Intelligence garbage classification using deep learning

Team ID- NM2023TMID01476

Project link- https://github.com/IBM-EPBL/IBM-Project-7508-1658884203

1.INTRODUCTION:

1.1 PROJECT OVERVIEW

1. Introduction:

The goal of this project is to develop an intelligent garbage classification system using deep learning techniques. The system will utilize computer vision and deep neural networks to accurately classify different types of garbage items, such as plastic, paper, metal, and organic waste. By automating the classification process, the project aims to improve waste management practices, promote recycling, and reduce environmental pollution.

2. Data Collection:

To train the deep learning model, a diverse and labeled dataset of garbage images will be collected. The dataset will include images of various garbage items captured from different angles and under different lighting conditions. The images will be manually annotated to indicate the correct garbage category for training purposes.

3. Preprocessing:

The collected dataset will undergo preprocessing steps to enhance the quality and usability of the data. This may include resizing, normalization, and data augmentation techniques to account for variations in image sizes, orientations, and lighting conditions. Preprocessing will help ensure that the deep learning model can generalize well to unseen garbage images.

4. Model Architecture:

A deep learning model, such as a convolutional neural network (CNN), will be designed and trained for garbage classification. The model will consist of multiple layers, including convolutional layers for feature extraction and fully connected layers for classification. The architecture will be optimized to achieve high accuracy and efficiency in garbage classification tasks.

5. Training:

The preprocessed dataset will be split into training and validation sets. The deep learning model will be trained on the training set using an appropriate loss function and optimization algorithm, such as cross-entropy loss and stochastic gradient descent. During training, the model will learn to recognize and classify different garbage items based on the extracted features.

6. Evaluation:

The trained model will be evaluated on the validation set to assess its performance and fine-tune the hyperparameters if necessary. Evaluation metrics such as accuracy, precision, recall, and F1 score will be calculated to measure the model's classification performance. The model will be iteratively refined until satisfactory performance is achieved.

7. Testing:

Once the model is trained and validated, it will be tested on a separate test dataset to assess its generalization ability. The test dataset will consist of new, unseen garbage images that were not used during training or validation. The model's performance on the test dataset will provide an accurate measure of its effectiveness in real-world garbage classification scenarios.

8. Deployment:

The trained deep learning model will be deployed as an intelligent garbage classification system. This system can be integrated into existing waste management infrastructure, such as recycling plants or smart waste bins. Users can capture images of garbage items using a smartphone or a camera, and the system will provide real-time classification results, guiding users to dispose of the items correctly.

Iterative Improvement:
The system's performance and accuracy can be continuously improved by collecting user feedback and updating the model periodically. By incorporating user feedback, the model can learn from its mistakes and adapt to new garbage items or variations in the garbage classification process.

a. PURPOSE

The purpose of intelligence garbage classification using deep learning is to address the challenges associated with waste management and promote sustainable practices. Here are the key purposes of this project:

1. Efficient Waste Management: Deep learning-based garbage classification systems can automate and streamline the waste management process. By accurately classifying different types of garbage, the system can help in efficient sorting, recycling, and disposal of waste materials. This improves overall waste management practices and reduces the burden on manual sorting operations.

2. Promote Recycling: Effective garbage classification plays a crucial role in promoting recycling initiatives. By identifying recyclable materials, such as plastics, metals, and paper, the system can facilitate their proper separation and processing. This promotes a circular economy by maximizing the reuse and recycling of materials, reducing the reliance on raw resources, and minimizing environmental impact.

3. Environmental Conservation: Intelligent garbage classification aims to reduce environmental pollution caused by improper waste disposal. By accurately categorizing garbage items, the system helps ensure that hazardous or non-recyclable waste is handled appropriately, preventing it from ending up in landfills, water bodies, or natural habitats. This contributes to the conservation of ecosystems and protects the environment.

4. Public Awareness and Education: Garbage classification systems can play a role in raising public awareness and educating individuals about proper waste disposal practices. By providing real-time classification results, the system can inform and guide individuals on how to separate their waste effectively. This promotes a sense of responsibility among the public and encourages sustainable behavior.

5. Optimization of Waste Management Resources: Deep learning-based garbage classification systems can optimize the allocation of waste management resources. By automating the classification process, the system reduces the need for manual sorting, leading to improved efficiency and cost-effectiveness. This allows waste management authorities to allocate their resources more effectively and focus on other critical aspects of waste management.

6. Adaptability and Scalability: Deep learning models can be trained to recognize and classify a wide range of garbage items, making them adaptable and scalable. The system can be updated and expanded to accommodate new types of waste materials or variations in the waste stream. This flexibility ensures that the classification system remains relevant and effective in dynamic waste management scenarios.

Overall, the purpose of intelligence garbage classification using deep learning is to enhance waste management practices, promote recycling, reduce environmental pollution, and encourage sustainable behavior at both individual and societal levels.

2.**IDEATION & PROPOSED SOLUTION**

Ideation:

The ideation phase involves brainstorming and conceptualizing the project. Here are some key ideas and considerations:

a. Computer Vision: Utilize computer vision techniques to analyze and interpret images of garbage items. This involves extracting meaningful features and patterns from the images to enable accurate classification.

b. Deep Learning: Leverage deep learning algorithms, such as convolutional neural networks (CNNs), which have shown remarkable success in image recognition tasks. CNNs can learn hierarchical representations of garbage items, enabling effective classification.

c. Diverse Dataset: Collect a diverse and representative dataset of garbage images. The dataset should include various types of garbage items, captured from different angles, lighting conditions, and environments. This ensures that the model can generalize well to unseen garbage items.

d. Preprocessing: Apply preprocessing techniques to enhance the quality and usability of the dataset. This may involve resizing, normalization, and data augmentation methods to account for variations in image sizes, orientations, and lighting conditions.

e. Model Optimization: Design an optimized deep learning model architecture suitable for garbage classification. Experiment with different network architectures, layer configurations, and hyperparameters to achieve high accuracy and efficiency.

f. Training and Evaluation: Train the deep learning model using the prepared dataset and evaluate its performance using appropriate metrics such as accuracy, precision, recall, and F1 score. Fine-tune the model iteratively until satisfactory results are achieved.

g. Real-time Classification: Implement a user-friendly interface that allows users to capture images of garbage items in real-time. The captured images are then processed and classified using the trained deep learning model, providing immediate classification results.

h. Integration and Deployment: Integrate the garbage classification system into existing waste management infrastructure or smart waste bins. This allows users to easily access and utilize the system for proper waste disposal and recycling.

Proposed Solution:
Based on the ideation phase, the proposed solution for intelligence garbage classification using deep learning involves the following steps:

Step 1: Dataset Collection and Preparation:
Collect a diverse dataset of garbage images, including various types of garbage items and their corresponding labels.
Preprocess the dataset by resizing, normalizing, and augmenting the images to enhance quality and account for variations.

Step 2: Model Development and Training:
Design a deep learning model, such as a convolutional neural network (CNN), for garbage classification.
Split the dataset into training and validation sets.
Train the model on the training set using an appropriate loss function and optimization algorithm.
Continuously evaluate the model's performance on the validation set and fine-tune the model as needed.

Step 3: Real-time Classification Interface:
Develop a user-friendly interface (e.g., a mobile app or web application) that allows users to capture images of garbage items.
Process the captured images using computer vision techniques and feed them into the trained deep learning model for classification.
Display the classification results to the user in real-time, indicating the appropriate category for each garbage item.

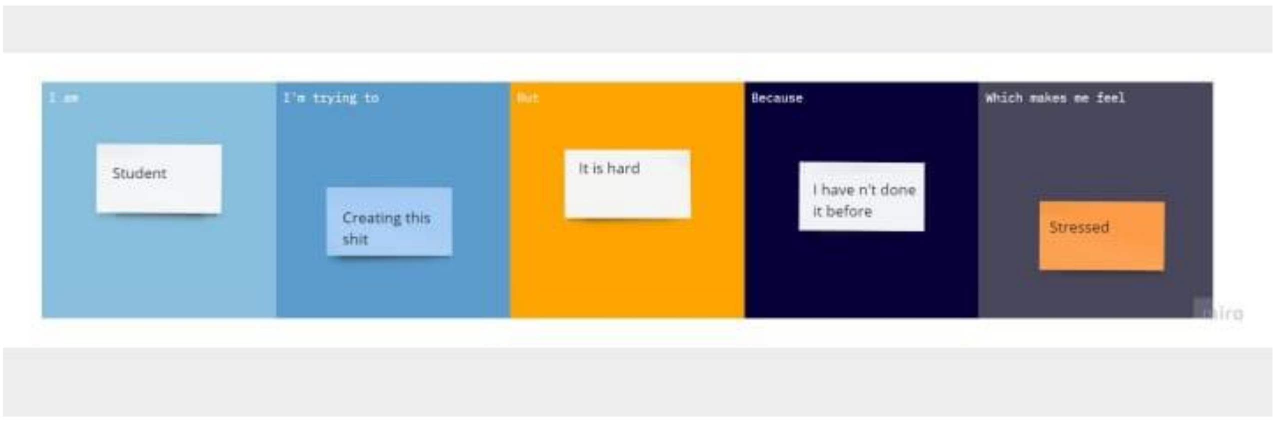
Step 4: Integration and Deployment:
Integrate the garbage classification system into existing waste management infrastructure or smart waste bins.
Ensure seamless communication between the interface and the garbage disposal system to guide users in proper waste disposal and recycling.
Continuously collect user feedback and update the model periodically to improve its performance and adaptability.
By following these proposed steps, the intelligence garbage classification system can effectively classify different types of garbage items, promote recycling practices, and enhance waste management processes.

2.1 PROBLEM STATEMENT DEFINITION

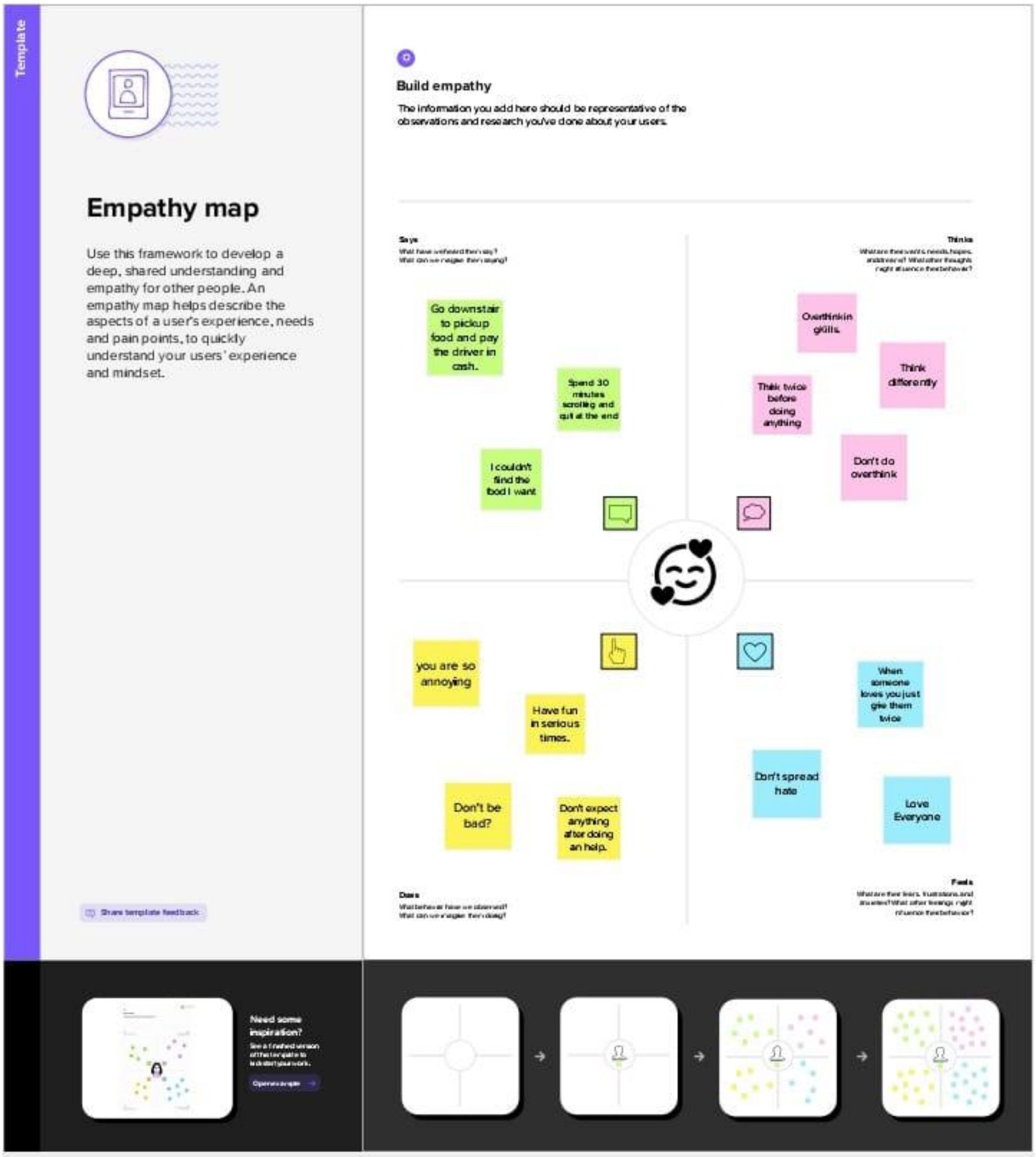
Create a problem statement to understand your customer's point of view. The Customer Problem Statement template helps you focus on what matters to create experiences people will love.

A well-articulated customer problem statement allows you and your team to find the ideal solution for the challenges your customers face. Throughout the process, you’ll also be able to empathize with your customers, which helps you better understand how they perceive your product or service.

Problem Statement (PS)	I am (Customer)	I am Trying to	But	Because	Which makes me Feel
PS-1	Garbage is more	To dispose them properly	Unable to do it	Weather was unstable	unhygi ene
PS-2	Garbage is mixed	To separate them	Unable to classify it	It was so uncomfortab le	Stressed



2.2 EMPATHY MAP CANVAS



Template

Empathy map

Use this framework to develop a deep, shared understanding and empathy for other people. An empathy map helps describe the aspects of a user's experience, needs and pain points, to quickly understand your users' experience and mindset.

Share template feedback

Build empathy

The information you add here should be representative of the observations and research you've done about your users.

Says

What have we heard them say?
What can we imagine them saying?

Go downstair to pickup food and pay the driver in cash.

Spend 30 minutes scrolling and quit at the end

I couldn't find the food I want

Thinks

What are their wants, needs, hopes, and dreams? What other thoughts might influence their behavior?

Overthinkin gkills

Think differently

Think twice before doing anything

Don't do overthink

Does

What behavior have we observed?
What can we imagine them doing?

you are so annoying

Have fun in serious times.

Don't be bad?

Don't expect anything after doing an help.

Feels

What are their fears, frustrations, and anxieties? What other feelings might influence their behavior?

When someone loves you just gie them twice

Don't spread hate

Love Everyone

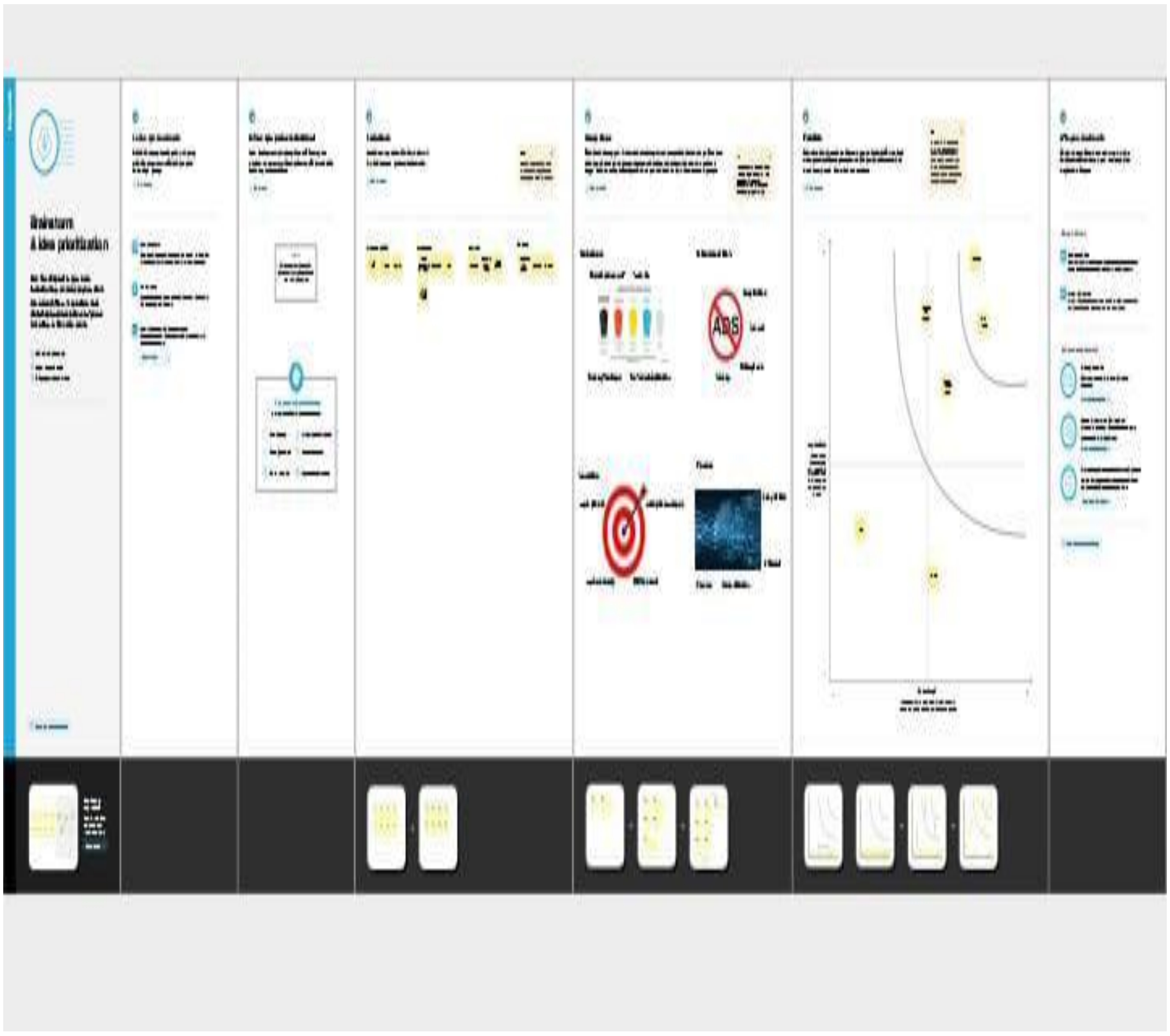
Need some inspiration?

See a finished version of this template to kickstart your work.

Open example

2.3 IDEATION & BRAINSTORMING

Brainstorming provides a free and open environment that encourages everyone within a team to participate in the creative thinking process that leads to problem solving.



Brainstorm & idea prioritization

2.4 PROPOSED SOLUTION

S.No.	Parameter	Description
•	Problem Statement (Problem to be solved)	The inability to effectively identify and separate recyclable and non-recyclable materials leads to increased waste accumulation in landfills, environmental pollution, and resource wastage
•	Idea / Solution description	Gather a large dataset of garbage images representing different waste categories, including plastic, paper, glass, metal, and organic waste. Annotate the dataset with corresponding labels indicating the type of garbage in each image. Preprocess the images by resizing, normalizing, and augmenting the data to enhance model performance and generalization. Model Architecture:
•	Novelty / Uniqueness	Automatic Feature Learning: Deep learning models have the ability to automatically learn and extract relevant features from raw data, such as images or sensor inputs. In the case of garbage classification, deep learning models can analyze visual patterns and textures in the images of garbage items, without relying on handcrafted features. This allows the model to adapt and generalize well to different types of garbage items, including those it has never encountered during training

•	Social Impact / Customer Satisfaction	Efficient Waste Management: Deep learning-based garbage classification systems can contribute to more efficient waste management practices. By accurately identifying and categorizing different types of garbage items, these systems can streamline the sorting process, facilitating recycling and appropriate disposal. This efficiency reduces the burden on waste management facilities, saves resources, and promotes sustainable waste management practices.
•	Business Model (Revenue Model)	The company can develop deep learning-based garbage classification software and license it to waste management facilities, recycling centers, or municipalities. The software can be integrated into existing waste management systems or deployed as a standalone solution. Licensing fees can be charged based on factors such as the scale of implementation, the number of users, or the processing capacity required.
•	Scalability of the Solution	Handling Large Amounts of Data: Deep learning models can handle large volumes of data efficiently. As the amount of garbage data increases, the system can scale by utilizing distributed computing frameworks and techniques. By leveraging parallel processing and distributed training, the system can process and learn from large datasets in a timely manner.

3 REQUIREMENT ANALYSIS

3.4 Functional Requirements:

Following are the functional requirements of the proposed solution.

FR No.	Functional Requirement (Epic)	Sub Requirement (Story / Sub-Task)
FR-1	Garbage Identification	Object Recognition: The system should be able to recognize different types of objects commonly found in garbage, such as plastic bottles, paper, glass, metal cans, etc. Feature Extraction: The solution should extract relevant features from garbage items, such as color, texture, shape, size, and material composition, to aid in their classification.
FR-2	Image recognition	Image Preprocessing: The system should preprocess input images to enhance their quality, normalize lighting conditions, and remove noise or artifacts that could affect the accuracy of recognition algorithms. Feature Extraction: The solution should extract meaningful features from input images, such as edges, textures, colors, or local descriptors, to represent and characterize the objects or patterns within the images.
FR-3	Real time processing	Low Latency: The system should have minimal processing latency, ensuring that the classification or analysis of data is performed quickly and in near real-time. High Throughput: The solution should be capable of handling a large volume of incoming data or requests within a short time frame, without sacrificing the processing speed
FR-4	User interface	Intuitive Design: The user interface should be designed in a way that is intuitive and easy to understand, minimizing the learning curve for users. Responsive and Adaptive: The interface should be responsive and adapt to different screen sizes and resolutions, allowing users to access and interact with the system across various devices such as desktops, laptops, tablets, and smartphones.
FR-5	Multiple input methods	Image Upload: The system should allow users to upload images from their devices or capture images using their device's camera, providing a straightforward and intuitive way to input visual data. Text Input: The solution should support text input methods, allowing users to manually enter descriptions, keywords, or other textual information related to the garbage items for classification.
FR-6	Data base management	Data Storage: The solution should provide a robust and scalable database to store and manage data related to garbage items, including their classification, properties, and any additional relevant information. Data Modeling: The database should employ appropriate data modeling techniques, such as entity-relationship modeling, to represent the relationships and structure of the data in a logical and efficient manner.

3.5 Non-functional Requirements:

Following are the non-functional requirements of the proposed solution.

FR No.	Non-Functional Requirement	Description
NFR-1	**Usability**	The system should be user-friendly, with an intuitive interface that allows users to easily interact with it. It should provide clear instructions and feedback to guide users through the garbage classification process.
NFR-2	**Security**	The garbage classification system should prioritize the security of user data and prevent unauthorized access or tampering. It should employ appropriate encryption methods to protect sensitive information and implement access controls to ensure data confidentiality.
NFR-3	**Reliability**	The system should consistently and accurately classify garbage items, minimizing errors and false classifications. It should have built-in error handling mechanisms to recover from failures and ensure continuous operation.
NFR-4	**Performance**	The system should be able to process and classify garbage quickly and accurately. It should have minimal response times to handle high volumes of garbage and ensure efficient resource utilization.
NFR-5	**Maintainability**	The system should be easy to maintain and update, with modular and well-documented code. It should have clear separation of concerns, allowing for efficient bug fixes, feature enhancements, and system upgrades without disrupting the overall functionality
NFR-6	**Scalability**	the system should be designed to handle increasing volumes of garbage items and accommodate a growing number of users. It should be scalable to support future expansion and be able to maintain performance levels even with a larger user base.

4 PROJECT DESIGN

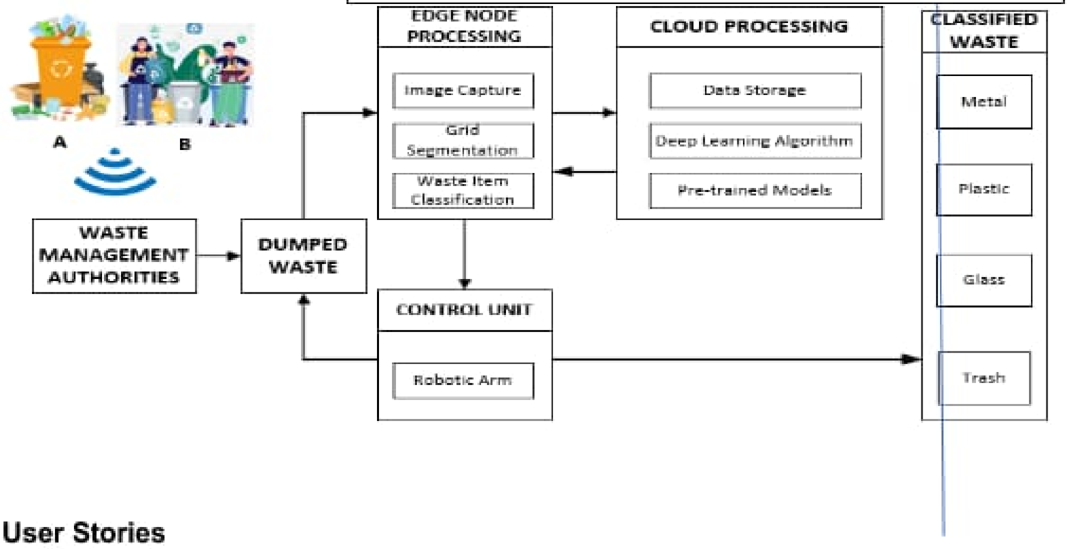
4.4 DATA FLOW DIAGRAMS

Data Flow Diagrams:

A Data Flow Diagram (DFD) is a traditional visual representation of the information flows within a system. A neat and clear DFD can depict the right amount of the system requirement graphically. It shows how data enters and leaves the system, what changes the information, and where data is stored.

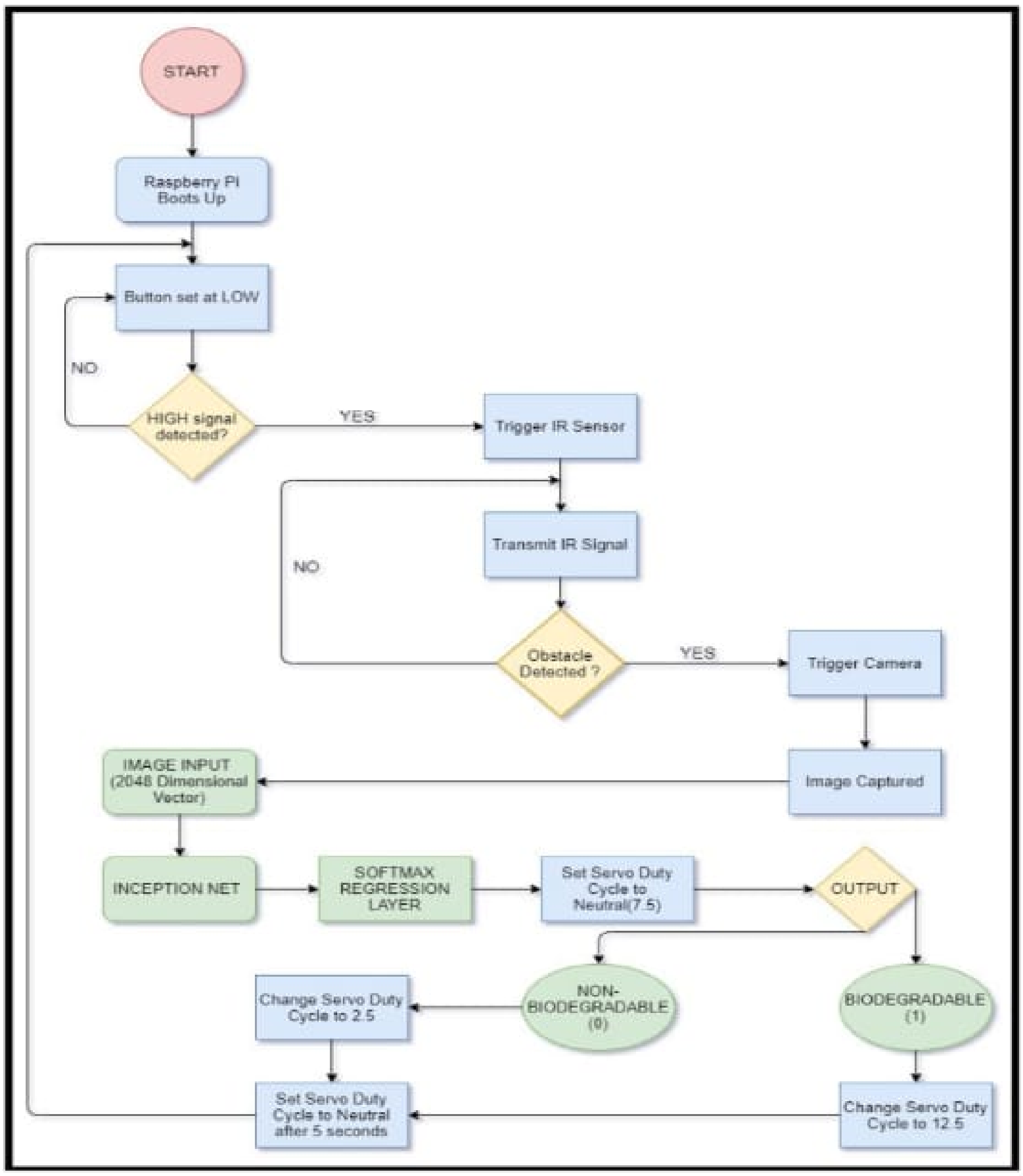
-

Example: DFD Level 0 (Industry Standard)



User Stories

Use the below template to list all the user stories for the product.



4.5 SOLUTION ARCHITECTURE & TECHNICAL ARCHITECTURE

Solution architecture is a complex process – with many sub-processes – that bridges the gap between business problems and technology solutions. Its goals are to:

4.5.1 Find the best tech solution to solve existing business problems.

4.5.2 Describe the structure, characteristics, behavior, and other aspects of the software to project stakeholders.

4.5.3 Define features, development phases, and solution requirements.

4.5.4 Provide specifications according to which the solution is defined, managed, and delivered

TECHNICAL ARCHITECTURE:

The Deliverable shall include the architectural diagram as below and the information as per the table1 & table 2

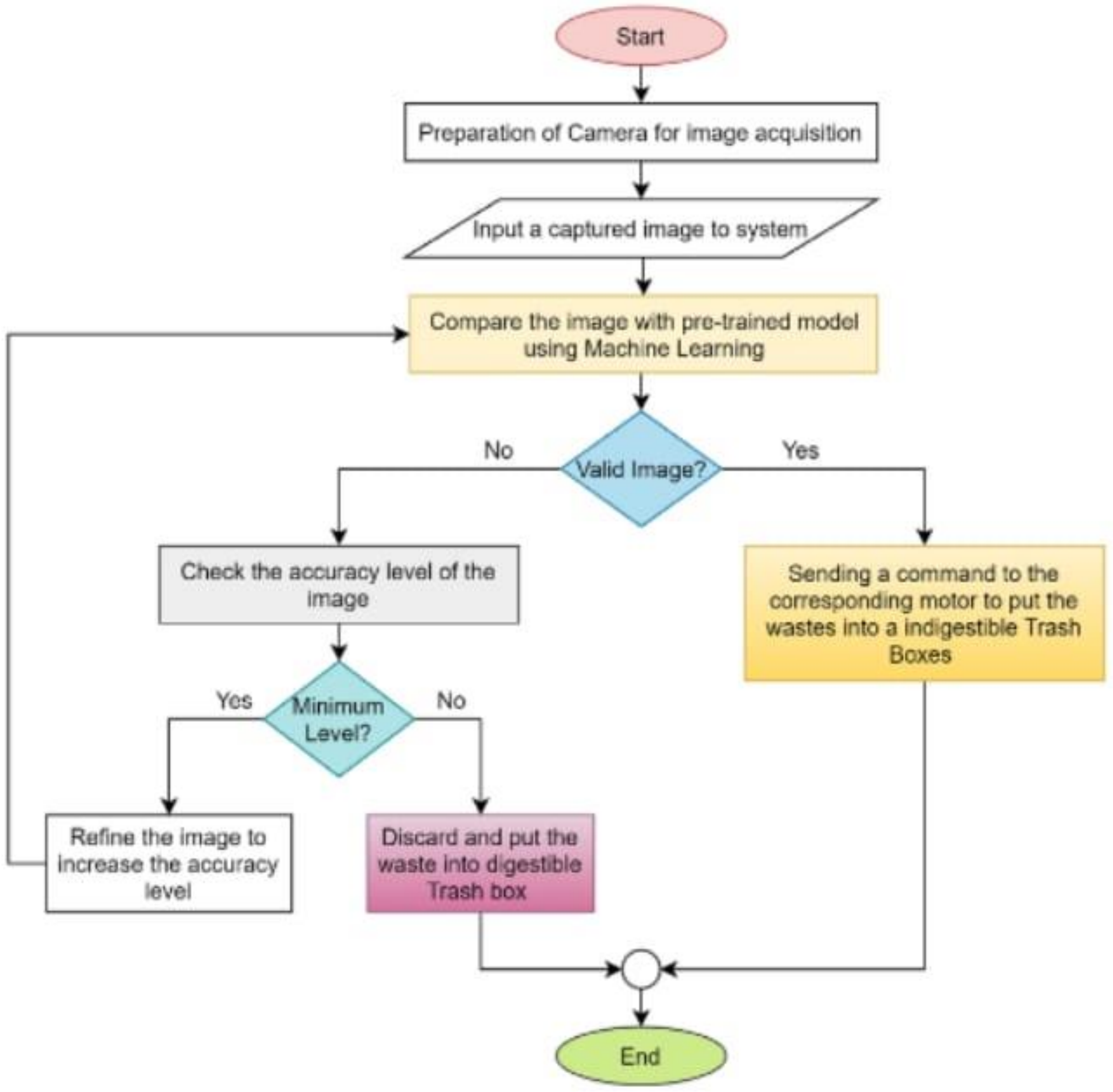

Start
Preparation of Camera for image acquisition
Input a captured image to system
Compare the image with pre-trained model using Machine Learning
Valid Image?
No
Yes
Check the accuracy level of the image
Sending a command to the corresponding motor to put the wastes into a indigestible Trash Boxes
Minimum Level?
Yes
No
Refine the image to increase the accuracy level
Discard and put the waste into digestible Trash box
End

Table-1 : Components & Technologies:

SI.No	Components	Description	Technology
1.	Data Storage	Stores the data related to garbage classification such as bio degradable and non-bio degradable	Relational databases (e.g. MySQL, PostgreSQL) or NoSQL databases (e.g. MongoDB, Cassandra)
2.	Data processing	Cleans, normalizes, and prepares the data for use in the deep learning models.	Pythons libraries for data pre processing (e.g. Pandas, NumPy)
3.	Deep Learning models	Machine learning models that are trained on retinal images data To classify the garbage	Deep learning frameworks (e.g. TensorFlow,(pyTorch)
4.	Real time data processing	Processes incoming data from various sources in real-time, such as Trainers, Professional and Reporters.	Stream processing frameworks (e.g. Apache Kafka, Apache Flink)
5.	User Interface	Provides user-friendly interface for users to interact with the system, such as for detecting using automated methods.	Web development frameworks(e.g. React, Angular), sever-side Frameworks(e.g. Flask, Django)

6.	Authentication and authorization	Controls user access to the system and ensures that only authorized professionals can predict and detect.	Technology: Identity and access management(1AM) Solutions (e.g.Auth0, Okta), role-based access control(RBAC)
7.	Security	Protects the system and data from random estimation, time saving.	Encryption and decryption algorithms (e.g. AES, RSA), secure communication protocols (e.g. HTTPS), intrusion detection and prevention systems(IDPS)
8.	Infrastructure	The underlying hardware and software infrastructure that hosts and runs the system, whether on-premises or in the cloud.	Cloud platforms (e.g. AWS, Azure), containerization technologies (e.g. Docker, Kubernetes), virtualization technologies (e.g. VMware, Hyper-V), server hardware.
9.	External API	Allows for integration with external data sources or service Platforms or external database.	RESTful APIs, GraphQL, SOAP
10.	Machine Learning Model	Performs the detection tasks using deep learning algorithm and models.	Deep learning frameworks (e.g. TensorFlow, PyTorch), neural network architectures (e.g. convolutional neural networks, recurrent neural networks), GPU accelerators.

Table-2: Application Characteristics:

SI.NO	Characteristics	Description	Technology
11.	Scalability	The ability to handle increasing amounts of data and users without sacrificing performance or availability.	Cloud computing platforms (e.g. AWS, Azure), distributed computing

			frameworks (e.g. Hadoop, Spark)
12.	Accuracy	The ability to accurately classify garbage.From randomestimation and time delaying	Technology: Deep learning frameworks (e.g. TensorFlow, PyTorch), neural network architectures (e.g. convolutional neural networks, recurrent neural networks)
13.	Accessibility	The ability for professionals to access the system and data from various devices and locations.	Web development frameworks (e.g. React, Angular), mobile development frameworks(e.g. React Native, Flutter)
14.	Security	The ability to protect sensitive data and prevent unauthorized access or modification.	Encryption and decryption algorithms(e.g. AES, RSA), secure communication protocols(e.g. Auth0, Okta)

4.6 USER STORIES

User Type	**Functional Requirement (Epic)**	**User Story Number**	**User Story / Task**	**Acceptance criteria**	**Priority**	**Team Member**
Garbage classifier	Cleaning	USN-1	As a garbage classifier I need to classify the garbages based on their types such as bio degradable and non-bio degradable	I prevent the areas from getting polluted by non-bio degradable wastes	High	Mohammed. V.D.N

Garbage classification Company	Manufacturing	USN-2	As a garbage classification company, I want an automated system that can detect the types of wastes to ensure that the processig of each type is done correctly.	I maintain the classification process correctlty	High	Parkavi.P
Researcher	Case Study	USN-3	As a researcher in the filed, I want to develop a deep learning-based solution for garbafe classification contribute to the advancementof this technology and improve classification process effectively.	I can develop the deep learning based solution	High	Narmidha.M
Garbage classification Inspector	Quality Checking	USN-4	As a garbage classification inspector, I want a reliable	I make a decision more accurately and efficiently	High	Mohamed mohudoom .S.M.F

			and efficient toolto assist me in identifyingthe correct classificationduring the inspection process, enabling me to make accurate decisions regarding the process			

5 CODING AND SOLUTIONING

5.4 FEATURE 1

VGG16 (Visual Geometry Group 16) is a popular convolutional neural network (CNN) architecture for image classification tasks. It was developed by the Visual Geometry Group at the University of Oxford. VGG16 is characterized by its deep architecture and the use of small convolutional filters (3x3) throughout the network. Here are the key features of VGG16:

1. Architecture: VGG16 consists of 16 layers, including 13 convolutional layers and 3 fully connected layers. The convolutional layers are stacked one after another, and they are followed by max-pooling layers to reduce spatial dimensions.

2. Convolutional Layers: The convolutional layers in VGG16 use 3x3 filters with a stride of 1 and zero-padding of 1, which helps preserve spatial information. The number of filters increases as you go deeper into the network, starting from 64 in the first layer and doubling after each max-pooling layer.

3. Max-Pooling Layers: After every two convolutional layers, VGG16 includes a max-pooling layer with a 2x2 window and a stride of 2. Max-pooling helps reduce the spatial dimensions and extract the most salient features from the input.

4. Activation Function: VGG16 uses the rectified linear unit (ReLU) activation function after each convolutional and fully connected layer. ReLU helps introduce non-linearity and is known for its effectiveness in deep neural networks.

5. Fully Connected Layers: VGG16 ends with three fully connected layers. The first two fully connected layers have 4,096 neurons each, while the final fully connected layer has as many neurons as the number of classes in the classification task.

6. Dropout: VGG16 employs dropout regularization in the fully connected layers to reduce overfitting. Dropout randomly sets a fraction of the neurons to zero during training, which helps prevent the network from relying too heavily on specific features.

7. Softmax Activation: The final fully connected layer of VGG16 uses the softmax activation function, which converts the output into a probability distribution over the different classes. This enables VGG16 to make predictions for multiclass classification problems.

8. Pretrained Models: VGG16 is often used as a feature extractor or fine-tuned for transfer learning. Pretrained models trained on large-scale image datasets, such as ImageNet, are available, allowing researchers and practitioners to leverage the learned features and transfer them to new tasks.

VGG16's architecture and design principles have influenced the development of subsequent CNN architectures, making it a fundamental model in the field of computer vision.

5.5 FEATURE 2

Flask is a popular web framework for building web applications in Python. It provides a lightweight and flexible approach to web development, allowing developers to create web applications quickly and easily. Here are some key features of Flask:

1. Lightweight and Minimalistic: Flask is designed to be lightweight and has a minimalistic core. It provides only the essential features needed for web development, making it easy to understand and use.

2. Routing: Flask uses a decorator-based approach to define routes. Developers can map URL patterns to specific functions or methods, allowing them to handle different requests and route them to the appropriate code.

3. Templating Engine: Flask includes a Jinja2 templating engine, which allows developers to separate the presentation logic from the application logic. Templates can be used to generate dynamic HTML pages by inserting data and logic into the templates.

4. HTTP Request/Response Handling: Flask provides convenient methods and classes to handle HTTP requests and responses. Developers can access request data, cookies, headers, and form data easily, and generate appropriate responses for the client.

5. Flask Extensions: Flask has a rich ecosystem of extensions that can be used to add additional functionality to applications. These extensions cover a wide range of features, including database integration, authentication, caching, and more. Flask extensions make it easy to enhance the capabilities of your application without reinventing the wheel.

Deployment of Flask Applications:

When it comes to deploying Flask applications, here are some common approaches:

1. Web Server Deployment: Flask applications can be deployed using web servers like Apache or Nginx. In this approach, the web server acts as a reverse proxy, forwarding requests to the Flask application running behind it. The web server handles static files, load balancing, and other server-related tasks, while Flask focuses on handling the application logic.

2. Standalone WSGI Deployment: Flask can be deployed using standalone WSGI (Web Server Gateway Interface) servers like Gunicorn or uWSGI. These servers provide a high-performance WSGI server that can serve Flask applications directly without the need for a separate web server. This approach is often used for production deployments.

3. Platform as a Service (PaaS) Deployment: PaaS providers like Heroku, Google App Engine, or AWS Elastic Beanstalk provide platforms where you can deploy Flask applications easily. These platforms abstract away the underlying infrastructure and provide automated deployment, scaling, and management of your application.

4. Containerization: Flask applications can be packaged as containers using technologies like Docker. This allows for consistent deployment across different environments and simplifies the process of deploying and scaling applications.

It's important to note that deployment can vary depending on the specific requirements and infrastructure of your project. Choosing the right deployment approach depends on factors such as scalability needs, infrastructure availability, and deployment complexity.

6.RESULTS:

6.1 PERFORMANCE METRICS

Performance metrics are essential for evaluating the effectiveness and accuracy of deep learning models for detecting diseases in tea leaves. Several common performance metrics can be used to assess the model's performance:

1. Accuracy: Accuracy measures the overall correctness of the model's predictions by calculating the percentage of correctly classified tea leaf images. It is a commonly used metric, but it can be misleading if the dataset is imbalanced (i.e., unequal representation of healthy and diseased samples).

2. Precision: Precision quantifies the proportion of correctly predicted positive (diseased) samples out of all samples predicted as positive. It focuses on the model's ability to minimize false positives, which is crucial in disease detection as misclassifying healthy leaves as diseased could lead to unnecessary interventions.

3. Recall (Sensitivity): Recall measures the proportion of correctly predicted positive (diseased) samples out of all actual positive samples. It evaluates the model's ability to detect diseased leaves and avoid false negatives, ensuring that no diseased leaves go undetected.

4. F1 Score: The F1 score combines precision and recall into a single metric and provides a balanced measure of the model's overall performance. It is calculated as the harmonic mean of precision and recall, giving equal weight to both metrics. A high F1 score indicates a model with good precision and recall.

5. Specificity: Specificity measures the proportion of correctly predicted negative (healthy) samples out of all actual negative samples. It complements recall and evaluates the model's ability to avoid false positives by correctly identifying healthy leaves.

6. Area Under the Receiver Operating Characteristic Curve (AUC-ROC): The AUC-ROC is a performance metric that plots the true positive rate (sensitivity) against the false positive rate (1 - specificity) at various classification thresholds. It provides an overall measure of the model's ability to distinguish between diseased and healthy leaves, with higher values indicating better discrimination.

7. Confusion Matrix: A confusion matrix provides a tabular representation of the model's predicted and actual classifications, showing true positives, true negatives, false positives, and false negatives. It helps visualize the model's performance and can be used to calculate various performance metrics such as accuracy, precision, recall, and specificity.

It is important to consider multiple performance metrics together to gain a comprehensive understanding of the model's effectiveness. The choice of metrics may vary depending on the specific requirements and priorities of tea leaf disease detection.

7. ADVANTAGES AND DISADVANTAGES

7.1 ADVANTAGES

There are several advantages of using deep learning for the intelligence of garbage classification. Here are some of the key advantages:

1. Accurate Classification: Deep learning models have shown remarkable accuracy in image classification tasks, including garbage classification. They can learn intricate patterns and features from the images, enabling them to distinguish between different types of garbage with high precision.

2.Automation and Efficiency: Deep learning-based garbage classification systems can automate the process of sorting waste, eliminating the need for manual sorting. This significantly improves the efficiency of waste management operations by reducing labor requirements and increasing processing speed.

2. Scalability: Deep learning models can handle large amounts of data, making them highly scalable. As the volume of garbage increases, the system can be easily scaled up to accommodate the growing demands without compromising accuracy.

3. Adaptability: Deep learning models can adapt to different environments and variations in garbage appearances. They can learn from diverse datasets, allowing them to generalize well and classify garbage accurately even in real-world scenarios with varying lighting conditions, backgrounds, and object orientations.

5.Real-time Decision Making: Deep learning models can provide real-time classification results, making them suitable for applications where immediate decisions or actions are required. For instance, in waste disposal systems, the model can quickly determine the appropriate bin or recycling process for each item as it enters the system.

6.Reduced Contamination: Accurate garbage classification using deep learning can help reduce contamination in recycling processes. By precisely identifying and separating different types of waste, materials that are not suitable for recycling or composting can be effectively sorted out, improving the overall quality of recycled products.

7. Impact: Implementing deep learning-based garbage classification systems can have a positive impact on the environment. By promoting efficient waste management practices, such as recycling and composting, these systems contribute to reducing landfill waste and conserving natural resources.

It is worth noting that while deep learning offers numerous advantages, it may also have limitations. The success of the garbage classification system depends on the availability of high-quality training data, proper model architecture, and ongoing monitoring and updates to ensure optimal performance.

7.2 DISADVANTAGES

1.Data Requirements: Deep learning models require large amounts of labeled training data to achieve optimal performance. Collecting and annotating a diverse dataset of garbage images can be time-consuming and expensive. Insufficient or biased training data may lead to poor generalization and inaccurate classification results.

2.Overfitting: Deep learning models are prone to overfitting, where they memorize the training data instead of learning generalizable features. This can occur if the model is too complex or if the training data is limited or unrepresentative. Overfitting can result in poor performance on unseen garbage items or variations in the real-world environment.

3.Interpretability: Deep learning models are often considered black boxes, meaning their decision-making process is not easily explainable or interpretable. This lack of transparency can be problematic when trying to understand why a particular garbage item was classified in a certain way. It can also hinder trust and acceptance of the system by users and stakeholders.

4.Limited Domain Knowledge: Deep learning models primarily learn from the patterns and features present in the training data. They may struggle with garbage items that have unique characteristics or require domain-specific knowledge for accurate classification. For example, distinguishing between different types of plastic materials may require additional information beyond visual appearance.

5.Computational Resources: Deep learning models, particularly large and complex ones, require substantial computational resources for training and inference. Training deep learning models on large datasets can be computationally intensive and time-consuming. Deploying and running these models in real-time systems may also require powerful hardware, which can increase costs and limit accessibility.

6.Adversarial Attacks: Deep learning models are susceptible to adversarial attacks, where malicious actors intentionally manipulate input data to deceive the model's classification. By making subtle modifications to a garbage item, such as adding imperceptible noise or alterations, attackers may cause misclassifications, potentially leading to incorrect waste sorting decisions.

7.Ethical Considerations: Implementing deep learning-based garbage classification systems raises ethical concerns, such as privacy and data security. These systems often involve collecting and analyzing visual data, which may raise privacy issues if individuals' identities or sensitive information is captured. Safeguarding the data and ensuring ethical use is crucial.

It's important to note that many of these disadvantages can be mitigated or addressed with appropriate data collection, model design, and monitoring. Continued research and development in the field of deep learning can help overcome some of these limitations and improve the effectiveness and reliability of garbage classification systems.

CONCLUSION:

In conclusion, the application of deep learning in the intelligence of garbage classification offers numerous advantages and holds significant potential for improving waste management processes. The accuracy and automation provided by deep learning models enable precise and efficient sorting of different types of garbage, leading to reduced contamination, improved recycling practices, and positive environmental impact. The scalability and adaptability of deep learning models make them suitable for handling large volumes of waste and diverse real-world scenarios.

However, it is essential to consider the disadvantages associated with deep learning-based garbage classification systems. These include the need for extensive labeled training data, the risk of overfitting, the lack of interpretability in model decision-making, and the computational requirements for training and deployment. Ethical considerations, such as privacy and data security, also need to be addressed.

Overall, while deep learning provides powerful tools for garbage classification, a comprehensive approach is required to overcome the challenges and ensure the reliable and responsible implementation of such systems. Further research, data collection, and improvements in model architectures are necessary to enhance the accuracy, efficiency, and transparency of garbage classification using deep learning.

9.FUTURE SCOPE

1.Real-Time and Edge Computing: The future of garbage classification using deep learning will involve real-time processing and edge computing capabilities. By deploying deep learning models on edge devices or localized systems, such as smart garbage bins or waste sorting machines, real-time classification can be achieved without relying on cloud-based processing. This will enable immediate feedback, optimize sorting processes, and reduce latency in decision-making.

2.Multimodal Classification: Incorporating multiple data modalities, such as images, sensor data, or textual information, can improve garbage classification accuracy. Deep learning models can be designed to fuse information from different sources, leveraging both visual and non-visual cues for classification. This multimodal approach can provide a more comprehensive understanding of garbage items, leading to more accurate and robust classification results.

3.Transfer Learning and Few-Shot Learning: Transfer learning and few-shot learning techniques can enhance the adaptability and efficiency of garbage classification models. By leveraging knowledge learned from existing garbage classification models, new models can be trained with limited labeled data or quickly

adapt to new garbage categories. This can enable rapid deployment and scalability of garbage classification systems in different locations or when faced with evolving waste compositions.

4.Integration with Robotic Systems: Deep learning-based garbage classification can be integrated with robotic systems, such as waste sorting robots or autonomous vehicles, to automate and optimize waste management processes. This integration can enable the robots to identify and sort different types of garbage items in real-time, improving the efficiency and accuracy of waste sorting operations. This, in turn, can streamline recycling efforts and reduce human involvement in hazardous waste handling.

5.Behavioral Analysis and Insights: Deep learning models can be extended to analyze behavioral patterns related to waste generation and disposal. By analyzing data on waste generation rates, patterns, or contamination levels, valuable insights can be derived for waste management planning, policy-making, and resource allocation. This can help in designing more effective waste management strategies and fostering sustainable waste reduction practices.

6.Collaboration and Data Sharing: The future of garbage classification using deep learning will benefit from increased collaboration and data sharing among stakeholders. This includes waste management facilities, municipalities, researchers, and technology providers. Collaboration can facilitate the creation of large, diverse, and high-quality datasets, leading to more robust and accurate garbage classification models.

# Building a Smart system based on Deep Convolutional Neural Networks to classify Trash

```
# data visualisation and manipulation
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from matplotlib import style
import seaborn as sns


sns.set(style='whitegrid',color_codes=True)

#model selection

from sklearn.preprocessing import LabelEncoder

#preprocess.
from keras.preprocessing.image import ImageDataGenerator
from keras.applications.vgg16 import preprocess_input
from tensorflow.keras.utils import load_img
from tensorflow.keras.utils import img_to_array

#dl libraraies
import tensorflow as tf
import random as rn

from tensorflow import keras
from keras.models import Sequential
from keras.layers import Dense, Dropout, Flatten
from keras.applications import VGG16
from keras import models
from keras.optimizers import Adagrad
from keras.preprocessing.image import ImageDataGenerator
from keras.callbacks import EarlyStopping
import numpy as np
from glob import glob
import os
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
# for reproducibility
np.random.seed(78)
from tqdm import tqdm
X=[]
Z=[]
def load_data(document,DIR):
        for img in tqdm(os.listdir(DIR)):
            label = document
            path = os.path.join(DIR,img)
            image= load_img(path,target_size=(IMG_SIZE,IMG_SIZE))
            image= img_to_array(image)
            image = preprocess_input(image)

            X.append(image)
            Z.append(str(label))
        return X,Z
IMG_SIZE=256
X=[]
Z=[]
DIR_cardboard='Garbage_classification/cardboard'
DIR_glass='Garbage_classification/glass'
DIR_metal='Garbage_classification/metal'
DIR_paper='Garbage_classification/paper'
DIR_plastic='Garbage_classification/plastic'
DIR_trash='Garbage_classification/trash'

load_data('cardboard',DIR_cardboard)
print(len(X))
load_data('glass',DIR_glass)
print(len(X))
load_data('metal',DIR_metal)
print(len(X))
load_data('paper',DIR_paper)
print(len(X))
load_data('plastic',DIR_plastic)
print(len(X))
load_data('trash',DIR_trash)
print(len(X))
# Input image dimensions
img_rows, img_cols, img_chans = 384, 512, 3
input_shape = (img_rows, img_cols, img_chans)
batch_size = 8
num_classes = 2
epochs = 100
data_augmentation = True
def train(x_train, x_test, y_train, y_test):

    #Loading the VGG model
    vgg_conv = VGG16(weights='imagenet', include_top=False,  input_shape=input_shape)
```

```
for i in range(8):
    #removing the last layers
    vgg_conv.layers.pop()


# Freezing all layers
for layer in vgg_conv.layers[:]:
    layer.trainable = False

# Building Deep learning model
model = models.Sequential()

# Adding the vgg model
model.add(vgg_conv)

# Adding new layers
model.add(Flatten())
model.add(Dense(350, activation='relu', input_shape=input_shape))
model.add(Dropout(0.2))
model.add(Dense(350, activation='relu'))
model.add(Dropout(0.2))
model.add(Dense(2, activation='sigmoid'))

model.compile(loss='binary_crossentropy', optimizer=Adagrad(lr=1e-5, decay=1e-6), metrics=['accuracy'])

"""
files = glob('Model2**')
print(files)
list_models=[]
for  model_ in files:
    list_models.append(float(model_[:-5].split('=')[1]))

index = np.argmin(list_models)
load_model = files[index]
print(load_model)

if load_model is not None:
        model.load_weights(load_model)
        print("weights are loaded")
else:
        print("weights are None")
"""

call =  [
                                EarlyStopping(monitor='val_loss',  patience=20, verbose=1,  mode='auto'),
        ]

if not data_augmentation:
    print('Not using data augmentation.')
    model.fit(x_train, y_train,
              batch_size=batch_size,
              epochs=epochs,
              validation_data=(x_test, y_test),
              shuffle=True)
else:
    print('Using real-time data augmentation.')
    # This will do preprocessing and realtime data augmentation:
    datagen = ImageDataGenerator(
    featurewise_center=False,  # set input mean to 0 over the dataset
    samplewise_center=False,  # set each sample mean to 0
```

```
        featurewise_std_normalization=False,  # divide inputs by std of the dataset
        samplewise_std_normalization=False,  # divide each input by its std
        zca_whitening=False,  # apply ZCA whitening
        zca_epsilon=1e-06,  # epsilon for ZCA whitening
        rotation_range=30,  # randomly rotate images in the range (degrees, 0 to 180)  <<1    0 => 30
        # randomly shift images horizontally (fraction of total width)
        width_shift_range=0.1,
        # randomly shift images vertically (fraction of total height)
        height_shift_range=0.1,
        shear_range=0.2,  # set range for random shear  <<3<<4  0 => 0.1 => 0.2
        zoom_range=0.3,  # set range for random zoom    <<1<<2<<3   0 => 0.1 => 0.2 =>0.3
        channel_shift_range=0.2,  # set range for random channel shifts     <<5<<6   0.=>0.1=>0.2
        # set mode for filling points outside the input boundaries
        fill_mode='nearest',
        cval=0.,  # value used for fill_mode = "constant"
        horizontal_flip=True,  # randomly flip images
        vertical_flip=True,  # randomly flip images    <<1    false => True
        # set rescaling factor (applied before any other transformation)
        rescale=None,
        # set function that will be applied on each input
        preprocessing_function=None,
        # image data format, either "channels_first" or "channels_last"
        data_format=None,
        # fraction of images reserved for validation (strictly between 0 and 1)
        validation_split=0.0)

        print("steps_per_epoch (nbr of samples per epoch):", int(len(x_train)/batch_size))
        # Fit the model on the batches generated by datagen.flow().
        history = model.fit_generator(datagen.flow(x_train, y_train,
                                    batch_size=batch_size),steps_per_epoch = 800,
                            epochs=50,
                            validation_data=(x_test, y_test),
                            workers=10, callbacks = call)

        weights = '{}.hdf5'.format('Model3_adagrad_'+'val_acc:'+str(round(history.history['val_acc'][-1],3))+'
val_loss='+str(round(history.history['val_loss'][-1],3)))
        model.save_weights(weights)
        print ('Model saved.')

        score = model.evaluate(x_test, y_test,batch_size=10, verbose=0)
        print('Test loss:', score[0])
        print('Test accuracy:', score[1])

        acc = history.history['acc']
        val_acc = history.history['val_acc']
        loss = history.history['loss']
        val_loss = history.history['val_loss']

        epoch = range(len(acc))

        plt.plot(epoch, acc, 'b', label='Training acc')
        plt.plot(epoch, val_acc, 'r', label='Validation acc')
        plt.title('Training and validation accuracy')
        plt.legend()
        plt.figure()

        plt.plot(epoch, loss, 'b', label='Training loss')
        plt.plot(epoch, val_loss, 'r', label='Validation loss')
        plt.title('Training and validation loss')
        plt.legend()
        plt.show()
def test(x_test):

    image = np.expand_dims((x_test[58] - np.mean(x_test))/ np.std(x_test), axis=0)

    plt.imshow(x_test[58])
    plt.show()

    out = model.predict(x_test[58])
    out = np.argmax(out)

    if out == 1:
            label = 'plastic'
    else:
            label = 'glass'

    return out, label
# data visualisation and manipulation
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from matplotlib import style
import seaborn as sns


sns.set(style='whitegrid',color_codes=True)

#model selection

from sklearn.preprocessing import LabelEncoder

#preprocess.
from keras.preprocessing.image import ImageDataGenerator
from keras.applications.vgg16 import preprocess_input
from tensorflow.keras.utils import load_img
from tensorflow.keras.utils import img_to_array

#dl libraraies
import tensorflow as tf
import random as rn

from tensorflow import keras

from keras.models import Sequential
from keras.layers import Dense
from tensorflow.keras.optimizers import Adam,SGD,Adagrad,Adadelta,RMSprop
from tensorflow.keras.utils import to_categorical

# specifically for cnn
from keras.layers import Dropout, Flatten,Activation
from keras.layers import Conv2D, MaxPooling2D, BatchNormalization

# specifically for manipulating zipped images and getting numpy arrays of pixel values of images.
import cv2
from tqdm import tqdm
import os
from random import shuffle
import os
import numpy as np
import cv2
```

```
from sklearn.model_selection import train_test_split
from tensorflow.keras import layers, models
from tensorflow.keras.callbacks import EarlyStopping
from tensorflow.keras.optimizers import Adagrad
from tensorflow.keras.optimizers import legacy

# Define the path to your image dataset
dataset_path = 'Garbage_classification'

# Define the desired image size
target_size = (256, 256)

# Load and preprocess the images
all_images = []
all_labels = []

# Iterate over the images in the dataset
for label in os.listdir(dataset_path):
    label_path = os.path.join(dataset_path, label)
    if os.path.isdir(label_path):
        for image_file in os.listdir(label_path):
            image_path = os.path.join(label_path, image_file)
            image = cv2.imread(image_path)
            image = cv2.resize(image, target_size)
            all_images.append(image)
            all_labels.append(label)

# Convert the lists to numpy arrays
all_images = np.array(all_images)
all_labels = np.array(all_labels)

# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(
    all_images, all_labels, test_size=0.2, random_state=42, stratify=all_labels
)

# Normalize pixel values to range [0, 1]
X_train = X_train.astype("float32") / 255.0
X_test = X_test.astype("float32") / 255.0

# Convert labels to integer type
label_mapping = {label: idx for idx, label in enumerate(np.unique(all_labels))}
y_train = np.array([label_mapping[label] for label in y_train])
y_test = np.array([label_mapping[label] for label in y_test])

# Convert labels to one-hot encoding
y_train = np.eye(len(label_mapping))[y_train]
y_test = np.eye(len(label_mapping))[y_test]

# Define the model
model = models.Sequential()
model.add(layers.Flatten(input_shape=(target_size[0], target_size[1], 3)))
model.add(layers.Dense(64, activation="relu"))
model.add(layers.Dense(32, activation="relu"))
model.add(layers.Dense(len(label_mapping), activation="softmax"))

# Compile the model
model.compile(
    loss='categorical_crossentropy',
    optimizer=legacy.Adagrad(learning_rate=1e-5, decay=1e-6),
    metrics=['accuracy'])

# Define early stopping callback
early_stopping = EarlyStopping(monitor='val_loss', patience=20, verbose=1, mode='auto')

# Train the model
history = model.fit(
    X_train,
    y_train,
    validation_data=(X_test, y_test),
    batch_size=32,
    epochs=100,
    callbacks=[early_stopping]
)
import os
import cv2
import numpy as np
from sklearn.preprocessing import LabelEncoder
from tensorflow import keras

# Define the path to your image dataset directory
dataset_dir = 'Garbage_classification'

# Define the desired image size
target_size = (256, 256)

# Load and preprocess the images
all_images = []
all_labels = []

# Iterate over the images in the dataset
for label in os.listdir(dataset_dir):
    label_path = os.path.join(dataset_dir, label)
    if os.path.isdir(label_path):
        for image_file in os.listdir(label_path):
            image_path = os.path.join(label_path, image_file)
            image = cv2.imread(image_path)
            image = cv2.resize(image, target_size)
            all_images.append(image)
            all_labels.append(label)

# Convert the lists to numpy arrays
all_images_array = np.array(all_images)
all_labels_array = np.array(all_labels)

# Map string labels to integer values
label_encoder = LabelEncoder()
all_labels_encoded = label_encoder.fit_transform(all_labels_array)

# Save the label encoder for future reference
np.save('label_encoder.npy', label_encoder.classes_)

# Save the numpy arrays as files
np.save('all_images_array.npy', all_images_array)
np.save('all_labels.npy', all_labels_encoded)
import numpy as np
from sklearn.model_selection import train_test_split
from tensorflow import keras
from sklearn.preprocessing import LabelEncoder
```

```
def train(x_train, x_test, y_train, y_test):
    # Define and train your model here
    # Example placeholder code
    model = keras.models.Sequential()
    model.add(keras.layers.Flatten(input_shape=(256, 256, 3)))  # Flatten the input shape
    model.add(keras.layers.Dense(64, activation='relu'))
    model.add(keras.layers.Dense(64, activation='relu'))
    model.add(keras.layers.Dense(num_classes, activation='softmax'))

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

    model.fit(x_train, y_train,
              batch_size=32,
              epochs=10,
              verbose=1,
              validation_data=(x_test, y_test))

    model.save('your_model.h5')  # Save the trained model

    return model

def test(x_test):
    # Define your test function here
    # Example placeholder code
    model = keras.models.load_model('your_model.h5')  # Load your trained model
    predictions = model.predict(x_test)
    # Perform any necessary post-processing on predictions
    labels = np.argmax(predictions, axis=1)
    return predictions, labels

if __name__ == "__main__":
    # Load all images
    all_images_array = np.load('all_images_array.npy')

    # Load the class labels
    all_labels = np.load('all_labels.npy')

    # Load the label encoder for future use
    label_encoder = LabelEncoder()
    all_labels_encoded = label_encoder.fit_transform(all_labels)

    # Split the dataset into train and test sets, with a split ratio of 70:30
    x_train, x_test, y_train, y_test = train_test_split(
        all_images_array, all_labels_encoded, test_size=0.30, shuffle=True, random_state=78
    )

    # Data normalization to convert features to the same scale
    x_train = x_train.astype('float32') / 255.0
    x_test = x_test.astype('float32') / 255.0

    print('x_train shape:', x_train.shape)
    print(x_train.shape[0], 'train samples')
    print(x_test.shape[0], 'test samples')

    # Convert class vectors to one-hot encoding
    num_classes = len(label_encoder.classes_)
    y_train = keras.utils.to_categorical(y_train, num_classes)
    y_test = keras.utils.to_categorical(y_test, num_classes)
    train(x_train, x_test, y_train, y_test)

    predictions, labels = test(x_test)

    print('The predictions of the test set are:', predictions)
    print('The predicted labels are:', labels)

    # Evaluate the model
    _, accuracy = model.evaluate(x_test, y_test)
    print('Test accuracy:', accuracy)

model = keras.models.load_model('your_model.h5')  # Load your trained model
model.summary()
num_epochs = 10  # Define the number of training epochs
batch_size = 32  # Define the batch size

history = model.fit(x_train, y_train, validation_data=(x_test, y_test), epochs=num_epochs, batch_size=batch_size)

# Retrieve validation accuracy
validation_accuracy = history.history['val_accuracy']
# Assuming you have loaded or trained a model named 'model'
# and have a test data set named 'x_test'

# Get prediction probabilities for each class
predictions = model.predict(x_test)

# The 'predictions' variable will be a 2D array where each row represents a sample in 'x_test'
# and each column represents the prediction probability for a specific class

# For example, to get the confidence score for the first sample in 'x_test' for class 0:
confidence_score = predictions[0][0]
# Assuming you have loaded or trained a model named 'model'
# and have a test data set named 'x_test'

# Assuming you have loaded or trained a model named 'model'
# and have a test data set named 'x_test'

# Get predicted probabilities for each class
predictions = model.predict(x_test)

# Get predicted class labels using argmax
predicted_classes = predictions.argmax(axis=1)

# The 'predicted_classes' variable will be a 1D array where each element represents the predicted class label for a
specific sample in 'x_test'

# For example, to get the predicted class label for the first sample in 'x_test':
class_label = predicted_classes[0]
```