DAYANANDA SAGAR UNIVERSITY

KUDLU GATE, BANGALORE – 560068

Bachelor of Technology
in
COMPUTER SCIENCE AND ENGINEERING

Major Project Phase-II Report

PLASTIC DETECTION IN THE SURROUNDINGS USING MACHINE LEARNING

By

Santosh G C	**ENG18CS0245**
Praveen R	**ENG18CS0216**
Santosh Hugar	**ENG18CS0246**
Retish Kajuluri	**ENG18CS0227**

Under the supervision of
Prof. Kalpana B N
Assistant Professor, Department of Computer Science and Engineering

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING,
SCHOOL OF ENGINEERING
DAYANANDA SAGAR UNIVERSITY,
BANGALORE

(2021-2022)

DAYANANDA SAGAR UNIVERSITY

School of Engineering
Department of Computer Science & Engineering
Kudlu Gate, Bangalore – 560068
Karnataka, India

CERTIFICATE

This is to certify that the Phase-II project work titled "**PLASTIC DETECTION IN THE SURROUNDINGS USING MACHINE LEARNING**" is carried out by **Santhosh G C(ENG18CS0245), Praveen R(ENG18CS0216), Santosh Hugar(ENG18CS0246), Retish Kajuluri(ENG18CS0227),** a bonafide students of Bachelor of Technology in Computer Science andEngineering at the School of Engineering, Dayananda Sagar University, Bangalore in partial fulfilment for the award of degree in Bachelor of Technology in Computer Science and Engineering, during the year **2021-2022**.

Prof.Kalpana B N	**Dr. Girisha G S**	**Dr. A Srinivas**
Assistant Professor Dept. of CSE, School of Engineering Dayananda Sagar University	Chairman CSE School of Engineering Dayananda Sagar University	Dean School of Engineering Dayananda Sagar University
Date:21/06/2022	Date:21/06/2022	Date:21/06/2022

Name of the Examiner	**Signature of Examiner**
1.	
2.	

DECLARATION

We, **Santhosh (ENG18CS0245),Praveen R(ENG18CS0216),Santosh Hugar(ENG18CS0246), Retish Kajuluri (ENG18CS0270),** are students of the Eighth Semester B.Tech in **Computer Science and Engineering**, at School of Engineering, **Dayananda Sagar University**, hereby declare that the phase-II project titled **"PLASTIC DETECTION IN THE SURROUNDINGS USING MACHINE LEARNING"** has been carried out by us and submitted in partial fulfilment for the award of degree in **Bachelor of Technology in Computer Science and Engineering** during the academic year **2021-2022**.

Student **Signature**

Name: Santhosh G C

USN: ENG18CS0245

Name: Praveen R

USN: ENG18CS0216

Name: Retish Kajuluri
USN: ENG18CS0227

Name: Santosh Hugar

USN: ENG18CS0246

Place: Bangalore

Date :

ACKNOWLEDGEMENT

It is a great pleasure for us to acknowledge the assistance and support of many individuals who have been responsible for the successful completion of this project work.

First, we take this opportunity to express our sincere gratitude to School of Engineering & Technology, Dayananda Sagar University for providing us with a great opportunity to pursue ourBachelor's degree in this institution.

*We would like to thank **Dr. A Srinivas. Dean**, **School of Engineering & Technology**, **Dayananda Sagar University** for his constant encouragement and expert advice.*

*It is a matter of immense pleasure to express our sincere thanks to **Dr. Girisha GS , Chairman**, Department of **Computer Science and Engineering**, **Dayananda Sagar University,** for providing right academic guidance that made our task possible.*

*We would like to thank our guide **Prof.Kalpana B N,** Assistant Professor, Dept. of Computer Science and Engineering, Dayananda Sagar University, for sparing her valuable time to extend help in every step of our project work, which paved the way for smooth progress and fruitful culmination of the project.*

*We would like to thank our **Project Coordinators Dr. Meenakshi Malhotra and Dr. Bharanidharan N,** and all the staff members of Computer Science and Engineering for their support.*

We are also grateful to our family and friends who provided us with every requirement throughout the course.

We would like to thank one and all who directly or indirectly helped us in the Project work.

TABLE OF CONTENTS

LIST OF ABBREVIATIONS

ML	Machine Learning
DL	Deep Learning
KNDVI	kernel Normalized Difference Vegetation Index
NN	Neural Network
CNN	Convolution Neural Network
KNN	K-Nearest Neighbors
VS Code	Visual Studio Code

LIST OF FIGURES

ABSTRACT

One of the major innovations of the past century has been the introduction and wide adoption of plastics for many day-to-day activities. With our plastic use set to increase by 40% in the next ten years, this paints a depressing picture because excessive use leads to several health issues.

One of those is Microplastics. Microplastics may be tiny but they are a big problem. Researchers recently tested various organs from people who had passed away for microplastics, and they found traces of it in every organ. BPA free alternatives are still plastics, and can still have toxic effects on us. According to a 2011 study published by NCBI, harmful chemicals in plastic materials can cause adverse health outcomes including cancer, birth defects, developmental and reproductive issues, endocrine disruption and compromised immunity. According to a 2019, WWF study, the average human potentially eats around 2000 microplastics weekly. While there is a plenty of research about the potential side effects of plastic on the human body, there is still much more to learn. This project mainly uses machine learning to identify how much we are exposed to plastics in our surroundings and helps in quantifying the usage of plastics.

CHAPTER 1 INTRODUCTION

The world has rapidly produced plastic due to it is a highly demanded industrial material for its cost-effectiveness. However, the rapid consumption of single-use plastics and waste mismanagement has currently caused 37% of world plastic waste being poorly managed [1]. Waste mismanagement refers to inadequate waste collection, treatment, disposal and management system. Mismanaged plastic waste has caused plastic leaking into nature and entering our food chain as microplastics. Microplastics exist in different types of sizes, shapes and colours with some spherical, fibrous or random appearance. According to Crawford & Quinn [2], microplastic is generally defined as any piece of plastic in size along its longest dimensions, which its standardized size can be categorized into macroplastic (≥25 mm), mesoplastic (<25 mm to 5 mm), plasticle (<5 mm), microplastic plus mini-microplastic (<5 mm to 1 μm), and finally nanoplastic (<1 μm). On the other hand, microplastics can be further classified as pellet, microbead, fragment, fibre, film, and foam, as shown in Table 1. However, four main categories, i.e., bead, fibre, fragment and film, are commonly found in microplastic studies [3, 4, 5]. Fragments are rigid particles with angular and irregular shaped, and some fragments particles are thick with sharp crooked edges. It is believed that fragments primarily come from hard plastics through fragmentation. Fibres are long thin or thread-like with a slender shaped piece particle. It comes from fabrics, nets, fishing lines, and ropes. Beads are spherical, or an aggregate of spheres shaped piece particle which came from the cosmetics.

Nowadays, plastics are indispensable as storage, protection and packaging materials in all areas of the industry worldwide. At the same time, the number of different types of plastics is constantly increasing, and the processes for recycling plastics are constantly being improved, in order to guarantee the highest possible quality for reuse. Nevertheless, plastics can only be reused if they are properly disposed of. Unfortunately, this is not always the case due to, for example, a lack of waste disposal systems. As a result, the environmental impact of plastic is currently growing immeasurably. American researchers found out that in 2010, a total of 275 million metric tons of plastic waste was produced in coastal regions of the earth, and, according to their calculation, 4.8 to 12.7 million metric tons of waste is disposed of in our oceans (Jambeck et al., 2015).

This Project focus on detecting the total objects in the room and classifying the plastic objects and quantifying it with respect to the total objects with that the user can see how much they are depending on plastic in their daily life.

1.1 PURPOSE

With the increasing repercussions of the growing amount of plastic there is an immediate threat to the environment and us. With the project, we intend to understand and work out how much the public is dependent on plastic on daily basis. Based on our findings we also try to understand the impact that it will have in the future.

1.2 SCOPE

Create an Interactable dynamic progressive application. Users are allowed to give images or video as the input, and plastic will be detected present in the user input, and it will be quantified. The quantity of plastic present in the user's surroundings will be displayed.

CHAPTER 2
PROBLEM DEFINITION

CHAPTER 2 PROBLEM DEFINITION

2.1 PROBLEM STATEMENT

As we all know the effects of plastic but still, we use plastic a lot. Since there are many disadvantages of using plastic, it is mandatory to keep a track of how much we are using it. Every day one or the other way we use the plastic, and we should reduce the usage of plastic. Reducing the use of plastic is important because plastic production requires an enormous amount of energy and resources. This causes carbon emissions and contributes to global warming. Recycling plastic is not efficient-only 9% of plastic ever produced has been recycled.

CHAPTER 3
LITERATURE REVIEW

CHAPTER 3 LITERATURE REVIEW

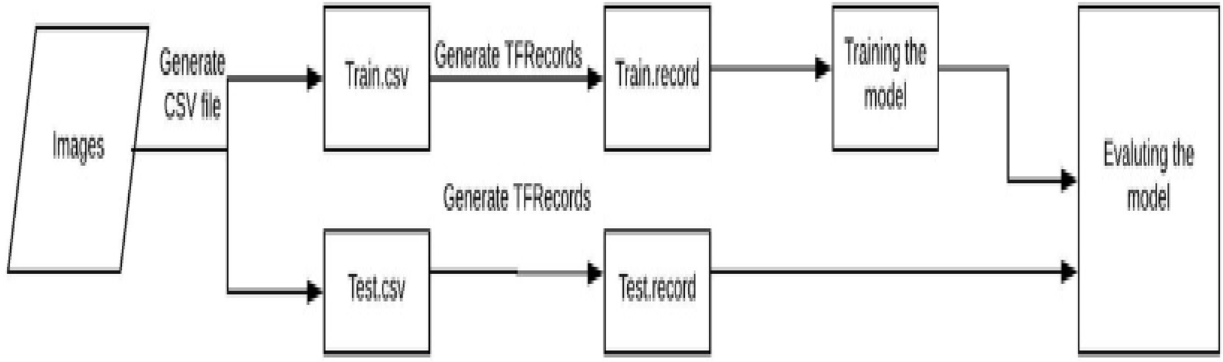
TITLE	CONFERENCE NAME AND YEAR	TECHNOLOGY USED	REULT	WHAT YOU INFER
Using Deep Learning to Quantify, Monitorand Remove Marine Plastic.	Published in Towards Data Data Science, June 21, 2021	YOLOv5- S API	Using our model, we can now detecton average 85% (mAP) of all epipelagic plastic in the ocean. We achieve this level of precision basedon a Neural Network architecture called YOLOv5-S	Data is filtered and choose the best. Every image is taken as input in different styles and trained the model.
Deep learning for plastic waste classification system	Applied Computatio nal Intelligence and Soft Computing / 2021	CNN	The result of the experiment show that 15 layer achieves better performance for images of 120*120 pixel compared to 23 layer network for 227*227 pixel.	The research results in Europe showed that the investment outlays for obtaining primary raw materials are much higher than the outlays incurred in relation to the use of secondary raw materials obtained from production waste or waste after use.

Detection of marine floating plastic using Sentinel-2 Imagery and machine learning models	Cornel University 27,May 2021	Neural Network	Neural Network(NN) based AC processor used. A total of 54 validated plastic pixels colled from Greece and Cyprus were used to train the supervised Ml Models.	2 The present study utilized high-resolution Sentinel 2A/B satellite imagery and advanced machine learning models to detect and classify marine floating plastics in Mytilene (Greece), Limassol (Cyprus), Calabria (Italy) and Beirut (Lebanon).
Machine Learning for aquatic plastic litter detection, classification, and quantification	Published by IOP Publishing Ltd Environmenl Research Letters, 16 November 2020	PLQ-CNN PLD-CNN Random Forest	In this paper the authors used the CNN algorithm. It was aimed at providing estimations of litter in a survey region as counts of litter items and predict plastic types from the imagery.	The given data is analyzed and fitted into multiple classification models to generate a prediction.

CHAPTER 4
PROJECT DESCRIPTION

CHAPTER 4 PROJECT DESCRIPTION

4.1 PROPOSED DESIGN FLOWCHART



4.2 PROPOSED DESIGN

When designing the CNN model for plastic type classification into 8 different types such as metal, cardboard, plastic can, plastic tub, plastic bottle, trash, glass other. The first step is collection of appropriate images of plastics and next step involves in renaming of these images. After that we need to fix the size of the input image. At time the high resolution of images may lead to overloading of the computational units and increase the memory. At same time the lower resolute image makes the identification difficult.

Due to its poor quality the object identification process may be affected and decease the performance of the model. We can use the image pixel width * height with 64*64 and 120 * 120 pixels. After this we need to decide the number of layers to be used inside the CNN model. We analysed the model with different approaches. First, we used LeNet-5 in introduced by Yann LeCun, LeNet-5 has 7 layers in which input layer is excluded. The LeNet-5 input layer is of 32*32 image size. Secondly, we used the AlexNet (Figure 3) structure. In this the first layer consisted of 64 filter of size 11*11.

In our proposed work, the simplified model for skin cancer identification is used. In this model the preparation of the input image for validation and learning is the important phase. In addition to convolutional layer, we have use max-pooling layer, activation function, drop-out layer, SoftMaxlayer. The convolutional layer extracts the feature and passes it to the next layer. The system as a whole is examined, and the system's inputs are recognised. The different procedures are linked to the organisations' outputs. The goal of system analysis is to become aware of the problem, identify the important and decisional variables, analyse and synthesise the numerous components, and come up with an optimal or at least adequate solution or plan of action.

CHAPTER 5
REQUIREMENTS

CHAPTER 5 REQUIREMENTS

5.1 FUNCTIONAL REQUIREMENTS

5.1.1 OPERATING REQUIREMENTS

- Dataset
- Applied Plastic Waste Detection Field
- Image Pre-processing
- Image Data Augmentation

5.2 NON-FUNCTIONAL REQUIREMENTS

5.2.1 SOFTWARE REQUIREMENTS

- Operating System : Windows10
- Front end : Anaconda, Spider
- Coding Language : Python

5.2.2 HARDWARE REQUIREMENTS

- Processor Type : Windows 8 or above
- Speed : 2.50GHZ or more
- RAM : 4GB RAM or above
- Hard disk : 250GB or more
- Keyboard : 101/102 Standard Keys
- Mouse : Optical Mouse

CHAPTER 6

METHODOLOGY

CHAPTER 6 METHODOLOGY

6.1 MODULES DESCRIPTION

In this project, contains the following modules

- Dataset Description
- Data Pre-processing
- Image Data Augmentation
- Feature Extraction

6.1.1 Dataset Description

In this project, we will create three separate machine learning models for diagnosing three different diseases, therefore we will use various datasets from the UCI Machine Learning repository and Kaggle for each ailment.

6.1.2 Data Preprocessing

Prior to model training, image pre-processing is performed. The main goal is to remove unnecessary information from the picture, increase the detectability of important and important details hidden in the image, and substantially decrease the data to enhance the feature extraction of the model. The fusion findings' ultimate output is achieved. Another option is to combine the characteristics learnt from various models and then finalize findings of the Process flow in fusion procedures previously discussed. There are fourteen pieces of literature regarding multi model fusion that have been gathered.

6.1.3 Image Data Augmentation

Plastic Waste data are difficult to collect due to the problems of personal privacy and professional equipment involved in the collection process of the plastic waste dataset. Accordingly, less plastic waste data has been collected. Some plastic waste rarity makes the data collection of this category less, resulting in the uneven distribution of the collected datasets. In deep learning, small-scale datasets can easily lead to insufficient model learning and over fitting.

The solving of the shortcomings of the traditional plastic waste diagnostic process and image recognition technology of plastic waste based on machine learning. Image recognition based on machine learning is an interdisciplinary field integrating plastic waste imaging, mathematical modelling, and computer technology through feature engineering and machine learning classification algorithms to complete the recognition.

6.1.4 Feature Extraction

The Feature Extraction procedure is used to update the crucial data for outcome characteristics. This method aims to reduce the number of resources required to explain a huge quantity of data. The process of minimizing the number of characteristics is known as feature extraction. This is also used to increase the speed and efficacy of supervised learning.

CHAPTER 7
EXPERIMENTATION

CHAPTER 7 EXPERIMENTATION

CNN is a model which is designed to process arrays of data such as images. The first step here will be to resize all images as CNN cannot train images of different sizes. We compute the mean for both dimensions and resize all the images. The sequential model is used here. In the first layer which is the Convolution layer, we will place the filter on top of the input matrix and then compute the value and will be doing a stride jump of 1. This extract features from the image. Also, we can use Padding if the filter does not fit perfectly in the input image. Here we will be using the Relu activation function.

Max pooling selects the maximum element and extracts the most prominent features from the image. Last is the fully connected layer, where the input to the fully connected layer will be the output from the Max Pooling Layer; it is flattened and then fed into the fully connected layer.

Object detection works in two ways:

- The first division permits networks to isolate the tasks of locating objects and classifying them (Faster R-CNN);
- The second division allows networks to predict class scores and bounding boxes (YOLO and SSD networks).

Nevertheless, most object detection tutorials are done by drawing bounding boxes around the input image to define the objects and their locations.
However, object detection is different from image identification, although both usually go hand-in-hand. The difference lies in how models detect the objects. In image identification, detectors see and label an entire image, while only objects within an image are detected in object detection. More like zooming into the pexels of the image.
The closest example of object TF detection is the Google Lens, an image and object recognition program. Google Lens works by identifying objects for curious users. All users need to begin is to take a picture of the thing. Google Lens then identifies the real-world object and fetches the required information from the internet.

CHAPTER 8
TESTING AND RESULT

CHAPTER 8 TESTING AND RESULT

Test Case 1:

Figure 8.1: Plastic Cover Detection

Figure 8.2: Cardboard Detection

Figure 8.3: Glass Detection

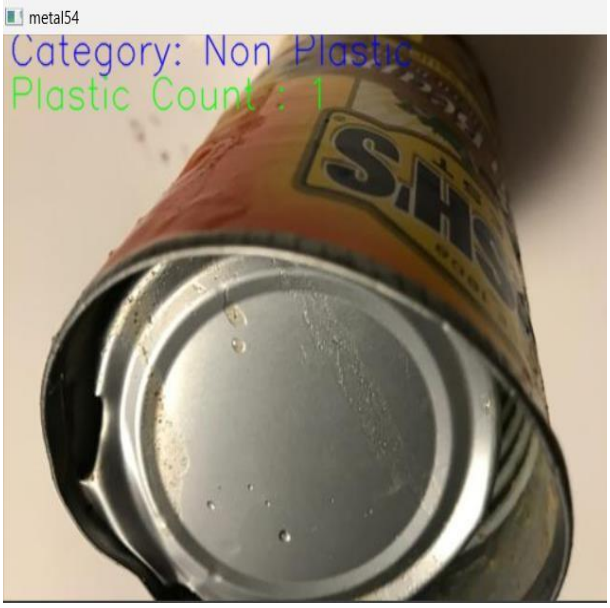


Figure 8.4: Metal Detection

Figure 8.5: Paper Detection

Figure 8.6: Plastic Tub Detection

Figure 8.7: Plastic Bottle Detection

Figure 8.8: Trash Detection

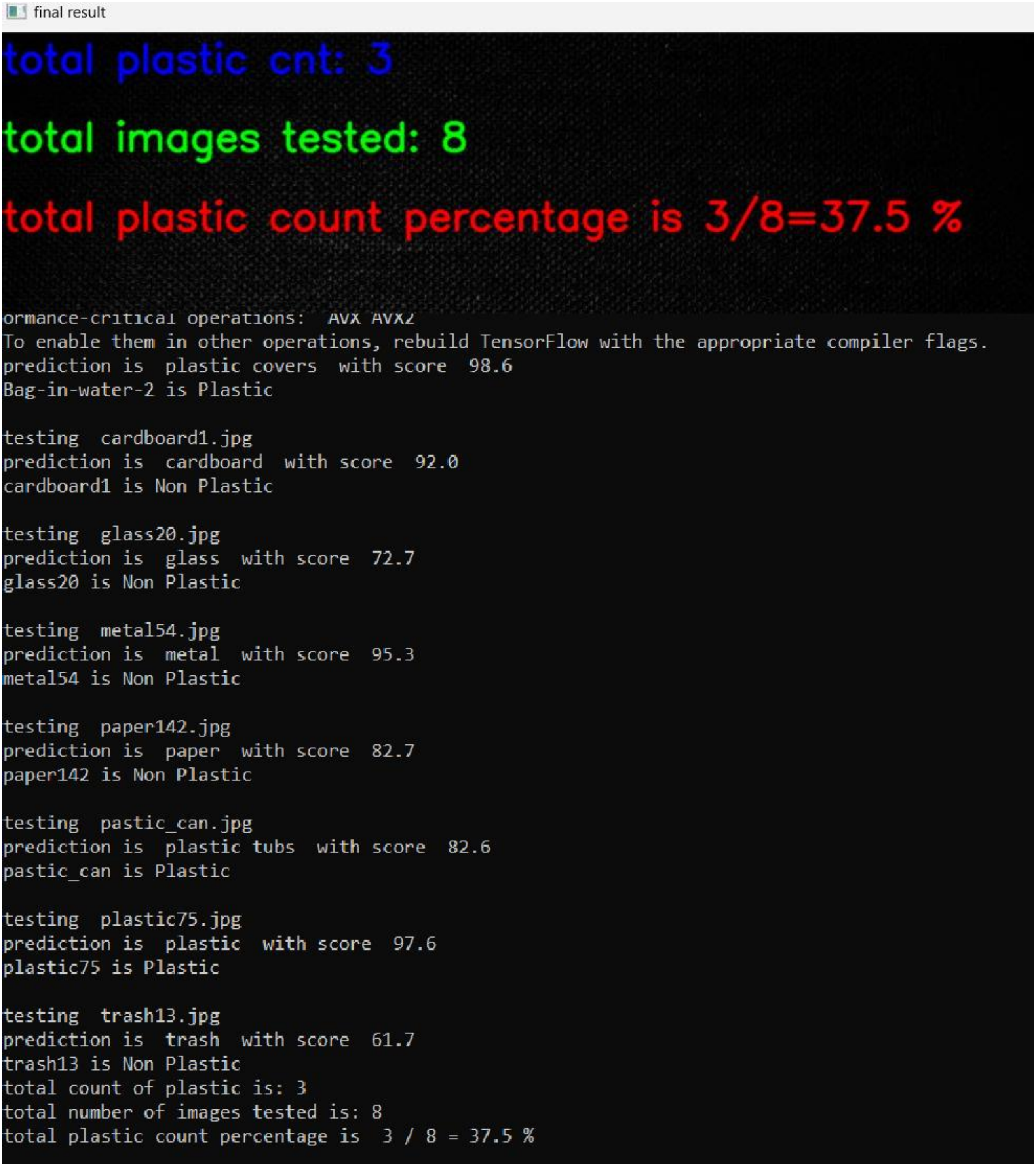


Figure 8.9: Test case 1Final Output

Testcase 2:

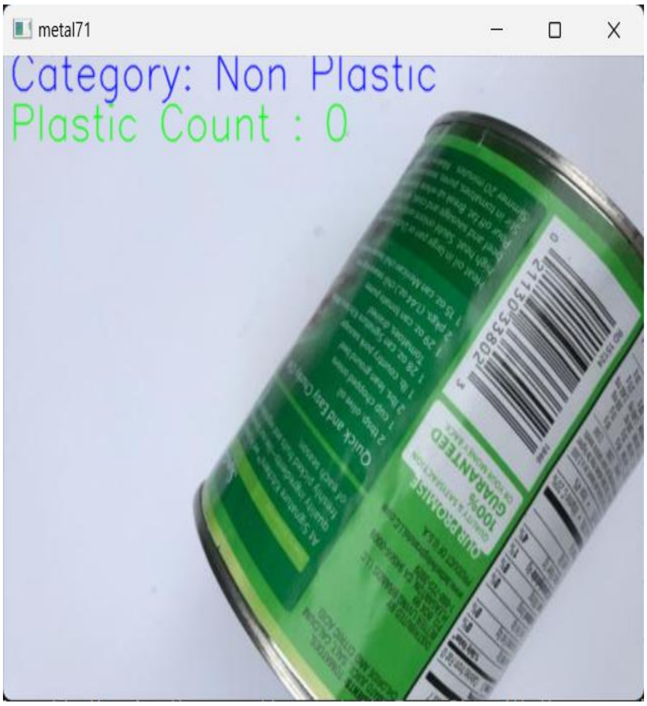


FIGURE 8.10: METAL DETECTION

FIGURE8.11: PLASTIC BAG DETECTION

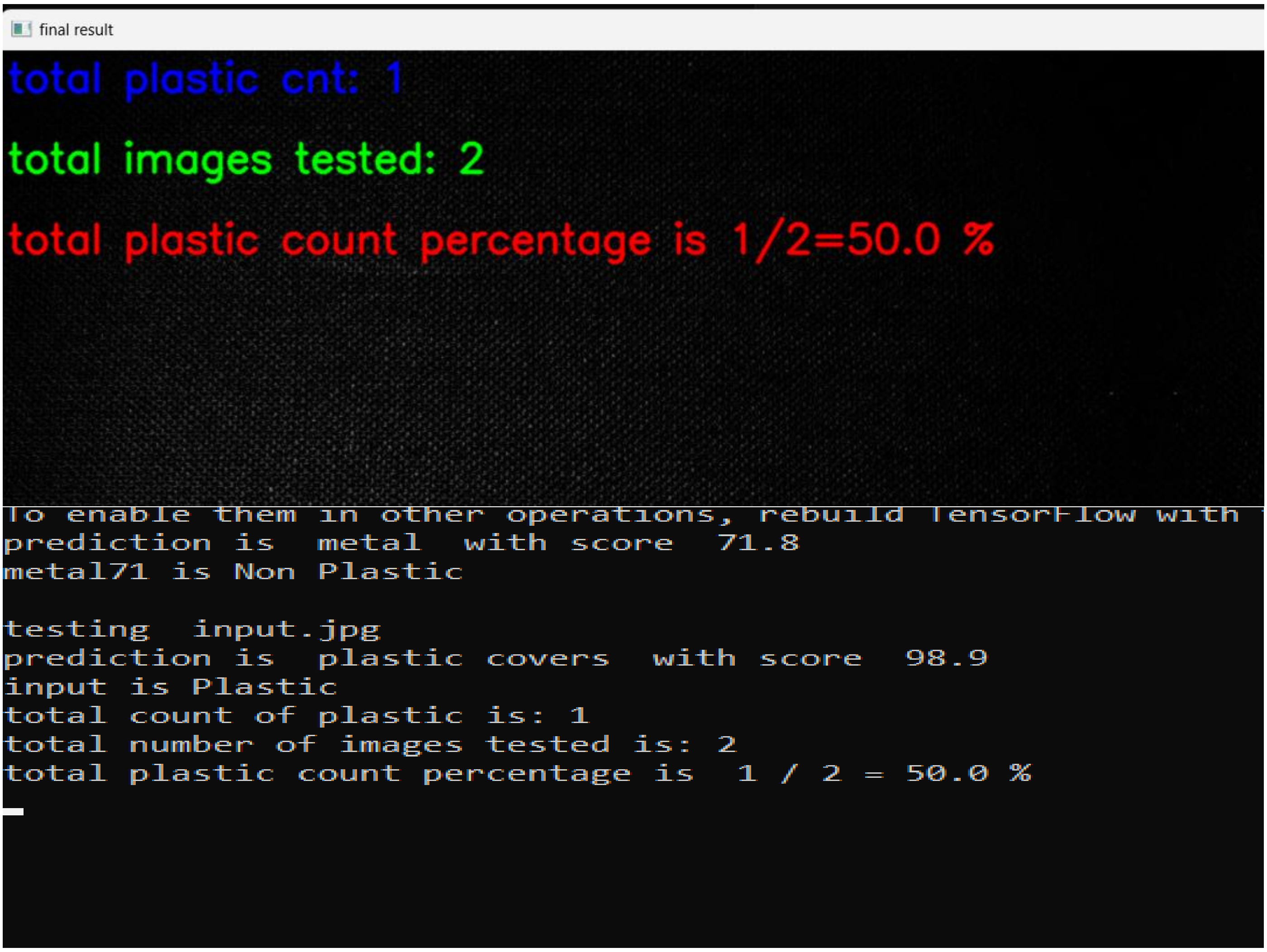


FIGURE 8.12: TESTCASE 2 FINAL OUTPUT

CHAPTER 9 CONCLUSION AND FUTURE WORK

As the name indicates it helps to detect the plastic objects present in an area and it will show the percentage of plastic being present in that area with respect to total objects discovered. So, using these results one knows the amount of plastic is being used in their day to days life. Since plastic causes more effects on human beings/ animals it has to be reduced.

In the future, more advanced machine learning algorithms will required to improve the object detection efficiency. frequently after the training period to improve performance. Furthermore, to minimiz over fitting and improve the accuracy of deployed models, datasets should be enlarged on diverse demo-graphics. Finally, to improve the performance of learning models, more relevant feature selection approaches should be applied.sssss

REFERENCES

[1] Solid waste issue: Sources, composition, disposal, recycling, and valorization” 2018 By panelHussein I.Abdel-ShafyaMona S.M.Mansou.

[2] A review on au tomated sorting of source-separated municipal solid waste for recycling” 2016 By Sathish Paulraj Gundupalli 1, Subrata Hait 2, Atul Thakur 3.

[3] Influence of shape and size of the particles on jigging separation of plastics mixture,” 2015 Fernando Pita 1, Ana Castilho 2.

[4] Intelligent solid waste processing using optical sensor based sorting technology,” 2010 Jiu Huang; Thomas Pretz; Zhengfu Bian All Authors.

[5] Advanced waste-splitting by sensor based sorting on the example of the MTPlant oberlaa,” 2020 Janusz Bobulskia and Mariusz Kubanekb .

[6] Upgrading the quality of mixed recycled aggregates from construction and demolition waste by using near-infrared sorting technology,” 2015 InigoVegas,Kris Broosb,Peter Nielsenb, Oliver Lambertz,Amaia Lisbonaa.

[7] Case study: interpretability of fuzzy systems applied to nonlinear modelling and control,” 2017 Krzysztof Cpalka.

[8] Real-time hyperspectral processing for automatic nonferrous material sorting,” 2012Artzai Picon, Aranzazu Bereciartua, Jone Echazarra, Ovidiu Ghita, Paul F. Whelan, Pedro M. Iriondo.

[9] Industrial application for inline material sorting using hyperspectral imaging in the NIR range,” 2005 Petra Tatzer,Markus Wolf,Thomas Panner.

[10] Sorting of polypropylene resins by color in MSW using visible reflectance spectroscopy,” 2010 S. M. Safavi, H. Masoumi, S. S. Mirian, and M. Tabrizchi.

[11] utilization of hyperspectral imaging for impurities detection in secondary plastics,” 2012 S. Serranti, A. Gargiulo, G. Bonifazi, A. Toldy, S. Patachia, and R. Buican.

[12] Characterization of post-consumer polyolefin wastes by hyperspectral imaging for quality control in recycling processes,” 2011 S. Serranti, A. Gargiulo, and G. Bonifazi.

[13] Rapid discrimination of plastic packaging ma terials using MIR spectroscopy coupled with independent components analysis (ICA)," 2014 A. Kassouf, J. Maalouly, D. N. Rutledge, H. Chebib, and V. Ducruet.

[14] Deep: convolutional neural network applies to face recognition in small and medium databases," 2018 M. Wang, Z. Wang, and J. Li.

[15] Application of foreground object patterns analysis for event detection in an innovative video surveil lance system," 2015 D. Frejlichowski, K. Go´sciewska, P. Forczmanski, and ´ R. Hofman.

[16] PET waste classification method and plastic waste database - WaDaBa," 2018 J. Bobulski and J. Piatkowski.

[17] Waste classification system using image processing and convolutional neural networks," 2019 J. Bobulski and M. Kubanek.

[18] CNN use for plastic garbage classification method," 2019 J. Bobulski and M. Kubanek.

APPENDIX A

Train.py

```
import argparse
#from datetime import datetime
import hashlib
import os.path
import random
import re
import struct
import sys
import tarfile

import numpy as np
from six.moves import urllib
import tensorflow.compat.v1 as tf

from tensorflow.python.framework import graph_util
from tensorflow.python.framework import tensor_shape
from tensorflow.python.platform import gfile
from tensorflow.python.util import compat

FLAGS = None

# These are all parameters that are tied to the particular model architecture
# we're using for Inception v3. These include things like tensor names and their
# sizes. If you want to adapt this script to work with another model, you will
# need to update these to reflect the values in the network you're using.
DATA_URL = 'http://download.tensorflow.org/models/image/imagenet/inception-2015-12-
05.tgz'

BOTTLENECK_TENSOR_NAME = 'pool_3/_reshape:0'
BOTTLENECK_TENSOR_SIZE = 2048
MODEL_INPUT_WIDTH = 299
MODEL_INPUT_HEIGHT = 299
MODEL_INPUT_DEPTH = 3
JPEG_DATA_TENSOR_NAME = 'DecodeJpeg/contents:0'
RESIZED_INPUT_TENSOR_NAME = 'ResizeBilinear:0'
MAX_NUM_IMAGES_PER_CLASS = 2 ** 27 - 1  # ~134M


def create_image_lists(image_dir, testing_percentage, validation_percentage):
    """
    if not gfile.Exists(image_dir):
        print("Image directory '" + image_dir + "' not found.")
        return None
    result = {}
    sub_dirs = [x[0] for x in gfile.Walk(image_dir)]
```

```
    is_root_dir = True
    for sub_dir in sub_dirs:
        if is_root_dir:
            is_root_dir = False
            continue
        extensions = ['jpg', 'jpeg', 'JPG', 'JPEG']
        file_list = []
        dir_name = os.path.basename(sub_dir)
        if dir_name == image_dir:
            continue
        print("Looking for images in '" + dir_name + "'")
        for extension in extensions:
            file_glob = os.path.join(image_dir, dir_name, '*.' + extension)
            file_list.extend(gfile.Glob(file_glob))
        if  not  file_list:
            print('No files found')
            continue
        if len(file_list) < 20:
            print('WARNING: Folder has less than 20 images, which may cause
issues.')
        elif len(file_list) > MAX_NUM_IMAGES_PER_CLASS:
            print('WARNING: Folder {} has more than {} images. Some images will '
                  'never be selected.'.format(dir_name, MAX_NUM_IMAGES_PER_CLASS))
        label_name = re.sub(r'[^a-z0-9]+', ' ', dir_name.lower())
        training_images = []
        testing_images = []
        validation_images = []
        for file_name in file_list:
            base_name  =  os.path.basename(file_name)
            hash_name = re.sub(r'_nohash_.*$', '', file_name)

            hash_name_hashed = hashlib.sha1(compat.as_bytes(hash_name)).hexdigest()
            percentage_hash = ((int(hash_name_hashed, 16) %
                                (MAX_NUM_IMAGES_PER_CLASS + 1)) *
                               (100.0 / MAX_NUM_IMAGES_PER_CLASS))
            if percentage_hash < validation_percentage:
                validation_images.append(base_name)
            elif percentage_hash < (testing_percentage + validation_percentage):
                testing_images.append(base_name)
            else:
                training_images.append(base_name)
        result[label_name] = {
            'dir':  dir_name,
            'training': training_images,
            'testing': testing_images,
            'validation': validation_images,
            }
    return result
```

```
def get_image_path(image_lists, label_name, index, image_dir, category):

    if label_name not in image_lists:
        tf.logging.fatal('Label does not exist %s.', label_name)
    label_lists = image_lists[label_name]
    if category not in label_lists:
        tf.logging.fatal('Category does not exist %s.', category)
    category_list = label_lists[category]
    if not category_list:
        tf.logging.fatal('Label %s has no images in the category %s.', label_name,
category)
    mod_index = index % len(category_list)
    base_name = category_list[mod_index]
    sub_dir = label_lists['dir']
    full_path = os.path.join(image_dir, sub_dir, base_name)
    return full_path


def get_bottleneck_path(image_lists, label_name, index, bottleneck_dir, category):
    """"

    return get_image_path(image_lists, label_name, index, bottleneck_dir,
                          category) + '.txt'


def create_inception_graph():

    with tf.Graph().as_default() as graph:
        model_filename = os.path.join(FLAGS.model_dir,
'classify_image_graph_def.pb')
        with gfile.FastGFile(model_filename, 'rb') as f:
            graph_def = tf.GraphDef()
            graph_def.ParseFromString(f.read())
            bottleneck_tensor, jpeg_data_tensor, resized_input_tensor = (
                tf.import_graph_def(graph_def, name='', return_elements=[
                    BOTTLENECK_TENSOR_NAME, JPEG_DATA_TENSOR_NAME,
                    RESIZED_INPUT_TENSOR_NAME]))
    return graph, bottleneck_tensor, jpeg_data_tensor, resized_input_tensor


def run_bottleneck_on_image(sess, image_data, image_data_tensor, bottleneck_tensor):

  bottleneck_values = sess.run(
        bottleneck_tensor,
        {image_data_tensor: image_data})
    bottleneck_values = np.squeeze(bottleneck_values)
    return bottleneck_values
```

```
def maybe_download_and_extract():

    dest_directory = FLAGS.model_dir
    if not os.path.exists(dest_directory):
        os.makedirs(dest_directory)
    filename = DATA_URL.split('/')[-1]
    filepath = os.path.join(dest_directory, filename)
    if not os.path.exists(filepath):
        def _progress(count, block_size, total_size):
            sys.stdout.write('\r>> Downloading %s %.1f%%' %
                       (filename,
                        float(count * block_size) / float(total_size) * 100.0))
            sys.stdout.flush()

        filepath, _ = urllib.request.urlretrieve(DATA_URL, filepath, _progress)
        print()
        statinfo = os.stat(filepath)
        print('Successfully downloaded', filename, statinfo.st_size, 'bytes.')
    tarfile.open(filepath, 'r:gz').extractall(dest_directory)


def ensure_dir_exists(dir_name):

    if not os.path.exists(dir_name):
        os.makedirs(dir_name)


def write_list_of_floats_to_file(list_of_floats, file_path):

    s = struct.pack('d' * BOTTLENECK_TENSOR_SIZE, *list_of_floats)
    with open(file_path, 'wb') as f:
        f.write(s)


def read_list_of_floats_from_file(file_path):
    with open(file_path, 'rb') as f:
        s = struct.unpack('d' * BOTTLENECK_TENSOR_SIZE, f.read())
        return list(s)


bottleneck_path_2_bottleneck_values = {}


def create_bottleneck_file(bottleneck_path, image_lists, label_name, index,
                           image_dir, category, sess, jpeg_data_tensor,
                           bottleneck_tensor):
```

```
    print('Creating bottleneck at ' + bottleneck_path)
    image_path = get_image_path(image_lists, label_name, index,
                                image_dir, category)
    if not gfile.Exists(image_path):
        tf.logging.fatal('File does not exist %s', image_path)
    image_data =  gfile.FastGFile(image_path,  'rb').read()
    try:
        bottleneck_values = run_bottleneck_on_image(
            sess, image_data, jpeg_data_tensor, bottleneck_tensor)
    except:
        raise RuntimeError('Error during processing file %s' % image_path)

    bottleneck_string = ','.join(str(x) for x in bottleneck_values)
    with open(bottleneck_path, 'w') as bottleneck_file:
        bottleneck_file.write(bottleneck_string)


def get_or_create_bottleneck(sess, image_lists, label_name, index, image_dir,
                             category, bottleneck_dir, jpeg_data_tensor,
                             bottleneck_tensor):

    label_lists = image_lists[label_name]
    sub_dir = label_lists['dir']
    sub_dir_path = os.path.join(bottleneck_dir, sub_dir)
    ensure_dir_exists(sub_dir_path)
    bottleneck_path = get_bottleneck_path(image_lists, label_name, index,
                                          bottleneck_dir, category)
    if not os.path.exists(bottleneck_path):
        create_bottleneck_file(bottleneck_path, image_lists, label_name, index,
                               image_dir, category, sess, jpeg_data_tensor,
                               bottleneck_tensor)
    with open(bottleneck_path, 'r') as bottleneck_file:
        bottleneck_string = bottleneck_file.read()
    did_hit_error = False
    try:
        bottleneck_values = [float(x) for x in bottleneck_string.split(',')]
    except ValueError:
        print('Invalid float found, recreating bottleneck')
        did_hit_error = True
    if did_hit_error:
        create_bottleneck_file(bottleneck_path, image_lists, label_name, index,
                               image_dir, category, sess, jpeg_data_tensor,
                               bottleneck_tensor)
        with open(bottleneck_path, 'r') as bottleneck_file:
            bottleneck_string = bottleneck_file.read()
            bottleneck_values = [float(x) for x in bottleneck_string.split(',')]
    return bottleneck_values
```

```
def cache_bottlenecks(sess, image_lists, image_dir, bottleneck_dir,
                      jpeg_data_tensor, bottleneck_tensor):
    how_many_bottlenecks = 0
    ensure_dir_exists(bottleneck_dir)
    for label_name, label_lists in image_lists.items():
        for category in ['training', 'testing', 'validation']:
            category_list = label_lists[category]
            for index, unused_base_name in enumerate(category_list):
                get_or_create_bottleneck(sess, image_lists, label_name, index,
                                 image_dir, category, bottleneck_dir,
                                 jpeg_data_tensor, bottleneck_tensor)

                how_many_bottlenecks += 1
                if how_many_bottlenecks % 100 == 0:
                    print(str(how_many_bottlenecks) + ' bottleneck files created.')


def get_random_cached_bottlenecks(sess, image_lists, how_many, category,
                                  bottleneck_dir, image_dir, jpeg_data_tensor,
                                  bottleneck_tensor):

    class_count = len(image_lists.keys())
    bottlenecks = []
    ground_truths = []
    filenames = []
    if how_many >= 0:
        for unused_i in range(how_many):
            label_index = random.randrange(class_count)
            label_name = list(image_lists.keys())[label_index]
            image_index  =  random.randrange(MAX_NUM_IMAGES_PER_CLASS  +  1)
            image_name = get_image_path(image_lists, label_name, image_index,
                                  image_dir, category)
            bottleneck = get_or_create_bottleneck(sess, image_lists, label_name,
                                       image_index, image_dir, category,
                                       bottleneck_dir, jpeg_data_tensor,
                                       bottleneck_tensor)
            ground_truth = np.zeros(class_count, dtype=np.float32)
            ground_truth[label_index] = 1.0
            bottlenecks.append(bottleneck)
            ground_truths.append(ground_truth)
            filenames.append(image_name)
    else:
        for label_index, label_name in enumerate(image_lists.keys()):
            for image_index, image_name in enumerate(
                image_lists[label_name][category]):
                image_name = get_image_path(image_lists, label_name, image_index,
                                    image_dir, category)
                bottleneck = get_or_create_bottleneck(sess, image_lists, label_name,
                                         image_index, image_dir, category,
```

```
                                              bottleneck_dir, jpeg_data_tensor,
                                              bottleneck_tensor)
                ground_truth = np.zeros(class_count, dtype=np.float32)
                ground_truth[label_index] = 1.0
                bottlenecks.append(bottleneck)
                ground_truths.append(ground_truth)
                filenames.append(image_name)
    return bottlenecks, ground_truths, filenames


def get_random_distorted_bottlenecks(
    sess, image_lists, how_many, category, image_dir, input_jpeg_tensor,
    distorted_image, resized_input_tensor, bottleneck_tensor):
    class_count = len(image_lists.keys())
    bottlenecks = []
    ground_truths = []
    for unused_i in range(how_many):
        label_index = random.randrange(class_count)
        label_name = list(image_lists.keys())[label_index]
        image_index = random.randrange(MAX_NUM_IMAGES_PER_CLASS + 1)
        image_path = get_image_path(image_lists, label_name, image_index, image_dir,
                                    category)
        if not gfile.Exists(image_path):
            tf.logging.fatal('File does not exist %s', image_path)
        jpeg_data = gfile.FastGFile(image_path, 'rb').read()
        distorted_image_data = sess.run(distorted_image,
                                        {input_jpeg_tensor: jpeg_data})
        bottleneck = run_bottleneck_on_image(sess, distorted_image_data,
                                             resized_input_tensor,
                                             bottleneck_tensor)
        ground_truth = np.zeros(class_count, dtype=np.float32)
        ground_truth[label_index] = 1.0
        bottlenecks.append(bottleneck)
        ground_truths.append(ground_truth)
    return bottlenecks, ground_truths


def should_distort_images(flip_left_right, random_crop, random_scale,
                          random_brightness):

    return (flip_left_right or (random_crop != 0) or (random_scale != 0) or
          (random_brightness != 0))


def add_input_distortions(flip_left_right, random_crop, random_scale,
                          random_brightness):


    jpeg_data = tf.placeholder(tf.string, name='DistortJPGInput')
```

```
    decoded_image = tf.image.decode_jpeg(jpeg_data, channels=MODEL_INPUT_DEPTH)
    decoded_image_as_float = tf.cast(decoded_image, dtype=tf.float32)
    decoded_image_4d = tf.expand_dims(decoded_image_as_float, 0)
    margin_scale = 1.0 + (random_crop / 100.0)
    resize_scale = 1.0 + (random_scale / 100.0)
    margin_scale_value = tf.constant(margin_scale)
    resize_scale_value = tf.random_uniform(tensor_shape.scalar(),
                                           minval=1.0,
                                           maxval=resize_scale)
    scale_value = tf.multiply(margin_scale_value, resize_scale_value)
    precrop_width = tf.multiply(scale_value, MODEL_INPUT_WIDTH)
    precrop_height = tf.multiply(scale_value, MODEL_INPUT_HEIGHT)
    precrop_shape = tf.stack([precrop_height, precrop_width])
    precrop_shape_as_int = tf.cast(precrop_shape, dtype=tf.int32)
    precropped_image = tf.image.resize_bilinear(decoded_image_4d,
                                                precrop_shape_as_int)
    precropped_image_3d = tf.squeeze(precropped_image, squeeze_dims=[0])
    cropped_image = tf.random_crop(precropped_image_3d,
                                   [MODEL_INPUT_HEIGHT, MODEL_INPUT_WIDTH,
                                    MODEL_INPUT_DEPTH])
    if flip_left_right:
        flipped_image = tf.image.random_flip_left_right(cropped_image)
    else:
        flipped_image = cropped_image
    brightness_min = 1.0 - (random_brightness / 100.0)
    brightness_max = 1.0 + (random_brightness / 100.0)
    brightness_value = tf.random_uniform(tensor_shape.scalar(),
                                         minval=brightness_min,
                                         maxval=brightness_max)
    brightened_image = tf.multiply(flipped_image, brightness_value)
    distort_result = tf.expand_dims(brightened_image, 0, name='DistortResult')
    return jpeg_data, distort_result


def variable_summaries(var):
    """Attach a lot of summaries to a Tensor (for TensorBoard visualization)."""
    with tf.name_scope('summaries'):
        mean = tf.reduce_mean(var)
        tf.summary.scalar('mean', mean)
        with tf.name_scope('stddev'):
            stddev = tf.sqrt(tf.reduce_mean(tf.square(var - mean)))
        tf.summary.scalar('stddev', stddev)
        tf.summary.scalar('max', tf.reduce_max(var))
        tf.summary.scalar('min', tf.reduce_min(var))
        tf.summary.histogram('histogram', var)


def add_final_training_ops(class_count, final_tensor_name, bottleneck_tensor):
```

```
    with tf.name_scope('input'):
        bottleneck_input = tf.placeholder_with_default(
                bottleneck_tensor, shape=[None, BOTTLENECK_TENSOR_SIZE],
                name='BottleneckInputPlaceholder')

        ground_truth_input = tf.placeholder(tf.float32,
                                            [None, class_count],
                                            name='GroundTruthInput')

    # Organizing the following ops as `final_training_ops` so they're easier
    # to see in TensorBoard
    layer_name = 'final_training_ops'
    with tf.name_scope(layer_name):
        with tf.name_scope('weights'):
            initial_value = tf.truncated_normal([BOTTLENECK_TENSOR_SIZE,
class_count],
                                                stddev=0.001)

            layer_weights = tf.Variable(initial_value, name='final_weights')

            variable_summaries(layer_weights)
        with tf.name_scope('biases'):
            layer_biases = tf.Variable(tf.zeros([class_count]), name='final_biases')
            variable_summaries(layer_biases)
        with tf.name_scope('Wx_plus_b'):
            logits = tf.matmul(bottleneck_input, layer_weights) + layer_biases
            tf.summary.histogram('pre_activations', logits)

    final_tensor = tf.nn.softmax(logits, name=final_tensor_name)
    tf.summary.histogram('activations', final_tensor)

    with tf.name_scope('cross_entropy'):
        cross_entropy = tf.nn.softmax_cross_entropy_with_logits(
                labels=ground_truth_input, logits=logits)
        with tf.name_scope('total'):
            cross_entropy_mean = tf.reduce_mean(cross_entropy)
    tf.summary.scalar('cross_entropy', cross_entropy_mean)

    with tf.name_scope('train'):
        optimizer = tf.train.GradientDescentOptimizer(FLAGS.learning_rate)
        train_step = optimizer.minimize(cross_entropy_mean)

    return (train_step, cross_entropy_mean, bottleneck_input, ground_truth_input,
            final_tensor)


def add_evaluation_step(result_tensor, ground_truth_tensor):
```

```
    with tf.name_scope('accuracy'):
        with tf.name_scope('correct_prediction'):
            prediction = tf.argmax(result_tensor, 1)
            correct_prediction = tf.equal(
                    prediction, tf.argmax(ground_truth_tensor, 1))
        with tf.name_scope('accuracy'):
            evaluation_step = tf.reduce_mean(tf.cast(correct_prediction,
tf.float32))
    tf.summary.scalar('accuracy', evaluation_step)
    return evaluation_step, prediction


def main(_):

    if tf.gfile.Exists(FLAGS.summaries_dir):
        tf.gfile.DeleteRecursively(FLAGS.summaries_dir)
    tf.gfile.MakeDirs(FLAGS.summaries_dir)
    maybe_download_and_extract()
    graph, bottleneck_tensor, jpeg_data_tensor, resized_image_tensor = (
            create_inception_graph())

    image_lists = create_image_lists(FLAGS.image_dir, FLAGS.testing_percentage,
                                     FLAGS.validation_percentage)
    class_count = len(image_lists.keys())
    if class_count == 0:
        print('No valid folders of images found at ' + FLAGS.image_dir)
        return -1
    if class_count == 1:
        print('Only one valid folder of images found at ' + FLAGS.image_dir +
              ' - multiple classes are needed for classification.')
        return -1
    do_distort_images = should_distort_images(
            FLAGS.flip_left_right, FLAGS.random_crop, FLAGS.random_scale,
            FLAGS.random_brightness)

    with tf.Session(graph=graph) as sess:

        if do_distort_images:
                 (distorted_jpeg_data_tensor,
             distorted_image_tensor) = add_input_distortions(
                     FLAGS.flip_left_right, FLAGS.random_crop,
                     FLAGS.random_scale, FLAGS.random_brightness)
        else:
            cache_bottlenecks(sess, image_lists, FLAGS.image_dir,
                        FLAGS.bottleneck_dir, jpeg_data_tensor,
                        bottleneck_tensor)
                (train_step, cross_entropy, bottleneck_input, ground_truth_input,
         final_tensor) = add_final_training_ops(len(image_lists.keys()),
```

```
                                    FLAGS.final_tensor_name,
                                    bottleneck_tensor)
        evaluation_step, prediction = add_evaluation_step(
                final_tensor, ground_truth_input)
        merged = tf.summary.merge_all()
        train_writer = tf.summary.FileWriter(FLAGS.summaries_dir + '/train',
                                             sess.graph)

        validation_writer = tf.summary.FileWriter(
                FLAGS.summaries_dir + '/validation')
        init = tf.global_variables_initializer()
        sess.run(init)
        for i in range(FLAGS.how_many_training_steps):
            if do_distort_images:
                (train_bottlenecks,
                 train_ground_truth) = get_random_distorted_bottlenecks(
                         sess, image_lists, FLAGS.train_batch_size, 'training',
                         FLAGS.image_dir, distorted_jpeg_data_tensor,
                         distorted_image_tensor, resized_image_tensor,
bottleneck_tensor)
            else:
                (train_bottlenecks,
                 train_ground_truth, _) = get_random_cached_bottlenecks(
                         sess, image_lists, FLAGS.train_batch_size, 'training',
                         FLAGS.bottleneck_dir, FLAGS.image_dir, jpeg_data_tensor,
                         bottleneck_tensor)
            train_summary, _ = sess.run(
                    [merged, train_step],
                    feed_dict={bottleneck_input: train_bottlenecks,
                               ground_truth_input: train_ground_truth})
            train_writer.add_summary(train_summary, i)

            is_last_step = (i + 1 == FLAGS.how_many_training_steps)
            if (i % FLAGS.eval_step_interval) == 0 or is_last_step:
                train_accuracy, cross_entropy_value = sess.run(
                        [evaluation_step, cross_entropy],
                        feed_dict={bottleneck_input: train_bottlenecks,
                                   ground_truth_input: train_ground_truth})
                validation_bottlenecks, validation_ground_truth, _ = (
                        get_random_cached_bottlenecks(
                                sess, image_lists, FLAGS.validation_batch_size,
'validation',
                                FLAGS.bottleneck_dir, FLAGS.image_dir,
jpeg_data_tensor,
                                bottleneck_tensor))
                validation_summary, validation_accuracy = sess.run(
                        [merged, evaluation_step],
                        feed_dict={bottleneck_input: validation_bottlenecks,
```

```
                                   ground_truth_input: validation_ground_truth})
                validation_writer.add_summary(validation_summary, i)
                print('Step: %d, Train accuracy: %.4f%%, Cross entropy: %f,
Validation accuracy: %.1f%% (N=%d)' % (i,
                        train_accuracy * 100, cross_entropy_value,
validation_accuracy * 100, len(validation_bottlenecks)))

        # We've completed all our training, so run a final test evaluation on
        # some new images we haven't used before.
        test_bottlenecks, test_ground_truth, test_filenames = (
                get_random_cached_bottlenecks(sess, image_lists,
FLAGS.test_batch_size,
                                      'testing', FLAGS.bottleneck_dir,
                                      FLAGS.image_dir, jpeg_data_tensor,
                                      bottleneck_tensor))
        test_accuracy, predictions = sess.run(
                [evaluation_step, prediction],
                feed_dict={bottleneck_input: test_bottlenecks,
                           ground_truth_input: test_ground_truth})
        print('Final test accuracy = %.1f%% (N=%d)' % (
                test_accuracy * 100, len(test_bottlenecks)))

        if FLAGS.print_misclassified_test_images:
            print('=== MISCLASSIFIED TEST IMAGES ===')
            for i, test_filename in enumerate(test_filenames):
                if predictions[i] != test_ground_truth[i].argmax():
                    print('%70s  %s' % (test_filename,
                              list(image_lists.keys())[predictions[i]]))

        # Write out the trained graph and labels with the weights stored as
        # constants.
        output_graph_def = graph_util.convert_variables_to_constants(
                sess, graph.as_graph_def(), [FLAGS.final_tensor_name])
        with gfile.FastGFile(FLAGS.output_graph, 'wb') as f:
            f.write(output_graph_def.SerializeToString())
        with gfile.FastGFile(FLAGS.output_labels, 'w') as f:
            f.write('\n'.join(image_lists.keys()) + '\n')


if___name___== '__main__':
    parser = argparse.ArgumentParser()
    parser.add_argument(
        '--image_dir',
        type=str,
        default='dataset',
        help='Path to folders of labeled images.'
        )
    parser.add_argument(
        '--output_graph',
```

```
        type=str,
        default='logs/output_graph.pb',
        help='Where to save the trained graph.'
        )
    parser.add_argument(
        '--output_labels',
        type=str,
        default='logs/output_labels.txt',
        help='Where to save the trained graph\'s labels.'
        )
    parser.add_argument(
        '--summaries_dir',
        type=str,
        default='logs/retrain_logs',
        help='Where to save summary logs for TensorBoard.'
        )
    parser.add_argument(
        '--how_many_training_steps',
        type=int,
        default=1000,
        help='How many training steps to run before ending.'
        )
    parser.add_argument(
        '--learning_rate',
        type=float,
        default=0.01,
        help='How large a learning rate to use when training.'
        )
    parser.add_argument(
        '--testing_percentage',
        type=int,
        default=10,
        help='What percentage of images to use as a test set.'
        )
    parser.add_argument(
        '--validation_percentage',
        type=int,
        default=10,
        help='What percentage of images to use as a validation set.'
        )
    parser.add_argument(
        '--eval_step_interval',
        type=int,
        default=100,
        help='How often to evaluate the training results.'
        )
    parser.add_argument(
        '--train_batch_size',
```

```
        type=int,
        default=100,
        help='How many images to train on at a time.'
        )
    parser.add_argument(
        '--test_batch_size',
        type=int,
        default=-1,
        help=
        )
    parser.add_argument(
        '--validation_batch_size',
        type=int,
        default=100,
        help=
        )
    parser.add_argument(
        '--print_misclassified_test_images',
        default=False,
        help=
,
        action='store_true'
        )
    parser.add_argument(
        '--model_dir',
        type=str,
        default='logs/imagenet',
        help=
        )
    parser.add_argument(
        '--bottleneck_dir',
        type=str,
        default='/tmp/bottleneck',
        help='Path to cache bottleneck layer values as files.'
        )
    parser.add_argument(
        '--final_tensor_name',
        type=str,
        default='final_result',
        help=
        )
    parser.add_argument(
        '--flip_left_right',
        default=False,
        help=
,
        action='store_true'
        )
```

```
    parser.add_argument(
        '--random_crop',
        type=int,
        default=0,
        help=
        )
    parser.add_argument(
        '--random_scale',
        type=int,
        default=0,
        help=
        )
    parser.add_argument(
        '--random_brightness',
        type=int,
        default=0,
        help=
        )
    FLAGS, unparsed = parser.parse_known_args()
    tf.app.run(main=main, argv=[sys.argv[0]] + unparsed)
```

classify.py

```
import tensorflow as tf
import sys
import os
import cv2
totalcnt1=0
def main(img,plastic):
    global totalcnt1
    cnt=0
    # Disable tensorflow compilation warnings
    os.environ['TF_CPP_MIN_LOG_LEVEL']='2'
    import tensorflow as tf

    image_path = img#"cardboard1.jpg"
    #image_path=file

    # Read the image_data
    image_data = tf.io.gfile.GFile(image_path, 'rb').read()


    # Loads label file, strips off carriage return
    label_lines = [line.rstrip() for line
                       in tf.io.gfile.GFile("logs/output_labels.txt")]

    # Unpersists graph from file
    with tf.io.gfile.GFile("logs/output_graph.pb", 'rb') as f:
```

```
        graph_def = tf.compat.v1.GraphDef()
        graph_def.ParseFromString(f.read())
        _ = tf.import_graph_def(graph_def, name='')

    with tf.compat.v1.Session() as sess:
        # Feed the image_data as input to the graph and get first prediction
        softmax_tensor = sess.graph.get_tensor_by_name('final_result:0')

        predictions = sess.run(softmax_tensor, \
                 {'DecodeJpeg/contents:0': image_data})

        # Sort to show labels of first prediction in order of confidence
        top_k = predictions[0].argsort()[-len(predictions[0]):][::-1]

        for node_id in top_k:
            human_string = label_lines[node_id]
            #print(human_string)
            score = predictions[0][node_id]
            if cnt==0:
                res=label_lines[node_id]
                res2=score*100
                res2=str(res2)
                res2=res2[0:4]
                cnt=cnt+1
                print ("prediction is ",res," with score ",res2)
                if res in plastic:
                    print(str(img)[:-4]+" is Plastic")
                    totalcnt1=totalcnt1+1
                    res="plastic"
                else:
                    print(str(img)[:-4]+" is Non Plastic")
                    res="Non Plastic"
            #print('%s (score = %.5f)' % (human_string, score))
        return res,res2
plastic=["plastic","plastic covers","plastic tubs"]
j=0
tot=0
for item in ["Bag-in-water-
2.jpg","cardboard1.jpg","glass20.jpg","metal54.jpg","paper142.jpg","pastic_can.jpg",
"plastic75.jpg","trash13.jpg"]:
    print("")
    tot=tot+1
    print("testing ",item)
    res,res2=main(item,plastic)
    frame=cv2.imread(item)
    font = cv2.FONT_HERSHEY_SIMPLEX

    # org
```

```
    org = (5,20)

    # fontScale
    fontScale = 1

    # Blue color in BGR
    color = (255, 0, 0)

    # Line thickness of 2 px
    thickness = 1

    # Using cv2.putText() method
    frame = cv2.putText(frame, 'Category: '+res, org, font,
                        fontScale, color, thickness, cv2.LINE_AA)
    color = (0, 255, 0)
    org = (5, 50)
    frame = cv2.putText(frame, 'Plastic Count : '+str(totalcnt1), org, font,
                        fontScale, color, thickness, cv2.LINE_AA)
    cv2.imshow(str(item)[:-4],frame)
    cv2.waitKey(0)
print("total count of plastic is:",totalcnt1)
print("total number of images tested is:",tot)
print("total plastic count percentage is
",totalcnt1,"/",tot,"=",(totalcnt1/tot)*100,"%")
result="total plastic count percentage is
"+str(totalcnt1)+"/"+str(tot)+"="+str((totalcnt1/tot)*100)+" %"
frame=cv2.imread("result.jpg")
font = cv2.FONT_HERSHEY_SIMPLEX

# org
org = (5, 30)

# fontScale
fontScale = 1

# Blue color in BGR
color = (255, 0, 0)

# Line thickness of 2 px
thickness = 2

# Using cv2.putText() method
frame = cv2.putText(frame, "total plastic cnt: "+str(totalcnt1), org, font,
                    fontScale, color, thickness, cv2.LINE_AA)
color = (0, 255, 0)
org = (5, 90)
frame = cv2.putText(frame, "total images tested: "+str(tot), org, font,
                    fontScale, color, thickness, cv2.LINE_AA)
```

```
color = (0, 0, 255)
org = (5, 150)
frame = cv2.putText(frame, result, org, font,
                   fontScale, color, thickness, cv2.LINE_AA)
cv2.imshow("final result",frame)
cv2.waitKey(0)
```