

Machine Learning Project Report: CNN Model for image classification

Oscar Andre Dorantes Victor , 2009045, Jesus Israel Prado Pineda, 2009107,
Brandon Isaac Pacheco Chan, 2009102, Yuliana Alejandra Molina Cortés, 2009096,
Jesus Eduardo Casas Navarro, 2009024, Manuel Nicolas Nahib Franco Valencia 1909066

Abstract—This project explores the application of machine learning techniques, specifically Convolutional Neural Networks (CNN), in the domain of waste management. It aims to develop a model that accurately classifies waste images into 'Recyclable' and 'Organic' categories, utilizing TensorFlow, Keras, and other Python libraries. The approach encompasses data preprocessing, model training, and evaluation, with a focus on achieving high accuracy and precision. The results demonstrate the potential of machine learning in aiding environmental conservation efforts.

Index Terms—Machine Learning, CNN model, Supervised learning, classification.

CONTENTS

I	Introduction	1
II	Objectives	1
III	Methods and Tools	1
IV	Development	2
IV-A	Import Libraries	2
IV-B	Counter	3
IV-C	Visualization	3
IV-D	Visualization of Classification	3
IV-E	Model Architecture and Training	4
IV-F	Data Generator	4
IV-G	Prediction Function Implementation	4
IV-H	Documentation and Reproducibility	5
IV-I	Optimization and Refinement	5
V	Analysis and Results	5
VI	Conclusion	5
VII	References	5
References		5

I. INTRODUCTION

THis project combines machine learning and environmental care, focusing on improving how waste is managed. Traditionally, sorting waste into categories like 'Recyclable' and 'Organic' is done by hand, which can be slow and inaccurate. The objective is to use machine learning to improve the speed and accuracy of the process, and therefore, be able to apply it in robotics.

In this project, machine learning is combined with environmental care, with a focus on enhancing waste management methods. The traditional practice of manually sorting waste into categories like 'Recyclable' and 'Organic' often results in a slow and inaccurate process. The goal is to employ machine learning for a swifter and more precise approach, potentially integrating it into robotics.

The utilization of Convolutional Neural Networks (CNNs), renowned in machine learning for their ability to effectively recognize and classify images, is a key component of this project. The intention is to develop a system capable of autonomously distinguishing between recyclable and organic waste by training a CNN with various waste images. The purpose extends beyond merely creating a smart tool; it aims to make a meaningful environmental impact.

The project aspires to demonstrate the potential of technology, particularly AI, in addressing significant environmental challenges. It emphasizes not only the technical aspects but also the promotion of further research and the application of machine learning for environmental benefit.

Finally, it is desired that this project demonstrates the ways in which technology, particularly AI, can be utilized to address critical environmental issues. The focus is not solely on the technical details, but also on fostering more research and promoting the use of machine learning in aiding our planet.

II. OBJECTIVES

The primary objectives of this project are to:

- 1) Develop and train a machine learning model capable of accurately classifying images into 'Recyclable' and 'Organic' waste categories.
- 2) Use image processing techniques to prepare and augment data for the machine learning model.
- 3) Implement a robust evaluation framework to assess the performance of the model using various metrics.
- 4) Explore the potential of machine learning algorithms in enhancing waste management and recycling processes.
- 5) Provide a comprehensive, reproducible, and well-documented computational notebook that serves as a blueprint for similar environmental sustainability projects.

III. METHODS AND TOOLS

The project makes use of a variety of Python ecosystem tools and packages. Key among these are TensorFlow and

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

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

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

The convolutional layers are, therefore, the core of a CNN. Here, filters process the input image to extract pivotal features such as edges, textures, and shapes. Following this, the convolutional layer outputs undergo processing in the pooling layers. These layers downscale the feature maps, effectively decreasing their spatial dimensions but preserving essential information.

Finally, the data processed by the pooling layers is directed through one or more fully connected layers. These layers are crucial for the final step of the process, where they contribute to the prediction or classification of the image based on the extracted and processed features. Additionally, one primary advantage of CNNs is their ability to minimize the need for extensive pre-processing of images.

IV. DEVELOPMENT

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

A. Import Libraries

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

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

The for loop iterates through the contents of the directory /content/Archivos/DATASET using os.walk(). os.walk() is a function that traverses a directory and its subdirectories, returning a tuple that contains the path of the current directory,

a list of subdirectories, and a list of filenames in the current directory. In this case, only dirname (the path of the current directory) is used, and the information about subdirectories and files is ignored.

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

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

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

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

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

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

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

subsection Creation of the DataFrame

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

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

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

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

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

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

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

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

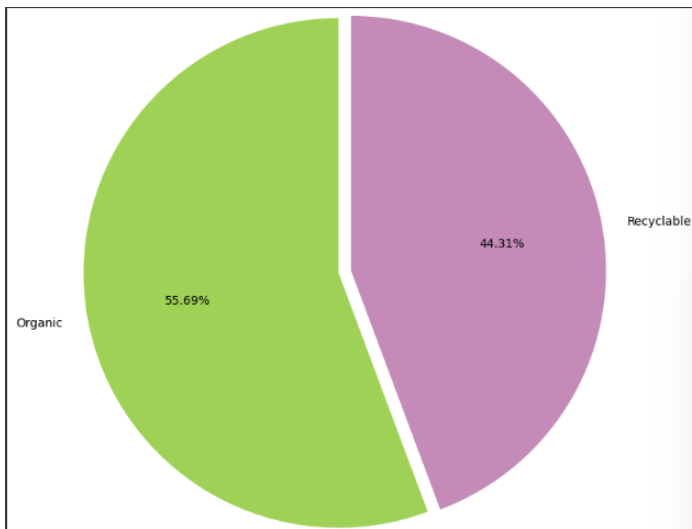


Fig. 1. Distribution of labels in the data

```
from keras.models import Sequential
from keras.layers import Conv2D, MaxPooling2D,
    Activation, Dropout, Flatten, Dense,
    BatchNormalization
from keras.preprocessing.image import
    ImageDataGenerator, img_to_array, load_img
from tensorflow.keras.utils import plot_model
from glob import glob

x_data = []
y_data = []

for category in glob(train_path+'/*'):
    for file in tqdm(glob(category+'/*')):
        img_array=cv2.imread(file)
        img_array = cv2.cvtColor(img_array, cv2.
            COLOR_BGR2RGB)
        x_data.append(img_array)
        y_data.append(category.split("/")[-1])

data=pd.DataFrame({'image': x_data, 'label': y_data
    })
data.shape
from collections import Counter
Counter(y_data)
```

B. Counter

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

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

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

C. Visualization

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

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

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

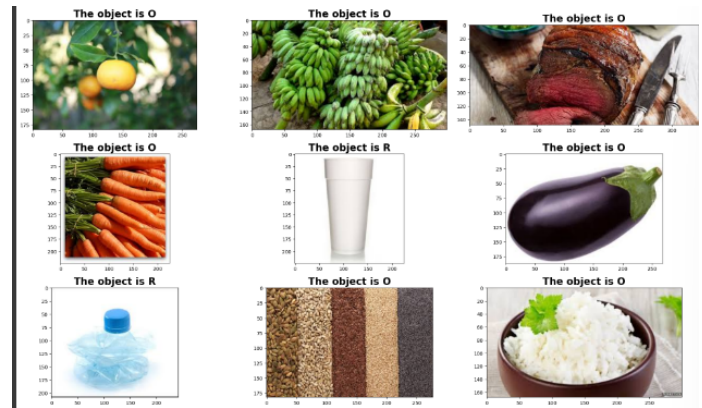


Fig. 2. O means Organic and R, Recyclable.

D. Visualization of Classification

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

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

```
plt.subplot(4,3,i)
index=np.random.randint(15000) selects a random index
within the range of 0 to 14,999. This is used to display a
random image from your data.
```

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

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

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

E. Model Architecture and Training

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

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

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

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

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

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

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

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

The architecture involves:

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

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

F. Data Generator

`train_generator` and `test_generator` are image data generators created using `train_datagen` and `test_datagen`, which are instances of `ImageDataGenerator`.

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

`train_path`: Path to the training directory. `target_size`: Size to which the input images will be resized (in this case, 224x224 pixels). `batch_size`: Batch size, which is set to `batch_size`. `color_mode`: Color mode of the images (in this case, "rgb").

`class_mode`: Classification mode (in this case, "categorical" for categorical classification). `test_datagen.flow_from_directory` does the same as `train_datagen.flow_from_directory`, but applies to the test directory (`test_path`) instead of the training directory.

Here we obtain the accuracy of the model:




Fig. 3. Model Accuracy

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

G. Prediction Function Implementation

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

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

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

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

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

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

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

If 'result' is equal to 0, the message "This image is Recyclable" is displayed. If 'result' is equal to 1, the message "This image is Organic" is displayed. The messages are displayed in blue color ("33[94m") to make them more visible.

```
def predict_func(img):
    plt.figure(figsize=(6,4))
    plt.imshow(cv2.cvtColor(img, cv2.COLOR_BGR2RGB))
    plt.tight_layout()
    img = cv2.resize(img, (224, 224))
    img = np.reshape(img, [-1, 224, 224, 3])
```

```

result = np.argmax(model.predict(img))
if result == 0: print("\033[94m"+"This image ->
Recyclable"+" \033[0m")
elif result ==1: print("\033[94m"+"This image ->
Organic"+" \033[0m")
test_img = cv2.imread("/content/Archivos/DATASET/
TEST/O/O_12571.jpg")
predict_func(test_img)
test_img = cv2.imread("/content/Archivos/DATASET/
TEST/R/R_10027.jpg")
predict_func(test_img)

```

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

H. Documentation and Reproducibility

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

I. Optimization and Refinement

Optimization efforts focus on model tuning and refinement:

```

# Hyperparameter tuning
model.optimizer.lr = 0.001
model.epochs = 50
# Employing dropout and batch normalization
model.add(Dropout(0.5))
model.add(BatchNormalization())

```

These techniques aim to improve the model's generalization capabilities and performance.

V. ANALYSIS AND RESULTS

The model demonstrated commendable performance in classifying waste images. A high accuracy suggests that the model is effective in generalizing from the training data to new, unseen data. Precision and recall values provide insights into the model's ability to minimize false positives and false negatives, respectively.

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

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

VI. CONCLUSION

At the end of this project, we have a machine learning model that can sort waste into the right categories really well. This shows that Machine Learning Models can be a big help in making things better for the environment. Our model, which uses CNNs, is a great example of how we can use technology to make waste management better and more automatic. But this project is about more than just making a good model. It shows us how we can use technology to help with environmental problems. It could be made better by making it recognize more types of waste and by testing it in real-life situations.

VII. REFERENCES

The link for the notebook of the project is:

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

REFERENCES

- [1] TensorFlow, "Convolutional Neural Network (CNN)," in TensorFlow Core, [online]. Available: <https://www.tensorflow.org/tutorials/images/cnn>.
- [2] Techsash, "Waste Classification Data," Kaggle, [Online]. Available: <https://www.kaggle.com/datasets/techsash/waste-classification-data>.
- [3] L. Chen, S. Li, Q. Bai, J. Yang, S. Jiang, and Y. Miao, "Review of Image Classification Algorithms Based on Convolutional Neural Networks," *Sensors*, vol. 21, no. 22, Art. no. 4712, 2021, doi: 10.3390/s21224712.
- [4] Y. Kong, X. Ma, and C. Wen, "A New Method of Deep Convolutional Neural Network Image Classification Based on Knowledge Transfer in Small Label Sample Environment," *Sensors*, vol. 22, no. 3, Art. no. 898, 2022, doi: 10.3390/s22030898.
- [5] N. Sharma, V. Jain, and A. Mishra, "An Analysis Of Convolutional Neural Networks For Image Classification," *Procedia Computer Science*, vol. 132, pp. 377-384, 2018, doi: 10.1016/j.procs.2018.05.198.