Garbage classification using IBM cloud and deep learning

Akshay Markhedkar, Anirudh Lodh, Utkarsh Saxena, Abhinav Tanwar

School of computing science and engineering, VIT Bhopal University

Introduction

The accumulation of non-recyclable waste on landfills all around the world and the huge amount of time that most of its materials take to biodegrade can affect in a significant manner our lifestyle in the near future if we, as a society, don't act to prevent this from happening. Moreover, among the most notorious risks for humans, waste accumulation can enhance the disease spread via vectors such as flies, mosquitoes, and many more insects. Also, in addition to the ruin of the beauty of natural habitats, deforestation, and terrain occupation to provide enough space to landfills, soil and water can be susceptible to be polluted due to the toxic chemicals located in improperly treated materials. At the same time, that pollution can alter the food chain, which inevitably leads to more diseases and health issues for humans and the natural ecosystems worldwide.

There are mainly three reasons why waste accumulation is becoming an increasingly severe problem during the last 50 years. The first one is the absence of recyclable items available on the market, even though companies have been developing more sustainable and ecological products for a long time. The second reason is another of the most popular problems present nowadays, denoted as *overpopulation*. The fact of having to supply a large number of people with resources of all kinds supposes a very complex logistic challenge when dealing with trash generation, so there is an increase in the percentage of products that could be recycled but, in contrast, end in a landfill or even the ocean, affecting the lives of thousands of marine species. Finally, the third reason consists of the lack of involvement that we show as a society over these kinds of problems such as climate change.

To show the waste accumulation problem in terms of data, it's essential to know that the world population produces between 7 and 9 billion tons of waste every year, of which 70% is mistreated, ending in landfills with the risk of polluting natural environments and causing new threats for human health like *ocean microplastics*. This data refers to the total amount of used, unwanted, and discarded objects that humankind creates. However, there is a distinction between the total waste produced and the so-called *Municipal Solid Waste (MSW),* which only includes garbage generated in urban centers or their areas of influence. Concerning the amount of *MSW* with respect to the rest of the waste, approximately 2 billion tons of urban garbage is produced yearly, with around 33% of that not adequately managed. It means that each person generates from *0.1* to *4.5* kilograms of waste every day, with an average of *0.7kg*. In addition, it's expected that the *MSW* will increase to *3.4* billion tons by the year 2050 due to the rapidly growing global population and the need for the intensive use of natural resources for the development of industry and the sustaining of our civilization.

Ideally, a fully implemented *circular economy* model would essentially be an excellent solution for the accumulation problem, besides the climate change and even supply crisis in some places of our world. That's because its three principles *(eliminating waste and pollution, circulating products and materials, and regenerating nature)* lead towards an effective way of managing natural resources that sometimes we don't correctly value. Nevertheless, it isn't easy to carry out such a complex plan in a complete manner due mainly to technology, engineering, and logistic limitations.

Machine Learning in environment care

Despite these limitations, some emerging new state-of-the-art technologies are starting to change how we see and face these problems. One of the most notorious nowadays is Machine Learning, a branch of Artificial Intelligence that makes it possible for machines to learn specific and complex tasks like classification, prediction, decision making, content generation, etc., by using large amounts of data in combination with advanced learning algorithms inspired on the way we humans learn. Automating such tasks using machine learning can sometimes be extremely useful for humans due to its scalability and performance.

Regarding its relationship with sustainability and circular economy implementation, machine learning has the potential to automate a wide variety of tasks. From data trend predictions that improve the quality of the air we breathe and finding patterns on data collected to measure global warming over time,

identifying waste in natural environments, or even classifying between several types of garbage materials to boost the performance of waste treatment plants. So the proper identification and differentiation of recyclable items from the rest of the litter can also be a significant advance towards a sustainable circular economy model. However, machine learning is now present in more processes related to environmental care than we think. For example, a correct energy or product demand prediction made by a model can avoid the waste of natural resources. Also, advanced enough models can even discover new materials by working with chemical structures, improving the efficiency and recyclability of everyday products.

Objective

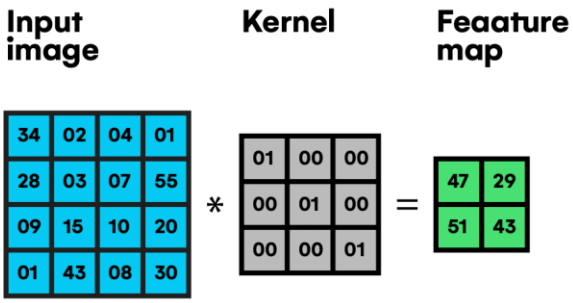
The goal of this article is to contribute to the development and improvement of machine learning techniques focused on solving environmental issues like the above-mentioned *(waste accumulation, global warming, pollution, etc.)* by creating a model capable of sorting between nine different types of waste depending on the fabrication materials, and thus its *recyclability*. Furthermore, it will be subject to experimentation by professionals and subject matter experts since its open-source. That experimentation is key when improving the performance of models in the field of machine learning, along with the data gathering and processing techniques used in the process.

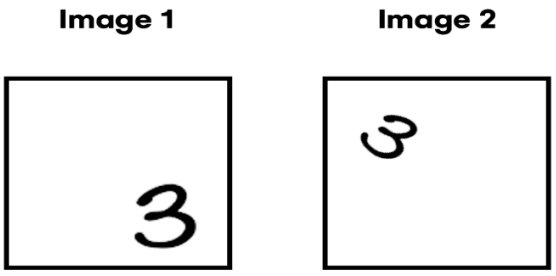
Also, intuitively explaining the internal functioning of machine learning algorithms and the benefits they could bring to humankind in the future can raise the understanding and trust that society has over such new exponential technologies. Additionally, focusing on the use of new disruptive methods to attempt to solve environmental problems increases the awareness that we all have about taking care of our planet, giving place to the possible implementation of new sustainable habits in our daily lives.

Proposed work

So, to build a classifier that correctly sorts out different types of waste, we will need a model architecture denominated as *Convolutional Neural Network* since our dataset will be composed mainly of *labeled* images, meaning that each image has a corresponding label that indicates the correct prediction *(type of waste material)* the model will have to provide as output. At the same time, the model will also need a *fully-connected network* after the convolutional module to transform an arbitrary response given by it to a set of values with a particular structure that will allow us to determine the class predicted by the model.

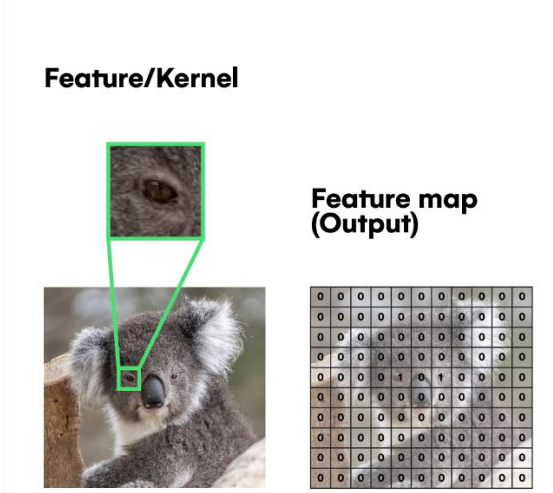
The aim of using a *Convolutional Network* to process an image resides in its ability to extract certain patterns or features from images with an invariance in position, rotation, and scale. To understand the power of these properties when detecting features in images, let's consider an example.



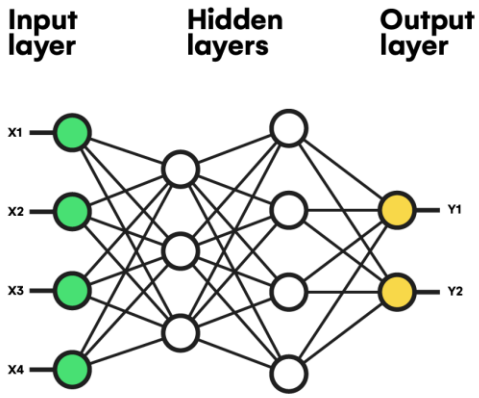


For instance, we could use an entire fully-connected network if we want to detect the handwritten number three as a feature. Still, it would only *'see'* the digit in a specific image area and rotation, giving wrong predictions when the element is placed in a random position or rotated. In contrast, a model with a convolutional part can effectively detect the feature independently of its properties.

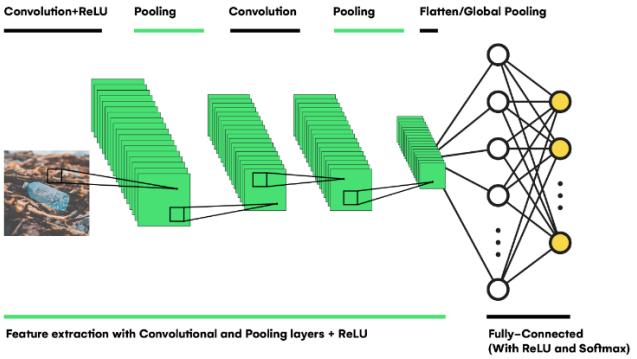
If you wonder how this can be done, this architecture's fundamental operation is the *Convolution*, hence the name *Convolutional Network*. In a nutshell, the Convolution operation takes as input the pixels of an image and outputs a feature map, which is no more than a matrix of values that can perfectly represent an image. As you can see in the above animation, a matrix colored in dark grey called *kernel* goes through all the input image pixels *(blue matrix),* multiplying each pixel value of the kernel by the corresponding pixel values of the input image. After that, it places the sum of all the resulting values in the feature map matrix sequentially. To preserve the dimensionality of the feature map with respect to the input image, you can add *padding,* which is a line of null values around the image. Also, you can change parameters like stride, which determines the *'steps'* that the kernel takes to the left while traversing the image.



A machine learning model can modify the values of a kernel to detect complex features like human faces, car wheels, etc. For example, as you can see above, you can pass an eye detector kernel through an image using Convolution and know where the eyes are located just by looking at the output feature map.



Before jumping into the complete network, let's first look at the basics of neural networks. This mathematical model learns patterns and relationships in a set of data by imitating how our human brain works. The most fundamental component of a neural network is the neuron, which takes *n* input values and computes its *weighted sum (sum of all inputs multiplied by a corresponding weight parameter)*. Then, it passes that value through an activation function to introduce non-linearity and outputs the result of that function. Thus, a combination of multiple neurons organized in different layers enables the network to acquire 'knowledge.'



Finally, let's see how all the previous elements work together into a complete *Convolutional Model* like the one represented above. First, the input image is passed through a series of convolutional layers, which subsequently extract features with an increasing complexity using multiple kernels with different values in each layer. Then, in combination with the convolutional layers and to reduce the image's dimensionality, some pooling layers perform a convolution with specific variations like average, max, or min element, instead of the classic weighted

sum. These operations also contribute to the preservation of information across the network.

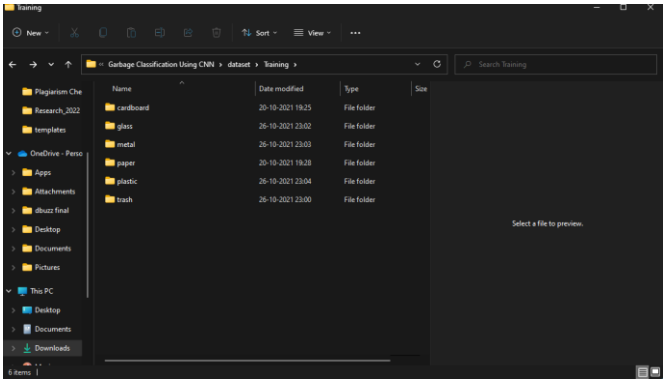
After extracting features, a *Flatten* or a *Global Pooling* layer transforms the output of the last convolutional layer and feeds a *fully-connected (dense)* network that performs classification from the learned patterns. The final layer of the entire network contains as many neurons as the classes we have to predict. With a *softmax* activation function, the layer can assign a probability for each neuron that represents how present a feature class is present in the input image.

Implementation

To build the *Convolutional Neural Network* programmatically, we will be using Python code running on the platform Jupyter notebook, which provides a fast, comfortable, and robust cloud environment for training such large models. Furthermore, the *Python* programming language has a wide variety of libraries oriented to constructing machine learning algorithms that make training and evaluation much more accessible. Such libraries are TensorFlow, Keras, NumPy, Matplotlib, etc.

Data collection and processing

Before building, training, and evaluating our model, we must gather a dataset of labeled waste images. There are many resources on the internet where you can download a dataset to use in a machine learning project, not only images dataset but also numerically based datasets like traffic, sales, etc.



Dataset division is carried out in the following displayed format.

After having the data ready, let's start to build the *Jupyter Notebook* with all the Python code. But, first, we need to import the libraries that we are going to need:

```
import tensorflow as tf
import numpy as np
import matplotlib.pyplot as plt
import random
import math
import os
import cv2 as cv
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import PolynomialFeatures, StandardScaler,
LabelEncoder
from sklearn.model_selection import train_test_split
```

Afterward, we have to split our data into two datasets *(train and test)*. Each of them is only used in the corresponding phase of the project *(training and evaluation of the model)*. This operation is critical to make the model capable of generalizing

from the provided data to any input that the user will use in production. The usual splitting rates are 70/30, 80/20, and 90/10 for training and testing, respectively.

The following code uses the *Keras* API of the *TensorFlow* library to preprocess the dataset located into a folder by resizing all the images to a standard dimension of 256x256 and setting a batch size of 128, meaning that in the training process, the data will pass through the network in chunks of 128 images.

```
DIR = "/content/WasteClassificationNeuralNetwork/WasteImagesDataset"

train_dataset =
tf.keras.preprocessing.image_dataset_from_directory(DIR,
validation_split=0.1, subset="training", seed=42, batch_size=128,
smart_resize=True, image_size=(256, 256))

test_dataset =
tf.keras.preprocessing.image_dataset_from_directory(DIR,
validation_split=0.1, subset="validation", seed=42, batch_size=128,
smart_resize=True, image_size=(256, 256))
```

Also, we can store the number of classes in a variable extracting it from the train dataset object *(9 classes in this case)* and use *tf.data.AUTOTUNE* to optimize the performance of both training and testing dataset objects.

```
classes = train_dataset.class_names
numClasses = len(train_dataset.class_names)
print(classes)

AUTOTUNE = tf.data.AUTOTUNE

train_dataset = train_dataset.prefetch(buffer_size=AUTOTUNE)
test_dataset = test_dataset.prefetch(buffer_size=AUTOTUNE)
```

Model building and training

As we are dealing with a relatively large dataset *(+5000 images),* it's convenient to use a common technique when training a model on such an amount of data called *Transfer Learning*. That refers to replacing the convolutional part of your model with an already trained one. So before training, your model will be able to extract useful features from the input images thanks to the trained convolutional part. Moreover, it will only have to train the last few dense layers, reducing the computing power and time involved. There are a lot of trained convolutional models available, but the most common ones are included in the *Keras API* that we are currently using for the project.

```
baseModel = tf.keras.applications.MobileNetV3Large(input_shape=(256,
256,3), weights='imagenet', include_top=False, classes=numClasses)
for layers in baseModel.layers[:-6]:
  layers.trainable=False

last_output = baseModel.layers[-1].output
x = tf.keras.layers.Dropout(0.45) (last_output)
x = tf.keras.layers.GlobalAveragePooling2D()(x)
x = tf.keras.layers.BatchNormalization() (x)
x = tf.keras.layers.Dense(256, activation = tf.keras.activations.elu,
kernel_regularizer=tf.keras.regularizers.l1(0.045),
activity_regularizer=tf.keras.regularizers.l1(0.045),
kernel_initializer='he_normal')(x)
x = tf.keras.layers.Dropout(0.45) (x)
x = tf.keras.layers.Dense(numClasses, activation='softmax')(x)

model = tf.keras.Model(inputs=baseModel.input,outputs=x)
```

As you can see in the code, the model is built upon the *MobileNetV3Large* pre-trained model without its original final *Dense* layers, which are replaced by a hidden layer of 256 neurons that receives data from a Global Average Pooling operation, a *Batch Normalization* layer for fixing Internal covariate shift, an *ELU* activation function, and several *Dropouts*. After that, the last layer contains as many neurons as output classes determined by the *'numClasses'* variable. Note that we need to

add *L1 regularization* to prevent overfitting, the leading cause of the lack of generalization.

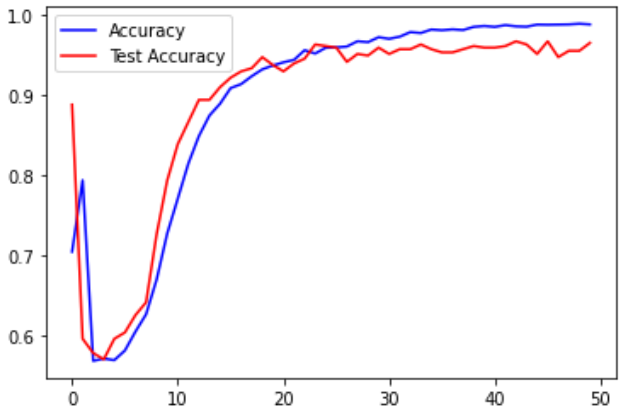
In addition to transfer learning, sometimes it's appropriate to unfreeze the last few convolutional layers of the pre-trained convolutional model. That practice is called *Fine Tuning* and improves the model's overall performance. Here, only the last six layers are unfrozen *(able to be trained)*.

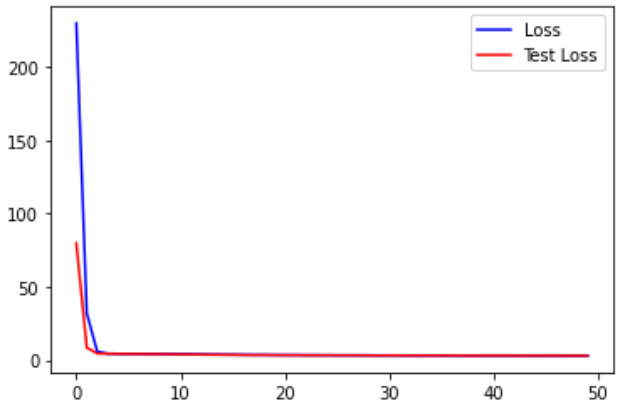
```
epochs=epochs)
history = model.fit(train_data, validation_data=test_data,
epochs = 20

metrics=['accuracy'])
loss='sparse_categorical_crossentropy(from_logits=True)',
)
model.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=0.00125
```

Once the model is built, we compile it by assigning the *SparseCategoricalCrossentropy* loss function, the *Adam* algorithm for optimizing all the network parameters, and the accuracy metric. Finally, the model is trained using the *fit()* function during 50 epochs *(times that the entire dataset passes through the network)*.

Model evaluation

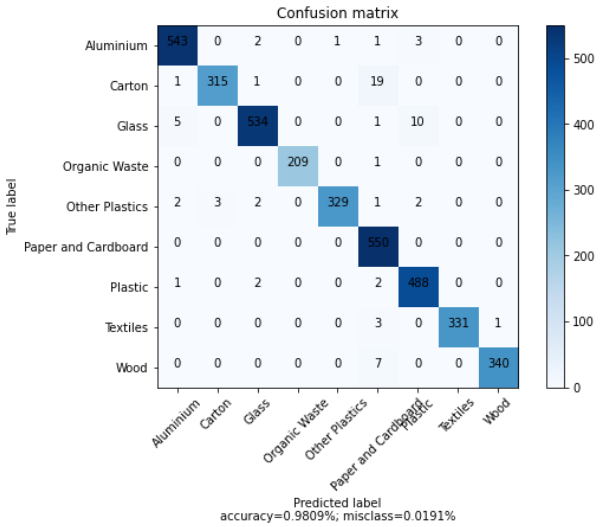
The training time it takes depends on the hardware, model complexity, and dataset size.





After completing the training, we can plot the loss and accuracy values over epochs in a graph using the *Matplotlib* library to inspect how the model behaved during the process. As you can see on the left chart, the accuracy increases during the very first epochs in a similar way for both train and test values *(one for each dataset)* until there comes the point when the test accuracy goes below the train accuracy *(blue line)*. That's an indicator of overfitting; the greater the difference between the two values of training and test, the greater the overfitting and less generalizable the model is. Despite the loss of generalization, there is not much than 2% of the difference in this case, from 98.75% accuracy on training to 96.45% on testing, which

doesn't affect in a significant way the results of the model.



We can observe the final results by building a *Confusion Matrix* and evaluating the model with data from the two datasets. The accuracy obtained reaches 98%, but the overfitting issue presented before can decrease this value to *97%* or even *96%*. Nevertheless, the best way to test the model's performance is by deploying it to production and evaluating it with a large amount of *'unseen'* data.

To make predictions over a single image, which is the primary use of this model, we can convert the input image to an array of values by preprocessing it with the Keras API and use the *predict()* function to get predictions from the model.

```
path = "/content/img.jpg"

img = tf.keras.preprocessing.image.load_img(path, target_size=(256,
256))
img_array = tf.keras.preprocessing.image.img_to_array(img)
img_array = tf.expand_dims(img_array, 0)

predictions = model.predict(img_array)
```

For instance, this image of plastic spoons would get the following result:

```
[1.1614963e-01 1.0170896e+00 3.3458400e-01 3.6209685e-01 9.5969460e+01
 5.7956848e-02 1.9298062e+00 8.9533217e-02 1.2331158e-01]
 ['Aluminium', 'Carton', 'Glass', 'Organic Waste', 'Other Plastics',
'Paper and Cardboard', 'Plastic', 'Textiles', 'Wood']
Prediction:  Other Plastics 95.96946239471436%
```

Applications

Having seen the facilities that modern libraries like TensorFlow provide for building such complex machine learning algorithms, it's convenient to analyze the real benefits of this model when dealing with environmental threats. The first one is its weight, occupying only 30MB of disk space thanks to the mobile-optimized convolutional model that we used when performing *Transfer Learning (MobileNetV3)* makes the model easy to

deploy, which encourages users to take advantage of it. Additionally, its results in the evaluation phase and the generalization capability despite the small overfitting observed in the loss and accuracy curves make it robust enough to be scalable to a larger dataset.

But to build an extensive dataset and scale it over time, a powerful infrastructure is needed to store the data. Also, the collaboration of many persons is even more necessary to gather and interpret the data humanly. That's the reason why initiatives like *Joinus4theplanet (a platform that connects and provides people the opportunity to solve environmental issues)* focus so much on the cooperation that makes the society able to transform the world and preserve its environment. Moreover, collaboration is one of the fundamental requirements to improve machine learning and technology overall. Hence, we must work together if we want a balanced nature and a world free of pollution or any kind of dangers.

Conclusion

In conclusion, we proposed a waste classification system that is able to separate different components of waste using the Machine

learning tools. This system can be used to automatically classify waste and help in reducing human intervention and preventing infection and pollution. From the result, when tested against the trash dataset, we got an accuracy of 87%. The separation process of the waste will be faster and intelligent using our system without or reducing human involvement. If more image is added to the dataset, the system accuracy can be improved In the future, we will tend to improve our system to be able to categories more waste item, by turning some of the parameters used.