Goal

To detect if an image/video stream has a vehicle in it or not using Tensorflow Object Detection API

Libraries Used

Target Software versions	
OS	Windows 10, Ubuntu16.04
Python	3.6
TensorFlow	1.6
CUDA Toolkit	v9.0 (For GPU version of Tensorflow)
CuDNN	v7.0.5 (For GPU version of Tensorflow)
Anaconda	Python 3.6

Environment setup

1. Refer this link for end-to-end installation https://tensorflow-object-detection-api-tutorial.readthedocs.io/en/latest/install.html or refer the instructions below.

Installation

General Remarks

- There are two different variations of TensorFlow that you might wish to install, depending on whether you would like TensorFlow to run on your CPU or GPU, namely TensorFlow CPU and TensorFlow GPU. I will proceed to document both and you can choose which one you wish to install.
- If you wish to install both TensorFlow variants on your machine, ideally you should install each variant under a different (virtual) environment. If you attempt to install both TensorFlow CPU and TensorFlow GPU, without making use of virtual environments, you will either end up failing, or when we later start running code there will always be an uncertainty as to which variant is being used to execute your code.
- To ensure that we have no package conflicts and/or that we can install several different versions/variants of TensorFlow (e.g. CPU and GPU), it is generally recommended to use a virtual environment of some sort. For the purposes of this tutorial we will be creating and managing our virtual environments using Anaconda, but you are welcome to use the virtual environment manager of your choice (e.g. virtualenv).

Install Anaconda Python 3.6 (Optional)

Although having Anaconda is not a requirement in order to install and use TensorFlow, I suggest doing so, due to it's intuitive way of managing packages and setting up new virtual environments. Anaconda is a pretty useful tool, not only for working with TensorFlow, but in general for anyone working in Python, so if you haven't had a chance to work with it, now is a good chance.

- Go to https://www.anaconda.com/download/
- Download Anaconda Python 3.6 version
- If disk space is an issue for your machine, you could install the minified version of Anaconda (i.e. Miniconda).
- When prompted for a "Destination Folder" you can chose whichever you wish, but I generally tend to use `C:\Anaconda3`, to keep things simple. Putting Anaconda under `C:\Anaconda3` also ensures that you don't get the awkward `` `Destination Folder` contains spaces `` warning.

TensorFlow Installation

As mentioned in the Remarks section, there exist two generic variants of TensorFlow, which utilise different hardware on your computer to run their computationally heavy Machine Learning algorithms. The simplest to install, but also in most cases the slowest in terms of performance, is TensorFlow CPU, which runs directly on the CPU of your machine. Alternatively, if you own a (compatible) Nvidia graphics card, you can take advantage of the available CUDA cores to speed up the computations performed by TesnsorFlow, in which case you should follow the guidelines for installing TensorFlow GPU.

TensorFlow CPU

Getting setup with an installation of TensorFlow CPU can be done in 3 simple steps.

Create a new Conda virtual environment (Optional)

- Open a new *Anaconda/Command Prompt* window
- Type the following command:

```
conda create -n tensorflow_cpu pip python=3.6
```

- The above will create a new virtual environment with name `tensorflow_cpu`
- Now lets activate the newly created virtual environment by running the following in the *Anaconda Prompt* window:

```
activate tensorflow_cpu
```

Once you have activated your virtual environment, the name of the environment should be displayed within brackets at the beggining of your cmd path specifier, e.g.:

```
(tensorflow_cpu) C:\Users\sglvladi>
```

Install TensorFlow CPU for Python

- Open a new *Anaconda/Command Prompt* window and activate the *tensorflow_gpu* environment (if you have not done so already)
- Once open, type the following on the command line:

```
pip install --ignore-installed --upgrade tensorflow
```

- Wait for the installation to finish

Test your Installation

- Open a new *Anaconda/Command Prompt* window and activate the *tensorflow_cpu* environment (if you have not done so already)
- Start a new Python interpreter session by running:

```
python
```

- Once the interpreter opens up, type:

```
>>> import tensorflow as tf
```

- If the above code shows an error, then check to make sure you have activated the *tensorflow_gpu* environment and that tensorflow_gpu was successfully installed within it in the previous step.
- Then run the following:

```
>>> hello = tf.constant('Hello, TensorFlow!')
>>> sess = tf.Session()
```

- Once the above is run, if you see a print-out similar (but not identical) to the one below, it means that you could benefit from installing TensorFlow by building the sources that correspond to you specific CPU. Everything should still run as normal, just slower than if you had built TensorFlow from source.

```
2018-03-21 22:10:18.682767: I C:\tf_jenkins\workspace\rel-
win\M\windows\PY\36\tensorflow\core\platform\cpu_feature_guard.cc
:140] Your CPU supports instructions that this TensorFlow binary
was not compiled to use: AVX2
```

- Finally, for the sake of completing the test as described by TensorFlow themselves (see here), let’s run the following:

```
>>> print(sess.run(hello))
b'Hello, TensorFlow!'
```

TensorFlow GPU

The installation of *TesnorFlow GPU* is slightly more involved than that of *TensorFlow CPU*, mainly due to the need of installing the relevant Graphics and CUDE drivers. There's a nice Youtube tutorial (see here), explaining how to install TensorFlow GPU. Although it describes different versions of the relevant components (including TensorFlow itself), the installation steps are generally the same with this tutorial.

Before proceeding to install TesnsorFlow GPU, you need to make sure that your system can satisfy the following requirements:

Prerequisites
Nvidia GPU (GTX 650 or newer)
CUDA Toolkit v9.0
CuDNN v7.0.5
Anaconda with Python 3.6 (Optional)

Install CUDA Toolkit

Follow this link to download and install CUDA Toolkit v9.0.

Install CUDNN

- Go to https://developer.nvidia.com/rdp/cudnn-download
- Create a user profile if needed and log in
- Select cuDNN v7.0.5 (Feb 28, 2018), for CUDA 9.0
- Download cuDNN v7.0.5 Library for Windows 10
- Extract the contents of the zip file (i.e. the folder named `cuda`) inside `<INSTALL_PATH>\NVIDIA GPU Computing Toolkit\CUDA\v9.0\`, where `<INSTALL_PATH>` points to the installation directory specified during the installation of the CUDA Toolkit. By default `<INSTALL_PATH>` = `C:\Program Files`.

Set Your Environment Variables

- Go to *Start* and Search "environment variables"
- Click the Environment Variables button
- Click on the `Path` system variable and select edit
- Add the following paths

```
<INSTALL_PATH>\NVIDIA GPU Computing Toolkit\CUDA\v9.0\bin

<INSTALL_PATH>\NVIDIA GPU Computing Toolkit\CUDA\v9.0\libnvvp
```

```
<INSTALL_PATH>\NVIDIA GPU Computing Toolkit\CUDA\v9.0\extras\CUPTI\libx64

<INSTALL_PATH>\NVIDIA GPU Computing Toolkit\CUDA\v9.0\cuda\bin
```

Update your GPU drivers

- Go to http://www.nvidia.com/Download/index.aspx
- Select your GPU version to download
- Install the driver

Create a new Conda virtual environment

- Open a new *Anaconda/Command Prompt* window
- Type the following command:

```
conda create -n tensorflow_gpu pip python=3.6
```

- The above will create a new virtual environment with name `tensorflow_gpu`
- Now lets activate the newly created virtual environment by running the following in the *Anaconda Promt* window:

```
activate tensorflow_gpu
```

Once you have activated your virtual environment, the name of the environment should be displayed within brackets at the beggining of your cmd path specifier, e.g.:

```
(tensorflow_gpu) C:\Users\sglvladi>
```

Install TensorFlow GPU for Python

- Open a new *Anaconda/Command Prompt* window and activate the *tensorflow_gpu* environment (if you have not done so already)
- Once open, type the following on the command line:

```
pip install --ignore-installed --upgrade tensorflow-gpu
```

- Wait for the installation to finish

Test your Installation

- Open a new *Anaconda/Command Prompt* window and activate the *tensorflow_gpu* environment (if you have not done so already)
- Start a new Python interpreter session by running:

```
python
```

- Once the interpreter opens up, type:

```
>>> import tensorflow as tf
```

- If the above code shows an error, then check to make sure you have activated the *tensorflow_gpu* environment and that tensorflow_gpu was successfully installed within it in the previous step.
- Then run the following:

```
>>> hello = tf.constant('Hello, TensorFlow!')
>>> sess = tf.Session()
```

- Once the above is run, you should see a print-out similar (but not identical) to the one bellow:

```
2018-03-21 21:46:18.962971: I C:\tf_jenkins\workspace\rel-
win\M\windows-
gpu\PY\36\tensorflow\core\common_runtime\gpu\gpu_device.cc:1212]
Found device 0 with properties:
name: GeForce GTX 770 major: 3 minor: 0 memoryClockRate(GHz):
1.163
pciBusID: 0000:02:00.0
totalMemory: 2.00GiB freeMemory: 1.63GiB
2018-03-21 21:46:18.978254: I C:\tf_jenkins\workspace\rel-
win\M\windows-
gpu\PY\36\tensorflow\core\common_runtime\gpu\gpu_device.cc:1312]
Adding visible gpu devices: 0
2018-03-21 21:46:19.295152: I C:\tf_jenkins\workspace\rel-
win\M\windows-
gpu\PY\36\tensorflow\core\common_runtime\gpu\gpu_device.cc:993]
Creating TensorFlow device
(/job:localhost/replica:0/task:0/device:GPU:0 with 1414 MB
memory) -> physical GPU (device: 0, name: GeForce GTX 770, pci
bus id: 0000:02:00.0, compute capability: 3.0)
```

- Finally, for the sake of completing the test as described by TensorFlow themselves (see here), let's run the following:

```
>>> print(sess.run(hello))
b'Hello, TensorFlow!'
```

TensorFlow Models Installation

Now that you have installed TensorFlow, it is time to install the models used by TensorFlow to do its magic.

Install Prerequisites

Building on the assumption that you have just created your new virtual environment (whether that's *tensorflow_cpu*,`tensorflow_gpu` or whatever other name you might have used), there are some packages which need to be installed before installing the models.

Prerequisite packages	
Name	**Tutorial version-build**
pillow	5.0.0-py36h0738816_0
lxml	4.2.0-py36heafd4d3_0
jupyter	1.0.0-py36_4
matplotlib	2.2.2-py36h153e9ff_0
opencv	3.3.1-py36h20b85fd_1

The packages can be install by running:

```
conda install <package_name>(=<version>)
```

where `<package_name>` can be replaced with the name of the package, and optionally the package version can be specified by adding the optional specifier `=<version>` after `<package_name>`.

Alternatively, if you don't want to use Anaconda you can install the packages using `pip`:

```
pip install <package_name>(=<version>)
```

but you will need to install `opencv-python` instead of `opencv`.

Downloading the TensorFlow Models

- Create a new folder under a path of your choice and name it `TensorFlow`. (e.g. `C:\Users\sglvladi\Documents\TensorFlow`).
- From your *Anaconda/Command Prompt* `cd` into the `TensorFlow` directory.
- To download the models you can either use Git to clone the TensorFlow Models repo inside the `TensorFlow` folder, or you can simply download it as a ZIP and extract it's contents inside the `TensorFlow` folder. To keep things consistent, in the latter case you will have to rename the extracted folder `models-master` to `models`. [1]
- You should now have a single folder named `models` under your `TensorFlow` folder, which contains another 4 folders as such:

```
TensorFlow
└─ models
   ├── official
   ├── research
   ├── samples
   └── tutorials
```

[1] The latest repo commit when writing this tutorial is da903e0.

Adding necessary Environment Variables

Since a lot of the scripts we will use require packages from `Tensorflow\models\research\object_detection` to be run, I have found that it's convenient to add the specific folder to our environmental variables.

For Linux users, this can be done by either adding to `~/.bashrc` or running the following code:

```
export
PYTHONPATH=$PYTHONPATH:<PATH_TO_TF>/TensorFlow/models/research/object_detecti
on
```

For Windows users, the following folder must be added to your `Path` environment variable (See Set Your Environment Variables):

```
    <PATH_TO_TF>\TensorFlow\models\research\object_detection
```

For whatever reason, some of the TensorFlow packages that we will need to use to do object detection, do not come pre-installed with our tensorflow installation.

For Linux users ONLY, the Installation docs suggest that you either run, or add to `~/.bashrc` file, the following command, which adds these packages to your PYTHONPATH:

```
# From tensorflow/models/research/
export PYTHONPATH=$PYTHONPATH:`pwd`:`pwd`/slim
```

For Windows, the only way that I found works best, is to simply add the following folders to your `Path` environment variable (See also Set Your Environment Variables):

```
    <PATH_TO_TF>\TensorFlow\models\research\slim

    <PATH_TO_TF>\TensorFlow\models\research\slim\datasets

    <PATH_TO_TF>\TensorFlow\models\research\slim\deployment

    <PATH_TO_TF>\TensorFlow\models\research\slim\nets

    <PATH_TO_TF>\TensorFlow\models\research\slim\preprocessing

    <PATH_TO_TF>\TensorFlow\models\research\slim\scripts
```

where `<PATH_TO_TF>` replaces the absolute path to your `TesnorFlow` folder. (e.g. `<PATH_TO_TF>` = `C:\Users\sglvladi\Documents` if `TensorFlow` resides within your `Documents` folder)

Protobuf Installation/Compilation

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

This should be done as follows:

- Head to the protoc releases page
- Download the latest `*-win32.zip` release (e.g. `protoc-3.5.1-win32.zip`)
- Create a folder in `C:\Program Files` and name it `Google Protobuf`.
- Extract the contents of the downloaded `*-win32.zip`, inside `C:\Program Files\Google Protobuf`
- Add `C:\Program Files\Google Protobuf\bin` to your `Path` environment variable (see Set Your Environment Variables)
- In a new *Anaconda/Command Prompt* [2], `cd` into `TensorFlow/models/research/` directory and run the following command:

```
# From TensorFlow/models/research/
protoc object_detection/protos/*.proto --python_out=.
```

If you are on Windows and using version 3.5 or later, the wildcard will not work and you have to run this in the command prompt:

```
# From TensorFlow/models/research/
for /f %i in ('dir /b object_detection\protos\*.proto') do protoc
object_detection\protos\%i --python_out=.
```

[2] NOTE: You MUST open a new *Anaconda/Command Prompt* for the changes in the environment variables to take effect.

Test your Installation

- Open a new *Anaconda/Command Prompt* window and activate the *tensorflow_gpu* environment (if you have not done so already)
- `cd` into `TensorFlow\models\research\object_detection` and run the following command:

```
# From TensorFlow/models/research/object_detection
jupyter notebook
```

- This should start a new `jupyter notebook` server on your machine and you should be redirected to a new tab of your default browser.
- Once there, simply follow sentdex's Youtube video to ensure that everything is running smoothly.
- If, when you try to run `In [11]:`, Python crashes, have a look at the *Anaconda/Command Prompt* window you used to run the `jupyter notebook` service and check for a line similar (maybe identical) to the one below:

```
2018-03-22 03:07:54.623130: E C:\tf_jenkins\workspace\rel-
win\M\windows-
gpu\PY\36\tensorflow\stream_executor\cuda\cuda_dnn.cc:378] Loaded
runtime CuDNN library: 7101 (compatibility version 7100) but
source was compiled with 7003 (compatibility version 7000).  If
using a binary install, upgrade your CuDNN library to match.  If
building from sources, make sure the library loaded at runtime
matches a compatible version specified during compile
configuration.
```

- If the above line is present in the printed debugging, it means that you have not installed the correct version of the cuDNN libraries. In this case make sure you re-do the Install CUDNN step, making sure you instal cuDNN v7.0.5.

LabelImg Installation(Required if you want to create your own dataset)

Create a new Conda virtual environment

To deal with the fact that `labelImg` (on Windows) requires the use of `pyqt4`, while `tensorflow 1.6` (and possibly other packages) require `pyqt5`, we will create a new virtual environment in which to run `labelImg`.

- Open a new *Anaconda/Command Prompt* window
- Type the following command:

```
conda create -n labelImg pyqt=4
```

- The above will create a new virtual environment with name `labelImg`
- Now lets activate the newly created virtual environment by running the following in the *Anaconda Promt* window:

```
activate labelImg
```

Once you have activated your virtual environment, the name of the environment should be displayed within brackets at the beginning of your cmd path specifier, e.g.:

```
(labelImg) C:\Users\sglvladi>
```

Downloading labelImg

- Inside you `TensorFlow` folder, create a new directory, name it `addons` and then `cd` into it.
- To download the package you can either use Git to clone the labelImg repo inside the `TensorFlow\addons` folder, or you can simply download it as a ZIP and extract it's contents inside the `TensorFlow\addons` folder. To keep things consistent, in the latter case you will have to rename the extracted folder `labelImg-master` to `labelImg`. [3]

- You should now have a single folder named `addons\labelImg` under your `TensorFlow` folder, which contains another 4 folders as such:

```
TensorFlow
├─ addons
│  └── labelImg
└─ models
├── official
├── research
├── samples
└── tutorials
```

[3] The latest repo commit when writing this tutorial is 8d1bd68.

Installing dependencies and compiling package

- Open a new *Anaconda/Command Prompt* window and activate the *tensorflow_gpu* environment (if you have not done so already)
- `cd` into `TensorFlow\addons\labelImg` and run the following commands:

```
conda install pyqt=4
conda install lxml
pyrcc4 -py3 -o resources.py resources.qrc
```

Test your installation

- Open a new *Anaconda/Command Prompt* window and activate the *tensorflow_gpu* environment (if you have not done so already)
- `cd` into `TensorFlow\addons\labelImg` and run the following command:

```
python labelImg.py
# or
python  labelImg.py [IMAGE_PATH] [PRE-DEFINED CLASS FILE]
```

Important:

1. After all the above steps are completed, replace the file named object_detection_tutorial.ipynb at location <PATH_TO_TF>\TensorFlow\models\research\object_detection\
by the file named object_detection_tutorial_ctl.ipynb that you downloaded from github and add it to the same location.

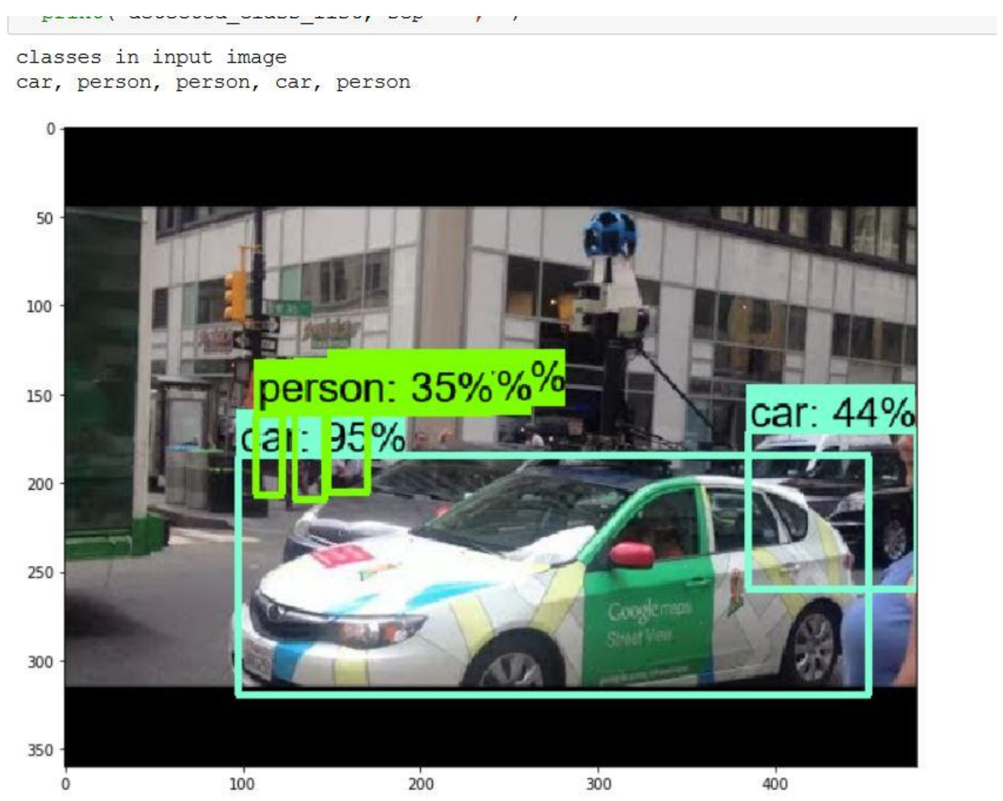
2. Replace file named visualization_utils.py under location <PATH_TO_TF>\TensorFlow\models\research\object_detection\utils by file named visualization_utils.py that you downloaded from github.

3. Download the file named object_detection_in_video_ctl.ipynb at the location <PATH_TO_TF>\TensorFlow\models\research\object_detection\

Running the script on images and expected output

1. To run the script on your images, put your test images under the folder <PATH_TO_TF>\TensorFlow\models\research\object_detection\test_images and name your images in the format images1.jpg, images2.jpg ….imagesn.jpg for n number of images.

2. Go to location <PATH_TO_TF>\TensorFlow\models\research\object_detection and run object_detection_in_video_ctl.ipynb file. If everything works fine you will get an output like



```
classes in input image
car, person, person, car, person
```

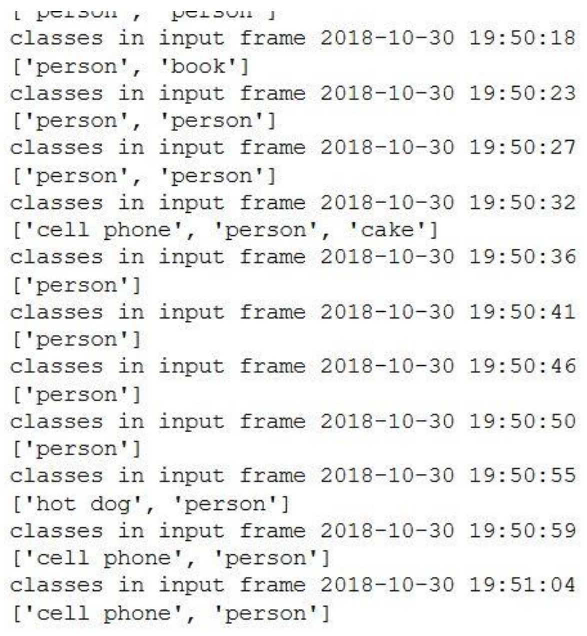
The output consists of bounding boxes of the object detected by the algorithm and will be drawn on image along with their class name and accuracy.

The script will also print all the classes identified in the image. These values are stored in list named detected_class_list.

Running the script on webcam stream and expected output

To detect objects using your webcam go to location
<PATH_TO_TF>\TensorFlow\models\research\object_detection and run the script
object_detection_in_video_ctl.ipynb
This will start your webcam and classes detected in each frame with their timestamp(epochs) will be logged in a file named output_video_file.txt under the same location.

The output on screen will look like this.



```
classes in input frame 2018-10-30 19:50:18
['person', 'book']
classes in input frame 2018-10-30 19:50:23
['person', 'person']
classes in input frame 2018-10-30 19:50:27
['person', 'person']
classes in input frame 2018-10-30 19:50:32
['cell phone', 'person', 'cake']
classes in input frame 2018-10-30 19:50:36
['person']
classes in input frame 2018-10-30 19:50:41
['person']
classes in input frame 2018-10-30 19:50:46
['person']
classes in input frame 2018-10-30 19:50:50
['person']
classes in input frame 2018-10-30 19:50:55
['hot dog', 'person']
classes in input frame 2018-10-30 19:50:59
['cell phone', 'person']
classes in input frame 2018-10-30 19:51:04
['cell phone', 'person']
```

Refer file below to see sample output.

output_video_file.txt

Help/Questions

Drop an email at sk3126@rit.edu