311] Adding visible gpu devices: 0
2018-04-09 10:49:16.222829 I C:\tf-jenkins\workspace\rel-win\Windows-gpu\PV36\tensorflow\core\common_runtime\gpu\gpu_device.cc:993] Creating TensorFlow device (/job:localhost/replica:0/task:0/device:GPU:0 with 18668 MB memory) -> physical GPU (device: 0, name: Tesla K80, pci bus id: b554:00:00.0, compute capability: 3.7)
INFO:tensorflow:Restoring parameters from C:\tensorflow\models\research\object_detection\Faster_Rcnn_Inception_V2_coco_2018_01_28\model.ckpt
INFO:tensorflow:Done running local_init_op.
INFO:tensorflow:Starting Session.
INFO:tensorflow:Saving checkpoint to path training/model.ckpt
INFO:tensorflow:Starting Queues.
INFO:tensorflow:global_step/sec: 0
INFO:tensorflow:Recording summary at step 0
INFO:tensorflow:global step 1: loss = 3.5963 (7.268 sec/step)
INFO:tensorflow:global step 2: loss = 3.1998 (6.978 sec/step)
INFO:tensorflow:global step 3: loss = 2.8359 (6.261 sec/step)
INFO:tensorflow:global step 4: loss = 2.9347 (3.894 sec/step)
INFO:tensorflow:global step 5: loss = 2.8547 (0.371 sec/step)
INFO:tensorflow:global step 6: loss = 2.5921 (0.318 sec/step)
INFO:tensorflow:global step 7: loss = 2.4935 (0.389 sec/step)
INFO:tensorflow:global step 8: loss = 2.8825 (1.380 sec/step)
INFO:tensorflow:global step 9: loss = 1.9838 (0.303 sec/step)
INFO:tensorflow:global step 10: loss = 2.0555 (0.294 sec/step)
INFO:tensorflow:global step 11: loss = 1.4747 (0.339 sec/step)
INFO:tensorflow:global step 12: loss = 1.4656 (0.386 sec/step)
INFO:tensorflow:global step 13: loss = 1.3814 (0.306 sec/step)
INFO:tensorflow:global step 14: loss = 1.4787 (0.309 sec/step)
INFO:tensorflow:global step 15: loss = 0.9763 (0.385 sec/step)
INFO:tensorflow:global step 16: loss = 0.9382 (0.307 sec/step)
```

## Image training with Faster RCNN.

Each step of training reports the loss. It will start set up our graphic card (Tesla K80) then get the parameters of our model.config. This step start with high los and get lower and lower as training progresses. For my training on the Faster-RCNN-Inception-V2 model, it started at about 3.0 and quickly dropped below 0.8. I recommend allowing your model to train until the loss consistently drops below 0.05, which will take about 40,000 steps, or about 2-3 hours depending on how powerful your CPU and GPU are.

*\*Note: The loss numbers will be different if a different model is used. MobileNet-SSD starts with a loss of about 40, and should be trained until the loss is consistently under 2.*

```
(tensorflow) C:\tensorflow\models\research\object_detection\python\train.py --logtostderr --train_dir=training/ --pipeline_config_path=training/ssd_mobilenet_v1_pets.config
Instructions for updating:
Please switch to tf.train.create_global_step
INFO:tensorflow:depth of additional conv before box predictor: 0
INFO:tensorflow:depth of additional conv before box predictor: 0
INFO:tensorflow:depth of additional conv before box predictor: 0
INFO:tensorflow:depth of additional conv before box predictor: 0
INFO:tensorflow:Summary name /clone_loss is illegal; using clone_loss instead.
WARNING:tensorflow:From C:\Anaconda\envs\tensorflow\lib\site-packages\tensorflow\contrib\slim\python\slim\learning.py:736: Supervisor.__init__ (from tensorflow.python.training.supervisor) is deprecated and will be removed in a future version.
Instructions for updating:
Please switch to tf.train.MonitoredTrainingSession
2018-04-09 11:00:14.999684 I C:\tf-jenkins\workspace\rel-win\Windows-gpu\PV36\tensorflow\core\platform\cpu_feature_guard.cc:148] Your CPU supports instructions that this TensorFlow binary was not compiled to use: AVX2
2018-04-09 11:00:15.549727 I C:\tf-jenkins\workspace\rel-win\Windows-gpu\PV36\tensorflow\core\common_runtime\gpu\gpu_device.cc:1121] Found device 0 with properties:
  name: Tesla K80 major: 3 minor: 7 memoryClockRate(GHz): 0.8235
  pciBusID: b554:00:00.0
TotalMemory: 11.8616 freeMemory: 11.06616
2018-04-09 11:00:15.549831 I C:\tf-jenkins\workspace\rel-win\Windows-gpu\PV36\tensorflow\core\common_runtime\gpu\gpu_device.cc:1311] Adding visible gpu devices: 0
2018-04-09 11:00:15.871578 I C:\tf-jenkins\workspace\rel-win\Windows-gpu\PV36\tensorflow\core\common_runtime\gpu\gpu_device.cc:993] Creating TensorFlow device (/job:localhost/replica:0/task:0/device:GPU:0 with 18667 MB memory) -> physical GPU (device: 0, name: Tesla K80, pci bus id: b554:00:00.0, compute capability: 3.7)
INFO:tensorflow:Restoring parameters from C:\tensorflow\models\research\object_detection/ssd_mobilenet_v1_coco_2017_11_17/model.ckpt
INFO:tensorflow:Running local_init_op.
INFO:tensorflow:Done running local_init_op.
INFO:tensorflow:Starting Session.
INFO:tensorflow:Saving checkpoint to path training/model.ckpt
INFO:tensorflow:Starting Queues.
INFO:tensorflow:global_step/sec: 0
INFO:tensorflow:Recording summary at step 0
INFO:tensorflow:global step 1: loss = 51.5241 (17.285 sec/step)
INFO:tensorflow:global step 2: loss = 49.8464 (4.268 sec/step)
INFO:tensorflow:global step 3: loss = 45.5994 (2.998 sec/step)
INFO:tensorflow:global step 4: loss = 41.4389 (2.838 sec/step)
INFO:tensorflow:global step 5: loss = 39.0524 (2.744 sec/step)
INFO:tensorflow:global step 6: loss = 36.3848 (2.738 sec/step)
INFO:tensorflow:global step 7: loss = 32.7891 (2.842 sec/step)
INFO:tensorflow:global step 8: loss = 30.6353 (2.828 sec/step)
INFO:tensorflow:global step 9: loss = 27.4123 (2.754 sec/step)
INFO:tensorflow:global step 10: loss = 24.8867 (2.816 sec/step)
INFO:tensorflow:global step 11: loss = 22.1632 (2.848 sec/step)
INFO:tensorflow:global step 12: loss = 19.8167 (2.774 sec/step)
INFO:tensorflow:global step 13: loss = 18.3729 (2.789 sec/step)
INFO:tensorflow:global step 14: loss = 16.8964 (2.763 sec/step)
INFO:tensorflow:global step 15: loss = 14.9764 (2.735 sec/step)
INFO:tensorflow:global step 16: loss = 13.5772 (2.654 sec/step)
INFO:tensorflow:global step 17: loss = 13.0722 (2.615 sec/step)
```

### *Image Training SSD-Mobilenet*

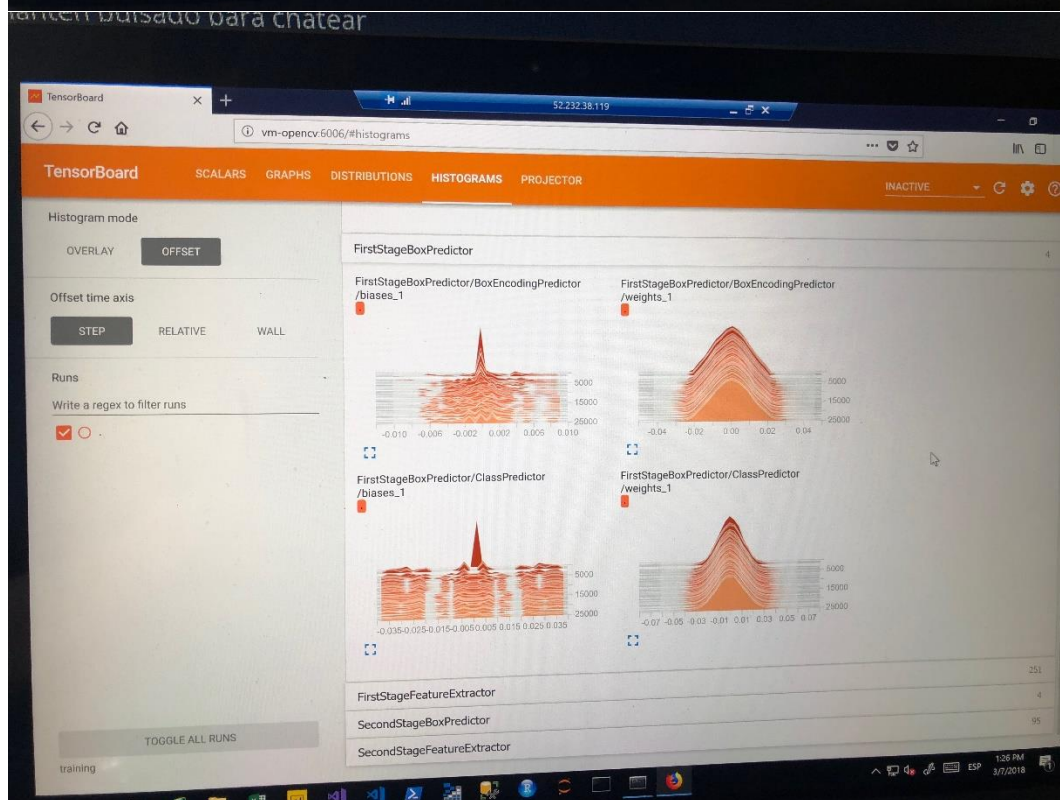
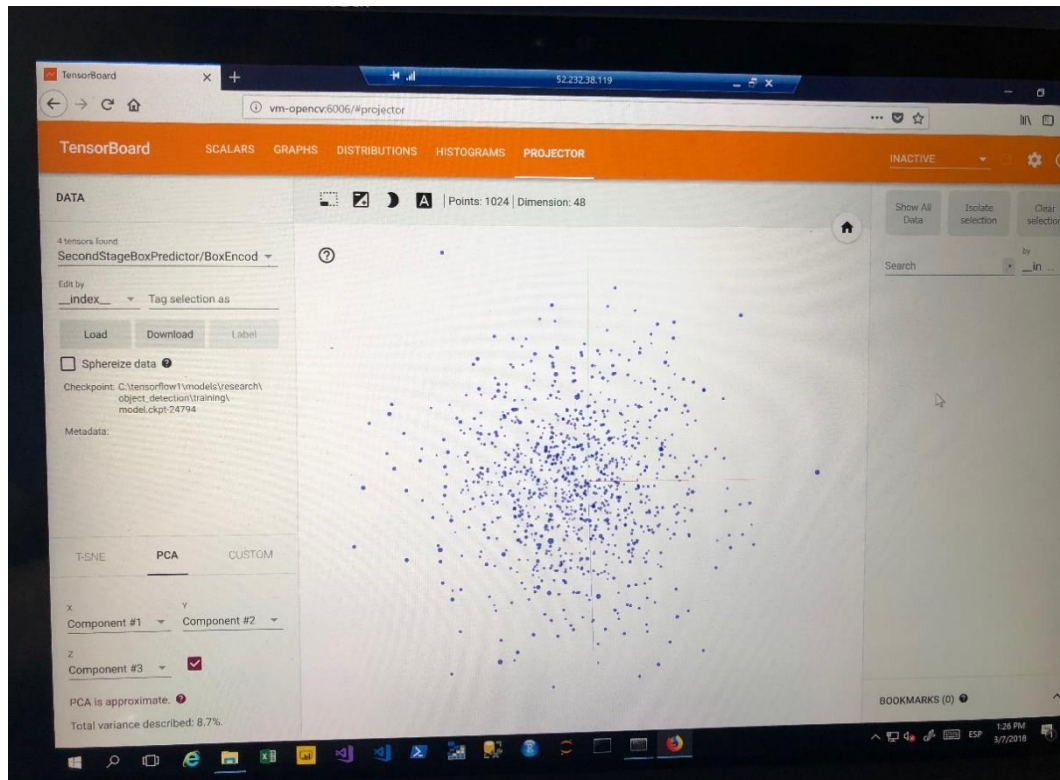
You can view that FasterRCNN training loss is more faster than SSD-Mobilenet. Also you can view progress of the training job by using TensorBoard. Thanks to tensorboard I have been able to explain in my final project through graphs like training has been from beginning to end, demonstrating also differences between both models.

[https://www.tensorflow.org/programmers\\_guide/summaries\\_and\\_tensorboard](https://www.tensorflow.org/programmers_guide/summaries_and_tensorboard)

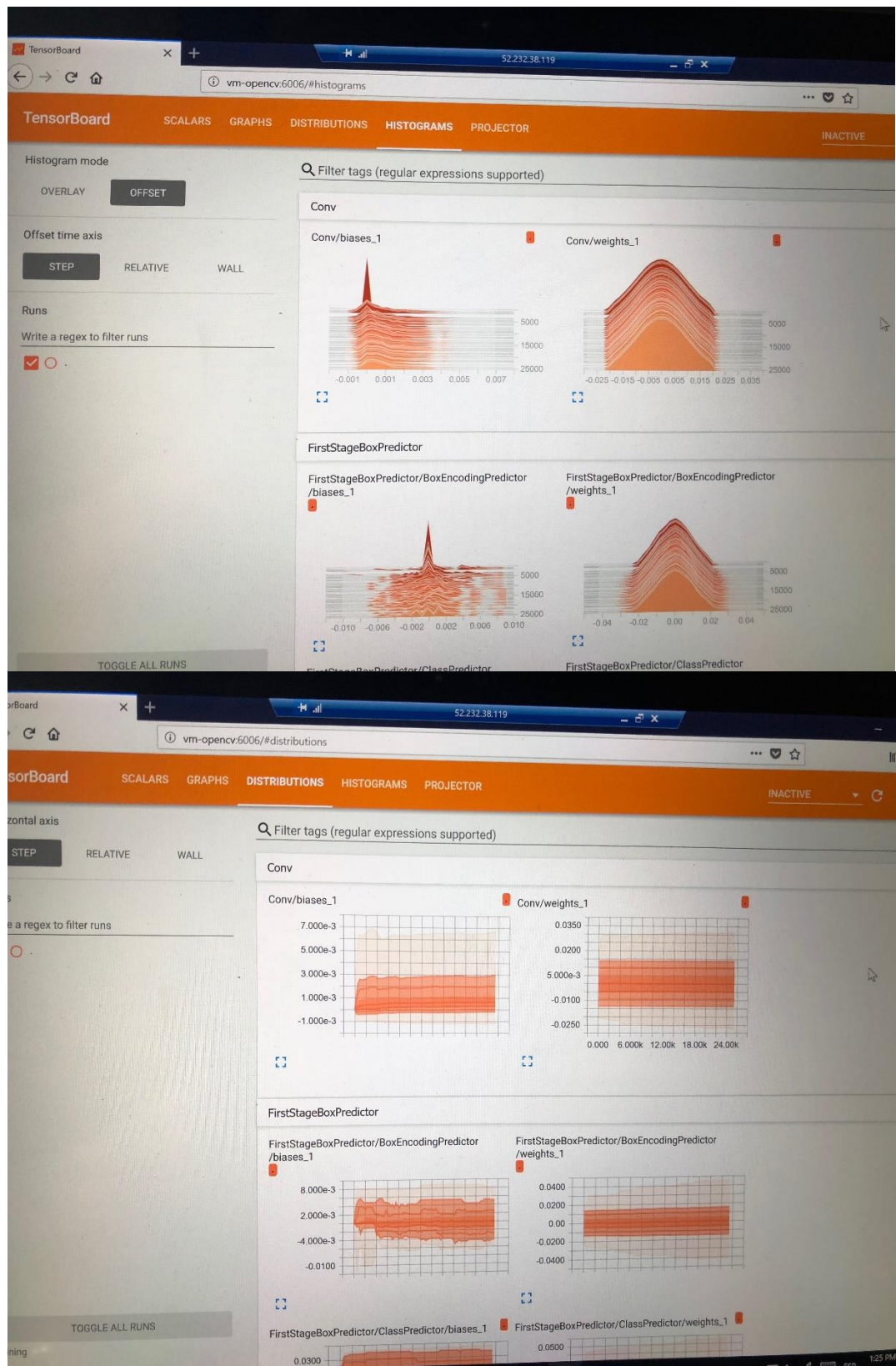
To do this, open a new instance of Anaconda Prompt, activate the tensorflow virtual environment, change to the C:\tensorflow1\models\research\object\_detection directory, and issue the following command:

```
(tensorflow) C:\tensorflow1\models\research\object_detection>tensorboard --logdir=training
```

This will create a webpage on your local machine at PCNAME:6006, which can be viewed through a web browser. The TensorBoard page provides information and graphs that show how the training is progressing. One important graph is the Loss graph, which shows the overall loss of the classifier over time.







The training routine periodically saves checkpoints about every five minutes. You can terminate the training by pressing Ctrl+C while in the command prompt window. I typically wait until just after a checkpoint has been saved to terminate the training. You can terminate training and start it later, and it will restart from the last saved



checkpoint. The checkpoint at the highest number of steps will be used to generate the frozen inference graph.

## **7. Create and exporting the frozen inference graph**

Now that training is complete, the last step is to generate the frozen inference graph (.pb file). From the \object\_detection folder, issue the following command, where "XXXX" in "model.ckpt-XXXX" should be replaced with the highest-numbered .ckpt file in the training folder:

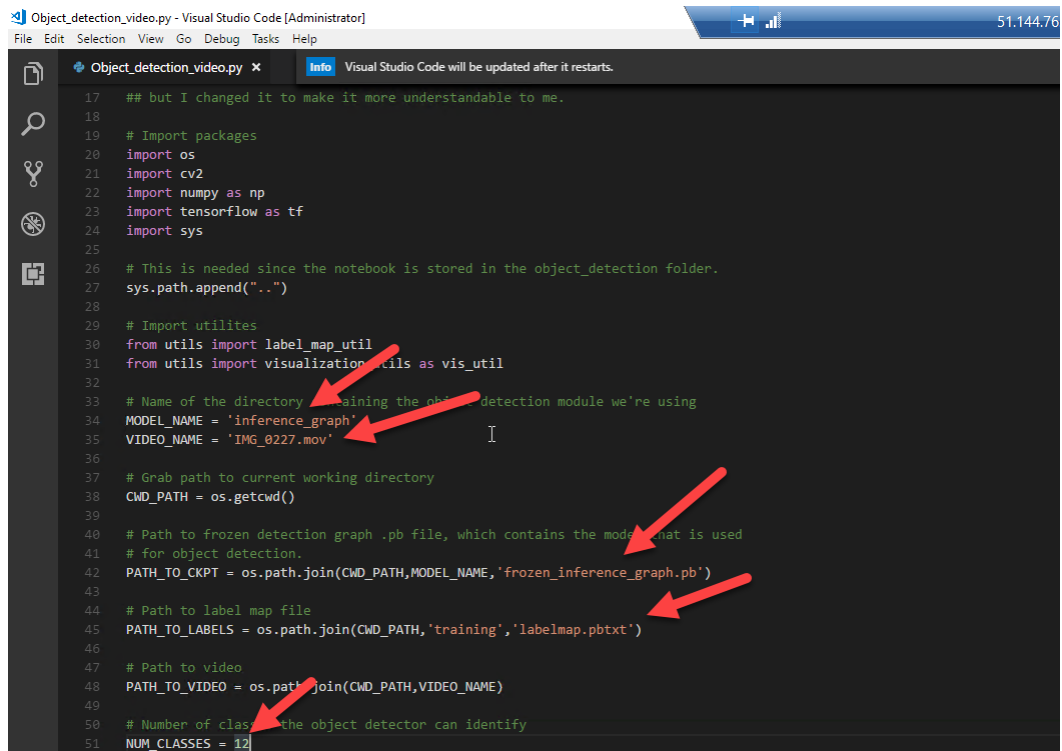
```
python export_inference_graph.py --input_type image_tensor --pipeline_config_path
training/faster_rcnn_inception_v2_pets.config --trained_checkpoint_prefix training/model.ckpt-
XXXX --output_directory inference_graph
```

This creates a frozen\_inference\_graph.pb file in the \object\_detection\inference\_graph folder. The .pb file contains the object detection classifier.

## **8. Testing and using your newly trained object detection classifier**

Now that we have our classifier ready we can put it to the test. I enclosed them in the github repository Python scripts developed by Evan Juras to test it on an image, video or source of the webcam.

Before executing the Python scripts, we will have to change several code variables, then those changes are shown in the image:



```
17 ## but I changed it to make it more understandable to me.
18
19 # Import packages
20 import os
21 import cv2
22 import numpy as np
23 import tensorflow as tf
24 import sys
25
26 # This is needed since the notebook is stored in the object_detection folder.
27 sys.path.append("../")
28
29 # Import utilites
30 from utils import label_map_util
31 from utils import visualization_utils as vis_util
32
33 # Name of the directory containing the object detection module we're using
34 MODEL_NAME = 'inference_graph'
35 VIDEO_NAME = 'IMG_0227.mov'
36
37 # Grab path to current working directory
38 CWD_PATH = os.getcwd()
39
40 # Path to frozen detection graph .pb file, which contains the model that is used
41 # for object detection.
42 PATH_TO_CKPT = os.path.join(CWD_PATH, MODEL_NAME, 'frozen_inference_graph.pb')
43
44 # Path to label map file
45 PATH_TO_LABELS = os.path.join(CWD_PATH, 'training', 'labelmap.pbtxt')
46
47 # Path to video
48 PATH_TO_VIDEO = os.path.join(CWD_PATH, VIDEO_NAME)
49
50 # Number of classes the object detector can identify
51 NUM_CLASSES = 12
```

Image Example of changes in Object\_Detection\_video.py

## 9.Common errors throughout the Project

1. Google may add more .proto files to the object\_detection/protos folder, so it may be necessary to add more files to the "protoc" command at 13:13. You can do this by adding ".\object\_detection\protos\FILENAME.proto" to the end of the long command string for each new file.
2. When running the "python train.py" command, if you get an error that says "TypeError: \_\_init\_\_() got an unexpected keyword argument 'dct\_method'.", then remove the "dct\_method=dct\_method" argument from line 110 of the object\_detection/data\_decoders/tf\_example\_decoder.py file.
3. When running "python train.py", if you get an error saying "google.protobuf.text\_format.ParseError: 110:25 : Expected string but found: """, try re-typing the quotation marks around each of the filepaths. If you copied the filepaths over from my GitHub tutorial, the quotation marks sometimes copy over as a different character type, and TensorFlow doesn't like that.
4. For train.py, if you get an error saying "TypeError: Expected int32, got range(0, 3) of type 'range' instead.", it is likely an issue with the learning\_schedules.py file. In the \object\_detection\utils\learning\_schedules.py file, change line 153

5. from `tf.constant(range(num_boundaries), dtype=tf.int32),` to  
`tf.constant(list(range(num_boundaries)), dtype=tf.int32),`.

## **10.Next post!**

The third post will explain another way of recognizing and classifying images (20 artworks) using scikit learn and python without having to use models of TensorFlow, CNTK or other technologies which offer models of convolved neural networks. Moreover, we will explain how to set up your own web app with python. For this part a fairly simple API which will collect information about the captured image of our mobile application in Xamarin will be needed, so it will inference with our model made in python, and it will return the corresponding prediction. With that methodology we can get easy classification without heavy hardware like TensorFlow or CNTK.

*\*Note: Also explain how can I prove my frozen inference model in Android!*