Final Proyect - Supervised Learning solution 1st Delivery

Freddy Alonzo Mondragon, Brenda Estefania Castillo Fernandez, Mariana Guadalupe Chi Centeno, Oswaldo Josue Gomez Gonzalez, Jhair Alejandro Cach Rosas, Denzel Guillermo Chuc Chuc
Computational Robotics Engineering, Universidad Politecnica de Yucatan, Victor Ortiz Santiago

***Abstract*—This project consists of an autonomous robotic camera using the Robot Operating System (ROS) for real-time vehicle identification. By incorporating a pre-trained neural network and Gazebo simulation, the system classifies various vehicles with high accuracy. This innovation has significant implications for industrial automation, marking a leap in autonomous robotics and AI. The project's success could demonstrate a scalable solution for intelligent transportation systems, promising advancements in traffic management and industrial efficiency.**

***Index Terms*—Article submission, IEEE, IEEEtran, journal, LaTeX, paper, template, typesetting.**

I. Introduction

In the realm of robotics and artificial intelligence, the ability of robots to interact with their environment and make informed decisions is essential. One exciting challenge in this field is to develop autonomous systems that can identify and respond to specific objects in their surroundings. This project focuses on the creation and development of a robotic system using the Robot Operating System (ROS) and simulation in Gazebo.

The main objective of this project is to simulate a robotic camera capable of detecting and labeling various types of vehicles in a controlled environment. To achieve this goal, a pre-trained neural network has been implemented. This neural network has been fed with a meticulously collected dataset, including images of different types of vehicles. These images have been processed and used to train the model, allowing the robot to accurately identify vehicles in real-time.

Developing this capability has practical applications in various industries, such as industrial automation, logistics, and autonomous transportation. Furthermore, this project provides an opportunity to explore the complexities of the interaction between robotics, machine learning, and control software, paving the way for future advancements in the field of autonomous robotics and artificial intelligence.

II. Development

A. Virtual environment

First, it created the virtual environment in Gazebo, a widely used simulator in robotics. In this simulated space, four types of vehicles were configured: a truck, a firetruck, an ambulance, and the robot Rover. These vehicles were meticulously modeled to faithfully represent their real-world counterparts, providing a realistic environment for testing and experiments.

Once the virtual environment was established, the implementation of the robotic camera system began. The Robot Operating System (ROS) framework was utilized to create and control the simulated camera. This camera was designed to capture images of the surrounding environment, which would later be used for vehicle detection and classification.

B. Data Processing

After setting up the virtual environment, the next step involved creating a node with the ability to capture images of everything the camera observed in the simulated environment. This node played a fundamental role by providing a constant flow of visual data, which was crucial for training and enhancing the vehicle recognition system.

During this capturing process, a substantial number of images were obtained for each type of vehicle present in the virtual environment: trucks, fire trucks, ambulances, and the robot Robert. To efficiently organize this diverse set of images, four separate folders were created, each corresponding to a type of vehicle in the simulated environment.

Subsequently, these images were divided into two main categories: ”training” and ”validation”. The division into these two folders allowed for a structured approach to training the neural network. The ”training” folder contained a large number of training images, essential for the model to learn the distinctive features of each type of vehicle. Meanwhile, the ”validation” folder housed a separate set of images used for validating and evaluating the model's performance during the training process. This separation facilitated continuous model checking and improvement, ensuring its accuracy and robustness.

C. *Supervised Learning*

Supervised learning, also known as supervised machine learning, is a subcategory of machine learning and artificial intelligence. It is defined by its use of labeled datasets to train algorithms that to classify data or predict outcomes accurately. As input data is fed into the model, it adjusts its

weights until the model has been fitted appropriately, which occurs as part of the cross validation process.

Supervised learning is when we teach or train the machine using data that is well-labelled. Which means some data is already tagged with the correct answer. After that, the machine is provided with a new set of examples(data) so that the supervised learning algorithm analyses the training data(set of training examples) and produces a correct outcome from labeled data.

How supervised learning work:
Supervised learning uses a training set to teach models to yield the desired output. This training dataset includes inputs and correct outputs, which allow the model to learn over time. The algorithm measures its accuracy through the loss function, adjusting until the error has been sufficiently minimized.

Supervised learning can be separated into two types of problems when data mining—classification and regression:

- **Classification:** Classification uses an algorithm to accurately assign test data into specific categories. It recognizes specific entities within the dataset and attempts to draw some conclusions on how those entities should be labeled or defined. Common classification algorithms are linear classifiers, support vector machines (SVM), decision trees, k-nearest neighbor, and random forest, which are described in more detail below.
- **Regression:** Regression is used to understand the relationship between dependent and independent variables. It is commonly used to make projections, such as for sales revenue for a given business. Linear regression, logistical regression, and polynomial regression are popular regression algorithms.

Supervised learning algorithms:
Various algorithms and computations techniques are used in supervised machine learning processes:

- **Neural networks:** Primarily leveraged for deep learning algorithms, neural networks process training data by mimicking the interconnectivity of the human brain through layers of nodes. Each node is made up of inputs, weights, a bias (or threshold), and an output. If that output value exceeds a given threshold, it “fires” or activates the node, passing data to the next layer in the network. Neural networks learn this mapping function through supervised learning, adjusting based on the loss function through the process of gradient descent. When the cost function is at or near zero, we can be confident in the model’s accuracy to yield the correct answer.
- **Naive bayes:** Naive Bayes is classification approach that adopts the principle of class conditional independence from the Bayes Theorem. This means that the presence of one feature does not impact the presence of another in the probability of a given outcome, and each predictor has an equal effect on that result. There are three types of Naïve Bayes classifiers: Multinomial Naïve Bayes, Bernoulli Naïve Bayes, and Gaussian Naïve Bayes. This technique is primarily used in text classification, spam identification, and recommendation systems.
- **Linear regression:** Linear regression is used to identify the relationship between a dependent variable and one or more independent variables and is typically leveraged to make predictions about future outcomes. When there is only one independent variable and one dependent variable, it is known as simple linear regression. As the number of independent variables increases, it is referred to as multiple linear regression. For each type of linear regression, it seeks to plot a line of best fit, which is calculated through the method of least squares. However, unlike other regression models, this line is straight when plotted on a graph.
- **Logistic regression:** While linear regression is leveraged when dependent variables are continuous, logistic regression is selected when the dependent variable is categorical, meaning they have binary outputs, such as ”true” and ”false” or ”yes” and ”no.” While both regression models seek to understand relationships between data inputs, logistic regression is mainly used to solve binary classification problems, such as spam identification.
- **Support vector machines (SVM):** A support vector machine is a popular supervised learning model developed by Vladimir Vapnik, used for both data classification and regression. That said, it is typically leveraged for classification problems, constructing a hyperplane where the distance between two classes of data points is at its maximum. This hyperplane is known as the decision boundary, separating the classes of data points (e.g., oranges vs. apples) on either side of the plane.
- **K-nearest neighbor:** K-nearest neighbor, also known as the KNN algorithm, is a non-parametric algorithm that classifies data points based on their proximity and association to other available data. This algorithm assumes that similar data points can be found near each other. As a result, it seeks to calculate the distance between data points, usually through Euclidean distance, and then it assigns a category based on the most frequent category or average. Its ease of use and low calculation time make it a preferred algorithm by data scientists, but as the test dataset grows, the processing time lengthens, making it less appealing for classification tasks. KNN is typically used for recommendation engines and image recognition.
- **Random forest:** Random forest is another flexible supervised machine learning algorithm used for both classification and regression purposes. The ”forest” references a collection of uncorrelated decision trees, which are then merged together to reduce variance and create more accurate data predictions.

Applications of Supervised learning:
Supervised learning can be used to solve a wide variety of problems, including:

- **Spam filtering:** Supervised learning algorithms can be trained to identify and classify spam emails based on their

content, helping users avoid unwanted messages.
- **Image classification:** Supervised learning can automatically classify images into different categories, such as animals, objects, or scenes, facilitating tasks like image search, content moderation, and image-based product recommendations.
- **Medical diagnosis:** Supervised learning can assist in medical diagnosis by analyzing patient data, such as medical images, test results, and patient history, to identify patterns that suggest specific diseases or conditions.
- **Fraud detection:** Supervised learning models can analyze financial transactions and identify patterns that indicate fraudulent activity, helping financial institutions prevent fraud and protect their customers.
- **Natural language processing (NLP):** Supervised learning plays a crucial role in NLP tasks, including sentiment analysis, machine translation, and text summarization, enabling machines to understand and process human language effectively.

Advantages of Supervised learning

1. Supervised learning allows collecting data and produces data output from previous experiences.
2. Helps to optimize performance criteria with the help of experience.
3. Supervised machine learning helps to solve various types of real-world computation problems.
4. It performs classification and regression tasks.
5. It allows estimating or mapping the result to a new sample.
6. We have complete control over choosing the number of classes we want in the training data.

Disadvantages of Supervised learning

1. Classifying big data can be challenging.
2. Training for supervised learning needs a lot of computation time. So, it requires a lot of time.
3. Supervised learning cannot handle all complex tasks in Machine Learning.
4. Computation time is vast for supervised learning.
5. It requires a labelled data set.
6. It requires a training process.

Steps Involved in Supervised Learning:

- First Determine the type of training dataset
- Collect/Gather the labelled training data.
- Split the training dataset into training dataset, test dataset, and validation dataset.
- Determine the input features of the training dataset, which should have enough knowledge so that the model can accurately predict the output.
- Determine the suitable algorithm for the model, such as support vector machine, decision tree, etc.
- Execute the algorithm on the training dataset. Sometimes we need validation sets as the control parameters, which are the subset of training datasets.
- Evaluate the accuracy of the model by providing the test set. If the model predicts the correct output, which means our model is accurate.

A. Neural Network

A neural network was created, whose main objective is to classify images, as mentioned above, two folders were created that include the images to be classified. One folder was used for training and the other for validation.
The neural network started by calling all the necessary libraries, in this case we worked with Tensorflow, however, the following specific libraries were also used:

”from keras.preprocessing.image import ImageDataGenerator”

This library is used for image processing, it allows to generate batches of augmented training data from an existing set of images. This is to improve the performance of learning models. The next library is ”from tensorflow.keras.models import Sequential” which is used to create neural network models sequentially, i.e., the model is created by adding one layer after another in order. We also used the library ”from tensorflow.keras.layers import Convolution2D, MaxPooling2D” these are the layers where the convolutions and maxpooling are going to be done. Then we used ”from tensorflow.keras.layers import Dropout, Flatten, Dense” in the dropout layer, it applies a regularization to the neural network by randomly turning off a percentage of the units during training to prevent overfitting. The flatten layer serves to make the transition from convolutional layers to fully connected layers in a neural network. In the Dense layer each of the neurons it has is connected to all the neurons in the previous layer. It is used to finalize the network. The next library is ”from tensorflow.keras.optimizers.import Adam” which is used to train the neural network, adjusting weights during training to minimize the loss function.

```
from keras.preprocessing.image import
    ImageDataGenerator
from tensorflow.keras.models import
    Sequential
from tensorflow.keras.layers import
    Convolution2D, MaxPooling2D
from tensorflow.keras.layers import
    Dropout, Flatten, Dense
from tensorflow.keras.optimizers import
    Adam

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

Subsequently, the dataset file was loaded. Training and validation variables were created for the directory of images, and the total number of training and validation images was also specified. Then, the parameters were specified in which it was defined that the model will have 20 epochs, which means that the model will see the whole training dataset during the training process 20 times. The images were also resized

to 150x150 pixels using length, height. A batch size of 32 was used, i.e. 32 data samples to be used in each iteration of the training. Then the steps were created, which are the number of times the information will be processed in each of the epochs. To then define the validation steps, which at the end of each of the epochs will run 200 sets of data to show that the algorithm is learning. Then the convolution filters were generated, the first convolution will have a filter of 32 and the second convolution will have a filter of 64, i.e. the image will have a depth of 32 in the first convolution and 64 in the second convolution. The filter sizes were also defined, for filter 1 the size is (3, 3) and for filter two, the size is (2, 2). Then the filter size to be used in maxpooling (2, 2) was defined. The number of classes to be classified and the lr (learning rate) were defined to know how big are the adjustments that the neural network will make to approach an optimal solution.

```
# Training and validation directories
data_entrenamiento =
    '/content/drive/MyDrive/
images_dataset/training'
data_validacion =
    '/content/drive/MyDrive/
images_dataset/validation'

total_training_images = 2130 # Replaces
    with the total number of training
    images
total_imagenes_validacion = 1287 #
    Replaces with the total number of
    validation images

# Parameters
epocas = 20
longitud, altura = 150, 150
batch_size = 32
pasos = total_imagenes_entrenamiento //
    batch_size
validation_steps =
    total_imagenes_validacion //
    batch_size
filtrosConv1 = 32
filtrosConv2 = 64
tamano_filtro1 = (3, 3)
tamano_filtro2 = (2, 2)
tamano_pool = (2, 2)
clases = 4
lr = 0.0005
```

Then we proceeded with the image preprocessing, to then give it to the neural network. For this, a generator is created that specifies how we are going to process the information. The generator was told that the images will be rescaled to 1. / 255, i.e. each of the pixels has a range of 1 to 255, and was rescaled to all pixel values are from 0 to 1 to make the training more efficient. Then the images are tilted so that the algorithm learns that the transports are not always going to be seen standing still, in the same way the zoom range is created so that some images are zoomed and others are not, and so the algorithm learns that a complete transport will not always appear, sometimes sections will appear, finally it takes an image and inverts it, so that the network learns directionality.

Then the same was done for the validation dataset, where only the images are rescaled, since it is not necessary to flip, zoom or tilt them.

Afterwards, a variable was created, where it generates the images for the training, what it does is to enter the directory that was previously specified, it enters each of the folders that it has and preprocesses each of the images that are found, to these images everything that we had already declared at the beginning is changed and it is added that the class mode is categorical format. This process is done for both training and validation images.

```
# Image preprocessing
entrenamiento_datagen =
    ImageDataGenerator(
     rescale=1. / 255,
     shear_range=0.2,
     zoom_range=0.2,
     horizontal_flip=True)

test_datagen =
    ImageDataGenerator(rescale=1. / 255)

entrenamiento_generador =
    entrenamiento_datagen.
flow_from_directory(
     data_entrenamiento,
     target_size=(altura, longitud),
     batch_size=batch_size,
     class_mode='categorical')

validacion_generador =
    test_datagen.flow_from_directory(
     data_validacion,
     target_size=(altura, longitud),
     batch_size=batch_size,
     class_mode='categorical')
```

The next step was to create the convolutional network, this network is on a variable called cnn and is declared to be Sequential, that is, the network will have several layers that will be stacked sequentially. Subsequently, the first layer is created, where it was told that it will be a convolution that will have the number of filters, the size of the filter, which is what will make the filters to the corners, then it was specified the size of the images that will be delivered and finally it was told that the activation function that is used is relu, then a MaxPooling layer was added in which the size was also specified, after this the following convolutional layer was added, with all the other elements of the first layer with a slight change in the size of the filter, since for this layer the filter2 was used, followed by a MaxPooling layer. Now we start the classification, where we change the images

with a depth to a single dimension but with all the necessary information collected, after flattening the information, we added a layer where all the neurons are connected with the neurons of the last layer, also it is stated that during the training each step of the network, turn off 50% of the neurons, to avoid over adjusting. The last layer will say the percentage of probability that an image is a certain transport, i.e. if it says that there is 80% probability that it is a fire truck, 5% that it is a Rover, 5% that it is an ambulance and 10% that it is a truck, then it is assumed that the value that has the highest percentage is the correct classification.

Then the model was compiled, during the training its loss function is categorical_crossentropy, then the optimizer that was used is Adam, where the learning rate that was previously stated was specified, the metric with which it will be optimized is the accuracy. After compiling the model, it is finally trained, the training was performed using a data generator and using the epochs already declared previously.

Once the training is completed, the model and its weights are saved in files so that they can be reused in the future without the need to retrain from scratch.

```
# Create CNN model
cnn = Sequential()
cnn.add(Convolution2D(filtrosConv1,
    tamano_filtro1, padding="same",
    input_shape=(longitud, altura, 3),
    activation='relu'))
cnn.add(MaxPooling2D(pool_size=
tamano_pool))
cnn.add(Convolution2D(filtrosConv2,
    tamano_filtro2, padding="same",
    activation='relu'))
cnn.add(MaxPooling2D(pool_size=
tamano_pool))
cnn.add(Flatten())
cnn.add(Dense(256, activation='relu'))
cnn.add(Dropout(0.5))
cnn.add(Dense(clases,
    activation='softmax'))

# Compile the model
cnn.compile(loss=
'categorical_crossentropy',
            optimizer=Adam(lr=lr),
            metrics=['accuracy'])

# Train the model
cnn.fit(
    entrenamiento_generador,
    steps_per_epoch=pasos,
    epochs=epocas,
    validation_data=validacion_generador,
    validation_steps=validation_steps)

cnn.save('/content/modelo/modelo.h5')
cnn.save_weights('/content/modelo/
pesos.h5')
```

In this part of the code the first thing to do is import the essential libraries: ”numpy” for mathematical operations and functions, ”keras.preprocessing.image” for loading images and converting to arrays, which are the appropriate input format for CNN. Additionally, ”keras.models” is used to import the pre-existing model structure.

The process begins by setting the expected input dimensions for the images (150x150 pixels) and specifying the paths to the model architecture and their trained weights. Subsequently, the CNN model is retrieved from the designated ”model” file, and its corresponding weights are loaded from ”model_weights”, ensuring that the model has the necessary parameters to make predictions.

The ”predict” function summarizes the steps required to process an input image: loading the image at the specified dimensions, converting it to an array, and then expanding its dimensions to fit the model’s expected input. form, which is typically a batch of images, even if only one image is processed. After preprocessing, the image is fed into the CNN using the “predict” function, resulting in a probability distribution between the possible classes. The ‘np.argmax‘ function is then applied to this distribution to determine the most likely class for the given image, generating an impression of the corresponding vehicle type, such as ”pred: Ambulance” for index 0.

The code concludes with a practical example, invoking the ”predict” function with an image path that presumably leads to the image of a fire engine. The predicted output is the printout of the model prediction for the vehicle type depicted in the image, along with the output of the predicted class index.

```
import numpy as np
from keras.preprocessing.image import
    load_img, img_to_array
from keras.models import load_model

longitud, altura = 150, 150
modelo = '/content/modelo/modelo.h5'
pesos_modelo = '/content/modelo/pesos.h5'
cnn = load_model(modelo)
cnn.load_weights(pesos_modelo)

def predict(file):
  x = load_img(file,
      target_size=(longitud, altura))
  x = img_to_array(x)
  x = np.expand_dims(x, axis=0)
  array = cnn.predict(x)
  result = array[0]
  answer = np.argmax(result)
  if answer == 0:
    print("pred: Ambulancia")
  elif answer == 1:
```

```
    print("pred: Bus")
  elif answer == 2:
    print("pred: Carro de Bomberos")
  elif answer == 3:
    print("pred: Rover")

  return answer

predict('/content/drive/MyDrive/
images_dataset/validation/carro_bomberos/
image_2871.png')
```

B. Implementation

Finally, once the model was created and trained, it was imported into another node. This integration allowed the robotic camera to fulfill its role of vehicle identification and classification seamlessly. The trained model, fine-tuned through extensive iterations, was now capable of processing the real-time visual data captured by the camera in the virtual environment. Upon implementation, the camera node utilized the imported model to identify and classify vehicles effectively. As the camera observed the surroundings, it processed the incoming images and accurately determined the type of vehicle present—whether it was a truck, a fire truck, an ambulance, or the robot Robert.

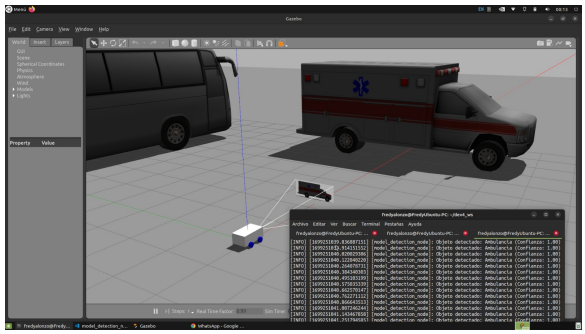

Fig. 1. Function Evidence

D. *Unsupervised Learning*

Unsupervised learning in artificial intelligence is a type of machine learning that learns from data without human supervision. Unlike supervised learning, unsupervised machine learning models are given unlabeled data and allowed to discover patterns and insights without any explicit guidance or instruction.

Whether you realize it or not, artificial intelligence and machine learning are impacting every aspect of daily life, helping to turn data into insights that can improve efficiencies, reduce costs, and better inform decision-making. Today, businesses are using machine learning algorithms to help power personalized recommendations, real-time translations, or even automatically generate text, images, and other types of content.

How does unsupervised learning work?

As the name suggests, unsupervised learning uses self-learning algorithms—they learn without any labels or prior training. Instead, the model is given raw, unlabeled data and has to infer its own rules and structure the information based on similarities, differences, and patterns without explicit instructions on how to work with each piece of data.

Unsupervised learning algorithms are better suited for more complex processing tasks, such as organizing large datasets into clusters. They are useful for identifying previously undetected patterns in data and can help identify features useful for categorizing data.

Imagine that you have a large dataset about weather. An unsupervised learning algorithm will go through the data and identify patterns in the data points. For instance, it might group data by temperature or similar weather patterns.

While the algorithm itself does not understand these patterns based on any previous information you provided, you can then go through the data groupings and attempt to classify them based on your understanding of the dataset. For instance, you might recognize that the different temperature groups represent all four seasons or that the weather patterns are separated into different types of weather, such as rain, sleet, or snow.

Fig. 2. An ML model clustering similar data points.

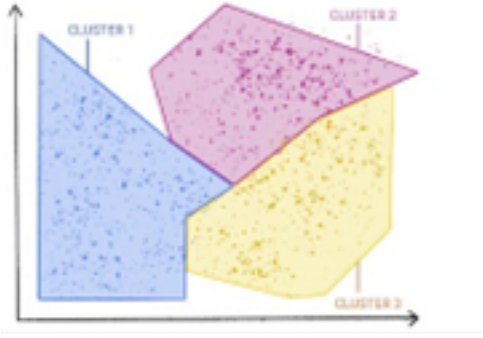

Fig. 3. Groups of clusters with natural demarcations

Unsupervised machine learning methods
In general, there are three types of unsupervised learning tasks: clustering, association rules, and dimensionality reduction. Below we'll delve a little deeper into each type of unsupervised learning technique.

– **Clustering:** Clustering is a technique for exploring raw, unlabeled data and breaking it down into groups (or clusters) based on similarities or differences. It is used in a variety of applications, including customer segmentation, fraud detection, and image analysis. Clustering algorithms split data into natural groups by finding similar structures or patterns in uncategorized data.
  Clustering is one of the most popular unsupervised machine learning approaches. There are several types of unsupervised learning algorithms that are used for clustering, which include exclusive, overlapping, hierarchical, and probabilistic.

  • Exclusive clustering: Data is grouped in a way where a single data point can only exist in one cluster. This is also referred to as "hard" clustering. A common example of exclusive clustering is the K-means clustering algorithm, which partitions data points into a user-defined number K of clusters.
  • Overlapping clustering: Data is grouped in a way where a single data point can exist in two or more clusters with different degrees of membership. This is also referred to as "soft" clustering.
  • Hierarchical clustering: Data is divided into distinct clusters based on similarities, which are then repeatedly merged and organized based on their hierarchical relationships. There are two main types of hierarchical clustering: agglomerative and divisive clustering. This method is also referred to as HAC—hierarchical cluster analysis.
  • Probabilistic clustering: Data is grouped into clusters based on the probability of each data point belonging to each cluster. This approach differs from the other methods, which group data points based on their similarities to others in a cluster.

– **Association:** Association rule mining is a rule-based approach to reveal interesting relationships between data points in large datasets. Unsupervised learning algorithms search for frequent if-then associations—also called rules—to discover correlations and co-occurrences within the data and the different connections between data objects.
  It is most commonly used to analyze retail baskets or transactional datasets to represent how often certain items are purchased together. These algorithms uncover customer purchasing patterns and previously hidden relationships between products that help inform recommendation engines or other cross-selling opportunities. You might be most familiar with these rules from the "Frequently bought together" and "People who bought this item also bought" sections on your favorite online retail shop.
  Association rules are also often used to organize medical datasets for clinical diagnoses. Using unsupervised machine learning and association rules can help doctors identify the probability of a specific diagnosis by comparing relationships between symptoms from past patient cases.
  Typically, Apriori algorithms are the most widely used for association rule learning to identify related collections of items or sets of items. However, other types are used, such as Eclat and FP-growth algorithms.

– **Dimensionality reduction:** Dimensionality reduction is an unsupervised learning technique that reduces the number of features, or dimensions, in a dataset. More data is generally better for machine learning, but it can also make it more challenging to visualize the data.
  Dimensionality reduction extracts important features from the dataset, reducing the number of irrelevant or random features present. This method uses principle component analysis (PCA) and singular value decomposition (SVD) algorithms to reduce the number of data inputs without compromising the integrity of the properties in the original data.

Unsupervised Learning Applications:
Most executives would have no problem identifying use cases for supervised machine learning tasks; the same cannot be said for unsupervised learning.
One reason this may be is down to the simple nature of risk. Unsupervised learning introduces much more risk than unsupervised learning since there's no clear way to measure results against ground truth in an offline manner, and it may be too risky to conduct an online evaluation.
Nonetheless, there are several valuable unsupervised learning use cases at the enterprise level. Beyond using unsupervised techniques to explore data, some common use cases in the real-world include:

• **Natural language processing (NLP):** Google News is known to leverage unsupervised learning to categorize articles based on the same story from various news outlets. For instance, the results of the football transfer window can all be categorized under football.
• **Image and video analysis:** Visual Perception tasks such as object recognition leverage unsupervised learning.
• **Anomaly detection:** Unsupervised learning is used to identify data points, events, and/or observations that deviate from a dataset's normal behavior.
• **Customer segmentation:** Interesting buyer persona profiles can be created using unsupervised learning. This helps businesses to understand their customers' common traits and purchasing habits, thus, enabling them to align their products more accordingly.
• **Recommendation Engines:** Past purchase behavior

coupled with unsupervised learning can be used to help businesses discover data trends that they could use to develop effective cross-selling strategies.

A. Neural Network

To initiate the unsupervised phase, you first have to load the dataset and bring in all the necessary libraries. Afterward, the Image Directory was established along with parameters such as the number of epochs, the length and height of the images, and the number of classes for the algorithm. Following this, the glob function was employed to gather lists of paths for each image in the directory. Subsequently, all the images were loaded, converted into matrices, and normalized.

Once the data had been normalized, the K-Means algorithm was applied to the normalized features of the images, and the outcome was stored in a file.

After that step, t-SNE was utilized to reduce the dimensionality of the normalized data to 2 dimensions and display the results on a graph, each class being represented by a distinctive color. In this context, yellow signifies the Rover, purple corresponds to the ambulance, blue denotes the bus, and green represents the firetruck. In the final steps, the K-Means model was loaded. Then, a new image was loaded and preprocessed for predicting its cluster using the loaded K-Means model. Lastly, the new image is presented visually, and the prediction result is printed, specifying the cluster to which the new image belongs.

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

import os
from glob import glob
import numpy as np
import joblib
from keras.preprocessing.image import
    load_img, img_to_array
from sklearn.cluster import KMeans
from sklearn.preprocessing import
    StandardScaler
from sklearn.manifold import TSNE
from sklearn.metrics import
    silhouette_score
import matplotlib.pyplot as plt

#Image directory
data_total =
    '/content/drive/MyDrive/all_images'

# Parameters
epocas = 100
longitud, altura = 150, 150
clases = 4

# Get list of paths of all images
lista_imagenes =
    glob(os.path.join(data_total,
    '*.png'))

# Load all images and convert them to
    numpy arrays
X = [img_to_array(load_img(img_path,
    target_size=(altura, longitud))) for
    img_path in lista_imagenes]
X = np.array(X)

# Make sure images have the same shape
X = X.reshape(-1, altura * longitud * 3)

# Data normalization
scaler = StandardScaler()
X_normalized = scaler.fit_transform(X)

# Apply K-Means directly to normalized
    features
kmeans = KMeans(n_clusters=clases,
    random_state=42, n_init=epocas)
labels = kmeans.fit_predict(X_normalized)

# Save the K-Means model
joblib.dump(kmeans, '/content/modelo/
kmeans_model_cnn_features.joblib')

# Cluster name mapping
mapeo_nombres = {
    0: "Ambulancia",
    1: "Bus",
    2: "Carro_de_bomberos",
    3: "Rover"
}

# Assign names to clusters
nombres_clusters = [mapeo_nombres[label]
    for label in labels]

# Visualization of results with t-SNE
tsne = TSNE(n_components=2,
    random_state=42)
X_tsne = tsne.fit_transform(X_normalized)

plt.scatter(X_tsne[:, 0], X_tsne[:, 1],
    c=labels, cmap='viridis')

# Annotate the points with the names of
    the clusters
for i, nombre in
    enumerate(nombres_clusters):
    plt.annotate(nombre, (X_tsne[i, 0],
        X_tsne[i, 1]))

plt.title('Visualizacion de Clusteres
    con t-SNE')
plt.show()
```

```
# Silhouette coefficient
silhouette_avg = silhouette_score(X_normalized, labels)
print(f'Coeficiente de Silueta: {silhouette_avg}')
```

```
import numpy as np
import joblib
from keras.preprocessing.image import load_img, img_to_array
import matplotlib.pyplot as plt

# Image path you want to predict
ruta_imagen_nueva = '/content/drive/MyDrive/images_dataset/training/ambulancia/image_424.png'

# Load the trained K-Means model
modelo_kmeans = joblib.load('/content/modelo/kmeans_model_cnn_features.joblib')

# Load the new image and adjust its size
img_nueva = load_img(ruta_imagen_nueva, target_size=(150, 150))
img_array_nueva = img_to_array(img_nueva)
img_array_nueva_flattened = img_array_nueva.reshape(1, -1)

# Make the prediction with the K-Means model
label_predicho = modelo_kmeans.predict(img_array_nueva_flattened)

# Cluster name mapping
mapeo_nombres = {
    0: "Ambulancia",
    1: "Bus",
    2: "Carro_de_bomberos",
    3: "Rover"
}

# Get the predicted cluster name
nombre_predicho = mapeo_nombres[label_predicho[0]]

# View the image
plt.imshow(img_nueva)
plt.title(f'Prediccion: {nombre_predicho}')
plt.show()

print(f'La nueva imagen pertenece al cluster: {nombre_predicho}')
```

B. Implementation

This implementation showcases the practical application of unsupervised learning in vehicle classification using K-Means clustering. It effectively combines data preparation, model training, and visualization techniques. The use of t-SNE for visualizing clustering outcomes, along with the computation of the silhouette coefficient, provides a clear and quantitative assessment of the model's performance. The model's capability to classify new data, demonstrated in the prediction phase, highlights its practicality for real-world scenarios. Overall, this approach is a robust example of employing machine learning techniques for meaningful and efficient image classification, particularly in vehicle identification.

Fig. 4. Function Evidence

E. *Reinforcement Learning*

Reinforcement learning (RL), a subset of machine learning, revolves around making decisions that maximize rewards in specific situations. It is widely applied in software and machines to determine optimal behavior or paths. Unlike supervised learning, where the model is trained with labeled data, reinforcement learning lacks predefined answers. Instead, an agent learns through trial and error, gaining experience and adapting its actions to perform tasks.

Reinforcement Learning Overview: Reinforcement Learning (RL) is essentially the science of decision-making, aiming to learn optimal behavior in an environment to achieve maximum reward. It relies on accumulating data through a trial-and-error approach, distinguishing itself from supervised and unsupervised learning paradigms.

Algorithms in reinforcement learning learn from outcomes, receiving feedback after each action to evaluate correctness. This method is effective for automated systems making numerous decisions without human guidance, embodying an autonomous, self-teaching system that refines itself through experience.

Illustratively, consider an agent navigating obstacles to reach a reward. In this scenario, the agent learns by exploring

various paths, optimizing its route to minimize hurdles and maximize rewards. Each correct step adds to its reward, while incorrect ones deduct from it, with the cumulative reward calculated upon reaching the final goal.

Key Aspects of Reinforcement Learning:

- **Input:** An initial state from which the model commences.
- **Output:** Multiple possible outputs, representing various solutions to a given problem.
- **Training:** Based on input, the model returns a state, and the user rewards or punishes the model based on its output.
- **Continual learning:** The model keeps learning from its experiences.
- **Decision criteria:** The best solution is determined by maximizing rewards.

Types of Reinforcement:

- **Positive Reinforcement:** Strengthens behavior by increasing the frequency of desired actions.
- **Negative Reinforcement:** Reinforces behavior by removing or avoiding negative conditions.

Advantages of Reinforcement Learning:

1. Reinforcement learning can be used to solve very complex problems that cannot be solved by conventional techniques.
2. The model can correct the errors that occurred during the training process.
3. In RL, training data is obtained via the direct interaction of the agent with the environment.
4. Reinforcement learning can handle environments that are non-deterministic, meaning that the outcomes of actions are not always predictable. This is useful in real-world applications where the environment may change over time or is uncertain.
5. Reinforcement learning can be used to solve a wide range of problems, including those that involve decision making, control, and optimization.
6. Reinforcement learning is a flexible approach that can be combined with other machine learning techniques, such as deep learning, to improve performance.

Disadvantages of Reinforcement Learning:

1. Reinforcement learning is not preferable to use for solving simple problems.
2. Reinforcement learning needs a lot of data and a lot of computation.
3. Reinforcement learning is highly dependent on the quality of the reward function. If the reward function is poorly designed, the agent may not learn the desired behavior.
4. Reinforcement learning can be difficult to debug and interpret. It is not always clear why the agent is behaving in a certain way, which can make it difficult to diagnose and fix problems.

A. Neural Network

For reinforcement learning, similar to unsupervised learning, the process began by importing all the necessary libraries and loading images from the dataset. After that, a Deep Q-Network model was defined using Keras to enable the model to learn decision-making based on the current state. Subsequently, the environment was created, always used as training to simulate the interaction between the DQN model and the environment. This environment initializes with the dataset, passed as a parameter, and sets the current index to 0. It then resets the environment, setting the current index to 0 and returning the image of the new state. It takes an action as a parameter, calculates the ”reward” with the current action, increments the index to get the next image in the next step, and returns the next image and the reward.

Lastly, it implements the logic to calculate the reward depending on the action and truth, comparing the action with the actual category of the image in the dataset. If the action is correct, it returns 1; if incorrect, it returns -1. Then, parameters and configurations were set, specifying the number of possible actions, the state's shape, batch size, the number of training episodes, and the maximum number of steps per episode. After creating the environment and the DQN model, a training loop with episodes was implemented, where each episode represents an attempt to learn. It begins by resetting the environment to an initial state. Within each episode, the model makes decisions over multiple steps, choosing actions influenced by its own predictions. The model interacts with the environment, receiving the next state and a reward associated with the taken action. Subsequently, the model updates based on the acquired experience, adjusting its weights to improve decision-making in the future. This process repeats for several episodes, and the model's performance is evaluated based on the total reward obtained in each episode. If the reward indicates a completely correct or incorrect prediction, a message is printed, and the episode loop is interrupted. Ultimately, the model is trained to enhance decision-making through continuous interaction with the simulated environment. Finally, the trained model and its weights are saved in a file.

Lastly, the model is loaded from the h5 format file. Then, a function called ‘predict_image‘ is defined, taking the path of an image as input, loading and preprocessing it, and making a prediction using the loaded model. The function returns the image and the predicted category. Subsequently, the path of a new image (‘new_image_path‘) is specified to be used for prediction using the previously mentioned function. The prediction is visually displayed, including the image and the predicted category, using the Matplotlib library.

```
import os
import shutil
import numpy as np
import tensorflow as tf
from tensorflow.keras.models import
    Sequential
from tensorflow.keras.layers import
    Dense, Flatten
```

```
from tensorflow.keras.optimizers import Adam
from collections import deque
from google.colab import drive
from tensorflow.keras.preprocessing.image import ImageDataGenerator, load_img, img_to_array

drive.mount('/content/drive')

# Paths to category folders
categorias = ['ambulancia', 'bus', 'carro_bomberos', 'rover']
dataset = []

for categoria in categorias:
    categoria_path = f'/content/drive/MyDrive/images_dataset/training/{categoria}'

    for imagen_file in os.listdir(categoria_path):
        imagen_path = os.path.join(categoria_path, imagen_file)
        imagen = load_img(imagen_path, target_size=(150, 150))
        imagen_array = img_to_array(imagen) / 255.0  # Normalizar entre 0 y 1
        dataset.append({'image': imagen_array, 'category': categorias.index(categoria)})

# Basic implementation of the DQN model
def build_dqn_model(input_shape, num_actions):
    model = Sequential()
    model.add(Flatten(input_shape=input_shape))
    model.add(Dense(64, activation='relu'))
    model.add(Dense(32, activation='relu'))
    model.add(Dense(num_actions, activation='linear'))
    model.compile(optimizer=Adam(learning_rate=0.001), loss='mse')
    return model

#Basic environment implementation
class SimpleEnvironment:
    def __init__(self, dataset):
        self.dataset = dataset
        self.current_index = 0

    def reset(self):
        self.current_index = 0
        return self.dataset[self.current_index]['image']

    def step(self, action):
        # Get reward based on action take
        reward = self.calculate_reward(action)

        # Update the index to get the next image in the next step
        self.current_index += 1

        # Get the next image
        next_state = self.dataset[self.current_index]['image']

        return next_state, reward

    def calculate_reward(self, action):
        # Implement logic to calculate reward based on action and truth
        # You can compare the action with the actual category of the current image
        return 1 if action == self.dataset[self.current_index]['category'] else -1

# Parameters
num_actions = 4  # Number of vehicle categories
state_shape = (150, 150, 3)  # Adjust according to the dimensions of your images
batch_size = 32
num_episodes = 100
max_steps_per_episode = 100

# Create a DQN environment and model
env = SimpleEnvironment(dataset)  # Adjust according to your data
model = build_dqn_model(input_shape=state_shape, num_actions=num_actions)

# Training loop
for episode in range(num_episodes):
    state = env.reset()
    state = np.reshape(state, (1,) + state_shape)
    total_reward = 0

    for step in range(max_steps_per_episode):
```

```
        # Choose a stock with the DQN model
        action = np.argmax(model.predict(state))

        # Take action in the environment and get the next status and reward
        next_state, reward = env.step(action)
        next_state = np.reshape(next_state, (1,) + state_shape)

        # Update the DQN model with experience
        target = reward + 0.95 * np.amax(model.predict(next_state)[0])
        target_f = model.predict(state)
        target_f[0][action] = target
        model.fit(state, target_f, epochs=1, verbose=0)

        state = next_state
        total_reward += reward

        if total_reward == 1 or total_reward == -1:
            # The reward indicates that the prediction was correct or incorrect
            print(f"Episodio {episode + 1}, Paso {step + 1}, Recompensa: {total_reward}")
            break

# Save the model
model.save('/content/modelo/modelo_rl.h5')
model.save_weights('/content/modelo/pesos_rl.h5')
```

```
import numpy as np
from tensorflow.keras.models import load_model
from tensorflow.keras.preprocessing.image import load_img, img_to_array
import matplotlib.pyplot as plt

# Load model from .h5 file
modelo_rl = load_model('/content/modelo/modelo_rl.h5')

# Function to preprocess an image and make a prediction
def predecir_imagen(ruta_imagen):
    imagen = load_img(ruta_imagen, target_size=(150, 150))
    imagen_array = img_to_array(imagen) / 255.0  # Normalize between 0 and 1
    imagen_array = np.reshape(imagen_array, (1, 150, 150, 3))  # Adjust dimensions
    prediccion = modelo_rl.predict(imagen_array)
    categoria_predicha = np.argmax(prediccion)
    return imagen, categoria_predicha

# Path of a new image to predict
ruta_nueva_imagen = '/content/drive/MyDrive/images_dataset/training/ambulancia/image_424.png'

# Get the prediction
imagen, categoria_predicha = predecir_imagen(ruta_nueva_imagen)

# Show image
plt.imshow(imagen)
plt.title(f"La imagen pertenece a la categoria: {categoria_real}")
plt.axis('off')
plt.show()
```

B. Implementation

The implementation employs reinforcement learning for image classification, utilizing a Deep Q-Network (DQN) model. It learns from environmental feedback, adapting through a structured training loop. This approach showcases DQN's capability in practical image classification, illustrating its effectiveness in real-world scenarios. The well-organized prediction mechanism further emphasizes the model's adaptability and learning efficiency.

Fig. 5. Function Evidence

III. Conclusion

In conclusion, this project has successfully developed a robotic camera system that excels in detecting and identifying different vehicle types. By integrating the Robot Operating System (ROS) with a pre-trained neural network, the camera demonstrates remarkable real-time vehicle recognition capabilities. Its accuracy and efficiency in distinguishing various vehicles mark a significant advancement in autonomous robotics and computer vision.

The system's precision in vehicle categorization has broad applications, from improving traffic management to streamlining industrial logistics. This successful implementation illustrates the transformative potential of this technology in transportation and automation. It paves the way for safer, more efficient, and smarter robotic systems, highlighting a pivotal evolution in these industries.

References

[1] "What is Supervised Learning? — IBM". IBM in Deutschland, Österreich und der Schweiz — IBM. Available: https://www.ibm.com/topics/supervised-learning#:~:text=the%20next%20step-,What%20is%20supervised%20learning?,data%20or%20predict%20outcomes%20accurately.

[2] "Applications of Machine Learning - GeeksforGeeks". GeeksforGeeks. Available: https://www.geeksforgeeks.org/machine-learning-introduction/?ref=ml_lbp

[3] "Supervised Machine learning - Javatpoint". www.javatpoint.com. Available: https://www.javatpoint.com/supervised-machine-learning

[4] "What is unsupervised learning? — Google Cloud," Google Cloud, 2023. Available: https://cloud.google.com/discover/what-is-unsupervised-learning#:~:text=Unsupervised%20learning%20in%20artificial%20intelligence,any%20explicit%20guidance%20or%20instruction.

[5] K. Pykes, "Introduction to Unsupervised Learning," Datacamp.com, Mar. 31, 2023. Available: https://www.datacamp.com/blog/introduction-to-unsupervised-learning.

[6] "What is Unsupervised Learning? — IBM," Ibm.com, 2023. Available: https://www.ibm.com/topics/unsupervised-learning.

[7] Li, Y. (2018) Deep reinforcement learning: An overview, arXiv.org. Available at: https://arxiv.org/abs/1701.07274.

[8] Reinforcement learning (2023) GeeksforGeeks. Available at: https://www.geeksforgeeks.org/what-is-reinforcement-learning/