Machine Learning Final Project Report
Andrzej Jackowski

Real Time Image Detection Through Use of the Tensor-Flow API

Introduction

In robotics computer vision is an important topic for navigation. While a robot is able to be connected to a camera to “see” the outside world, it is unable to recognize objects within the world to determine if they are of importance or not. In the past this has been remedied through various computer vision techniques such as taking advantage of contours or bright distinct colors. One novel solution has been the use of “April tags”, tags with a distinct asymmetric pattern that can be recognized easily through computer vision.

However, when dealing with new environments such as for an exploration rover on another planet, such as the NASA rovers, or navigating a disaster zone, such as the scenario for the DARPA robotics challenge, one will not necessarily have the luxury of preparing the environment with distinct tags in advance.

Using a combination of Machine learning and visual processing, it could be theoretically possible to train a machine learning model to detect any object that the operator would want the robot to recognize. This is not a new problem, several API’s and computer vision libraries have been created to address the topic of object recognition in machine learning.

The google Tensor-Flow API includes many examples and support programs to allow one to use the API to develop their own machine learning application. The API also includes a example models library that includes many optimized models for object recognition. The model being explored here is the Single Shot Multibox Detector (SSD) model with the mobilenet architecture. The SSD model uses a convolutional neural network which predicts bounding boxes of an object using features in the image. The mobilenet architecture is a lightweight convolutional neural network model developed by google for lightweight applications. This makes it ideal for real-time object detection however, the reduced detection time does come at the price of accuracy.

Here we will be using transfer learning on a pre-trained model and retraining the last layer of a pre-trained network to recognize an object that we desire.

What is important for this application is that the object is able to be detected in real time with moderate hardware, this example is being performed on a single i7 processor, which can be replicated in a robotic application with single board computers (SBC) such as the NVIDIA Jetson and the Intel Nuc.

The Goal of this project is to build a real time object detection application that can detect a desired object(s) in a video stream.

Procedure

Note: this procedure will follow the steps I took to get this project working and will address many issues with software compatibility that I had encountered and issues that may not be typical for a user and resulted from my particular operating system setup, the readme has a slightly more direct procedure for training one's own model, this is documentation of the work put into the project, however it may show some considerations that were overlooked in the readme.

Initial setup

My initial operating system was Ubuntu 14.04.
Tensor flow 1.4.1 -CPU version
Python 3.6 with Anaconda

Software required for project:

Tensor Flow Version 1.4 or above GPU or CPU (GPU is better for training)
OpenCV 3.0.0*
Protobuf 3.4 or above*
Python 3.6*
Anaconda
pip
Pillow
lxml

*Denotes installation or compatibility issues that were encountered

For background this operating system was originally configured for use with Robot Operating System, ROS (Indigo). ROS is compatible for python 2.7.

First we install the dependencies, python 3.6, Anaconda, tensor-flow

Pillow and lxml can be installed using pip.

Google has released a github repository with example models for tensor-flow to help users begin with learning how to use the API. The repository can be found here alongside the Tensor Flow open source library:

https://github.com/tensorflow/models

I cloned the git into a workspace directory where most of the development will take place looking into the models directory we find models->research->object-detection

This folder contains most of the support programs required to train one's own image detection application.

Next protobuf was installed:
https://github.com/google/protobuf/releases

I used protobuf-python-3.5.0 for the final project however there were some minor issues when installing for ubuntu 14.04 usig apt-get I needed to download and run the files from the release page.

With protobuf installed we cd to the research folder in the terminal. As per the documentation of the object-detection folder we run the following commands in the Workspace->models->research directory:

```
protoc object_detection/protos/*.proto --python_out=.
```

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

Inside the object detection folder is a juypter notebook example of the object detection API that was used as the basis for my project. After the commands above have been executed I ran the example notebook.

The example notebook shows the image detection operating with a pre trained neural network.

Now I moved to create my own dataset and retrain the model to detect my custom object.

I chose the NASA meatball as my object of interest as it has a relatively unique shape and it is something that I had printed on objects that were readily available to me.

I obtained approximately 130 images that possessed the NASA meatball and labeled the location of the nasa meatball in each image. I found the program labelImg that gave me an intuitive GUI to label the images properly:

https://github.com/tzutalin/labelImg

I created another directory in my workspace directory to host the image set. I had the following file structure:

```
Workspace->
        -> models(contains the Tensor flow example directory)

        -> dataset (separate directory for me to configure files)
                ->images
                        -> train
                        -> test
                -> data (used later)
```

After using the image labeling software xml files were generated for each image dictating where the object of interest is in each image. Next I needed to generate the tensor flow records for the images.

I found two python scripts that I modified to perfrom the operation from the following repository:

Convert xml files to CSV files:
https://github.com/datitran/raccoon_dataset/blob/master/xml_to_csv.py

I made the following modifications for the program to fit my dataset:

```
def main():
    for directory in ['train','test']:
        image_path = os.path.join(os.getcwd(), 'images/{}'.format(directory))
        xml_df = xml_to_csv(image_path)
        xml_df.to_csv('data/{}_labels.csv'.format(directory), index=None)
        print('Successfully converted xml to csv.')
```

Generate tensor-flow record files:
https://github.com/datitran/raccoon_dataset/blob/master/generate_tfrecord.py

I changed the label mapping in this program to fit my dataset:

```
def class_text_to_int(row_label):
    if row_label == 'NASA’:
        return 1
    else:
        None
```

More labels can be added if multiple objects can be detected, pending on time I may add a second label. I ran the program with the following command

```
python generate_tfrecord.py --csv_input=data/train_labels.csv --output_path=train.record
```

I ran the python scripts as per the instructions within the code. Note: I used python3 when calling all python program here.

Next I moved all the files in the dataset directory into the models->research-> object_detection directory.

Now I was able to train the model. From the research directory it may be necessary to re export the python path with the following command. (this is necessary for each instance of the terminal if you wish to run something from the image detection directory.

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

Next I obtained the pretrained model, I used the SSD model with mobilenet. As stated earlier I utilized the mobilnet model due to its lightweight architecture which will be essential for this program to run in realtime.

http://download.tensorflow.org/models/object_detection/ssd_mobilenet_v1_coco_11_06_2017.tar.gz

I created a training directory inside the object_detection directory and placed the pre-trained model in the training directory.
I extracted the model file and modified the configuration file for my own dataset. The config file I used is within this repository.

Next I created an object-detection.pbtxt file that contained my NASA label, this file is also included in the git directory:

```
item {
  id: 1
  name: 'NASA'
}
```

Making sure the python path was configured I ran the included training program(note the use of python3):

```
python3 train.py --logtostderr --train_dir=training
--pipeline_config_path=training/ssd_mobilenet_v1_pets.config
```

Unfortunately at this point things went rapidly downhill.(**Note** the next section covers several bugs and software incompatibility issues with Tensor flow and Ubuntu)

I got the error:

```
 ImportError: dlopen: cannot load any more object with static TLS
```

After several hours of research I discovered that for some reason the training program imported so many libraries that it overflowed a buffer within the operating system.
To address this problem I installed ubuntu 16.04 on my computer and carried over my dataset directory.

At this point since I was working with a fresh operating system install I had to reinstall all the required software stated above.

This is where I ran into another error. I was having compatibility issues with python 3.5, the default on ubuntu 16.04, and anaconda. I then reconfigured and installed python 3.6 on my machine through terminal commands. Unbeknownst to me there is a serious bug within

Ubuntu that when installing python 3.6 from 3.5. When installing python a bug in the operating system prevents the terminal from being reopened. As a result I was unable to use the terminal after closing the terminal window. This persisted even after a system shutdown. Turns out that the default gnome terminal requires python to operate and it needs to be reconfigured to continue usage.

After much online research I discovered that using the low level unix kernel I was able to access a terminal that allowed me to fix the problem.

Next I reinstalled protobuf using version 3.5.0.
(back to the procedure)

After fixing those issues I finally was able to train the model for my data set. I was training on my CPU so this process took 16 hours, I did not use an external server like google cloud due to fear of more incompatibility errors.
I trained the model until I got an average loss error of 1.2, normally I would train the model for longer but I wanted to see if there was any progress.
I was able to view the progress of the model training form the tensor board application.

Next I used the export_inference_graph function to obtain my trained model. As the model was trained checkpoint files were generated in the training directory.

```
python3 export_inference_graph.py \
    --input_type image_tensor \
    --pipeline_config_path training/ssd_mobilenet_v1_coco.config \
    --trained_checkpoint_prefix training/model.ckpt-5621 \
    --output_directory NASA_inference_graph
```

Note the checkpoint file is the number of steps that I trained the model for, the output directory is one that was created by the program to store the trained model.

Next the jupyter notebook in the object_detection directory was modified to utilize my trained model instead of the default model. Using few test images I deemed that this model was successful at recognizing the NASA meatball.

However there were a few errors, the model mistakenly recognized a drawing of the Dallas cowboys logo as the NASA meatball, this is a interesting note, since the dallas cowboy logo has a similar color set to the NASA meatball.

Next was the integration with OpenCV to make this a real-time object detector.

The difficulty of this section was properly configuring OpenCV to work with python 3.6. Initially I downloaded the latest version of OpenCV, this was a mistake (Note the rest of the procedure section is discussing issues with the video integration)

When using the pip or apt-get install of python 3, the VideoCapture function would not return any data, this resulted in a error when trying to read from a video stream.

This was because the ffmpeg codec was used for video processing was assumed to be part of the operating system however, this has not been true since Ubuntu 13. This normally isn't a problem for the default installation of OpenCV for other languages.
A second problem is that the default installations for OpenCV for python are configured for Python 2.7.
As a result a custom installation of OpenCV had to be done from source after a significant amount of searching online I was able to find the proper method for installing OpenCV for python 3.6. I included the custom cmake command that configures OpenCV properly for Anaconda and Python 3.6 in this repository.

After OpenCV was installed Note: I had installed version 3.3.0 I created the program that would obtain the video capture for the object detector, however the program immediately had a compilation conflict with tensor flow.

I had installed protobuf 3.5 in order to use the tensor flow image recognition library, Tensor Flow uses multiple protoc libraries for image optimization, unfortunately, OpenCV 3.3.0 uses an older version of protobuf and this lead to an incompatibility problem that caused a compilation error.

After some more research I discovered that an older version of the OpenCV library, OpenCV 3.0.0, did not utilize protobuf. I reinstalled OpenCV and finally got the integrated program to function.

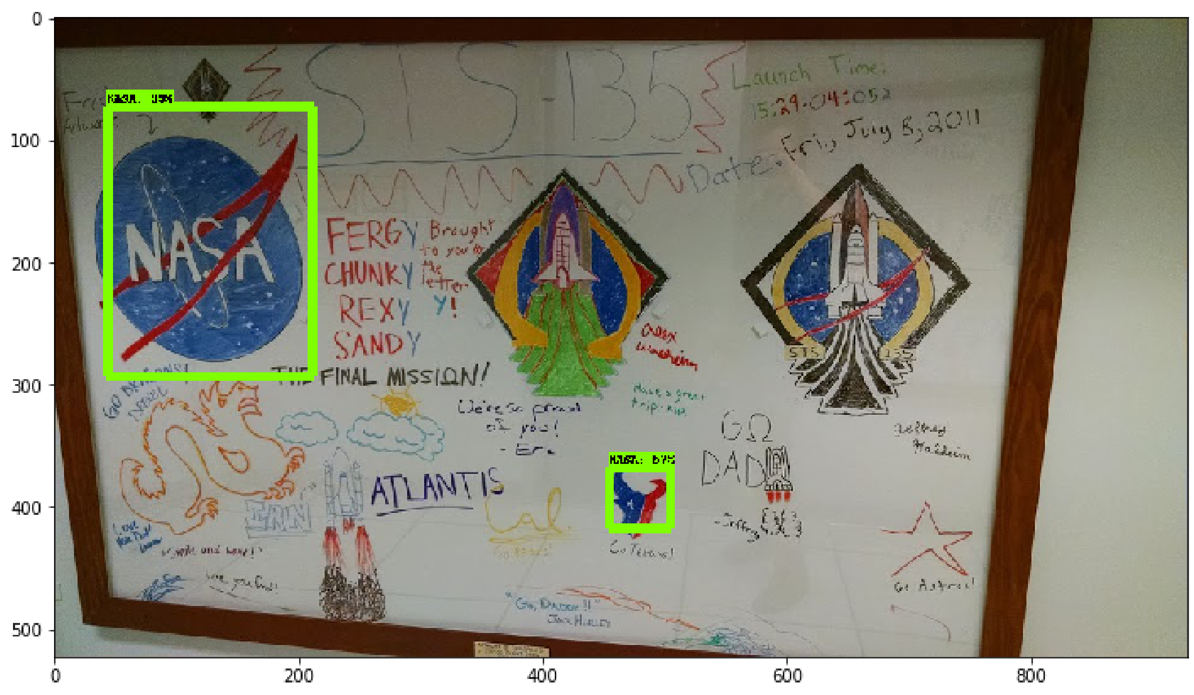
With all the training completed the application was packaged and stored on this repository with relevant custom files.

Results and Discussion

The image detection applications was ultimately successful, even despite the fact that it was not trained to the optimal point.

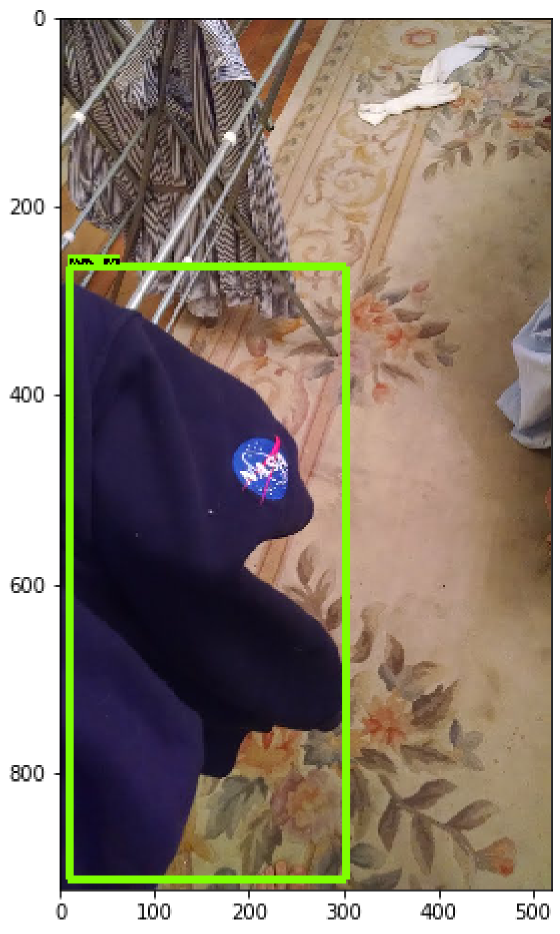
Two version of the program were made, a version that interpertits static images, using the example jupyter notebook, and the imgRec program in this repository.

The jupyter notebook version was easily able to detect the NASA logo in the given images. There was only one anomaly where the image detection was set to a drawing of the NASA logo where the program not only detected the NASA logo but also detected a drawing of the Dallas cowboys logo:

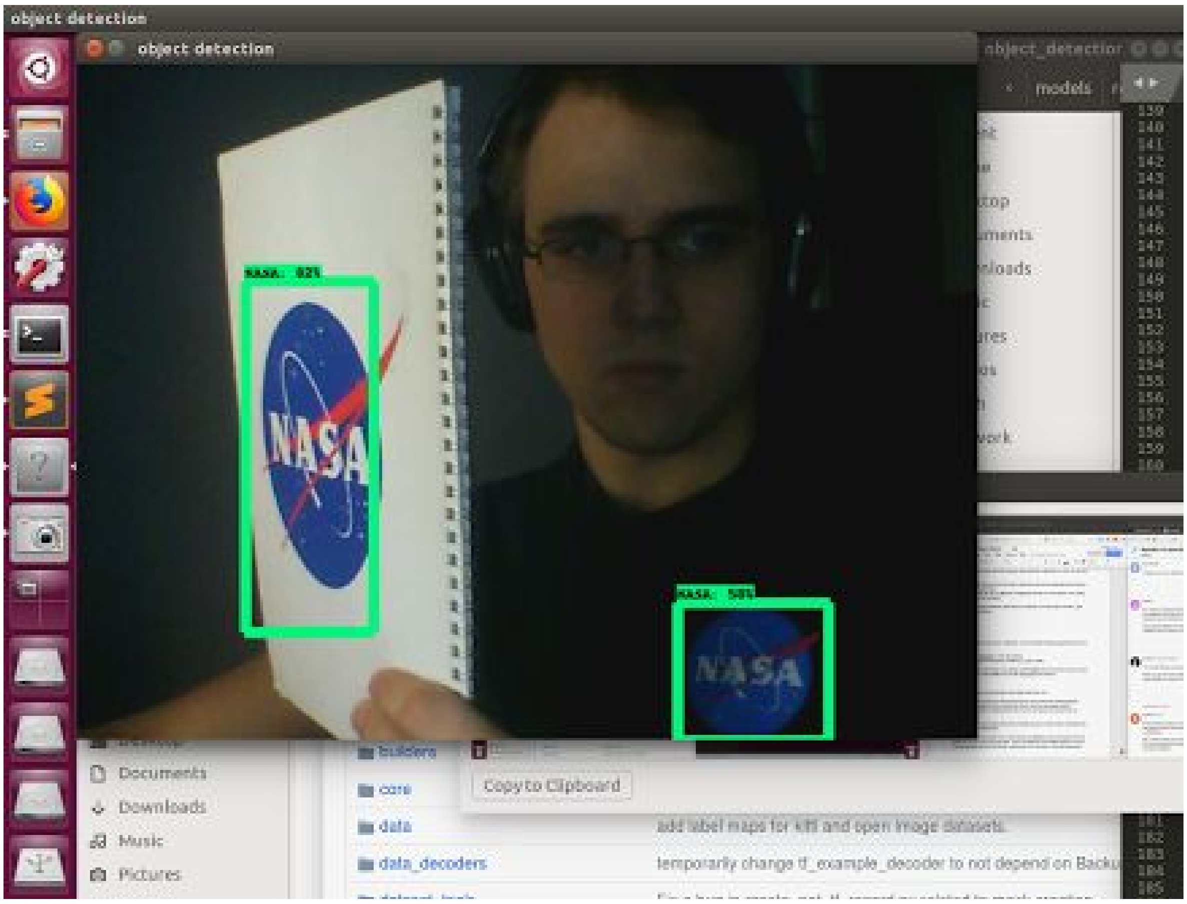


This is most likely due to the similar color schemes between the two images.

Another anomaly was when detecting the logo on a sweater, it detects the whole sleeve. Shown below:



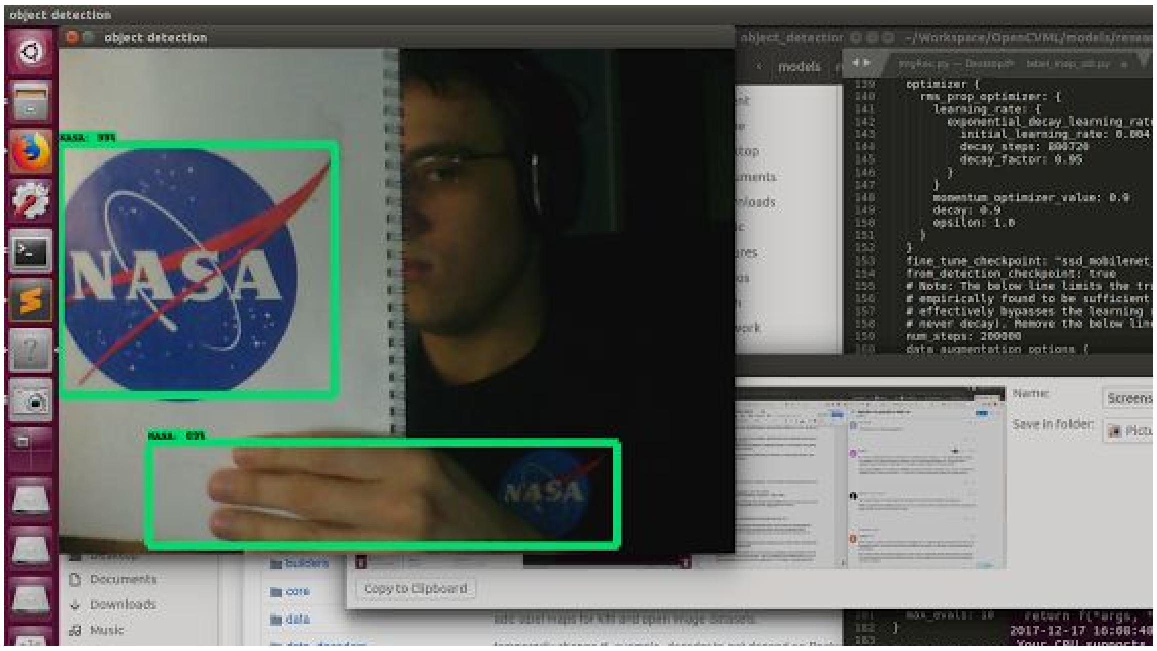
When trying to detect the NASA logo in real time, the application also had success. It was able to track the NASA logo on a notebook, citing 99% accuracy. This was however an easy test for the program for the notebook had a very clear and unobstructed view of the logo.



The real time image recognition program was able to detect the logo at a relatively steep angle (approx 60 degrees from the camera until recognition failed, showed above). The program had less success with objects such as clothing with the logo. It was able to detect the nasa logo on the sleeve of a sweater but not consistently.

Once huge factor is also blur, when an object moves fast across the screen the object blurs and as a result the computer is no longer able to recognize it. An immediate solution would be to use a higher quality camera than the webcam on my laptop that would be able to obtain a clearer image of moving objects. Another solution and a further possible direction with the project would be to also train blurred images to see if a moving object would also be recognized.

Another anomaly is the application detecting my hand at a very specific angle. This has been a repeatable result, however it is a very specific scenario. The misdetection is shown below:



```
    optimizer {
      rms_prop_optimizer: {
        learning_rate: {
          exponential_decay_learning_rate
            initial_learning_rate: 0.004
            decay_steps: 800720
            decay_factor: 0.95
          }
        }
        momentum_optimizer_value: 0.9
        decay: 0.9
        epsilon: 1.0
      }
    }
    fine_tune_checkpoint: "ssd_mobilenet_
    from_detection_checkpoint: true
    # Note: The below line limits the tr
    # empirically found to be sufficient
    # effectively bypasses the learning r
    # never decay). Remove the below line
    num_steps: 200000
    data_augmentation_options {
```

This has also occasionally resulted with heads as well however they have been brief instances.

The only other limitation would be the processing performance. While this does operate in real time there is a noticeable delay, this would not run fast on a weak processor such as a raspberry pi. However more advanced robots that utilize convolutional neural networks use stronger processors such as a an NVIDIA Jetson or an intel Nuc which runs the same i7 processor ats the computer used here.

Ultimately this has been a successful project, a real time object detector was created and can be expanded to be used in a robotic application such through the use of a robot architecture such as ROS, assuming one would be able to work past some of the compatibility issues.