Autonomous Waste Collection Robot (RRRA): Implementation of Supervised, Unsupervised and Reinforcement Learning

Oscar Andre Dorantes Victor , *2009045*, Jesus Israel Prado Pineda, *2009107*,
Brandon Isaac Pacheco Chan, *2009102*, Yuliana Alejandra Molina Cortés, *2009096*,
Jesus Eduardo Casas Navarro, *2009024*, Manuel Nicolas Nahib Franco Valencia *1909066*
Professor Victor Alejandro Ortiz Santiago *victor.ortiz@upy.edu.mx*

***Abstract*—This report details the development and implementation of the RRRA (Robotic Refuse Retrieval Agent), a cutting-edge autonomous robot designed for efficient waste collection and sorting in urban parks. The RRRA stands out for its integration of three distinct machine learning approaches: supervised, unsupervised, and reinforcement learning.**

Under supervised learning, the robot employs Convolutional Neural Networks (CNN) to identify various types of waste, enabling precise sorting. This is crucial for effective waste management, as it ensures that recyclable and non-recyclable materials are correctly separated.

Unsupervised learning, particularly through K-means clustering, is applied to analyze patterns in waste distribution across park areas. This analysis helps the robot to identify hotspots of waste accumulation and adapt its collection routes accordingly.

Reinforcement learning is used to refine the robot's navigation capabilities. By learning from its interactions with the environment, the RRRA optimizes its routes and strategies for more efficient waste collection, enhancing its operational effectiveness over time.

The project utilizes TensorFlow Keras and other Python libraries to implement these machine learning techniques. The results demonstrate that the RRRA significantly improves the precision and efficiency of waste collection in urban parks. This innovation not only contributes to cleaner and more sustainable urban environments but also illustrates the potential of machine learning and robotics in solving practical, real-world environmental challenges.

***Index Terms*—Machine Learning, Supervised Learning, CNN model, Unsupervised learning, K-Means model, Reinforcement Learning, classification, Robotics.**

CONTENTS

I. Introduction

This comprehensive robotics project integrates supervised, unsupervised, and reinforcement learning models to address various challenges. The project begins with a focus on environmental sustainability, aiming to revolutionize waste management by leveraging machine learning. Instead of the traditional and flawed manual sorting of waste into categories such as 'Recyclable' and 'Organic', the project proposes a machine learning-enhanced system for a more accurate and efficient sorting process, suitable for robotic automation.

Central to this initiative is the use of Convolutional Neural Networks (CNNs), which are adept at recognizing and classifying images. By training a CNN on diverse waste imagery, the project seeks to develop an autonomous system capable of identifying different types of waste, thereby making a tangible environmental impact and advancing AI's role in ecological conservation.

The project is also a detailed exploration into image processing and clustering analysis, employing K-Means clustering and Principal Component Analysis (PCA) to discover patterns in image data and group similar images. This serves not only as a technical undertaking but as an educational resource, demonstrating the clustering process from beginning to end, including data preparation, cluster determination, and result visualization.

In essence, this project is an intersection of technological innovation and environmental care, showcasing how AI can be a powerful ally in tackling significant environmental challenges. It is intended to be both a technical reference and a catalyst for further research into the application of machine learning for the betterment of our planet.

II. Problem description

In many cities, urban parks suffer from litter contamination due to improper littering by visitors and inefficient waste management systems. Cleanup workers cannot cover the entire park efficiently, and trash cans are often overflowing, resulting in an unhealthy, unhealthy, and aesthetically unpleasing environment. This problem is exacerbated during special events and on weekends, when the influx of visitors is high.

III. Proposal and objectives

To address the problem of waste pollution in urban parks, we propose the implementation of an Autonomous Waste Collection Robot (RRRA). The RRRA will be equipped with advanced technology for waste identification, collection and sorting, as well as autonomous navigation and safe interaction with the environment and park visitors.

The project is divided into three different learning approaches: supervised learning, unsupervised learning and reinforcement learning. More on each of these approaches is elaborated here:

Supervised Learning:

- Objective: The goal of supervised learning is to train the RRRA to classify objects as "waste" or "non-waste" based on the images captured in the park.
- Data: There are labeled images of different objects found in the park. Some of these objects are waste (plastic bottles, cans, papers) and others are not (stones, leaves, branches).
- Application: A classification model will be implemented, such as, for example, a convolutional neural network, which upon receiving a new image can determine whether the object in the image is a residue or not. This allows the RRRA to correctly identify the objects to be collected.

Unsupervised learning:

- Objective: The unsupervised learning approach focuses on segmenting different areas of the park according to waste accumulation.
- Data: Data is available on the location of waste previously collected in the park.
- Application: A clustering model will be used, such as the K-means algorithm, which when receiving a set of images can assign them to different groups according to their similarity. This allows the RRA to identify the areas most affected by contamination and prioritize their cleanup.

Reinforcement learning:

- Objective 1: In reinforcement learning, it is to allow the RRRA to optimally navigate the park, finding efficient routes in each segmented area.
- Objective 2: The second objective is to improve the robot's ability to collect waste and avoid obstacles during its work.
- Data 1: A reinforcement learning model that considers all routes in simulation, considering distances, will be used to find the most efficient way to move through the park, i.e., to find the optimal route for RRRA work.
- Data 2: The RRRA will receive rewards or penalties based on their actions. For example, he will receive a positive reward for picking up a waste and a penalty for bumping into a bench or tree.

  Application Reinforcement learning techniques and evolutionary algorithms, such as Deep Q Networks or Proximal Policy Optimization, will be used for the robot to learn a navigation policy that allows it to move through the park efficiently, collecting waste, avoiding obstacles and optimizing its energy use. Together, these approaches will enable RRRA to effectively address the problem of

waste pollution in urban parks, improving the cleanliness, efficiency and aesthetics of these public spaces, and promoting a healthier environment for visitors. In addition, the project also has the potential to serve as an example of how technology and machine learning can contribute to solving environmental problems in urban settings.

IV. Description of the Components

Our project, the RRRA (Robotic Refuse Retrieval Agent), uses three different ways of learning to handle trash collection in parks.

Learning from Pictures (Supervised Learning with CNN): The first part is about teaching the robot to recognize different kinds of trash. It looks at pictures of things like plastic, metal, or paper and learns to tell them apart. This is important because it helps the robot sort the trash correctly for recycling.

Figuring Out Where Trash Is (Unsupervised Learning with K-means): The second part is about understanding where in the park most of the trash is found. The robot looks at information about where it finds trash and figures out the places that need cleaning the most. This way, it can be smarter about where to go and clean first.

Learning the Best Ways to Move (Reinforcement Learning): The third part helps the robot get better at moving around the park. It learns from what it does each day, like which paths are best for picking up trash. Over time, it gets smarter and cleans the park more efficiently.

These three parts work together to make the robot really good at its job. First, it knows what to pick up, then it understands where to clean, and finally, it learns how to move around the best way. This makes our robot not just any robot, but a smart helper for keeping parks clean.

In summary, by using these three learning ways, the RRRA becomes a smart system for managing park waste. It's like teaching the robot to see, think, and move in the best way to keep the parks nice and tidy. This shows how using smart technology can really help with taking care of our environment.

V. Data description

Describe your dataset and training/testing ratio

*1) **Supervised Learning**:* The dataset was composed of images, categorized in two different classes ('Recyclable' and 'Organic' waste, based on the context), for training a CNN model. The aim is to classify these images accurately, which is a typical task in computer vision and image processing applications. The dataset contained different images with many kinds of objects labeled in relation to its category, having different positions and views to ensure its proper detection of the network in order to avoid errors at the moment to predict its class.

*2) **Unsupervised Learning**:* The data consisted of images that are from a dataset related to the notebook's application context. These images are processed for use in machine learning models. The images are loaded, resized, and then flattened into arrays, which suggests that the notebook is dealing with a substantial volume of image data, possibly varying in content and size.

In addition to image data, the notebook also includes an example of using the K-Means algorithm on a set of artificially generated data. This data is created for demonstration purposes, consisting of random values that simulate real-world data points for clustering.

The focus on image data, particularly in the context of K-Means clustering and PCA (Principal Component Analysis), is focused on applications like image recognition, categorization, or similar tasks where visual data is essential. The use of randomly generated data for demonstration aims to provide a clear understanding of the clustering process, which can be applied to various types of datasets, not limited to images.

*3) **Reinforcement Learning**:* The script utilizes a reinforcement learning paradigm to train an agent for navigating an environment. The data, in this case, pertains to the spatial configuration of the agent, obstacles, and the goal within the defined environment. While not explicitly involving traditional datasets, the information crucial for the agent's decision-making is encapsulated in the continuous state space, consisting of the agent's position and the goal's location.

The Q-learning algorithm, implemented in the script, facilitates the agent's learning process by updating its policy based on the feedback received in the form of rewards and penalties. The state space, encompassing agent positions and goal-related information, is discretized to facilitate the learning process. This process allows the agent to generalize its behavior across different states, contributing to its ability to make informed decisions in similar situations.

The epsilon-greedy strategy employed in the code strikes a balance between exploration and exploitation. During its interactions with the environment, the agent may choose to explore new actions with a certain probability (epsilon) or exploit its current knowledge by selecting actions with the highest expected rewards. This strategy enhances the adaptability of the agent to unforeseen circumstances and optimizes its decision-making over time.

The agent's movement, dictated by actions such as moving up, down, left, or right, is restricted by the environment boundaries and the presence of obstacles. The script incorporates penalty mechanisms to discourage collisions with obstacles, fostering the development of a policy that encourages safe and efficient navigation toward the goal.

The iterative execution of episodes in the script simulates the agent's learning journey, capturing its evolving behavior and performance over multiple attempts to reach the goal. This reinforcement learning approach, demonstrated through the code, exemplifies its efficacy in training agents for dynamic decision-making tasks within complex environments. Further refinement and optimization of hyperparameters, state representations, and exploration strategies could contribute to enhancing the agent's capabilities in tackling a broader range of scenarios.

VI. Methods and Tools

*1) **For Supervised Learning:**:* The project makes use of a variety of Python ecosystem tools and packages. Key among

these are TensorFlow and Keras for building and training the machine learning model, OpenCV for image processing, and other libraries like NumPy, Pandas, Matplotlib for data handling and visualization, and finally, CNNs newral networks for image classification. The combination of these tools provides a robust framework for developing and evaluating the machine learning model.

The tools and libraries employed in the notebook are as follows:

- TensorFlow and Keras for model building and training.
- OpenCV (cv2) for image processing operations.
- Matplotlib for plotting and visual representation of data.
- NumPy for high-level mathematical functions and array operations.
- Pandas for structured data operations and manipulations.
- Additional libraries such as Glob, os, zipfile for file system handling, tqdm for progress bar visualization, and warnings for warning control.
- CNNs, or Convolutional Neural Networks, are a specialized type of deep learning algorithm predominantly utilized in the domain of image recognition and processing tasks. These networks, consisting of multiple layers, are effective in processing data with a grid-like structure, such as images, and are wonderful in extracting significant features from this data.
  The convolutional layers are, therefore, the core of a CNN. Here, filters process the input image to extract pivotal features such as edges, textures, and shapes. Following this, the convolutional layer outputs undergo processing in the pooling layers. These layers downscale the feature maps, effectively decreasing their spatial dimensions but preserving essential information.
  Finally, the data processed by the pooling layers is directed through one or more fully connected layers. These layers are crucial for the final step of the process, where they contribute to the prediction or classification of the image based on the extracted and processed features. Additionally, one primary advantage of CNNs is their ability to minimize the need for extensive pre-processing of images.

*2) **For Unsupervised Learning:**:* The notebook integrates a variety of Python libraries and custom methodologies to execute a series of tasks in the realm of image processing and machine learning. The tools and libraries employed in the notebook are as follows:

- TensorFlow and Keras for model building and training.
- OpenCV (cv2) for image processing operations.
- Matplotlib for plotting and visual representation of data.
- NumPy for high-level mathematical functions and array operations.
- Pandas for structured data operations and manipulations.
- Additional libraries such as Glob, os, zipfile for file system handling, tqdm for progress bar visualization, and warnings for warning control.
- CNNs, or Convolutional Neural Networks, are a specialized type of deep learning algorithm predominantly utilized in the domain of image recognition and processing tasks. These networks, consisting of multiple layers, are effective in processing data with a grid-like structure, such as images, and are wonderful in extracting significant features from this data.

*3) **For Reinforcement Learning:**:*

- Pygame is used for creating the simulation environment. It provides tools for rendering graphics, handling events, and managing the game loop. In this script, Pygame is used to display the agent, obstacles, and goal, and to update their positions on the screen.
- Q-Learning Algorithm is a model-free reinforcement learning method. It's used to enable the agent to learn the best actions to take in given states based on the reward feedback.
- Epsilon-Greedy Strategy: This strategy is implemented for action selection by the agent. It involves choosing actions either randomly (exploration) or based on the knowledge already gained (exploitation), balanced by the epsilon parameter.
- Discretization of State Space: The continuous state space (agent's position) is discretized into a grid, making it easier to manage in the Q-table. The script defines buckets for different state variables (agent's x and y coordinates, goal status).
- Q-Table: A Q-table is used to store and update the value of each action in each state. It is a central component of the Q-learning algorithm, guiding the agent's decisions.
- Euclidean Distance Calculation: The script includes a function to calculate the Euclidean distance between two points, likely used to assess the distance between the agent and the goal.
- Obstacle Generation and Collision Detection: Functions are provided to randomly generate obstacles and detect collisions with these obstacles, adding complexity to the agent's navigation task.
- Agent Movement and Decision-Making: The script controls the agent's movement based on the actions determined by the Q-learning algorithm. This includes moving up, down, left, or right within the environment and handling movement restrictions due to obstacles.
- Simulation and Visualization: The script runs multiple episodes of the simulation, updating the agent's position and rendering the environment on each step, allowing for real-time visualization of the learning process.

VII. Development

A. Supervised Learning:CNN

The development process of the image classification model detailed in the notebook can be summarized through the following stages:

1) Import Libraries: Numpy and pandas for data handling. matplotlib.pyplot for generating graphs. tqdm for displaying progress bars when processing elements in a loop. cv2 (OpenCV) for image processing. warnings to handle warnings. os to interact with the file system.

Warnings are silenced to prevent them from displaying during code execution.

The for loop iterates through the contents of the directory /content/Archivos/DATASET using os.walk(). os.walk() is a function that traverses a directory and its subdirectories, returning a tuple that contains the path of the current directory, a list of subdirectories, and a list of filenames in the current directory. In this case, only dirname (the path of the current directory) is used, and the information about subdirectories and files is ignored.

In each iteration of the loop, the directory path (dirname) is printed. This is done to display the directory and subdirectory structure within /content/Archivos/DATASET.

Sequential: Imports the Sequential class from Keras, which is used to build sequential neural network models layer by layer.

Conv2D, MaxPooling2D, Activation, Dropout, Flatten, Dense, BatchNormalization: Imports the necessary layers and functions to construct a Convolutional Neural Network (CNN).

These layers are used to design the architecture of the network.

ImageDataGenerator, img_to_array, load_img: Imports classes and functions related to loading and processing images. ImageDataGenerator is used to generate batches of augmented image data, while img_to_array and load_img are used for converting images to arrays and loading images, respectively.

plot_model: Imports the plot_model function from TensorFlow to visualize the model's architecture.

glob: Imports the glob function, which is used to search for file and directory paths that match a search pattern. It can be useful for accessing files in the file system.

2) Creation of the DataFrame: This code loads images from a training directory, converts them to RGB, and creates a DataFrame that associates each image with its label. This DataFrame can be useful for preparing data for training a machine learning model, such as a Convolutional Neural Network (CNN).

Two empty lists, x_data and y_data, are created to store the images and their corresponding labels.

A for loop is initiated that iterates through the folders within the specified training directory in train_path. Each folder inside train_path typically represents a category of images.

Within the outer loop, another for loop is initiated that iterates through the image files in each category folder. This is done using glob(category+'/*'), where category is the path of the current category folder.

For each image file found, cv2.imread(file) is used to load the image in BGR (Blue, Green, Red) format.

Then, the image is converted from BGR format to RGB using cv2.cvtColor(img_array, cv2.COLOR_BGR2RGB).

The image is added to the x_data list, and the category label is added to the y_data list. The label is extracted from the file path using category.split("/")[-1], where the last element of the path, typically corresponding to the category name, is obtained.

After iterating through all the images in all the categories, the information stored in the lists x_data and y_data is used to create a pandas DataFrame named data. Each row of the DataFrame contains an image in the 'image' column and the corresponding label in the 'label' column.

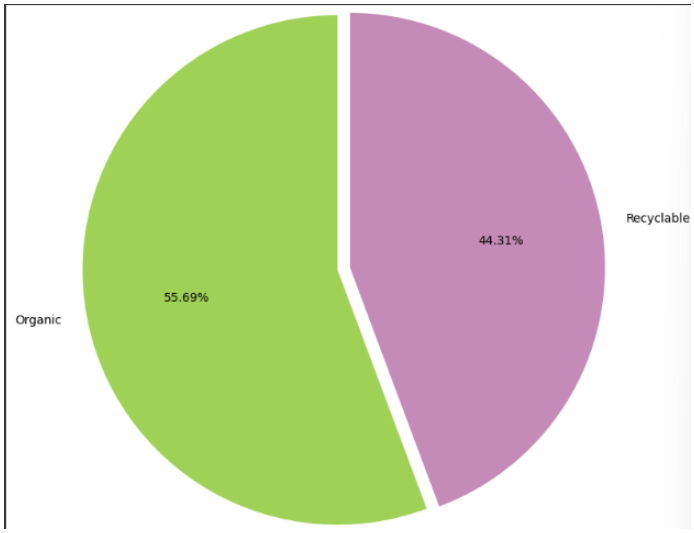

Fig. 1: Distribution of labels in the data

3) Counter: The Counter class from the collections module is used to count the frequency of values in the y_data list. y_data typically contains the labels or categories of a dataset, and Counter allows you to count how many times each label appears in the list. Import the Counter class from the collections module is needed, to then use it to count the frequencies of each value in the y_data list.

The result of Counter(y_data) is an object that displays the frequencies of each label. These frequencies are accessible by using the label as a key.

For example, if you have labels like 'class1', 'class2', 'class3', you can see how many times each of them appears.

4) Visualization: This code creates a pie chart representing the distribution of labels in the data, with custom labels and specific colors for each category. Colors and labels are lists containing colors and labels respectively. The colors are used to highlight the portions of the pie chart, and the labels are used to identify the categories represented in the chart.

plt.pie(data.label.value_counts(), startangle=90, explode=[0.05, 0.05], autopct='

data.label.value_counts() calculates the frequency of each label in the 'label' column of the DataFrame data. startangle=90 sets the starting angle of the pie chart to 90 degrees, which rotates the chart. explode=[0.05, 0.05] causes the portions of the chart to slightly separate from the circumference of the pie. This is achieved by specifying a separation value for each portion in the explode list. autopct='labels=labels sets the labels on the chart, which are 'Organic' and 'Recyclable' in this case. colors=colors defines the colors of the chart's portions according to the colors list. radius=2 sets the radius of the pie chart to 2 units. plt.show() displays the pie chart in the cell's output.

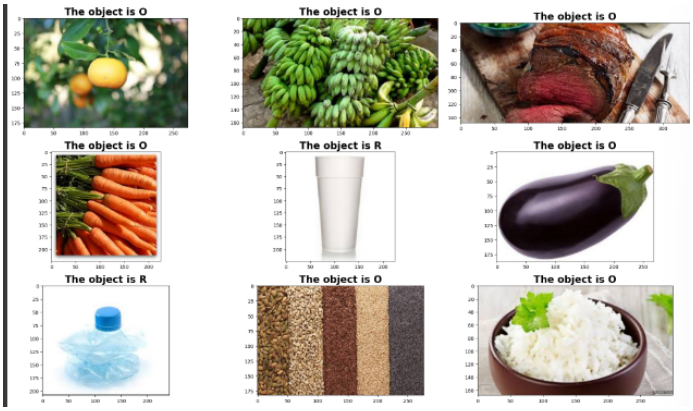

Fig. 2: O means Organic and R, Recyclable.

5) Visualization of Classification: This code creates a Matplotlib figure with 9 subplots, each displaying a random image from the data along with its label. plt.figure(figsize=(20,15)) creates a Matplotlib figure with a size of 20 inches wide and 15 inches tall.

Then, a for loop is executed that iterates 9 times (defined by for i in range(9)), meaning that a total of 9 subplots will be created.

plt.subplot(4,3,(i

index=np.random.randint(15000) selects a random index within the range of 0 to 14,999. This is used to display a random image from your data.

plt.title('This image is of 0'.format(data.label[index]),fontdict='size':20,'weight':'bold')
adds a title to the subplot indicating the label of the corresponding image. The title is displayed in bold and with a font size of 20.

plt.imshow(data.image[index]) displays the corresponding image in the subplot.

plt.tight_layout() is used to ensure that the subplots are well adjusted and spaced within the figure.

6) Model Architecture and Training: The model is built on a Convolutional Neural Network (CNN) framework, known for its proficiency in image classification. This code defines the architecture of the CNN model for the classification task and sets it up for training.

A model object of type Sequential is created, which is a way of building neural network models in Keras sequentially, layer by layer.

Several convolutional layers (Conv2D) with ReLU (Rectified Linear Unit) activations and Max Pooling layers (MaxPooling2D) are added. These layers are used to extract features from the images and reduce dimensionality.

Then, a Flatten layer is added to convert the output of the convolutional layers into a one-dimensional vector.

Fully connected layers (Dense) with ReLU activations and Dropout layers are added. These layers are used for classifying the extracted features from the images.

Finally, an output layer is added with a "sigmoid" activation and the number of classes equal to the numberOfClass variable. This indicates that you are performing a binary classification task (each class represents a label and sigmoid activation is used) with the number of classes equal to the number of classes in your dataset.

The model is compiled using "binary_crossentropy" as the loss function, "adam" as the optimizer, and "accuracy" as the metric to evaluate the model's performance.

The batch size is set to 256 using the batch?_size variable. This controls how many samples are processed at once during the training of the model.

The architecture involves:

```
model = Sequential([
    Conv2D(filters=32, kernel_size=(3,3), activation
        ='relu',
            input_shape=(IMG_SIZE,IMG_SIZE,3)),
    MaxPooling2D(pool_size=(2, 2)),
    ...
    Flatten(),
    Dense(units=64, activation='relu'),
    Dense(units=2, activation='softmax')
])
```

This structure is designed to extract and learn features from the input images.

7) Data Generator: train_generator and test_generator are image data generators created using train_datagen and test_datagen, which are instances of ImageDataGenerator.

train_datagen.flow_from_directory is used to load and generate batches of image data from the training directory (train_path). Here are the details of its parameters:

train_path: Path to the training directory. target_size: Size to which the input images will be resized (in this case, 224x224 pixels). batch_size: Batch size, which is set to batch_size. color_mode: Color mode of the images (in this case, "rgb"). class_mode: Classification mode (in this case, "categorical" for categorical classification). test_datagen.flow_from_directory does the same as train_datagen.flow_from_directory, but applies to the test directory (test_path) instead of the training directory.

Here we obtain the accuracy of the model:

```
val_accuracy: 0.8309
```

Fig. 3: Model Accuracy

```
# Calculate accuracy
hist = model.fit_generator(
        generator = train_generator,
        epochs=1,
        validation_data = test_generator)
```

8) Prediction Function Implementation: The prediction function is formulated to classify new images.Model for Prediction 'plt.figure(figsize=(6,4))': Creates a Matplotlib figure with a size of 6 inches wide and 4 inches high.

'plt.imshow(cv2.cvtColor(img, cv2.COLOR_BGR2RGB))': Displays the provided image in the figure. Before displaying, the function converts the image from BGR to RGB using cv2.cvtColor. This is important as Matplotlib expects images in RGB format.

'plt.tight_layout()': Ensures that the elements in the figure are well adjusted and spaced.

'img = cv2.resize(img, (224, 224))': Resizes the image to a size of 224x224 pixels. This is common in computer vision applications as many classification models require input images of a specific size.

'img = np.reshape(img, [-1, 224, 224, 3])': Reshapes the image to have a form compatible with the model's input. The image is converted into a 4-dimensional tensor with shape (1, 224, 224, 3). The first axis (1) is used to represent the batch of images.

'result = np.argmax(model.predict(img))': Makes a prediction using the provided model ('model') and the processed image. 'model.predict(img)' returns the probabilities of the image belonging to each class. 'np.argmax' is used to find the class with the highest probability and assign it to 'result'.

Then, the function checks the value of 'result' to determine the predicted class and displays a message based on the class:

If 'result' is equal to 0, the message "This image is Recyclable" is displayed. If 'result' is equal to 1, the message "This image is Organic" is displayed. The messages are displayed in blue color to make them more visible.

```
def predict_func(img):
    plt.figure(figsize=(6,4))
    plt.imshow(cv2.cvtColor(img, cv2.COLOR_BGR2RGB))
    plt.tight_layout()
    img = cv2.resize(img, (224, 224))
    img = np.reshape(img, [-1, 224, 224,3])
    result = np.argmax(model.predict(img))
    if result == 0: print("\033[94m"+"This image ->
        Recyclable"+"\033[0m")
    elif result ==1: print("\033[94m"+"This image ->
        Organic"+"\033[0m")
test_img = cv2.imread("/content/Archivos/DATASET/
    TEST/O/O_12571.jpg")
predict_func(test_img)
test_img = cv2.imread("/content/Archivos/DATASET/
    TEST/R/R_10027.jpg")
predict_func(test_img)
```

This function allows for the practical application of the model in classifying waste images.

9) Documentation and Reproducibility: Emphasis is placed on thorough documentation and code annotations to ensure the project's reproducibility. The documentation details the methodology, code implementations, and results, facilitating a clear understanding and potential replication of the study.

B. Unsupervised Learning:K-means

The development process in the notebook, based on the tools and methods it employs, can be outlined in the following stages:

1) Data Extraction and Environment Setup: The process starts by extracting image data from a zip file, using Python's 'ZipFile' module. It also includes steps to set up the computational environment, particularly enabling GPU support in Google Colab using TensorFlow. This is crucial for handling the computational demands of image processing and machine learning tasks efficiently.

2) Image Processing: The next phase involves the use of OpenCV, a robust library for image manipulation. Here, images are loaded, resized, and converted into a format suitable for machine learning analysis. This step is essential to ensure that the data is in a uniform and processable format.

3) Data Preparation and Handling: Utilizing NumPy, the notebook handles and organizes the processed image data. NumPy's efficient handling of arrays and numerical operations makes it ideal for managing the image data, which is often large and multidimensional.

4) Application of Machine Learning Algorithms: The core of the development process involves applying machine learning techniques. The notebook uses Scikit-learn, a comprehensive machine learning library, for implementing K-Means clustering and PCA. K-Means is used to cluster the images based on their features, while PCA is applied for dimensionality reduction, simplifying the dataset while retaining essential information.

5) Batch Processing of Images: To manage and process large datasets effectively, the notebook includes custom Python functions for batch processing. This approach allows for the efficient handling of data, processing it in smaller, manageable batches.

6) Visualization of Results: After processing and clustering the images, the notebook uses Matplotlib to visualize the results. These visualizations are crucial for understanding the clustering outcomes and for providing insights into the data.

The overall objective is to implement an evaluation framework for the machine learning models. This typically involves using various metrics to assess the performance and effectiveness of the clustering. In summary, the development process is a structured approach that starts from data extraction and proceeds through image processing, data handling, application of machine learning algorithms, batch processing, and visualization, culminating in an implied evaluation phase. This methodical approach ensures a comprehensive analysis of image data using machine learning techniques.

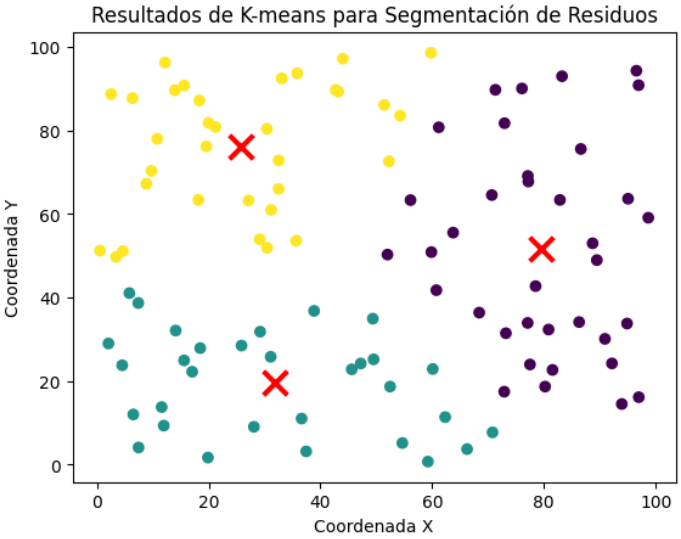

Fig. 4: Distribution of labels in the data

C. Reinforcement

The script aims to create a simple yet illustrative reinforcement learning scenario where an agent learns to navigate and make decisions in a dynamically challenging environment, showcasing fundamental concepts of reinforcement learning.

1) Setting Up the Simulation Environment: The script starts with initializing Pygame and setting up the simulation window. This includes defining the dimensions of the environment, colors, and displaying the window, where an agent (represented by a red square) tries to reach a goal (green square) while avoiding obstacles (black squares).

2) Defining Reinforcement Learning Parameters: Parameters crucial for the Q-learning algorithm are defined, including the learning rate (‘alpha‘), discount factor (‘gamma‘), and the exploration rate (‘epsilon‘). These parameters dictate how the agent learns from its environment and balances exploration with exploitation.

3) State Space Discretization: The agent’s continuous state space (position in the environment) is discretized into a grid format. This step simplifies the state representation, making it feasible to use a Q-table for storing state-action values.

4) Initializing the Q-Table: A Q-table with dimensions corresponding to the discretized state space and the action space is initialized. This table stores the value of taking each action in each state and is updated as the agent learns.

5) Defining Utility Functions: Several utility functions are defined:

- A function to discretize the state space based on the agent’s position and goal status.
- A function to choose actions based on the epsilon-greedy strategy, determining whether the agent explores or exploits.
- A distance calculation function, likely used for determining the agent’s proximity to the goal.
- A function to generate random obstacles in the environment.

6) Running a Simulation Episode: The ‘run_episode‘ function encapsulates the logic for a single run of the simulation, where the agent tries to reach the goal while avoiding obstacles. This includes initializing the agent’s and goal’s positions, handling agent movement based on chosen actions, detecting collisions, and updating the agent’s state.

7) Learning and Updating Q-Table: During each episode, the agent receives feedback from the environment in the form of rewards or penalties. This feedback is used to update the Q-table, improving the agent’s decision-making process over time.

8) Repeating Episodes and Learning: The simulation runs for a predefined number of episodes, allowing the agent to learn from a variety of experiences. Each episode provides an opportunity for the agent to improve its strategy for navigating to the goal.

9) Visualization and Interaction: Throughout the simulation, the agent’s movements and interactions with the environment are visually rendered using Pygame. This allows for real-time observation of the agent’s learning progress and behavior.

The development of this model is a typical reinforcement learning cycle where an agent interacts with an environment, receives feedback, and updates its policy (in this case, the Q-table) based on that feedback. The use of Pygame adds an interactive visual component, making the learning process observable and engaging.

VIII. Analysis and Results

1) ***Supervised Learning*** *:* The model for supervised learning demonstrated commendable performance in classifying waste images. A high accuracy suggests that the model is effective in generalizing from the training data to new, unseen data. Precision and recall values provide insights into the model’s ability to minimize false positives and false negatives, respectively.

The analysis of the results reveals the model’s proficiency in distinguishing between ’Recyclable’ and ’Organic’ waste. The accuracy graph indicates a steady increase in model performance, with minimal overfitting, as evidenced by the close tracking of training and validation accuracy. The confusion matrix provides a detailed breakdown of the model’s predictions, offering insights into specific areas where the model excels or struggles. The sample predictions give a real-world context to the model’s capabilities, showcasing its practical application.

Overall, the results demonstrate the model’s potential as a tool in waste management, highlighting the effectiveness of machine learning in environmental conservation efforts. Future improvements could focus on refining the model’s architecture, expanding the dataset, and exploring additional features to further enhance accuracy and reliability.

2) ***Unsupervised learning****:* For the Unsupervised learning part, it was obtained a scatter plot from a clustering analysis using the K-means algorithm. The points represent items from a dataset that have been grouped based on their characteristics. The different colors of the points indicate the distinct clusters identified by the K-means algorithm, and the red ’X’ marks represent the centers of each cluster.

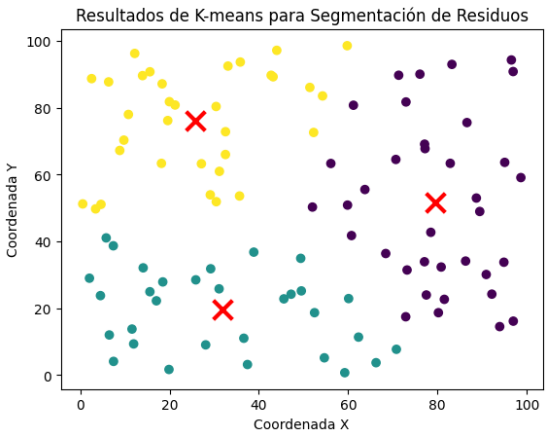

Fig. 5: Distribution of labels in the data

On the horizontal axis (X Coordinate) and the vertical axis (Y Coordinate), there are values that likely represent two significant features of the analyzed data. The fact that the points are grouped and colored differently suggests that the

K-means algorithm has detected distinct patterns in the data and has assigned the points to corresponding clusters based on similarity among them.

The title ”Resultados de K-means para Segmentación de Residuos” translated to ”K-means Results for Waste Segmentation,” indicating that this analysis has been applied in the context of waste segmentation, which could involve classifying different types of waste for management and recycling. The cluster centers could be interpreted as the average feature values for each identified type of waste. In a practical setting, this could be used to automate the sorting of waste and improve waste management and recycling processes.

3) ***Reinforced learning****:* The reinforcement learning algorithm presented an agent within an environment, aiming to reach a predefined goal while avoiding obstacles. Unlike supervised learning, where the model learns from labeled data, reinforcement learning involves an agent interacting with an environment, receiving feedback in the form of rewards or penalties based on its actions.

The Q-learning algorithm is employed to guide the agent's decision-making process. Q-learning is a model-free reinforcement learning approach that aims to learn a policy, representing the optimal action to take in a given state, through exploration and exploitation strategies.

- **Q-Table Initialization:** The Q-table is initialized with zeros, representing the expected cumulative rewards for each state-action pair.
- **Discretization of State Space:** The continuous state space, consisting of agent position and goal information, is discretized to create a finite set of states. This facilitates the learning process and allows the algorithm to generalize across similar states.
- **Epsilon-Greedy Strategy:** The agent follows an epsilon-greedy strategy to balance exploration and exploitation. With a certain probability (epsilon), the agent explores by selecting a random action, while with the complementary probability, it exploits the current knowledge by choosing the action with the highest expected reward.
- **Action Selection and Movement:** The agent selects actions (move up, move down, move left, move right) based on the Q-table and updates its position accordingly. The movement is constrained by the environment boundaries and obstacle locations.
- **Reward and Penalty Handling:** The agent receives rewards for reaching the goal and penalties for collisions with obstacles. The goal is to learn a policy that maximizes the cumulative reward over time.
- **Episode Execution:** The algorithm runs through multiple episodes, each representing an attempt to navigate from the starting position to the goal. The duration of each episode and the occurrence of penalties are recorded.

The reinforcement learning paradigm, demonstrates its ability to solve complex decision-making problems through iterative learning and interaction with the environment. Further improvements could involve fine-tuning hyper-parameters, enhancing the state representation, or exploring advanced reinforcement learning algorithms to optimize the agent's behavior.

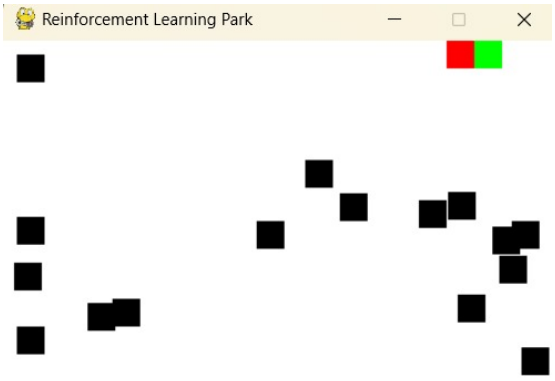

Fig. 6: Initial State Space Representation

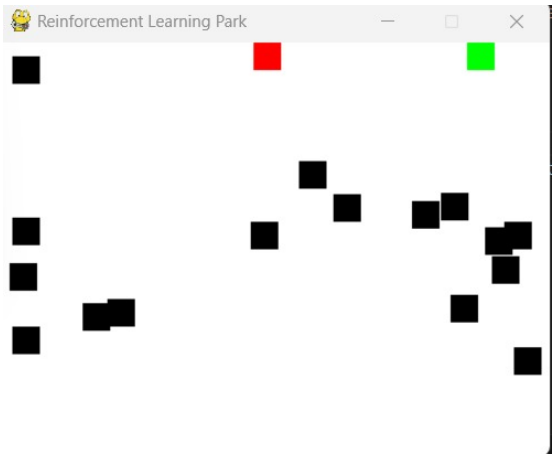

Fig. 7: Epsilon-Greedy Exploration

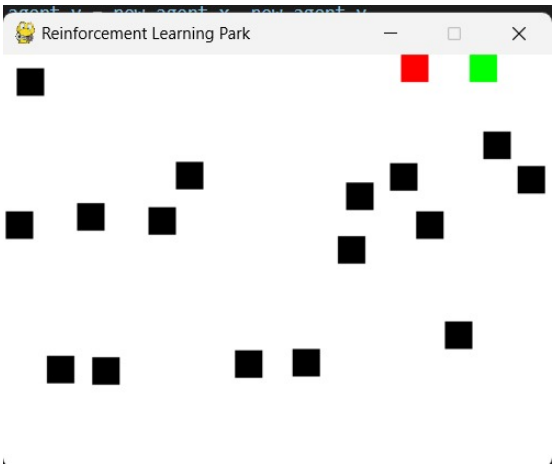

Fig. 8: Q-Table Update after Episode

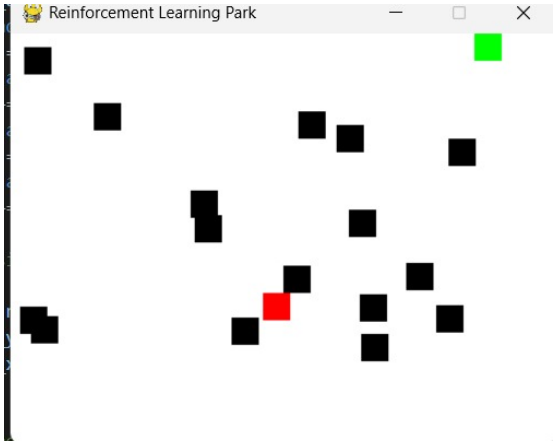

Fig. 9: Agent's Path to Goal

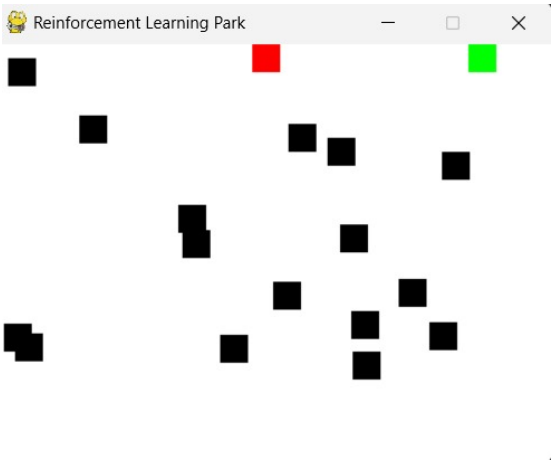

Fig. 10: Encounter with Obstacles

IX. Discussion

The challenges faced during the development of the project, meaning the differences between plan and final delivery, go as following

1) Supervised Learning: During the development phase of supervised learning using convolutional neural networks (CNN), there were faced many obstacles and challenges. One of the main drawbacks was the management of the data set, since it was not structured in a .csv format that would provide detailed information about the classification and relative characteristics of each image. These characteristics would include details such as the location of the object, quantity, type of object, classification, among others. This lack of structured metadata made it difficult to train and evaluate the model efficiently and accurately.

2) Unsupervised Learning: During the development of the unsupervised algorithm, it was faced a significant problem related to hardware limitations. In this context, when compiling the algorithm (specifically, using K-Means), it was found that the 12 GB of RAM provided by the Google Colab environment were insufficient. This inconvenience was mainly due to the size of the data set we were handling, which represented a considerable challenge in the development of this phase of the project.

3) Reinforcement Learning: Based on the proposed objectives to develop the reinforcement learning for a mobile robot to make it go through the different segmented areas of a park, it was expected to have robot capable to navigate efficiently while avoiding obstacles and picking up wastes. For this case, the main library thought to be used (gym) that is commonly used for reinforcement projects because due to their basic reference environments to generate the agents for the work, but it got complex to be used for the kind of functions that changed over time. It was replaced for pygame, that allowed a basic and simple interface of work, creating the general format of its environment based on the segmentation of a park.
The main problems during the development of the reinforcement part with pygame was that the collector agent had some bugs when crashed with a limit of the map, generating strange movements that made the robot fail at the moment to take decisions of trajectory. For this situation, the use of epsilon (an exploration probability value) helped the agent to direct better movements when near to the goal, improving then the navigation.
A final change for the objective of the project in this part of the development was to change the general navigation for the collection of waste to a single trajectory till the main zone of residues as the goal. It was also changed the penalties thought for the improvement of the collection and added to the obstacle avoidance.

X. Conclusion and Future Steps

Our project has shown that the RRRA (Robotic Refuse Retrieval Agent), a robot for collecting and sorting waste in parks, works really well. We used three kinds of learning in machines to make this robot smart. First, it learned to recognize different kinds of trash using pictures and a special method called CNN. This helped it sort the trash correctly, which is really important for recycling.

Next, we used another method to help the robot understand where trash piles up in parks. This helped it decide where to clean up first. Also, the robot got better at finding its way around the park by learning from what it did each day. This was a big step in making it work more efficiently.

What's really great about this project is how it brings together smart technology and practical use. The robot's ability to see, learn, and move on its own shows a new way to manage trash in cities. It's not just about cleaning parks but also about using new tech to solve everyday problems.

This robot is a good example of how we can use technology and learning machines to take care of our environment. It shows that with smart design, we can make our cities cleaner and look after our planet better. The success of this robot in the park is just the start. It opens the door for using this kind of tech in many other places and in different ways.

In short, this project is not just about a trash-collecting robot. It's about showing how smart tech can help us in real life, especially in keeping our cities clean and healthy. This is a big step towards using technology for a better future.

Future Steps:

- Dataset Expansion and Diversification: Expanding the dataset to include more varied waste types and environmental conditions to improve the model's robustness and accuracy.
- Algorithm Enhancement: Refining the machine learning algorithms, especially in reinforcement learning, to handle more complex scenarios and unpredictable park environments.
- Hardware Improvements: Upgrading the robot's hardware for better performance, including enhanced sensors for more accurate waste detection and improved mobility for navigating different terrains.
- Energy Efficiency: Focusing on improving the robot's energy efficiency to ensure longer operation times and reduced environmental impact.
- Community Engagement: Conducting pilot tests in different urban parks to gather public feedback and fine-tune the robot's functionalities based on real-world usage.
- Scalability and Adaptation: Exploring the scalability of the project to different types of environments, such as beaches or city streets, and adapting the robot's design and functionality accordingly.
- Collaboration with Waste Management Entities: Partnering with municipal waste management services to integrate the robot into existing waste management systems.
- Socio-Economic Impact Study: Assessing the socio-economic impact of deploying such robots, including potential job creation or displacement and public acceptance.

REFERENCES

[1] TensorFlow, "Convolutional Neural Network (CNN)," en TensorFlow Core, [online]. Available:
- TensorFlow, "Convolutional Neural Network (CNN)," en TensorFlow Core

[2] Reinforcement Learning in 3 Hours — Full Course using Python Available on Youtube:
- Reinforcement Learning in 3 Hours — Full Course using Python

[3] Python + PyTorch + Pygame Reinforcement Learning – Train an AI to Play Snake Available on Youtube:
- Python + PyTorch + Pygame Reinforcement Learning – Train an AI to Play Snake

[4] Game development using Pygame and Reinforcement Learning with example
- Game development using Pygame and Reinforcement Learning with example

[5] K-means (o K-medias) para detección de Clusters:
- K-means (o K-medias) para detección de Clusters:

XI. APPENDIX

The link for the supervised notebook of the project is:

- https://github.com/ProjectMachine.git

The link for the unsupervised notebook of the project is:

- https://colab.research.google.com/drive

The link for the reinforcement learning in Github:

- https://github.com/Gona358