SINGULARITY

DEPARTMENT OF PHYSICS AND ASTROPHYSICS

2018-2019

Waste Segregation System

Submitted By:-

Shivani Dhasmana
Shivangi Pandey
Narendra Kumar
Nikita Singh
Manika Nagpal
Gaurav

ACKNOWLEDGEMENT

The Singularity group acknowledges the deep sense of gratitude to all the faculty members Prof. Amitabh Mukherjee, Prof. Vinay Gupta, Prof. Nivedita Deo, and Prof. K. Sreenivas for their spirited encouragement, healthy criticism and inspiring discussions throughout session January 2019 to May 2019. We are also fortunate to have a company of all the motivated students of electronics lab 2018-2019, without their help and cooperation it would have been difficult for us to enjoy the academic environment of the lab. We are also thankful to our team for their cooperative behaviour and their valuable efforts in the group. At the end, the Singularity, is also grateful to the lab staff for the constant help during the course of work.

ABSTRACT

In the last decade the user base of machine learning has grown dramatically. From a relatively small circle in computer science, engineering, and mathematics departments the users of machine learning now include students and researchers from every corner of the academic universe, as well as members of industry data scientists, entrepreneurs, and machine learning enthusiasts.

In this project, an attempt has been made to solve the problem of waste segregation using Machine Learning and Deep Learning algorithms.

CONTENTS

INTRODUCTION

MACHINE LEARNING

Machine Learning (ML) can be explained as automating and improving the learning process of computers based on their experiences without being actually programmed i.e. without any human assistance. The process starts with feeding good quality data and then training our machines(computers) by building machine learning models using the data and different algorithms. The choice of algorithms depends on what type of data do we have and what kind of task we are trying to automate.

We may use machine learning everyday and perhaps may have no idea that they are driven by Machine learning. Few Machine learning examples are virtual personal assistants like Siri, Alexa, social media services like smart reply in emails, Youtube watch next, Email spam and malware filtering, People you may know in Facebook, Product recommendation, Online fraud detection and many more.

A more formal and most widely used definition of machine learning is that of Carnegie Mellon University Professor Tom Mitchell:

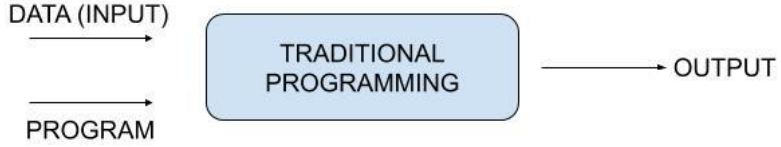
A computer program is said to learn from experience **E** with respect to some class of tasks **T** and performance measure **P**, if its performance at tasks in **T**, as measured by **P**, improves with experience **E.**

EXAMPLE: The game of playing checkers

E = the experience of playing many games of checkers
T = the task of playing checkers.
P = the probability that the program will win the next game

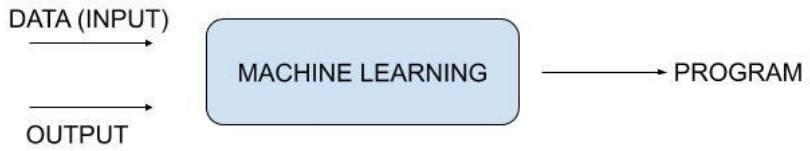
Basic Difference Between ML and Traditional Programming

- **Traditional Programming :** In traditional programming, we feed in the data and make a program to operate upon it to get the output.



Machine Learning : In machine learning, we feed in the trained data i.e. (Data as well as the output associated with each data point), develop an

algorithm to teach the system the relationship between data and output to enable it to develop its own program.



HOW MACHINE LEARNING WORKS?

- Gathering the data and Pre-processing :

  First step is gathering past data in any form suitable for processing. The better the quality and amount of data, the more suitable it will be for modelling.

  Sometimes, the data collected is in the raw form and it needs to be pre-processed, so some data processing needs to be done.
  Example: Some tuples may have missing values for certain attributes, an, in this case, it has to be filled with suitable values in order to perform machine learning or any form of data mining.
  Missing values for numerical attributes such as the price of the house may be replaced with the mean value of the attribute whereas missing values for categorical attributes may be replaced with the attribute with the highest mode. This invariably depends on the types of filters we use. If data is in the form of text or images then converting it to numerical form will be required, be it a list or array or matrix. Simply, Data is to be made relevant and consistent. It is to be converted into a format understandable by the machine.

- Dividing the data :

  The input data then is divided into training, cross-validation and test sets. The ratio between the respective sets must be 6:2:2

- Building model from the training set

  Suitable algorithms and techniques are applied on the on the training set to build a model.

- Validation and testing
  The model is then tested on the validation set to validate its working. If it turns out to be fine, then the conceptualized model is tested with data which was not fed at the time of training and evaluating its performance using metrics such as F1 score, precision and recall.

TYPES OF MACHINE LEARNING

Supervised and Unsupervised learning

Supervised Learning

Supervised learning as the name indicates the presence of a supervisor as a teacher. Basically, supervised learning is a learning in which we teach or train the machine using data which is well labelled that means the data is already tagged with the correct answer. After that, the machine is provided with a new set of examples(data) so that supervised learning algorithm analyses the training data (set of training examples) and produces a correct outcome from labelled data.

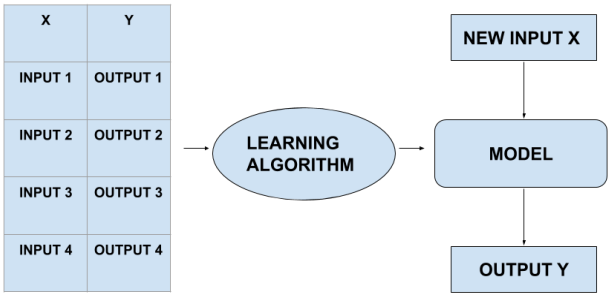
EXAMPLE

For instance, suppose you are given a basket filled with different kinds of fruits. Now the first step is to train the machine with all different fruits one by one like this:

If shape of object is rounded and depression at top having color Red then it will be labelled as –**Apple**.
If shape of object is long curving cylinder having color Green-Yellow then it will be labelled as –**Banana**.
Now suppose after training the data, you have given a new separate fruit say Banana from basket and asked to identify it.

Since the machine has already learned the things from previous data and this time have to use it wisely. It will first classify the fruit with its shape and color and would confirm the fruit name as BANANA and put it in Banana category. Thus the machine learns the things from training data(basket containing fruits) and then apply the knowledge to test data(new fruit).

Supervised learning is where you have input variables (x) and an output variable (Y) and you use an algorithm to learn the mapping function from the input to the output **Y = f(X)**. The goal is to approximate the mapping function so well that when you have new input data (x) that you can predict the output variables (Y) for that data



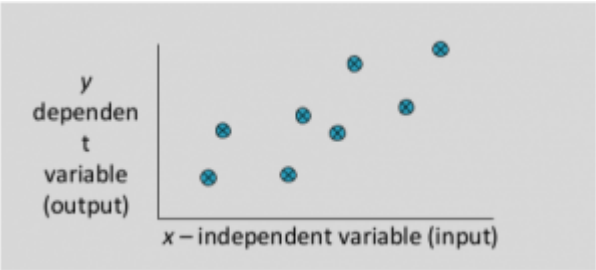
.

Supervised learning can further be classified into two categories of algorithms: Classification and Regression.

- ***Classification***
- A classification problem is when the output variable is a category, such as "Red" or "blue" or "disease" and "no disease". A classification model attempts to draw some conclusion from observed values. Given one or more inputs a classification model will try to predict the value of one or more outcomes.
  For example, when filtering emails "spam" or "not spam", when looking at transaction data, "fraudulent", or "authorized". In short Classification either predicts categorical class labels or classifies data (construct a model) based on the training set and the values (class labels) in classifying attributes and uses it in classifying new data. There are a number of classification models. Classification models include logistic regression, decision tree, random forest, gradient-boosted tree, multilayer perceptron, one-vs-rest, and Naive Bayes.

Regression

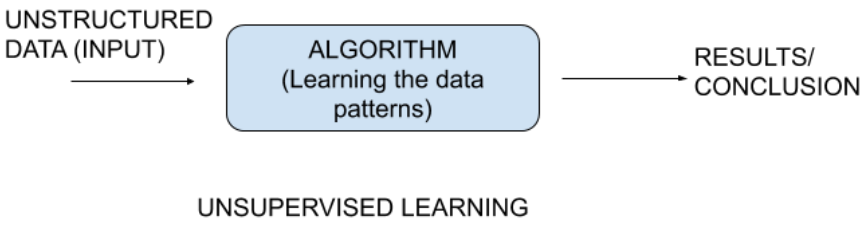
- A regression problem is when the output variable is a real or continuous value, such as "salary" or "weight". Many different models can be used, the simplest is the linear regression. It tries to fit data with the best hyper-plane which goes through the points.



Both problems have as goal the construction of a succinct model that can predict the value of the dependent attribute from the attribute variables. The difference between the two tasks is the fact that the dependent attribute is numerical for regression and categorical for classification.

Unsupervised Learning

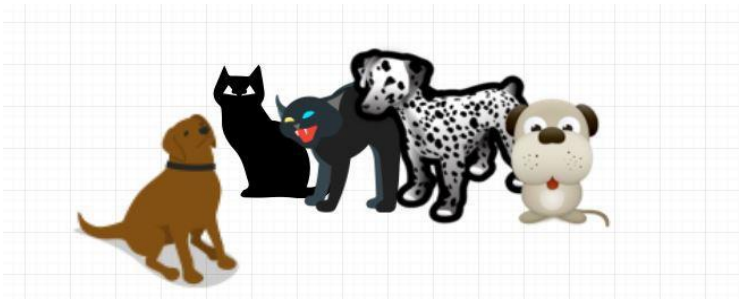
Unsupervised learning is the training of machine using information that is neither classified nor labeled and allowing the algorithm to act on that information without guidance. Here the task of machine is to group unsorted information according to similarities, patterns and differences without any prior training of data.



UNSUPERVISED LEARNING

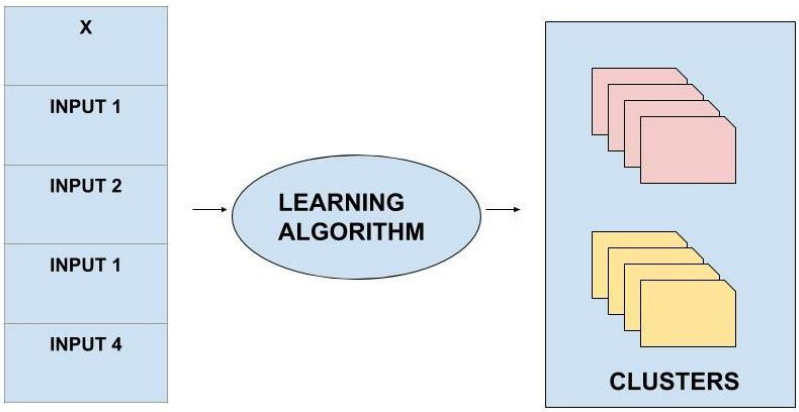
Unlike supervised learning, no teacher is provided. That means no training will be given to the machine. Therefore, machine is restricted to find the hidden structure in unlabeled data by our-self.

For instance, suppose it is given an image having both dogs and cats which have not seen ever. Thus the machine has no idea about the features of dogs and cat so we can't categorize it in dogs and cats. But it can categorize them according to their similarities, patterns, and differences i.e., we can easily categorize the above picture into two parts. First may contain all pics having **dogs** in it and second part may contain all pics having **cats** in it. Here you didn't learn anything before, means no training data or examples.



Unsupervised learning classified into two categories of algorithms:

- ***Clustering***
- A clustering problem is where you want to discover the inherent groupings in the data, such as grouping customers by purchasing behaviour. Broadly this technique is applied to group data based on different patterns, our machine model finds. For example in above figure we are not given output parameter value, so this technique will be used to group clients based on the input parameters provided by our data.



- ***Association***

- This technique is a rule based ML technique which finds out some very useful relations between parameters of a large data set. For e.g. shopping stores use algorithms based on this technique to find out relationship between sale of one product w.r.t to others sale based on customer behavior. Once trained well, such models can be used to increase their sales by planning different offers. An association rule learning problem is where you want to discover rules that describe large portions of your data, such as people that buy X also tend to buy Y.

IMAGE FUNDAMENTALS

In order to work with images for object detection one must know the fundamentals of it, What image is? What it is made of?

An image is a single picture that represents something. It may be picture of a person, people, or animals, an outdoor scene, a microphotograph of an electronic component, or the result of medical imaging.

PIXELS: BUILDING BLOCKS OF IMAGES

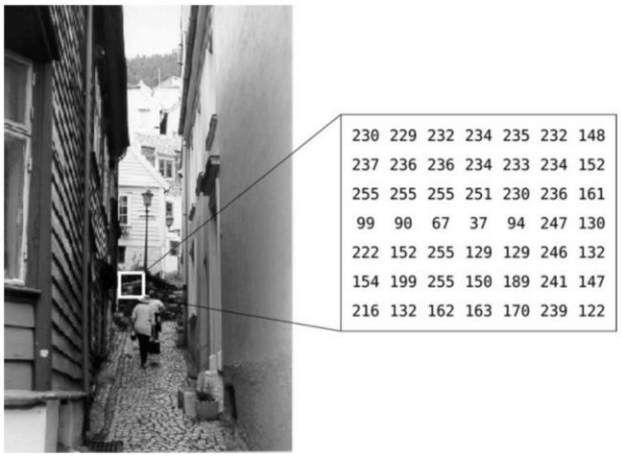
In digital image, a pixel, or picture element is a physical point in raster image, or the smallest addressable element in an all points addressable display device; so, it is the smallest controllable element of a picture represent on screen.

Each pixel is sample of an original image. The intensity of each pixel is variable in color imaging system, a color is typically represented by three or four component intensities such as red, green, and blue.

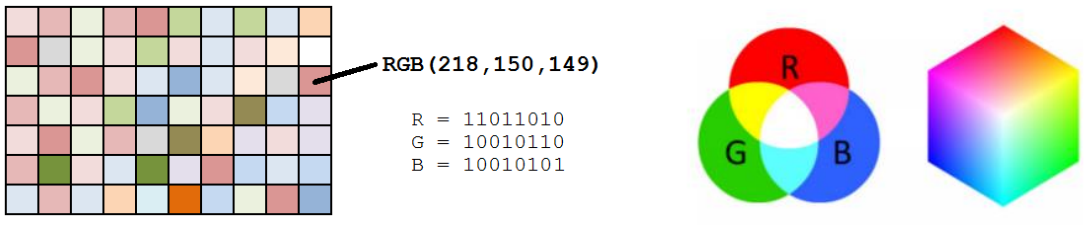
Pixels can be represented in two ways;

1. Grayscale: Each pixel is shade of gray, normally from 0(black) to 255(white). This range means that each pixel can be represented by eight bits, or exactly one byte. This is a very natural range for image file handling. Other grayscale range are used.

   But generally they are power of 2. Such image in medicine(X-Rays), images of printed works, and indeed different gray level is sufficient for recognition of most natural objects.



2. True color or RGB: Here each pixel has a particular color; that color being described by the amount of red, green, and blue in it. If each of these components has a range 0-255, this gives a total 255^3=16,777,216 different possible colors in the image. This is enough colors for any image. Since the total no of bits required for each pixel is 24, such images are also called 24-bit color images.



On these pixel matrices, the technique of convolutional neural network is implemented.

DEEP LEARNING

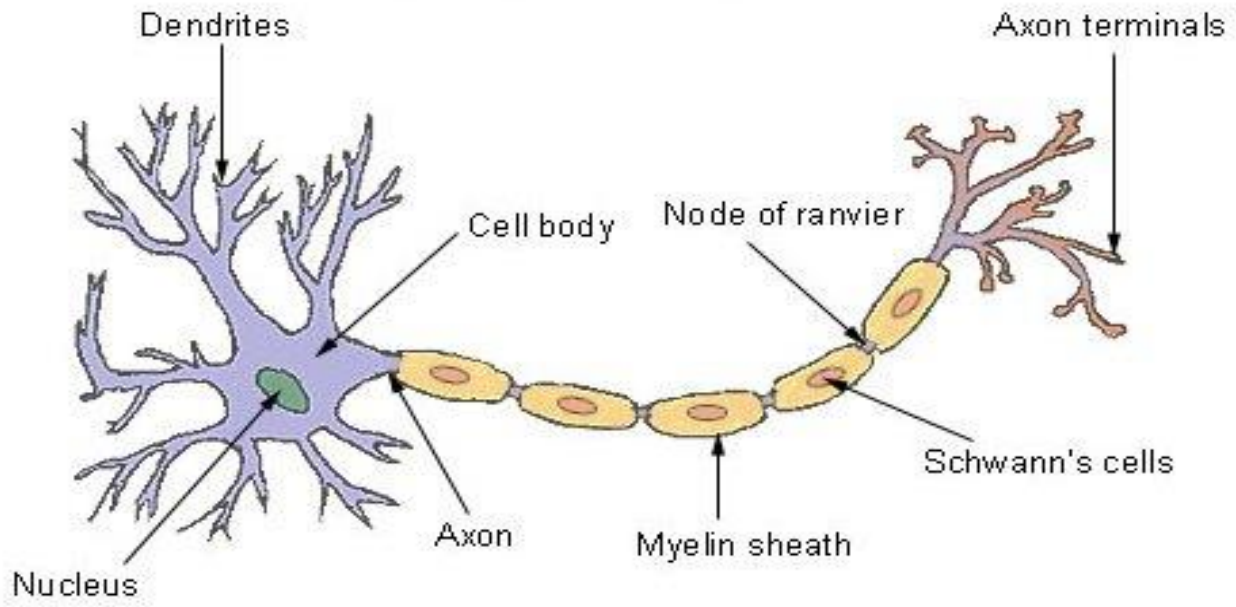
Deep learning (also known as deep structured learning or hierarchical learning) is part of a broader family of machine learning methods based on artificial neural networks. Learning can be supervised, semi-supervised or unsupervised just like in Machine Learning.

Deep Learning (DL) and Neural Network (NN) is currently driving some of the most ingenious inventions in today's century. Their incredible ability to learn from data and environment makes them the first choice of machine learning scientists.

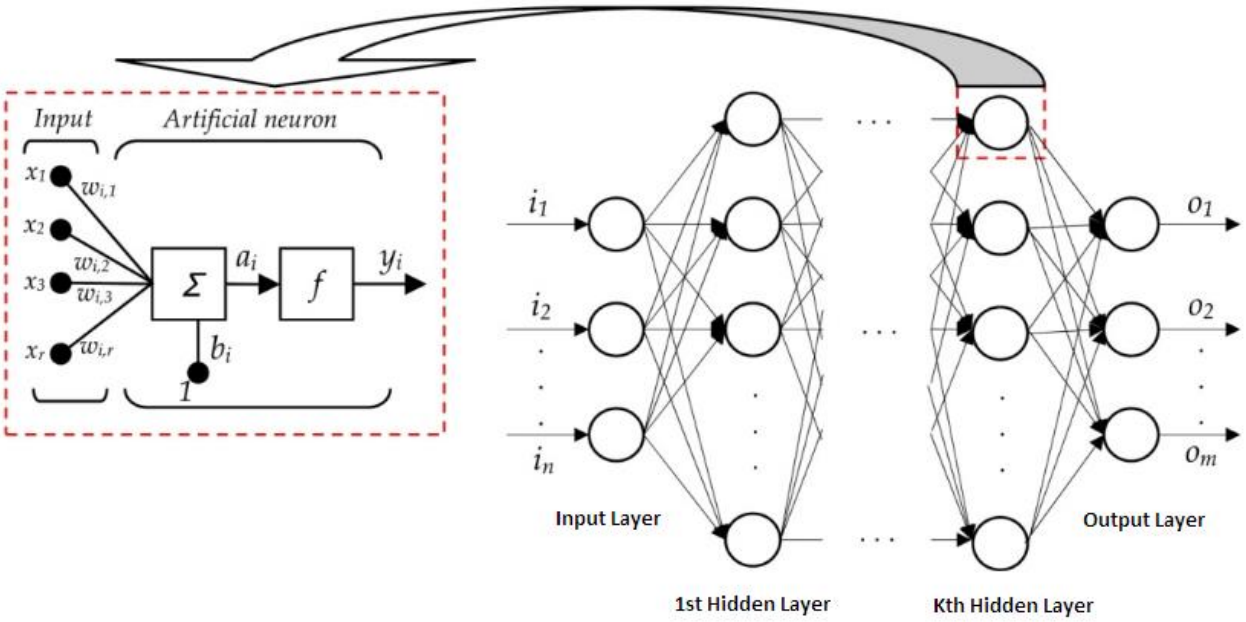
Deep Learning and Neural Network lies in the heart of products such as self driving cars, image recognition software, recommender systems etc. Evidently, being a powerful algorithm, it is highly adaptive to various data types as well.

NEURAL NETWORKS

Neural network is inspired by Human neuron. In human nervous system each neuron takes input from numerous other neurons through the dendrites. It then performs the required processing on the input and sends another electrical pulse through the axiom into the terminal nodes from where it is transmitted to numerous other neurons.



ANN works in a very similar fashion. The general structure of a neural network looks like:



This figure depicts a typical neural network with working of a single neuron explained separately.

The input to each neuron are like the dendrites. Just like in human nervous system, a neuron (artificial though!) collates all the inputs and performs an operation on them. Lastly, it transmits the output to all other neurons (of the next layer) to which it is connected. Neural Network is divided into layer of 3 types:

1. **Input Layer:** The training observations are fed through these neurons
2. **Hidden Layers:** These are the intermediate layers between input and output which help the Neural Network learn the complicated relationships involved in data.
3. **Output Layer:** The final output is extracted from previous two layers. For Example: In case of a classification problem with 5 classes, the output later will have 5 neurons.

So, the neural network is an approximation function in which the network tries to learn the parameters (weights) in hidden layers which when multiplied with the input gives you a predicted output close to the desired output.

There are various methods designed to apply deep learning. Each proposed method has a specific use case like the kind of data you have, whether it is supervised or unsupervised learning you would like to apply, what type of task you would want to solve with the data. So depending on these factors, you choose one of the methods that can best solve your problem.

Some of the deep learning methods are:

- Convolutional Neural Network,
- Recurrent Neural Network,
- Long short-term memory.

CONVOLUTIONAL NEURAL NETWORK (CNN)

A neural network consists of several different layers such as the **input** layer, at least one **hidden** layer, and an **output** layer. They are best used in object detection for recognizing **patterns** such as edges (vertical/horizontal), shapes, colours, and textures. The hidden layers are convolutional layers in this type of neural network which acts like a **filter** that first receives **input**, transforms it using a specific pattern/feature, and sends it to the next layer. With more convolutional layers, each time a new input is sent to the next convolutional layer, they are changed in different ways. For example, in the first convolutional layer, the filter may identify shape/colour in a region (i.e. brown), and the next one may be able to conclude the object it really is (i.e. an ear or paw), and the last convolutional layer may classify the object as a dog. Basically, as more and more layers the input goes through, the more sophisticated patterns the future ones can detect.

Convolutional implementation of sliding windows

Before we discuss the implementation of the sliding window using convents, let's analyze how we can convert the fully connected layers of the network into convolutional layers. Fig. 4 shows a simple convolutional network with two fully connected layers each of shape (400,).

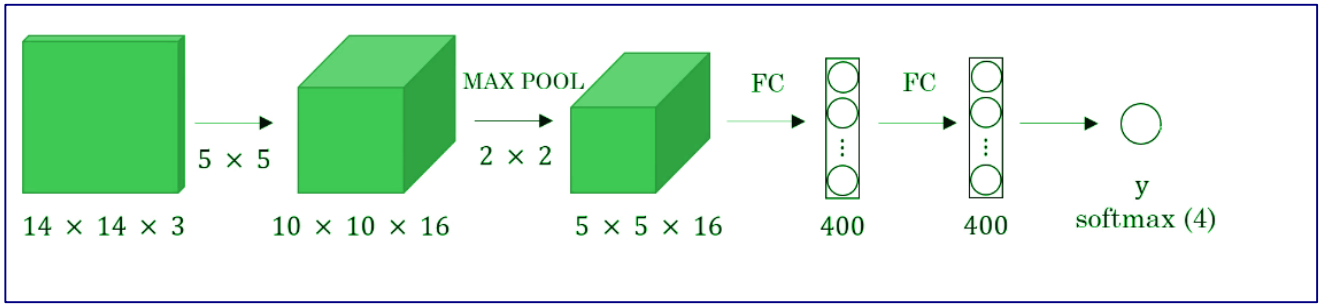


Fig 4. Sliding windows mechanism

A fully connected layer can be converted to a convolutional layer with the help of a **1D convolutional layer**. The width and height of this layer are equal to one and the number of filters are equal to the shape of the fully connected layer. An example of this is shown in Fig 5.

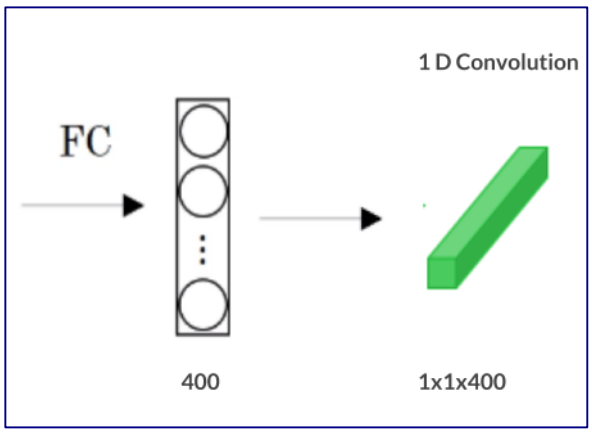


Fig 5. Converting a fully connected layer into a convolutional layer

We can apply this concept of conversion of a fully connected layer into a convolutional layer to the model by replacing the fully connected layer with a 1-D convolutional layer. The number of the filters of the 1D convolutional layer is equal to the shape of the fully connected layer. This representation is shown in Fig 6. Also, the output softmax layer is also a convolutional layer of shape (1, 1, 4), where 4 is the number of classes to predict.

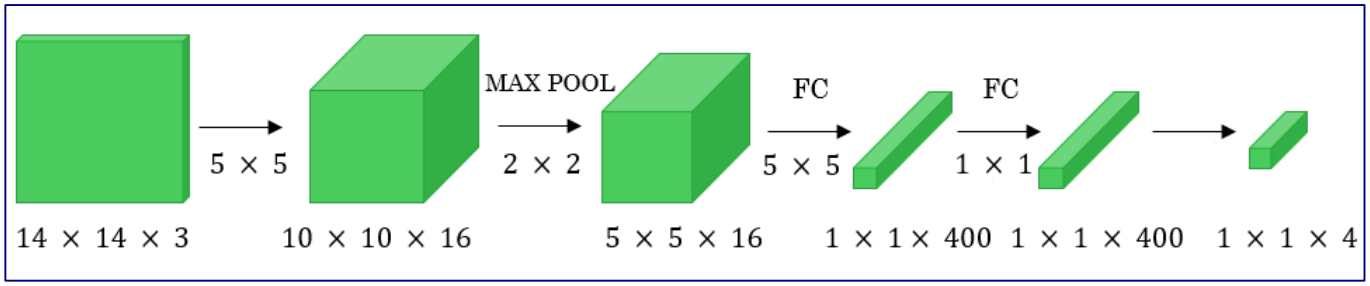
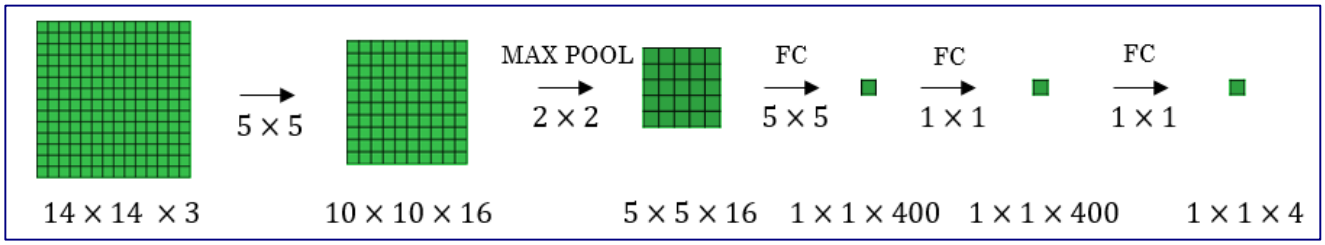


Fig 6. Convolutional representation of fully connected layers.

Now, let's extend the above approach to implement a convolutional version of sliding window. First, let's consider the ConvNet that we have trained to be in the following representation (no fully connected layers).



Let's assume the size of the input image to be **16 × 16 × 3**. If we're to use a sliding window approach, then we would have passed this image to the above ConvNet four times, where each time the sliding window crops a part of the input image of size **14 × 14 × 3** and pass it through the ConvNet. But instead of this, we feed the full image (with shape **16 × 16 × 3**) directly into the trained ConvNet (see Fig. 7). This results in an output matrix of shape **2 × 2 × 4**. Each cell in the output matrix represents the result of a possible crop and the classified value of the cropped image. For example, the left cell of the output (the green one) in Fig. 7 represents

the result of the first sliding window. The other cells represent the results of the remaining sliding window operations.

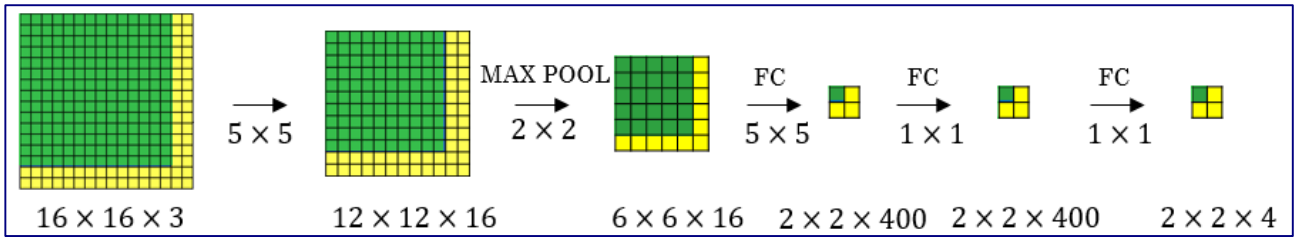


Fig 7. Convolutional implementation of the sliding window

Note that the stride of the sliding window is decided by the number of filters used in the Max Pool layer. In the example above, the Max Pool layer has two filters, and as a result, the sliding window moves with a stride of two resulting in four possible outputs. The main advantage of using this technique is that the sliding window runs and computes all values simultaneously. Consequently, this technique is really fast. Although a weakness of this technique is that the position of the bounding boxes is not very accurate.

OBJECT DETECTION

Object Detection is modeled as a classification problem where we take windows of fixed sizes from input image at all the possible locations feed these patches to an image classifier.

Each window is fed to the classifier which predicts the class of the object in the window (or background if none is present). Hence, we know both the class and location of the objects in the image. Idea is that we resize the image at multiple scales and we count on the fact that our chosen window size will completely contain the object in one of these resized images. Most commonly, the image is down sampled(size is reduced) until certain condition typically a minimum size is reached. On each of these images, a fixed size window detector is run. It's common to have as many as 64 levels on such pyramids. Now, all these windows are fed to a classifier to detect the object of interest. This will help us solve the problem of size and location.

In the next few sections, we will discuss the steps to implement Object Detection using Python.

Step-1 Configuring the development environment

In this section, we will explain the python modules needed to perform the task of Object Detection.

OpenCV

OpenCV is a cross-platform library using which we can develop real-time computer vision applications. It mainly focuses on image processing, video capture and analysis including features like face detection and object detection.

TensorFlow

It is an open-source machine learning library for research and production. TensorFlow offers APIs for beginners and experts to develop for desktop, mobile, web, and cloud. TensorFlow is a free and open-source software library for dataflow and differentiable programming across a range of tasks. It is a symbolic math library, and is also used for machine learning applications such as neural networks. The name TensorFlow derives from the operations that such neural networks perform on multidimensional data arrays, which are referred to as *tensors*.

NumPy

Array is a central data structure of the `numpy` library. The library's name is short for "Numeric Python" or "Numerical Python". NumPy is a Python library that is the core library for scientific computing in Python. It contains a collection of tools and techniques that can be used to solve on a computer mathematical models of problems in Science and Engineering. One of these tools is a high-performance multidimensional array object that is a powerful data

structure for efficient computation of arrays and matrices. To work with these arrays, there's a vast amount of high-level mathematical functions operate on these matrices and arrays.

Pillow

The most important class in the Python Imaging Library is the Image class, defined in the module with the same name. You can create instances of this class in several ways; either by loading images from files, processing other images, or creating images from scratch.

The Python Imaging Library supports a wide variety of image file formats. To read files from disk, use the open() function in the Image module. You don't have to know the file format to open a file. The library automatically determines the format based on the contents of the file.

To save a file, use the save() method of the Image class. When saving files, the name becomes important. Unless you specify the format, the library uses the filename extension to discover which file storage format to use.

The Python Imaging Library also allows you to work with the individual bands of an multi-band image, such as an RGB image. The split method creates a set of new images, each containing one band from the original multi-band image. The merge function takes a mode and a tuple of images, and combines them into a new image.

Matplotlib

This library is very flexible and has a lot of handy, built-in defaults that will help you out tremendously. As such, you don't need much to get started: you need to make the necessary imports, prepare some data, and you can start plotting with the help of the plot() function! When you're ready, don't forget to show your plot using the show() function.

ImageAI

It is a python library built to empower developers, reseachers and students to build applications and systems with self-contained Deep Learning and Computer Vision capabilities using simple and few lines of code. This documentation is provided to provide detailed insight into all the classes and functions available in ImageAI, coupled with a number of code examples.

Step-2 Training and Testing the dataset

Each data point in our training dataset consists of:

1. An image.

2. The label/category (i.e., dog, cat, panda, etc.) of the image

Again, it's important that each of these images have labels associated with them because our supervised learning algorithm will need to see these labels to "teach

itself" how to recognize each category. Keeping this in mind, let's go ahead and work through the four steps to constructing a deep learning model.

Step #1: Gather Dataset

The first step is to gather the initial dataset. We need the images as well as the labels associated with each image. These labels should come from a finite set of categories, such as: categories = dog, cat, panda, etc.

Furthermore, the number of images for each category should be approximately uniform (i.e. the same number of examples per category). If we have twice the number of cat images than dog images, and five times the number of panda images than cat images, then our classifier will become naturally biased to overfitting into these heavily-represented categories.

Class imbalance is a common problem in machine learning and there exist a number of ways to overcome it but the best method to avoid learning problems due to class imbalance is to simply avoid class imbalance entirely.

Step #2: Split Dataset

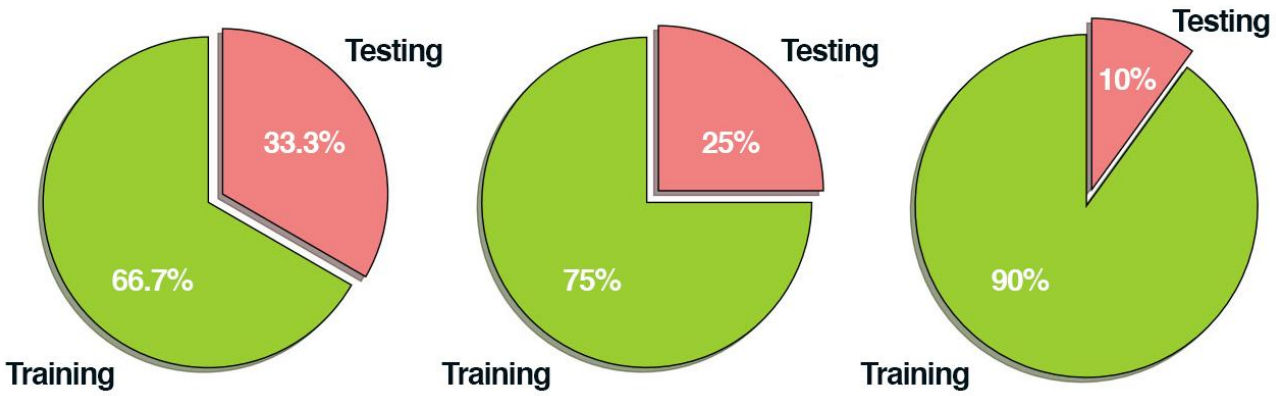
Now that we have our initial dataset, we need to split it into two parts:

1. A training set

2. A testing set

A training set is used by our classifier to "learn" what each category looks like by making predictions on the input data and then correct itself when predictions are wrong. After the classifier has been trained, we can evaluate the performing on a testing set.

It's extremely important that the training set and testing set are independent of each other and do not overlap! If you use your testing set as part of your training data, then your classifier has an unfair advantage since it has already seen the testing examples before and "learned" from them. Instead, you must keep this testing set entirely separate from your training process and use it only to evaluate your network.



Step #3: Training Network

Given our training set of images, we can now train our dataset. The goal here is for the network to learn how to recognize each of the categories in our labelled data. When the model makes a mistake, it learns from its mistake and improves itself.

Training the network also involved annotating each image with its category using LabelImg tool. This tool is a graphical image annotation tool. It is written in Python and uses Qt for its graphical interface. Annotations are saved as XML files in PASCAL VOC format, the format used by ImageNet. The simple offline interface makes the annotation process pretty fast, even though it does not support many hotkey shortcuts.

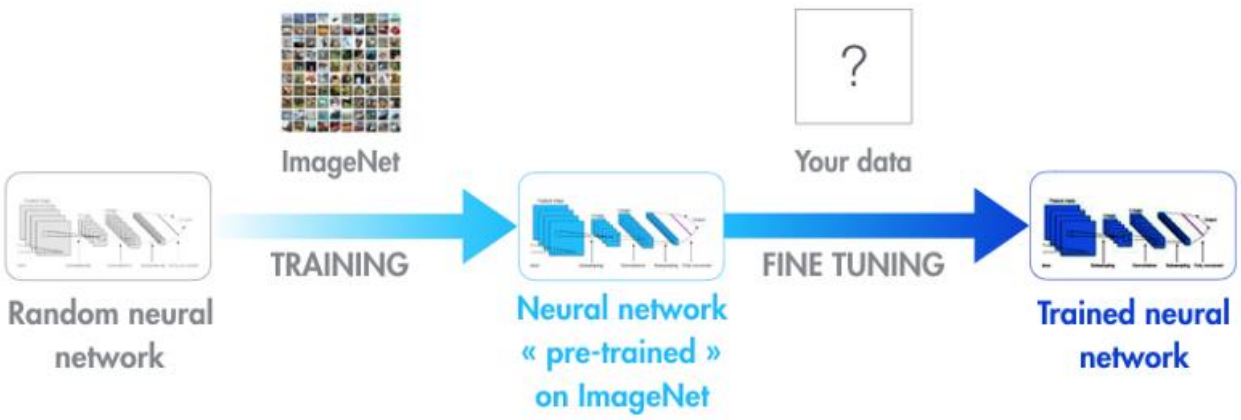
After that, we used transfer learning program to evaluate and train our dataset. Transfer learning makes use of the knowledge gained while solving one problem and applying it to a different but related problem. For example, knowledge gained while learning to recognize cars can be used to some extent to recognize trucks.

Pre-Training: When we train the network on a large dataset(for example: ImageNet) , we train all the parameters of the neural network and therefore the model is learned. It may take hours on your GPU.

Fine Tuning: We can give the new dataset to fine tune the pre-trained CNN. Consider that the new dataset is almost similar to the original dataset used for pre-training. Since the new dataset is similar, the same weights can be used for extracting the features from the new dataset.

If the new dataset is very small, it's better to train only the final layers of the network to avoid overfitting, keeping all other layers fixed. So remove the final layers of the pre-trained network. Add new layers. Retrain only the new layers.

If the new dataset is very much large, retrain the whole network with initial weights from the pretrained model.



The output we obtained from this transfer learning program (Appendix A) are a frozen inference graph and label map.

Frozen Inference Graph: If the model is trained and saved in the Keras API then you will probably have saved an hdf5 file of your model in which the network architecture and weights are saved together in one file. This file can be called and loaded back into the Keras API for inference. Sadly, however, this file type is not recognised by TensorFlow APIs and is also unnecessarily large to store, load in, and perform inference on.

Similarly, if one writes a model in the TensorFlow Python API, then the training procedure will save a TensorFlow graph, using Google's ProtoBuf library, and a series of checkpoint files. The graph stores the information about the architecture of the network with Variable ops, whereas the checkpoint files contain the values of the weights at various stages of training (depending on how regularly your session checkpoints during training). These can normally be loaded in for inference in a TensorFlow Python API session during which weights from the checkpoint files are inserted into the Variable ops in the graph. Yet, this is inefficient when just performing inference. A saved Keras .h5 file, on the other hand, is simply the graph and final state checkpoint file combined together, while still keeping the hyperparameters stored as Variable ops. (Note: a detailed understanding of the above is not necessary, I just add it for those wanting a more detailed explanation.)

Freezing the model means producing a singular file containing information about the graph and checkpoint variables, but saving these hyperparameters as constants within the graph structure. This eliminates additional information saved in the checkpoint files such as the gradients at each point, which are included so that the model can be reloaded and training continued from where you left off. As this is not needed when serving a model purely for inference they are discarded in freezing. A frozen model is a file of the Google .pb file type.

Label map: contains label of trained objects and corresponding index given to it. It was used to draw a bounding box for each object in each of the training and validation images.

Step #4: Testing the dataset

Lastly, we need to evaluate our trained network. For each of the images in our testing set, we present them to the network and ask it to predict what it thinks the label of the image is. We then tabulate the predictions of the model for an image in the testing set.

In this part, we are going to test our model and see if it does what we had hoped. In order to do this, we need to export the label map and inference graph.

.

Next whichever class has the highest value for particular sample, that sample belongs to the class with highest score. Then we compare with expected class and predicted class and generate the accuracy percentage.

We run the inference on all the input images one by one, which will provide us the output of images in which objects are detected with labels and the percentage/score of that object

We created a Tensorflow session in the initialization part of the class and running it on test images.
There are models in the Tensor Flow API you can use depending on your needs. If you want a high-speed model that can work on detecting video feed at high fps, the single shot detection(SSD) network works best. Some other object detection networks detect objects by sliding different sized boxes across the image and running the classifier many times on different sections of the image; this can be very resource consuming. As its name suggests, the SSD network determines all bounding box probabilities in one go; hence, it is a vastly faster model. However, with single shot detection, you gain speed at the cost of accuracy. We'll use single shot detection as the bounding box framework, but for the neural network architecture, we will use the MobileNet model, which is designed to be used in mobile applications.

Waste segregation problem

AIM

TO DESIGN A PROGRAM TO DISTINGUISH BETWEEN BIO-DEGRADABLE AND NON-BIODEGRADABLE ITEMS AND SEGREGATE WASTE IN A BIN ACCORDING TO THE RESPONSE.

THEORY

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 Convolutional Neural Network (CNN). 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 automatically and isolate it so the physical work will be diminished. It can be castoff in large scale industries for waste management purpose.

A. Software Unit

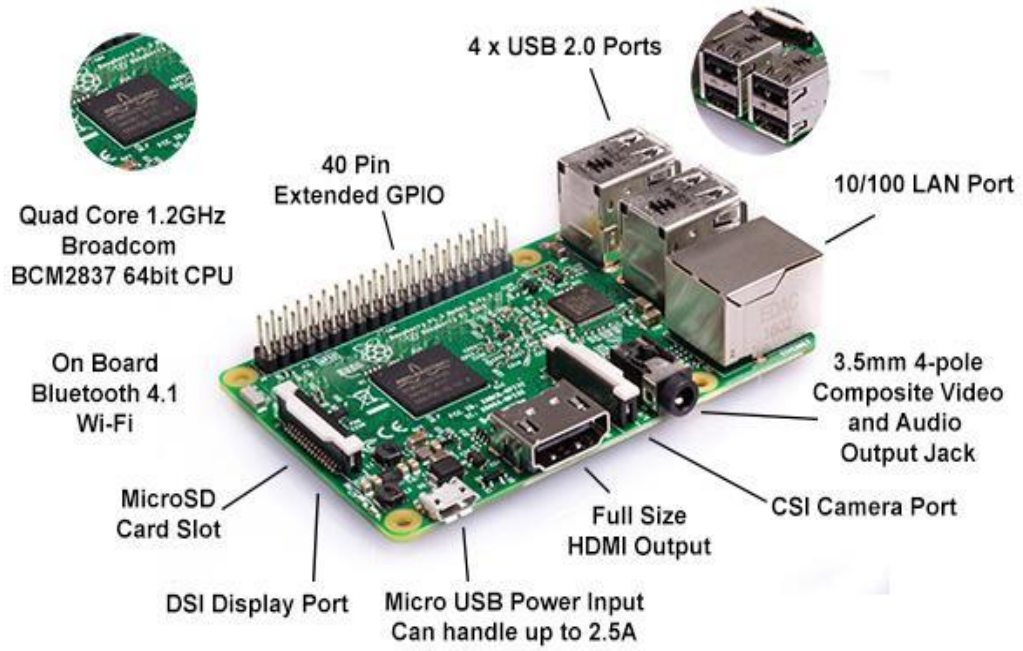
The first stage is image acquisition stage. It catches image from camera with the goal that it can be passed for handling and recognition of picture. After picture has been saved, different strategies for handling can be connected to the picture to perform a wide range of vision undertakings. After analyzing, 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 two categories namely biodegradable and non-biodegradable.

B. Hardware Unit

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. It is the backend process for classifying the images and to sort the waste autonomously.

Raspberry pi

The Raspberry Pi is a low cost, credit-card sized computer that plugs into a computer monitor or TV, and uses a standard keyboard and mouse. It is a capable little device that enables people of all ages to explore computing, and to learn how to program in languages like Scratch and Python. It's capable of doing everything you'd expect a desktop computer to do, from browsing the internet and playing high-definition video, to making spreadsheets, word-processing, and playing games.

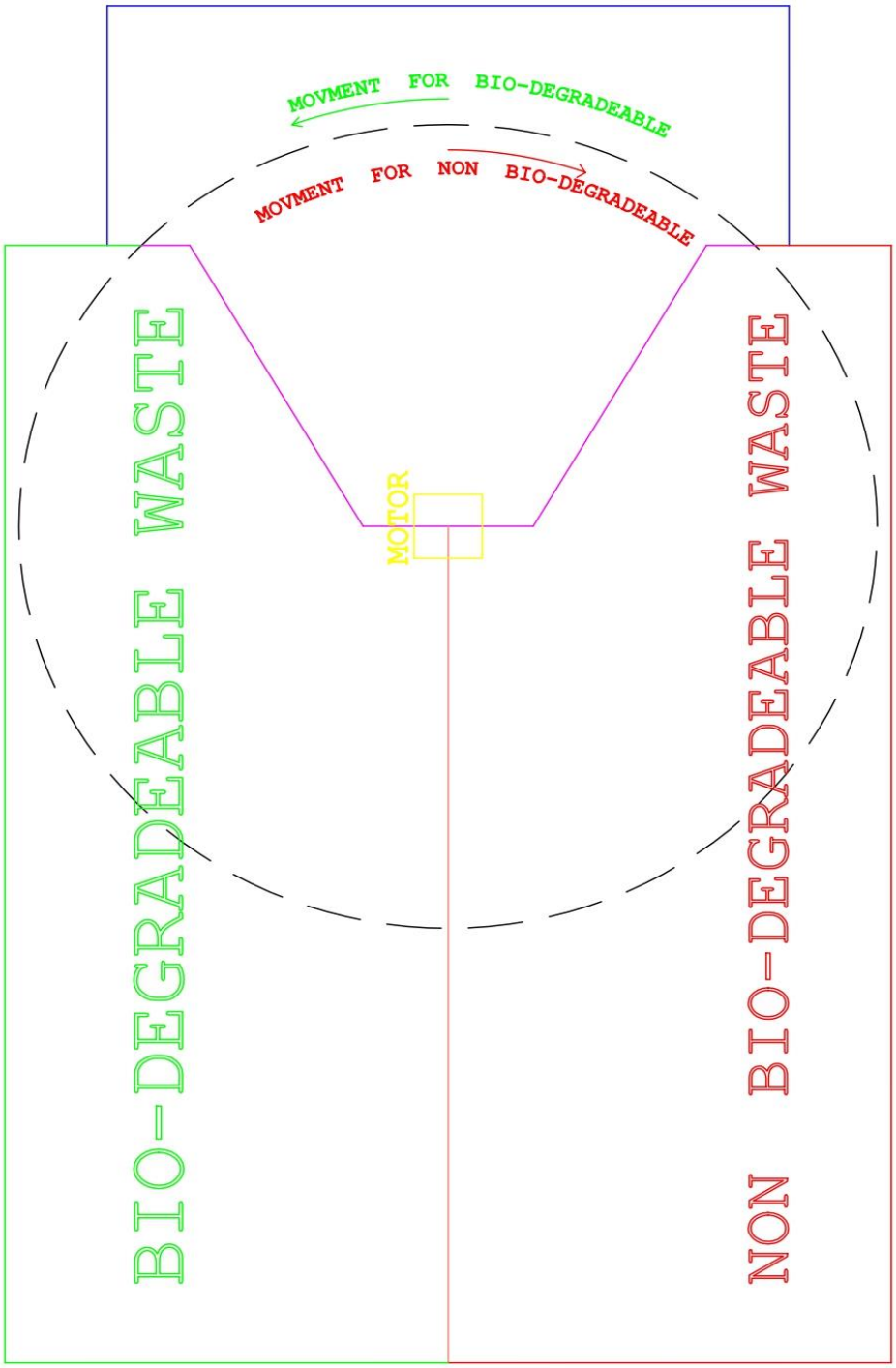


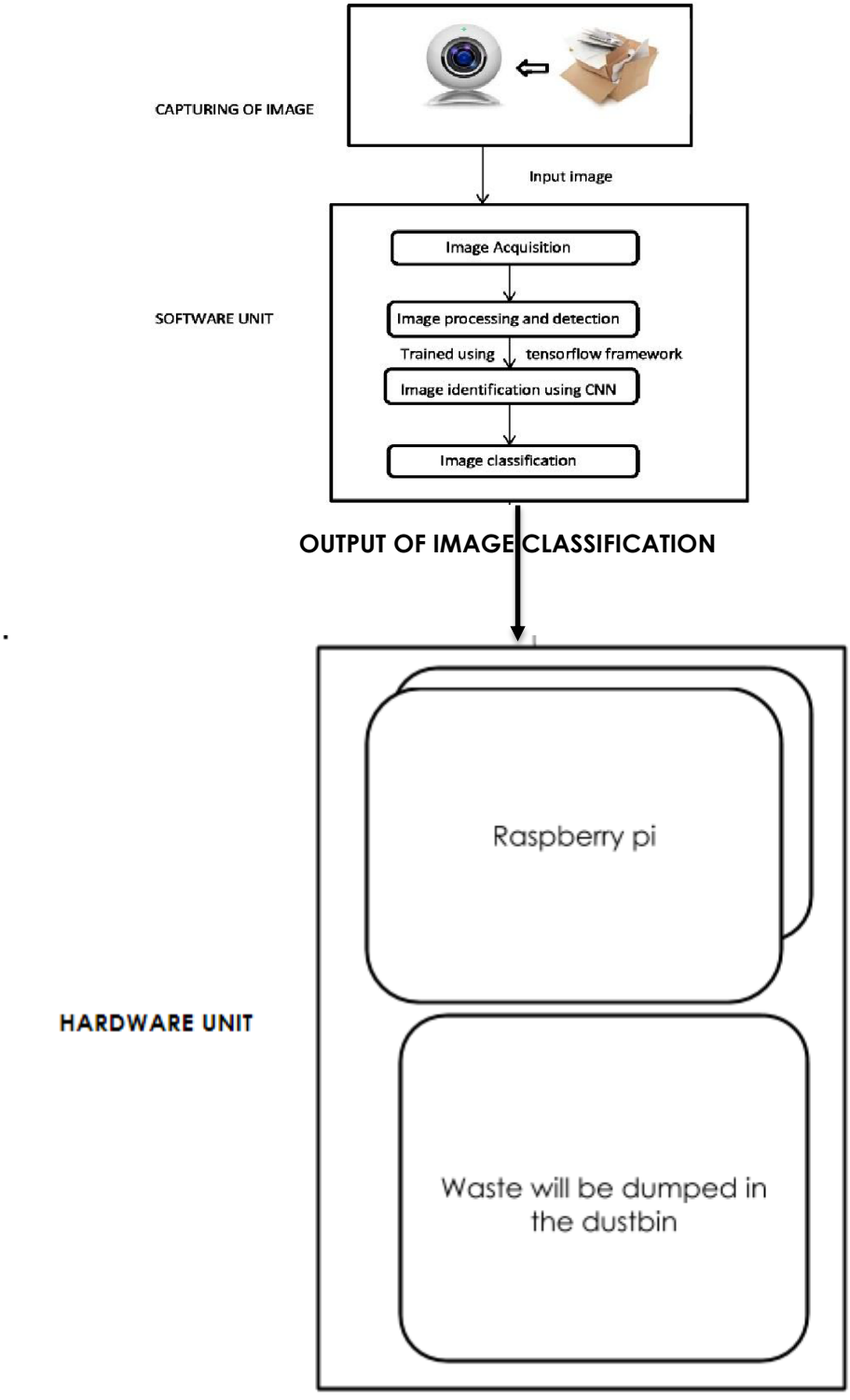
The raspberry pi board comprises a program memory (RAM), processor and graphics chip, CPU, GPU, Ethernet port, GPIO pins, Xbee socket, UART, power source connector. And various interfaces for other external devices. It also requires mass storage, for that we use an SD flash memory card.

It also provides a set of GPIO (general purpose input/output) pins that allow you to control electronic components for physical computing and explore the Internet of Things.

Design of the Waste Segregator bin

The bin is designed to specially to serve the purpose of waste segregation. There are 2 cabins : One for Biodegradable and other for Non-Biodegradable. Servo motor is connected to the rotating platform as shown in figure below.





Architecture diagram for Waste Segregation

We have taken use of the MSCOCO's Label Map and Frozen Inference Graph and Custom Object's Label Map and Frozen Inference Graph. These two are excellent object detection dataset with 80 classes, 80,000 training images and 40,000 validation images that are pre-stored in this and freezing a model means changing the variables (weights, biases, etc.) into constants. That means no more training is required and the variable has attained a constant value. The concept of freezing saves the required data alone, reducing the size of the model.

We run the program in the Raspberry pi version 2.2 by downloading and establishing all the libraries and finally configuring the environment.

WORKING

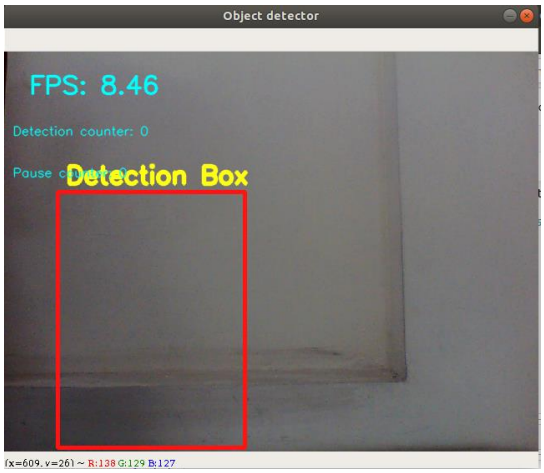
1. Run the program in Appendix A on Raspberry pi.

   Some features of the Program :

a) It will open a window named object detector which will take feed from the usbcam attached to raspberry pi.
b) It will detect any object (From Table 1 in Appendix) : Creates bounding box around it and displays its name along with accuracy.
    - Each frame from the video feed is passed on as an image to the tensorflow object detection session which returns name of the class detected and accuracy with which detection is made.
c) Program is designed as such that the window's resolution can be modified by tweaking the parameters. A small detection box is made within the entire frame whose coordinates can also be modified by the program.
    - If the object is inside the detection box , then only the condition (Biodegradable or Non-Biodegradable) is checked and with each passing frame the detection counter variable is incremented.
    - If detection counter variable exceeds 10 i.e. object has been detected inside the detection for 10 frames, then the object detection pauses for 30 frames for relevant action to take place.
    - Message will be displayed on the screen.
    - The program then prompts the servo motor attached to hardware to turn in the direction corresponding to side of the bin where the object is to be disposed according to its nature.

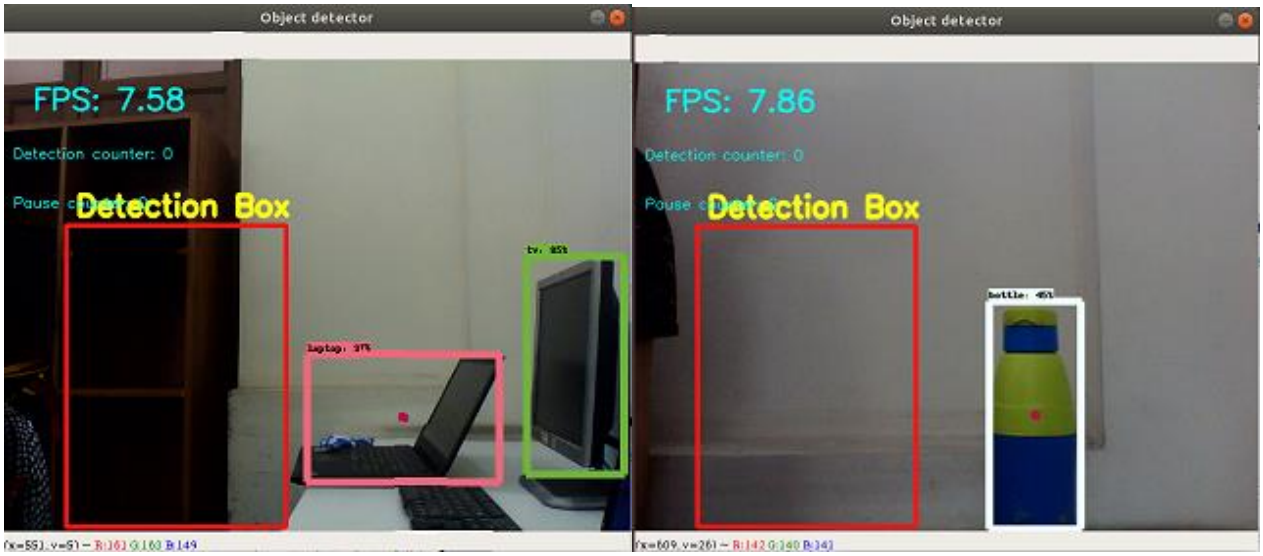
OBSERVATIONS

1. Object detector window, the detection box.



2. Detection of objects from live video feed.

Note detection counter and Pause counter are both zero as object is not detected inside the detection box.



3. Object detected as Non-Boidegradable when it is inside the detection box

Note after message is displayed the pause counter is non-zero implying the detection is paused till it is less than 30.

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 biodegradable or non-biodegradable. As the system works independently, there is no need of human mediation to control or to do any dreary assignment from this time forward. The system is limited to the objects which look like metals but are not metals. In future, the system can be upgraded to the better detection of waste by using advanced algorithms of machine learning.

SHORTCOMINGS

The list of objects it can detect and distinguish is very less so training of a number of common objects needs to be done and the pretrained ssd_mobilenet_v2 has to be finetuned to detect additional objects too.

Also the computation in raspberry pi is very slow. So, program and other parameters needs to be optimized for pi.

APPENDIX

Appendix A1

```
import numpy as np
import tflearn
from tflearn.datasets import mnist
MNIST_data= mnist.read_data_sets(one_hot=True)
data_train = MNIST_data.validation
                                        data_test= MNIST_data.test
                                        X, y= data_train._images, data_train._labels
                                        tflearn.init_graph(num_cores=4)
                                        net = tflearn.input_data(shape=[None,784])
net = tflearn.fully_connected(net,10,activation='relu')
net = tflearn.fully_connected(net,100,activation='relu')
net = tflearn.fully_connected(net,100,activation='relu')
net = tflearn.fully_connected(net,10,activation='softmax')
net= tflearn.regression(net,loss='categorical_crossentropy', optimizer='adam')
#net= tflearn.regression(net,loss='categorical_crossentropy', optimizer='sgd')
model= tflearn.DNN(net)
model.fit(X,y,n_epoch=1, batch_size=10, show_metric = True)
```

Appendix A2

```
# -*- coding: utf-8 -*-
"""First-Tensorflow-Object-Detection-Training-Colab.ipynb

Automatically generated by Colaboratory.

Original file is located at
    https://colab.research.google.com/drive/1SgYz3kyqPqv1i4QTNMgz-zf6GXx1kxBf

# TRAINING AN OBJECT DETECTION MODEL

**CONFIGS AND HYPERPARAMETERS**
"""

#Link of github repository
repo_url = 'https://github.com/shivanidhasmana/object_detection_demo'

# Number of training steps.
num_steps = 1000  # 200000

# Number of evaluation steps.
num_eval_steps = 50

MODELS_CONFIG = {
    'ssd_mobilenet_v2': {
        'model_name': 'ssd_mobilenet_v2_coco_2018_03_29',
        'pipeline_file': 'ssd_mobilenet_v2_coco.config',
        'batch_size': 12
    },
    'faster_rcnn_inception_v2': {
        'model_name': 'faster_rcnn_inception_v2_coco_2018_01_28',
```

```
        'pipeline_file': 'faster_rcnn_inception_v2_pets.config',
        'batch_size': 12
    },
    'rfcn_resnet101': {
        'model_name': 'rfcn_resnet101_coco_2018_01_28',
        'pipeline_file': 'rfcn_resnet101_pets.config',
        'batch_size': 8
    }
}

# Pick the model you want to use
# Select a model in `MODELS_CONFIG`.
selected_model = 'ssd_mobilenet_v2'

# Name of the object detection model to use.
MODEL = MODELS_CONFIG[selected_model]['model_name']

# Name of the pipline file in tensorflow object detection API.
pipeline_file = MODELS_CONFIG[selected_model]['pipeline_file']

# Training batch size fits in Colabe's Tesla K80 GPU memory for selected model.
batch_size = MODELS_CONFIG[selected_model]['batch_size']

"""**CLONE THE GITHUB REPOSITORY WHICH HAS TEST AND TRAIN IMAGES AND ALL THE REQUIREMENTS**"""

import os

# %cd /content

repo_dir_path = os.path.abspath(os.path.join('.', os.path.basename(repo_url)))

!git clone {repo_url}
```

```
# %cd {repo_dir_path}
!git pull

"""**INSTALL REQUIRED PACKAGES**"""

# %cd /content
!git clone --quiet https://github.com/tensorflow/models.git

!apt-get install -qq protobuf-compiler python-pil python-lxml python-tk

!pip install -q Cython contextlib2 pillow lxml matplotlib

!pip install -q pycocotools

# %cd /content/models/research
!protoc object_detection/protos/*.proto --python_out=.

import os
os.environ['PYTHONPATH'] += ':/content/models/research/:/content/models/research/slim/'

!python object_detection/builders/model_builder_test.py

"""**PREPARE `TFRECORD` FILES**

Use the following scripts to generate the `tfrecord` files.
```bash
# Convert train folder annotation xml files to a single csv file,
# generate the `label_map.pbtxt` file to `data/` directory as well.
python xml_to_csv.py -i data/images/train -o data/annotations/train_labels.csv -l data/annotations

# Convert test folder annotation xml files to a single csv.
python xml_to_csv.py -i data/images/test -o data/annotations/test_labels.csv
```
```

Generate `train.record`

python generate_tfrecord.py --csv_input=data/annotations/train_labels.csv --output_path=data/annotations/train.record --img_path=data/images/train --label_map data/annotations/label_map.pbtxt

Generate `test.record`

python generate_tfrecord.py --csv_input=data/annotations/test_labels.csv --output_path=data/annotations/test.record --img_path=data/images/test --label_map data/annotations/label_map.pbtxt

```

"""

# %cd {repo_dir_path}

# Convert train folder annotation xml files to a single csv file,

# generate the `label_map.pbtxt` file to `data/` directory as well.

!python xml_to_csv.py -i data/images/train -o data/annotations/train_labels.csv -l data/annotations

# Convert test folder annotation xml files to a single csv.

!python xml_to_csv.py -i data/images/test -o data/annotations/test_labels.csv

# Generate `train.record`

!python generate_tfrecord.py --csv_input=data/annotations/train_labels.csv --output_path=data/annotations/train.record --img_path=data/images/train --label_map data/annotations/label_map.pbtxt

# Generate `test.record`

!python generate_tfrecord.py --csv_input=data/annotations/test_labels.csv --output_path=data/annotations/test.record --img_path=data/images/test --label_map data/annotations/label_map.pbtxt

test_record_fname = '/content/object_detection_demo/data/annotations/test.record'

train_record_fname = '/content/object_detection_demo/data/annotations/train.record'
```

```
label_map_pbtxt_fname                                                    =
'/content/object_detection_demo/data/annotations/label_map.pbtxt'

"""**DOWNLOAD BASE MODEL**"""

# %cd /content/models/research

import os
import shutil
import glob
import urllib.request
import tarfile
MODEL_FILE = MODEL + '.tar.gz'
DOWNLOAD_BASE = 'http://download.tensorflow.org/models/object_detection/'
DEST_DIR = '/content/models/research/pretrained_model'

if not (os.path.exists(MODEL_FILE)):
    urllib.request.urlretrieve(DOWNLOAD_BASE + MODEL_FILE, MODEL_FILE)

tar = tarfile.open(MODEL_FILE)
tar.extractall()
tar.close()

os.remove(MODEL_FILE)
if (os.path.exists(DEST_DIR)):
    shutil.rmtree(DEST_DIR)
os.rename(MODEL, DEST_DIR)

!echo {DEST_DIR}
!ls -alh {DEST_DIR}

fine_tune_checkpoint = os.path.join(DEST_DIR, "model.ckpt")
fine_tune_checkpoint
```

```
"""**CONFIGURING A TRAINING PIPELINE**"""

import os
pipeline_fname = os.path.join('/content/models/research/object_detection/samples/configs/', pipeline_file)

assert os.path.isfile(pipeline_fname), '`{}` not exist'.format(pipeline_fname)

def get_num_classes(pbtxt_fname):
    from object_detection.utils import label_map_util
    label_map = label_map_util.load_labelmap(pbtxt_fname)
    categories = label_map_util.convert_label_map_to_categories(
        label_map, max_num_classes=90, use_display_name=True)
    category_index = label_map_util.create_category_index(categories)
    return len(category_index.keys())

import re

num_classes = get_num_classes(label_map_pbtxt_fname)
with open(pipeline_fname) as f:
    s = f.read()
with open(pipeline_fname, 'w') as f:

    # fine_tune_checkpoint
    s = re.sub('fine_tune_checkpoint: ".*?"',
               'fine_tune_checkpoint: "{}"'.format(fine_tune_checkpoint), s)

    # tfrecord files train and test.
    s = re.sub(
        '(input_path: ".*?)(train.record)(.*?")', 'input_path: "{}"'.format(train_record_fname), s)
    s = re.sub(
        '(input_path: ".*?)(val.record)(.*?")', 'input_path: "{}"'.format(test_record_fname), s)
```

```
    # label_map_path
    s = re.sub(
        'label_map_path: ".*?"', 'label_map_path: "{}"'.format(label_map_pbtxt_fname), s)

    # Set training batch_size.
    s = re.sub('batch_size: [0-9]+',
               'batch_size: {}'.format(batch_size), s)

    # Set training steps, num_steps
    s = re.sub('num_steps: [0-9]+',
               'num_steps: {}'.format(num_steps), s)

    # Set number of classes num_classes.
    s = re.sub('num_classes: [0-9]+',
               'num_classes: {}'.format(num_classes), s)
    f.write(s)

!cat {pipeline_fname}

model_dir = 'training/'
# Optionally remove content in output model directory to fresh start.
!rm -rf {model_dir}
os.makedirs(model_dir, exist_ok=True)

"""**RUN TENSORBOARD (Optional)**
FOR GRAPH VISUALISATION
"""

!wget https://bin.equinox.io/c/4VmDzA7iaHb/ngrok-stable-linux-amd64.zip
!unzip -o ngrok-stable-linux-amd64.zip

LOG_DIR = model_dir
```

```
get_ipython().system_raw(
    'tensorboard --logdir {} --host 0.0.0.0 --port 6006 &'
    .format(LOG_DIR)
)

get_ipython().system_raw('./ngrok http 6006 &')

"""**GET TENSORBOARD LINK**"""

! curl -s http://localhost:4040/api/tunnels | python3 -c \
    "import sys, json; print(json.load(sys.stdin)['tunnels'][0]['public_url'])"

"""**TRAIN THE MODEL**"""

!python /content/models/research/object_detection/model_main.py \
    --pipeline_config_path={pipeline_fname} \
    --model_dir={model_dir} \
    --alsologtostderr \
    --num_train_steps={num_steps} \
    --num_eval_steps={num_eval_steps}

!ls {model_dir}

# Legacy way of training(also works).

# !python /content/models/research/object_detection/legacy/train.py --logtostderr --train_dir={model_dir} --pipeline_config_path={pipeline_fname}

"""**EXPORTING A TRAINED INFERENCE GRAPH**

Once your training job is complete, you need to extract the newly trained inference graph, which will be later used to perform the object detection. This can be done as follows:

"""

import re
```

```
import numpy as np

output_directory = './fine_tuned_model'

lst = os.listdir(model_dir)
lst = [l for l in lst if 'model.ckpt-' in l and '.meta' in l]
steps=np.array([int(re.findall('\d+', l)[0]) for l in lst])
last_model = lst[steps.argmax()].replace('.meta', '')

last_model_path = os.path.join(model_dir, last_model)
print(last_model_path)
!python /content/models/research/object_detection/export_inference_graph.py \
    --input_type=image_tensor \
    --pipeline_config_path={pipeline_fname} \
    --output_directory={output_directory} \
    --trained_checkpoint_prefix={last_model_path}

!ls {output_directory}

"""**DOWNLOAD THE MODEL `.pb` FILE**"""

import os

pb_fname = os.path.join(os.path.abspath(output_directory), "frozen_inference_graph.pb")
assert os.path.isfile(pb_fname), '`{}` not exist'.format(pb_fname)

!ls -alh {pb_fname}

from google.colab import files
files.download(pb_fname)

"""**DOWNLOAD THE `label_map.pbtxt` FILE**"""
```

```
from google.colab import files

files.download(label_map_pbtxt_fname)


'''### DOWNLOAD THE MODIFIED PIPELINE

If you plan to use OpenVINO toolkit to convert the `.pb` file to inference faster on Intel's hardware (CPU/GPU, Movidius, etc.)

'''


files.download(pipeline_fname)


# !tar cfz fine_tuned_model.tar.gz fine_tuned_model

# from google.colab import files

# files.download('fine_tuned_model.tar.gz')


'''**RUN INFERENCE TEST**


Test with images in repository `object_detection_demo/test` directory.

'''


import os

import glob


# Path to frozen detection graph. This is the actual model that is used for the object detection.

PATH_TO_CKPT = pb_fname


# List of the strings that is used to add correct label for each box.

PATH_TO_LABELS = label_map_pbtxt_fname


# If you want to test the code with your images, just add images files to the PATH_TO_TEST_IMAGES_DIR.

PATH_TO_TEST_IMAGES_DIR =  os.path.join(repo_dir_path, "test")


assert os.path.isfile(pb_fname)
```

```
assert os.path.isfile(PATH_TO_LABELS)
TEST_IMAGE_PATHS = glob.glob(os.path.join(PATH_TO_TEST_IMAGES_DIR, "*.*"))
assert len(TEST_IMAGE_PATHS) > 0, 'No image found in `{}`.'.format(PATH_TO_TEST_IMAGES_DIR)
print(TEST_IMAGE_PATHS)

# %cd /content/models/research/object_detection

import numpy as np
import os
import six.moves.urllib as urllib
import sys
import tarfile
import tensorflow as tf
import zipfile

from collections import defaultdict
from io import StringIO
from matplotlib import pyplot as plt
from PIL import Image

# This is needed since the notebook is stored in the object_detection folder.
sys.path.append("..")
from object_detection.utils import ops as utils_ops



# This is needed to display the images.
# %matplotlib inline



from object_detection.utils import label_map_util

from object_detection.utils import visualization_utils as vis_util
```

```
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='')


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)


def load_image_into_numpy_array(image):
    (im_width, im_height) = image.size
    return np.array(image.getdata()).reshape(
        (im_height, im_width, 3)).astype(np.uint8)

# Size, in inches, of the output images.
IMAGE_SIZE = (12, 8)


def run_inference_for_single_image(image, graph):
    with graph.as_default():
        with tf.Session() as sess:
            # Get handles to input and output tensors
            ops = tf.get_default_graph().get_operations()
            all_tensor_names = {
                output.name for op in ops for output in op.outputs}
```

```
        tensor_dict = {}
        for key in [
            'num_detections', 'detection_boxes', 'detection_scores',
            'detection_classes', 'detection_masks'
        ]:
            tensor_name = key + ':0'
            if tensor_name in all_tensor_names:
                tensor_dict[key] = tf.get_default_graph().get_tensor_by_name(
                    tensor_name)
        if 'detection_masks' in tensor_dict:
            # The following processing is only for single image
            detection_boxes = tf.squeeze(
                tensor_dict['detection_boxes'], [0])
            detection_masks = tf.squeeze(
                tensor_dict['detection_masks'], [0])
            # Reframe is required to translate mask from box coordinates to image
coordinates and fit the image size.
            real_num_detection = tf.cast(
                tensor_dict['num_detections'][0], tf.int32)
            detection_boxes = tf.slice(detection_boxes, [0, 0], [
                                real_num_detection, -1])
            detection_masks = tf.slice(detection_masks, [0, 0, 0], [
                                real_num_detection, -1, -1])
            detection_masks_reframed = utils_ops.reframe_box_masks_to_image_masks(
                detection_masks, detection_boxes, image.shape[0], image.shape[1])
            detection_masks_reframed = tf.cast(
                tf.greater(detection_masks_reframed, 0.5), tf.uint8)
            # Follow the convention by adding back the batch dimension
            tensor_dict['detection_masks'] = tf.expand_dims(
                detection_masks_reframed, 0)
        image_tensor = tf.get_default_graph().get_tensor_by_name('image_tensor:0')


        # Run inference
```

```
            output_dict = sess.run(tensor_dict,
                                   feed_dict={image_tensor: np.expand_dims(image, 0)})

            # all outputs are float32 numpy arrays, so convert types as appropriate
            output_dict['num_detections'] = int(
                output_dict['num_detections'][0])
            output_dict['detection_classes'] = output_dict[
                'detection_classes'][0].astype(np.uint8)
            output_dict['detection_boxes'] = output_dict['detection_boxes'][0]
            output_dict['detection_scores'] = output_dict['detection_scores'][0]
            if 'detection_masks' in output_dict:
                output_dict['detection_masks'] = output_dict['detection_masks'][0]
        return output_dict


for image_path in TEST_IMAGE_PATHS:
    image = Image.open(image_path)
    # the array based representation of the image will be used later in order to prepare the
    # result image with boxes and labels on it.
    image_np = load_image_into_numpy_array(image)
    # Expand dimensions since the model expects images to have shape: [1, None, None, 3]
    image_np_expanded = np.expand_dims(image_np, axis=0)
    # Actual detection.
    output_dict = run_inference_for_single_image(image_np, detection_graph)
    # Visualization of the results of a detection.
    vis_util.visualize_boxes_and_labels_on_image_array(
        image_np,
        output_dict['detection_boxes'],
        output_dict['detection_classes'],
        output_dict['detection_scores'],
        category_index,
        instance_masks=output_dict.get('detection_masks'),
        use_normalized_coordinates=True,
```

```
    line_thickness=8)
plt.figure(figsize=IMAGE_SIZE)
plt.imshow(image_np)
```

Appendix A3

********imagedetectioncode**********

```
from imageai.Detection import VideoObjectDetection
import os
import cv2
execution_path = os.getcwd()
camera = cv2.VideoCapture(0)
detector = VideoObjectDetection()
detector.setModelTypeAsYOLOv3()
detector.setModelPath(os.path.join(execution_path , "yolo.h5"))
detector.loadModel()
video_path=detector.detectObjectsFromVideo(camera_input=camera,output_file_path=
os.path.join(execution_path,"camera_detected_1"),frames_per_second=29,
log_progress=True)
print(camera_path)
```

********videodetectioncode**********

```
from imageai.Detection import VideoObjectDetection
execution_path = os.getcwd()
print(execution_path)
detector = VideoObjectDetection()
detector.setModelTypeAsYOLOv3()
detector.setModelPath( os.path.join(execution_path , "yolo.h5"))
detector.loadModel()
video_path=detector.detectObjectsFromVideo(input_file_path=os.path.join(
execution_path,"trafficmini.mp4"),output_file_path=os.path.join(execution_path,
"traffic_mini_detected_1"), frames_per_second=29, log_progress=True)
print(video_path)
```

Appendix A4

*********facedetection*********

```
import numpy as np
import cv2
dataset = cv2.CascadeClassifier('haarcascade_frontalface_alt.xml')
image = cv2.imread('Harry_potter.jpg', cv2.COLOR_BGR2GRAY) #Converts RGB to Gray scale
faces = dataset.detectMultiScale(image) #Image zoomed to detect pixels/edges easily
#faces has x y width height
#1.3 to zoom in image for betterreading of pixels
for x,y,w,h in faces:cv2.rectangle(image,(x,y),(x+w,y+h),(255,0,0),4)
#Here BGR not RGB Range of RGB is 0 to 255
#4:Width of rectangle around face
cv2.imwrite('result5.jpg',image)
```

Appendix B1

```
import os
import glob
import cv2
#displays the camera feed in a cv2.namedWindow and will take an snapshot when you hit SPACE. It will also quit if you hit ESC.
cam = cv2.VideoCapture(0)


cv2.namedWindow("test")


img_counter = 0


path = '/home/shivangi/Documents/pythonf/self/test/img'


while True:
    ret, frame = cam.read()
    cv2.imshow("test", frame)
    if not ret:
        break
    k = cv2.waitKey(1)


    if k%256 == 27:
        # ESC pressed
        print("Escape hit, closing...")
        break
    elif k%256 == 32:
        # SPACE pressed
        img_name = os.path.join(path , "test{}.png").format(img_counter)
        cv2.imwrite(img_name, frame)
        print("{} written!".format(img_name))
        img_counter += 1
```

```
cam.release()

repo_dir_path = "/home/shivangi/Documents/pythonf/self/test"


# Path to frozen detection graph. This is the actual model that is used for the object detection.

PATH_TO_CKPT = "/home/shivangi/Documents/pythonf/self/ssd_mobilenet_v1_coco_2018_01_28/frozen_inference_graph.pb"


# List of the strings that is used to add correct label for each box.

PATH_TO_LABELS = "/home/shivangi/Documents/pythonf/self/mscoco_label_map.pbtxt"


# If you want to test the code with your images, just add images files to the PATH_TO_TEST_IMAGES_DIR.

PATH_TO_TEST_IMAGES_DIR =  os.path.join(repo_dir_path, "img")

assert os.path.isfile(PATH_TO_CKPT)

assert os.path.isfile(PATH_TO_LABELS)

TEST_IMAGE_PATHS = glob.glob(os.path.join(PATH_TO_TEST_IMAGES_DIR, "*.*"))

assert len(TEST_IMAGE_PATHS) > 0, 'No image found in `{}`.'.format(PATH_TO_TEST_IMAGES_DIR)

print(TEST_IMAGE_PATHS)

import numpy as np

import os

import six.moves.urllib as urllib

import sys

import tarfile

import tensorflow as tf

import zipfile


from collections import defaultdict

from io import StringIO

from matplotlib import pyplot as plt

from PIL import Image

sys.path.append("..")

from object_detection.utils import ops as utils_ops
```

```
label_map_pbtxt_fname = PATH_TO_LABELS

def get_num_classes(pbtxt_fname):
    from object_detection.utils import label_map_util
    label_map = label_map_util.load_labelmap(pbtxt_fname)
    categories = label_map_util.convert_label_map_to_categories(
        label_map, max_num_classes=90, use_display_name=True)
    category_index = label_map_util.create_category_index(categories)
    return len(category_index.keys())

num_classes = get_num_classes(label_map_pbtxt_fname)
from object_detection.utils import label_map_util

from object_detection.utils import visualization_utils as vis_util


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='')


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)


def load_image_into_numpy_array(image):
```

```
    (im_width, im_height) = image.size
    return np.array(image.getdata()).reshape(
        (im_height, im_width, 3)).astype(np.uint8)

# Size, in inches, of the output images.
IMAGE_SIZE = (12, 8)



def run_inference_for_single_image(image, graph):
    with graph.as_default():
        with tf.Session() as sess:
            # Get handles to input and output tensors
            ops = tf.get_default_graph().get_operations()
            all_tensor_names = {
                output.name for op in ops for output in op.outputs}
            tensor_dict = {}
            for key in [
                'num_detections', 'detection_boxes', 'detection_scores',
                'detection_classes', 'detection_masks'
            ]:
                tensor_name = key + ':0'
                if tensor_name in all_tensor_names:
                    tensor_dict[key] = tf.get_default_graph().get_tensor_by_name(
                        tensor_name)
            if 'detection_masks' in tensor_dict:
                # The following processing is only for single image
                detection_boxes = tf.squeeze(
                    tensor_dict['detection_boxes'], [0])
                detection_masks = tf.squeeze(
                    tensor_dict['detection_masks'], [0])
                # Reframe is required to translate mask from box coordinates to image
coordinates and fit the image size.
                real_num_detection = tf.cast(
```

```
            tensor_dict['num_detections'][0], tf.int32)
        detection_boxes = tf.slice(detection_boxes, [0, 0], [
                        real_num_detection, -1])
        detection_masks = tf.slice(detection_masks, [0, 0, 0], [
                        real_num_detection, -1, -1])
        detection_masks_reframed = utils_ops.reframe_box_masks_to_image_masks(
          detection_masks, detection_boxes, image.shape[0], image.shape[1])
        detection_masks_reframed = tf.cast(
          tf.greater(detection_masks_reframed, 0.5), tf.uint8)
        # Follow the convention by adding back the batch dimension
        tensor_dict['detection_masks'] = tf.expand_dims(
          detection_masks_reframed, 0)
      image_tensor = tf.get_default_graph().get_tensor_by_name('image_tensor:0')

      # Run inference
      output_dict = sess.run(tensor_dict,
                             feed_dict={image_tensor: np.expand_dims(image, 0)})

      # all outputs are float32 numpy arrays, so convert types as appropriate
      output_dict['num_detections'] = int(
        output_dict['num_detections'][0])
      output_dict['detection_classes'] = output_dict[
        'detection_classes'][0].astype(np.uint8)
      output_dict['detection_boxes'] = output_dict['detection_boxes'][0]
      output_dict['detection_scores'] = output_dict['detection_scores'][0]
      if 'detection_masks' in output_dict:
        output_dict['detection_masks'] = output_dict['detection_masks'][0]
  return output_dict



for image_path in TEST_IMAGE_PATHS:
  image = Image.open(image_path)
  # the array based representation of the image will be used later in order to prepare the
```

```
# result image with boxes and labels on it.
image_np = load_image_into_numpy_array(image)
# Expand dimensions since the model expects images to have shape: [1, None, None, 3]
image_np_expanded = np.expand_dims(image_np, axis=0)
# Actual detection.
output_dict = run_inference_for_single_image(image_np, detection_graph)
# Visualization of the results of a detection.
vis_util.visualize_boxes_and_labels_on_image_array(
    image_np,
    output_dict['detection_boxes'],
    output_dict['detection_classes'],
    output_dict['detection_scores'],
    category_index,
    instance_masks=output_dict.get('detection_masks'),
    use_normalized_coordinates=True,
    line_thickness=8)
plt.figure(figsize=IMAGE_SIZE)
plt.imshow(image_np)
plt.savefig('xyz.jpg')
```

Appendix B2

```
import numpy as np
import os
import six.moves.urllib as urllib
import sys
import tarfile
import tensorflow as tf
import zipfile
import cv2

from collections import defaultdict
from io import StringIO
from matplotlib import pyplot as plt
from PIL import Image
from object_detection.utils import label_map_util
from object_detection.utils import visualization_utils as vis_util

# Set up camera constants
IM_WIDTH = 640
IM_HEIGHT = 480
# Define Detection box coordinates (top left and bottom right)
TL_inside = (int(IM_WIDTH*0.1),int(IM_HEIGHT*0.35))
BR_inside = (int(IM_WIDTH*0.45),int(IM_HEIGHT-5))



repo_dir_path = "/home/shivangi/Documents/pythonf/self/test"


# Path to frozen detection graph. This is the actual model that is used for the object detection.
PATH_TO_CKPT = "/home/shivangi/Documents/pythonf/self/ssd_mobilenet_v1_coco_2018_01_28/frozen_inference_graph.pb"
```

```
# List of the strings that is used to add correct label for each box.
PATH_TO_LABELS = "/home/shivangi/Documents/pythonf/self/mscoco_label_map.pbtxt"

# Number of classes to detect
NUM_CLASSES = 90



# Load a (frozen) 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 other parameters ####

# Initialize frame rate calculation
frame_rate_calc = 1
freq = cv2.getTickFrequency()
font = cv2.FONT_HERSHEY_SIMPLEX

# Loading label map
# Label maps map indices to category names, so that when our 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)


# Initialize control variables used for object detector
detected_inside = False

inside_counter = 0

pause = 0
pause_counter = 0

#************************************
def obj_detector(frame):

    # Use globals for the control variables so they retain their value after function exits
    global detected_inside, detected_outside
    global inside_counter, outside_counter
    global pause, pause_counter,check
    outside_counter=0
```

```
    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 'visualize 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=0.40)

# Draw boxes defining "inside" location.
    cv2.rectangle(frame,TL_inside,BR_inside,(20,20,255),3)
    cv2.putText(frame,"Detection Box",(TL_inside[0]+10,TL_inside[1]-10),font,1,(20,255,255),3,cv2.LINE_AA)
    # Check the class of the top detected object by looking at classes[0][0].
    # If the top detected object is a cat (17) or a dog (18) (or a teddy bear (88) for test purposes),
    # find its center coordinates by looking at the boxes[0][0] variable.
    # boxes[0][0] variable holds coordinates of detected objects as (ymin, xmin, ymax, xmax)
    if (((int(classes[0][0]) == 36) or (int(classes[0][0]) == 38) or (int(classes[0][0]) == 39) or (int(classes[0][0]) == 52) or (int(classes[0][0]) == 53) or (int(classes[0][0]) == 54) or (int(classes[0][0]) == 55) or (int(classes[0][0]) == 56) or (int(classes[0][0]) == 57) or (int(classes[0][0]) == 58) or (int(classes[0][0]) == 59) or (int(classes[0][0]) == 60) or (int(classes[0][0]) == 61) or (int(classes[0][0]) == 62) or (int(classes[0][0]) == 64) or (int(classes[0][0]) == 84) or (int(classes[0][0]) == 2) or (int(classes[0][0]) == 3) or (int(classes[0][0]) == 4) or (int(classes[0][0]) == 5) or (int(classes[0][0]) == 6) or (int(classes[0][0]) == 7) or (int(classes[0][0]) == 8) or (int(classes[0][0]) == 9) or (int(classes[0][0]) == 10) or (int(classes[0][0]) == 13) or (int(classes[0][0]) == 14) or (int(classes[0][0]) == 28) or (int(classes[0][0]) == 31) or (int(classes[0][0]) == 32) or
```

```
(int(classes[0][0]) == 33) or (int(classes[0][0]) == 37) or (int(classes[0][0]) == 40) or
(int(classes[0][0]) == 43) or (int(classes[0][0]) == 44) or (int(classes[0][0]) == 46) or
(int(classes[0][0]) == 47) or (int(classes[0][0]) == 48) or (int(classes[0][0]) == 49) or
(int(classes[0][0]) == 50) or (int(classes[0][0]) == 51) or (int(classes[0][0]) == 63) or
(int(classes[0][0]) == 65) or (int(classes[0][0]) == 67) or (int(classes[0][0]) == 70) or
(int(classes[0][0]) == 72) or (int(classes[0][0]) == 73) or (int(classes[0][0]) == 74) or
(int(classes[0][0]) == 75) or (int(classes[0][0]) == 76) or (int(classes[0][0]) == 77) or
(int(classes[0][0]) == 78) or (int(classes[0][0]) == 79) or (int(classes[0][0]) == 80) or
(int(classes[0][0]) == 81) or (int(classes[0][0]) == 82) or (int(classes[0][0]) == 85) or
(int(classes[0][0]) == 87) or (int(classes[0][0]) == 88) or (int(classes[0][0]) == 89) or
(int(classes[0][0]) == 90) and (pause == 0))):

    x = int(((boxes[0][0][1]+boxes[0][0][3])/2)*IM_WIDTH)

    y = int(((boxes[0][0][0]+boxes[0][0][2])/2)*IM_HEIGHT)


     # Draw a circle at center of object

    cv2.circle(frame,(x,y), 5, (75,13,180), -1)

    # If object is in inside box, increment inside counter variable

    if ((x > TL_inside[0]) and (x < BR_inside[0]) and (y > TL_inside[1]) and (y < BR_inside[1])):

       inside_counter = inside_counter + 1


    if (((int(classes[0][0]) == 36)) or (int(classes[0][0]) == 38) or (int(classes[0][0]) == 39) or
(int(classes[0][0]) == 52) or (int(classes[0][0]) == 53) or (int(classes[0][0]) == 54) or
(int(classes[0][0]) == 55) or (int(classes[0][0]) == 56) or (int(classes[0][0]) == 57) or
(int(classes[0][0]) == 58) or (int(classes[0][0]) == 59) or (int(classes[0][0]) == 60) or
(int(classes[0][0]) == 61) or (int(classes[0][0]) == 62) or (int(classes[0][0]) == 64) or
(int(classes[0][0]) == 84) and (pause == 0)):

       check = 1

    elif (((int(classes[0][0] == 2) or (int(classes[0][0]) == 3) or (int(classes[0][0]) == 4) or
(int(classes[0][0]) == 5) or (int(classes[0][0]) == 6) or (int(classes[0][0]) == 7) or
(int(classes[0][0]) == 8) or (int(classes[0][0]) == 9) or (int(classes[0][0]) == 10) or
(int(classes[0][0]) == 13) or (int(classes[0][0]) == 14) or (int(classes[0][0]) == 28) or
(int(classes[0][0]) == 31) or (int(classes[0][0]) == 32) or (int(classes[0][0]) == 33) or
(int(classes[0][0]) == 37) or (int(classes[0][0]) == 40) or (int(classes[0][0]) == 43) or
(int(classes[0][0]) == 44) or (int(classes[0][0]) == 46) or (int(classes[0][0]) == 47) or
(int(classes[0][0]) == 48) or (int(classes[0][0]) == 49) or (int(classes[0][0]) == 50) or
(int(classes[0][0]) == 51) or (int(classes[0][0]) == 63) or (int(classes[0][0]) == 65) or
(int(classes[0][0]) == 67) or (int(classes[0][0]) == 70) or (int(classes[0][0]) == 72) or
(int(classes[0][0]) == 73) or (int(classes[0][0]) == 74) or (int(classes[0][0]) == 75) or
(int(classes[0][0]) == 76) or (int(classes[0][0]) == 77) or (int(classes[0][0]) == 78) or
(int(classes[0][0]) == 79) or (int(classes[0][0]) == 80) or (int(classes[0][0]) == 81) or
(int(classes[0][0]) == 82) or (int(classes[0][0]) == 85) or (int(classes[0][0]) == 87) or
(int(classes[0][0]) == 88) or (int(classes[0][0]) == 89) or (int(classes[0][0]) == 90))) and (pause
== 0)):

       check = 2
```

```
    # If object has been detected inside for more than 10 frames, set detected_inside flag
    # and display the message
    if inside_counter > 5:
        detected_inside = True
        inside_counter = 0
        # Pause object detection by setting "pause" flag
        pause = 1
    # If pause flag is set, draw message on screen.
    if pause == 1:
        if detected_inside == True:
            if (check == 1):

cv2.putText(frame,"Biodegradable",(int(IM_WIDTH*.1),int(IM_HEIGHT*.5)),font,1,(0,0,0),5,cv2.
LINE_AA)


cv2.putText(frame,"Biodegradable",(int(IM_WIDTH*.1),int(IM_HEIGHT*.5)),font,1,(95,176,23),3
,cv2.LINE_AA)
            elif (check == 2):
                cv2.putText(frame,"Non-
biodegradable",(int(IM_WIDTH*.1),int(IM_HEIGHT*.5)),font,1,(0,0,0),5,cv2.LINE_AA)
                cv2.putText(frame,"Non-
biodegradable",(int(IM_WIDTH*.1),int(IM_HEIGHT*.5)),font,1,(95,176,23),3,cv2.LINE_AA)
        inside_counter=0
        pause_counter = pause_counter + 1
        if pause_counter > 30:
            pause = 0
            pause_counter = 0
            detected_inside = False


    # Draw counter info
    cv2.putText(frame,'Detection counter: ' +
str(max(inside_counter,outside_counter)),(10,100),font,0.5,(255,255,0),1,cv2.LINE_AA)
    cv2.putText(frame,'Pause counter: ' +
str(pause_counter),(10,150),font,0.5,(255,255,0),1,cv2.LINE_AA)
```

```
    return frame



# Helper code
def load_image_into_numpy_array(image):
    (im_width, im_height) = image.size
    return np.array(image.getdata()).reshape(
        (im_height, im_width, 3)).astype(np.uint8)

# Initialize USB webcam feed
camera = cv2.VideoCapture(0)
ret = camera.set(3,IM_WIDTH)
ret = camera.set(4,IM_HEIGHT)

# Continuously capture frames and perform object detection on them
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()

    # Pass frame into pet detection function
    frame = obj_detector(frame)

    # Draw FPS
    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)

# FPS calculation
t2 = cv2.getTickCount()
time1 = (t2-t1)/freq
frame_rate_calc = 1/time1

# Press 'q' to quit
if cv2.waitKey(1) == ord('q'):
    cv2.destroyAllWindows()
    camera.release()
    break
```

REFERENCES

- **Deep Learning with Python** by Françhois Chollet
- **Machine Learning Refined: Foundations, Algorithms, and Applications** by Aggelos Konstantinos Katsaggelos, Jeremy Watt, and Reza Borhani
- **Deep Learning for computer Vision with Python (Starter Bundle)** by Adrian Rosebrock
- **NPTEL Lectures for Course : Introduction to Machine Learning** by Sudeshna Sarkar
- https://www.datacamp.com/community/tutorials/object-detection-guide
- https://www.analyticsvidhya.com/blog/2016/03/introduction-deep-learning-fundamentals-neural-networks/