So, with this code… I ran into many errors so if you encounter one, odds are I have seen it and can help you with it. I was not able to get this to run with TensorFlow 2.0. The source code I used was written for 1.0. I have tried many times to use 2.0 but it does not work. If you are going to use TensorFlow-GPU then you will have to use TensorFlow 1.15, CUDA 10.0 and CUDNN 7.4. This is the only combination that works because it needs to be CUDA 10+ and the only TF that supports that is TF 1.15 and 2+. We can try to make it compatible with TF 2.0(TF 2.0 stops working in step 7) but for right now, this was the best I could do.

I have spent a lot of time with this and can explain how I developed this. It is loosely based off of [this tutorial](https://tensorflow-object-detection-api-tutorial.readthedocs.io/en/latest/). The file hierarchy goes as such:

**additional**: used for installations and other programs. In here:

**Anaconda**: This is a nice program that allows you to create multiple environments where you can install packages. I currently have four that I can switch between: TF 2.0 CPU, TF 2.0 GPU, TF 1.15 GPU, TF 1.15 CPU.

**CUDA 10.0**: The drivers needed to run TensorFlow GPU

**CUDNN 7.42**: The additional files needed for TensorFlow GPU

**LabelImg**: The program that allows you to create bounding boxes on images and generates a corresponding .xml file

**data:** used to hold the label\_map, .record files, and csv files

**images:** contains all the images that are used to train the model

**Test**: A random ~10% of all images and their corresponding xml files

**Training**: The other ~90% of images and their corresponding xml files

**models: The core of this code is based off the** [official TensorFlow models](https://github.com/tensorflow/models). The code in here is directly from this repository. Although the program does not use most of this code, I left it alone because it is not worth trying to trim the fat and risk breaking the program. But if we need to we can at the end of the semester

**scripts**: placed all scripts needed to run the application

**trained\_model**: this is where trained inference graph will be placed

**test\_images**: This is where user places test images not used to train/test the model.

**training**: The training is based off the [TensorFlow detection model zoo](https://github.com/tensorflow/models/blob/master/research/object_detection/g3doc/detection_model_zoo.md).

**ssd\_mobilenet\_v2\_coco**: This model has been trained on the [COCO](http://cocodataset.org/#home) dataset (this was the same used for our demo). We can test with other models. I just chose this one because it was recommended in many tutorials, and I just needed to see if one would work.

**pipeline**: This is the config file from ssd\_mobilenet\_v2\_coco. I made modifications to fit our classifiers

**How to use**

**Step 1**: Place the ‘Tensorflow’ folder in the root of your C:/ and [install CUDA](https://developer.nvidia.com/cuda-10.0-download-archive). Then place the C:\Tensorflow\additional\cuda folder into C:\Program Files\NVIDIA GPU Computing Toolkit\ CUDA\v10.0\

**Step 2**: Create the environment variables. Make sure the code below works, if it does not then you will have to manually enter in the variables

SET PATH=C:\Program Files\NVIDIA GPU Computing Toolkit\CUDA\v10.0\bin;%PATH%

SET PATH=C:\Program Files\NVIDIA GPU Computing Toolkit\CUDA\v10.0\libnvvp;%PATH%

SET PATH=C:\Program Files\NVIDIA GPU Computing Toolkit\CUDA\ v10.0\extras\CUPTI\libx64;%PATH%

SET PATH=C:\Program Files\NVIDIA GPU Computing Toolkit\ CUDA\v10.0\cuda\bin;%PATH%

SET PATH=C:\Tensorflow\models;%PATH%

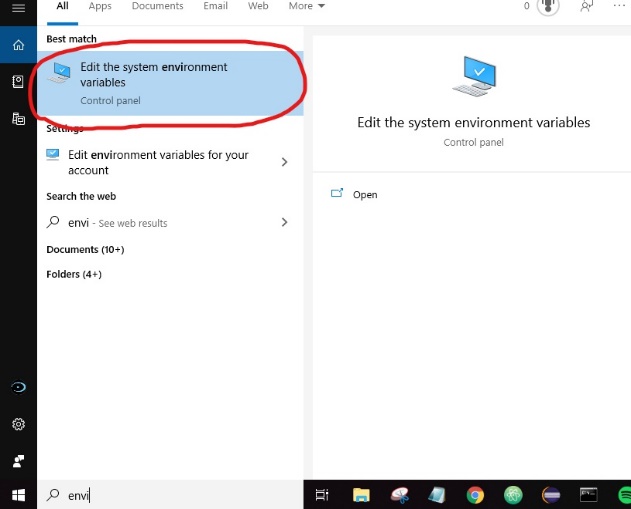
SET PATH=C:\Tensorflow\models\research;%PATH%

SET PATH=C:\Tensorflow\models\research\slim;%PATH%

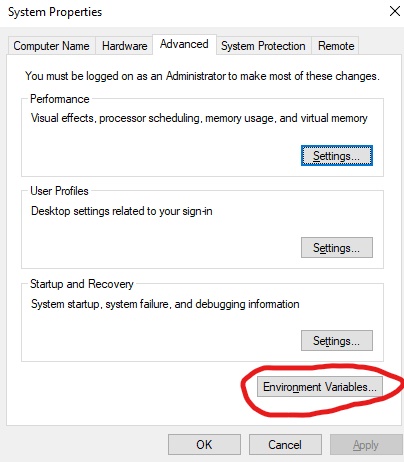
SET PATH=C:\Tensorflow\models\research\object\_detection;%PATH%

To check:

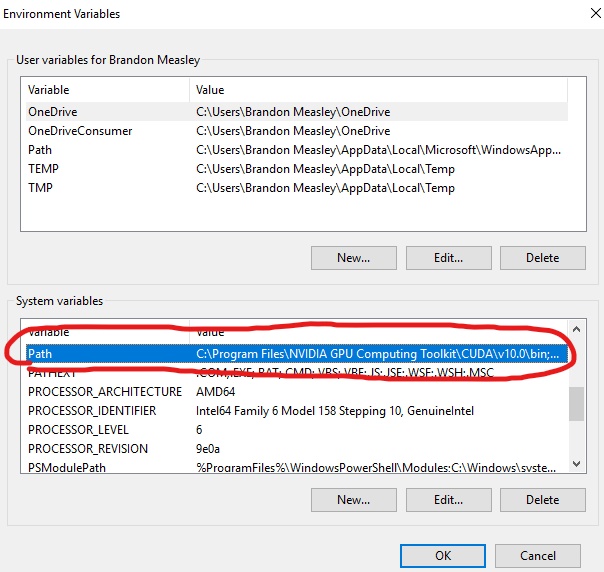
**Step 1**: click on start and type “environment variables”



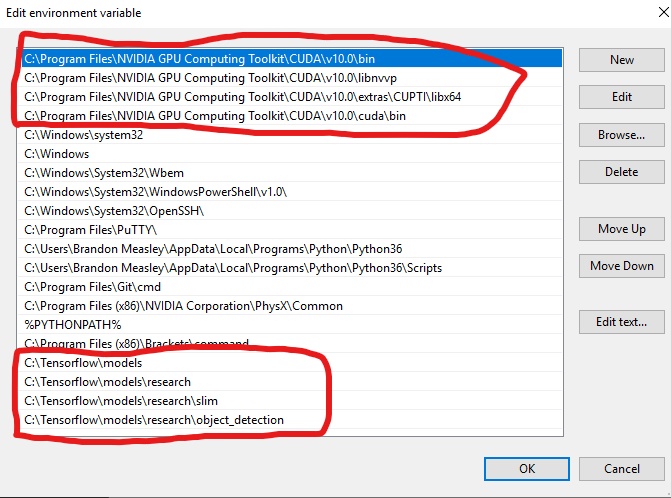
**Step 2**: click on “environment variables”



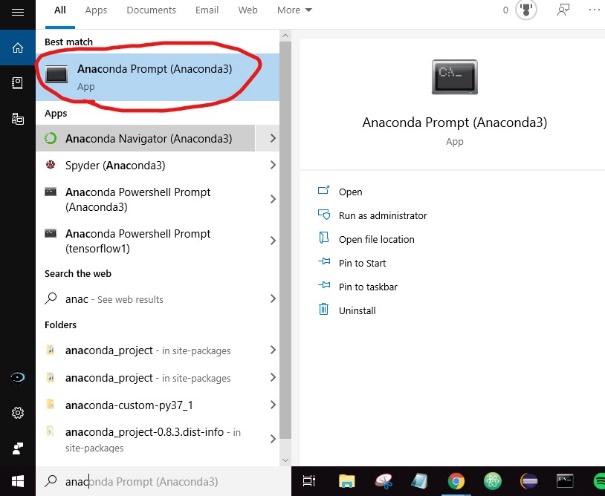
**Step 3**: double click on path in the “system variables” section



**Step 4**: Check to see if the following variables are there. If not enter them manually by clicking “new”. Make sure you select “ok” when done or the variables will not save. Close the window and reopen to ensure the changes were made. Note: the order does not matter



**Step 3**: Download Anaconda (double click on the installer found in C:\Tensorflow\additional). Run the anaconda prompt (see image below). All code will be done in here. Create and activate your environment. The environment you are currently in is in parenthesis next to the current directory. Every time you open anaconda the environment will be in the base environment. Make sure you activate your environment by running the command. Note in our case environment name will be ‘tf’, but you can name it whatever you like. To activate: conda activate <environment name>



**Step 4**: Install the proper packages: just copy all of the following commands at once, and paste them in the anaconda command window

conda create -n tf pip python=3.6

conda activate tf

pip install tensorflow-gpu==2.0

conda install pillow

conda install lxml

conda install jupyter

conda install matplotlib

conda install openc

conda install opencv

conda install cython

conda install pandas

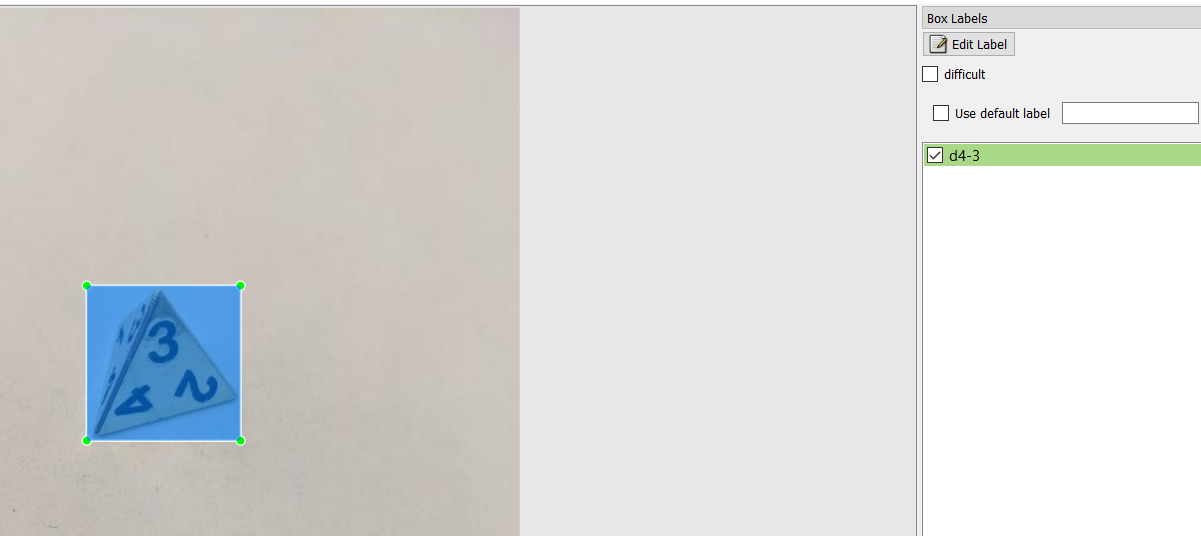
pip install git+https://github.com/philferriere/cocoapi.git#subdirectory=PythonAPI

**Step 5**: Use the labeling program to label the classifiers with your image. Use the [tutorial](https://github.com/tzutalin/labelImg) to learn the program. It is very useful to learn the hot keys because it will save a lot of time. I already labeled and added the files for a d4 dice

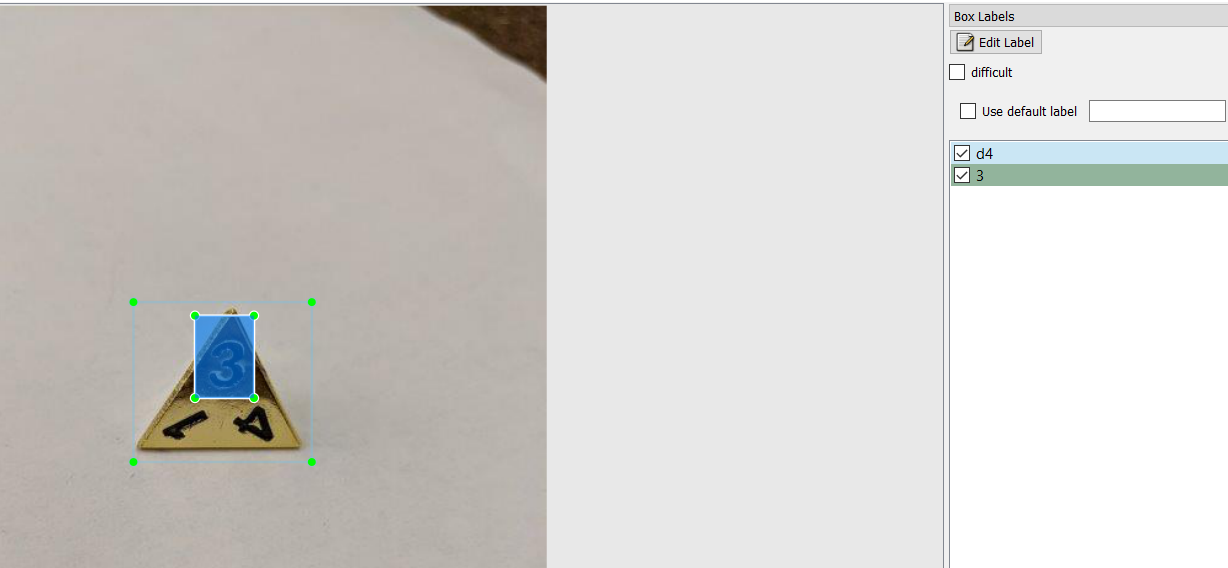
(Important note: when you create bounding boxes and hit save, it saves an xml file that hard codes the name of the image to it. Make sure before you begin, the images have the names you want, because if you change the image name it does not matter. The name of the xml files do not have to match the image, but it is best if they do for organizational purposes.

cd C:\Tensorflow\additional\labelImg & python labelImg.py

II have tried two different versions of labelling. The first version worked well. The second version could not detect a d4 but was not able to detect the value, so I found that the first version was better.



Version 1 of labelling



Version 2 of labelling

**Step 6**: Convert the xml files you made to the csv files and make the .record files. These files will be stored in C:\Tensorflow\data

cd C:\Tensorflow\scripts & python xml\_to\_csv.py

cd C:\Tensorflow\scripts & python generate\_tfrecord.py --label=../data/label\_map.pbtxt --csv\_input=../data/train\_labels.csv --img\_path=../images/train --output\_path=../data/train.record

cd C:\Tensorflow\scripts & python generate\_tfrecord.py --label=../data/label\_map.pbtxt --csv\_input=../data/test\_labels.csv --img\_path=../images/test --output\_path=../data/test.record

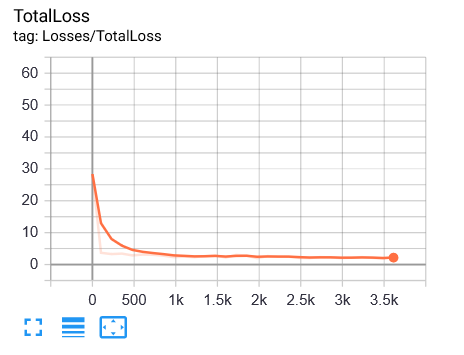
**Step 7**: Train the model. Note: I attempted to get rid of the deprecation messages, but it only removed some of them, and removed the step count and loss percentage. Make sure the only thing in the training directory is the ssd\_mobilenet\_v2\_coco and pipeline. Additionally, make sure the pipeline file is not empty. Before running the file keep a copy of the pipeline file somewhere on your machine. When this code is ran, it often erases the contents in pipline.config

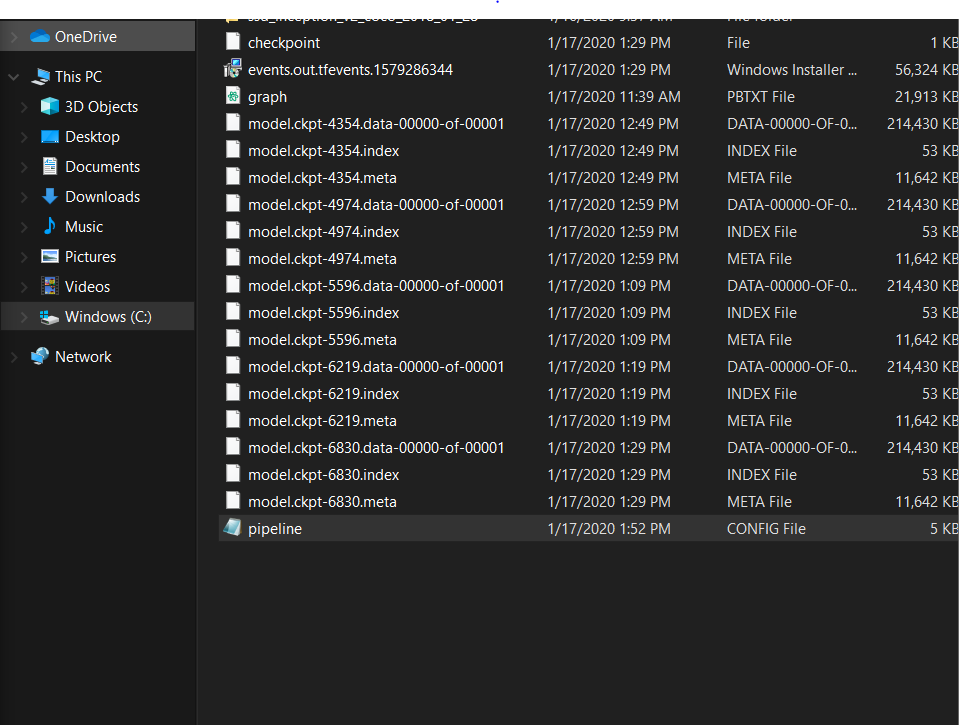
pip install tensorflow-gpu==1.15

cd C:\Tensorflow\scripts & python train.py --logtostderr --train\_dir=../training/ --pipeline\_config\_path=../training/pipeline.config

When I tested the program, it showed many deprecation messages. After a while it started training. It will display the loss percentage every 1 steps and saves a checkpoint in C:\tesnroflow\training every ~550 steps. This program will run for 200,00 steps (roughly 80 hours on my machine) so let it run as long as possible, but you will have to end it early. Most sources say to wait till the average loss is less than one. To see more real time statistics while the model is being trained, open a new anaconda prompt window, activate your environment, and open Tensorboard. Copy the url and paste it in the browser.

tensorboard –-logdir=C:\Tensorflow\training



**Step 8**: Export the inference graph from your most recent checkpoint. Copy the entirty of the following command. Make sure the C:\Tensorflow\trained\_model only contains one item, the test\_images folder. In the training folder the train.py placed all the model checkpoints. It saves the five most recent checkpoints. In the command use the most recent checkpoint, in the example below it would be model.ckpt-6830.mete.config. Make sure that there is an index and data file or it will not work

cd C:\Tensorflow\scripts & python export\_inference\_graph.py \ --input\_type image\_tensor \ --pipeline\_config\_path ../training/pipeline.config \ --trained\_checkpoint\_prefix ../training/model.ckpt-<highest value> \ --output\_directory ../trained\_model/

**Step 9**: Test. The following command will execute a test using the inference graph and the images in C:\Tensorflow\trained\_model\test\_images. The command will open each image. Note: when ran, the first image in the directory will appear. Press any key to get to the next image, once all pictures have been scrolled through the command will end. If you close the window before scrolling though all images, it will be stuck in a loop and you will have to exit the prompt.

cd C:\Tensorflow\scripts & python test.py

**Thoughts**

Obviously, there are many things we can improve on. I believe we can write scripts so that you run a script that does everything. Downloads anaconda, CUDA, places CUDNN, all packages, puts the TensorFlow folder in root, sets the environment variables and then opens LabelImg in the image directory. After they are done labeling we can write another script that converts the xml files, .record files, runs the training, and exports the inference graph. This way all the user must do is run a file, create the labels, and run another file. We can also improve the deprecation messages and test other models to see which is the fastest and most accurate. We can also focus on getting this to TF 2.0 and then finally converting it to a Core ML model. Let me know what you think. If you like this, one of us can post this to the github that way we can have version control. I don’t know the legality of placing cuda, cunna and anaconda in our github so I am just sending it as a .zip

**Concerns**

Labeling the data: for the test I just trained the d4 images. I had to redo the 3 value. Because of this it has different lighting, background and a lot more data. Therefore the training is not very accurate, so I have to redo and relabel a lot of the data

**Additional Information**

If you are adding/modifying classes, make sure you make the changes here:

* C:\Tensorflow\scripts\generate\_tfrecord.py - lines 37-47 - change/add
* C:\Tensorflow\training\pipeline.config - line 3 - change num\_classes
* C:\Tensorflow\scripts\test.py - line 16 - change num\_classes
* C:\Tensorflow\data\label\_map.pbtxt – entire file