

Intelligent Waste Classification System

Using Vision Transformer

Submitted in partial fulfillment of the requirements for
the award of

Bachelor of Engineering degree in Computer Science and Engineering
by

Menti Aditya Gowrish (Reg. No. 37110448)

Mukesh Kumar Dahlan (Reg. No. 37110458)



**DEPARTMENT OF COMPUTER SCIENCE AND
ENGINEERING SCHOOL OF COMPUTING**

SATHYABAMA

INSTITUTE OF SCIENCE AND TECHNOLOGY

(DEEMED TO BE UNIVERSITY)

**Accredited with Grade “A” by NAAC JEPPIAAR NAGAR, RAJIV
GANDHI SALAI, CHENNAI – 600 119**

MARCH - 2021



SATHYABAMA

INSTITUTE OF SCIENCE AND TECHNOLOGY
(DEEMED TO BE UNIVERSITY)

Accredited with "A" grade by NAAC
Jeppiaar Nagar, Rajiv Gandhi Salai, Chennai – 600 119

www.sathyabama.ac.in



DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

BONAFIDE CERTIFICATE

This is to certify that this project report is the bonafide work of **MENTI ADITYA GOWRISH** (Reg. No. 37110448) and **MUKESH KUMAR DAHLAN** (Reg. No. 37110458) who carried out the project entitled "INTELLIGENT WASTE CLASSIFICATION SYSTEM USING VISION TRANSFORMER" under my supervision from **August 2020** to **March 2021**.

Internal Guide

Dr. Subhashini,M.E.,Ph.d.

Head of the Department

Submitted for Viva voce Examination held on _____

Internal Examiner

External Examiner

DECLARATION

I **MENTI ADITYA GOWRISH** hereby declare that the Project Report entitled **“INTELLIGENT WASTE CLASSIFICATION**

USING VISION TRANSFORMER” is done by me under the guidance of **Dr. SUBHASHINI M.E.,Ph.d.**, Department of Computer Science and Engineering at Sathyabama Institute of Science and Technology is submitted in partial fulfillment of the requirements for the award of Bachelor of Engineering degree in Computer Science and Engineering.

DATE:
PLACE: CHENNAI

SIGNATURE OF THE CANDIDATE

ACKNOWLEDGEMENT

I am pleased to acknowledge my sincere thanks to **Board of Management of SATHYABAMA** for their kind encouragement in doing this project and for completing it successfully. I am grateful to them.

I convey my thanks to **Dr. T. Sasikala, M.E., Ph.D., Dean**, School of Computing, **Dr. S. Vigneswari, M.E., Ph.D., and Dr. L. Lakshmanan, M.E., Ph.D., Heads of the Department** of Computer Science and Engineering for providing me necessary support and details at the right time during the progressive reviews.

I would like to express my sincere and deep sense of gratitude to my Project Guide **Dr. Subhashini, M.E., Ph.D. Professor**, for her valuable guidance, suggestions and constant encouragement paved way for the successful completion of my project work.

I wish to express my thanks to all Teaching and Non-teaching staff members of the **Department of Computer Science and Engineering** who were helpful in many ways for the completion of the project.

ABSTRACT

The issue of waste management is a growing concern, ranging from agricultural fields to industries, villages to cities. Our project aims to contribute to this issue, by solving the problem of segregation of waste by using the latest advances in computer Vision, We propose the use of Vision Transformer from the paper An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale, which will give highly efficient results in lesser time compared to models based on Convolutional Neural Networks which loses a lot of valuable information and ignores the relation between the part of images and the whole. Self-Attention in vision transformers gives them the capability to understand the connection between inputs, where operations can be processed parallelly on multiple GPUs and CPUs which cannot be done in the case of Convolutional Neural Networks. This project allows us to build an effective system that can classify waste in real time without human intervention in large scale industries and also in regular houses.

TABLE OF CONTENTS

	ABSTRACT	v
	LIST OF FIGURES	viii
CHAPTER No.	TITLE	PAGE No.
1.	INTRODUCTION	1
	1.1 ABOUT MACHINE LEARNING	1
	1.2 MOTIVATION	18
	1.3 RESEARCH AND SIGNIFICANCE	18
2.	AIM AND SCOPE	19
	2.1 AIM OF PROJECT	19
	2.1.1 OBJECTIVES	19
	2.2 SCOPES	19
	2.2.1 ADVANTAGES	20
3.	SYSTEM DESIGN & METHODOLOGY	21
	3.1 EXISTING SYSTEM	21
	3.2 SYSTEM CAPABILITIES	21
	3.3 REQUIREMENT SPECIFICATION	22
	3.3.1 SOFTWARE REQUIREMENTS	22
	3.3.2 HARDWARE REQUIREMENTS	22
4.	SOFTWARE DEVELOPMENT METHODOLOGY	23
	4.1 DESCRIPTION OF DIAGRAM	23
	4.2 POSITIONAL EMBEDDING	25
	4.3 KEY FINDINGS	26
	4.4 REPRESENTING AN IMAGE AS SEQUENCE OF PATCHES	26
	4.5 ATTENTION DISTENCE AND VISUALISATION	27
6.	RESULTS AND DISCUSSION	29
7.	CONCLUSION AND FUTURE WORK	30

REFERENCES	31
-------------------	-----------

APPENDIX

A. SOURCE CODE	32
B. SCREENSHOTS	35
C. PLAGIARISM REPORT	42

LIST OF FIGURES

FIGURE No.	FIGURE NAME	PAGE No.
1.1.1	APPLICATIONS OF MACHINE LEARNING	3
1.1.4.1	ML,DL and AI	6
1.1.4.2	Neural Network	7
1.1.4.1.1	Difference between ML and DL	9
1.1.4.1.2	Neural Network Performance	10
1.1.4.2.1	Artificial Neural Network	11
1.1.4.3.1	Neural Network Architecture	12
1.1.5.1	Computer Vision	13
1.1.5.1.1	Pixel Value Representation	14
1.1.5.1.2	Pixel Storage	15
1.1.5.1.3	RBG Representation	15
1.1.6.1	Vision Transformer	17
4.1.1	ARCHITECTURE DIAGRAM	23
4.1.2	TRANSFORMER ENCODER	24
4.1.3	VIT MODELS	24
4.2.1	POSITIONAL EMBEDDING	25
4.4.1	GRAPH OF VIT PERFORMANCE	27
4.5.1	CLASSIFICATION	28
5.1	RESULT	29

LIST OF ABBREVIATIONS

ABBREVIATION	EXPANSION
ML	Machine Learning
DL	Deep Learning
AI	Artificial Intelligence
Conv	Convolution Neural Network
ViT	Vision Transformer
CNN	Convolution Neural Network
ANN	Artificial Neural Network
BERT	Bidirectional Encoder Representation from Transformer

CHAPTER 1

INTRODUCTION

1.1 INTRODUCTION

1.1.1 MACHINE LEARNING:

INTRODUCTION:

Machine Learning is the science (and art) of programming computers so they can learn from data. Here is a slightly more general definition: [Machine Learning is the] field of study that gives computers the ability to learn without being explicitly programmed. —Arthur Samuel, 1959 And a more engineering-oriented one: A computer program is said to learn from experience E with respect to some task T and some performance measure P , if its performance on T , as measured by P , improves with experience E . —Tom Mitchell, 1997 For example, your spam filter is a Machine Learning program that can learn to flag spam given examples of spam emails (e.g., flagged by users) and examples of regular (non spam, also called “ham”) emails. The examples that the system uses to learn are called the training set. Each training example is called a training instance (or sample). In this case, the task T is to flag spam for new emails, the experience E is the training data, and the performance measure P needs to be defined; for example, you can use the ratio of correctly classified emails. This particular performance measure is called accuracy and it is often used in classification tasks. If you just download a copy of Wikipedia, your computer has a lot more data, but it is not suddenly better at any task. Thus, it is not Machine Learning.

To summarize, Machine Learning is great for:

- Problems for which existing solutions require a lot of hand-tuning or long lists of rules: one Machine Learning algorithm can often simplify code and perform better.
- Complex problems for which there is no good solution at all using a traditional approach: the best Machine Learning techniques can find a solution.
- Fluctuating environments: a Machine Learning system can adapt to new data.
- Getting insights about complex problems and large amounts of data.

1.1.2 TYPES OF MACHINE LEARNING SYSTEMS:

There are so many different types of Machine Learning systems that it is useful to classify them in broad categories based on:

- Whether or not they are trained with human supervision (supervised, unsupervised, semi-supervised, and Reinforcement Learning)
- Whether or not they can learn incrementally on the fly (online versus batch learning)
- Whether they work by simply comparing new data points to known data points, or instead detect patterns in the training data and build a predictive model, much like scientists do (instance-based versus model-based learning)

These criteria are not exclusive; you can combine them in any way you like. For example, a state-of-the-art spam filter may learn on the fly using a deep neural network model trained using examples of spam and ham; this makes it an online, model based, supervised learning system. Let's look at each of these criteria a bit more closely.

Supervised/Unsupervised Learning:

Machine Learning systems can be classified according to the amount and type of supervision they get during training. There are four major categories: supervised learning, unsupervised learning, semi-supervised learning, and Reinforcement Learning.

Supervised learning:

In supervised learning, the training data you feed to the algorithm includes the desired solutions, called labels. A typical supervised learning task is classification. The spam filter is a good example of this: it is trained with many example emails along with their class (spam or ham), and it must learn how to classify new emails. Another typical task is to predict a target numeric value, such as the price of a car, given a set of features (mileage, age, brand, etc.) called predictors. This sort of task is called regression. To train the system, you need to give it many examples of cars, including both their predictors and their labels (i.e., their prices).

Here are some of the most important supervised learning algorithms:

k-Nearest Neighbours, Linear Regression, Logistic Regression, Support Vector Machines (SVMs), Decision Trees and Random Forests, Neural networks.

Unsupervised learning:

In unsupervised learning, as you might guess, the training data is unlabeled. The system tries to learn without a teacher.

Here are some of the most important unsupervised learning algorithms:

- Clustering — K-Means — DBSCAN — Hierarchical Cluster Analysis (HCA)
- Anomaly detection and novelty detection — One-class SVM — Isolation Forest
- Visualization and dimensionality reduction — Principal Component Analysis (PCA) — Kernel PCA — Locally-Linear Embedding (LLE) — t-distributed Stochastic Neighbor Embedding (t-SNE)
- Association rule learning — Apriori — Eclat

1.1.3 APPLICATION OF MACHINE LEARNING:

We are using machine learning in our daily life even without knowing it such as Google Maps, Google assistant, Alexa, etc. Below are some most trending real-world applications of Machine Learning:

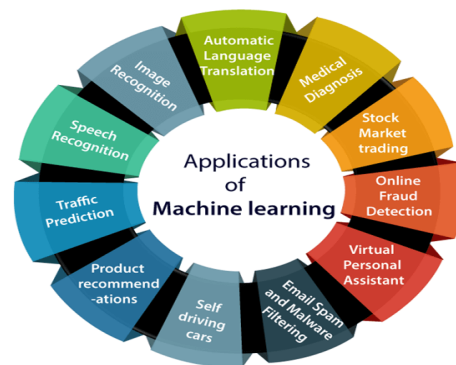


Fig : 1.3.1 Applications of Machine Learning

1.1.3.1. Image Recognition:

Image recognition is one of the most common applications of machine learning. It is used to identify objects, persons, places, digital images, etc. The popular use case of image recognition and face detection is, **Automatic friend tagging suggestion**:

Facebook provides us a feature of auto friend tagging suggestion. Whenever we upload a photo with our Facebook friends, then we automatically get a tagging suggestion with name, and the technology behind this is machine learning's **face detection** and **recognition algorithm**.

It is based on the Facebook project named "**Deep Face**," which is responsible for face recognition and person identification in the picture.

1.1.3.2. Speech Recognition:

While using Google, we get an option of "**Search by voice**," it comes under speech recognition, and it's a popular application of machine learning.

Speech recognition is a process of converting voice instructions into text, and it is also known as "**Speech to text**", or "**Computer speech recognition**." At present, machine learning algorithms are widely used by various applications of speech recognition. **Google assistant, Siri, Cortana,** and **Alexa** are using speech recognition technology to follow the voice instructions.

1.1.3.3. Traffic prediction:

If we want to visit a new place, we take help of Google Maps, which shows us the correct path with the shortest route and predicts the traffic conditions.

It predicts the traffic conditions such as whether traffic is cleared, slow-moving, or heavily congested with the help of two ways:

- o **Real Time location** of the vehicle from Google Map app and sensors
- o **Average time has taken** on past days at the same time.

Everyone who is using Google Map is helping this app to make it better. It takes information from the user and sends back to its database to improve the performance.

1.1.3.4. Product recommendations:

Machine learning is widely used by various e-commerce and entertainment companies such as **Amazon, Netflix**, etc., for product recommendation to the user. Whenever we search for some product on Amazon, then we started getting an advertisement for the same product while internet surfing on the same browser and this is because of machine learning.

Google understands the user interest using various machine learning algorithms and suggests the product as per customer interest.

As similar, when we use Netflix, we find some recommendations for entertainment series, movies, etc., and this is also done with the help of machine learning.

1.1.3.5. Self-driving cars:

One of the most exciting applications of machine learning is self-driving cars. Machine learning plays a significant role in self-driving cars. Tesla, the most popular car manufacturing company is working on self-driving car. It is using unsupervised learning method to train the car models to detect people and objects while driving.

1.1.3.6. Email Spam and Malware Filtering:

Whenever we receive a new email, it is filtered automatically as important, normal, and spam. We always receive an important mail in our inbox with the important symbol and spam emails in our spam box, and the technology behind this is Machine learning. Below are some spam filters used by Gmail:

- o Content Filter
- o Header filter
- o General blacklists filter
- o Rules-based filters
- o Permission filters

Some machine learning algorithms such as **Multi-Layer Perceptron**, **Decision tree**, and **Naïve Bayes classifier** are used for email spam filtering and malware detection.

1.1.3.7. Virtual Personal Assistant:

We have various virtual personal assistants such as **Google assistant**, **Alexa**, **Cortana**, **Siri**. As the name suggests, they help us in finding the information using our voice instruction. These assistants can help us in various ways just by our voice instructions such as Play music, call someone, Open an email, Scheduling an appointment, etc.

These virtual assistants use machine learning algorithms as an important part. These assistant record our voice instructions, send it over the server on a cloud, and decode it using ML algorithms and act accordingly.

1.1.3.8. Online Fraud Detection:

Machine learning is making our online transaction safe and secure by detecting fraud transaction. Whenever we perform some online transaction, there may be various ways that a fraudulent transaction can take place such as **fake accounts**, **fake ids**, and **steal money** in the middle of a transaction. So to detect this, **Feed Forward Neural network** helps us by checking whether it is a genuine transaction or a fraud transaction.

For each genuine transaction, the output is converted into some hash values, and these values become the input for the next round. For each genuine transaction, there is a specific pattern which gets change for the fraud transaction hence, it detects it and makes our online transactions more secure.

1.1.3.9. Stock Market trading:

Machine learning is widely used in stock market trading. In the stock market, there is always a risk of up and downs in shares, so for this machine learning's **long short term memory neural network** is used for the prediction of stock market trends.

1.1.3.10. Medical Diagnosis:

In medical science, machine learning is used for diseases diagnoses. With this, medical technology is growing very fast and able to build 3D models that can predict the exact position of lesions in the brain.

It helps in finding brain tumors and other brain-related diseases easily.

1.1.3.11. Automatic Language Translation:

Nowadays, if we visit a new place and we are not aware of the language then it is not a problem at all, as for this also machine learning helps us by converting the text into our known languages. Google's GNMT (Google Neural Machine Translation)

provide this feature, which is a Neural Machine Learning that translates the text into our familiar language, and it called as automatic translation.

The technology behind the automatic translation is a sequence to sequence learning algorithm, which is used with image recognition and translates the text from one language to another language.

1.1.4 Deep Learning:

Deep Learning is a subset of Machine Learning, which on the other hand is a subset of Artificial Intelligence. Artificial Intelligence is a general term that refers to techniques that enable computers to mimic human behavior. Machine Learning represents a set of algorithms trained on data that make all of this possible.

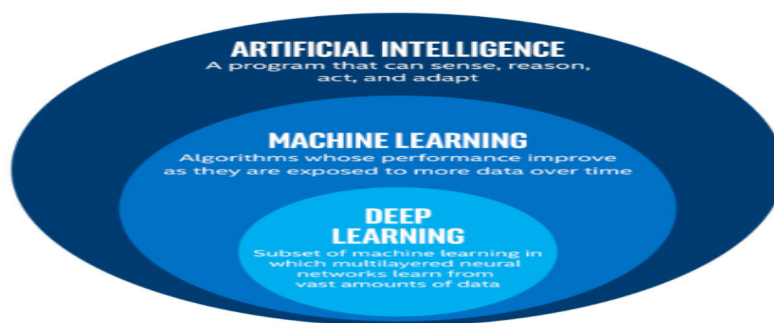


Fig:1.1.4.1 ML,DL and AI

Deep Learning, on the other hand, is just a type of Machine Learning, inspired by the structure of a human brain. Deep learning algorithms attempt to draw similar conclusions as humans would by continually analyzing data with a given logical structure. To achieve this, deep learning uses a multi-layered structure of algorithms called neural networks.

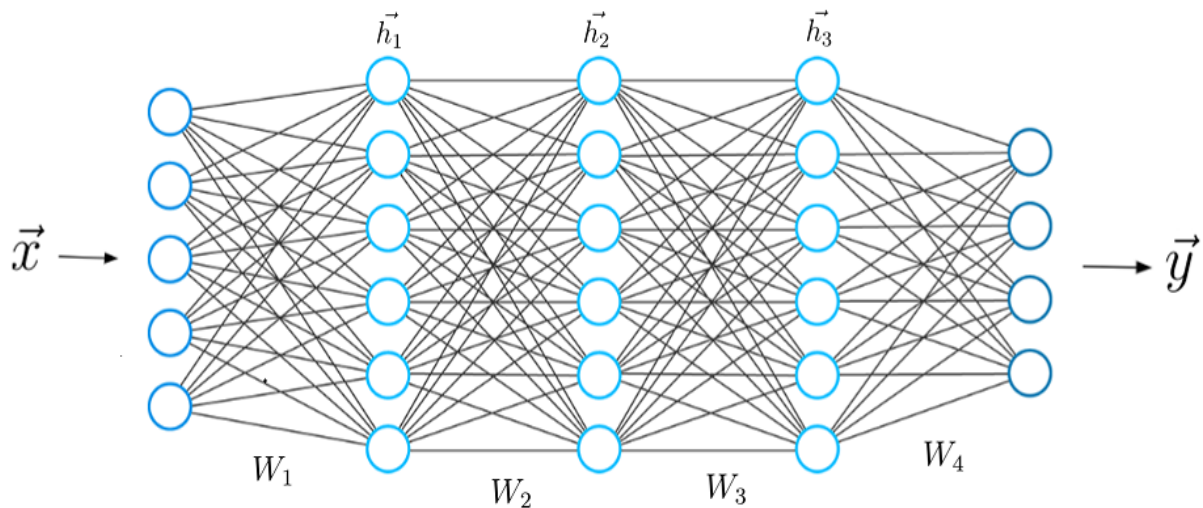


Fig 1.1.4.2 Neural Network

The design of the neural network is based on the structure of the human brain. Just as we use our brains to identify patterns and classify different types of information, neural networks can be taught to perform the same tasks on data.

The individual layers of neural networks can also be thought of as a sort of filter that works from gross to subtle, increasing the likelihood of detecting and outputting a correct result.

The human brain works similarly. Whenever we receive new information, the brain tries to compare it with known objects. The same concept is also used by deep neural networks.

Neural networks enable us to perform many tasks, such as clustering, classification or regression. With neural networks, we can group or sort unlabeled data according to similarities among the samples in this data. Or in the case of classification, we can train the network on a labeled dataset in order to classify the samples in this dataset into different categories.

Artificial neural networks have unique capabilities that enable deep learning models to solve tasks that machine learning models can never solve.

All recent advances in artificial intelligence in recent years are due to deep learning. Without deep learning, we would not have self-driving cars, chatbots or personal assistants like Alexa and Siri. The Google Translate app would continue to be as primitive as 10 years ago (before Google switched to neural networks for this App), and Netflix or YouTube would have no idea which movies or TV series we like or dislike. Behind all these technologies are neural networks.

We can even go so far as to say that today a new industrial revolution is taking place, driven by artificial neural networks and deep learning.

At the end of the day, deep learning is the best and most obvious approach to real machine intelligence we've had so far.

1.1.4.1 Why Deep Learning:

Long before deep learning was used, traditional machine learning methods were mainly used. Such as Decision Trees, SVM, Naïve Bayes Classifier and Logistic Regression.

These algorithms are also called flat algorithms. Flat here means that these algorithms can not normally be applied directly to the raw data (such as .csv, images, text, etc.). We need a preprocessing step called Feature Extraction.

The result of Feature Extraction is a representation of the given raw data that can now be used by these classic machine learning algorithms to perform a task. For example, the classification of the data into several categories or classes.

Feature Extraction is usually quite complex and requires detailed knowledge of the problem domain. This pre-processing layer must be adapted, tested and refined over several iterations for optimal results.

On the other side are the artificial neural networks of Deep Learning. These do not need the Feature Extraction step.

The layers are able to learn an implicit representation of the raw data directly and on their own. Here, a more and more abstract and compressed representation of the raw data is produced over several layers of an artificial neural-nets. This compressed representation of the input data is then used to produce the result. The result can be, for example, the classification of the input data into different classes.

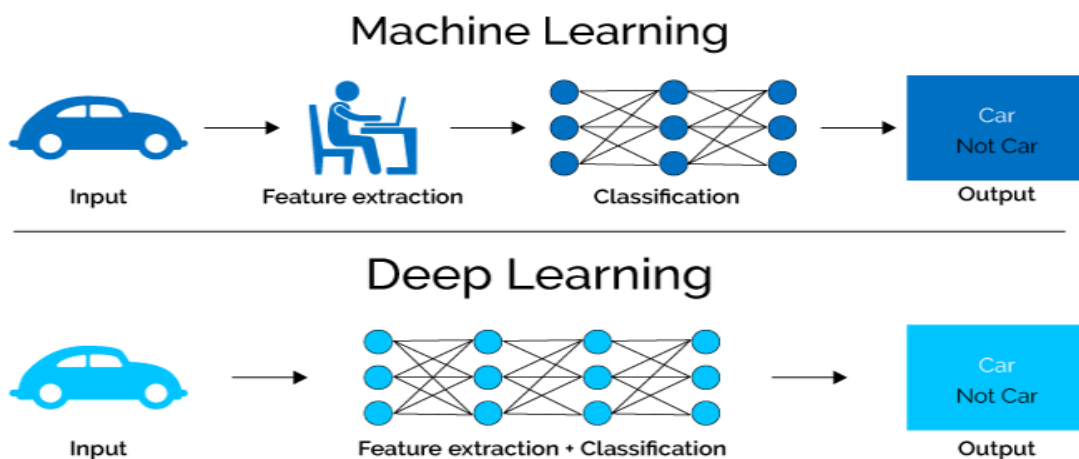


Fig:1.1.4.1.1 Difference between ML and DL

During the training process, this step is also optimized by the neural network to obtain the best possible abstract representation of the input data. This means that the models of deep learning thus require little to no manual effort to perform and optimize the feature extraction process.

Let us look at a concrete example. For example, if you want to use a machine learning model to determine if a particular image is showing a car or not, we humans first need to identify the unique features or features of a car (shape, size, windows, wheels, etc.) extract the feature and give them to the algorithm as input data.

In this way, the algorithm would perform a classification of the images. That is, in machine learning, a programmer must intervene directly in the action for the model to come to a conclusion.

In the case of a deep learning model, the feature extraction step is completely unnecessary. The model would recognize these unique characteristics of a car and make correct predictions.

That completely without the help of a human.

In fact, refraining from extracting the characteristics of data applies to every other task you'll ever do with neural networks. Just give the raw data to the neural network, the rest is done by the model.

The second huge advantage of Deep Learning and a key part in understanding why it's becoming so popular is that it's powered by massive amounts of data. The "Big Data Era" of technology will provide huge amounts of opportunities for new innovations in deep learning. As per Andrew Ng, the chief scientist of China's major search engine Baidu and one of the leaders of the Google Brain Project,

"The analogy to deep learning is that the rocket engine is the deep learning models and the fuel is the huge amounts of data we can feed to these algorithms."

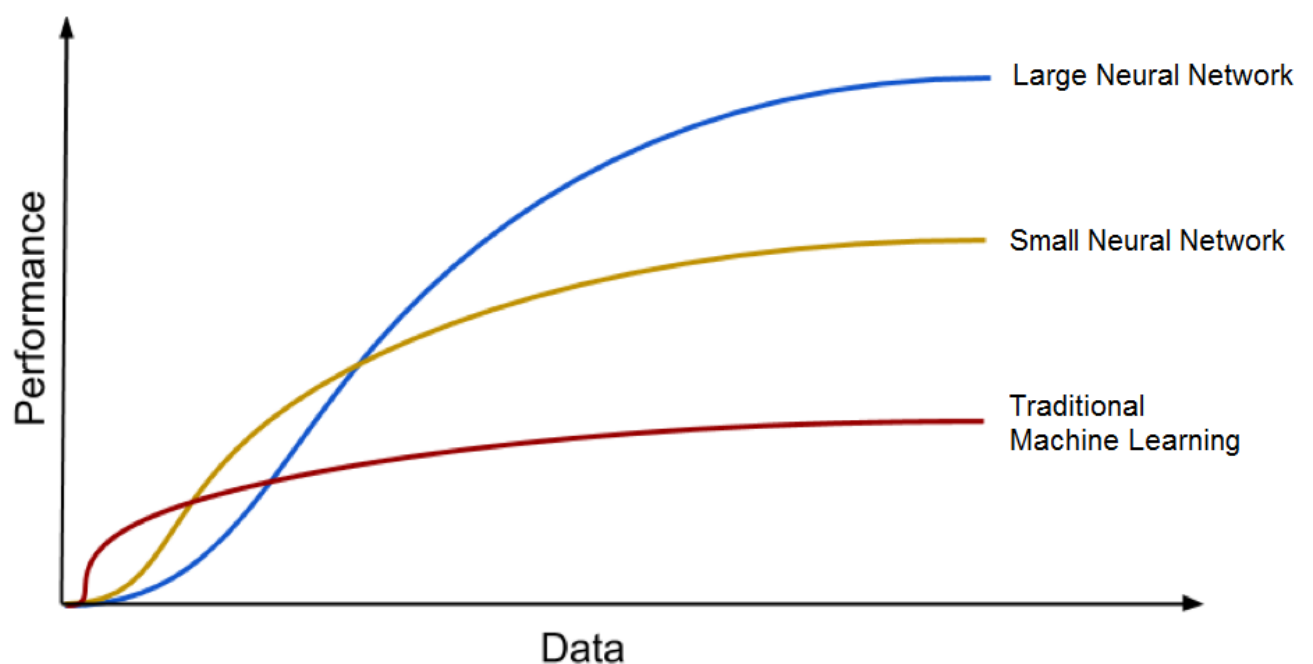


Fig:1.1.4.1.2 Neural Network Performance

Deep Learning models tend to increase their accuracy with the increasing amount of training data, whereas traditional machine learning models such as SVM and Naive Bayes classifier stop improving after a saturation point.

1.1.4.2 Artificial Neural Networks:

A neural network generally consists of a collection of connected units or nodes. We call these nodes neurons. These artificial neurons loosely model the biological neurons of our brain.

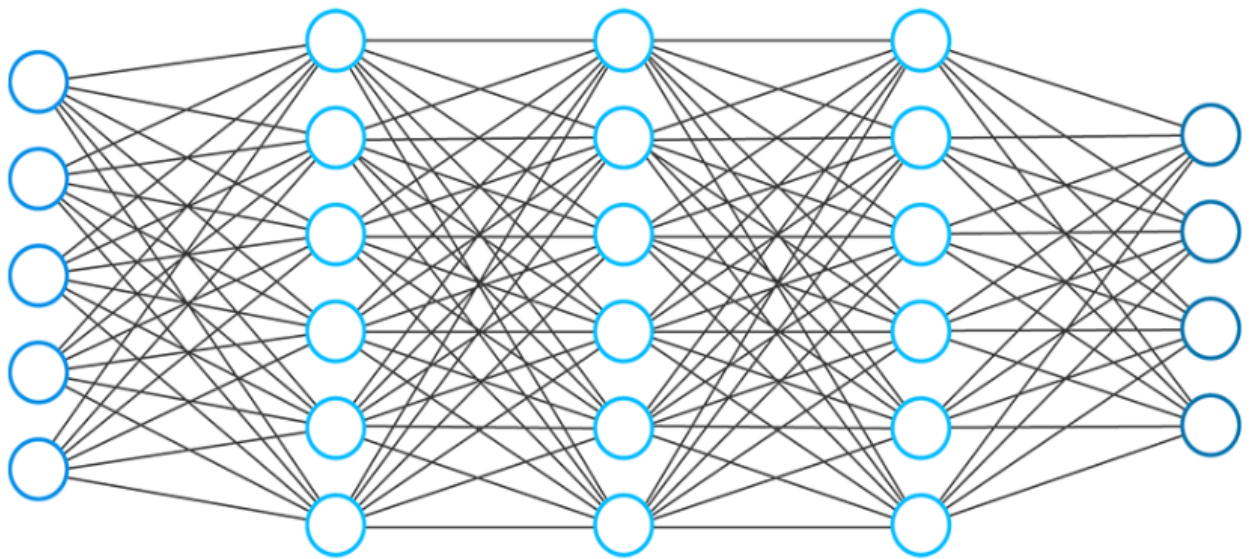


Fig:1.1.4.2.1 Artificial Neural Network

A neuron is simply a graphical representation of a numeric value (e.g. 1.2, 5.0, 42.0, 0.25, etc.). Any connection between two artificial neurons can be considered as an axon in a real biological brain.

The connections between the neurons are realized by so-called weights, which are also nothing more than numerical values

When an artificial **neural network learns, the weights between neurons are changing** and so does the strength of the connection. Meaning: Given training data and a particular task such as classification of numbers, we are looking for certain set weights that allow the neural network to perform the classification. The set of weights

is different for every task and every dataset. We can not predict the values of these weights in advance, but the **neural network has to learn them**. The process of learning we also call as training.

1.1.4.3 Typical Neural Network Architecture:

The typical neural network architecture consists of several layers. We call the first layer as the input layer.

The input layer receives the input $\mathbf{x}$, data from which the neural network learns. In our previous example of classification of handwritten numbers, these input $\mathbf{x}$ would represent the images of these numbers ($\mathbf{x}$ is basically an entire vector where each entry is a pixel).

The input layer has the same number of neurons as there are entries in the vector $\mathbf{x}$. Meaning: each input neuron represents one element in the vector $\mathbf{x}$.

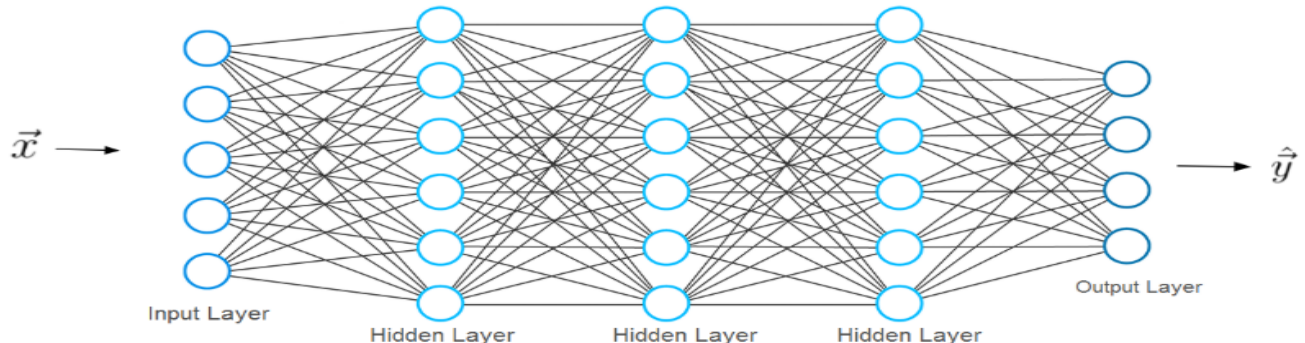


Fig:1.1.4.3.1 Neural Network Architecture

The last layer is called the output layer, which outputs a vector y representing the result that the neural network came up with. The entries in this vector represent the values of the neurons in the output layer. In our case of classification, each neuron in the last layer would represent a different class.

In this case, the value of an output neuron gives the probability that the handwritten digit given by the features $\mathbf{x}$ belongs to one of the possible classes (one of the digits

0–9). As you can imagine the number of output neurons must be the same as there are classes.

In order to obtain a prediction vector $\mathbf{y}$, the network must perform certain mathematical operations. These operations are performed in the layers between the input and output layers. We call these layers the hidden layers. Now let's discuss how the connections between the layers look like.

1.1.5 Computer Vision:

Computer vision is the field of computer science that focuses on replicating parts of the complexity of the human vision system and enabling computers to identify and process objects in images and videos in the same way that humans do. Until recently, computer vision only worked in limited capacity.

Thanks to advances in artificial intelligence and innovations in deep learning and neural networks, the field has been able to take great leaps in recent years and has been able to surpass humans in some tasks related to detecting and labeling objects.

One of the driving factors behind the growth of computer vision is the amount of data we generate today that is then used to train and make computer vision better.

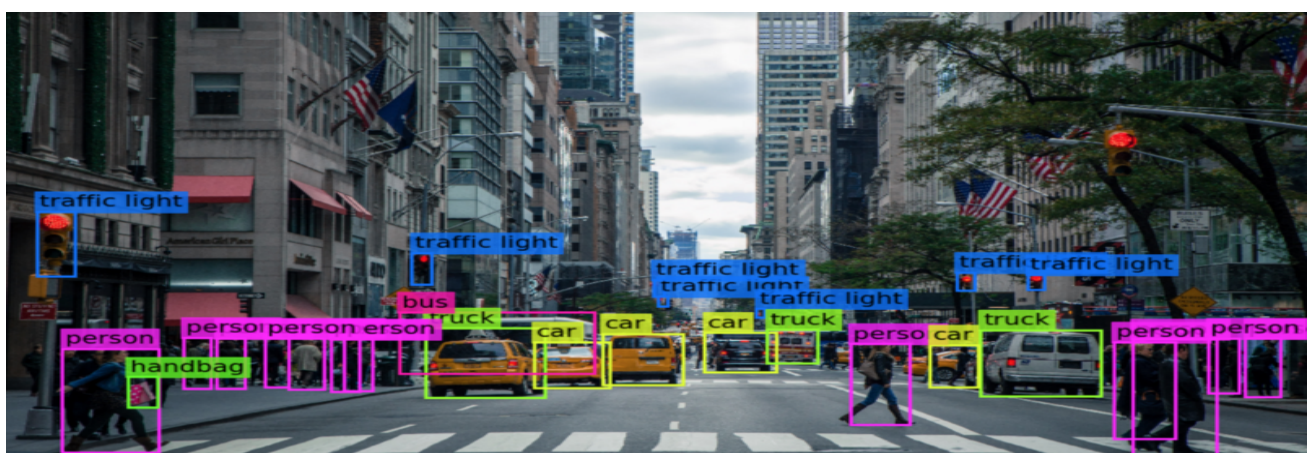


Fig:1.1.5.1 Computer Vision

Along with a tremendous amount of visual data (*more than 3 billion images are shared online every day*), the computing power required to analyze the data is now accessible. As the field of computer vision has grown with new hardware and algorithms so has the accuracy rates for object identification. In less than a decade, today's systems have reached 99 percent accuracy from 50 percent making them more accurate than humans at quickly reacting to visual inputs.

Early experiments in computer vision started in the 1950s and it was first put to use commercially to distinguish between typed and handwritten text by the 1970s, today the applications for computer vision have grown exponentially.

1.1.5.1 How Computer Vision Works:

One of the major open questions in both Neuroscience and Machine Learning is: How exactly do our brains work, and how can we approximate that with our own algorithms? The reality is that there are very few working and comprehensive theories of brain computation; so despite the fact that Neural Nets are supposed to “mimic the way the brain works,” nobody is quite sure if that's actually true.

The same paradox holds true for computer vision — since we're not decided on how the brain and eyes process images, it's difficult to say how well the algorithms used in production approximate our own internal mental processes.

On a certain level Computer vision is all about pattern recognition. So one way to train a computer how to understand visual data is to feed it images, lots of images thousands, millions if possible that have been labeled, and then subject those to various software techniques, or algorithms, that allow the computer to hunt down patterns in all the elements that relate to those labels.

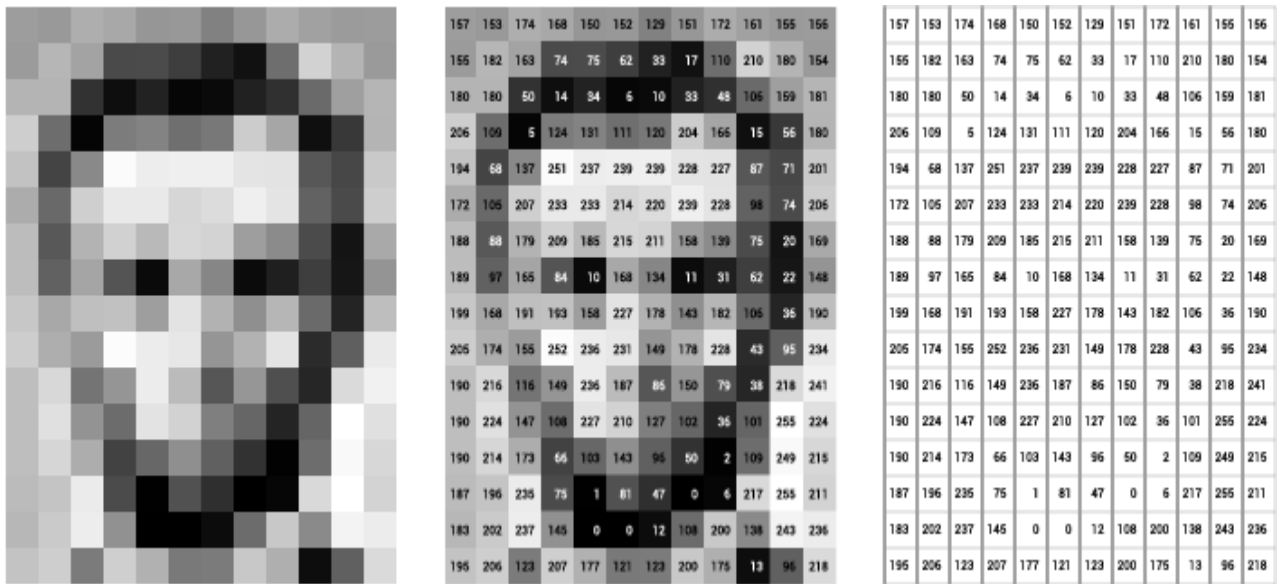


Fig:1.1.5.1.1 Pixel Value Representation

This way of storing image data may run counter to your expectations, since the data certainly *appears* to be two-dimensional when it is displayed. Yet, this is the case, since computer memory consists simply of an ever-increasing linear list of address spaces.

How the pixels look:

H	E	L	L	O
O	P	E	N	F
R	A	M	E	W
O	R	K	S	!

How the pixels are numbered:

0	1	2	3	4
5	6	7	8	9
10	11	12	13	14
15	16	17	18	19

How the pixels are stored in computer memory:

H	E	L	L	O	O	P	E	N	F	R	A	M	E	W	O	R	K	S	!
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

↑ ↑ ↑ ↑
0 1 2 3...

Fig:1.1.5.1.2 Pixel Storage

Computers usually read color as a series of 3 values — red, green, and blue (RGB) — on that same 0–255 scale. Now, each pixel actually has 3 values for the computer to store in addition to its position. If we were to colorize President Lincoln, that would lead to $12 \times 16 \times 3$ values, or 576 numbers.

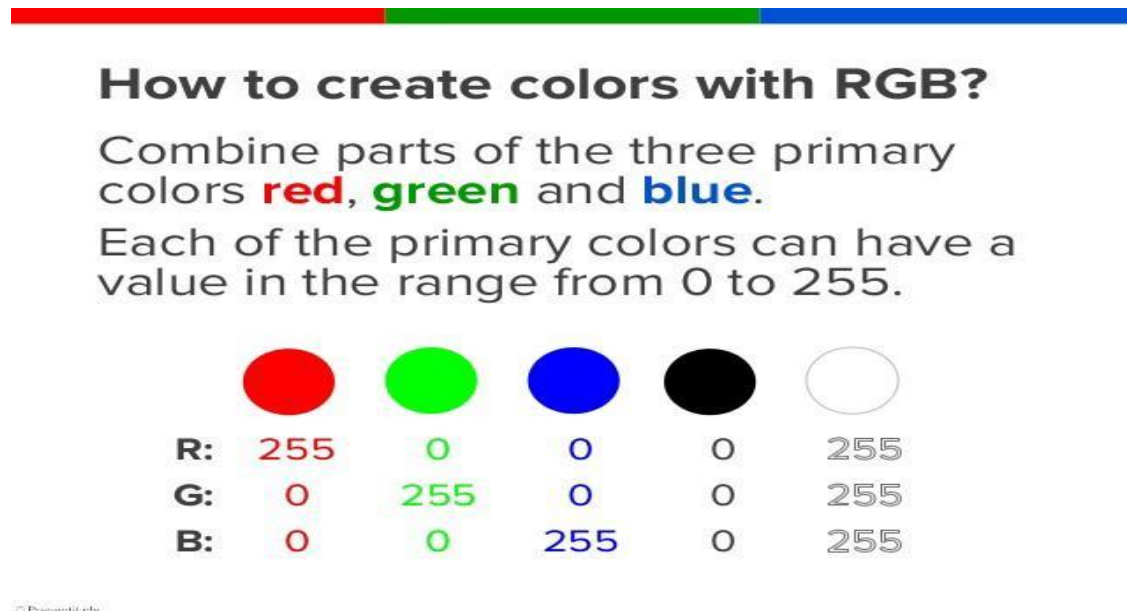


Fig:1.1.5.1.3 RGB Representation

That's a lot of memory to require for one image, and a lot of pixels for an algorithm to iterate over. But to train a model with meaningful accuracy especially when you're talking about Deep Learning you'd usually need tens of thousands of images, and the more the merrier.

1.1.5.2 Applications of Computer Vision:

- **Object Classification:** What broad category of object is in this photograph?
- **Object Identification:** Which type of a given object is in this photograph?
- **Object Verification:** Is the object in the photograph?
- **Object Detection:** Where are the objects in the photograph?
- **Object Landmark Detection:** What are the key points for the object in the photograph?
- **Object Segmentation:** What pixels belong to the object in the image?

- **Object Recognition:** What objects are in this photograph and where are they?

Outside of just recognition, other methods of analysis include:

- **Video motion analysis** uses computer vision to estimate the velocity of objects in a video, or the camera itself.
- In **image segmentation**, algorithms partition images into multiple sets of views.
- **Scene reconstruction** creates a 3D model of a scene inputted through images or video.
- In **image restoration**, noise such as blurring is removed from photos using Machine Learning based filters.

1.1.6 Vision Transformers:

Vision Transformers were first introduced in the paper AN IMAGE IS WORTH 16X16 WORDS: TRANSFORMERS FOR IMAGE RECOGNITION AT SCALE by the Google Brain team in late October 2020.

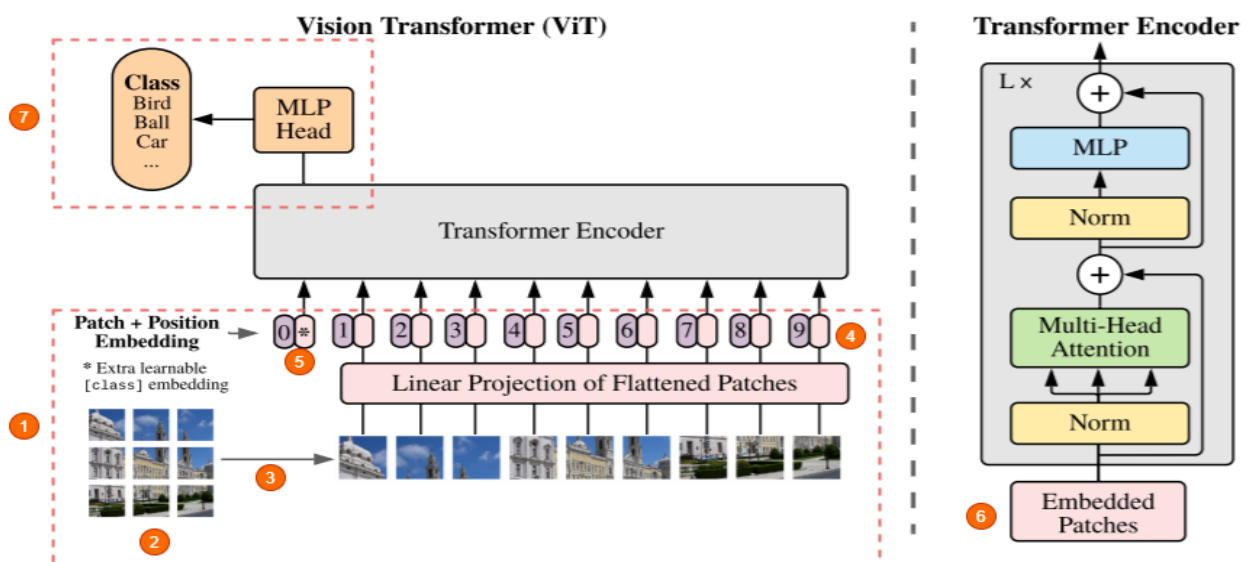


Fig:1.1.6.1 Vision Transformer

(1) They are only using the Encoder part of the transformer but the difference is in how they are feeding the images into the network.

(2) They are breaking down the image into fixed size patches. So one of these patches can be of dimension 16x16 or 32x32 as proposed in the paper. More patches means more simpler it is to train these networks as the patches themselves get smaller. Hence we have that in the title - "An Image is worth 16x16 words".

(3) The patches are then unrolled (flattened) and sent for further processing into the network.

(4) Unlike NNs here the model has no idea whatsoever about the position of the samples in the sequence, here each sample is a patch from the input image. So the image is fed **along with a positional embedding vector** and into the encoder. One thing to note here is the positional embeddings are also learnable so you don't actually feed hard-coded vectors w.r.t to their positions.

(5) There is also a special token at the start just like BERT.

(6) So each image patch is first unrolled (flattened) into a big vector and gets multiplied with an embedding matrix which is also learnable, creating embedded patches. And these embedded patches are combined with the positional embedding vector and that gets fed into the Transformer.

(7) With the only difference being, instead of a decoder the output from the encoder is passed directly into a Feed Forward Neural Network to obtain the classification output.

1.2 MOTIVATION

The present way of separating waste/garbage is the hand-picking method, whereby someone is employed to separate out the different objects/materials. The person, who separate waste, is prone to diseases due to the harmful substances in the garbage. With this in mind, it motivated us to develop an automated system which is able to sort the waste. and this system can take short time to sort the waste, and it will be more accurate in sorting than the manual way. With the system in place, the beneficial separated waste can still be recycled and converted to energy and fuel for

the growth of the economy. The system that is developed for the separation of the accumulated waste is based on the Vision Transformer.

1.3 RESEARCH AND SIGNIFICANCE

The Vision Transformer treats an input image as a sequence of patches, akin to a series of word embeddings generated by a natural language processing (NLP) Transformer. ViT demonstrates excellent performance when trained on sufficient data, outperforming a comparable state-of-the-art CNN with four times fewer computational resources. Using ViT separation process of the waste will be faster and intelligent using the proposed waste material classification system without or reducing human involvement.

CHAPTER 2

AIM AND SCOPE

2.1 AIM OF PROJECT

The primary aim of this proposed system i.e. (Intelligent Waste Classification System using Vision Transformer) is to deal with the billion tons of waste around the world. The present way of separating waste/garbage is the hand-picking method, whereby someone is employed to separate out the different objects/materials. The person, who separates waste, is prone to diseases due to the harmful substances in the garbage. With this in mind, it motivated us to develop an automated system which is able to sort the waste. and this system can take short time to sort the waste, and it will be more accurate in sorting than the manual way.

2.1.1 OBJECTIVES

The proposed model “Intelligent Waste Classification System using Vision Transformer” is based on the new technologies which makes the whole process efficient. It is going to classify the waste into different groups/types such as glass, metal, paper, cardboard and plastic and trash using intelligent waste classification system instead of hand-picking method. It is going to decrease the training time by 80%. It is going to understand the connection between inputs. It is going to handle variable-sized input using stacks of Self-Attention instead of CNNs and RNNs.

2.2 SCOPE

This model will help to achieve excellent results with pure transformer architecture applied directly to a sequence of image patches for classification tasks. It also outperforms the state-of-the-art convolutional networks on many image classification tasks while requiring substantially fewer computational resources (at least 4 times fewer than SOTA CNN) to pre-train. An automated recognition system using deep learning algorithm has been implemented to classify wastes as biodegradable and non-biodegradable. Efficient segregation of solid wastes helps to reduce the amount

of waste buried in the ground, thereby improving the recycling rate, and safeguards the soil from pollution. Careful and robust classification of waste materials is important given the strict controls required for storage, treatment and disposal of hazardous waste, in addition to the higher costs involved compared with other waste types

2.2.1 ADVANTAGES

- Time saving as it requires less amount of datasets.
- High Accuracy with Less Computation Time for Training.
- It provides authentication and keeps the system secured.
- It gives the capability to understand the connection between inputs

CHAPTER 3

SYSTEM DESIGN & METHODOLOGY

3.1 EXISTING SYSTEM

A Convolutional Neural Network (ConvNet/CNN) is a Deep Learning algorithm which can take in an input image, assign importance (learnable weights and biases) to various aspects/objects in the image and be able to differentiate one from the other. The pre-processing required in a ConvNet is much lower as compared to other classification algorithms. While in primitive methods filters are hand-engineered, with enough training, ConvNets have the ability to learn these filters/characteristics. The architecture of a ConvNet is analogous to that of the connectivity pattern of Neurons in the Human Brain and was inspired by the organization of the Visual Cortex. Individual neurons respond to stimuli only in a restricted region of the visual field known as the Receptive Field. A collection of such fields overlap to cover the entire visual area.

3.2 SYSTEM CAPABILITIES

But CNN do not encode the position and orientation of the object into their predictions. They completely lose all their internal data about the pose and the orientation of the object and they route all the information to the same neurons that may not be able to deal with this kind of information. A CNN makes predictions by looking at an image and then checking to see if certain components are present in that image or not.

Artificial neurons output a single scalar. In addition, CNNs use convolutional layers that, for each kernel, replicate that same kernel's weights across the entire input volume and then output a 2D matrix, where each number is the output of that kernel's convolution with a portion of the input volume. So we can look at that 2D matrix as output of

replicated feature detector. Then all kernel's 2D matrices are stacked on top of each other to produce output of a convolutional layer.

Then, we try to achieve viewpoint invariance in the activities of neurons. We do this by the means of max pooling (e.g. 2×2 pixels) that consecutively looks at regions in the above described 2D matrix and selects the largest number in each region. As result, we get what we wanted — invariance of activities. Invariance means that by changing the input a little, the output still stays the same. And activity is just the output signal of a neuron. In other words, when in the input image we shift the object that we want to detect by a little bit, networks activities (outputs of neurons) will not change because of max pooling and the network will still detect the object.

The above described mechanism is not very good, because max pooling loses valuable information and also does not encode relative spatial relationships between features. Because of this, CNN are not actually invariant to large transformations of the input data.

3.3 REQUIREMENT SPECIFICATION

This proposed software runs effectively on a computing system that has the minimum requirements. Undertaking all the equipment necessities are not satisfied but rather exist in their systems administration between the customer's machines already. So the main need is to introduce appropriate equipment for the product. The requirements are split into two categories, namely:

3.3.1 Software Requirements

The basic software requirements to run the program are:

- Anaconda Navigator.
- Jupyter Notebook.
- Windows 7 and above.
- Tensorflow Library file.
- Keras Library file.
- Pytorch Library file.
- Vision Transformer Library file.

3.3.2 Hardware Requirements

The basic hardware required to run the program are:

Good internet connection.

8GB+Ram.

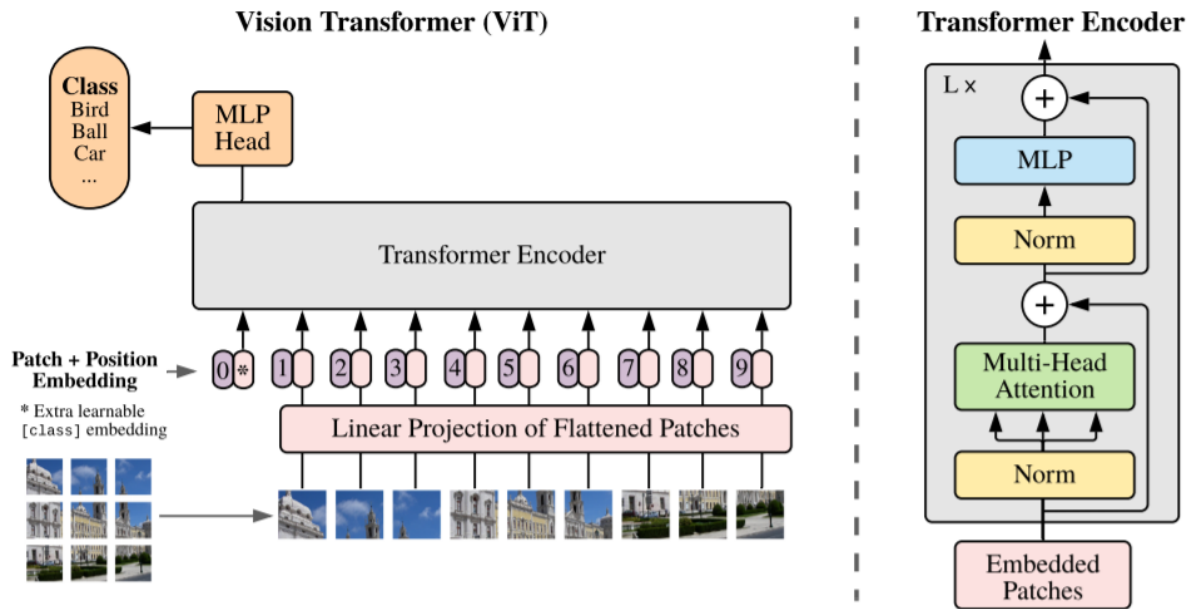
50 GB of free storage.

i5 6th generation processor or better.

CHAPTER 4

SOFTWARE DEVELOPMENT METHODOLOGY

4.1 DESCRIPTION OF DIAGRAM



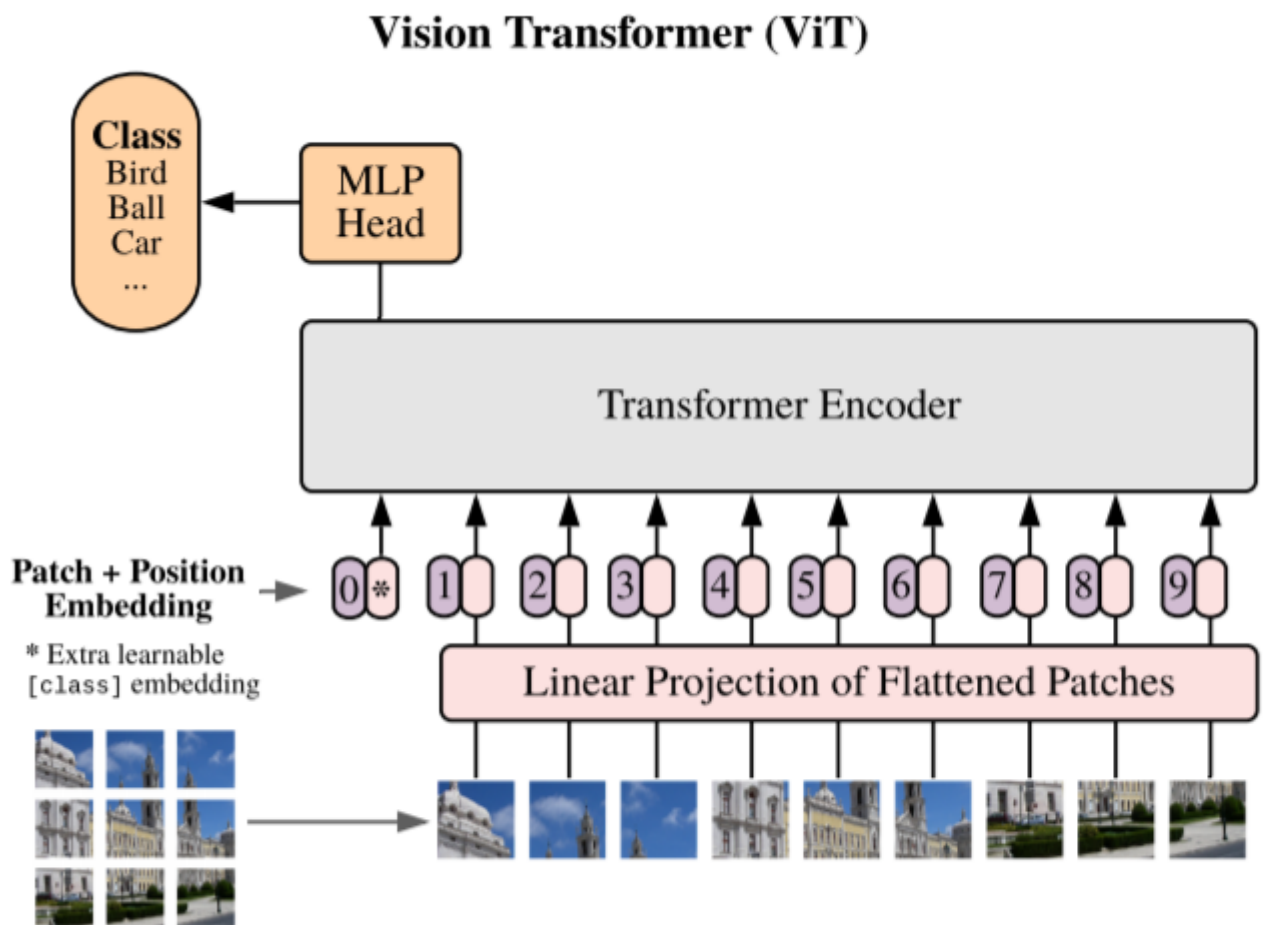


Fig 4.1.1: Architecture DIAGRAM

The total architecture is called Vision Transformer (ViT in short). Let's examine it step by step.

1. Split an image into patches.
2. Flatten the patches.
3. Produce lower-dimensional linear embeddings from the flattened patches
4. Add positional embeddings.
5. Feed the sequence as an input to a standard transformer encoder.
6. Pretrain the model with image labels (fully supervised on a huge dataset).
7. Fine-tune on the downstream data-set for image classification.

Image patches are basically the sequence tokens (like words). In fact, the encoder block is identical to the original transformer proposed by Vaswani et al. (2017) as we have extensively described:

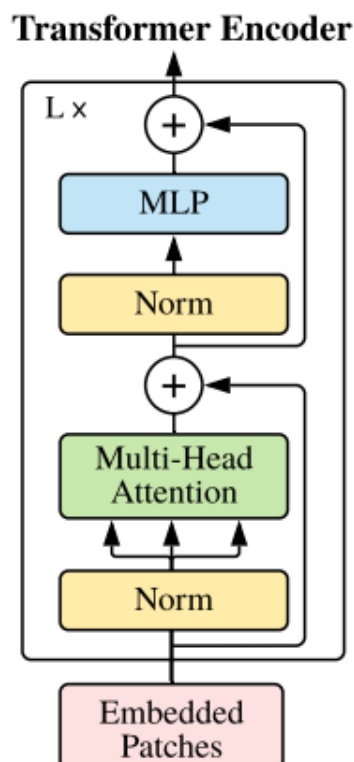


Fig 4.2 Transformer Encoder

The only thing that changes is the number of those blocks. To this end, and to further prove that with more data they can train larger ViT variants, 3 models were proposed:

Model	Layers	Hidden size D	MLP size	Heads	Params
ViT-Base	12	768	3072	12	86M
ViT-Large	24	1024	4096	16	307M
ViT-Huge	32	1280	5120	16	632M

Fig 4.3 ViT Models

Heads refer to multi-head attention, while the MLP size refers to the blue module in the figure. MLP stands for multi-layer perceptron but it's actually a bunch of linear transformation layers.

Hidden size D is the embedding size, which is kept fixed throughout the layers. Why keep it fixed? So that we can use short residual skip connections.

In case you missed it, there is **no** decoder in the game. Just an extra linear layer for the final classification called MLP head.

ViT is pretrained on the large dataset and then fine-tuned to small ones. The only modification is to discard the prediction head (MLP head) and attach a new $D \times K$ linear layer, where K is the number of classes of the small dataset.

To fine-tune in higher resolutions, 2D interpolation of the pre-trained position embeddings is performed. The reason is that they model positional embeddings with trainable linear layers. Having that said, the key engineering part of this paper is all about feeding an image in the transformer.

4.2 Positional embeddings

Even though many positional embedding schemes were applied, no significant difference was found. This is probably due to the fact that the transformer encoder operates on a patch-level. Learning embeddings that capture the order relationships between patches (spatial information) is not so crucial. It is relatively easier to understand the relationships between patches of $P \times P$ than of a full image Height $\times$ Width.

Hence, after the low-dimensional linear projection, a **trainable** position embedding is added to the patch representations. It is interesting to see what these position embeddings look like after training.

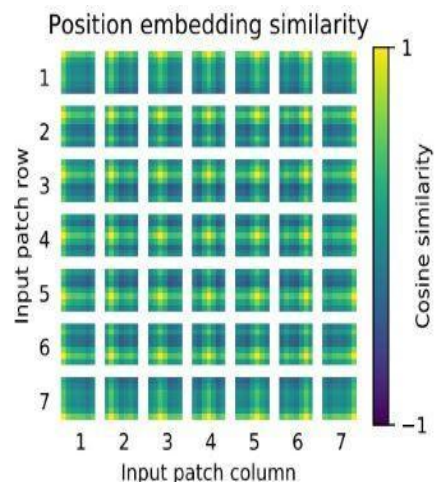


Fig 4.2.1 Positional Embedding

First, there is some kind of 2D structure. Second, patterns across rows (and columns) have similar representations. For high resolutions, a sinusoidal structure was used.

4.3 Key findings

In the early conv days, we used to visualize the early layers.

Why?

Because we believe that well-trained networks often show nice and **smooth** filters.

We don't need successive conv. layers to get to 128-away pixels anymore. With convolutions without dilation, the receptive field is increased linearly. Using self-attention we have interaction between pixels representations in the 1st layer and pairs of representations in the 2nd layer and so on.

4.4 Representing an image as a sequence of patches

I was also super curious how you can elegantly reshape the image in patches. For an input image $(x) \in \mathbb{R}^{H \times W \times C}$ and patch size p , we want to create N image patches denoted as $(x)_p \in \mathbb{R}^{N \times (P \times C)}$, where $N = HWP^2$. N is the sequence length similar to the words of a sentence.

For an input image $x \in \mathbb{R}^{H \times W \times C}$ and patch size P we want to create N image patches denoted as $x_p \in \mathbb{R}^{N \times (P \times C)}$, where $N = HWP^2$. N is the sequence length/tokens similar to the word embeddings for a sentence.

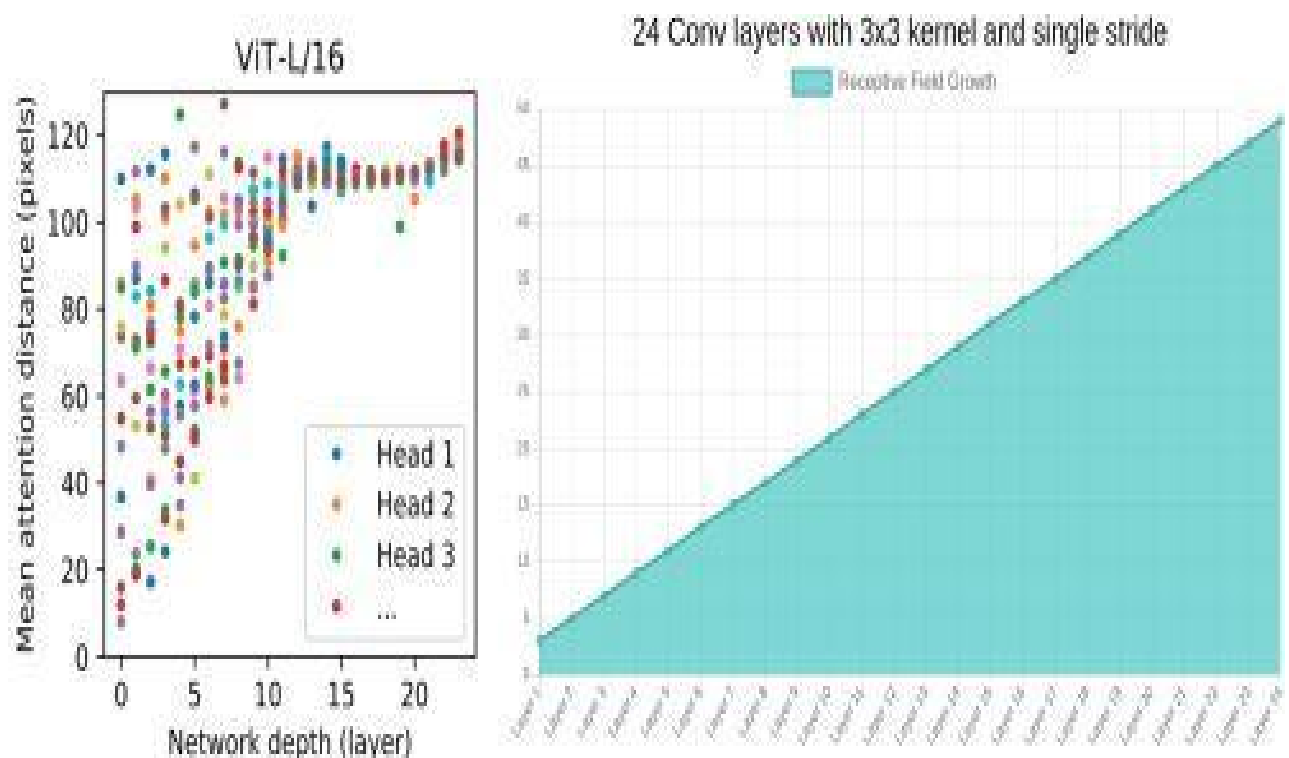


Fig 4.4.1 GRAPH OF VIT PERFORMANCE

Based on the diagram on the left from ViT, one can argue that:

There are indeed heads that attend to the whole patch already in the early layers. One can justify the performance gain based on the early access pixel interactions. It seems more critical for the early layers to have access to the whole patch (global info). In other words, the heads that belong to the upper left part of the image may be the core reason for superior performance. Interestingly, the attention distance increases with network depth similar to the receptive field of local operations. There are also attention heads with consistently small attention distances in the low layers. On the right, a 24-layer with standard 3x3 convolutions has a receptive field of less than 50. We would approximately need 50 conv layers, to attend to a ~100 receptive field, without dilation or pooling layers. To enforce this idea of highly localized attention heads, the authors experimented with hybrid models that apply a ResNet before the Transformer. They found less highly localized heads, as expected. Along with filter visualization, it suggests that it may serve a similar function as early convolutional layers in CNNs.

4.5 Attention distance and visualization

However, I find it critical to understand how they measured the mean attention distance. It's analogous to the receptive field, but not exactly the same. Attention distance was computed as the average distance between the query pixel and the rest of the patch, multiplied by the attention weight. They used 128 example images and averaged their results. An example: if a pixel is 20 pixels away and the attention weight is 0.5 the distance is 10. Finally, the model attends to image regions that are semantically relevant for classification, as illustrated below:

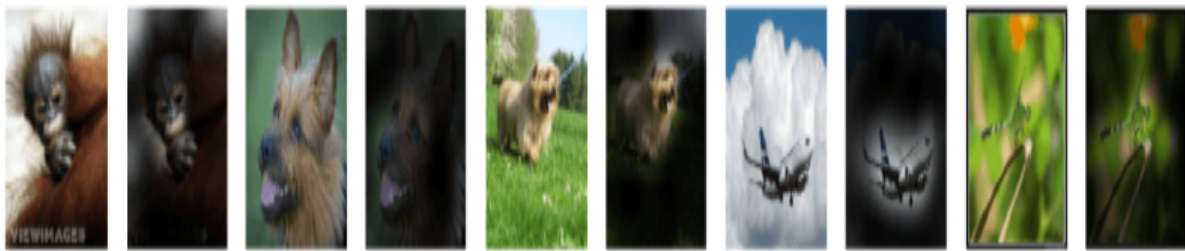


Fig4.5.1 Classification

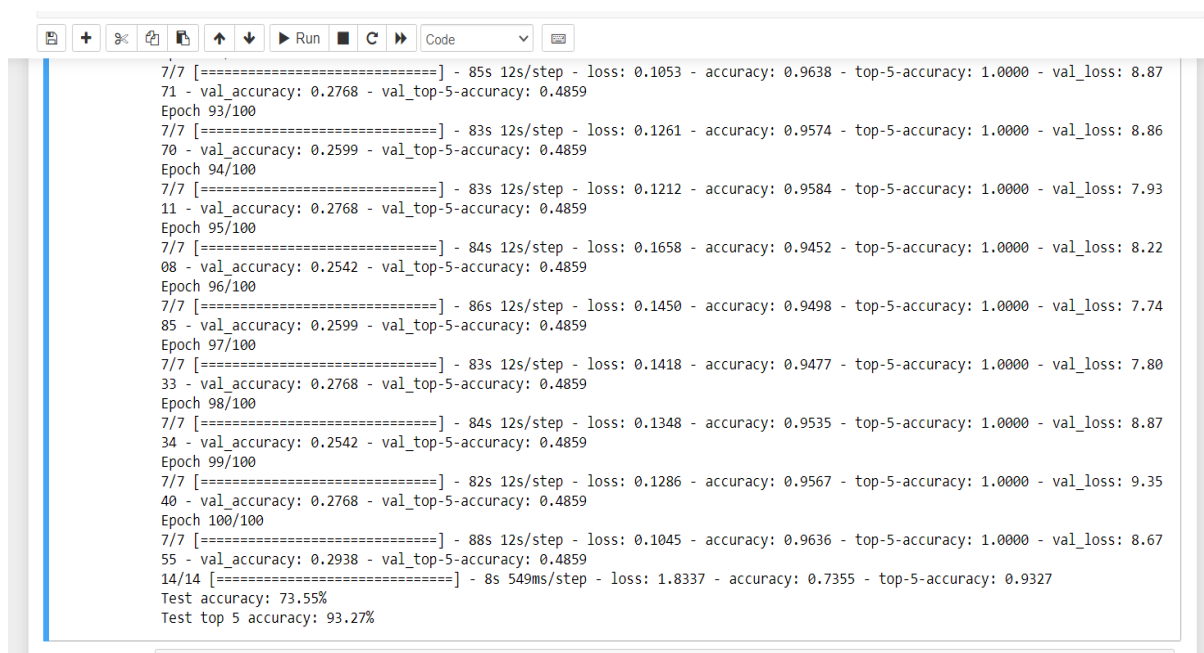
The model learns to encode distance within the image in the similarity of position embedding, i.e. closer patches tend to have more similar position embedding. Further, the row-column structure appears; patches in the same row/column have similar embedding. Finally, a sinusoidal structure is sometimes apparent for larger grids. Self-attention allows ViT to integrate information across the entire image even in the lowest layers. We investigate to what degree the network makes use of this capability. The “attention distance” is analogous to receptive field size in CNNs. We find that some heads attend to most of the image already in the lowest layers, showing that the ability to integrate information globally is indeed used by the model.

This highly localized attention is less pronounced in hybrid models that apply a ResNet before the Transformer, suggesting that it may serve a similar function as early convolutional layers in CNNs. Further, the attention distance increases with network depth.

CHAPTER 5

RESULTS AND DISCUSSION

As a result of this system getting introduced in Waste Management, it will segregate the waste products into their respective classes. The public will be able to understand recycling process better which will further result in taking quick actions.



```
7/7 [=====] - 85s 12s/step - loss: 0.1053 - accuracy: 0.9638 - top-5-accuracy: 1.0000 - val_loss: 8.87
71 - val_accuracy: 0.2768 - val_top-5-accuracy: 0.4859
Epoch 93/100
7/7 [=====] - 83s 12s/step - loss: 0.1261 - accuracy: 0.9574 - top-5-accuracy: 1.0000 - val_loss: 8.86
70 - val_accuracy: 0.2599 - val_top-5-accuracy: 0.4859
Epoch 94/100
7/7 [=====] - 83s 12s/step - loss: 0.1212 - accuracy: 0.9584 - top-5-accuracy: 1.0000 - val_loss: 7.93
11 - val_accuracy: 0.2768 - val_top-5-accuracy: 0.4859
Epoch 95/100
7/7 [=====] - 84s 12s/step - loss: 0.1658 - accuracy: 0.9452 - top-5-accuracy: 1.0000 - val_loss: 8.22
08 - val_accuracy: 0.2542 - val_top-5-accuracy: 0.4859
Epoch 96/100
7/7 [=====] - 86s 12s/step - loss: 0.1450 - accuracy: 0.9498 - top-5-accuracy: 1.0000 - val_loss: 7.74
85 - val_accuracy: 0.2599 - val_top-5-accuracy: 0.4859
Epoch 97/100
7/7 [=====] - 83s 12s/step - loss: 0.1418 - accuracy: 0.9477 - top-5-accuracy: 1.0000 - val_loss: 7.80
33 - val_accuracy: 0.2768 - val_top-5-accuracy: 0.4859
Epoch 98/100
7/7 [=====] - 84s 12s/step - loss: 0.1348 - accuracy: 0.9535 - top-5-accuracy: 1.0000 - val_loss: 8.87
34 - val_accuracy: 0.2542 - val_top-5-accuracy: 0.4859
Epoch 99/100
7/7 [=====] - 82s 12s/step - loss: 0.1286 - accuracy: 0.9567 - top-5-accuracy: 1.0000 - val_loss: 9.35
40 - val_accuracy: 0.2768 - val_top-5-accuracy: 0.4859
Epoch 100/100
7/7 [=====] - 88s 12s/step - loss: 0.1045 - accuracy: 0.9636 - top-5-accuracy: 1.0000 - val_loss: 8.67
55 - val_accuracy: 0.2938 - val_top-5-accuracy: 0.4859
14/14 [=====] - 8s 549ms/step - loss: 1.8337 - accuracy: 0.7355 - top-5-accuracy: 0.9327
Test accuracy: 73.55%
Test top 5 accuracy: 93.27%
```

Fig 5.1 Result

After 100 epochs, the ViT model achieves around 73% accuracy and 93% top-5 accuracy on the test data. These are not competitive results on the Waste Classification dataset, as a CNN trained from scratch on the same data can achieve 89% accuracy.

To improve the model quality without pre-training, we can try to train the model for more epochs, use a larger number of Transformer layers, resize the input images, change the patch size, or increase the projection dimensions

CHAPTER 6

CONCLUSION AND FUTURE WORK

CONCLUSION

This system will make the system of Waste classification more efficient and flexible. This system can be used to automatically **classify** waste and help in reducing human intervention and preventing infection and pollution. From the result, when tested against the trash dataset, we got an accuracy of 87%. The separation process of the waste will be faster and intelligent using our system without or reducing human involvement. If more image is added to the dataset, the system accuracy can be improved. In the future, we will tend to improve our system to be able to categories more waste item, by turning some of the parameters used.

FUTURE WORK

In future, system will provide waste collection and rubbish disposal play an extremely important role in the global cleanliness and sustainability drive, with people's health. Waste management and recycling collection can help conserve our planet's natural beauty which can be flawed by thoughtless disposal of waste, fly-tipping and senseless littering. Landscapes can be ruined through littering and places of tourist interest can lose their attraction; it is also blight for those who live in areas where waste collection and recycling is not managed effectively and responsibly. Natural beauty is a legacy and a right for future generations and conserving it, as well as our natural resources, for their benefit is our responsibility today.

REFERENCES

- [1] Olugboja Adedeji, Zenghui Wang, " Intelligent Waste Classification System Using Deep Learning Convolutional Neural Network", 2nd International Conference on Sustainable Materials Processing and Manufacturing(2019), 607–612.
- [2] Umut Özkaya and Levent Seyfi, "Fine-Tuning Models Comparisons on Garbage Classification for Recyclability", (2018).
- [3] Bichen Wu, Chenfeng Xu, Xiaoliang Dai, " Visual Transformers: Token-based Image Representation and Processing for Computer Vision", (2020) .
- [4] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, " An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale", (2020).
- [5] https://www.knowledgeisle.com/wp-content/uploads/2019/12/2-Aur%C3%A9lien-G%C3%A9ron-Hands-On-Machine-Learning-with-Scikit-Learn-Keras-and-Tensorflow_-Concepts-Tools-and-Techniques-to-Build-Intelligent-Systems-O%E2%80%99Reilly-Media-2019.pdf
- [6] <https://towardsdatascience.com/wtf-is-image-classification-8e78a8235acb>
- [7] <https://towardsdatascience.com/end-to-end-pipeline-for-setting-up-multiclass-image-classification-for-data-scientists-2e051081d41c>
- [8] <https://medium.com/swlh/convolutional-neural-networks-for-multiclass-image-classification-a-beginners-guide-to-6dbc09fabbd>
- [9] https://scikit-learn.org/stable/user_guide.html
- [10] https://devdocs.io/scikit_learn/
- [11] <https://medium.com/analytics-vidhya/vision-transformers-bye-bye-convolutions-e929d022e4ab>
- [12] <https://medium.com/lsc-psd/introduction-of-self-attention-layer-in-transformer-fc7bff63f3bc#:~:text=%20Self-Attention%20%201%20Basic%20Structure.%20For%20every,In%20these%20two%20sentences%2C%20the%20relation..%20More%20>

APPENDIX

A. Sample Code:

```
import numpy as np
import pandas as pd
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
import tensorflow_addons as tfa
import matplotlib.pyplot as plt
import seaborn as sns
import os
from PIL import Image
import math
import cv2
from keras.preprocessing.image import ImageDataGenerator, load_img, img_to_array,
array_to_img
from sklearn.model_selection import train_test_split
from keras.preprocessing import image
from numpy import array
from numpy import argmax
from keras.utils import to_categorical
from numpy import array
from numpy import argmax
from keras.utils import to_categorical
```

```
data_augmentation = keras.Sequential(
    [
        layers.experimental.preprocessing.Normalization(),
        layers.experimental.preprocessing.Resizing(image_size, image_size),
        layers.experimental.preprocessing.RandomFlip("horizontal"),
        layers.experimental.preprocessing.RandomRotation(factor=0.02),
        layers.experimental.preprocessing.RandomZoom(
            height_factor=0.2, width_factor=0.2
        ),
    ],
    name="data_augmentation",
)
# Compute the mean and the variance of the training data for normalization.
data_augmentation.layers[0].adapt(x_train)
```

```
def create_vit_classifier():
    inputs = layers.Input(shape=input_shape)
    # Augment data.
```

```

augmented = data_augmentation(inputs)
# Create patches.
patches = Patches(patch_size)(augmented)
# Encode patches.
encoded_patches = PatchEncoder(num_patches, projection_dim)(patches)

# Create multiple layers of the Transformer block.
for _ in range(transformer_layers):
    # Layer normalization 1.
    x1 = layers.LayerNormalization(epsilon=1e-6)(encoded_patches)
    # Create a multi-head attention layer.
    attention_output = layers.MultiHeadAttention(
        num_heads=num_heads, key_dim=projection_dim, dropout=0.1
    )(x1, x1)
    # Skip connection 1.
    x2 = layers.Add()([attention_output, encoded_patches])
    # Layer normalization 2.
    x3 = layers.LayerNormalization(epsilon=1e-6)(x2)
    # MLP.
    x3 = mlp(x3, hidden_units=transformer_units, dropout_rate=0.1)
    # Skip connection 2.
    encoded_patches = layers.Add()([x3, x2])

# Create a [batch_size, projection_dim] tensor.
representation = layers.LayerNormalization(epsilon=1e-6)(encoded_patches)
representation = layers.Flatten()(representation)
representation = layers.Dropout(0.5)(representation)
# Add MLP.
features = mlp(representation, hidden_units=mlp_head_units, dropout_rate=0.5)
# Classify outputs.
logits = layers.Dense(num_classes)(features)
# Create the Keras model.
model = keras.Model(inputs=inputs, outputs=logits)
return model

```

```

def run_experiment(model):
    optimizer = tfa.optimizers.AdamW(
        learning_rate=learning_rate, weight_decay=weight_decay
    )

    model.compile(
        optimizer=optimizer,
        loss=keras.losses.SparseCategoricalCrossentropy(from_logits=True),
        metrics=[keras.metrics.SparseCategoricalAccuracy(name="accuracy"),
                  keras.metrics.SparseTopKCategoricalAccuracy(5, name="top-5-accuracy"),
        ],
    )

```

```

checkpoint_filepath = "./tmp/checkpoint"
checkpoint_callback = keras.callbacks.ModelCheckpoint(
    checkpoint_filepath,
    monitor="val_accuracy",
    save_best_only=True,
    save_weights_only=True,
)

history = model.fit(
    x=x_train,
    y=y_train,
    batch_size=batch_size,
    epochs=100,
    validation_split=0.1,
    #callbacks=[checkpoint_callback],
)

#model.load_weights(checkpoint_filepath)
_, accuracy, top_5_accuracy = model.evaluate(x_test, y_test)
print(f"Test accuracy: {round(accuracy * 100, 2)}%")
print(f"Test top 5 accuracy: {round(top_5_accuracy * 100, 2)}%")

return history

vit_classifier = create_vit_classifier()
history = run_experiment(vit_classifier)

```

B.SCREENSHOTS

```
In [310]: import numpy as np
import pandas as pd
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
import tensorflow_addons as tfa
import matplotlib.pyplot as plt
import seaborn as sns
import os
from PIL import Image
import math
import cv2
from keras.preprocessing.image import ImageDataGenerator, load_img, img_to_array, array_to_img
from sklearn.model_selection import train_test_split
from keras.preprocessing import image
from numpy import array
from numpy import argmax
from keras.utils import to_categorical
from numpy import array
from numpy import argmax
from keras.utils import to_categorical
```

```
In [311]: x_train = []
y_train = []
shape = (32,32)
train_path = r'C:\Users\Dell\Downloads\split-garbage-dataset\train'

for filename in os.listdir(r'C:\Users\Dell\Downloads\split-garbage-dataset\train'):
    if filename.split('.')[1] == 'jpg':
        img = cv2.imread(os.path.join(train_path,filename))

        # Splitting file names and storing the labels for image in list
```

```
In [311]: x_train = []
y_train = []
shape = (32,32)
train_path = r'C:\Users\Dell\Downloads\split-garbage-dataset\train'

for filename in os.listdir(r'C:\Users\Dell\Downloads\split-garbage-dataset\train'):
    if filename.split('.')[1] == 'jpg':
        img = cv2.imread(os.path.join(train_path,filename))

        # Splitting file names and storing the labels for image in list
        y_train.append(filename.split('_')[0])

        # Resize all images to a specific shape
        img = cv2.resize(img,shape)

        x_train.append(img)

# Converting labels into One Hot encoded sparse matrix
y_train = pd.get_dummies(y_train).values

# Converting train_images to array
x_train = np.array(x_train)

#x_train,x_val,y_train,y_val = train_test_split(x_train,y_train,random_state=1)
```

```

        projection_dim,
    ] # Size of the transformer layers
    transformer_layers = 8
    mlp_head_units = [2048, 1024] # Size of the dense layers of the final classifier

```

```

In [317]: data_augmentation = keras.Sequential(
    [
        layers.experimental.preprocessing.Normalization(),
        layers.experimental.preprocessing.Resizing(image_size, image_size),
        layers.experimental.preprocessing.RandomFlip("horizontal"),
        layers.experimental.preprocessing.RandomRotation(factor=0.02),
        layers.experimental.preprocessing.RandomZoom(
            height_factor=0.2, width_factor=0.2
        ),
    ],
    name="data_augmentation",
)
# Compute the mean and the variance of the training data for normalization.
data_augmentation.layers[0].adapt(x_train)

```

```

In [318]: def mlp(x, hidden_units, dropout_rate):
    for units in hidden_units:
        x = layers.Dense(units, activation=tf.nn.gelu)(x)
        x = layers.Dropout(dropout_rate)(x)
    return x

```

```

In [319]: class Patches(layers.Layer):
    def __init__(self, patch_size):
        super(Patches, self).__init__()
        self.patch_size = patch_size

```

```

In [320]: import matplotlib.pyplot as plt

plt.figure(figsize=(4, 4))
image = x_train[np.random.choice(range(x_train.shape[0]))]
plt.imshow(image.astype("uint8"))
plt.axis("off")

resized_image = tf.image.resize(
    tf.convert_to_tensor([image]), size=(image_size, image_size)
)
patches = Patches(patch_size)(resized_image)
print(f"Image size: {image_size} X {image_size}")
print(f"Patch size: {patch_size} X {patch_size}")
print(f"Patches per image: {patches.shape[1]}")
print(f"Elements per patch: {patches.shape[-1]}")

n = int(np.sqrt(patches.shape[1]))
plt.figure(figsize=(4, 4))
for i, patch in enumerate(patches[0]):
    ax = plt.subplot(n, n, i + 1)
    patch_img = tf.reshape(patch, (patch_size, patch_size, 3))
    plt.imshow(patch_img.numpy().astype("uint8"))
    plt.axis("off")

```

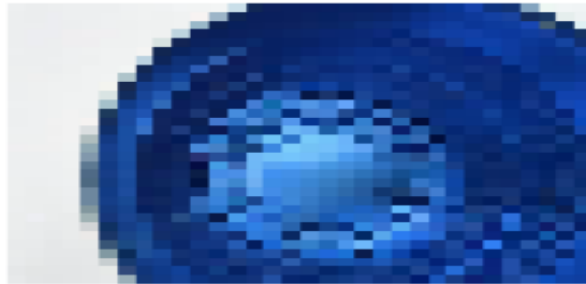
```

Image size: 72 X 72
Patch size: 6 X 6
Patches per image: 144
Elements per patch: 108

```

```
plt.axis("off")
```

```
Image size: 72 X 72  
Patch size: 6 X 6  
Patches per image: 144  
Elements per patch: 108
```



```
In [322]: def create_vit_classifier():  
    inputs = layers.Input(shape=input_shape)  
    # Augment data.  
    augmented = data_augmentation(inputs)  
    # Create patches.  
    patches = Patches(patch_size)(augmented)  
    # Encode patches.  
    encoded_patches = PatchEncoder(num_patches, projection_dim)(patches)  
  
    # Create multiple layers of the Transformer block.  
    for _ in range(transformer_layers):  
        # Layer normalization 1.  
        x1 = layers.LayerNormalization(epsilon=1e-6)(encoded_patches)  
        # Create a multi-head attention layer.  
        attention_output = layers.MultiHeadAttention(  
            num_heads=num_heads, key_dim=projection_dim, dropout=0.1  
        )(x1, x1)  
        # Skip connection 1.  
        x2 = layers.Add()([attention_output, encoded_patches])  
        # Layer normalization 2.  
        x3 = layers.LayerNormalization(epsilon=1e-6)(x2)  
        # MLP.  
        x3 = mlp(x3, hidden_units=transformer_units, dropout_rate=0.1)  
        # Skip connection 2.  
        encoded_patches = layers.Add()([x3, x2])  
  
    # Create a [batch_size, projection_dim] tensor.  
    representation = layers.LayerNormalization(epsilon=1e-6)(encoded_patches)  
    representation = layers.Flatten()(representation)  
    representation = layers.Dropout(0.5)(representation)  
    # Add MLP.  
    features = mlp(representation, hidden_units=mlp_head_units, dropout_rate=0.5)  
    # Classify outputs.  
    logits = layers.Dense(num_classes)(features)  
    # Create the Keras model.  
    model = keras.Model(inputs=inputs, outputs=logits)
```

```
7/7 [=====] - 87s 12s/step - loss: 0.1822 - accuracy: 0.9308 - top-5-accuracy: 1.0000 - val_loss: 9.77
14 - val_accuracy: 0.2712 - val_top-5-accuracy: 0.4859
Epoch 76/100
7/7 [=====] - 96s 13s/step - loss: 0.1724 - accuracy: 0.9359 - top-5-accuracy: 1.0000 - val_loss: 9.59
05 - val_accuracy: 0.2599 - val_top-5-accuracy: 0.4859
Epoch 77/100
7/7 [=====] - 92s 13s/step - loss: 0.1475 - accuracy: 0.9490 - top-5-accuracy: 1.0000 - val_loss: 9.59
90 - val_accuracy: 0.2825 - val_top-5-accuracy: 0.4859
Epoch 78/100
7/7 [=====] - 84s 12s/step - loss: 0.1491 - accuracy: 0.9438 - top-5-accuracy: 1.0000 - val_loss: 9.79
61 - val_accuracy: 0.2712 - val_top-5-accuracy: 0.4859
Epoch 79/100
7/7 [=====] - 85s 12s/step - loss: 0.1844 - accuracy: 0.9354 - top-5-accuracy: 1.0000 - val_loss: 8.64
09 - val_accuracy: 0.2768 - val_top-5-accuracy: 0.4859
Epoch 80/100
7/7 [=====] - 84s 12s/step - loss: 0.1971 - accuracy: 0.9296 - top-5-accuracy: 1.0000 - val_loss: 9.24
02 - val_accuracy: 0.2655 - val_top-5-accuracy: 0.4859
Epoch 81/100
7/7 [=====] - 87s 12s/step - loss: 0.2523 - accuracy: 0.9124 - top-5-accuracy: 1.0000 - val_loss: 7.88
92 - val_accuracy: 0.2373 - val_top-5-accuracy: 0.4859
Epoch 82/100
7/7 [=====] - 91s 13s/step - loss: 0.1855 - accuracy: 0.9438 - top-5-accuracy: 1.0000 - val_loss: 8.61
93 - val_accuracy: 0.2316 - val_top-5-accuracy: 0.4859
Epoch 83/100
7/7 [=====] - 87s 12s/step - loss: 0.1830 - accuracy: 0.9365 - top-5-accuracy: 1.0000 - val_loss: 8.93
32 - val_accuracy: 0.2655 - val_top-5-accuracy: 0.4915
Epoch 84/100
7/7 [=====] - 86s 12s/step - loss: 0.1418 - accuracy: 0.9487 - top-5-accuracy: 1.0000 - val_loss: 8.26
14 - val_accuracy: 0.2373 - val_top-5-accuracy: 0.4859
Epoch 85/100
7/7 [=====] - 90s 13s/step - loss: 0.1444 - accuracy: 0.9408 - top-5-accuracy: 1.0000 - val_loss: 8.35
34 - val_accuracy: 0.2599 - val_top-5-accuracy: 0.4859
Epoch 86/100
5/7 [=====>.....] - ETA: 27s - loss: 0.1068 - accuracy: 0.9647 - top-5-accuracy: 1.0000
```

📄 + 🔍 📄 ⬆️ ⬆️ ▶️ Run 🛑 ↺️ Code ▾ 🗨

```
7/7 [=====] - 85s 12s/step - loss: 0.1053 - accuracy: 0.9638 - top-5-accuracy: 1.0000 - val_loss: 8.87
71 - val_accuracy: 0.2768 - val_top-5-accuracy: 0.4859
Epoch 93/100
7/7 [=====] - 83s 12s/step - loss: 0.1261 - accuracy: 0.9574 - top-5-accuracy: 1.0000 - val_loss: 8.86
70 - val_accuracy: 0.2599 - val_top-5-accuracy: 0.4859
Epoch 94/100
7/7 [=====] - 83s 12s/step - loss: 0.1212 - accuracy: 0.9584 - top-5-accuracy: 1.0000 - val_loss: 7.93
11 - val_accuracy: 0.2768 - val_top-5-accuracy: 0.4859
Epoch 95/100
7/7 [=====] - 84s 12s/step - loss: 0.1658 - accuracy: 0.9452 - top-5-accuracy: 1.0000 - val_loss: 8.22
08 - val_accuracy: 0.2542 - val_top-5-accuracy: 0.4859
Epoch 96/100
7/7 [=====] - 86s 12s/step - loss: 0.1450 - accuracy: 0.9498 - top-5-accuracy: 1.0000 - val_loss: 7.74
85 - val_accuracy: 0.2599 - val_top-5-accuracy: 0.4859
Epoch 97/100
7/7 [=====] - 83s 12s/step - loss: 0.1418 - accuracy: 0.9477 - top-5-accuracy: 1.0000 - val_loss: 7.80
33 - val_accuracy: 0.2768 - val_top-5-accuracy: 0.4859
Epoch 98/100
7/7 [=====] - 84s 12s/step - loss: 0.1348 - accuracy: 0.9535 - top-5-accuracy: 1.0000 - val_loss: 8.87
34 - val_accuracy: 0.2542 - val_top-5-accuracy: 0.4859
Epoch 99/100
7/7 [=====] - 82s 12s/step - loss: 0.1286 - accuracy: 0.9567 - top-5-accuracy: 1.0000 - val_loss: 9.35
40 - val_accuracy: 0.2768 - val_top-5-accuracy: 0.4859
Epoch 100/100
7/7 [=====] - 88s 12s/step - loss: 0.1045 - accuracy: 0.9636 - top-5-accuracy: 1.0000 - val_loss: 8.67
55 - val_accuracy: 0.2938 - val_top-5-accuracy: 0.4859
14/14 [=====] - 8s 549ms/step - loss: 1.8337 - accuracy: 0.7355 - top-5-accuracy: 0.9327
Test accuracy: 73.55%
Test top 5 accuracy: 93.27%
```

```
In [366]: from keras.preprocessing import image
import numpy as np
test_img = r'C:\Users\Dell\Downloads\test\metal\metal192.jpg'
img = image.load_img(test_img, target_size = (32,32))
img = image.img_to_array(img, dtype=np.uint8)
img = np.array(img)
prediction = vit_classifier.predict(img[np.newaxis, ...])

#print("Predicted shape",p.shape)
#print("Probability:",np.max(prediction[0], axis=-1))
predicted_class = labels[np.argmax(prediction[0], axis=-1)]
print("Classified:",predicted_class,'\n')

plt.axis('off')
plt.imshow(img.squeeze())
plt.title("Loaded Image")
plt.show()
```

Classified: metal



```
[367]: from keras.preprocessing import image
import numpy as np
test_img = r'C:\Users\Dell\Downloads\test\glass\glass121.jpg'
img = image.load_img(test_img, target_size = (32,32))
img = image.img_to_array(img, dtype=np.uint8)
img = np.array(img)
prediction = vit_classifier.predict(img[np.newaxis, ...])

#print("Predicted shape",p.shape)
#print("Probability:",np.max(prediction[0], axis=-1))
predicted_class = labels[np.argmax(prediction[0], axis=-1)]
print("Classified:",predicted_class,'\n')

plt.axis('off')
plt.imshow(img.squeeze())
plt.title("Loaded Image")
plt.show()
```

Classified: glass



```
In [370]: from keras.preprocessing import image
import numpy as np
test_img = r'C:\Users\Dell\Downloads\test\paper\paper114.jpg'
img = image.load_img(test_img, target_size = (32,32))
img = image.img_to_array(img, dtype=np.uint8)
img = np.array(img)
prediction = vit_classifier.predict(img[np.newaxis, ...])

#print("Predicted shape",p.shape)
#print("Probability:",np.max(prediction[0], axis=-1))
predicted_class = labels[np.argmax(prediction[0], axis=-1)]
print("Classified:",predicted_class,'\n')

plt.axis('off')
plt.imshow(img.squeeze())
plt.title("Loaded Image")
plt.show()
```

Classified: paper



```
In [375]: from keras.preprocessing import image
import numpy as np
test_img = r'C:\Users\Dell\Downloads\test\plastic\plastic317.jpg'
img = image.load_img(test_img, target_size = (32,32))
img = image.img_to_array(img, dtype=np.uint8)
img = np.array(img)
prediction = vit_classifier.predict(img[np.newaxis, ...])

#print("Predicted shape",p.shape)
#print("Probability:",np.max(prediction[0], axis=-1))
predicted_class = labels[np.argmax(prediction[0], axis=-1)]
print("Classified:",predicted_class,'\n')

plt.axis('off')
plt.imshow(img.squeeze())
plt.title("Loaded Image")
plt.show()
```

Classified: plastic



```

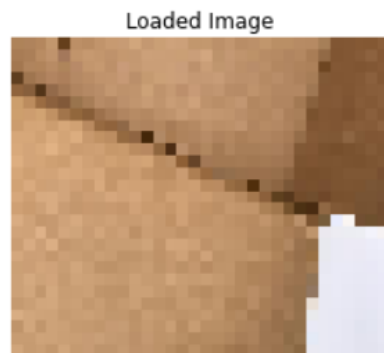
In [378]: from keras.preprocessing import image
import numpy as np
test_img = r'C:\Users\Dell\Downloads\test\cardboard\cardboard222.jpg'
img = image.load_img(test_img, target_size = (32,32))
img = image.img_to_array(img, dtype=np.uint8)
img = np.array(img)
prediction = vit_classifier.predict(img[np.newaxis, ...])

#print("Predicted shape",p.shape)
#print("Probability:",np.max(prediction[0], axis=-1))
predicted_class = labels[np.argmax(prediction[0], axis=-1)]
print("Classified:",predicted_class,'\n')

plt.axis('off')
plt.imshow(img.squeeze())
plt.title("Loaded Image")
plt.show()

```

Classified: cardboard



C.Publication with plagiarism report

