**North South University**



**Department of ELECTRICAL AND COMPUTER SCIENCE**

**REPORT**

**COURSE: CSE-499-A**

**SECTION-14**

**GROUP-1**

PROJECT TITLE: GARBAGE COLLECTOR ROBOT

Supervised BY: Dr. Shazzad Hosain

**SUBMITTED BY-**

|  |  |
| --- | --- |
| **NAME** | **ID** |
| **FARJANA ALAM** | **1620060042** |
| **MOIN SHAHRIYAR** | **1610586042** |
| **SARHAN OSMAN BHUIYA** | **1611008042** |

**ABSTRACT**

We are presenting a smart system that could present a viable solution towards efficient waste management which is based on embedded, digital image processing. The system is so designed that it will automatically detect and collect the garbage. This report describes the basic idea of detection and collection. The detection is done by using the image processing algorithm. Raspberry pi camera will capture an image of a particular area, and will store it as default image. Once an object has been detected, the camera will capture its image. It will identify the object as garbage, and then further send the signals. The edge detection algorithm is used for the differentiation of the scattered edges and compact and collinear edges of the garbage. Once the camera detects the garbage, it will calculate its position, calibrate the motors according to the position of the garbage so that it will go the acquired position and collect the garbage. Once the dustbin is full up to a certain limit, the level sensor in the bin will sense the level of the garbage and send it to the nearest garbage collector truck or bin.

**INTRODUCTION**

The proposed system concentrates on identification, classification and segregation of waste. The waste, which is in unsorted manner, is dumped in a landfill, which further creates hazardous health problems. The proposed system aims to recognize and categorize the waste autonomously, which require minimal human intervention. This entire process of recognition of waste material is based on the shape and size of the objects. The system will be trained through datasets by using machine learning technique such as SINGLE SHOT DETECTION (SSD). Utilizing Raspberry Pi the characterization result will be given to the equipment part of the framework with the goal that it will be dumped in its separate containers. The system will order waste automatic categorizing them as Plastic, Paper, Glass and Metal.

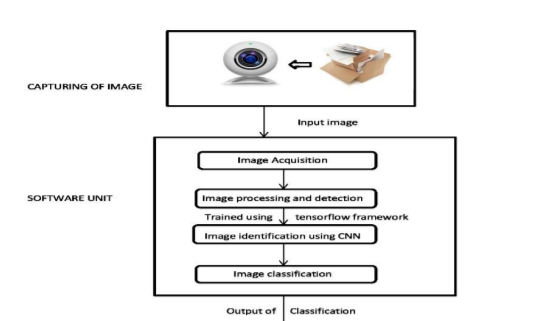
**IMPLEMENTATION FOR 499-A**

For our 499-A project we are making the software which will ultimately detect garbage and send signals to the bot to collect garbage. To complete our goal we went through the following steps:

* **SET UP RASPBERRY PI**
* **TRAINING AND DETECTION**

**PROPOSED PLAN**

The first stage is image acquisition stage. It catches image from camera with the goal that it can be passed for handling and recognition of picture. After picture has been saved, different strategies for handling can be connected to the picture to perform a wide range of vision undertakings. After analyzing, the image is processed and detected. The system is trained using Tensor flow framework. By relying on large datasets, the framework can recognize the picture and plan significant labels and classifications. The trained data is used to classify the waste into Plastic, Paper, Glass and Metal.



Input from camera is given to the raspberry pi module. This captured image then will be the main source of data for our system. The Raspberry Pi is a progression of little single-board PCs created in the United Kingdom by the Raspberry Pi Foundation to advance the educating of essential software engineering in schools and in creating nations. The Raspberry Pi Camera Module v2 is a high quality 5 megapixel. It is the backend process for classifying the images and to sort the waste autonomously.

The project has the basic idea of detection and collection. The detection is done by using the image processing algorithm. Raspberry pi camera will capture an image of a particular area, and will store it as default image. Raspberry pi camera will continue capturing images and will compare the captured images with default image .Once an object has been detected, the camera will capture its image. It will identify the object as garbage, and then further send the signals. The edge detection algorithm is used for the differentiation of the scattered edges and compact and collinear edges of the garbage.

**SETTING UP RASPBERRY PI**

* **Update the Raspberry Pi**
* **Install TensorFlow**
* **Install OpenCV**
* **Compile and Install Protobuf**
* **Set up TensorFlow directory structure**
* **Test out object detector**

## **STEPS**

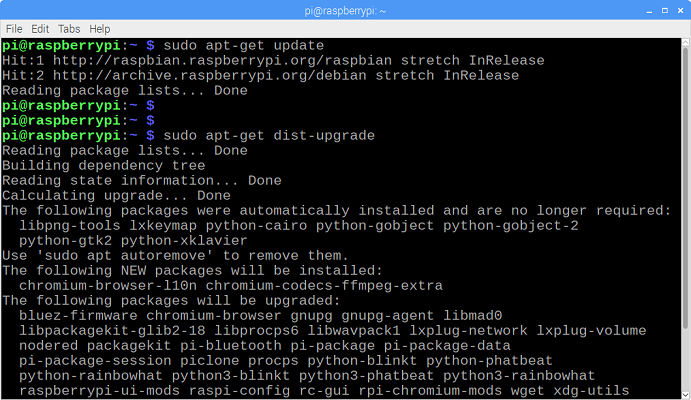
### **1. Update the Raspberry Pi**

First, the Raspberry Pi needs to be fully updated. Open a terminal and issue:

sudo apt-get update

sudo apt-get dist-upgrade

Depending on how long it’s been since we’ve updated our Pi, the upgrade could take anywhere between a minute and an hour.

[](https://github.com/EdjeElectronics/TensorFlow-Object-Detection-on-the-Raspberry-Pi/blob/master/doc/update.png)

### **2. Install TensorFlow**

Next, we’ll install TensorFlow. In the /home/pi directory, create a folder called ‘tf’, which will be used to hold all the installation files for TensorFlow and Protobuf, and cd into it:

mkdir tf

cd tf

A pre-built, Rapsberry Pi-compatible wheel file for installing the latest version of TensorFlow is available in the [“TensorFlow for ARM” GitHub repository](https://github.com/lhelontra/tensorflow-on-arm/releases). GitHub user lhelontra updates the repository with pre-compiled installation packages each time a new TensorFlow is released. Download the wheel file by issuing:

wget <https://github.com/lhelontra/tensorflow-on-arm/releases/download/v1.14.0/tensorflow-1.14.0-cp35-none-linux_armv7l.whl>

At the time this tutorial was written, the most recent version of TensorFlow was version 1.14.0. If a more recent version is available on the repository, we can download it rather than version 1.14.0.

Now that we’ve got the file, install TensorFlow by issuing:

sudo pip3 install /home/pi/tf/tensorflow-1.14.0-cp35-none-linux\_armv7l.whl

TensorFlow also needs the LibAtlas package. Install it by issuing (if this command doesn't work, issue "sudo apt-get update" and then try again):

sudo apt-get install libatlas-base-dev

While we’re at it, let’s install other dependencies that will be used by the TensorFlow Object Detection API. These are listed in TensorFlow’s Object Detection GitHub repository. Issue:

sudo pip3 install pillow lxml jupyter matplotlib cython

sudo apt-get install python-tk

Alright, that’s everything we need for TensorFlow! Next up: OpenCV.

### **3. Install OpenCV**

TensorFlow’s object detection examples typically use matplotlib to display images, but I prefer to use OpenCV because it’s easier to work with and less error prone. The object detection scripts in this guide’s GitHub repository use OpenCV. So, we need to install OpenCV.

To get OpenCV working on the Raspberry Pi, there’s quite a few dependencies that need to be installed through apt-get. If any of the following commands don’t work, issue “sudo apt-get update” and then try again. Issue:

sudo apt-get install libjpeg-dev libtiff5-dev libjasper-dev libpng12-dev

sudo apt-get install libavcodec-dev libavformat-dev libswscale-dev libv4l-dev

sudo apt-get install libxvidcore-dev libx264-dev

sudo apt-get install qt4-dev-tools

Now that we’ve got all those installed, we can install OpenCV. Issue:

pip3 install opencv-python

Alright, now OpenCV is installed!

### **4. Compile and Install Protobuf**

Okay, here comes the hard part. The TensorFlow object detection API uses Protobuf, a package that implements Google’s Protocol Buffer data format. Unfortunately, there’s currently no easy way to install Protobuf on the Raspberry Pi. We have to compile it from source ourselves and then install it. First, get the packages needed to compile Protobuf from source. Issue:

sudo apt-get install autoconf automake libtool curl

Then download the protobuf release from its GitHub repository by issuing:

wget https://github.com/google/protobuf/releases/download/v3.5.1/protobuf-all-3.5.1.tar.gz

If a more recent version of protobuf is available, download that instead. Unpack the file and cd into the folder:

tar -zxvf protobuf-all-3.5.1.tar.gz

cd protobuf-3.5.1

Configure the build by issuing the following command (it takes about 2 minutes):

./configure

Build the package by issuing:

make

The build process took 61 minutes on my Raspberry Pi. When it’s finished, issue:

make check

This process takes even longer, clocking in at 107 minutes on my Pi. According to other guides I’ve seen, this command may exit out with errors, but Protobuf will still work. If we see errors, we can ignore them for now. Now that it’s built, install it by issuing:

sudo make install

Then move into the python directory and export the library path:

python3 setup.py build --cpp\_implementation

python3 setup.py test --cpp\_implementation

sudo python3 setup.py install --cpp\_implementation

Then issue the following path commands:

export PROTOCOL\_BUFFERS\_PYTHON\_IMPLEMENTATION=cpp

export PROTOCOL\_BUFFERS\_PYTHON\_IMPLEMENTATION\_VERSION=3

Finally, issue:

sudo ldconfig

That’s it! Now Protobuf is installed on the Pi. Verify it’s installed correctly by issuing the command below and making sure it puts out the default help text.

protoc

For some reason, the Raspberry Pi needs to be restarted after this process, or TensorFlow will not work. Go ahead and reboot the Pi by issuing:

sudo reboot now

### **5. Set up TensorFlow Directory Structure and PYTHONPATH Variable**

Now that we’ve installed all the packages, we need to set up the TensorFlow directory. Move back to our home directory, then make a directory called “tensorflow1”, and cd into it.

mkdir tensorflow1

cd tensorflow1

Download the tensorflow repository from GitHub by issuing:

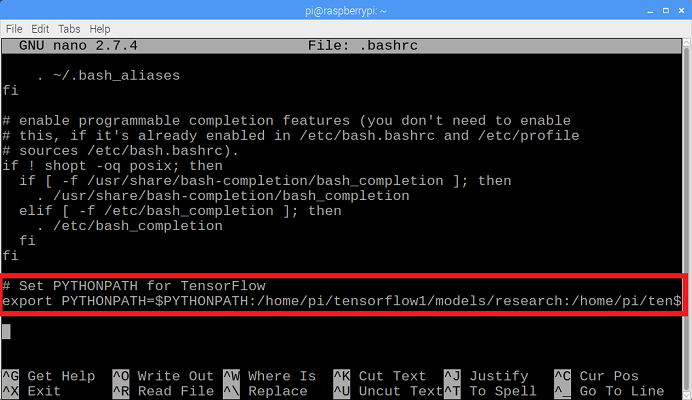
git clone --recurse-submodules https://github.com/tensorflow/models.git

Next, we need to modify the PYTHONPATH environment variable to point at some directories inside the TensorFlow repository we just downloaded. We want PYTHONPATH to be set every time we open a terminal, so we have to modify the .bashrc file. Open it by issuing:

sudo nano ~/.bashrc

Move to the end of the file, and on the last line, add:

export PYTHONPATH=$PYTHONPATH:/home/pi/tensorflow1/models/research:/home/pi/tensorflow1/models/research/slim

[](https://github.com/EdjeElectronics/TensorFlow-Object-Detection-on-the-Raspberry-Pi/blob/master/doc/bashrc.png)

Then, save and exit the file. This makes it so the “export PYTHONPATH” command is called every time we open a new terminal, so the PYTHONPATH variable will always be set appropriately. Close and then re-open the terminal.

Now, we need to use Protoc to compile the Protocol Buffer (.proto) files used by the Object Detection API. The .proto files are located in /research/object\_detection/protos, but we need to execute the command from the /research directory. Issue:

cd /home/pi/tensorflow1/models/research

protoc object\_detection/protos/\*.proto --python\_out=.

This command converts all the "name".proto files to "name\_pb2".py files. Next, move into the object\_detection directory:

cd /home/pi/tensorflow1/models/research/object\_detection

Now, we’ll download the SSD\_Lite model from the [TensorFlow detection model zoo](https://github.com/tensorflow/models/blob/master/research/object_detection/g3doc/detection_model_zoo.md). The model zoo is Google’s collection of pre-trained object detection models that have various levels of speed and accuracy. The Raspberry Pi has a weak processor, so we need to use a model that takes less processing poour. Though the model will run faster, it comes at a tradeoff of having loour accuracy. For this tutorial, we’ll use SSDLite-MobileNet, which is the fastest model available.

Google is continuously releasing models with improved speed and performance, so check back at the model zoo often to see if there are any better models.

Download the SSDLite-MobileNet model and unpack it by issuing:

wget http://download.tensorflow.org/models/object\_detection/ssdlite\_mobilenet\_v2\_coco\_2018\_05\_09.tar.gz

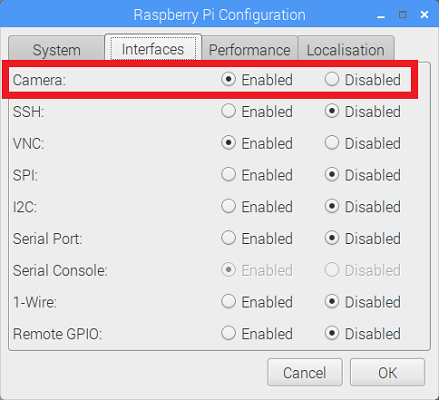
tar -xzvf ssdlite\_mobilenet\_v2\_coco\_2018\_05\_09.tar.gz

Now the model is in the object\_detection directory and ready to be used.

### **6. Detect Objects**

Okay, now everything is set up for performing object detection on the Pi! The Python script in this repository, Object\_detection\_picamera.py, detects objects in live feeds from a Picamera or USB webcam. Basically, the script sets paths to the model and label map, loads the model into memory, initializes the Picamera, and then begins performing object detection on each video frame from the Picamera.

If we’re using a Picamera, make sure it is enabled in the Raspberry Pi configuration menu.

[](https://github.com/EdjeElectronics/TensorFlow-Object-Detection-on-the-Raspberry-Pi/blob/master/doc/camera_enabled.png)

Download the Object\_detection\_picamera.py file into the object\_detection directory by issuing:

wget https://raw.githubusercontent.com/TensorFlow-Object-Detection-on-the-Raspberry-Pi/master/Object\_detection\_picamera.py

Run the script by issuing:

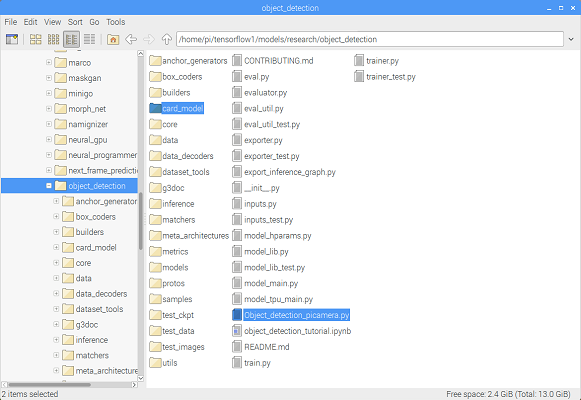
python3 Object\_detection\_picamera.py

The script defaults to using an attached Picamera. If we have a USB webcam instead, add --usbcam to the end of the command:

python3 Object\_detection\_picamera.py --usbcam

Once the script initializes (which can take up to 30 seconds), we will see a window showing a live view from our camera. Common objects inside the view will be identified and have a rectangle drawn around them.

With the SSDLite model, the Raspberry Pi 3 performs fairly well, achieving a frame rate higher than 1FPS. This is fast enough for most real-time object detection applications.

[](https://github.com/EdjeElectronics/TensorFlow-Object-Detection-on-the-Raspberry-Pi/blob/master/doc/directory.png)

Then, open the Object\_detection\_picamera.py script in a text editor. Go to the line where MODEL\_NAME is set and change the string to match the name of the new model folder. Then, on the line where PATH\_TO\_LABELS is set, change the name of the labelmap file to match the new label map. Change the NUM\_CLASSES variable to the number of classes our model can identify.

Now that we’ve installed all the packages, we need to set up the TensorFlow directory. Move back to our home directory, then make a directory called “tensorflow1”, and cd into it.

mkdir tensorflow1

cd tensorflow1

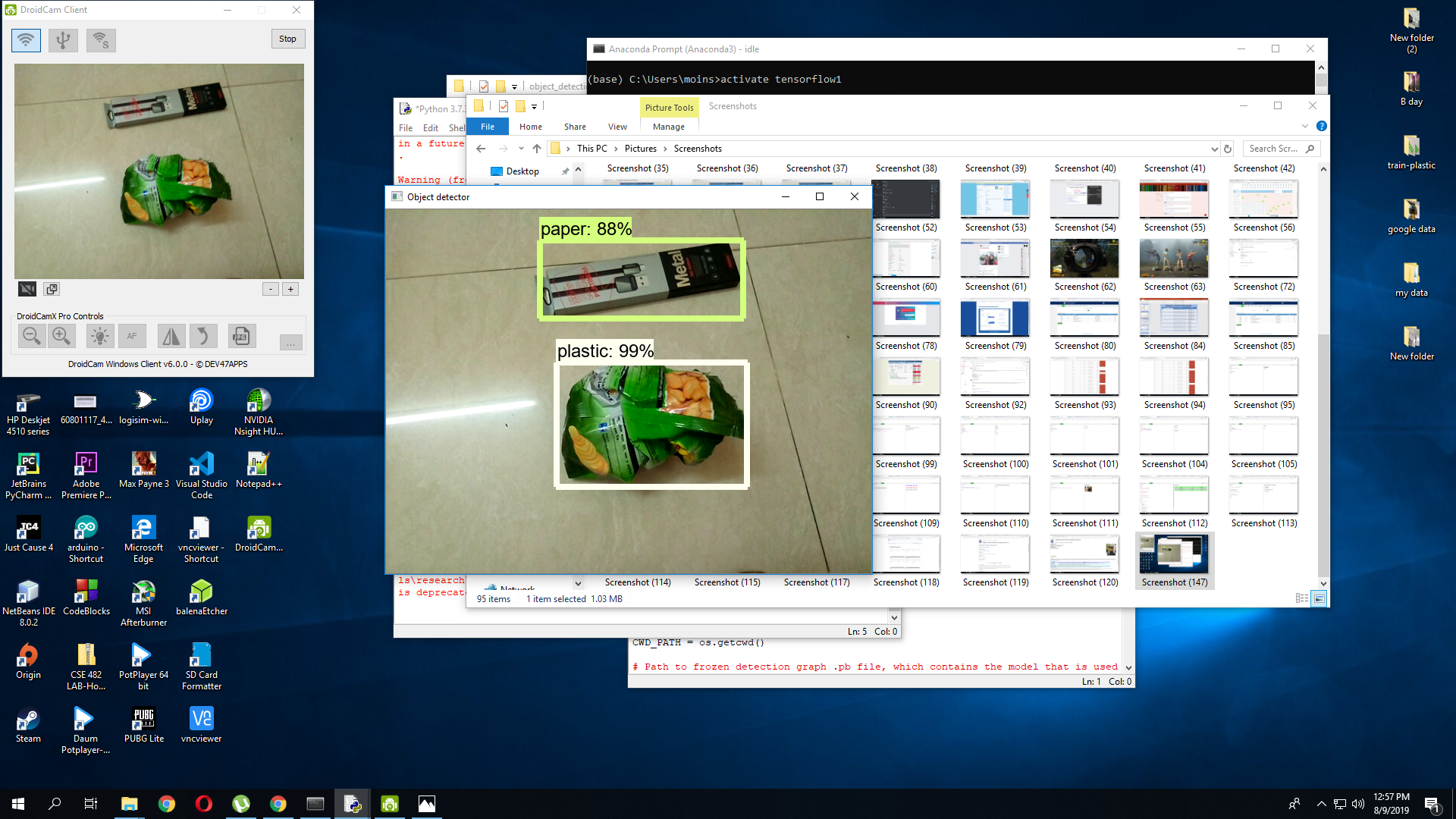
Download the tensorflow repository from GitHub by issuing:

git clone --recurse-submodules https://github.com/tensorflow/models.git

Next, we need to modify the PYTHONPATH environment variable to point at some directories inside the TensorFlow repository we just downloaded. We want PYTHONPATH to be set every time we open a terminal, so we have to modify the .bashrc file. Open it by issuing:

Now, when we run the script, it will use our model rather than the SSDLite\_MobileNet model. If we’re using my model, it will detect and identify any playing cards dealt in front of the camera.

**Note: We plan to run this on the Pi for extended periods of time (greater than 5 minutes), so we make sure to have a heatsink installed on the Pi's main CPU! All the processing causes the CPU to run hot. Without a heatsink, it will shut down due to high temperature.**



**TRAINING OBJECT DETECTION**

Here we explain how to train our own convolutional neural network object detection classifier for multiple objects, starting from scratch. We will have a program that can identify and draw boxes around specific objects in pictures, videos, or in a Raspberry Pi Camera.

We are using Tensor Flow’s Object Detection API to train a classifier for a single object. To set up Tensor Flow to train a model on Windows, there are several workarounds that need to be used in place of commands. We also provide instructions for training a classifier that can detect multiple objects, not just one. We used Tensor Flow-GPU v1.5. Tensor Flow-GPU allows our PC to use the video card to provide extra processing poour while training, so it will be used for this tutorial. Using Tensor Flow-GPU instead of regular Tensor Flow reduces training time by a factor of about 8 (3 hours to train instead of 24 hours).

* **Installing Anaconda, CUDA, and cuDNN**
* **Setting up the Object Detection directory structure and Anaconda Virtual Environment**
* **Gathering and labeling pictures**
* **Generating training data**
* **Creating a label map and configuring training**
* **Training**
* **Exporting the inference graph**
* **Testing and using our newly trained object detection classifier**

Tensor Flow provides several object detection models (pre-trained classifiers with specific neural network architectures) in its model zoo. We used the models SSDLite which has an architecture that allows for faster detection but with less accuracy, while some models (such as the Faster-RCNN model) give slower detection but with more accuracy. We initially started with the SSD-MobileNet-V2 model and it did do a very good job identifying the garbage in my images. Okay, now everything is set up for performing object detection on the Pi! The Python script in this repository, Object\_detection\_picamera.py, detects objects in live feeds from a Picamera or USB webcam. Basically, the script sets paths to the model and label map, loads the model into memory, initializes the Picamera, and then begins performing object detection on each video frame from the Picamera.

**WORKING OF SYSTEM TO RECOGNIZE AND DETECT THE IMAGES**

The first step is to train the images (data) after collection of the dataset. To obtain the accurate results, training of the dataset is very vital. A very large number of images are given as an input to train the dataset.

1. **Place the waste object in front of camera.**
2. **The camera will capture the image and it will transmit to the system.**
3. **The system will identify the object using Tensor Flow.**
4. **Further the object will be detected and classified using CNN algorithm. Thus, CNN will result the waste as degradable or non-degradable waste.**
5. **CNN will detect the waste and take the input as an array of pixel values.**
6. **The pixel values of image will be multiplied with the filter values.**
7. **The multiplication will be summed and the entire procedure will be repeated for the whole image.**
8. **Further max pooling will get an output, which has the maximum value in particular, window by reducing the parameters and generalizes the convolutional layer.**
9. **It then determines the features which most correlates to a particular class (dataset). Thus, the waste will be classified.**
10. **The result of classification will remain in Raspberry Pi.**
11. **Raspberry Pi will be programmed so that it instructs the motor and flap to dump the classified waste into the respective bins.**

## **Steps**

### **1. Install Anaconda, CUDA, and cuDNN**

Follow [this WeTube video by Mark Jay](https://www.youtube.com/watch?v=RplXYjxgZbw), which shows the process for installing Anaconda, CUDA, and cuDNN. We do not need to actually install TensorFlow as shown in the video, because we will do that later in Step 2. The video is made for TensorFlow-GPU v1.14, so download and install the CUDA and cuDNN versions for the latest TensorFlow version, rather than CUDA v10.0 and cuDNN v8.0 as instructed in the video. The [TensorFlow website](https://www.tensorflow.org/install/gpu) indicates which versions of CUDA and cuDNN are needed for the latest version of TensorFlow.

If we are using an older version of TensorFlow, make sure we use the CUDA and cuDNN versions that are compatible with the TensorFlow version we are using. [Here](https://www.tensorflow.org/install/source#tested_build_configurations) is a table showing which version of TensorFlow requires which versions of CUDA and cuDNN.

Be sure to install [Anaconda](https://www.anaconda.com/distribution/#download-section) as instructed in the video, because the Anaconda virtual environment will be used for the rest of this tutorial. (Note: The current version of Anaconda uses Python 3.7, which is not officially supported by TensorFlow. However, when creating an Anaconda virtual environment during Step 2d of this tutorial, we will tell it to use Python 3.5.)

Visit [TensorFlow's website](https://www.tensorflow.org/install) for further installation details, including how to install it on other operating systems (like Linux). The [object detection repository](https://github.com/tensorflow/models/tree/master/research/object_detection) itself also has [installation instructions](https://github.com/tensorflow/models/blob/master/research/object_detection/g3doc/installation.md).

### **2. Set up TensorFlow Directory 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.

This portion of the tutorial goes over the full set up required. It is fairly meticulous, but follow the instructions closely, because improper setup can cause unwieldy errors down the road.

#### **2a. Download TensorFlow Object Detection API repository from GitHub**

Create a folder directly in C: and name it “tensorflow1”. This working directory will contain the full TensorFlow object detection framework, as well as our 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 the C:\tensorflow1 directory we just created. Rename “models-master” to just “models”.

**Note: The TensorFlow models repository's code (which contains the object detection API) is continuously updated by the developers. Sometimes they make changes that break functionality with old versions of TensorFlow. It is always best to use the latest version of TensorFlow and download the latest models repository. If we are not using the latest version, clone or download the commit for the version we are using as listed in the table below.**

If we are using an older version of TensorFlow, here is a table showing which GitHub commit of the repository we should use. I generated this by going to the release branches for the models repository and getting the commit before the last commit for the branch. (They remove the research folder as the last commit before they create the official version release.)

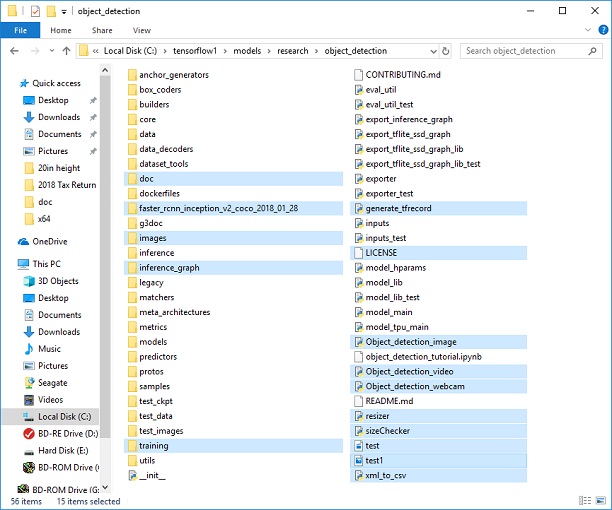
If we are using an older version of TensorFlow, make sure we use the CUDA and cuDNN versions that are compatible with the TensorFlow version we are using. [Here](https://www.tensorflow.org/install/source#tested_build_configurations) is a table showing which version of TensorFlow requires which versions of CUDA and cuDNN. Be sure to install [Anaconda](https://www.anaconda.com/distribution/#download-section) as instructed in the video, because the Anaconda virtual environment will be used for the rest of this tutorial. (Note: The current version of Anaconda uses Python 3.7, which is not officially supported by TensorFlow. However, when creating an Anaconda virtual environment during Step 2d of this tutorial, we will tell it to use Python 3.5.)

#### **2b. Download the SSDLite-Mobilenet V2-COCO model from TensorFlow's model zoo**

TensorFlow provides several object detection models (pre-trained classifiers with specific neural network architectures) in its [model zoo](https://github.com/tensorflow/models/blob/master/research/object_detection/g3doc/detection_model_zoo.md). Some models (such as the SSD-MobileNet model) have an architecture that allows for faster detection but with less accuracy, while some models (such as the Faster-RCNN model) give sloour detection but with more accuracy. I initially started with the SSD-MobileNet-V1 model, but it didn’t do a very good job identifying the cards in my images. I re-trained my detector on the Faster-RCNN-Inception-V2 model, and the detection worked considerably better, but with a noticeably sloour speed.We can choose which model to train our objection detection classifier on. If we are planning on using the object detector on a device with low computational poour (such as a smart phone or Raspberry Pi), use the SSD-MobileNet model. If we will be running our detector on a decently pooured laptop or desktop PC, use one of the RCNN models.This tutorial will use the SSDLite-Mobilenet -V2 model.

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. (We can overwrite the existing "README.md" file.) This establishes a specific directory structure that will be used for the rest of the tutorial.

At this point, here is what our \object\_detection folder should look like:

[](https://github.com/EdjeElectronics/TensorFlow-Object-Detection-API-Tutorial-Train-Multiple-Objects-Windows-10/blob/master/doc/object_detection_directory.jpg)

This repository contains the images, annotation data, .csv files, and TFRecords needed to train a "garbage\_model" detector. We can use these images and data to practice making our own. 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. We can ignore the \doc folder and its files; they are just there to hold the images used for this readme.

#### **2d. 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”. If Windows asks we if we would like to allow it to make changes to our computer, click Yes.

In the command terminal that pops up, create a new virtual environment called “tensorflow1” by issuing the following command:

C:\> conda create -n tensorflow1 pip python=3.5

Then, activate the environment and update pip by issuing:

C:\> activate tensorflow1

(tensorflow1) C:\>python -m pip install --upgrade pip

Install tensorflow-gpu in this environment by issuing:

(tensorflow1) C:\> pip install --ignore-installed --upgrade tensorflow-gpu

(Note: We can also use the CPU-only version of TensorFow, but it will run much sloour. If we want to use the CPU-only version, just use "tensorflow" instead of "tensorflow-gpu" in the previous command.)

Install the other necessary packages by issuing the following commands:

(tensorflow1) C:\> conda install -c anaconda protobuf

(tensorflow1) C:\> pip install pillow

(tensorflow1) C:\> pip install lxml

(tensorflow1) C:\> pip install Cython

(tensorflow1) C:\> pip install contextlib2

(tensorflow1) C:\> pip install jupyter

(tensorflow1) C:\> pip install matplotlib

(tensorflow1) C:\> pip install pandas

(tensorflow1) 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.)

#### **2e. Configure PYTHONPATH environment variable**

A PYTHONPATH variable must be created that points to the \models, \models\research, and \models\research\slim directories. Do this by issuing the following commands (from any directory):

(tensorflow1) C:\> set PYTHONPATH=C:\tensorflow1\models;C:\tensorflow1\models\research;C:\tensorflow1\models\research\slim

(Note: Every time the "tensorflow1" virtual environment is exited, the PYTHONPATH variable is reset and needs to be set up again. We can use "echo %PYTHONPATH% to see if it has been set or not.)

#### **2f. Compile Protobufs and run setup.py**

Next, 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](https://github.com/tensorflow/models/blob/master/research/object_detection/g3doc/installation.md) does not work on Windows. Every .proto file in the \object\_detection\protos directory must be called out individually by the command.

In the Anaconda Command Prompt, change directories to the \models\research directory:

(tensorflow1) C:\> cd C:\tensorflow1\models\research

Then 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\ssdlite mobilenet.proto .\object\_detection\protos\ ssdlite mobilenet \_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 .\object\_detection\protos\graph\_rewriter.proto .\object\_detection\protos\calibration.proto .\object\_detection\protos\flexible\_grid\_anchor\_generator.proto

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

**(Note: TensorFlow occassionally adds new .proto files to the \protos folder. If we get an error saying ImportError: cannot import name 'something\_something\_pb2' , we may need to update the protoc command to include the new .proto files.)**

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

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

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

#### **2g. Test TensorFlow setup to verify it works**

The TensorFlow Object Detection API is now all set up to use pre-trained models for object detection, or to train a new one. We can test it out and verify our installation is working by launching the object\_detection\_tutorial.ipynb script with Jupyter. From the \object\_detection directory, issue this command:

(tensorflow1) C:\tensorflow1\models\research\object\_detection> jupyter notebook object\_detection\_tutorial.ipynb

This opens the script in our default web browser and allows we to step through the code one section at a time. We can step through each section by clicking the “Run” button in the upper toolbar. The section is done running when the “In [ \* ]” text next to the section populates with a number (e.g. “In [1]”).

(Note: part of the script downloads the ssd\_mobilenet\_v1 model from GitHub, which is about 74MB. This means it will take some time to complete the section, so be patient.)

Once we have stepped all the way through the script, we should see two labeled images at the bottom section the page. If we see this, then everything is working properly! If not, the bottom section will report any errors encountered.

**Note: If we run the full Jupyter Notebook without getting any errors, but the labeled pictures still don't appear, try this: go in to object\_detection/utils/visualization\_utils.py and comment out the import statements on line 25 and 26 that include matplotlib. Then, try re-running the Jupyter notebook.**

### **3. Gather and Label 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.

#### **3a. 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 Garbage Model classifier, I have four different objects I want to detect. We took several images and combined with kaggle dataset to get our complete dataset. In total 2532 images for training and testing.



Make sure the images aren’t too large. 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. We can use the resizer.py script in this repository to reduce the size of the images.

After we have all the pictures we 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.

#### **3b. Label Pictures**

Here comes the fun part! With all the pictures gathered, it’s time to label the desired objects in every picture. LabelImg is a great tool for labeling images, and its GitHub page has very clear instructions on how to install and use it.

Download and install LabelImg, point it to our \images\train directory, and then draw a box around each object in each image. Repeat the process for all the images in the \images\test directory. This will take a while!

LabelImg saves .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 we have labeled and saved each image, there will be one .xml file for each image in the \test and \train directories.

### **4. Generate Training Data**

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, 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:

(tensorflow1) 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.

Next, open the generate\_tfrecord.py file in a text editor. Replace the label map starting at line 31 with our own label map, where each object is assigned an ID number. This same number assignment will be used when configuring the labelmap.pbtxt file in Step 5b.

def class\_text\_to\_int(row\_label):

if row\_label == 'glass':

return 1

elif row\_label == 'metal':

return 2

elif row\_label == 'paper':

return 3

elif row\_label == 'plastic':

return 4

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. Create Label Map and Configure Training**

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

#### **5a. 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. (Make sure 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 Garbage model):

item {

id: 1

name: 'glass'

}

item {

id: 2

name: 'metal'

}

item {

id: 3

name: 'paper'

}

item {

id: 4

name: 'plastic'

}

#### **5b. 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!

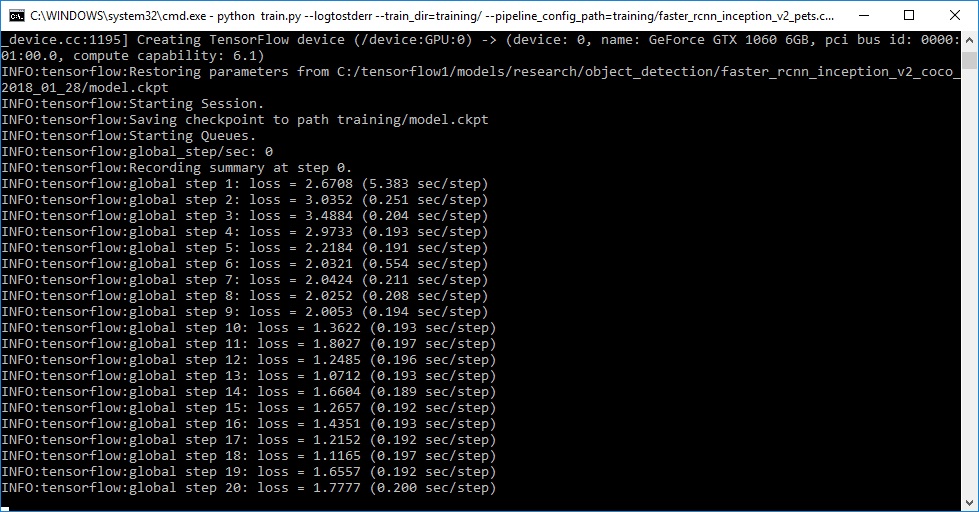
Navigate to C:\tensorflow1\models\research\object\_detection\samples\configs and copy the ssdlite-mobilenet\_v2.config file into the \object\_detection\training directory. Then, open the file with a text editor. There are several changes to make to the .config file, mainly changing the number of classes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

### **6. Run the Training**

Here we go! From the \object\_detection directory, issue the following command to begin training:

python train.py --logtostderr --train\_dir=training/ --pipeline\_config\_path=training/ssdlite-mobilenet\_v2.config

If everything has been set up correctly, TensorFlow will initialize the training. The initialization can take up to 30 seconds before the actual training begins. When training begins, it will look like this:

[](https://github.com/EdjeElectronics/TensorFlow-Object-Detection-API-Tutorial-Train-Multiple-Objects-Windows-10/blob/master/doc/training.jpg)

We can view the progress of the training job by using TensorBoard. To do this, open a new instance of Anaconda Prompt, activate the tensorflow1 virtual environment, change to the C:\tensorflow1\models\research\object\_detection directory, and issue the following command:

(tensorflow1) C:\tensorflow1\models\research\object\_detection>tensorboard --logdir=training

This will create a webpage on our local machine at OurPCName: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. We 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. We 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.

python train.py --logtostderr --train\_dir=training/ --pipeline\_config\_path=training/ ssdlite-mobilenet\_v2.config

### **7. Export 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/ ssdlite-mobilenet\_v2.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.

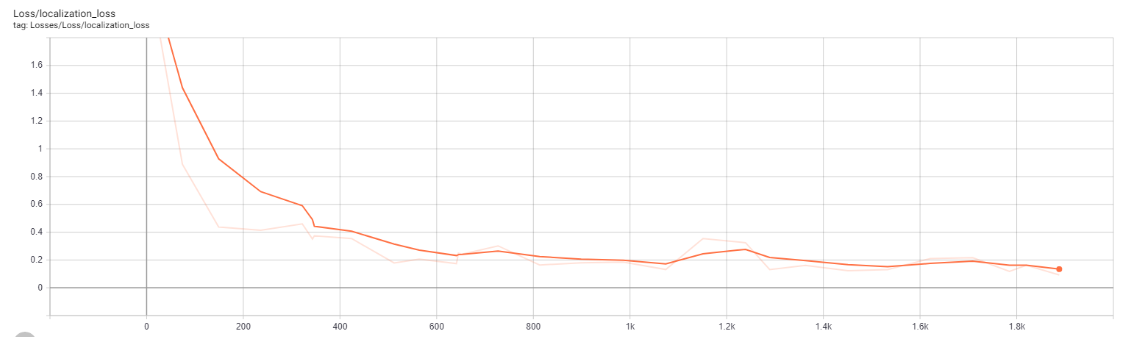


Fig: Localized Loss

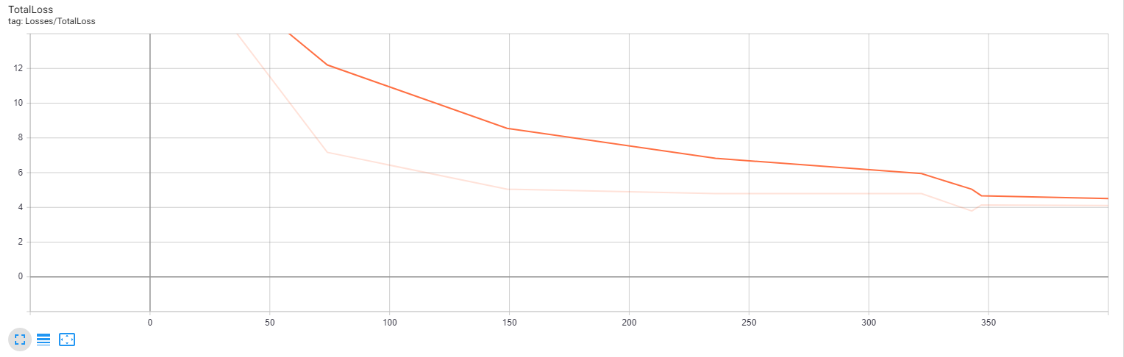
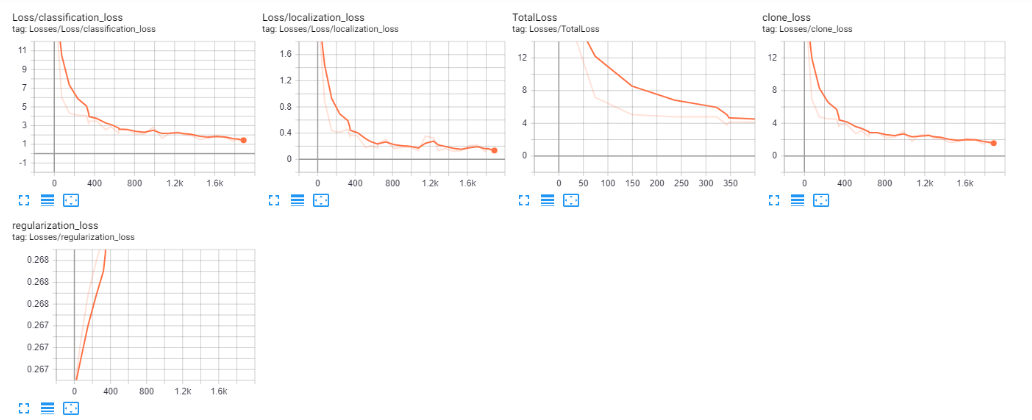
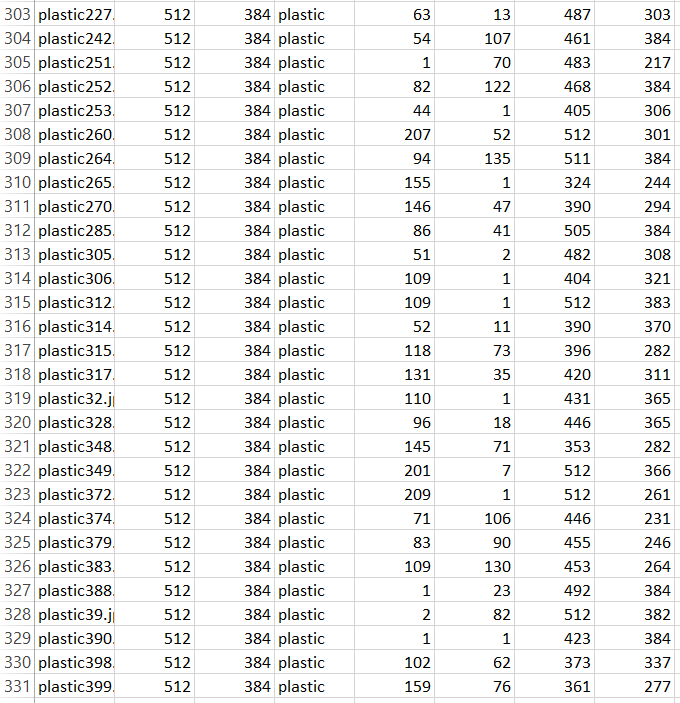
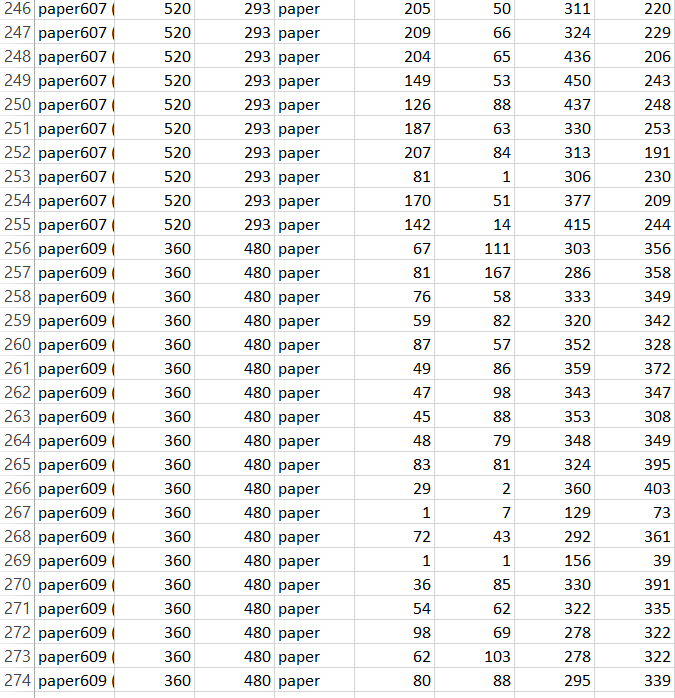
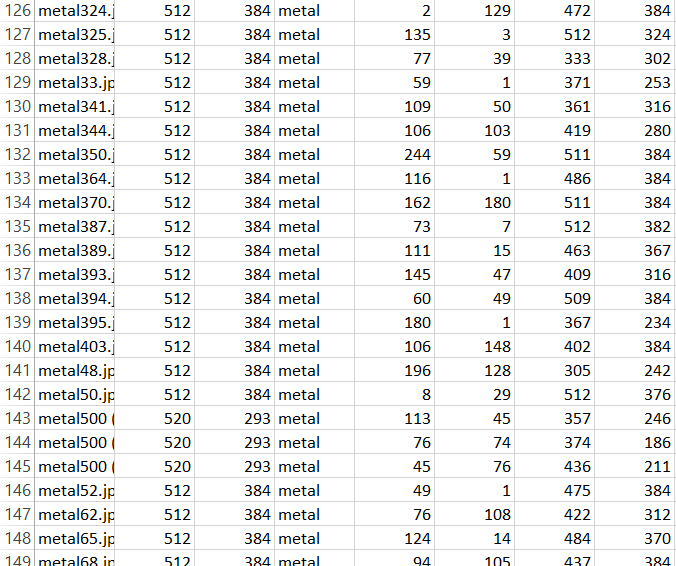
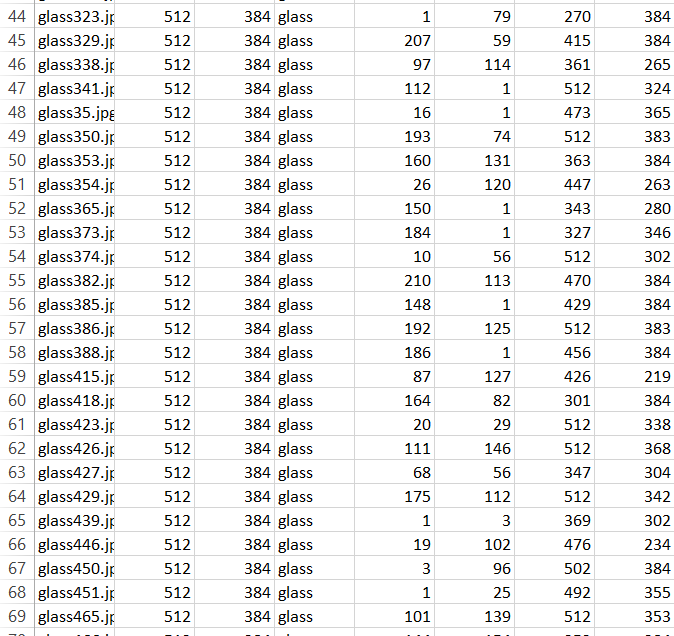


Fig: Total Loss



### **8. Creating CSV File**

A CSV file has been automatically created when we finished labeling our images for training purpose. It contains the file name, width, height, class, x-axis and y-axis of the images we labeled.



### **9. Use Our Newly Trained Object Detection Classifier!**

The object detection classifier is all ready to go! I’ve written Python scripts to test it out on an image, video, or webcam feed.

Before running the Python scripts, we need to modify the NUM\_CLASSES variable in the script to equal the number of classes we want to detect. Our NUM\_CLASSES:4

To test our object detector, move a picture of the object or objects into the \object\_detection folder, and change the IMAGE\_NAME variable in the Object\_detection\_image.py to match the file name of the picture.

To run any of the scripts, type “idle” in the Anaconda Command Prompt (with the “tensorflow1” virtual environment activated) and press ENTER. This will open IDLE, and from there, we can open any of the scripts and run them.

If everything is working properly, the object detector will initialize for about 10 seconds and then display a window showing any objects it’s detected in the image!

**CODE**

**######## Picamera Object Detection Using Tensorflow Classifier #########**

**# Description:**

**# This program uses a TensorFlow classifier to perform object detection.**

**# It loads the classifier uses it to perform object detection on a Picamera feed.**

**# It draws boxes and scores around the objects of interest in each frame from**

**# the Picamera. It also can be used with a webcam by adding "--usbcam"**

**# when executing this script from the terminal.**

**## Some of the code is copied from Google's example at**

**## https://github.com/tensorflow/models/blob/master/research/object\_detection/object\_detection\_tutorial.ipynb**

**## and some is copied from Dat Tran's example at**

**## https://github.com/datitran/object\_detector\_app/blob/master/object\_detection\_app.py**

**# Import packages**

**import os**

**import cv2**

**import numpy as np**

**from picamera.array import PiRGBArray**

**from picamera import PiCamera**

**import tensorflow as tf**

**import argparse**

**import sys**

**import RPi.GPIO as GPIO**

**import time**

**#set up GPIO pins for each class type**

**GPIO.setmode(GPIO.BCM)**

**GPIO.setup(18, GPIO.OUT)**

**GPIO.setup(23, GPIO.OUT)**

**GPIO.setup(24, GPIO.OUT)**

**GPIO.setup(25, GPIO.OUT)**

**count\_glass=0**

**count\_metal=0**

**count\_paper=0**

**count\_plastic=0**

**# Set up camera constants**

**IM\_WIDTH = 1280**

**IM\_HEIGHT = 720**

**#IM\_WIDTH = 640 Use smaller resolution for**

**#IM\_HEIGHT = 480 slightly faster framerate**

**# Select camera type (if user enters --usbcam when calling this script,**

**# a USB webcam will be used)**

**camera\_type = 'picamera'**

**parser = argparse.ArgumentParser()**

**parser.add\_argument('--usbcam', help='Use a USB webcam instead of picamera',**

**action='store\_true')**

**args = parser.parse\_args()**

**if args.usbcam:**

**camera\_type = 'usb'**

**# This is needed since the working directory is the object\_detection folder.**

**sys.path.append('..')**

**# Import utilites**

**from utils import label\_map\_util**

**from utils import visualization\_utils as vis\_util**

**# Name of the directory containing the object detection module we're using**

**MODEL\_NAME = 'garbage\_model'**

**# Grab path to current working directory**

**CWD\_PATH = os.getcwd()**

**# Path to frozen detection graph .pb file, which contains the model that is used**

**# for object detection.**

**PATH\_TO\_CKPT = os.path.join(CWD\_PATH,MODEL\_NAME,'frozen\_inference\_graph.pb')**

**# Path to label map file**

**PATH\_TO\_LABELS = os.path.join(CWD\_PATH,'data','labelmap.pbtxt')**

**# Number of classes the object detector can identify**

**NUM\_CLASSES = 4**

**## Load the label map.**

**# Label maps map indices to category names, so that when the convolution**

**# network predicts `5`, we know that this corresponds to `airplane`.**

**# Here we use internal utility functions, but anything that returns a**

**# dictionary mapping integers to appropriate string labels would be fine**

**label\_map = label\_map\_util.load\_labelmap(PATH\_TO\_LABELS)**

**categories = label\_map\_util.convert\_label\_map\_to\_categories(label\_map, max\_num\_classes=NUM\_CLASSES, use\_display\_name=True)**

**category\_index = label\_map\_util.create\_category\_index(categories)**

**# Load the Tensorflow model into memory.**

**detection\_graph = tf.Graph()**

**with detection\_graph.as\_default():**

**od\_graph\_def = tf.GraphDef()**

**with tf.gfile.GFile(PATH\_TO\_CKPT, 'rb') as fid:**

**serialized\_graph = fid.read()**

**od\_graph\_def.ParseFromString(serialized\_graph)**

**tf.import\_graph\_def(od\_graph\_def, name='')**

**sess = tf.Session(graph=detection\_graph)**

**# Define input and output tensors (i.e. data) for the object detection classifier**

**# Input tensor is the image**

**image\_tensor = detection\_graph.get\_tensor\_by\_name('image\_tensor:0')**

**# Output tensors are the detection boxes, scores, and classes**

**# Each box represents a part of the image where a particular object was detected**

**detection\_boxes = detection\_graph.get\_tensor\_by\_name('detection\_boxes:0')**

**# Each score represents level of confidence for each of the objects.**

**# The score is shown on the result image, together with the class label.**

**detection\_scores = detection\_graph.get\_tensor\_by\_name('detection\_scores:0')**

**detection\_classes = detection\_graph.get\_tensor\_by\_name('detection\_classes:0')**

**# Number of objects detected**

**num\_detections = detection\_graph.get\_tensor\_by\_name('num\_detections:0')**

**# Initialize frame rate calculation**

**frame\_rate\_calc = 1**

**freq = cv2.getTickFrequency()**

**font = cv2.FONT\_HERSHEY\_SIMPLEX**

**# Initialize camera and perform object detection.**

**# The camera has to be set up and used differently depending on if it's a**

**# Picamera or USB webcam.**

**### Picamera ###**

**if camera\_type == 'picamera':**

**# Initialize Picamera and grab reference to the raw capture**

**camera = PiCamera()**

**camera.resolution = (IM\_WIDTH,IM\_HEIGHT)**

**camera.framerate = 10**

**rawCapture = PiRGBArray(camera, size=(IM\_WIDTH,IM\_HEIGHT))**

**rawCapture.truncate(0)**

**for frame1 in camera.capture\_continuous(rawCapture, format="bgr",use\_video\_port=True):**

**t1 = cv2.getTickCount()**

**# Acquire frame and expand frame dimensions to have shape: [1, None, None, 3]**

**# i.e. a single-column array, where each item in the column has the pixel RGB value**

**frame = np.copy(frame1.array)**

**frame.setflags(write=1)**

**frame\_expanded = np.expand\_dims(frame, axis=0)**

**# Perform the actual detection by running the model with the image as input**

**(boxes, scores, classes, num) = sess.run(**

**[detection\_boxes, detection\_scores, detection\_classes, num\_detections],**

**feed\_dict={image\_tensor: frame\_expanded})**

**# Draw the results of the detection (aka 'visulaize the results')**

**vis\_util.visualize\_boxes\_and\_labels\_on\_image\_array(**

**frame,**

**np.squeeze(boxes),**

**np.squeeze(classes).astype(np.int32),**

**np.squeeze(scores),**

**category\_index,**

**use\_normalized\_coordinates=True,**

**line\_thickness=2,**

**min\_score\_thresh=0.80)**

**cv2.putText(frame,"FPS: {0:.2f}".format(frame\_rate\_calc),(30,50),font,1,(255,255,0),2,cv2.LINE\_AA)**

**# All the results have been drawn on the frame, so it's time to display it.**

**cv2.imshow('Object detector', frame)**

**t2 = cv2.getTickCount()**

**time1 = (t2-t1)/freq**

**frame\_rate\_calc = 1/time1**

**#Conditions to detect different class type and gives**

**#output from individual GPIO pins for each type**

**#glass**

**if(int(classes[0][0]) == 1):**

**count\_glass+=1**

**if(count\_glass>=5):**

**GPIO.output(18, GPIO.HIGH)**

**time.sleep(2)**

**count\_glass=0**

**GPIO.output(18, GPIO.LOW)**

**#metal**

**if(int(classes[0][0]) == 2):**

**count\_metal+=1**

**if(count\_metal>=5):**

**GPIO.output(23, GPIO.HIGH)**

**time.sleep(2)**

**count\_metal=0**

**GPIO.output(23, GPIO.LOW)**

**#paper**

**if(int(classes[0][0]) == 3):**

**count\_paper+=1**

**if(count\_paper>=5):**

**GPIO.output(24, GPIO.HIGH)**

**time.sleep(2)**

**count\_paper=0**

**GPIO.output(24, GPIO.LOW)**

**#plastic**

**if(int(classes[0][0]) == 4):**

**count\_plastic+=1**

**if(count\_plastic>=5):**

**GPIO.output(25, GPIO.HIGH)**

**time.sleep(2)**

**count\_plastic=0**

**GPIO.output(25, GPIO.LOW)**

**# Press 'q' to quit**

**if cv2.waitKey(1) == ord('q'):**

**break**

**rawCapture.truncate(0)**

**GPIO.cleanup()**

**camera.close()**

**### USB webcam ###**

**elif camera\_type == 'usb':**

**# Initialize USB webcam feed**

**camera = cv2.VideoCapture(0)**

**ret = camera.set(3,IM\_WIDTH)**

**ret = camera.set(4,IM\_HEIGHT)**

**while(True):**

**t1 = cv2.getTickCount()**

**# Acquire frame and expand frame dimensions to have shape: [1, None, None, 3]**

**# i.e. a single-column array, where each item in the column has the pixel RGB value**

**ret, frame = camera.read()**

**frame\_expanded = np.expand\_dims(frame, axis=0)**

**# Perform the actual detection by running the model with the image as input**

**(boxes, scores, classes, num) = sess.run(**

**[detection\_boxes, detection\_scores, detection\_classes, num\_detections],**

**feed\_dict={image\_tensor: frame\_expanded})**

**# Draw the results of the detection (aka 'visulaize the results')**

**vis\_util.visualize\_boxes\_and\_labels\_on\_image\_array(**

**frame,**

**np.squeeze(boxes),**

**np.squeeze(classes).astype(np.int32),**

**np.squeeze(scores),**

**category\_index,**

**use\_normalized\_coordinates=True,**

**line\_thickness=8,**

**min\_score\_thresh=0.85)**

**cv2.putText(frame,"FPS: {0:.2f}".format(frame\_rate\_calc),(30,50),font,1,(255,255,0),2,cv2.LINE\_AA)**

**# All the results have been drawn on the frame, so it's time to display it.**

**cv2.imshow('Object detector', frame)**

**t2 = cv2.getTickCount()**

**time1 = (t2-t1)/freq**

**frame\_rate\_calc = 1/time1**

**# Press 'q' to quit**

**if cv2.waitKey(1) == ord('q'):**

**break**

**camera.release()**

**cv2.destroyAllWindows()**

**CONCLUSION**

This system isolates waste automatically utilizing no sensors, however the energy of machine figuring out how to perceive as to which waste can be arranged as degradable or non-degradable. As the system works independently, there is no need of human mediation to control or to do any assignment. The proposed system concentrates on identification, classification and segregation of waste. Utilizing Raspberry Pi, the characterization result will be given to the equipment part of the framework with the goal that it will be dumped in its separate containers. The system will order waste automatic categorizing them as Plastic, Paper, Glass and Metal.

**REFERENCE**

* <https://github.com/tensorflow/models/tree/master/research/object_detection>
* <https://github.com/topics/object-detection>
* <https://github.com/bourdakos1/Custom-Object-Detection>
* <https://www.tensorflow.org/lite/models/object_detection/overview>
* <https://tensorflow-object-detection-api-tutorial.readthedocs.io/en/latest/training.html>
* <https://github.com/EdjeElectronics/TensorFlow-Object-Detection-on-the-Raspberry-Pi>
* <https://www.pyimagesearch.com/2017/10/16/raspberry-pi-deep-learning-object-detection-with-opencv/>