WASTE MANAGEMENT USING COMPUTER VISION

A report submitted in partial fulfillment of the requirements for

the award of the degree of

Bachelor of Technology

in

ELECTRONICS AND COMMUNICATION DEPARTMENT

By

HARGOVIND SINGH (ECE16U008)

NIPUN HALDAR (ECE16U014)

P R RAHUL HEBBAR (ECE16U017)

under the guidance of

Dr. M. Senthil Sivakumar

ELECTRONICS AND COMMUNICATION

INDIAN INSTITUTE OF INFORMATION TECHNOLOGY TIRUCHIRAPPALLI

TIRUCHIRAPPALLI – 620015.

JUNE 2020

BONAFIDE CERTIFICATE

This is to certify that the project work titled **"Waste Classification Using Computer Vision"** is a bonafide record of the work done by

P R RAHUL HEBBAR (ECE16U017)

NIPUN HALDAR (ECE16U014)

HARGOVIND SINGH (ECE16U008)

in partial fulfillment of the requirements for the award of the degree of Bachelor of Technology in Electronics and Communication Department of the Indian Institute of Information Technology Tiruchirappalli during the year 2016-2020. The contents of this report, in full or in parts, have not been submitted to any other institute or university for the award of any degree or diploma.

Dr. M. Senthil Sivakumar	**Dr. G. Seetharaman**
Asst. Professor	**Head of the Department**
Supervisor	

Project viva-voce held on _____

Internal Examiner	**External Examiner**

ABSTRACT

Municipal solid waste management (MSWM) is one of the major environmental problems of Indian cities. Improper management of municipal solid waste (MSW) causes hazards to inhabitants. Various studies reveal that about 90% of MSW is disposed of unscientifically in open dumps and landfills, creating problems to public health and the environment.

This report entails the integration of various new technologies like Internet of Things, Deep Learning and Machine Learning for smart waste management. Machine learning provides effective solutions, such as regression, classification, clustering, and correlation rules perception for IoT-based waste management. This project will focus on using image classification using a ground up built convolutional neural network model and transfer learning on an already pretrained model from Google. We will use the model built of higher accuracy to then integrate it with a hardware that can take pictures and classify the pictures as biodegradable or nonbiodegradable.

Keywords: **Machine learning, Internet of Things, Embedded Systems, Deep Learning, Computer Vision.**

Acknowledgement

I/We wish to record my deep sense of gratitude and profound thanks to my/our project supervisor **Dr. M. Senthil Sivakumar**, Assistant/Associate Professor, Electronics and Communication Department , Indian Institute of Information Technology Tiruchirappalli for his/her keen interest, inspiring guidance, constant encouragement with my/our project work during all stages, to bring this project report into fruition.

I/We thank **Dr. Narasimha Sarma N.V.S**, Director, Indian Institute of Information Technology Tiruchirappalli and **Dr. G. Seetharaman**, Associate Professor and Head, Department of Electronics and Communication, Indian Institute of Information Technology Tiruchirappalli for providing me/us all the facilities to complete my/our project work.

I/We also thank the faculty and non-teaching staff members of the Electronics and Communication Engineering, Indian Institute of Information Technology Tiruchirappalli for their valuable support throughout the course of my/our project work.

HARGOVIND SINGH

NIPUN HALDAR

P R RAHUL HEBBAR

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF ABBREVIATIONS

S. NO.	TERMS	ABBREVIATION
1.	Machine Learning	ML
2.	Convolutional Neural Networks	CNN
3.	Residual Neural Networks	ResNET
4.	Transfer Learning	TL
5.	Internet of Things	IoT
6.	Rectified Linear Unit	ReLU
7.	Artificial Intelligence	AI
8.	Metal-oxide-semiconductor field-effect transistor	MOSFET
9.	Radio Frequency Identification	RFID
10.	Input-Output	I/O
11.	Infrared	IR
12.	Camera Serial Interface	CSI

CHAPTER 1

INTRODUCTION

1.1 Deep Learning

Deep learning is part of a broader family of machine learning methods based on artificial neural networks with representation learning. Learning can be supervised, semi-supervised or unsupervised.

The adjective "deep" in deep learning comes from the use of multiple layers in the network. Early work showed that a linear perceptron cannot be a universal classifier, and that a network with a nonpolynomial activation function with one hidden layer of unbounded width can on the other hand be. Deep learning is a modern variation which is concerned with an unbounded number of layers of bounded size, which permits practical application and optimized implementation, while retaining theoretical universality under mild conditions. In deep learning the layers are also permitted to be heterogeneous and to deviate widely from biologically informed connectionist models, for the sake of efficiency, trainability and understandability, whence the "structured" part.

Andrew Ng from Coursera and Chief Scientist at Baidu Research formally founded Google Brain that eventually resulted in the productization of deep learning technologies across a large number of Google services. He has spoken and written a lot about what deep learning is and is a good place to start. In early talks on deep learning, Andrew described deep learning in the context of traditional artificial neural networks. In the 2013 talk titled “Deep Learning, Self-Taught Learning and Unsupervised Feature Learning” he described the idea of deep learning as:
“Using brain simulations, hope to:
– Make learning algorithms much better and easier to use.
– Make revolutionary advances in machine learning and AI.
I believe this is our best shot at progress towards real AI”
Later his comments became more nuanced. The core of deep learning according to Andrew is that we now have fast enough computers and enough data to actually train large neural networks. When discussing why now is the time that deep learning is taking off at ExtractConf 2015 in a talk titled “What data scientists should know about deep learning”, he commented:
“very large neural networks we can now have and ... huge amounts of data that we have access to”

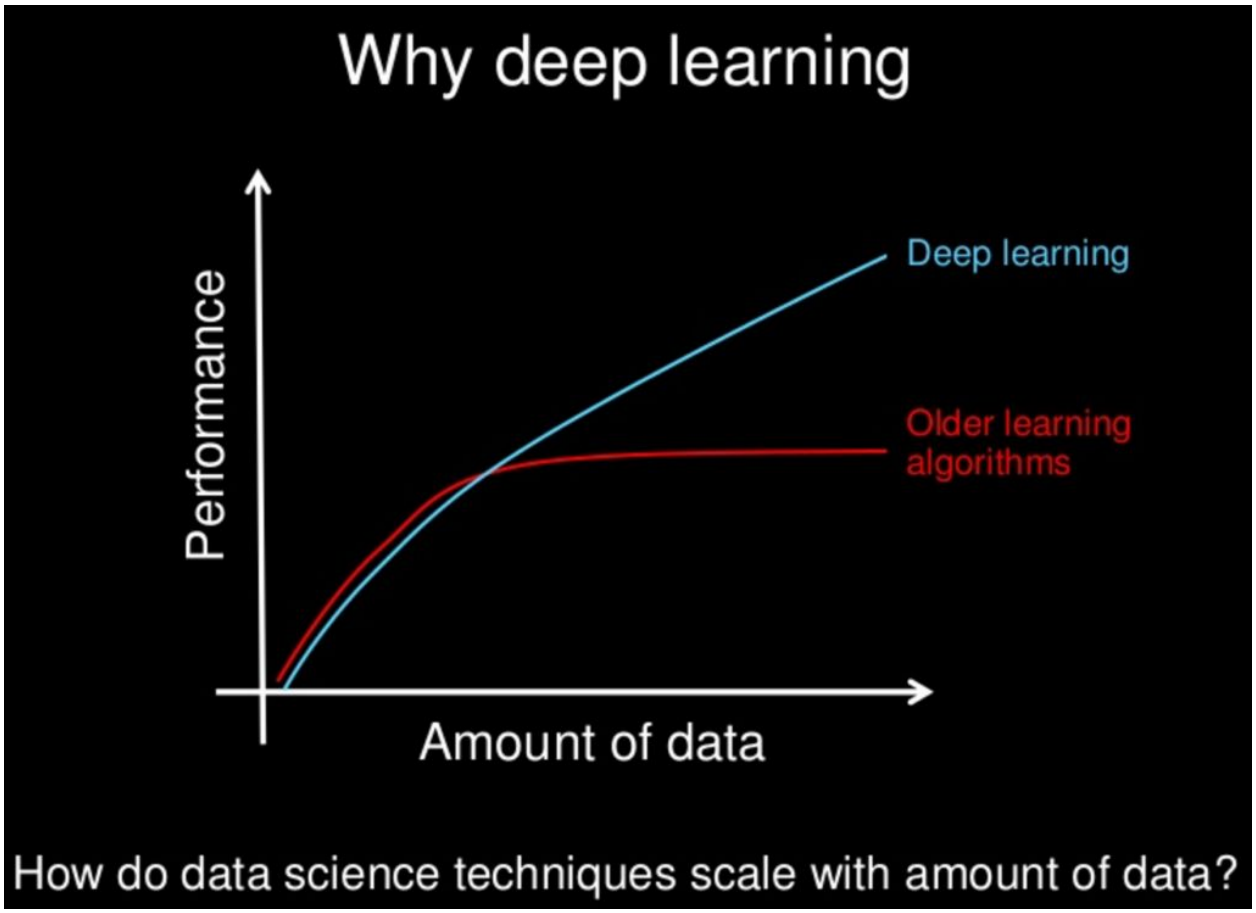

Fig 1. Amount of data affecting deep learning algorithms

1.1.1 Computer Vision

Computer vision is an interdisciplinary scientific field that deals with how computers can gain high-level understanding from digital images or videos. From the perspective of engineering, it seeks to understand and automate tasks that the human visual system can do.

The goal of computer vision is to understand the content of digital images. Typically, this involves developing methods that attempt to reproduce the capability of human vision.

Currently, the best algorithms for such tasks are based on convolutional neural networks. An illustration of their capabilities is given by the ImageNet Large Scale Visual Recognition Challenge; this is a benchmark in object classification and detection, with millions of images and hundreds of object classes. Performance of convolutional neural networks, on the ImageNet tests, is now close to that of humans. The best algorithms still struggle with objects that are small or thin, such as a small ant on a stem of a flower or a person holding a quill in their hand. They also have trouble with images that have been distorted with filters (an increasingly common phenomenon with modern

digital cameras). By contrast, those kinds of images rarely trouble humans. Humans, however, tend to have trouble with other issues. For example, they are not good at classifying objects into fine-grained classes, such as the particular breed of dog or species of bird, whereas convolutional neural networks handle this with ease.

Fig 2. Object recognition using computer vision

1.1.2 Convolutional Neural Network (CNN)

In deep learning, a convolutional neural network (CNN, or ConvNet) is a class of deep neural networks, most commonly applied to analyzing visual imagery. They are also known as shift invariant or space invariant artificial neural networks (SIANN), based on their shared-weights architecture and translation invariance characteristics. They have applications in image and video recognition, recommender systems, image classification, medical image analysis, natural language processing, and financial time series.

CNNs are regularized versions of multilayer perceptrons. Multilayer perceptrons usually mean fully connected networks, that is, each neuron in one layer is connected to all neurons in the next layer. The "fully-connectedness" of these networks makes them prone to overfitting data. Typical ways of regularization include adding some form of magnitude measurement of weights to the loss function. CNNs take a different approach towards regularization: they take advantage of the hierarchical pattern in data and assemble more complex patterns using smaller and simpler patterns. Therefore, on the scale of connectedness and complexity, CNNs are on the lower extreme.

CNNs use relatively little pre-processing compared to other image classification algorithms. This means that the network learns the filters that in traditional algorithms

were hand-engineered. This independence from prior knowledge and human effort in feature design is a major advantage.

A convolutional neural network consists of an input and an output layer, as well as multiple hidden layers. The hidden layers of a CNN typically consist of a series of convolutional layers that convolve with a multiplication or other dot product. The activation function is commonly a RELU layer, and is subsequently followed by additional convolutions such as pooling layers, fully connected layers and normalization layers, referred to as hidden layers because their inputs and outputs are masked by the activation function and final convolution.

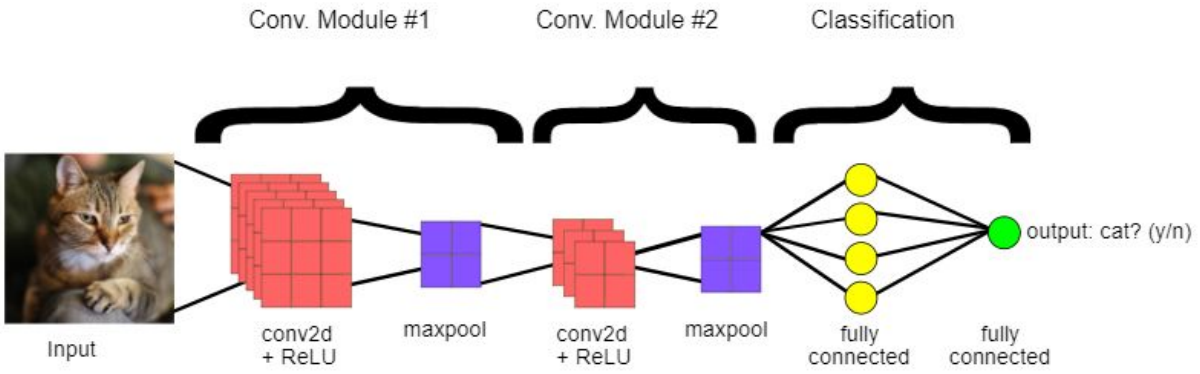

Fig 3. Architecture of Convolutional Neural Network

i. *Convolutional Layer* - When programming a CNN, the input is a tensor with shape (number of images) x (image height) x (image width) x (image depth). Then after passing through a convolutional layer, the image becomes abstracted to a feature map, with shape (number of images) x (feature map height) x (feature map width) x (feature map channels). A convolutional layer within a neural network should have the following attributes:
    - Convolutional kernels defined by a width and height (hyper-parameters).
    - The number of input channels and output channels (hyper-parameter).
    - The depth of the Convolution filter (the input channels) must be equal to the number channels (depth) of the input feature map.

    Convolutional layers convolve the input and pass its result to the next layer. This is similar to the response of a neuron in the visual cortex to a specific stimulus. Each convolutional neuron processes data only for its receptive field. Although fully connected feedforward neural networks can be used to learn features as well as classify data, it is not practical to apply this architecture to images. A very high number of neurons would be necessary, even in a shallow (opposite of deep) architecture, due to the very large input sizes associated with images, where each pixel is a relevant variable. For instance, a fully connected layer for a (small) image of size 100 x 100 has 10,000 weights for each neuron in the second layer. The convolution operation brings a solution to this problem as it reduces the

number of free parameters, allowing the network to be deeper with fewer parameters. For instance, regardless of image size, tiling regions of size 5 x 5, each with the same shared weights, requires only 25 learnable parameters. By using regularized weights over fewer parameters, the vanishing gradient and exploding gradient problems seen during backpropagation in traditional neural networks are avoided.

ii. *Pooling* - Convolutional networks may include local or global pooling layers to streamline the underlying computation. Pooling layers reduce the dimensions of the data by combining the outputs of neuron clusters at one layer into a single neuron in the next layer. Local pooling combines small clusters, typically 2 x 2. Global pooling acts on all the neurons of the convolutional layer. In addition, pooling may compute a max or an average. Max pooling uses the maximum value from each of a cluster of neurons at the prior layer. Average pooling uses the average value from each of a cluster of neurons at the prior layer.

iii. *Fully connected* - Fully connected layers connect every neuron in one layer to every neuron in another layer. It is in principle the same as the traditional multi-layer perceptron neural network (MLP). The flattened matrix goes through a fully connected layer to classify the images.

iv. *Receptive field* - In neural networks, each neuron receives input from some number of locations in the previous layer. In a fully connected layer, each neuron receives input from every element of the previous layer. In a convolutional layer, neurons receive input from only a restricted subarea of the previous layer. Typically the subarea is of a square shape (e.g., size 5 by 5). The input area of a neuron is called its receptive field. So, in a fully connected layer, the receptive field is the entire previous layer. In a convolutional layer, the receptive area is smaller than the entire previous layer. The subarea of the original input image in the receptive field is increasingly growing as getting deeper in the network architecture. This is due to applying over and over again a convolution which takes into account the value of a specific pixel, but also some surrounding pixels.

v. *Weights* - Each neuron in a neural network computes an output value by applying a specific function to the input values coming from the receptive field in the previous layer. The function that is applied to the input values is determined by a vector of weights and a bias (typically real numbers). Learning, in a neural network, progresses by making iterative adjustments to these biases and weights.

The vector of weights and the bias are called filters and represent particular features of the input (e.g., a particular shape). A distinguishing feature of CNNs is that many neurons can share the same filter. This reduces memory footprint because a single bias and a single vector of weights are used across all receptive

fields sharing that filter, as opposed to each receptive field having its own bias and vector weighting.

1.1.2 Transfer Learning

Transfer learning (TL) is a research problem in machine learning (ML) that focuses on storing knowledge gained while solving one problem and applying it to a different but related problem. For example, knowledge gained while learning to recognize cars could apply when trying to recognize trucks. This area of research bears some relation to the long history of psychological literature on transfer of learning, although formal ties between the two fields are limited. From the practical standpoint, reusing or transferring information from previously learned tasks for the learning of new tasks has the potential to significantly improve the sample efficiency of a reinforcement learning agent.

Andrew Ng said in his NIPS 2016 tutorial that TL will be the next driver of ML commercial success after supervised learning to highlight the importance of TL.

In 1993, Lorien Pratt published a paper on transfer in machine learning, formulating the discriminability-based transfer (DBT) algorithm. In 1997, the journal Machine Learning published a special issue devoted to transfer learning, and by 1998, the field had advanced to include multi-task learning, along with a more formal analysis of its theoretical foundations. Learning to Learn, edited by Pratt and Sebastian Thrun, is a 1998 review of the subject.

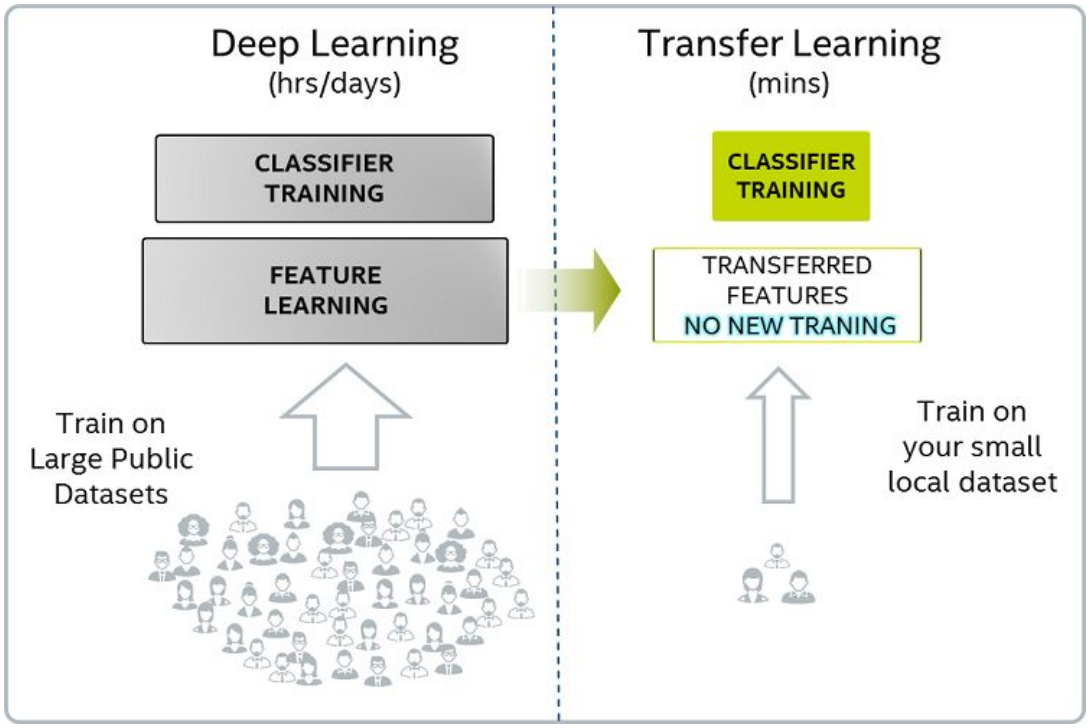

Fig 4.Transfer Learning

1.2 Internet of Things (IoT)

The Internet of things (IoT) is a system of interrelated computing devices, mechanical and digital machines provided with unique identifiers (UIDs) and the ability to transfer data over a network without requiring human-to-human or human-to-computer interaction.

The definition of the Internet of things has evolved due to the convergence of multiple technologies, real-time analytics, machine learning, commodity sensors, and embedded systems. Traditional fields of embedded systems, wireless sensor networks, control systems, automation (including home and building automation), and others all contribute to enabling the Internet of things. In the consumer market, IoT technology is most synonymous with products pertaining to the concept of the "smart home", covering devices and appliances (such as lighting fixtures, thermostats, home security systems and cameras, and other home appliances) that support one or more common ecosystems, and can be controlled via devices associated with that ecosystem, such as smartphones and smart speakers.

The main concept of a network of smart devices was discussed as early as 1982, with a modified Coca-Cola vending machine at Carnegie Mellon University becoming the first Internet-connected appliance, able to report its inventory and whether newly loaded drinks were cold or not. Mark Weiser's 1991 paper on ubiquitous computing, "The Computer of the 21st Century", as well as academic venues such as UbiComp and PerCom produced the contemporary vision of the IoT. In 1994, Reza Raji described the concept in IEEE Spectrum as "[moving] small packets of data to a large set of nodes, so as to integrate and automate everything from home appliances to entire factories". Between 1993 and 1997, several companies proposed solutions like Microsoft's at Work or Novell's NEST. The field gained momentum when Bill Joy envisioned device-to-device communication as a part of his "Six Webs" framework, presented at the World Economic Forum at Davos in 1999.

The term "Internet of things" was likely coined by Kevin Ashton of Procter & Gamble, later MIT's Auto-ID Center, in 1999, though he prefers the phrase "Internet for things". At that point, he viewed radio-frequency identification (RFID) as essential to the Internet of things, which would allow computers to manage all individual things.

Defining the Internet of things as "simply the point in time when more 'things or objects' were connected to the Internet than people", Cisco Systems estimated that the IoT was "born" between 2008 and 2009, with the things/people ratio growing from 0.08 in 2003 to 1.84 in 2010.

The key driving force behind the Internet of things is the MOSFET (metal-oxide-semiconductor field-effect transistor, or MOS transistor) which was

originally invented by Mohamed M. Atalla and Dawon Kahng at Bell Labs in 1959. The MOSFET is the basic building block of most modern electronics, including computers, smartphones, tablets and Internet services. MOSFET scaling miniaturization at a pace predicted by Dennard scaling and Moore's law has been the driving force behind technological advances in the electronics industry since the late 20th century. MOSFET scaling has been extended into the early 21st century with advances such as reducing power consumption, silicon-on-insulator (SOI) semiconductor device fabrication, and multi-core processor technology, leading up to the Internet of things, which is being driven by MOSFETs scaling down to nanoelectronic levels with reducing energy consumption.

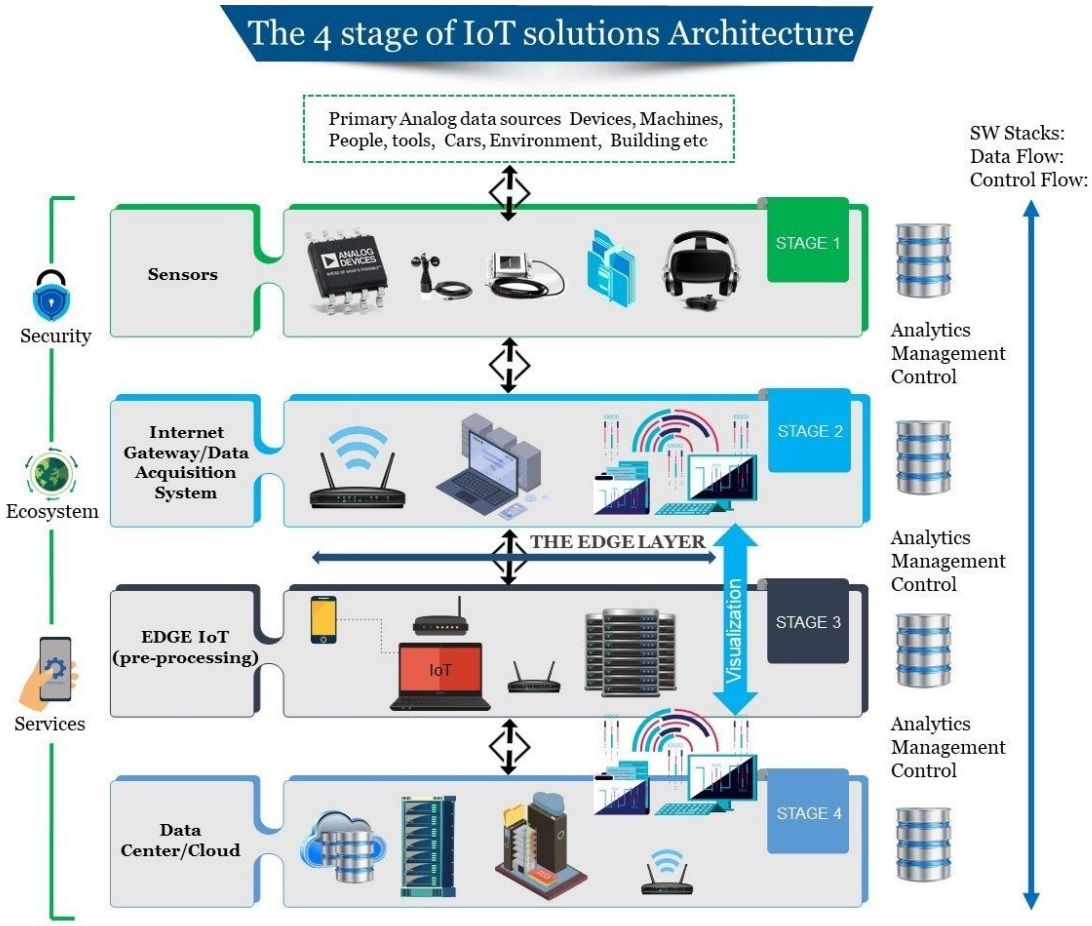

Fig 5. IoT Architecture

1.3 Motivation

Waste management is one of the primary problems that the whole world faces. As our civilization makes advancements we have exponentially more waste. The quantity of waste is so large that we can no longer just bury it or put them in landfills. So, proper waste management techniques need to be used to recycle and/or reuse the waste.

The first step in the process is to sort the waste into categories. If the waste is already sorted before reaching the dumpyard it is very easy to deal with it. That is where our project comes in, we sort the waste using computer vision during disposal. This will help to probably reduce the time it takes to sort it manually later on.

1.4 Objective of the Project

This project aims to create a smart waste classifier using computer vision and IoT . We will split the entire project into two sections, the hardware and the software.

In the software side we will be creating and training a convolutional neural network to classify an image into organic or recyclable. These predictions will be used to drive the hardware.

In the hardware side of the project, we will be using a development board with various network connectivity options like Bluetooth, WiFi or Ethernet such as RaspberryPi or NodeMCU. The board will have a camera connected to take images. Finally all that we have to do is to run the software on the hardware.

1.5 Organization of the project

For the software side, we will be using Google's open source library Tensorflow for handling datasets, data augmentation, building, training/retraining a convolutional neural network (CNN) and then use the model built to classify the images.

For the hardware, we will be using a Raspberry Pi development board because of the on-board ARM CPU and adequate RAM that can run the CNN model. Moreover for IoT applications it packs loads of connectivity options such as WiFi, Bluetooth and ethernet. It has various I/O options too, like USB ports, mini HDMI and the huge number of GPIO pins.

For seamless integration of our hardware and software, we will be using many other python libraries.

CHAPTER 2

LITERATURE REVIEW

2.1 Existing research works

2.1.1 Gradient-based learning applied to document recognition - Y. Lecun, L. Bottou, Y. Bengio and P. Haffner

Multilayer neural networks trained with the back-propagation algorithm constitute the best example of a successful gradient based learning technique. Given an appropriate network architecture, gradient-based learning algorithms can be used to synthesize a complex decision surface that can classify high-dimensional patterns, such as handwritten characters, with minimal preprocessing. This paper reviews various methods applied to handwritten character recognition and compares them on a standard handwritten digit recognition task. Convolutional neural networks, which are specifically designed to deal with the variability of two dimensional (2-D) shapes, are shown to outperform all other techniques.

Real-life document recognition systems are composed of multiple modules including field extraction, segmentation, recognition, and language modeling. A new learning paradigm, called graph transformer networks (GTN's), allows such multi-module systems to be trained globally using gradient-based methods so as to minimize an overall performance measure.

Two systems for online handwriting recognition are described. Experiments demonstrate the advantage of global training, and the flexibility of graph transformer networks.

A graph transformer network for reading a bank check is also described. It uses convolutional neural network character recognizers combined with global training techniques to provide record accuracy on business and personal checks. It is deployed commercially and reads several million checks per day.

Keywords - Convolutional neural networks, document recognition, finite state transducers, gradient-based learning, graph transformer networks, machine learning, neural networks, optical character recognition (OCR).

2.1.2 Simple convolutional neural network on image classification - T. Guo, J. Dong, H. Li and Y. Gao

In recent years, deep learning has been used in image classification, object tracking, pose estimation, text detection and recognition, visual saliency detection, action recognition and scene labeling. Among different types of models, Convolutional neural networks have demonstrated high performance on image classification. In this paper we builded a simple Convolutional neural network on image classification. On the basis of the Convolutional neural network, we also analyzed different methods of learning rate set and different optimization algorithms of solving the optimal parameters of the influence on image classification.

Image classification plays an important role in computer vision, it has a very important significance in our study, work and life. Image classification is process including image preprocessing, image segmentation, key feature extraction and matching identification. With the latest figures image classification techniques, we not only get the picture information faster than before, we apply it to scientific experiments, traffic identification, security, medical equipment, face recognition and other fields.

During the rise of deep learning, feature extraction and classifier has been integrated to a learning framework which overcomes the traditional method of feature selection difficulties. The idea of deep learning is to discover multiple levels of representation, with the hope that high-level features represent more abstract semantics of the data. One key ingredient of deep learning in image classification is the use of Convolutional architectures.

Keywords - Convolutional neural network; Deep learning; Image classification; learning rate; parametric solution

2.1.3 A case study on transfer learning in convolutional neural networks - C. D. Gürkaynak and N. Arica

In this work, a case study is performed on transfer learning approach in convolutional neural networks. Transfer learning parameters are examined on AlexNet, VGGNet and ResNet architectures for marine vessel classification task on MARVEL dataset. The results confirmed that transferring the parameter values of the first layers and fine-tuning the other layers, whose weights are initialized from pre-trained weights, performs better than training the network from scratch. It's also observed that preprocessing and regularization improves overall scores significantly.

Keywords - transfer learning; convolutional neural networks; alexnet; vggnet; resnet; marine vessel classification

CHAPTER 3

PROPOSED METHODOLOGY

3.1 Architecture of the solution

Our approach consists of two parts:

i. Software
ii. *Hardware*

3.1.1 Software

The software portion includes building a CNN from scratch and training it the dataset to classify waste products. In a later part we will see the usage of transfer learning to increase accuracy. The main objective of this part is to build a model that can be used to classify waste images. The model can be loaded on a development board and with minimal computing it can classify images of waste products.

The following tools are used during the software development:

i. *Python (Programming Language)* - Python is an interpreted, high-level, general-purpose programming language. Created by Guido van Rossum and first released in 1991, Python's design philosophy emphasizes code readability with its notable use of significant whitespace. Its language constructs and object-oriented approach aim to help programmers write clear, logical code for small and large-scale projects. Python is commonly used in artificial intelligence projects and machine learning projects with the help of libraries like TensorFlow, Keras, Pytorch and Scikit-learn. As a scripting language with modular architecture, simple syntax and rich text processing tools, Python is often used for natural language processing.

ii. *Kaggle (Dataset)* - Kaggle, a subsidiary of Google LLC, is an online community of data scientists and machine learning practitioners. Kaggle allows users to find and publish data sets, explore and build models in a web-based data-science environment, work with other data scientists and machine learning engineers, and enter competitions to solve data science challenges.

iii. *Tensorflow (open source platform for machine learning)* - TensorFlow is an end-to-end open source platform for machine learning. It has a comprehensive, flexible ecosystem of tools, libraries and community resources that lets researchers push the state-of-the-art in ML and developers easily build and deploy ML powered applications.

iv. *Tensorboard (visualization toolkit)* - TensorBoard provides the visualization and tooling needed for machine learning experimentation.

v. *Google Colab (Cloud Platform to train ML models)* - Colab allows anybody to write and execute arbitrary python code through the browser, and is especially well suited to machine learning, data analysis and education. More technically, Colab is a hosted Jupyter notebook service that requires no setup to use, while providing free access to computing resources including GPUs.

vi. *Visual Studio Code (text editor)* - Visual Studio Code is a free source-code editor made by Microsoft for Windows, Linux and macOS. Features include support for debugging, syntax highlighting, intelligent code completion, snippets, code refactoring, and embedded Git. Users can change the theme, keyboard shortcuts, preferences, and install extensions that add additional functionality. The source code is free and open-source, released under the permissive MIT License. The compiled binaries are freeware for any use.

vii. *GitHub (version control)* - It offers the distributed version control and source code management (SCM) functionality of Git, plus its own features. It provides access control and several collaboration features such as bug tracking, feature requests, task management, and wikis for every project.

3.2 Method 1 - Building a CNN from scratch

3.2.1 Building a Machine Learning Model

This section contains brief steps on how to build a machine learning model from scratch using Tensorflow. There are mainly three steps in building a machine learning model. These are as follows:

i. *Data Collection*
ii. *Data Pre-Processing*
iii. *Model Selection*

3.2.2 Data Collection

This is the first step in building any machine learning model. A deep learning model can not perform well if it is not fed enough data to train upon. There are many methods of data collection but we opted to get our dataset from Kaggle.

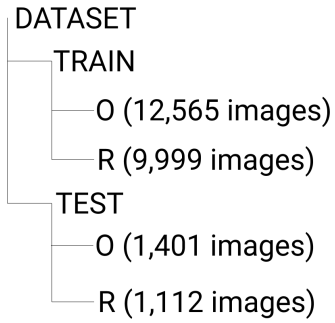

Fig. 1 Folder Structure

3.2.3 Data Pre-Processing

This is perhaps the most important and difficult step to build a successful machine learning model. This step is also known as "Data Cleaning". Data Pre-Processing ensures that our model is not fed any unwanted/bad data. There are many steps involved which are discussed below.

i. *Feature Scaling* - Often also known as "Normalization", is done to scale all the pixel values of the RGB image into a range from 0-1. This improves the learning speed as well as the model does not give more importance to any particular feature.
ii. *Resizing* - The images are of various dimensions, but our model can only take a certain dimension of images. So, the images are resized to 150px ✕ 150px before feeding into the model.
iii. *Data Augmentation* - The train folder has nearly 13,000 images of organic and 10,000 images of recyclable and the test folder has about 1000 images of both organic and recyclable respectively. But this might not be enough for our CNN to learn. So, we will augment the data using techniques such

as zooming, cropping, padding, and horizontal flipping to significantly increase the diversity of data available for training models, without actually collecting new data.

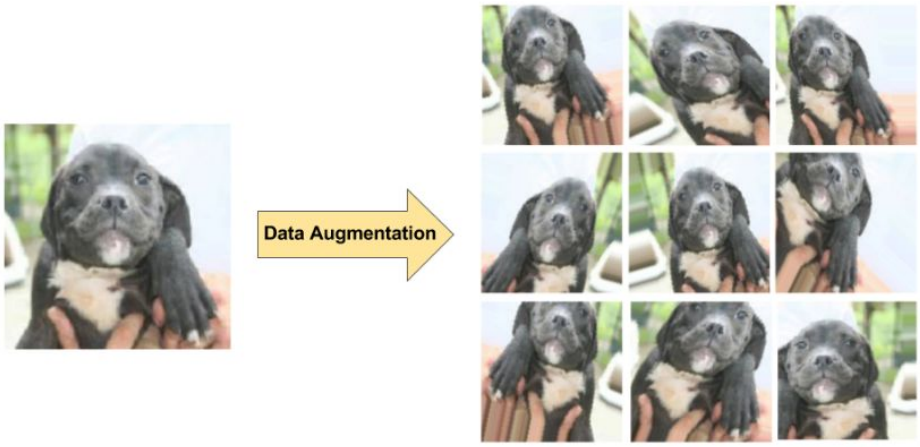

Fig 6. Data Augmentation example

3.2.4 Model Selection

This is the most time consuming process of all. This step contains hit and trial to select the perfect model and tune the hyper-parameters of the said model. We are using a CNN as it has proved to be very good in classifying images. Tensorflow provides us with all the functions necessary to build a CNN from scratch. Now the next step is to train and evaluate the CNN model.

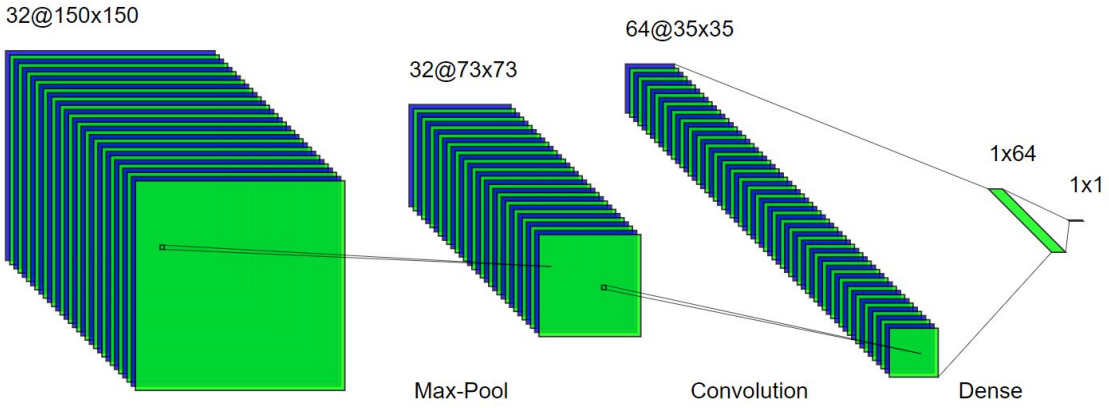

Fig 7. Visual representation of the CNN

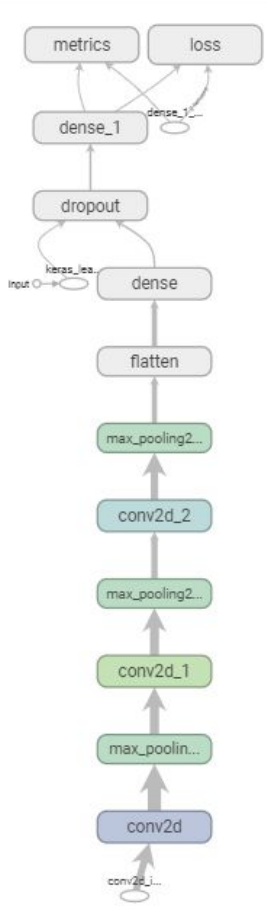

Fig 8. Architecture of CNN as visualized in "Tensorboard"

i. *Training* - Now that we have a compiled model, now we can feed our model with images and their corresponding class label and then find the error of prediction. With this error, we can readjust the weights before another image is feeded. This is the back propagation learning algorithm and this step is what enables a CNN to learn how to extract the right features and how to use them to perform predictions. The entire training data is fed fifty times as input to the model so that the model achieves its global optima.

ii. *Evaluating* - After the model is trained and all its weights have got adjusted appropriately, we will feed the model with a new set of example data and check how our model performs on new and different data. This is a crucial step as this helps us determine if our model has overfitted or under-fitted or is just right.

3.2.5 Code Snippet

The whole project along with the dataset, a pre-trained model and documentation can be found at GitHub (https://github.com/nipun24/waste-classification).

The gist of the code is given below.

i. *Data Pre-Processing:*

```
train_datagen =
tf.keras.preprocessing.image.ImageDataGenerator(
    rescale=1./255,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True)

test_datagen =
tf.keras.preprocessing.image.ImageDataGenerator(rescale=1./255)

train_generator = train_datagen.flow_from_directory(
    'DATASET/TRAIN',
    target_size=(150, 150),
    batch_size=batch_size,
    class_mode='binary')

validation_generator = test_datagen.flow_from_directory(
    'DATASET/TEST',
    target_size=(150, 150),
    batch_size=batch_size,
    class_mode='binary')
```

ii. *Building the Model:*

```
model = tf.keras.models.Sequential()
model.add(tf.keras.layers.Conv2D(32,  (3,  3),  input_shape=(150,
150, 3)))
model.add(tf.keras.layers.Activation('relu'))
model.add(tf.keras.layers.MaxPooling2D(pool_size=(2, 2)))

model.add(tf.keras.layers.Conv2D(32, (3, 3)))
model.add(tf.keras.layers.Activation('relu'))
model.add(tf.keras.layers.MaxPooling2D(pool_size=(2, 2)))

model.add(tf.keras.layers.Conv2D(64, (3, 3)))
model.add(tf.keras.layers.Activation('relu'))
model.add(tf.keras.layers.MaxPooling2D(pool_size=(2, 2)))

model.add(tf.keras.layers.Flatten())
model.add(tf.keras.layers.Dense(64))
model.add(tf.keras.layers.Activation('relu'))
model.add(tf.keras.layers.Dropout(0.5))
model.add(tf.keras.layers.Dense(1))
```

```
model.add(tf.keras.layers.Activation('sigmoid'))

model.compile(loss='binary_crossentropy',
              optimizer='rmsprop',
              metrics=['accuracy'])
```

iii. Training the Model:

```
model.fit(
    train_generator,
    batch_size=2000,
    steps_per_epoch=125,
    epochs=50)
```

iv. Evaluating the Model:

```
model.evaluate(validation_generator,steps=125)
```

3.3 Method 2 - Retraining a CNN

Transfer learning basically involves using an already existing images classifier knowledge and then fine tuning it to serve our purpose.

3.3.1 Building a Machine Learning Model

This section contains brief steps on how to build a machine learning model from scratch using Tensorflow. There are mainly three steps in building a machine learning model. These are as follows:

i. *Data Collection*
ii. *Data Pre-Processing*
iii. *Model Selection*

3.3.2 Data Collection

This is the first step in building any machine learning model. A deep learning model can not perform well if it is not fed enough data to train upon. There are many methods of data collection but we opted to get our dataset from Kaggle.

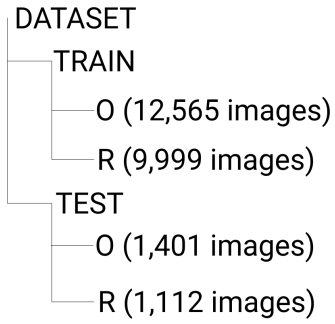

Fig 9. Folder Structure

3.3.3 Data Pre-Processing

This is perhaps the most important and difficult step to build a successful machine learning model. This step is also known as "Data Cleaning". Data Pre-Processing ensures that our model is not fed any unwanted/bad data. There are many steps involved which are discussed below.

i. *Feature Scaling* - Often also known as "Normalization", is done to scale all the pixel values of the RGB image into a range from 0-1. This improves the learning speed as well as the model does not give more importance to any particular feature.
ii. *Resizing* - The images are of various dimensions, but our model can only take a certain dimension of images. So, the images are resized to 150px ✕ 150px before feeding into the model.

iii. *Data Augmentation* - The train folder has nearly 13,000 images of organic and 10,000 images of recyclable and the test folder has about 1000 images of both organic and recyclable respectively. But this might not be enough for our CNN to learn. So, we will augment the data using techniques such as zooming, cropping, padding, and horizontal flipping to significantly increase the diversity of data available for training models, without actually collecting new data.

3.3.4 Model Selection

We will be using the Inception Resnet V2 model from Google. This is a convolutional neural network that is trained on 1.2 million images and tested on 50 thousand images - courtesy of ImageNet dataset. The network is 164 layers deep and can classify images into 1000 object categories. The network has a Top 5 Accuracy of 95.3 which makes it a clear choice of model to retrain on. Since our goal is to classify image into only two classes, we will be only using only the pretrained convolutional layers of the Inception Resnet V2 and then to this we will be adding our own dense decision layers so that out retrained model will be able to use the already existing knowledge.

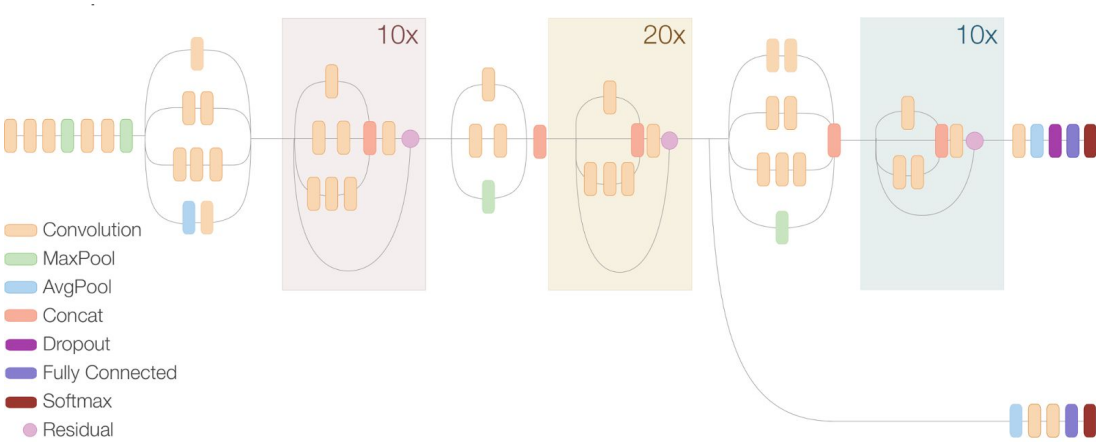

Fig 10. Inception Resnet architecture

We will be getting the Inception Resnet V2 model structure and weights using the Tensorflow Hub API. We will be keeping the convolutional pretrained weights constant and we will only be working on

i. *Training* - Now that we have a compiled model, now we can feed our model with images and their corresponding class label and then find the error of prediction. With this error, we can readjust the weights before another image is feeded. This is the back propagation learning algorithm and this step is what enables a CNN to learn how to extract the right features and how to use them to perform predictions. The entire training data is fed fifty times as input to the model so that the model achieves its global optima.

ii. *Evaluating* - After the model is trained and all its weights have got adjusted appropriately, we will feed the model with a new set of example data and check how our model performs on new and different data. This is a crucial step as this helps us determine if our model has overfitted or under-fitted or is just right.

3.3.5 Code Snippet

The whole project along with the dataset, a pre-trained model and documentation can be found at GitHub (https://github.com/nipun24/waste-classification).

The gist of the code is given below.

v. *Data Pre-Processing:*

```
train_datagen                                                    =
tf.keras.preprocessing.image.ImageDataGenerator(
    rescale=1./255,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True)

test_datagen                                                     =
tf.keras.preprocessing.image.ImageDataGenerator(rescale=1./255)

train_generator = train_datagen.flow_from_directory(
    'DATASET/TRAIN',
    target_size=(150, 150),
    batch_size=batch_size,
    class_mode='binary')

validation_generator = test_datagen.flow_from_directory(
    'DATASET/TEST',
    target_size=(150, 150),
    batch_size=batch_size,
    class_mode='binary')
```

vi. *Building the Model:*

```
base_model = tf.keras.applications.InceptionResNetV2(
             input_shape=(150,150,3),
             include_top=False,
             weights='imagenet')
base_model.trainable = False

model = tf.keras.models.Sequential(base_model)
model.add(tf.keras.layers.Flatten())
model.add(tf.keras.layers.Dense(256, activation='relu'))
model.add(tf.keras.layers.Dropout(0.5))
model.add(tf.keras.layers.Dense(1, activation='sigmoid'))
```

```
model.compile(loss='binary_crossentropy',
              optimizer='rmsprop',
              metrics=['accuracy'])
```

vii. Training the Model:

```
model.fit(
    train_generator,
    batch_size=2000,
    steps_per_epoch=125,
    epochs=50)
```

viii. Evaluating the Model:

```
model.evaluate(validation_generator,steps=125)
```

3.4 Hardware

As we are ready with our self-learning model(software part), now we need to integrate the model with hardware so that it can show actual action to common peoples.

We will look briefly at everything needed for hardware development:

i. *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. Processor speed ranges from 700 MHz to 1.4 GHz for the Pi 3 Model B+ or 1.5 GHz for the Pi 4; on-board memory ranges from 256 MiB to 1 GiB random-access memory (RAM), with up to 8 GiB available on the Pi 4. Secure Digital (SD) cards in MicroSDHC form factor (SDHC on early models) are used to store the operating system and program memory. The boards have one to five USB ports. For video output, HDMI and composite video are supported, with a standard 3.5 mm tip-ring-sleeve jack for audio output. Lower-level output is provided by a number of GPIO pins, which support common protocols like I²C. The B-models have an 8P8C Ethernet port and the Pi 3, Pi 4 and Pi Zero W have on-board Wi-Fi 802.11n and Bluetooth.

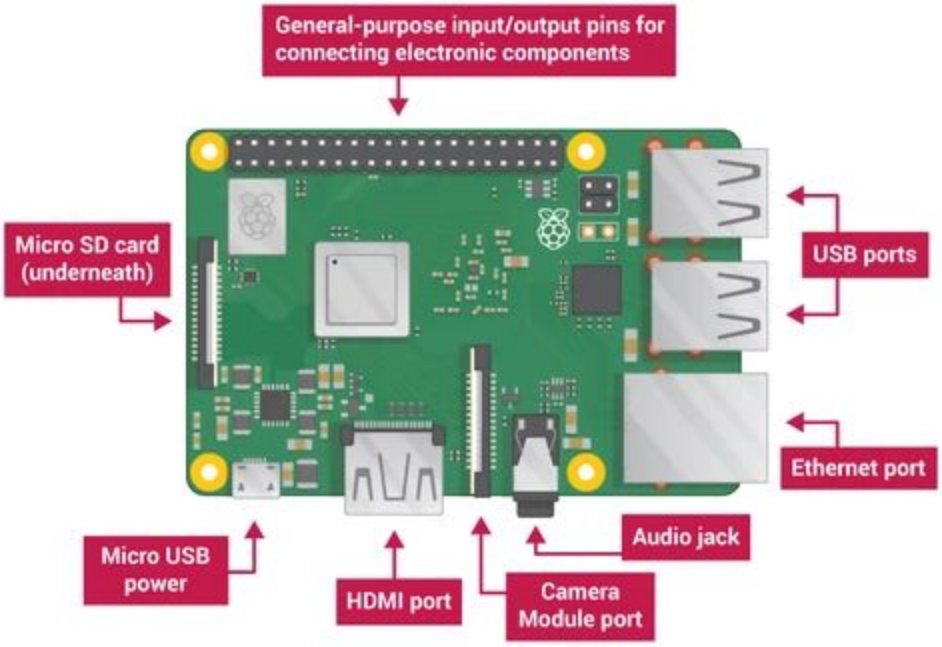

Fig 11. Raspberry Pi

ii. *Raspbian* - Raspberry Pi OS is a Debian-based operating system for Raspberry Pi. Since 2015 it has been officially provided by the Raspberry Pi Foundation as the primary operating system for the family of Raspberry Pi single-board computers.

iii. *Raspberry Pi Camera Module v2* - The v2 Camera Module has a Sony IMX219 8-megapixel sensor. The Camera Module is capable of high-definition video, as well as stills photographs. It attaches via a 15cm ribbon cable to the CSI port on the Raspberry Pi. It can be accessed through the MMAL and V4L APIs, and there are numerous third-party libraries built for it, including the Picamera Python library.

Fig 12. Raspberry Pi Camera Module V2

iv. *IR Proximity Sensor* - IR, in short for infrared, detects the presence of an object by emitting a beam of infrared light. It works similarly to ultrasonic sensors, though instead of using sonic waves, IR is transmitted.Infrared proximity sensors consist of an IR LED that emits, and a light detector for detection of reflection. It has an in-built signal processing circuit that determines an optical spot on the PSD.

Fig 13. IR Proximity Sensor

3.4.1 Building Hardware

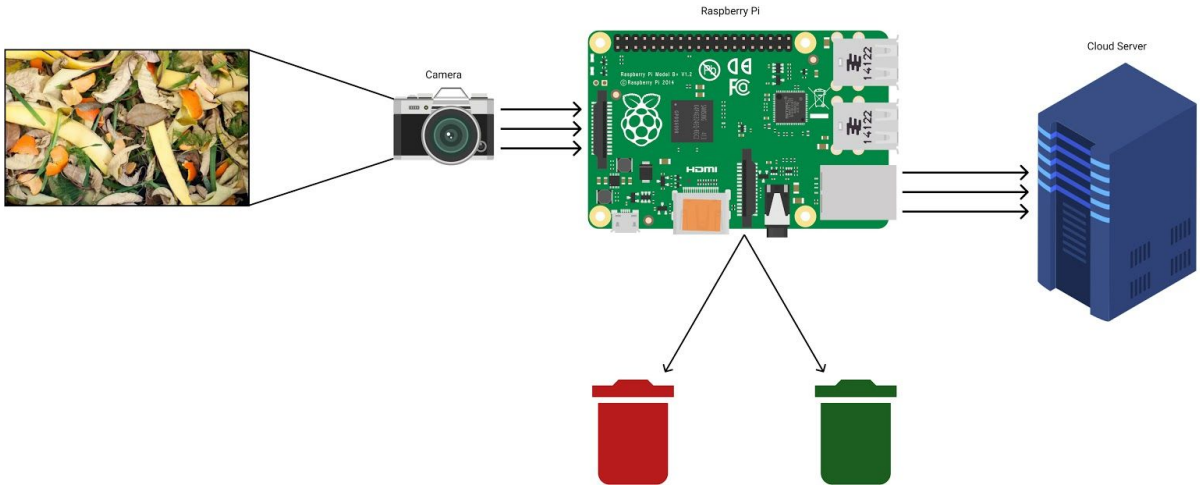

Fig 14. Hardware Architecture

The process setting up the hardware will be as follow:

i. First we will have to write the Raspbian image to a SD card with memory greater than 16GB. We can use any tool to write the image with few being Win32 Disk manager, Raspberry pi imager, Rufus Portable etc.

ii. Now we will insert the micro SD card into the SD card slot and give the Raspberry Pi the power through USB type B. Once powered we will wait for the Raspbian to boot.

iii. Now once the Raspbian has booted, we will download all software and python library dependencies using `sudo apt get install` and `pip`

iv. Once we have all the dependencies installed, we can write the code to screen proximity sensor activity, then take images, use our trained CNN model to make predictions and then finally control the GPIO pins based on the output.

3.4.2 Working

We will be using proximity sensors to sense whether someone is coming towards the dustbin or not. If yes it will send a signal to raspberry pi to power up the camera so that camera will awake only when someone is near the dustbin otherwise it will be off (sleeping mode, reducing power usage).

Once the camera is on, it will capture the image of garbage and send it using CSI. The image data is fed to the model as input to classify into the category and accordingly open the bin as an action. It will also send the data (input as well as output) to the server where we will be analysing the prediction and fine tuning the model.

CHAPTER 4

RESULTS AND DISCUSSION

4.1 Method 1: Building a CNN from Scratch

This section describes the performance of the CNN we trained from scratch. To visualise the course of training of our model, we will be using Tensorboard - a web app extension of Tensorflow that creates log files of all metric changes (like accuracy and loss) and then uses these metrics to give us very intuitive plots.

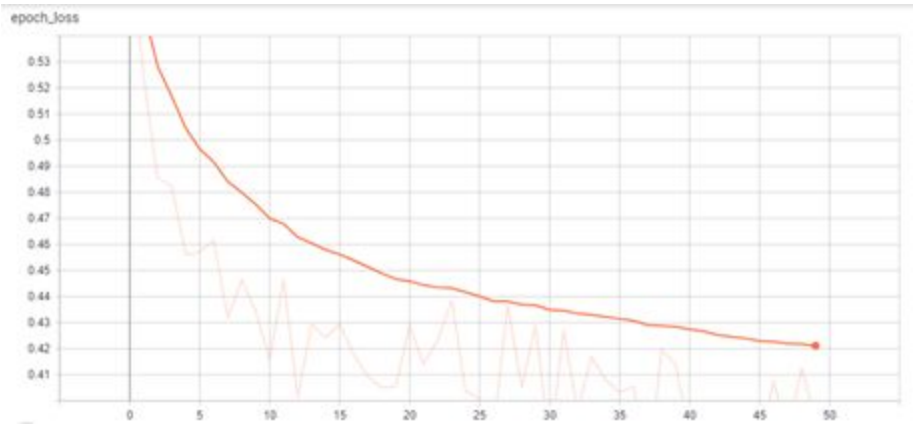

Fig 15. Loss graph during training

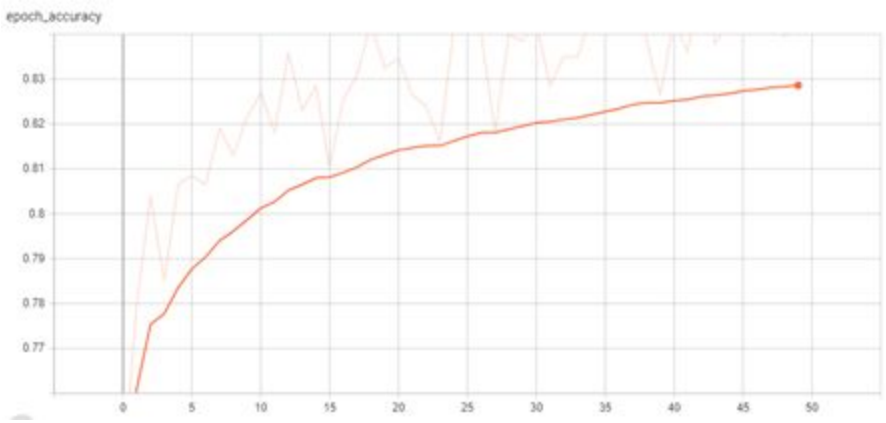

Fig 16. Accuracy graph during training

From the above graphs we can see that :

- The training loss of the model was about .42
- The training accuracy of the model was about .83 (83%)

Moreover, we can use the evaluate function on our model to see the accuracy on the test set.

```
1 model.evaluate(validation_generator,steps=125)

125/125 [==============================] - 3s 24ms/step - loss: 0.4069 - accuracy: 0.8800
[0.4068816304206848, 0.8799999952316284]
```

Fig 17. Test Accuracy of the model

We can show that out model is indeed learning through the following histograms:

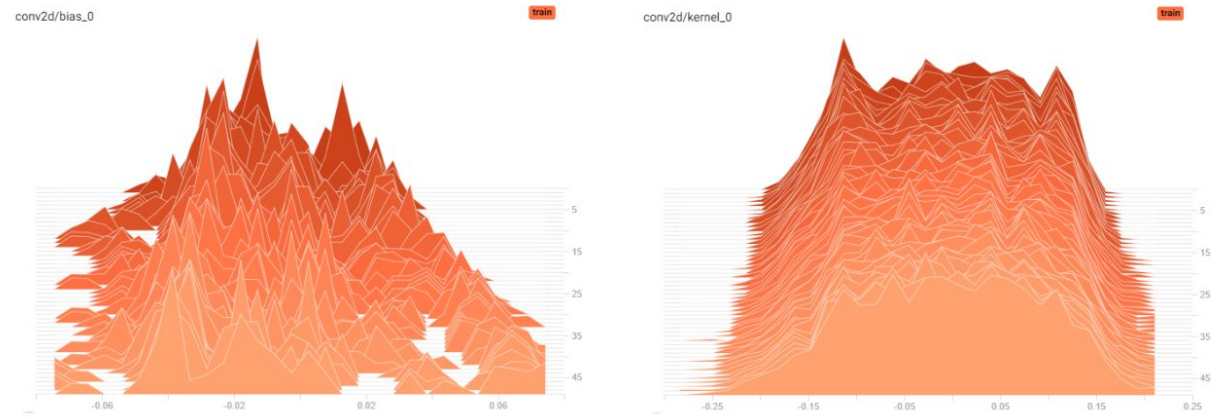

Fig 18. Input Layer Histogram

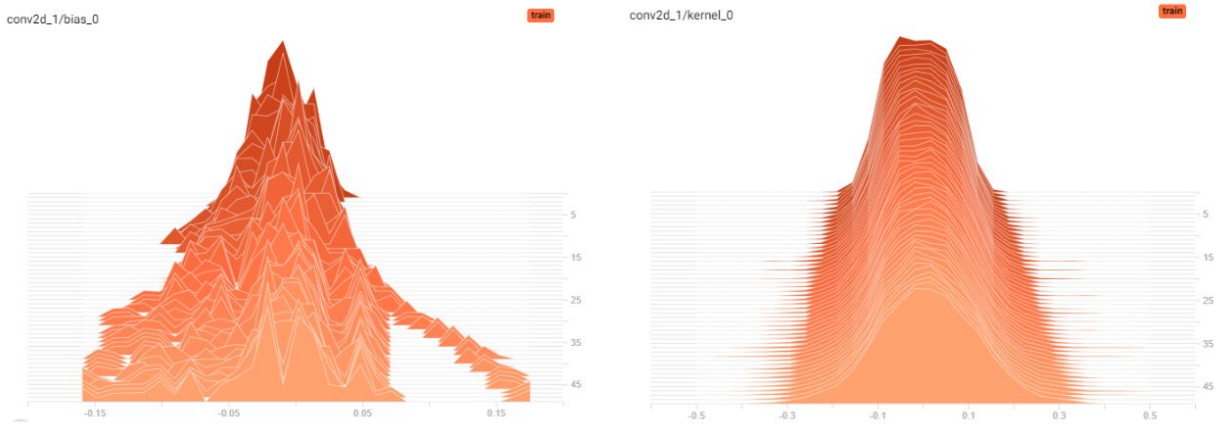

Fig 19. 1st hidden layer histogram

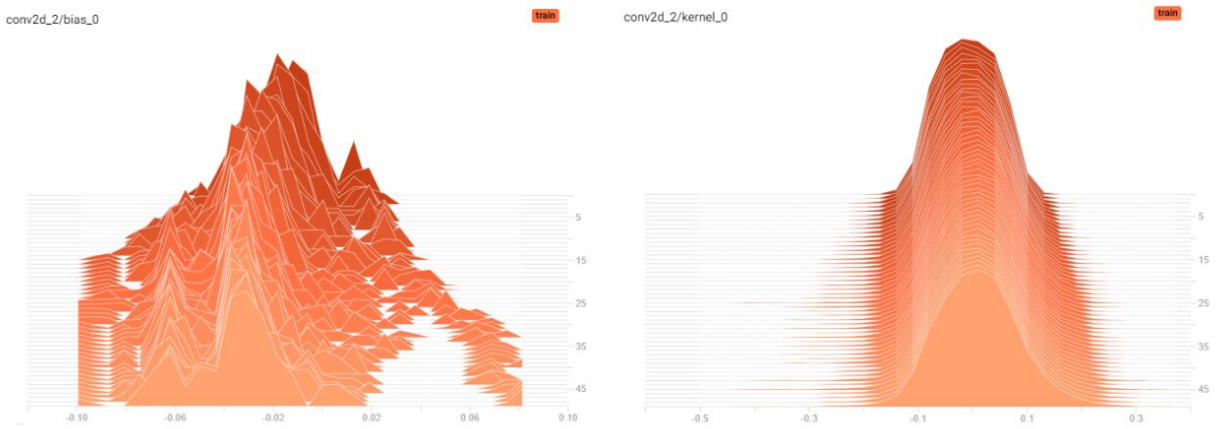

Fig 20. 2nd hidden layer histogram

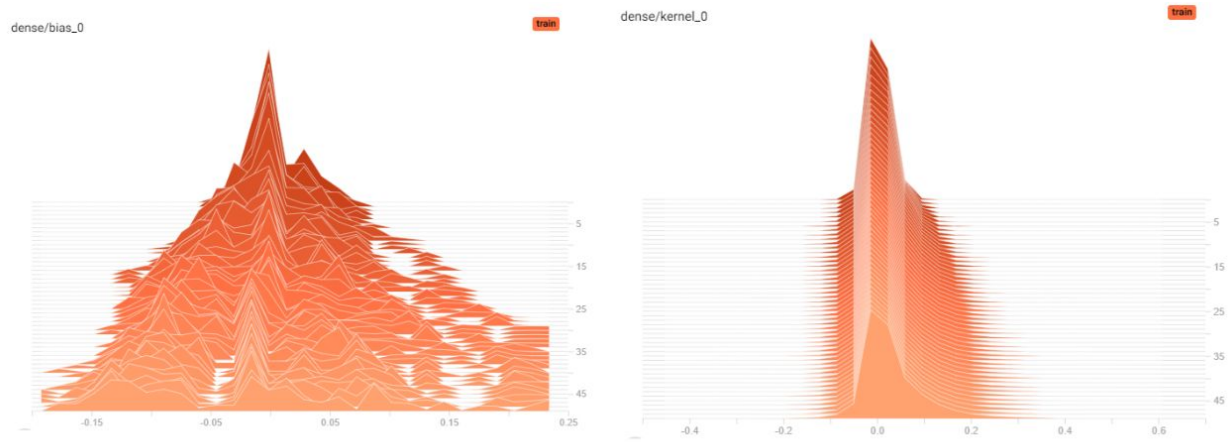

Fig 21. hidden dense layer histogram

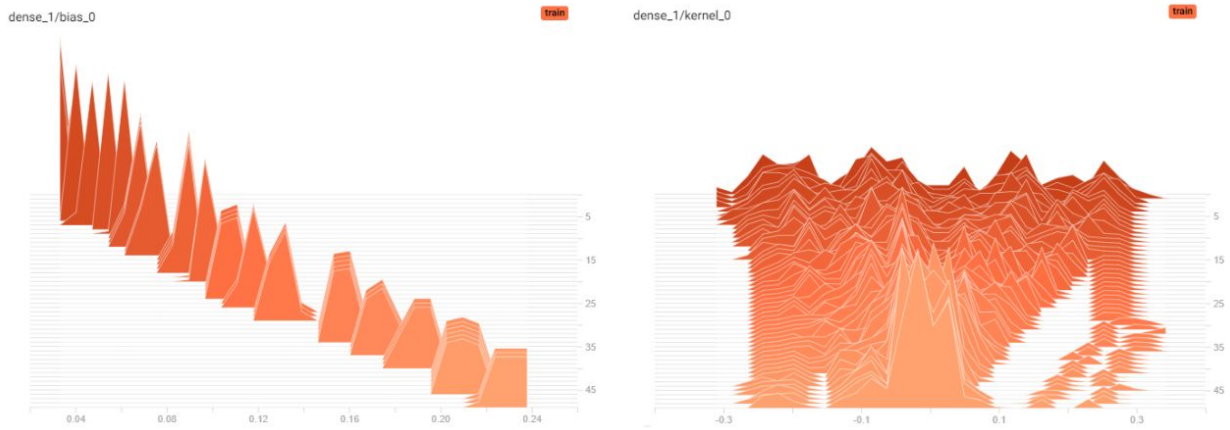

Fig 22. output layer histogram

4.2 Method 2: Retraining a CNN

This section describes the performance of the CNN we retrained using the Inception ResNET. To visualise the course of training of our model, we will be using Tensorboard - a web app extension of Tensorflow that creates log files of all metric changes (like accuracy and loss) and then uses these metrics to give us very intuitive plots.

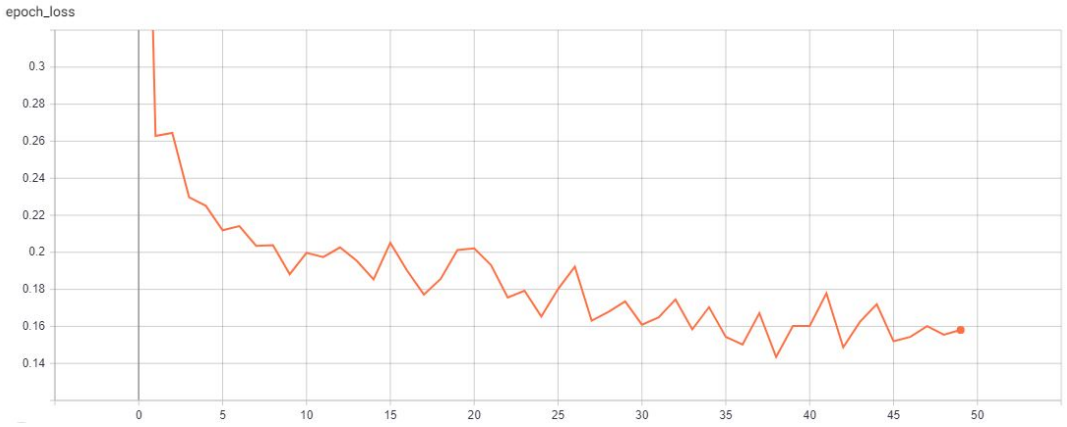

Fig 23. Loss graph during training

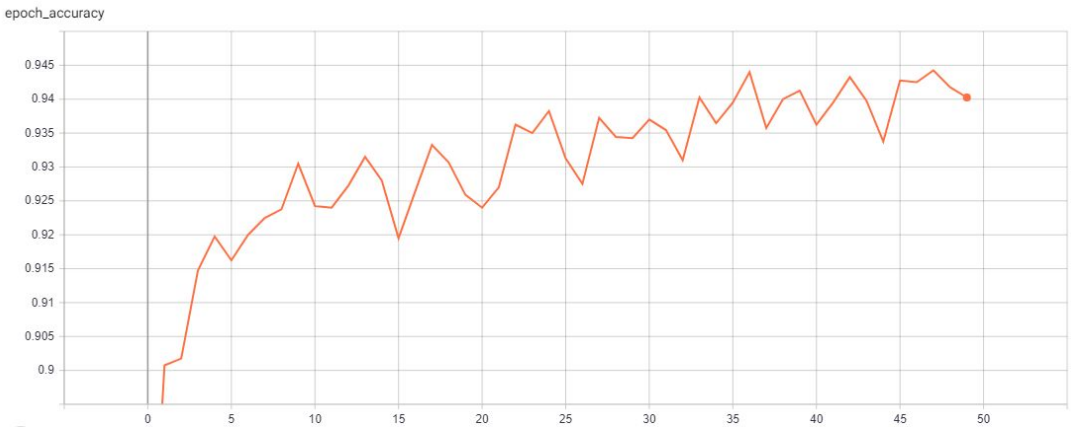

Fig 24. Accuracy graph during training

From the above graphs we can see that :

- The training loss of the model was about .16
- The training accuracy of the model was about .94 (94%)

Moreover, we can use the evaluate function on our model to see the accuracy on the test set.

```
1 model.evaluate(validation_generator)
```

```
158/158 [==============================] - 20s 124ms/step - loss: 0.2553 - accuracy: 0.9025
[0.2552744448184967, 0.902506947517395]
```

Fig 25. Test Accuracy of the model

4.3 On Raspberry Pi

On the raspberry pi, we will be running the Appendix B code to take pictures and make predictions on those images. Here we will look at how our model classifies the 10 images shown below. Clearly 5 images belong to Organic - 0 and the next 5 belong to class Recyclable.

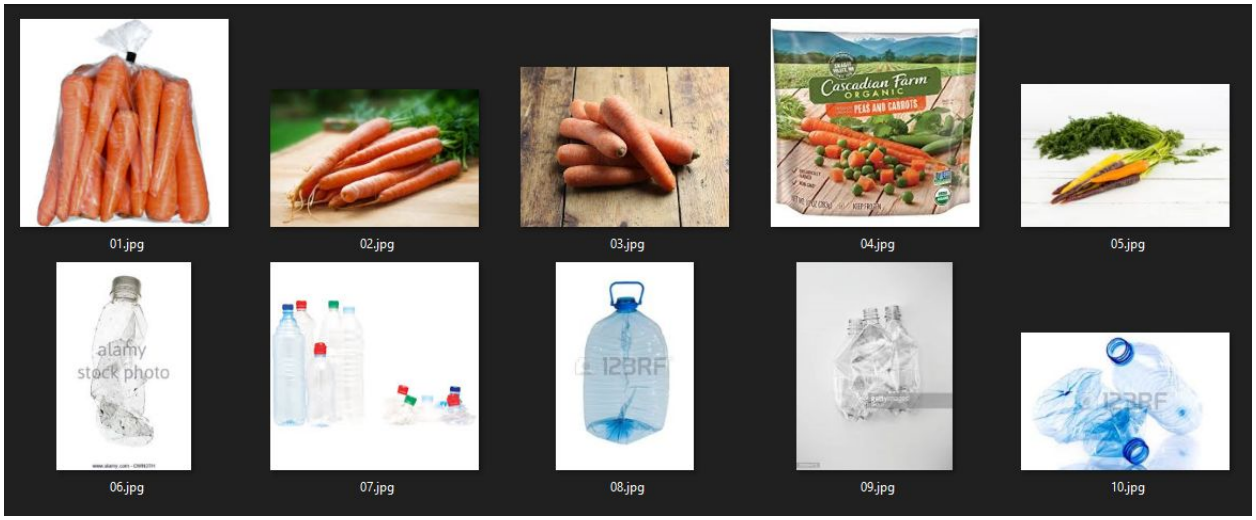

Fig 26. Sample Images

This is the output we get from our Appendix B code. As you can see, our model correctly predicted 9 out of 10 images to their respective classes. [Although we might argue that the plastic cover wrapped around the image 0 affected the prediction]

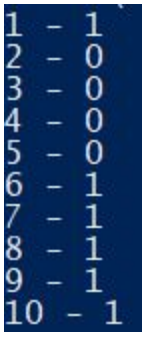

Fig 27. Our prediction

CHAPTER 5

CONCLUSION

From the report we can conclude that convolutional neural networks do work best for image classification and that we were able to achieve between 93-94% accuracy by using pre trained model knowledge. Our own model did perform very well against new images with an accuracy of 83% which is very good considering all the different objects that come under the broadest class of organic and recyclable. The pretrained model we used was able to predict images in millisecond.

This product if deployed correctly on a commercial level will help reduce the amount of waste which is accumulated in landfills unsegregated. Hopefully, this will be used towards the betterment of our environment.

APPENDIX A
SOURCE CODE - SOFTWARE DEVELOPMENT

The same code can be found on GitHub (https://github.com/nipun24/waste-classification)

waste-classification.py -

```
from zipfile import ZipFile
import tensorflow as tf

f = ZipFile('DATASET.zip', 'r')
f.extractall()

batch_size = 16

train_datagen = tf.keras.preprocessing.image.ImageDataGenerator(
    rescale=1./255,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True)

test_datagen = tf.keras.preprocessing.image.ImageDataGenerator(rescale=1./255)

train_generator = train_datagen.flow_from_directory(
    'DATASET/TRAIN',
    target_size=(150, 150),
    batch_size=batch_size,
    class_mode='binary')

validation_generator = test_datagen.flow_from_directory(
    'DATASET/TEST',
    target_size=(150, 150),
    batch_size=batch_size,
    class_mode='binary')

model = tf.keras.models.Sequential()
model.add(tf.keras.layers.Conv2D(32, (3, 3), input_shape=(150, 150, 3)))
model.add(tf.keras.layers.Activation('relu'))
model.add(tf.keras.layers.MaxPooling2D(pool_size=(2, 2)))

model.add(tf.keras.layers.Conv2D(32, (3, 3)))
model.add(tf.keras.layers.Activation('relu'))
model.add(tf.keras.layers.MaxPooling2D(pool_size=(2, 2)))

model.add(tf.keras.layers.Conv2D(64, (3, 3)))
model.add(tf.keras.layers.Activation('relu'))
model.add(tf.keras.layers.MaxPooling2D(pool_size=(2, 2)))

model.add(tf.keras.layers.Flatten())
model.add(tf.keras.layers.Dense(64))
model.add(tf.keras.layers.Activation('relu'))
model.add(tf.keras.layers.Dropout(0.5))
model.add(tf.keras.layers.Dense(1))
model.add(tf.keras.layers.Activation('sigmoid'))
```

```
model.compile(loss='binary_crossentropy',
              optimizer='rmsprop',
              metrics=['accuracy'])

model.fit(
    train_generator,
    batch_size=2000,
    steps_per_epoch=125,
    epochs=50)
model.save('waste-classifier.h5')

model.evaluate(validation_generator, steps=125)
```

waste-classification-transfer.py -

```
import tensorflow as tf

from google.colab import drive
drive.mount('/content/drive')

from zipfile import ZipFile
f = ZipFile('drive/My Drive/Dataset/classorig.zip', 'r')
f.extractall()

import os
os.listdir('./CONTENT')

batch_size = 16

train_datagen = tf.keras.preprocessing.image.ImageDataGenerator(
        rescale=1./255,
        shear_range=0.2,
        zoom_range=0.2,
        horizontal_flip=True,
        validation_split=0.1)

train_generator = train_datagen.flow_from_directory(
        './CONTENT',
        target_size=(150, 150),
        batch_size=batch_size,
        class_mode='binary',
        subset='training')

validation_generator = train_datagen.flow_from_directory(
        './CONTENT',
        target_size=(150, 150),
        batch_size=batch_size,
        class_mode='binary',
        subset='validation')
```

```
base_model =
tf.keras.applications.InceptionResNetV2(input_shape=(150,150,3),
                                        include_top=False,
                                        weights='imagenet')
base_model.trainable = False

model = tf.keras.models.Sequential(base_model)
model.add(tf.keras.layers.Flatten())
model.add(tf.keras.layers.Dense(256, activation='relu'))
model.add(tf.keras.layers.Dropout(0.5))
model.add(tf.keras.layers.Dense(1, activation='sigmoid'))

model.compile(loss='binary_crossentropy',
              optimizer= 'adam',
              metrics = ['accuracy'])

model.summary()

tensorboard_callback = tf.keras.callbacks.TensorBoard('logs',
histogram_freq=1)

model.fit(
    train_generator,
    batch_size=200,
    steps_per_epoch=125,
    epochs=30,
    callbacks=[tensorboard_callback])

model.save('waste-classifier-transfer.h5')

model = tf.keras.models.load_model('waste-classifier-transfer.h5')

model.evaluate(validation_generator)
```

APPENDIX B
SOURCE CODE - HARDWARE DEVELOPMENT

The same code can be found on GitHub (https://github.com/nipun24/waste-classification)

rasp-classifier.py -

```
import os
os.environ['TF_CPP_MIN_LOG_LEVEL'] = '3'

import cv2.cv2 as cv2
import tensorflow as tf
import numpy as np

model = tf.keras.models.load_model("./model save file/inception.h5")

imglist = []
for i in os.listdir('./images'):
    img = cv2.imread(os.path.join('./images/',i))
    img = cv2.resize(img, (150,150))
    img = cv2.cvtColor(img,  cv2.COLOR_BGR2RGB)
    img = img/255
    imglist.append(img)

imglist = np.asarray(imglist)

pred = model.predict(imglist)
fg = 1
for j in pred:
    print(f'{fg} - {np.argmax(j)}')
    fg += 1
```

REFERENCES

1. Y. Lecun, L. Bottou, Y. Bengio and P. Haffner, "Gradient-based learning applied to document recognition," in Proceedings of the IEEE, vol. 86, no. 11, pp. 2278-2324, Nov. 1998, doi: 10.1109/5.726791.

2. T. Guo, J. Dong, H. Li and Y. Gao, "Simple convolutional neural network on image classification," 2017 IEEE 2nd International Conference on Big Data Analysis (ICBDA), Beijing, 2017, pp. 721-724, doi: 10.1109/ICBDA.2017.8078730.

3. https://www.tensorflow.org/tutorials/images/cnn

4. https://www.tensorflow.org/tutorials/images/classification

5. https://blog.keras.io/building-powerful-image-classification-models-using-very-little-data.html

6. https://www.tensorflow.org/tensorboard/get_started

7. https://www.kaggle.com/techsash/waste-classification-data

8. https://developers.google.com/machine-learning/practica/image-classification/convolutional-neural-networks

9. https://en.wikipedia.org/wiki/Deep_learning

10. https://en.wikipedia.org/wiki/Internet_of_things

11. https://en.wikipedia.org/wiki/Convolutional_neural_network

12. https://towardsdatascience.com/a-comprehensive-hands-on-guide-to-transfer-learning-with-real-world-applications-in-deep-learning-212bf3b2f27a

13. https://www.python.org/

14. https://keras.io/

15. https://opencv.org/

16. https://www.raspberrypi.org/downloads/raspberry-pi-os/

17. https://www.tensorflow.org/tutorials/images/transfer_learning