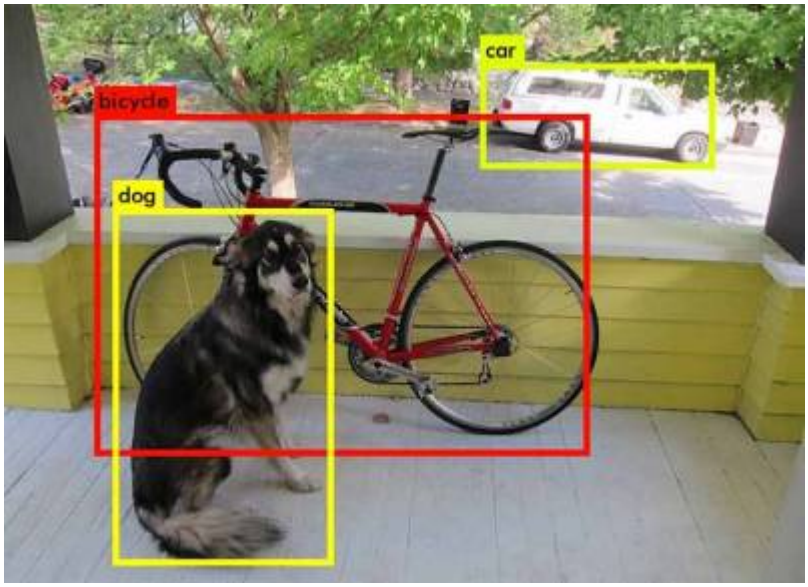
Object Detection

Introduction

An object detection algorithm is one that takes in an image, and outputs bounding boxes surrounding the objects of interest in the image. An example is shown below, a dog, bicycle, and a car.



Before you can use an object detection algorithm, you must first predefine what sort of objects (aka classes) you want to be able to detect. Next, you must train the algorithm on example images that contain the objects you want it to be able to learn. These images must be manually labeled by people, showing where the objects are in the image. In our experience, you probably need at least a few hundred example images for each class to get decent results.

After the algorithm has been trained on the example data, you can then use it to find objects in new images that it hasn't seen before! The process of using the object detection algorithm to find objects is also known as “inference”.

Tools

We currently use the Tensorflow object detection API (henceforth abbreviated as TFODA) for both training and inference. Previously we used the Darknet framework, however we found it rather difficult to use because the code is messy and uses rather old versions of libraries. The tensorflow object detection API is developed by more people, easier to use, and kept more up to date.

Setup

These instructions are for manual installation on your own computer, we really need to automate the installation of everything. Also, ideally we should be training on Kamiak, not our own computers.

Before you can start training, you first need to install tensorflow and the Tensorflow Object Detection API.

Tensorflow

Tensorflow can run either on your CPU, or on an Nvidia GPU (unfortunately AMD support isn't ready at this time). If you have an Nvidia GPU, it's highly recommended to use it, it is orders of magnitude faster. Instructions are adapted from here. To test your installation, open up python and run:

```
import tensorflow as tf
hello = tf.constant('Hello, TensorFlow!')
sess = tf.Session()
print(sess.run(hello))
```

CPU Installation

If you're running on CPU only, you simply need to run:J

```
pip install --user tensorflow
```

GPU Installation

If you're running on an Nvidia GPU, you'll need to install Nvidia's CUDA and CuDNN libraries. Currently, Tensorflow requires CUDA 9.0 and CuDNN 7.0, it's important you install the right versions. To install tensorflow for gpu, run:

```
pip install --user tensorflow-gpu
```

CUDA Installation

You can download CUDA 9.0 for Ubuntu 16.04 from here. To install, run:

```
sudo dpkg -i cuda-repo-ubuntu1604_9.0.176-1_amd64.deb
sudo apt-key adv --fetch-keys \
http://developer.download.nvidia.com/compute/cuda/repos/ubuntu1604/x86_64/7f
```

```
a2af80.pub
sudo aptitude update
sudo aptitude install cuda
```

CuDNN Installation

To download CuDNN you'll need to create an Nvidia account. Download (and create an account) here. You'll want to download CuDNN 7.0.x for CUDA 9.0, both the developer and runtime libraries. Install both of them.

Object Detection API

Instructions were adapted from here

Install dependencies:

```
sudo aptitude install protobuf-compiler
pip install --user Cython pillow lxml jupyter matplotlib
```

Download the Object Detection API from Github. I'm putting it in ~/.local, you can put it wherever you'd like, but you'll have to change all the instructions accordingly.

```
git clone git@github.com:tensorflow/models.git
~/.local/tensorflow_object_detection_api
```

Notice how long that took to clone? That's why we don't put binary files in our git repos!

Setup the API:

```
cd ~/.local/tensorflow_object_detection_api/research/
protoc object_detection/protos/*.proto --python_out=.
echo -e '# Tensorflow object detection api\nexport
PYTHONPATH=$PYTHONPATH:`pwd`:`pwd`/slim' >> ~/.bashrc
source ~/.bashrc
```

Test it:

```
python
~/.local/tensorflow_object_detection_api/research/object_detection/builders/
model_builder_test.py
```

Generating Training Data

To label our images, we use sloth. For more details, see (link). Sloth creates a json file that describes the labels for each image. The TFODA requires the example images and their labels to be packaged

into a specific file format called a TFrecord. We have a python script for converting the sloth json format in TFrecord files in our vision_dev repository. To create a TFrecord file, you need both the json file containing the labels, and the images the json file refers to. Usage is shown below:

```
./sloth_to_tf_record.py <input json file> <output directory>
```

sloth_to_tf_record.py uses the json file to find the image files, so if the script can't find images, look at the json file to see the path that it uses to locate each image. sloth_to_tfrecord.py actually generates several files which get stored in <output directory>, which must already exist (the script will not create the directory if it does not exist). The output files are:

- label_map.pbtext - defines all the image classes
- train.record - stores images and labels to be trained on
- test.record - stores images and labels for testing

label_map.pbtext is used internally by TFODA to map outputs to names, you shouldn't ever modify it yourself. You'll notice that the script actually creates two record files, train.record test.record. When doing machine learning, in general it is a good idea to separate your labeled data into a training set and a testing set. This is because of the fact that given enough time with the data, machine learning algorithms will specialize themselves in order to perfectly learn the training data at the expense of being able to generalize to images it hasn't seen before. We can take a subset on our labeled data (the test dataset), and periodically evaluate the algorithm's performance on it to figure out when to stop training (more details on this later). sloth_to_tf_record.py automatically allocates 10% of the dataset to put in the train.record file.

After the data is labeled we can start training!

Train Network

Instruction were adapted from here.

Workspace Setup

Setup your workspace with the following directory structure:

```
object_detection_workspace/
├── data
├── models
└── output
    ├── eval
    └── train
```

data/ will hold your training data (label_map.pbtext, train.record, and test.record). output/ and its subdirectories hold the outputs of the training and evaluation programs. models/ will store the various network architectures you're experimenting with (you might just have one model).

Getting a Model

Instructions were adapted from here. One of the advantages of using TFODA is that it is really easy to try different network architectures (models) and seeing their speed vs. accuracy tradeoffs. Example config files can be found https://github.com/tensorflow/models/tree/master/research/object_detection/samples/configs : here. You'll need to modify these config files a bit for your own use. In addition, most of the models have network weights that have been pretrained on some dataset. Starting from these check-points is usually much faster than training from scratch. You can locate them here.

Place the model config file and the pretrained model in the models/ directory of your workspace.

Start Training

The following commands assume your current working directory is the root of the workspace you created earlier.

To start training, run:

```
python
~/.local/tensorflow_object_detection_api/research/object_detection/train.py
\
  --logtostderr \
  --pipeline_config_path=<model config file> \
  --train_dir=output/train
```

To evaluate the performance of the network, run:

```
python
~/.local/tensorflow_object_detection_api/research/object_detection/eval.py \
  --logtostderr \
  --pipeline_config_path=<model config file> \
  --checkpoint_dir=output/train/ \
  --eval_dir=output/eval/
```

The eval.py script will notice every time the train.py script saves a new checkpoint, and evaluate its performance on the test dataset.

To visualize the training process, start up tensorboard:

```
tensorboard --logdir=outputs
```

Tensorboard is a little web server, you can access it at localhost:6006 in your browser.

Exporting a trained model for inference

To export checkpoint trained data for `robosub_object_detection` format you need to follow these instructions. Or run this:

```
python object_detection/export_inference_graph.py \
    --input_type image_tensor \
    --pipeline_config_path ${PIPELINE_CONFIG_PATH} \
    --trained_checkpoint_prefix ${TRAIN_PATH} \
    --output_directory output_inference_graph.pb
```

At this point, you should upload the label_map.pbtext and frozen_inference_graph.pb files into a uniquely named folder inside http://robosub.eecs.wsu.edu/data/vision/trained_models/, so its easy for other members to access the models.