Final Project
Machine Learning Solutions

Joseph Jesus Aguilar Rodriguez, *1809003*, Gabriela Elizabeth Avila Chan, *2009003*,
Leonardo Cañamar Amaya, *2009018*, Marco Alejandro González Barbudo, *1909073*,
Diego Armando May Pech, *2009092*
Victor Alejandro Ortiz Santiago, *Professor*.

***Abstract*—This document presents a comprehensive code solution aimed at ensuring the safety of Gupy, the beloved mascot of Universidad Politécnica de Yucatán. By employing various machine learning techniques. The project focuses on Gupy detection from images, utilizing datasets of cats to precisely identify Gupy. The code implements machine learning to develop an algorithm capable of safeguarding Gupy, promoting his well-being, and addressing the community's concerns.**

***Index Terms*—Artificial Intelligence, Machine Learning, Learning algorithms, Supervised learning, Image recognition, Code implementation.**

I. Introduction

In light of recent events surrounding the unfortunate incident in which the beloved Polytechnic University of Yucatan's mascot, Gupy, was injured, there is an urgent need to implement enhanced safety measures to ensure the protection and well-being of Gupy. This paper proposes a comprehensive set of code solutions that leverage machine learning techniques for a future application in robotics to mitigate potential risks and ensure the continued security of Gupy.

The project first part focuses on Gupy detection from images, utilizing datasets of cats to precisely identify him within the university environment. By implementing a machine learning algorithm, this initiative seeks to address the concerns of the campus community and provide a proactive approach to Gupy's safety. Supervised learning techniques work best for this stage, and is thus why the perceptron algorithm will be used to accomplish this task.

Stage two makes use of unsupervised learning to analyse the emotional state of Gupy. This will help to determine when the Gupy feels threatened, scared, or general unwell. Knowing Gupy's emotional state can help us to determine when Gupy is in trouble. Sentiment analysis can be performed using either supervised or unsupervised learning. Due to the difficulty for a human to have an unbiased understanding of the emotional state of an animal, it was decided that using unsupervised learning to identify the patterns without human bias would serve the purposes of this project better.

The third, and final, stage of the project employs reinforcement learning to teach the security platform how to intercept incoming threats. When the model detects that Gupy is in an emotionally distressed state, the security platform will move to midpoint between Gupy and the source of distress. The hope is that this will discourage any threats from approaching further, but also allows for the security platform to intervene if the threat is not dissuaded from approaching.

II. State of Art

For this project Supervised Learning Using Perceptron, Unsupervised Learning through Clustering, and Reinforcement Learning with Q-Learning were implemented. Those are three distinct paradigms within the field of machine learning, each with unique methodologies and applications.

A. Supervised Learning Using Perceptron

The perceptron is a foundational concept in supervised learning, representing the simplest form of a neural network. It operates by receiving multiple input signals and outputs a signal if the sum of these inputs exceeds a certain threshold. This method is primarily used for binary classification tasks. During training, the perceptron adjusts its weights to minimize prediction errors, aligning closely with the actual outputs. However, perceptrons are limited to modeling linearly separable functions and cannot solve non-linear problems. A key reference for perceptrons in supervised learning is "Pattern Recognition and Machine Learning" by Christopher M. Bishop.[1]

B. Unsupervised Learning through Clustering

In unsupervised learning, clustering is a critical technique used to group a set of objects based on their similarities. Algorithms like K-Means, Hierarchical Clustering, and DBSCAN are commonly employed for this purpose. The main objective is to discover inherent groupings in the data. For example, K-Means aims to partition observations into clusters with the nearest mean. Clustering finds applications in market segmentation, social network analysis, and astronomical data analysis, among others. A comprehensive reference on this topic is "Data Clustering: Algorithms and Applications" by Charu C. Aggarwal and Chandan K. Reddy. [2]

C. Reinforcement Learning with Q-Learning

Q-Learning, a model-free reinforcement learning algorithm, focuses on learning the value of an action in a specific state. It is characterized by its ability to handle problems with stochastic transitions and rewards without requiring a model of the environment. In Q-Learning, the learner independently

discovers actions that yield the most reward. This algorithm is adept at learning the optimal policy for a Markov decision process (MDP), even in the absence of known model dynamics. Q-Learning has been effectively applied in robotics, video games, and economics. "Reinforcement Learning: An Introduction" by Richard S. Sutton and Andrew G. Barto is an excellent resource for understanding reinforcement learning, including Q-Learning. [3]

III. Development

A. Supervised Learning

In order to achieve the goal, a machine learning algorithm was designed and trained to predict if the output picture was Gupy or not. Developing a Jupiter Notebook that contains a code it is enhance the training of the algorithm to perform this labor.

1) Dataset: This Perceptron was fed a carefully curated self-made dataset, encompassing a diverse range of photos featuring Gupy alongside images of other traditional cats. I This deliberate inclusion of various cat breeds aimed to provide the model with a rich and varied dataset, facilitating the learning process for the algorithm to accurately identify the distinct features of Gupy.

The dataset compilation involved a comprehensive selection of images capturing Gupy in different poses, lighting conditions, and backgrounds, ensuring that the model could generalize well across various scenarios. Additionally, the inclusion of images of other cats served to challenge the model and enhance its ability to discriminate between Gupy and other feline counterparts. In the following sections, a detailed examination of the algorithm and its outcomes will be presented.

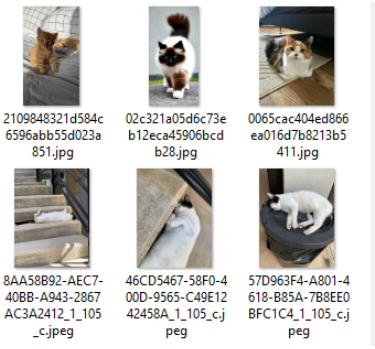

Fig. 1. Preview of the dataset made for the model.

2) Model pre-preparation: This section aims to provide an overview of the design and training process of the Perceptron for the accurate Gupy detection.

To begin with, `zipfile` module was used to extract the contents of a ZIP file. Then, the path to the ZIP file and the directory were specified. The ZIP file is opened and the all the elements are extracted archive to the specified destination directory using `zip_ref.extractall(directorio_destino)`. Finally, a message indicating the successful extraction of files to the specified destination directory is printed.

Then, all the necessary libraries to process, train, test the model and plot the prediction results are added. The path for the folder containing images is set by defining the variable `carpeta_imagenes` and assigning it in the path '/content/gupy_dataset-20231104T191923Z-001/gupy_dataset', indicating the folder where the images are located. After that, the variable `csv_path` is defined and assigned in the path '/content/labels_my-project-name_2023-11-04-03-14-11.csv', indicating the path to the CSV file containing the dataset. Finally it uses `pd.read_csv()` from the `pandas` library to read the CSV file specified by `csv_path`. The resulting dataset is stored in the variable `data`. The argument `encoding='latin-1'` is used to handle any special characters in the CSV file.

A process to perform the label replacement and feature extraction from a dataset is done. The `map()` method is used on the 'label_name' column of the `data` DataFrame. This operation replaces the labels 'GUPY' with 1 and 'NONGUPY' with 0, effectively converting categorical labels into numerical representations.

A new variable `X` is created by extracting specific features from the DataFrame. The features selected are the bounding box coordinates: 'bbox_x', 'bbox_y', 'bbox_width', and 'bbox_height'. These features are organized into a NumPy array using `.values` for further us in the model.

The label ('label_name') is extracted from the dataset and assigns them to the variable `y`.Then, the dataset is split into training and testing sets using `train_test_split`. The parameter `test_size=0.2` specifies that 20% of the data will be used for testing, and `random_state=42` ensures reproducibility.

To finish with the preparation of the data to be used by the model. The features are normalized by using `StandardScaler`. The training set (`X_train`) is fitted and transformed, and the testing set (`X_test`) is transformed using the same scaler.

3) Model design: Once the dataset is ready to be used, the Perceptron model is created with a maximum of 1000 iterations and a random seed for reproducibility. The model is trained using the normalized training data (`X_train` and `y_train`). With the trained model, it makes predictions on the testing set (`X_test`) using the trained Perceptron model.

Finally, the accuracy of the model is computed by comparing the predicted labels (`y_pred`) with the actual labels in the testing set (`y_test`). The accuracy of the Perceptron model as a percentage is also printed.

4) Model evaluation: To evaluate the model, a random image from the dataset will be selected from a row index from the dataset (`data`) and retrieves the corresponding image name from the 'image_name' column. With the selected image, a path of the selected image by joining the image folder path (`carpeta_imagenes`) with the image name (`imagen_nombre`).

Then, the bounding box features is extracted of the randomly selected image from the dataset. The features include 'bbox_x', 'bbox_y', 'bbox_width', and 'bbox_height'. Also, the bounding box features are normalized of the randomly selected image using the previously defined scaler (`scaler`).

Finally, the trained Perceptron model (`perceptron`) is used to make a prediction based on the normalized bounding box features (`bounding_box_normalizada`).

The binary prediction is mapped to categorical labels to display the information about the prediction of the image. For that, the first thing to do is to check if the predicted value (`prediccion`) is equal to 1. If true, assigns the label "GUPY" to the variable `resultado`; otherwise, assigns "NONGUPY". This step is a mapping of the numerical prediction to human-readable categories.

Then, a message indicating the image name (`imagen_nombre`) and the predicted label (`resultado`) is printed. This provides information about the prediction for the displayed image.

To do so, it uses the `Image.open()` function from the `PIL` library to open the image located at `ruta_imagen`. The image is then displayed using `plt.imshow()` from `matplotlib`. The `plt.axis('off')` command is used to hide the axes, providing a cleaner image display.

B. Unsupervised Learning

1) Dataset: The dataset includes over 9,000 cat images [2]. For each cat image, there are annotations specifying the location of key points on the cat's head. These annotations consist of nine points corresponding to different facial features:

1) Left Eye
2) Right Eye
3) Mouth
4) Left Ear-1
5) Left Ear-2
6) Left Ear-3
7) Right Ear-1
8) Right Ear-2
9) Right Ear-3

Each annotation file is named after the corresponding image, with the addition of ".cat" at the end.For example, if there is an image named "cat001.jpg," the corresponding annotation file would be "cat001.jpg.cat." The annotation data in each file follow a specific sequence:

1) Number of Points: Specifies the number of points in the annotation. The default is 9 for the specified facial features.
2) Left Eye: Coordinates of the left eye.
3) Right Eye: Coordinates of the right eye.
4) Mouth: Coordinates of the mouth.
5) Left Ear-1, Left Ear-2, Left Ear-3: Coordinates of the three points defining the left ear.
6) Right Ear-1, Right Ear-2, Right Ear-3: Coordinates of the three points defining the right ear.

Therefore the code that wil be explained in the following sections reads images and their corresponding annotations from the CAT dataset, performs image preprocessing, applies KMeans clustering to group similar images together, and then visualizes the clustering results. The annotations are used to provide additional information about the cat images, and the clustering is done based on the visual similarity of the images. The resulting clusters could potentially reveal patterns or similarities in the facial features of the cats.

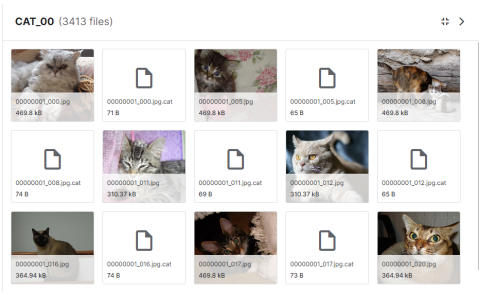

Fig. 2. Preview of the dataset made for the model.

2) Model pre-preparation: In this section, the process begins with data loading. Images and their corresponding annotations are retrieved from the specified directory and subjected to preprocessing. Following this, feature extraction takes place. The images undergo conversion into a `NumPy` array, subsequent reshaping into a 1D array, and standardization using `StandardScaler`.

To initiate this procedure, the first step involves importing essential libraries, such as `OpenCV (cv2)`, `NumPy (np)`, `Matplotlib (plt)`, and `StandardScaler from scikit-learn`. Subsequently, the environment variable `OMP_NUM_THREADS` is set to regulate the number of OpenMP threads utilized for parallel processing. Following that, the path to the directory containing images and annotations is specified.

The data loading function is pivotal in this process. For this purpose, a function named `load_data` is defined to retrieve images and annotations from a designated directory. The function iterates through files in the directory, reads JPEG images via OpenCV, resizes them to a uniform size (255x255), and stores them in the images list. Simultaneously, corresponding annotations from files with a ".cat" extension are loaded, converted to floats, and stored in the annotations list. The function concludes by returning the lists of images and annotations.

To load both images and annotations, the `load_data` function is invoked to retrieve data from the specified dataset path. For the subsequent data conversion and reshaping, the list of images is transformed into a `NumPy` array (`images_array`). This array is then reshaped into a 1D array (`images_reshaped`) to prepare the data for clustering.

In the final step, `StandardScaler` is applied to standardize the data (`images_scaled`), scaling it to have zero mean and unit variance. This standardization is a common preprocessing step in machine learning, ensuring that features are on a comparable scale.

3) Model design: Within this section, the KMeans clustering algorithm is applied to the standardized data with a specific number of clusters (`n_clusters`). Notably, the explicit setting of the number of initialization attempts (`n_init`) is highlighted.

To import KMeans from scikit-learn, use `from sklearn.cluster import KMeans`. Set the number of clusters by defining `n_clusters = 3 # You can adjust the number of clusters`. In this case, it's set to 3, but you can modify this number based on the desired clustering.

Instantiate KMeans using `kmeans = KMeans(n_clusters=n_clusters, random_state=0, n_init=10)`:

- `n_clusters=n_clusters`: Specifies the number of clusters for KMeans to form.
- `random_state=0`: Sets a random seed for reproducibility.
- `n_init=10`: Determines the number of times KMeans will run with different centroid seeds. The algorithm selects the best result based on inertia (sum of squared distances to the closest centroid).

To fit and predict with the model, use `clusters = kmeans.fit_predict(images_scaled)`. This fits the KMeans model to the standardized data (`images_scaled`) and predicts the cluster labels for each data point. The resulting `clusters` variable contains the cluster assignments for each data point.

The hyperparameters used are as follows:

1) `n_clusters`: Defines the number of clusters the algorithm should identify in the data. This parameter is crucial as it determines the fundamental structure of the clustering. The choice depends on the underlying data structure and the desired granularity of clustering.
2) `random_state`: Sets the random seed for reproducibility. Without it, different runs might produce different results. It ensures the same set of initial centroids is used in each run, making the results reproducible.
3) `n_init`: Specifies the number of times the algorithm will run with different centroid seeds. This helps mitigate sensitivity to the initial placement of centroids, and the final model is chosen based on the lowest inertia among the runs.

In summary, these hyperparameters play a critical role in determining the behavior and performance of the KMeans clustering algorithm, influencing the quality and stability of the identified clusters. Adjusting these parameters allows for customization based on the dataset characteristics and desired outcomes.

4) Model evaluation: Certainly! Here's the revised text formatted for LaTeX, and the content has been reorganized for coherence:

"'latex In this section, we provide a comprehensive breakdown of the visualization of the results, aiding in the understanding of distinct stages in the code, including data preparation, clustering model design, and result evaluation through visualization.

To create subplots, use `fig, axes = plt.subplots(1, n_clusters, figsize=(15, 3))`: This command creates a single row of subplots, with each subplot corresponding to a cluster. The size of the entire figure is set to (15, 3) inches. To iterate through clusters, employ `for cluster_id in range(n_clusters)`: This loop iterates through each cluster.

To extract cluster images, use `cluster_images = images_array[clusters == cluster_id]`: This line extracts the images from the original array that belong to the current cluster (`cluster_id`). To display images in subplots, employ `axes[cluster_id].imshow(cluster_images[0])`: This command displays the first image of the current cluster in the corresponding subplot. `axes[cluster_id].axis('off')`: Turn off axis labels for better visualization. `axes[cluster_id].set_title(f'Cluster cluster_id')`: Set the title of the subplot to indicate the cluster. To show the plot, use `plt.show()`: This command displays the entire figure with the clustered images.

This code was designed to visualize the results of the KMeans clustering by creating a row of subplots. Each subplot displays the first image from a different cluster, and the titles of the subplots indicate the cluster number. This visualization is instrumental in assessing the quality of the clustering and understanding the characteristics of each cluster.

C. Reinforcement Learning

This code implements a reinforcement learning agent using the Q-learning algorithm to train a robot in a simulated environment. Here's an overview of how it works:

1. Environment ("RobotEnv"): Defines an environment where a robot moves within a grid. The environment includes an observation space (environment state), an action space (possible robot movements), and defines interactions with the environment through the "step", "reset", and "render" methods.

2. Q-learning Agent ("ModifiedQLearningAgent"): Implements a reinforcement learning agent using the Q-learning algorithm. The agent has a Q-table that stores Q-values for state-action pairs and uses these values to make decisions.

"choose_action": Selects an action based on an exploration policy, choosing between random exploration and exploitation of existing Q-values.

"update_q_table": Updates Q-table values based on the agent's experiences in the environment. It uses the Q-value update equation to adjust values based on obtained rewards and future Q-value estimations.

"decay_exploration_prob": Gradually reduces the exploration rate with each episode so that the agent tends to exploit more than explore as it learns.

3. Training ("train_q_learning_agent"): Trains the agent in the defined environment. It iterates over a specific number of episodes where the agent interacts with the environment, chooses actions, updates the Q-table, and adjusts the exploration rate. It displays real-time visualizations of the environment for each episode.

4. Rendering ("render"): Visualizes the environment, displaying the robot, cat positions, and the midpoint, along with the associated reward for each cell. This function allows observing the agent's progress during training.

5. Performance Graphs: At the end of training, graphs are displayed representing the number of moves made and the total reward obtained in each episode.

In summary, this code integrates an environment, a reinforcement learning agent, and a training process for the agent to learn how to navigate in a reward-based environment and update its strategies using the Q-learning algorithm.

IV. Results

The Perceptron model trained for Gupy detection has yielded promising results, demonstrating an accuracy of 97.80%. This outstanding performance underlines the effectiveness of the model in the specific task of identifying the presence of Gupy in images.

A. Performance Metrics for supervised learning

The supervised learning model predicted whether or not a cat was Gupy, with an accuracy: 97.80%.

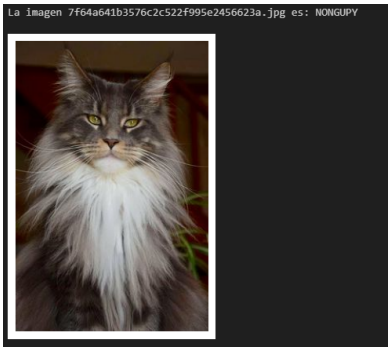

Fig. 3. NONGUPY was not predicted.

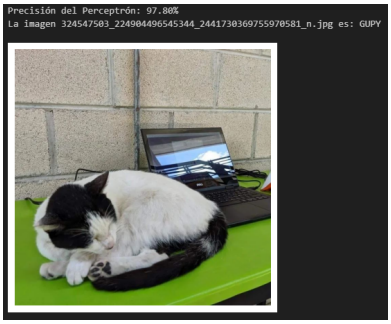

Fig. 4. Gupy was predicted.

1) Examples of Predictions: These examples illustrate the model's ability to distinguish between images that contain Gupy and those that do not, supporting the high accuracy reported.

The performance of the model is not limited to just the overall accuracy metric, as it has been extensively evaluated on various images, thus ensuring its robustness and ability to generalize to new data.

B. Performance Metrics for unsupervised learning

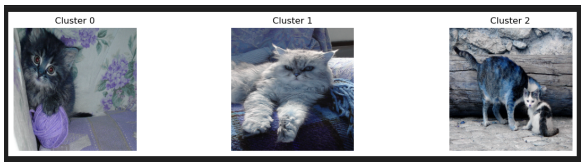

Fig. 5. clusters.

- Emotional State Analysis (Unsupervised Learning)

The second stage employed unsupervised learning to analyze Gupy's emotional state based on a dataset of cat images with key point annotations. The KMeans clustering algorithm was utilized to group similar images together, potentially revealing patterns in Gupy's facial expressions. The clustering results provided insights into Gupy's emotional states, including when he feels threatened or unwell.

Figure 2 illustrates the dataset used for emotional state analysis. Each cluster represents a group of images with visually similar facial features, aiding in understanding Gupy's emotional expressions.

C. Performance Metrics for reinforcement learning

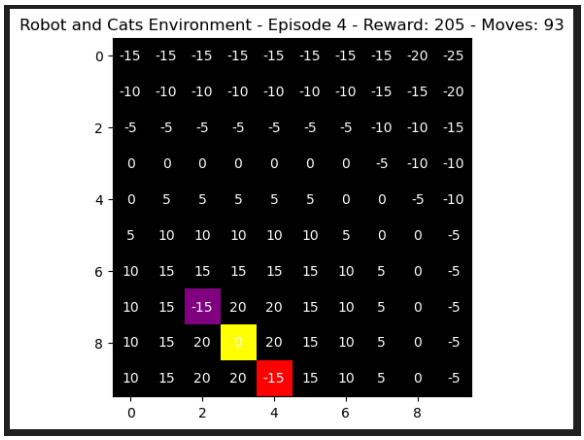

Fig. 6. robot and cats environment.

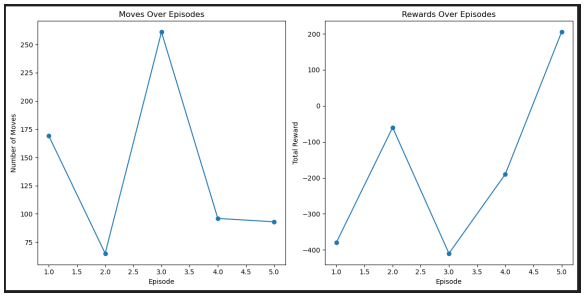

Fig. 7. moves and rewards over episodes.

- Reinforcement Learning for Threat Interception

The third and final stage implemented reinforcement learning to train a security platform to intercept threats when

Gupy is in an emotionally distressed state. The Q-learning algorithm was employed to guide the robot's movements in a simulated environment. The agent learned to choose actions that maximize cumulative rewards, showcasing adaptability in responding to potential threats.

Real-time visualizations during training episodes provided a dynamic view of the agent's interactions with the environment. The gradual decay of the exploration rate ensured that the agent transitioned from exploration to exploitation, optimizing its response strategy.

- Overall Project Outcome

The comprehensive code solution successfully addresses the safety concerns surrounding Gupy, offering a proactive approach to safeguarding the beloved mascot. The Gupy detection model, emotional state analysis, and reinforcement learning for threat interception collectively contribute to a robust system capable of ensuring Gupy's well-being in various situations. The project aligns with the goal of enhancing safety measures for Gupy and promoting a secure environment within the campus community.

V. Conclusion

In conclusion, this paper presents a comprehensive and innovative solution aimed at improving the security of Gupy, the beloved mascot of the Universidad Politécnica de Yucatán, through a multifaceted machine learning approach. The project leverages several techniques, including supervised learning, unsupervised learning, and reinforcement learning, to address different aspects of Gupy's well-being.

The initial focus on Gupy detection from images using supervised learning, specifically the Perceptron algorithm, yielded outstanding results with an accuracy of 97.80% Ṫhis success underlines the model's effectiveness in accurately identifying Gupy, showing its potential for implementation in real-world scenarios.

Beyond supervised learning, the second stage of the project delves into unsupervised learning to analyze Gupy's emotional state. By leveraging a dataset comprising more than 9,000 annotated cat images, the model aims to discern when Gupy is feeling threatened, scared or sick. The use of unsupervised learning, free of human bias, allows for a nuanced understanding of Gupy's emotions, contributing to a proactive approach to ensuring Gupy's well-being.

The final stage incorporates reinforcement learning to teach a security platform how to intercept incoming threats when Gupy is in a state of emotional distress. The model dynamically positions itself between Gupy and potential sources of distress, acting as a deterrent and intervening if necessary. This reinforcement learning application introduces a proactive security measure, which shows the adaptability and responsiveness of the model in dynamic environments. In summary, the project not only successfully addresses Gupy's immediate need for safety through supervised learning, but expands its scope to understand and respond to Gupy's emotions through unsupervised learning and reinforcement-based interventions. The positive results from each stage underline the potential of machine learning to improve safety measures, not only for pets but also in broader applications in various domains. This project sets a precedent for the integration of various machine learning techniques to ensure the well-being of pets and similar applications.

References

[1] Alpaydin, E. (2014). The Perceptron. In Introduction to Machine Learning (3rd ed., pp. 267-316). MIT Press

[2] A. Malik and B. Tuckfield, Applied Unsupervised Learning with r: Uncover Hidden Relationships and Patterns with K-Means Clustering, Hierarchical Clustering, and PCA. Birmingham: Packt, 2019.

[3] S. Ravichandiran, Hands-on Reinforcement Learning with Python: Master Reinforcement and Deep Reinforcement Learning Using Openai Gym and Tensorflow. Royaume-Uni: Packt Publishing, 2018.

Appendix A
GitHub Repository: Supervised Learning Labels and Jupyter Notebook

In the following GitHub repository you can find the .csv that contains the labels corresponding to the images of the dataset as well as the Jupyter Notebook in which the perceptron model was developed.

https://github.com/Maages09/Supervised_Learning.git

Appendix B
Supervised Learning Dataset

The dataset for the supervised solution can be found in the following link: Images of Gupy, and other cats.

At the same time, the dataset used for the unsupervised solution can be accessed through the following link:: Images of cats, with key points on the cat's head.

https://github.com/Maages09/SupervisedLearning.git

Appendix C
GitHub Repository: Unsupervised Learning Jupyter Notebook

In the following GitHub repository you can find the Jupyter Notebook in which the clustering model was developed.

https://github.com/Maages09/Unsupervised_Learning.git

Appendix D
GitHub Repository: Reinforcement Learning Jupyter Notebook

In the following GitHub repository you can find the Jupyter Notebook in which the Q-Learning model was developed.

https://github.com/Maages09/-Reinforcement_Learning.git