

Final Project - Supervised Learning solution

1st Delivery

Freddy Alonzo Mondragon, Brenda Castillo Fernandez, Mariana Chi Centeno, Josue Gomez Gonzalez, Jhair Cach Rosas, Denzel Chuc, *Computational Robotics Engineering, Universidad Politecnica de Yucatan*

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, L^AT_EX, 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.

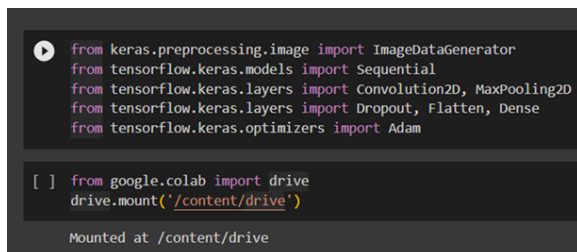
C. Neural Network

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

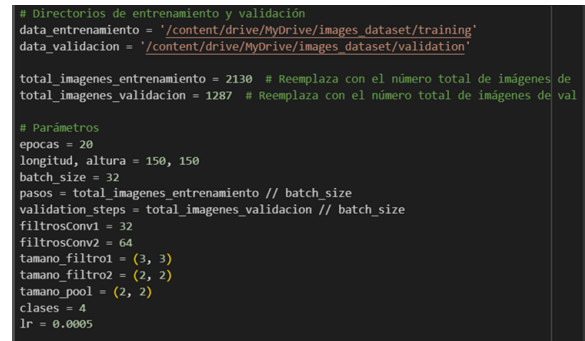
[ ] from google.colab import drive
drive.mount('/content/drive')

Mounted at /content/drive
```

Fig. 1. Code created

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.



```
# Directorios de entrenamiento y validación
data_entrenamiento = '/content/drive/MyDrive/images_dataset/training'
data_validacion = '/content/drive/MyDrive/images_dataset/validation'

total_imagenes_entrenamiento = 2130 # Reemplaza con el número total de imágenes de
total_imagenes_validacion = 1287 # Reemplaza con el número total de imágenes de val

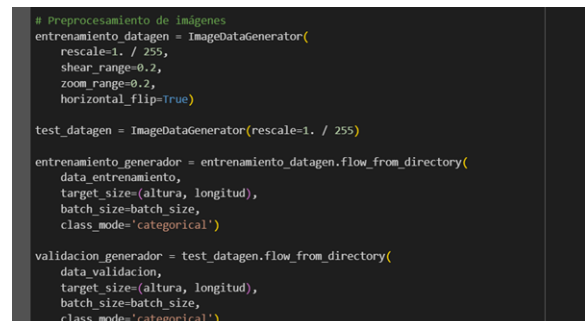
# Parámetros
epochas = 20
longitud, altura = 150, 150
batch_size = 32
pasos = total_imagenes_entrenamiento // batch_size
validation_steps = total_imagenes_validacion // batch_size
filtrosConv1 = 32
filtrosConv2 = 64
tamano_filtrol = (3, 3)
tamano_filtrol2 = (2, 2)
tamano_pool = (2, 2)
clases = 4
lr = 0.0005
```

Fig. 2. Code created

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.



```
# Preprocesamiento de imágenes
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')
```

Fig. 3. Code created

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.

```
# Crear modelo cnn
cnn = Sequential()
cnn.add(Convolution2D(filtersConv1, tamaño_filtrot, padding="same", input_shape=(longitud, altura, 3), activation='relu'))
cnn.add(MaxPooling2D(pool_size=tamaño_pool))
cnn.add(Convolution2D(filtersConv2, tamaño_filtrot, padding="same", activation='relu'))
cnn.add(MaxPooling2D(pool_size=tamaño_pool))
cnn.add(Flatten())
cnn.add(Dense(256, activation='relu'))
cnn.add(Dropout(0.5))
cnn.add(Dense(classes, activation='softmax'))

# Compilar el modelo
cnn.compile(loss='categorical_crossentropy',
            optimizer=Adam(lr=lr),
            metrics=['accuracy'])

# Entrenar el modelo
cnn.fit(
    entrenamiento_generator,
    steps_per_epoch=pasos,
    epochs=epochas,
    validation_data=validacion_generator,
    validation_steps=validacion_pasos)

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

Fig. 4. Code created

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.

```
1 import numpy as np
2 from keras.preprocessing.image import load_img, img_to_array
3 from keras.models import load_model
4
5 longitud, altura = 150, 150
6 modelo = 'content/models/modelo.hs'
7 pesos_modelo = 'content/models/pesos.hs'
8 cnn = load_model(modelo)
9 cnn.load_weights(pesos_modelo)
10
11 def predict(file):
12     x = load_img(file, target_size=(longitud, altura))
13     x = img_to_array(x)
14     x = np.expand_dims(x, axis=0)
15     array = cnn.predict(x)
16     result = array[0]
17     answer = np.argmax(result)
18     if answer == 0:
19         print("pred: Ambulancia")
20     elif answer == 1:
21         print("pred: Bus")
22     elif answer == 2:
23         print("pred: Carro de Bomberos")
24     elif answer == 3:
25         print("pred: Rover")
26
27 return answer
28
29 predict('content/drive/MyDrive/images_dataset/validation/carro_bomberos/image_2871.png')
```

Fig. 5. Code created

D. 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.

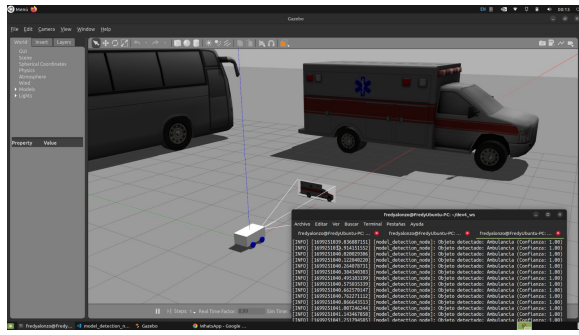


Fig. 6. Function Evidence

III. CONCLUSION

In conclusion, this project has successfully achieved the development of a robotic camera system capable of not only detecting vehicles but also identifying their specific types. Through the integration of the Robot Operating System (ROS) and the utilization of a pre-trained neural network, the robotic camera exhibited remarkable capabilities in real-time vehicle recognition. The system's accuracy and efficiency in differentiating between various types of vehicles signify a significant step forward in the realm of autonomous robotics and computer vision.

The ability to accurately identify and categorize vehicles has wide-ranging applications, from enhancing traffic management systems to optimizing logistics in industrial settings. The successful implementation of this technology showcases its potential to revolutionize transportation and automation industries, leading to safer, more efficient, and intelligent robotic systems.