CLASIFICADOR DE RESIDUOS BASADO EN IMÁGENES

Delgado Sammy[1], Hualpa Katherine[2], Montero Edwar[3] y Obregón Harold[4]

[1,2,3,4] *Universidad Nacional de Ingeniería. Facultad de Ingeniería Industrial y de Sistemas.*

Escuela de Ingeniería de Sistemas.

RESUMEN

Las aplicaciones de visión artificial y el análisis de imágenes son capaces de resolver varios problemas del sector industrial,científico o de seguridad en la actualidad. Una de ellas es la clasificación de imágenes. Por lo tanto, el objetivo de este paper es el de construir, diseñar y evaluar un procedimiento para resolver el problema de clasificación de residuos haciendo uso de algoritmos de machine learning en Python.Para ello se ha seguido la metodología CRISP-DM. El conjunto de datos utilizado consiste en 6 clases de imágenes de residuos con un número variable de imágenes en cada clase. Para la preparación del dataset se han realizado actividades como la técnica de extracción del histograma a color en tres dimensiones.Luego,en la preparación de los datos se han realizado actividades como la generación de atributos derivados y la eliminación de atributos con valores constantes y luego se les ha dividido en dos grupos de datos: los datos de entrenamiento y los datos de prueba. Se ha elaborado, además, una arquitectura de los componentes del sistema en sus diferentes niveles. Y por último, para el entrenamiento del modelo se ha hecho uso del árbol de decisión, random forest y redes neuronales convolucionales; posteriormente, se realizó la evaluación de los modelos a través de la matriz de confusión y su optimización.

ABSTRACT

Machine vision applications and image analysis are capable of solving various problems in today's industrial, scientific or security sector. One of them is image classification. Therefore, the objective of this paper is to build, design and evaluate a procedure to solve the residual classification problem using machine learning algorithms in Python. The data set used consists of 6 classes of residual images with a variable number of images in each class. For the preparation of the dataset, activities such as the three-dimensional color histogram extraction technique have been carried out. They have been divided into two groups of data: the training data and the test data. In addition, an architecture of the system components has been developed at its different levels. Finally, for model training, use was made of the decision tree, random forest and convolutional neural networks; subsequently, the models were evaluated through the confusion matrix and their optimization.

Palabras clave:machine learning, CRISP-DM,árbol de decisión,random forest, optimización.

1. INTRODUCCIÓN

En este paper se aplican conceptos de machine learning. Es decir, se refiere a una clase de algoritmos informáticos que aprenden de ejemplos en lugar de ser programados explícitamente para realizar una tarea. Dentro del aprendizaje automático, el problema de clasificación es el tema central. Este es un problema de modelado predictivo en el que se predice una etiqueta de clase para un ejemplo dado de datos de entrada. Aclarando eso, el problema que se va a tratar es el de clasificación de residuos haciendo uso de algoritmos de machine learning en Python siguiendo una metodología llamada CRISP-DM que es un método probado para orientar trabajos de minería de datos .

2. PLANTEAMIENTO DEL PROBLEMA

Los impactos ambientales generados por las actividades humanas son numerosos. Un ejemplo de impacto negativo es la generación de residuos sólidos o comúnmente llamada basura. Dentro de estos residuos sólidos hay materiales como vidrio, plástico, papel y metales que se podrían aprovechar para darles un nuevo uso, o lo que se conoce como reciclaje, pero la mala gestión de los residuos hace que el porcentaje de reciclaje sea mínimo y que la cantidad de residuos que llega a los vertederos principales sobrepase la capacidad de estos, generando emergencias sanitarias.
Después de 21 años de promulgada la Ley General de Residuos Sólidos, el Perú sufre aún graves problemas de limpieza pública. Cada día somos más habitantes urbanos y cada día en las ciudades el peruano produce más basura (en promedio un peruano genera más de medio kilo al día). El volumen de basura producido en el Perú está aumentando; hace 10 años era de 13 mil T/día, hoy alcanza las 18 mil T. El 50% de estos residuos no se disponen adecuadamente: tenemos ciudades sucias, calles, ríos, playas sucias, etc.
La clasificación de estos residuos contribuye considerablemente a la recuperación al obtener una mejor calidad de los residuos, aumentando sus posibilidades de ser incorporados en un proceso de reutilización o reciclaje.
El principio de las 3'Rs propone, en orden de importancia lo siguiente: Reducir, Reutilizar y Reciclar. El reciclaje (la última de las erres) tiene también sus ventajas en términos generales, ya que es de menor impacto ambiental obtener nueva materia a partir del proceso del reciclaje que elaborar nuevas materias; por lo general, se necesita menos consumo de energía, menor cantidad de agua y menor extracción de recursos naturales vírgenes que implican en la mayoría de los casos impactos negativos a los recursos naturales.

3. METODOLOGÍA DE DESARROLLO

Metodología CRISP-DM

CRISP-DM, que son las siglas de CRoss-Industry Standard Process for Data Mining, es un método probado para orientar sus trabajos de minería de datos .
• Como metodología, incluye descripciones de las fases normales de un proyecto, las tareas necesarias en cada fase y una explicación de las relaciones entre las tareas.

• Como modelo de proceso, CRISP-DM ofrece un resumen del ciclo vital de minería de datos

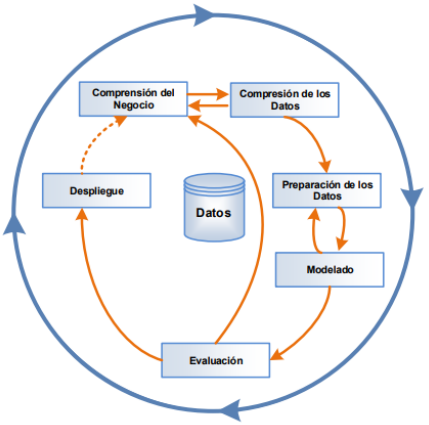


Figura 1:Metodología CRISP-DM

El ciclo vital del modelo contiene seis fases con flechas que indican las dependencias más importantes y frecuentes entre fases. La secuencia de las fases no es estricta. De hecho, la mayoría de los proyectos avanzan y retroceden entre fases si es necesario.

Las fases sobre las cuales se desarrollará la metodología CRISP-DM empleada en esta investigación serán las siguientes:

Comprensión del negocio: Esta fase se centra en entender los objetivos y requerimientos del proyecto desde una perspectiva de negocio, plasmando todo esto en una definición del problema de minería de datos y un plan preliminar diseñado para obtener los objetivos. Consta de 4 subfases:

- Establecimiento de los objetivos del negocio (Contexto inicial, objetivos, criterios de éxito).
- Evaluación de la situación (Inventario de recursos, requerimientos, supuestos, terminologías propias del negocio, etc.).
- Establecimiento de los objetivos de la minería de datos (objetivos y criterios de éxito).
- Generación del plan del proyecto (plan, herramientas, equipo y técnicas).

Para este proyecto, en esta fase se entenderá los objetivos planteados en el examen, así como la generación de un plan para la obtención de atributos que podamos extraer del conjunto de imágenes entregadas.

Comprensión de los datos: Esta fase se encarga de recopilar y familiarizarse con los datos, identificar los problemas de calidad de datos y ver las primeras potencialidades o subconjuntos de datos teniendo presente los objetivos del negocio. Consta de 4 subfases:

- Recopilación inicial de datos
- Descripción de los datos
- Exploración de los datos
- Verificación de calidad de datos

Preparación de los datos: Esta fase se encarga de obtener la "vista minable" o dataset. Consta de 5 subfases:

- Selección de los datos
- Limpieza de datos
- Construcción de datos
- Integración de datos
- Formateo de datos

Modelado: Esta fase se encarga de aplicar las técnicas de modelado o de minería de datos a los dataset del paso anterior. Consta de 4 subfases:

- Selección de la técnica de modelado
- Diseño de la evaluación
- Construcción del modelo
- Evaluación del modelo

Evaluación: es necesario evaluar los modelos de la fase anterior, pero ya no sólo desde un punto de vista estadístico respecto de los datos (cómo se realiza en la última subfase de la fase anterior), sino ver si el modelo se ajusta a las necesidades establecidas en la primera fase, es decir, si el modelo nos sirve para responder a algunos de los requerimientos del negocio. Consta de 3 subfases:

- Evaluación de resultados
- Revisar el proceso
- Establecimiento de los siguientes pasos o acciones

4. OBJETIVOS

- Objetivo General: Diseñar un prototipo de sistema que permita la identificación del tipo de residuo sólido que se presenta en la cámara a través de un algoritmo de machine learning que pueda clasificar los residuos de acuerdo a cada una de las siguientes clases: **cardboard, glass, metal, paper, plastic, trash**.

Para el cumplimiento de lo requerido, se han establecido cinco objetivos específicos:

- Preparación del Dataset
  Dado el conjunto de imágenes, se extraerán características importantes de los histogramas de cada canal RGB y otros como la suma de canales y la concatenación de la misma.

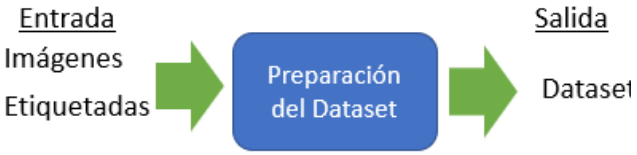


Figura 2: Preparación del dataset.

- Preparación de Datos
  Obtenido el dataset, se realizan modificaciones a los valores de manera que se optimicen para poder usarlos como entradas a algoritmos de machine learning.

Figura 3: Preparación de datos

- Entrenamiento del Modelo
  Posteriormente, el nuevo dataset se ingresará a algoritmos de machine learning para que logre identificar la relación entre los atributos y la variable objetivo.

Figura 4: Entrenamiento del modelo

- Evaluación del Modelo
  Se procede con ingresar los valores que fueron separados para probar el modelo; como resultado obtendremos métricas o indicadores respecto al rendimiento del modelo.

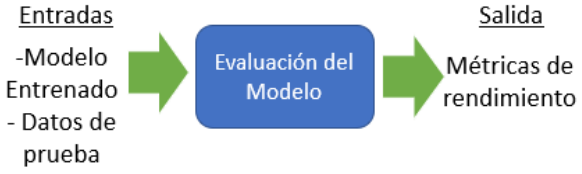


Figura 5: Evaluación del modelo

- Optimización

Se procede a variar los parámetros de entrenamiento del modelo, de tal manera que se logre un mejor resultado en cuanto al rendimiento del modelo. Se deberá obtener un accuracy mayor del 85%.

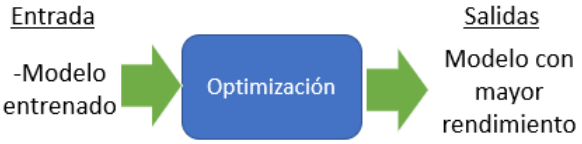


Figura 6: Optimización del modelo

5. ARQUITECTURA DE LOS COMPONENTES DEL SISTEMA

A continuación se mostrará la arquitectura de los componentes que se han considerado para resolver el problema. Se presentarán tres niveles de acuerdo a su profundidad de detalle:

Nivel 0:

Lo que se requiere es un sistema o modelo que logre clasificar los residuos de acuerdo a las clases antes mencionadas.

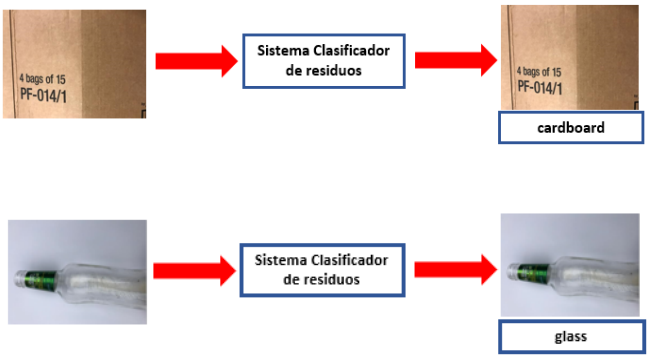


Figura 7: Arquitectura nivel 0.

Nivel 1:

Se presentan los objetivos específicos detallados anteriormente.

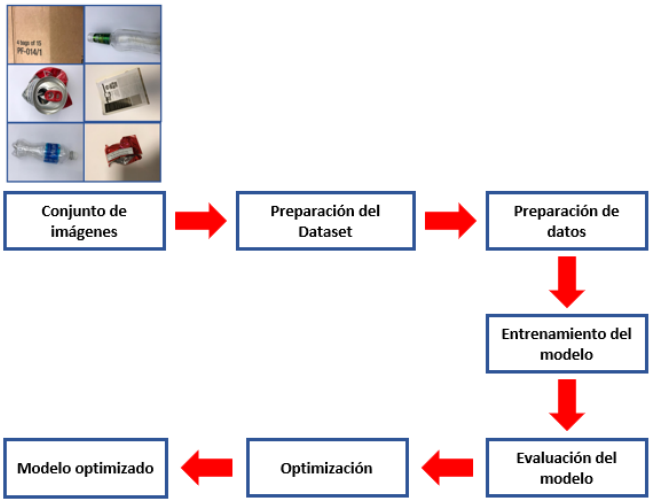


Figura 8: Arquitectura nivel 1.

Nivel 2:

Se desarrolla cada actividad que se ha realizado para la obtención del modelo que clasificará los residuos de acuerdo a su clase.

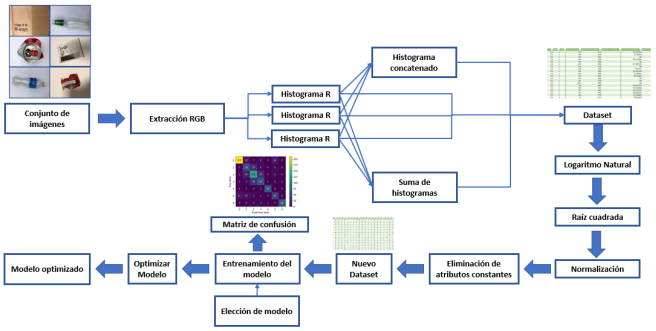


Figura 9: Arquitectura nivel 2.

6. DESARROLLO DE LOS COMPONENTES DEL SISTEMA

6.1 Preparación del Dataset

Para esto se hará uso de un conjunto de datos de imágenes de residuos de diferentes clases, para abordar el problema de clasificarlos de manera adecuada.

El sistema de etiquetado para la clasificación de residuos está basado en la siguiente numeración, que nos indicará a qué clase pertenece cada uno.

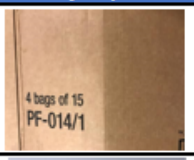
Clase	Categoría	Ejemplo
0	cardboard	
1	glass	
2	metal	
3	paper	
4	plastic	
5	trash	

Figura 10: Etiquetas.

Para elaborar la preparación de nuestro dataset realizaremos procesos de transformación a las imágenes con el fin de extraer características específicas

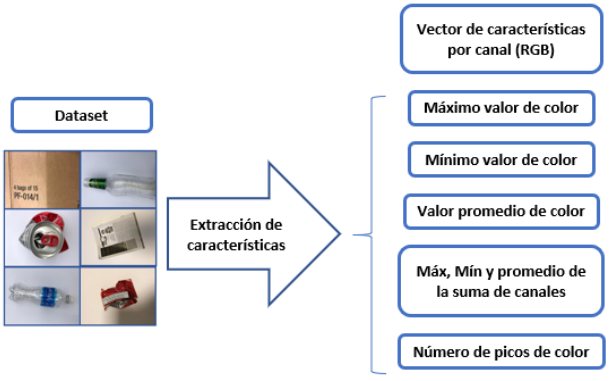


Figura 11: Proceso de transformación de imágenes.

Estas características se han conseguido

procesando las imágenes por cada canal (RGB) mediante su histograma de frecuencias correspondiente de cada tipo de residuo.

- valores minimos de color en el canal rojo, verde y azul
- valores maximos de color para el canal rojo, verde y azul
- número de picos de frecuencia (valores máximos y mínimos) para cada canal rojo, verde y azul
- valor promedio de color del histograma de frecuencias sumado por cada canal rojo, verde y azul

Primero, para hallar el vector que sirve como entrada a la gráfica del histograma se utilizó la siguiente función

```
def hist0(G):
    delF = 6
    hist = cv2.calcHist([G],[0],None,[256],[0,256])
    hist = np.array(hist).T[0]
    for i in range(delF): hist[i] = 0

    # frecuencia relativa

    return hist
```

Luego, se divide la imagen en los tres canales RGB para poder proceder a hallar el concatenado y la suma del mismo:

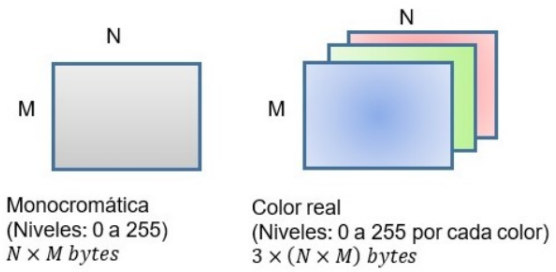


Figura 12. Canales RGB

```
I = cv2.imread(mypath + '\\' + f)  # lee imagen RGB

b,g,r = cv2.split(I)                       # get b, g,

histR=  hist0(r)
histG=  hist0(g)
histB=  hist0(b)
histS = histB + histG + histR
```

El resultado obtenido es el siguiente:

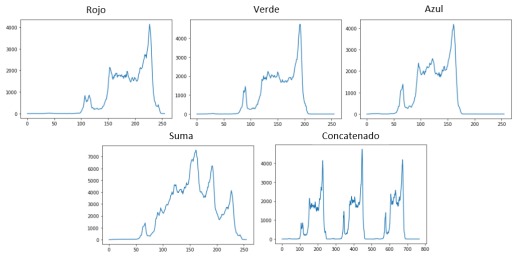


Figura 13: Histogramas de las fotos.

Para poder separar las imágenes de acuerdo a la clase a la que pertenecían y asignarles un código de clase (según Figura 10) para el dataset se utilizó el siguiente código:

```
def cardboard():
    return 0
def glass():
    return 1
def metal():
    return 2
def paper():
    return 3
def plastic():
    return 4
def trash():
    return 5
def default():
    return "Incorrect"

switcher = {
    "cardb": cardboard,
    'glass': glass,
    'metal': metal,
    'paper': paper,
    'plast': plastic,
    'trash': trash
    }

def ClaseImg(claseImg):
    return switcher.get(claseImg[:5], default)()
```

Para poder hallar el máximo, mínimo y el promedio de los valores de los histogramas se crearon las siguientes funciones de la librería numpy:

```
MaxValorConcat= np.max(histC)
MinValorConcat= np.min(histC)
PromedioConcat = np.average(histC)
```

Como los valores anteriores son puntuales, se decidió cambiar de un histograma a un gráfico de barras acumulativo en cinco

rangos; lo que se requiere es el valor de cada una de esas barras, para ello se creó la siguiente función:

```
def ObtenerValBarras(histCx):
    plt.hist(histCx, 5, [0, 256])

    ax= plt.gca()
    p = ax.patches
    Y1=p[-5].get_height()
    Y2=p[-4].get_height()
    Y3=p[-3].get_height()
    Y4=p[-2].get_height()
    Y5=p[-1].get_height()
    print(Y1," ", Y2, " ", Y3," ", Y4, " ", Y5," ")

    return Y1,Y2,Y3,Y4,Y5
```

la cual se llamó de la siguiente manera:

```
Yc1,Yc2,Yc3,Yc4,Yc5 =ObtenerValBarras(histC
Ys1,Ys2,Ys3,Ys4,Ys5 =ObtenerValBarras(histS
Yr1,Yr2,Yr3,Yr4,Yr5 =ObtenerValBarras(histR
Yg1,Yg2,Yg3,Yg4,Yg5 =ObtenerValBarras(histG
Yb1,Yb2,Yb3,Yb4,Yb5 =ObtenerValBarras(histB
```

también se logró identificar que el número de picos del histograma podría ser un valor importante de diferenciación, para ello se creó la siguiente función:

```
def find_peaks(a):
  x = np.array(a)
  max = np.max(x)
  lenght = len(a)
  ret = []
  for i in range(lenght):
      ispeak = True
      if i-1 > 0:
          ispeak &= (x[i] > 1 * x[i-1])
      if i+1 < lenght:
          ispeak &= (x[i] > 1 * x[i+1])

      ispeak &= (x[i] > 0.05 * max)
      if ispeak:
          ret.append(i)
  return len(ret)
```

La cual se llamó para los cinco histogramas desarrollados:

```
NumeroDePicosC=find_peaks(histC)
NumeroDePicosS=find_peaks(histS)
NumeroDePicosR=find_peaks(histR)
NumeroDePicosG=find_peaks(histG)
NumeroDePicosB=find_peaks(histB)
```

Una vez obtenido estos datos pasamos a generar nuestra dataset, mediante las siguientes líneas de código:

```
    data.append([f, Rows, Cols, Mats, clase,
    NumSegmentos, MaxValorConcat, MinValorConcat, PromedioConcat,
    MaxValorSuma, MinValorSuma, PromedioSuma,
    MaxValorRojo, MinValorRojo, PromedioRojo,
    MaxValorVerde, MinValorVerde, PromedioVerde,
    MaxValorAzul, MinValorAzul, PromedioAzul,
    Yc1,Yc2,Yc3,Yc4,Yc5,Ys1,Ys2,Ys3,Ys4,Ys5,Yr1,Yr2,Yr3,Yr4,Yr5,
    Yg1,Yg2,Yg3,Yg4,Yg5,Yb1,Yb2,Yb3,Yb4,Yb5,
    NumeroDePicosC,NumeroDePicosS,NumeroDePicosR,NumeroDePicosG,NumeroDePicosB])

title  = ['Name', 'Rows', 'Cols', 'Mats',  'clase' ,'Num Segmentos',
'MaxValor Concatenado' , 'MinValor Concatenado', 'Promedio Concatenado',
'MaxValor Suma' , 'MinValor Suma', 'Promedio Suma',
'MaxValor Rojo' , 'MinValor Rojo', 'Promedio Rojo',
'MaxValor Verde' , 'MinValor Verde', 'Promedio Verde',
'MaxValor Azul' , 'MinValor Azul', 'Promedio Azul',
'YC1','YC2','YC3','YC4','YC5','YS1','YS2','YS3','YS4',
'YS5','YR1','YR2','YR3','YR4','YR5','YG1','YG2','YG3','YG4','YG5',
'YB1','YB2','YB3','YB4','YB5',"Numero de Picos Concatenado",
"Numero de Picos Suma","Numero de Picos R","Numero de Picos G","Numero de Picos B"]

dataset= pd.DataFrame(data=data, columns = title)

dataset.to_csv('DatasetV2.csv')
```

6.2 Preparación de Datos

Para la etapa de preparación de datos nos enfocaremos en estandarizar los valores para que sus valores no se diferencien demasiado, ya que influirá de manera negativa en la precisión del modelo machine learning al cual va a ser aplicado.

Siguiendo la metodología CRISP-DM la preparación de datos incluye la selección y limpieza de los datos; para ello primero verificamos si el dataset incluye valores nulos:

```
df.isna().sum()
```

```
Unnamed: 0                  0
Name                        0
Rows                        0
Cols                        0
Mats                        0
clase                       0
Posicion                    0
Grados Rotacion             0
Ultima Letra                0
Num Segmentos               0
MaxValor Concatenado        0
MinValor Concatenado        0
Promedio Concatenado        0
MaxValor Suma               0
MinValor Suma               0
Promedio Suma               0
MaxValor Rojo               0
MinValor Rojo               0
Promedio Rojo               0
MaxValor Verde              0
MinValor Verde              0
Promedio Verde              0
MaxValor Azul               0
MinValor Azul               0
Promedio Azul               0
```

Figura 14: Verificación de valores nulos .

Como se observa, ninguna columna contiene algún valor nulo.

Luego de esto eliminamos las columnas con valores constantes que son las columnas "Rows", "Cols" y "Mats" . Además también se eliminan las columnas de valor mínimo ya que todas las filas tienen valor "0", estas columnas son: "MinValor Concatenado", "MinValor Suma", "MinValor Rojo", "MinValor Verde" y "MinValor Azul".

```
del df['Rows']
del df['Cols']
del df['Mats']
del df['MinValor Concatenado']
del df['MinValor Suma']
del df['MinValor Rojo']
del df['MinValor Verde']
del df['MinValor Azul']
```

Para completar con tener todos los valores numéricos, eliminamos la columna “Name”:

```
del df2['Name']
```

Ahora, se procede a normalizar las columnas, sin embargo no se puede proceder a normalizar todas las columnas indiscriminadamente, ya que se tiene que analizar los datos para ver si la normalización mantiene los niveles de variabilidad de frecuencia[4]. Así que se graficaron los histogramas de las columnas:

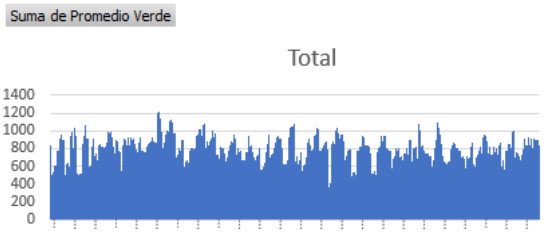


Figura 17: Histograma una columna promedio

y los valores normalizados:

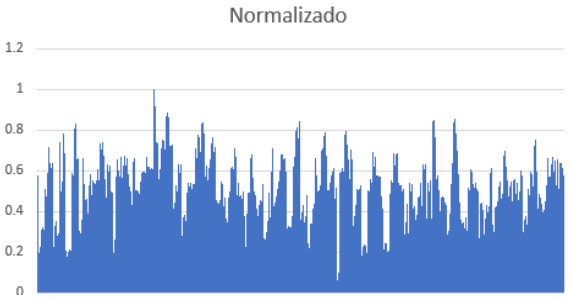


Figura 18: histograma normalizado una columna promedio

Como se puede observar, la variabilidad no se ve afectada. Luego de este análisis se decidió que las columnas a normalizar son las que tienen los números de picos para cada histograma:

```
DataANormalizar = df2[['Numero de Picos Concatenado',
                       'Numero de Picos Suma',
                       'Numero de Picos R',
                       'Numero de Picos G',
                       'Numero de Picos B']]
```

Bajo el mismo análisis se decidió aplicar logaritmo natural a las columnas que tienen valores altos:

```
DataALogaritmoN = df2[['MaxValor Concatenado',
                       'Promedio Concatenado',
                       'MaxValor Suma','Promedio Suma
                       'MaxValor Rojo','Promedio Rojo
                       'MaxValor Verde','Promedio Ver
                       'MaxValor Azul','Promedio Azul
```

```
DataALogaritmoN['MaxValor Concatenado'] = np.log(DataALogaritmoN['MaxValor Concate
DataALogaritmoN['Promedio Concatenado'] = np.log(DataALogaritmoN['Promedio Concate
DataALogaritmoN['MaxValor Suma'] = np.log(DataALogaritmoN['MaxValor Suma'])
DataALogaritmoN['Promedio Suma'] = np.log(DataALogaritmoN['Promedio Suma'])
DataALogaritmoN['MaxValor Rojo'] = np.log(DataALogaritmoN['MaxValor Rojo'])
DataALogaritmoN['Promedio Rojo'] = np.log(DataALogaritmoN['Promedio Rojo'])
DataALogaritmoN['MaxValor Verde'] = np.log(DataALogaritmoN['MaxValor Verde'])
DataALogaritmoN['Promedio Verde'] = np.log(DataALogaritmoN['Promedio Verde'] )
DataALogaritmoN['MaxValor Azul'] = np.log(DataALogaritmoN['MaxValor Azul'])
DataALogaritmoN['Promedio Azul'] = np.log(DataALogaritmoN['Promedio Azul'])
DataALogaritmoN.head()
```

Por último se añaden más columnas que son el producto de los valores máximos con los promedios

```
df3['Producto Concatenado'] = df3['MaxValor Concatenado']* df3['Promedio Concat
df3['Producto Suma'] = df3['MaxValor Suma']* df3['Promedio Suma']
df3['Producto Rojo'] = df3['MaxValor Rojo']* df3['Promedio Rojo']
df3['Producto Verde'] = df3['MaxValor Verde']* df3['Promedio Verde']
df3['Producto Azul'] = df3['MaxValor Azul']* df3['Promedio Azul']
```

Luego de esto se procederá a almacenar todos los procedimientos hechos hasta ahora

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

```
Mounted at drive
```

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

```
Drive already mounted at /content/drive; to attempt to for
```

```
[ ] df3.to_csv('DatasetFinal.csv')
    !cp data.csv "drive/My Drive/FinalIaData"
```

```
cp: cannot stat 'data.csv': No such file or directory
```

```
[ ]
```

6.3 Entrenamiento de los Modelos

- Árbol de Decisión

  Un árbol de decisión en Machine Learning es una estructura de árbol similar a un diagrama de flujo donde un nodo interno representa una característica (o atributo), la rama representa una regla de decisión y cada nodo hoja representa el resultado.

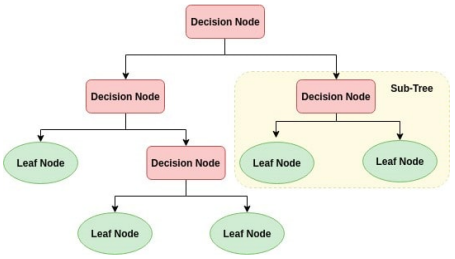


Figura 19. Estructura del Árbol de decisión

Teniendo en cuenta el dataset resultado de la preparación de datos en el archivo DatasetFinalV2.csv haremos uso del algoritmo del árbol de decisión.

Para ello leemos el archivo:

```
import pandas as pd
df = pd.read_csv("DatasetFinalV2.csv",sep=",")
df.head()
```

Lo cual nos da como resultado la vista de la estructura de nuestro dataset en el que se puede apreciar algunos de los atributos:

	clase	Num Segmentos	YC1	YC2	YC3	YC4	YC5	YS1	YS2	YS3	...
0	0	979	371.0	5.0	6.0	6.0	22.0	62.0	3.0	2.0	...
1	0	539	178.0	136.0	66.0	23.0	20.0	31.0	4.0	2.0	...
2	0	194	159.0	106.0	72.0	27.0	8.0	23.0	5.0	2.0	...
3	0	406	221.0	69.0	26.0	20.0	11.0	11.0	3.0	2.0	...
4	0	368	220.0	44.0	15.0	11.0	13.0	49.0	10.0	6.0	...

Figura 20 . Estructura de dataset

Definimos el target y realizamos la división de los datos del dataset para entrenar el modelo y para hacer las pruebas posteriormente, en este caso optamos por tener el 30% de los datos para la prueba y el 70% de los datos para el entrenamiento.

```
from sklearn.model_selection import train_test_split
#Definimos los datasets X e y con la variable target y los atributos
yp = df["clase"]
Xp = df.drop(["clase"],axis=1)

X_train, X_test, y_train, y_test = train_test_split(Xp, yp, test_size=0.3)
```

Importamos la librería sklearn para

poder uso de la clase DecisionTreeClassifier, en la cual vamos a establecer los parámetros de **criterio** en "entropía" para medir la calidad de una división teniendo en cuenta la ganancia de información y **max_depth** que irá variando de acuerdo a la profundidad máxima del árbol que deseemos.

```
from sklearn import tree
from sklearn.metrics import accuracy_score, confusion_matrix

tree_model = tree.DecisionTreeClassifier(criterion='entropy',max_depth
tree_model = tree_model.fit(X_train, y_train)
```

Mostramos la matriz de confusión para nuestro caso:

```
import matplotlib.pyplot as plt
from sklearn.metrics         import plot_confusion_matri
#Predecir en el test set
y_pred = tree_model.predict(X_test)

# Revisar la matriz de confusión ([TN,FN],[FP,TP])
confusion_matrix(y_test,y_pred)
```

```
plot_confusion_matrix(tree_model, X_test, y_test
plt.show()
```

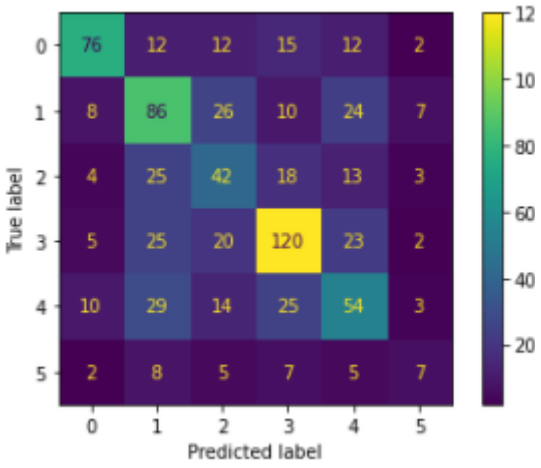


Figura 21. Matriz de confusión nivel de profundidad 10

Mostramos un reporte de las métricas resultantes como recall y precision.

```
from sklearn.metrics import classification_report
print(classification_report(y_test, y_pred))
```

Finalmente, mostramos de manera gráfica el resultado obtenido, el modelo árbol de decisión:

```
import graphviz
dot_data = tree.export_graphviz(tree_model,
                                out_file=None,
                                filled=True,
                                rounded=True,
                                max_depth = 2,
                                special_characters=True,
                                class_names = ['cardboard','glass','metal','paper','plastic''trash'],
                                feature_names = X_train.columns)
graph = graphviz.Source(dot_data)
graph
```

La salida del código anterior se muestra en el Anexo 1.

- Random Forest

En machine learning, el “random forest” (bosque aleatorio) es un algoritmo de aprendizaje supervisado. El "bosque" que construye es un conjunto de árboles de decisión

El bosque aleatorio crea múltiples árboles de decisión y los fusiona para obtener una predicción más precisa y estable.

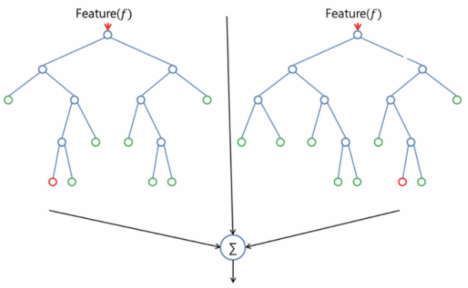


Figura 22. *Random forest con 2 árboles*

Tomando en cuenta el dataset “DatasetFinal.csv que fue resultado de la preparación de datos, haremos uso del algoritmo random forest.

Para ello primero importamos las librerias necesarias

```
# importanto las librerias necesarias
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
from google.colab import files
```

mostramos la estructura de nuestro dataset

```
df3 = pd.read_csv("/content/DatasetFinal.csv",sep=",")
df3.head()
```

	Unnamed: 0	Grados Rotacion	Num Segmentos	YC1	YC2	YC3	YC4	YC5	YS1	YS2	YS3	YS4	YS5	YR1	YR2
0	0	0	371	238.0	35.0	31.0	18.0	38.0	12.0	6.0	5.0	3.0	3.0	78.0	11.0
1	1	0	267	186.0	40.0	33.0	36.0	70.0	8.0	3.0	3.0	3.0	1.0	73.0	14.0
2	2	0	198	141.0	33.0	25.0	15.0	28.0	8.0	6.0	3.0	2.0	1.0	55.0	7.0
3	3	105	222	215.0	44.0	77.0	71.0	51.0	8.0	6.0	5.0	2.0	0.0	80.0	18.0
4	4	120	179	201.0	42.0	62.0	81.0	50.0	6.0	5.0	3.0	4.0	1.0	81.0	15.0

establecemos nuestros ejes

```
dfRf =df3.copy()
y = dfRf['ClaseNumerica']
X = dfRf.drop('ClaseNumerica', axis=1)
```

Mostrando un conteo de datos por cada clase, para este caso:

clase	descripcion
0	saludable
1	oscuro
2	mancha
3	moho
4	verdeada
5	enfermedad
6	gangrena

Figura 23. *Descripción de las clases*

```
y.value_counts()
```

```
0    809
2    591
3    305
1    287
6    257
5    247
4    194
Name: ClaseNumerica, dtype: int64
```

Dividimos las funciones y el objetivo en conjuntos de prueba y de entrenamiento, además establecemos el parámetro **“test_size”** para probar la precision del algoritmo **“RandomForestClassifier()”** con el conjunto de prueba y entrenamiento generado anteriormente

```
# Dividiendo las funciones y el objetivo en conjuntos de prueba y de entrenamiento
X_train, X_test, y_train, y_test = train_test_split(X, y,test_size=0.3, random_state=1, stratify=y)

# instanciar y ajustar mediante RandomForestClassifier
forest = RandomForestClassifier()
forest.fit(X_train, y_train)
```

```
RandomForestClassifier(bootstrap=True, ccp_alpha=0.0, class_weight=None,
                       criterion='gini', max_depth=None, max_features='auto',
                       max_leaf_nodes=None, max_samples=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, n_estimators=100,
                       n_jobs=None, oob_score=False, random_state=None,
                       verbose=0, warm_start=False)
```

luego:

```
# Hacer predicciones para el conjunto de prue
y_pred_test = forest.predict(X_test)
```

```
# ver la puntuacion de precision
accuracy_score(y_test, y_pred_test)
```

```
0.9392812887236679
```

Luego mostramos la matriz de confusión para el modelo Random Forest

```
# Ver matriz de confusión para datos de prueba y predicciones
confusion_matrix(y_test, y_pred_test)
```

```
array([[242,   0,   0,   0,   0,   1,   0],
       [  2,  70,  13,   0,   1,   0,   0],
       [  1,   6, 166,   1,   0,   1,   2],
       [  2,   0,   5,  82,   0,   0,   3],
       [  0,   0,   0,   0,  58,   0,   0],
       [  6,   1,   2,   0,   0,  65,   0],
       [  0,   0,   1,   0,   0,   1,  75]])
```

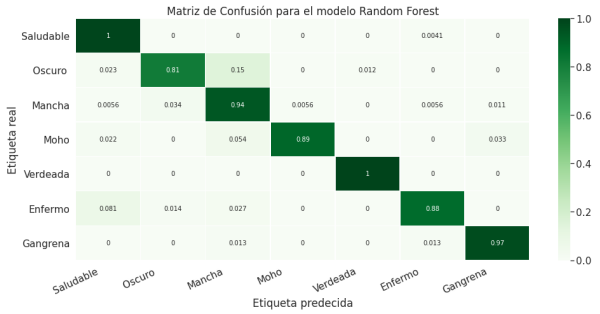


Figura 24. Matriz de confusión Random Forest

por último mostramos un informe de clasificación de para los datos de prueba y entrenamiento

```
print(classification_report(y_test, y_pred_test))
```

	precision	recall	f1-score	support
0	0.96	1.00	0.98	24
1	0.91	0.81	0.86	8
2	0.89	0.94	0.91	17
3	0.99	0.89	0.94	9
4	0.98	1.00	0.99	5
5	0.96	0.88	0.92	7
6	0.94	0.97	0.96	7
accuracy			0.94	80
macro avg	0.95	0.93	0.94	80
weighted avg	0.94	0.94	0.94	80

6.4 Evaluación de los Modelos

- Árbol de Decisión

  El árbol de decisión permite evaluar los valores de los atributos partiendo del más relevante entre ellos, tomando como ejemplo el nivel 1 de profundidad, podemos observar la siguiente matriz de confusión en la cual se observa que teniendo en cuenta solo uno de los atributos el modelo predice valores solo en el estado 0 (cardboard) y 1 (glass).

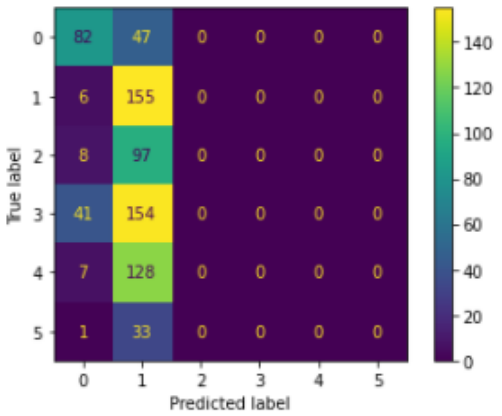


Figura 25. Matriz de confusión Árbol de Decisión

Tenemos como resultado el siguiente árbol

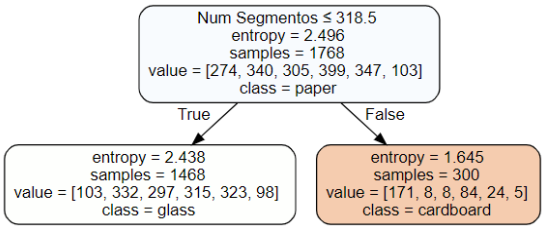


Figura 24. Árbol resultante nivel 1

Además de que los valores de *recall* y *precisión* de este caso:

	PRECISION	RECALL
Cardboard	0.57	0.64
Glass	0.25	0.96

$$RECALL = VP/(VP + FN)$$

$$RECALL = 82/(82 + 47) = 0.64$$

$$PRECISION\ (VPP) = VP/(VP + FP)$$

$$PRECISION\ (VPP) = 82/(82 + 6 + 8 + 41 + 7 + 1)$$

$$PRECISION\ (VPP) = 0.57$$

- Random Forest

  En el algoritmo random forest podemos variar el atributo test_size para obtener diferentes niveles de precisión a la hora de poder mostrar resultados en la predicción

  Por ejemplo, para un test_size = 0.3, obtenemos la siguiente matriz de confusión

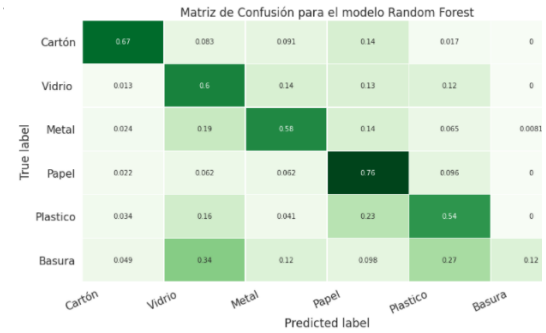


Figura 26. ***Matriz de Confusión Random Forest***

Tal como se ve en la imagen, las imágenes de la clase “Basura” son normalmente confundidas con imágenes correspondientes a la categoría vidrio, por lo que hay una debilidad en el modelo, producto del no balanceo de las imágenes.

Además de que sus valores de precision y recall para cada una de las clases en este caso resultan:

	precision	recall
0	0.88	0.69
1	0.50	0.59
2	0.58	0.56
3	0.59	0.76
4	0.58	0.54
5	0.60	0.07

Obteniendo un accuracy de 60.34%

- Redes Neuronales Convolucionales

Como se ha podido notar, el porcentaje del accuracy de los modelos anteriormente evaluados es bajo, por lo que se probará con ingresar las imágenes directamente a un modelo de redes neuronales convolucionales.

Luego de revisar las características de las imágenes con que se va a trabajar (para más detalle ver Anexo N° 3 - Entrenamiento de CNN) se procede a construir la red neuronal tomando como base el modelo ResNet.

```
# instanciando el modelo
input_shape = (256,256,3)
base_model = tf.keras.applications.ResNet50V2(include_top=False, input_shape=input_shape)

#hacer que las capas del modelo se puedan entrenar para un ajuste fino
base_model.trainable = True
```

Posteriormente se crean las etapas de aumento de datos y se realiza la arquitectura de la red neuronal.

Finalmente, se entrenó el modelo en 15 épocas, siendo la número 14 en donde se llegó a un máximo valor de accuracy: 89.50%

```
Epoch 1/15
64/64 [==============================] - 1079s 17s/step - loss: 1.6176 - accuracy: 0.3492 - val_loss: 1.1263 - val_accuracy: 0.5921
Epoch 2/15
64/64 [==============================] - 821s 13s/step - loss: 1.1239 - accuracy: 0.6217 - val_loss: 0.8273 - val_accuracy: 0.7050
Epoch 3/15
64/64 [==============================] - 732s 11s/step - loss: 0.8872 - accuracy: 0.7132 - val_loss: 0.6745 - val_accuracy: 0.7624
Epoch 4/15
64/64 [==============================] - 816s 13s/step - loss: 0.7273 - accuracy: 0.7641 - val_loss: 0.5871 - val_accuracy: 0.8099
Epoch 5/15
64/64 [==============================] - 766s 12s/step - loss: 0.6403 - accuracy: 0.7948 - val_loss: 0.5324 - val_accuracy: 0.8317
Epoch 6/15
64/64 [==============================] - 730s 11s/step - loss: 0.5500 - accuracy: 0.8284 - val_loss: 0.4910 - val_accuracy: 0.8396
Epoch 7/15
64/64 [==============================] - 715s 11s/step - loss: 0.4928 - accuracy: 0.8536 - val_loss: 0.4524 - val_accuracy: 0.8574
Epoch 8/15
64/64 [==============================] - 756s 12s/step - loss: 0.4473 - accuracy: 0.8586 - val_loss: 0.4304 - val_accuracy: 0.8713
Epoch 9/15
64/64 [==============================] - 778s 12s/step - loss: 0.3977 - accuracy: 0.8803 - val_loss: 0.4062 - val_accuracy: 0.8812
Epoch 10/15
64/64 [==============================] - 732s 11s/step - loss: 0.3754 - accuracy: 0.8927 - val_loss: 0.3860 - val_accuracy: 0.8772
Epoch 11/15
64/64 [==============================] - 747s 12s/step - loss: 0.3376 - accuracy: 0.8996 - val_loss: 0.3702 - val_accuracy: 0.8733
Epoch 12/15
64/64 [==============================] - 773s 12s/step - loss: 0.3152 - accuracy: 0.9095 - val_loss: 0.3602 - val_accuracy: 0.8812
Epoch 13/15
64/64 [==============================] - 748s 12s/step - loss: 0.2999 - accuracy: 0.9095 - val_loss: 0.3431 - val_accuracy: 0.8911
Epoch 14/15
64/64 [==============================] - 763s 12s/step - loss: 0.2780 - accuracy: 0.9130 - val_loss: 0.3336 - val_accuracy: 0.8950
Epoch 15/15
64/64 [==============================] - 745s 12s/step - loss: 0.2498 - accuracy: 0.9303 - val_loss: 0.3266 - val_accuracy: 0.8911
```

En la siguiente imagen se muestran las gráficas de la evolución del accuracy y del valor perdido:

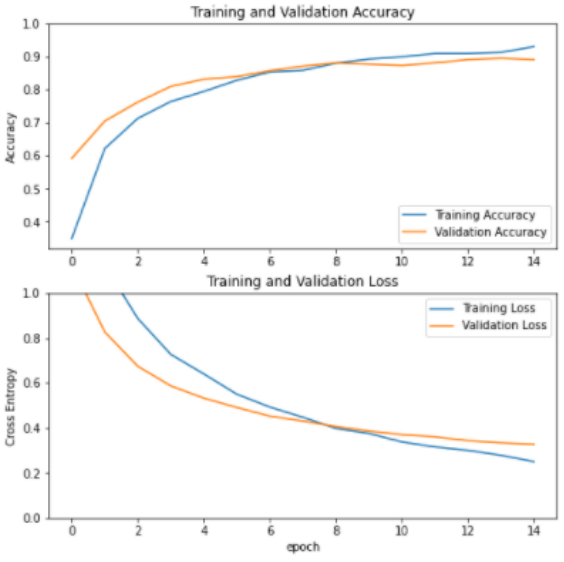


Figura 27. ***Gráficas de evolución de Accuracy y valor perdido***

6.5 Optimización

- Árbol de Decisión

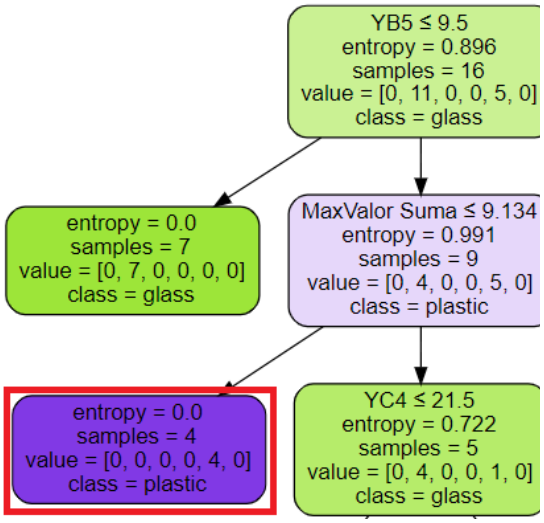
Teniendo en cuenta la división realizada del dataset, se realiza el entrenamiento del modelo con diferentes niveles de profundidad. En el árbol podemos observar las variaciones de entropía, lo cual se corrobora con las variaciones de los valores de *precision* y *recall* para poder definir la profundidad óptima del árbol.

Tras realizar el análisis para los niveles desde 4 a 15 podemos notar lo siguiente:

Para un nivel de profundidad de 10

	PRECISION	RECALL
0	0.72	0.59
1	0.46	0.53
2	0.35	0.40
3	0.62	0.62
4	0.41	0.40
5	0.29	0.21

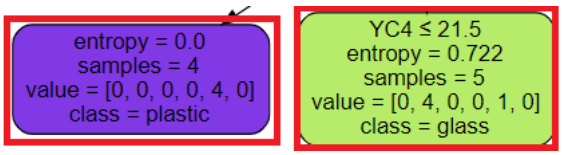
ACCURACY	**0.51**



Al analizar la siguiente salida, notamos que la entropía es 0, lo cual nos indica que en esa rama de decisión se alcanzó la máxima ganancia de información.

En la segunda hoja observamos que para los 5 registros existentes en esa rama, 4 pertenecen a la clase Glass y 1 pertenece a la clase Plastic, mientras que en la primera hoja todos pertenecen a la clase Plastic.

Donde **Values=[cardboard, glass, metal, paper, plastic, trash]** indica la cantidad de registros analizados pertenecientes a la clase indicada.



Para un nivel de profundidad de 15:

	PRECISION	RECALL
0	0.65	0.57
1	0.46	0.41
2	0.33	0.37
3	0.56	0.61
4	0.38	0.40
5	0.28	0.26

ACCURACY	**0.47**

Al observar los valores para los demás niveles de profundidad notamos que los valores de *precisión* y *recall* van disminuyendo mientras se agregan más divisiones, razón por la cual optamos por el nivel de profundidad de 10, ya que es el que mejores valores resultantes de VP y VPP tienen.

- Random Forest

  Para el uso del modelo random forest se hace uso de diferentes valores del tamaño de prueba test_size

  test_size= 0.4

	precision	recall
0	0.84	0.62
1	0.53	0.61
2	0.60	0.63
3	0.60	0.74
4	0.59	0.56
5	0.50	0.09

Accuracy: 60.83%

test_size=0.25

	precision	recall
0	0.82	0.66
1	0.53	0.58
2	0.56	0.62
3	0.62	0.77
4	0.65	0.59
5	0.86	0.18

Accuracy: 62.34%

test_size=0.2

	precision	recall
0	0.84	0.70
1	0.53	0.64
2	0.58	0.56
3	0.65	0.77
4	0.65	0.62
5	1.00	0.15

Accucary: 63.83%

Como se puede observar, el valor del porcentaje de entrenamiento con el que se obtuvo una mayor precisión es el test_size = 0.2 no obstante, el porcentaje es muy bajo para el objetivo planteado.

6.6 Integración con Raspberry Pi

Una vez finalizada la evaluación de los modelos, trabajaremos con la CNN ya que nos brinda un mayor accuracy.

Primero guardamos el modelo en una carpeta

```
# Guardar el Modelo
#model.save('path_to_my_model.h5')
model.save( '../content/drive/MyDrive/Colab Notebooks/composnet/modelo_entrenado.h5 ')
```

posteriormente en el raspberry se carga el modelo y se especifica la dirección donde se almacena la foto tomada por la cámara del raspberry:

```
import numpy as np
import tensorflow as tf
from tensorflow.keras.applications.resnet50 import preprocess_input
from tensorflow.keras.preprocessing import image
print("MODELOOOO=")

nuevo_modelo = tf.keras.models.load_model('./modelo_entrenado.h5')
# create a test dataset
test_ds = []
image_size = (256, 256)
for i in range(len(test_labels)):
    #path = tf.keras.utils.get_file(str(i), origin=test_urls[i])
    path1 = "./images/descarga.jpeg"
    path2 = "./images/descarga2.jpeg"
    img1 = tf.keras.preprocessing.image.load_img(path1, target_size=image_size)
    img2 = tf.keras.preprocessing.image.load_img(path2, target_size=image_size)

    #img = tf.keras.preprocessing.image_dataset_from_directory(path,target_size=
    test_ds.append(tf.keras.preprocessing.image.img_to_array(img1))
    test_ds.append(tf.keras.preprocessing.image.img_to_array(img2))
test_ds = np.array(test_ds)
```

Finalmente, en la consola se mostrarán los resultados:

```
MODELOOOO=
2021-08-06 19:01:20.159582: I tensorflow/compiler/mlir/mlir_graph_optimization_pass.
cc:116] None of the MLIR optimization passes are enabled (registered 2)
2021-08-06 19:01:20.189294: W tensorflow/core/platform/profile_utils/cpu_utils.cc:11
6] Failed to find bogomips or clock in /proc/cpuinfo; cannot determine CPU frequency
Inference:paper, 98.42% Confidence
Real Label:paper
```

Como se observa en la imagen anterior, se identificó correctamente a la imagen correspondiente a la clase papel con un nivel de 98.42% de confianza.

7. CONCLUSIONES

- Se logró realizar un sistema que permite la clasificación de residuos, obteniendo buenos resultados.
- El modelo de árbol de decisión permite identificar la importancia de variables a partir del análisis de muchos atributos presentes en el dataset, teniendo en cuenta la entropía y la ganancia de información lo cual además se puede observar de manera gráfica.
- El conjunto de imágenes fueron sometidos a tres diferentes algoritmos, el Árbol de Decisiones brindó valores de accuracy de 51%, Random Forest obtuvo un 62.02%, mientras que con el algoritmo CNN se obtuvo un porcentaje de 89%
- Gracias al aumento de datos y el aprendizaje de transferencia que se establecieron , se logró mejorar mejorar las funciones utilizadas para entrenar el modelo alcanzando aproximadamente un accuracy de casi 90%.
- Para la fase de entrenamiento del modelo se considero un epochs =15 (iteraciones) ya que era el valor mínimo en el que la precisión del modelo no mejoraba de manera significativa

9. REFERENCIAS

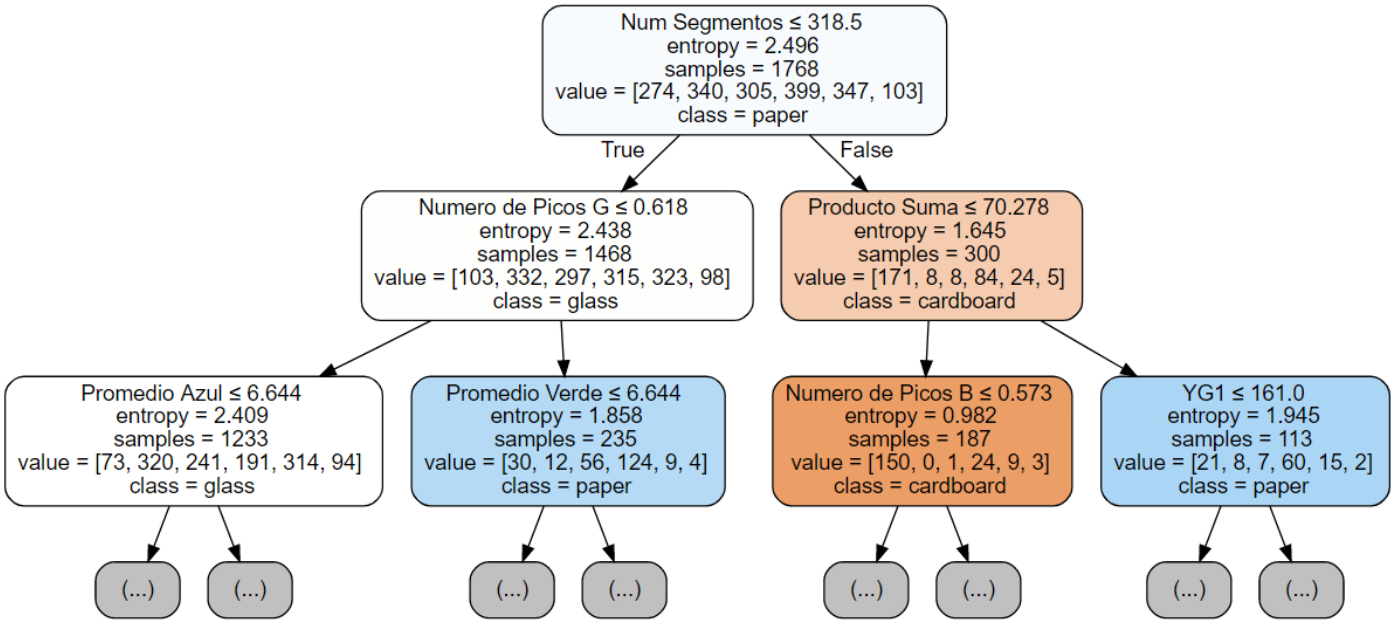
[1] IBM. (2012) Manual CRISP-DM de IBM SPSS.

[2] José Hernández Orallo, M.José Ramírez Quintana, and César Ferri Ramírez, Introducción a la Minería de Datos.: Editorial Pearson, 2004.

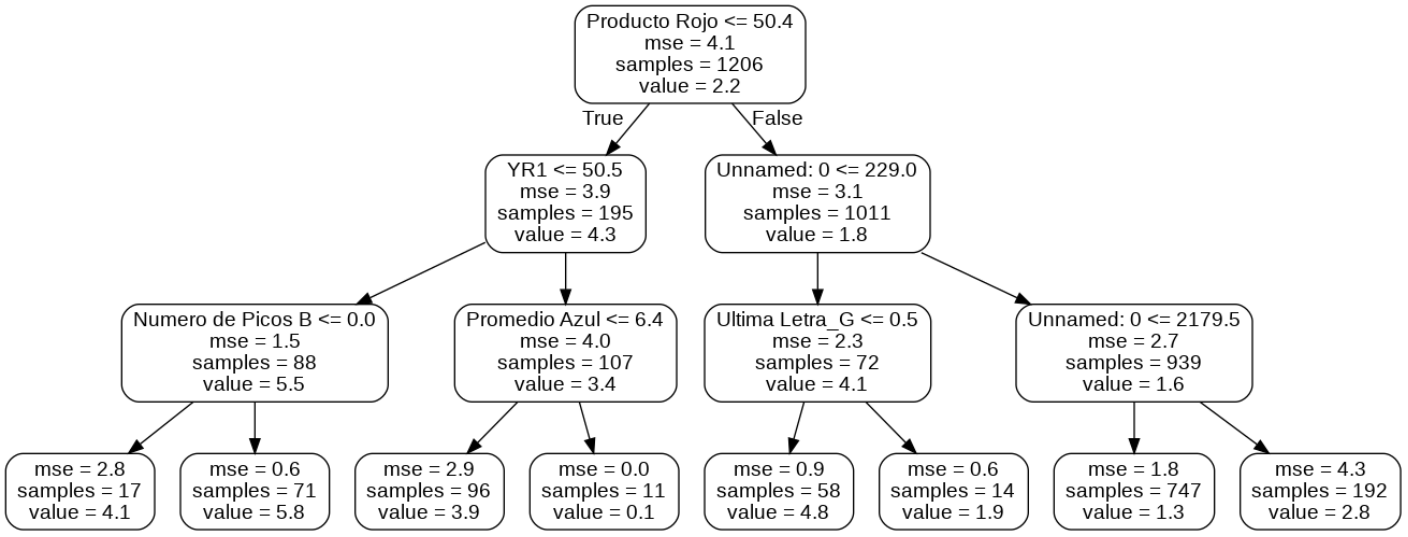
[3] AprendeIA (2019). Cambiando variables categóricas a numéricas.

[4] Cendrero, S. M. (2021, 16 junio). Precauciones a la hora de normalizar datos en Data Science - Think Big Empresas. Think Big. https://empresas.blogthinkbig.com/precauciones-la-hora-de-normalizar/

ANEXO 1: ÁRBOL DE DECISIONES - VISTA NIVEL PROFUNDIDAD 2



ANEXO 2: RANDOM FOREST - VISTA LIMITADA A TRES NIVELES



ANEXO 3: CÓDIGO FUENTE:

- Preparación del Dataset

```
from os import listdir
from os.path import isfile, join
import pandas as pd
import cv2
import numpy      as np
import matplotlib as mlp
from   matplotlib import pyplot as plt




##############################################################################################
##33
mypath = './images'

files = [f for f in listdir(mypath) if isfile(join(mypath, f))]

data = []           # data
NumImagen =0
def hist0(G):
    delF = 6
    hist = cv2.calcHist([G],[0],None,[256],[0,256])    # calcula histograma
    hist = np.array(hist).T[0]                         # extrae histograma
```

```
    for i in range(delF): hist[i] = 0                    # eliminamos primeros

    # frecuencia relativa

    return hist

def hist1(G):
    delF = 6
    hist = cv2.calcHist([G],[0],None,[256],[0,256])    # calcula histograma
    hist = np.array(hist).T[0]                          # extrae histograma
    for i in range(delF): hist[i] = 0                    # eliminamos primeros

    # frecuencia relativa
    Sum  = sum(hist) + 0.0
    hist = [round(s/Sum,4) for s in hist]

    hist = np.array(hist)
    return hist

def find_peaks(a):
  x = np.array(a)
  max = np.max(x)
  lenght = len(a)
  ret = []
  for i in range(lenght):
      ispeak = True
      if i-1 > 0:
```

```
            ispeak &= (x[i] > 1 * x[i-1])
        if i+1 < lenght:
            ispeak &= (x[i] > 1 * x[i+1])

        ispeak &= (x[i] > 0.05 * max)
        if ispeak:
            ret.append(i)
    return len(ret)

def ContarSegmentos(treshImage):
    cnts = cv2.findContours(treshImage, cv2.RETR_LIST, cv2.CHAIN_APPROX_SIMPLE)[-2]
    ## filter by area
    s1= 3
    s2 = 20
    xcnts = []
    for cnt in cnts:
        if s1<cv2.contourArea(cnt) <s2:
            xcnts.append(cnt)

    return len(xcnts)
def closePlots():
    plt.clf()
    plt.cla()
    plt.close("all")
def ObtenerValBarras(histCx):
    plt.hist(histCx, 5, [0, 256])
```

```
    ax= plt.gca()
    p = ax.patches
    Y1=p[-5].get_height()
    Y2=p[-4].get_height()
    Y3=p[-3].get_height()
    Y4=p[-2].get_height()
    Y5=p[-1].get_height()
    print(Y1," ", Y2, " ", Y3," ", Y4, " ", Y5," ")
    #for i in range(len(p)):
       # print(len(p))
       # print(p[i])

    return Y1,Y2,Y3,Y4,Y5




for f in files:

    I = cv2.imread(mypath + '\\' + f)  # lee imagen RGB
    G = cv2.cvtColor(I, cv2.COLOR_BGR2GRAY)
     thresh = cv2.adaptiveThreshold(G, 255, cv2.ADAPTIVE_THRESH_MEAN_C, cv2.THRESH_BINARY,
101, 3)
    NumSegmentos=ContarSegmentos(thresh)
    [Rows, Cols, Mats] = I.shape
```

```
    b,g,r = cv2.split(I)                       # get b, g, r

    histR=  hist0(r)
    histG=  hist0(g)
    histB=  hist0(b)
    histS = histB + histG + histR
    histC = np.concatenate((histB, histG,histR), axis=0)
    clase = f.split('_')[1]
    Posicion = f.split('_')[2]
    Grados_rotacion = f.split('_')[3]
    UltimaLetra = f.split('_')[4]
    UltimaLetra = UltimaLetra.split(".")[0]
    #-------------------------------------------------
    MaxValorConcat= np.max(histC)
    MinValorConcat= np.min(histC)
    PromedioConcat = np.average(histC)
    #-------------------------------------------------
    #-------------------------------------------------
    MaxValorSuma= np.max(histS)
    MinValorSuma= np.min(histS)
    PromedioSuma = np.average(histS)
    #-------------------------------------------------
    #-------------------------------------------------
    MaxValorRojo= np.max(histR)
    MinValorRojo= np.min(histR)
    PromedioRojo = np.average(histR)
    #-------------------------------------------------
```

```
    #-------------------------------------------------
    MaxValorVerde = np.max(histG)
    MinValorVerde = np.min(histG)
    PromedioVerde = np.average(histG)
    #-------------------------------------------------
    #-------------------------------------------------
    MaxValorAzul= np.max(histB)
    MinValorAzul= np.min(histB)
    PromedioAzul = np.average(histB)
    #-------------------------------------------------

    #Obtener los valores de cada barra ----------------------

    Yc1,Yc2,Yc3,Yc4,Yc5 =ObtenerValBarras(histC)
    Ys1,Ys2,Ys3,Ys4,Ys5 =ObtenerValBarras(histS)
    Yr1,Yr2,Yr3,Yr4,Yr5 =ObtenerValBarras(histR)
    Yg1,Yg2,Yg3,Yg4,Yg5 =ObtenerValBarras(histG)
    Yb1,Yb2,Yb3,Yb4,Yb5 =ObtenerValBarras(histB)


    #print(Y1," ", Y2, " ", Y3," ", Y4, " ", Y5," ")
    #-----------------------------------------------------


    #Numero de picos-------------------------------------
    NumeroDePicosC=find_peaks(histC)
    NumeroDePicosS=find_peaks(histS)
```

```
    NumeroDePicosR=find_peaks(histR)
    NumeroDePicosG=find_peaks(histG)
    NumeroDePicosB=find_peaks(histB)

    #-----------------------------------------------------
       data.append([f,  Rows,  Cols,  Mats,  clase,  Posicion,  Grados_rotacion,UltimaLetra,
NumSegmentos,
    MaxValorConcat, MinValorConcat, PromedioConcat,
    MaxValorSuma, MinValorSuma, PromedioSuma,
    MaxValorRojo, MinValorRojo, PromedioRojo,
    MaxValorVerde, MinValorVerde, PromedioVerde,
    MaxValorAzul, MinValorAzul, PromedioAzul,

Yc1,Yc2,Yc3,Yc4,Yc5,Ys1,Ys2,Ys3,Ys4,Ys5,Yr1,Yr2,Yr3,Yr4,Yr5,Yg1,Yg2,Yg3,Yg4,Yg5,Yb1,Yb2,Y
b3,Yb4,Yb5,NumeroDePicosC,NumeroDePicosS,NumeroDePicosR,NumeroDePicosG,NumeroDePicosB])

# ------------------
print("Casi termina...")
title  = ['Name', 'Rows', 'Cols', 'Mats', 'clase' , 'Posicion','Grados Rotacion','Ultima
Letra', 'Num Segmentos',
'MaxValor Concatenado' , 'MinValor Concatenado', 'Promedio Concatenado',
'MaxValor Suma' , 'MinValor Suma', 'Promedio Suma',
'MaxValor Rojo' , 'MinValor Rojo', 'Promedio Rojo',
'MaxValor Verde' , 'MinValor Verde', 'Promedio Verde',
'MaxValor          Azul'          ,          'MinValor          Azul',          'Promedio
Azul','YC1','YC2','YC3','YC4','YC5','YS1','YS2','YS3','YS4','YS5','YR1','YR2','YR3','YR4'
,'YR5','YG1','YG2','YG3','YG4','YG5','YB1','YB2','YB3','YB4','YB5","Numero    de    Picos
```

```
Concatenado","Numero de Picos Suma","Numero de Picos R","Numero de Picos G","Numero de
Picos B"]

dataset= pd.DataFrame(data=data, columns = title)

dataset.to_csv('DatasetV7.csv')
print("TERMINADO.")
```

- Preparación de Datos
  El código de la preparación de datos se encuentra en la siguiente enlace:
  https://colab.research.google.com/drive/1K152pfvICMQpCo8R1Tqg8Tt9GGn8HU_w?usp=sharing

- Entrenamiento del Árbol de Decisiones
  https://colab.research.google.com/drive/1IDlEmnTbAhOvYtE3KAD_80VdhOZkkblh?usp=sharing

- Entrenamiento del Random Forest
  https://colab.research.google.com/drive/1NZ2xn4tm0EZOR8DjOqMuLEnOD5T6YUbf?usp=sharing

- Entrenamiento CNN
  https://drive.google.com/file/d/1hsgDuHmFYcBIDz8B-QDK_aGMgMlnJxCM/view?usp=sharing

- Código ejecutado en el Raspberry

```
import numpy as np
import tensorflow as tf
from tensorflow.keras.applications.resnet50 import preprocess_input,
decode_predictions
from tensorflow.keras.preprocessing import image



#Carando Modelo
```

```
nuevo_modelo = tf.keras.models.load_model('./modelo_entrenado.h5')
#Se coloca los valores reales de las imágenes, esto se puede obviar ya que no es
necesario etiquetar las imágenes
# con lo que son realmente.
test_labels = [1, 3]

test_ds = []
image_size = (256, 256)
for i in range(len(test_labels)):
    #Se coloca la ruta de la foto que ha tomado la cámara del raspberry
    path1 = "./images/descarga.jpeg"
    path2 = "./images/descarga2.jpeg"
    img1 = tf.keras.preprocessing.image.load_img(path1, target_size=image_size)
    img2 = tf.keras.preprocessing.image.load_img(path2, target_size=image_size)

    #Añadir las imágenes al arreglo para que sean evaluadas
    test_ds.append(tf.keras.preprocessing.image.img_to_array(img1))
    test_ds.append(tf.keras.preprocessing.image.img_to_array(img2))
test_ds = np.array(test_ds)

test_inference = nuevo_modelo.predict_on_batch(test_ds)

class_names=['cardboard', 'glass', 'metal', 'paper', 'plastic', 'trash']
# Mostrar Resultado
print('Inference:{}, {:.2f}% Confidence\nReal Label:{}'
                          .format(class_names[np.argmax(test_inference[i])],  100  *
np.max(test_inference[i]), class_names[test_labels[i]]))
```

Conjunto de imágenes

Extracción RGB

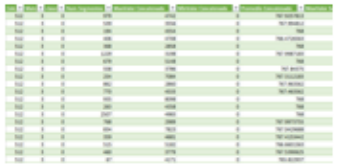
Histograma R

Histograma R

Histograma R

Histograma concatenado

Suma de histogramas



Dataset

Logaritmo Natural

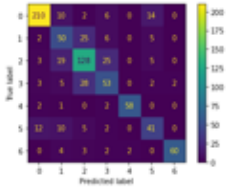
Raíz cuadrada

Normalización

Eliminación de atributos constantes

Nuevo Dataset

Entrenamiento del modelo



Matriz de confusión

Elección de modelo

Optimizar Modelo

Modelo optimizado