Universidad de Guadalajara.

Centro Universitario de Ciencias Exactas e Ingenierías.

DEPARTAMETO DE CIENCIAS COMPUTACIONALES

Ilustración 1: CUCEI

Materia: SIA2. **Carrera:** Ingeniería en computación.
Alumno: Gabriel Ángel Zárate Domínguez. **Código:** 217759094. **Sección:** D04.

Título: Proyecto: Detector de paisajes.

1.Objetivo: Crear una IA que sea capaz de detectar que tipo de paisaje es una fotografía, estableciendo previamente los tipos de paisajes.

2. ¿Por qué hacer este proyecto?:

El proyecto presentado en este documento esta proyectado para una hipotética pagina web, donde los usuarios pueden o no etiquetar sus fotos. Esta IA presenta una solución a este problema, pudiendo etiquetar las fotos que analiza. Presentando una solución a la falta de etiquetado de datos de paisajes, aprovechándonos de las redes convolucionales.

3.Desarrollo:

Para completar el objetivo planteado se hizo uso del lenguaje Python(3.8.0), aprovechando las bibliotecas: tensorflow(2.7.0), mtplotlib(v 3.5.0), numpy(v 1.21.4) y keras(v 1.1.2), también se ha de aclarar que se desarrollo usando el SO Windows 10, esto es importante de destacar debido a que el manejo de directorios que se usa puede que no sea compatible con algún otro SO.

La combinación del lenguaje Python, junto con las bibliotecas enlistadas del párrafo anterior. Permitieron al equipo desarrollador, la creación de un modelo IA tipo red convolucional.

3.1 Datasets:

Se recolectaron y clasificaron alrededor de 330 imágenes de paisajes, imágenes provenientes de la colección de datos de “Landscape Pictures” de la página Kaggle (link a la colección de datos: https://www.kaggle.com/arnaud58/landscape-pictures).

Las imágenes recolectadas se dividieron en las categorías:

- Acuático: Imágenes de playa, mar, costa, lagos o algún otro cuerpo de agua que sea el enfoque principal de la fotografía.

- Colina_Montana: Imágenes de colinas o montañas, donde estas son el foco principal de la fotografía.

- Desértico: Imágenes de desierto o paisajes claramente esteparios.

- Verde: Imágenes donde claramente predomine el verde. (Llanuras húmedas, bosque o selva.)

Cada una de estas categorías será clasificada y guardada en una carpeta del mismo nombre.

3.2 Desarrollo: Explicando los dos archivos .py

El directorio principal de la solución del software del proyecto consta de dos archivos .py, los cuales son la columna vertebral del mismo.

El primero existe bajo el nombre de “main.py”, el cual es el archivo principal del proyecto y el que se debe ejecutar si se desea hacer uso de la IA, el segundo existe bajo el nombre de “direcotory_handler.py” este segundo es quien maneja los directorios, imágenes y la conversión de estos a vectores numpy/listas a través de una clase del mismo nombre. Si alguna vez, algún lector de este documento necesitara modificar el proyecto, que sepa que, si posee algún SO tipo Linux por mencionar algún ejemplo, realmente solo necesitaría tener que cambiar esta clase.

3.2.1 Detallando el main.py y las opciones de ejecución:

Las primeras líneas de código dentro de “main.py” dentro de la función main(), nos arrojan lo siguiente:

```
int_workMode = -1
int_numeroDeRevisionesAlFinal = 10

strlist_directoriosOrigen   =   ["Acuatico", "Colina_Montana", "Desertico",
"Verde"]
strlist_directoriosFinales  =
fun_origin_end_directoriesAndClasses(strlist_directoriosOrigen)

int_xsize   = 144
int_ysize   = 144
int_canales = 3

dh_manager                  =
obj_directoryManager(strlist_directoriosOrigen)
dh_manager.workMode(int_workMode, strlist_directoriosOrigen,
strlist_directoriosFinales, int_xsize, int_ysize)
```

int_workMode : indica como se va a preparar los archivos del proyecto, estos modos son : -1, 0, 1.

En el modo -1: El proyecto ya cuenta con todos lo archivos preparados para su ejecución. Por lo cual ignorara los pasos preparativos.

En el modo 0: El proyecto será ejecutado por ***primera*** vez con esta colección de datos, ***asegúrese de tener antes las carpetas de origen de las imágenes antes de***

ejecutar el proyecto en este modo. Lo que hará la solución de software es que accederá a estas carpetas, renombrará los archivos con el nombre de la carpeta más un número y creará otra carpeta con el mismo nombre pero con el agregado “_resize”, lo mismo para las imágenes.

En el modo 1: El proyecto será ejecutado después de haberse instanciado por primera vez, en este modo solo se reajustarán las imágenes a un nuevo tamaño deseado.

int_numeroDeRevisionesAlFinal: Indican cuantas de las imágenes será desplegadas para ser observadas manualmente tras finalizar el proceso de entrenamiento y empezar el de testeo. Si no desea consultar nada manualmente, basta con poner a 0 esta variable.

strlist_directoriosOrigen: Una lista con el nombre de los directorios/carpetas de origen.

strlist_directoriosFinales: Los directorios donde se guardan las fotos generadas.

int_xsize: Es un entero el cual guarda el valor del pixale a lo largo del eje x, de las imágenes a reajustar su tamaño.

int_ysize: Es un entero el cual guarda el valor del pixale a lo largo del eje y, de las imágenes a reajustar su tamaño.

int_canales: Es un entero el cual guarda el valor de cuantos canales son usados dentro de la imagen, usualmente será 3 para imágenes a color, peor pudiera ser 1 para imágenes a escala de grises.

De estas variables las que se ven contemplan para ser modificadas son:

- **int_workMode:** Con valores enteros del -1 al 1 para indicar como tratar los archivos.
- **int_numeroDeRevisionesAlFinal:** Para observar manual mente el como trato las imágenes.
- **strlist_directoriosOrigen:** Indican las carpetas de origen del proyecto y con ello la cantidad de clases a existir.
- **int_xsize:** Tamaño del eje x de las imágenes al rescaldarlas.
- **int_ysize:** Tamaño del eje y de las imágenes al rescaldarlas.
- **int_canales:** Cantidad de canales que usa la imagen, usar 3 para RGB o 1 para escala de grises.

3.2.2 Detallando el main.py y las opciones de ejecución:

```
#Creamos el set de entrenamiento.
dh_manager.makeVectors(strlist_directoriosFinales, 40)
dh_manager.shuffle(160)
dh_manager.image_vector_state_declaration()
xtrain = np.array(dh_manager.return_vectorrgb())
ytrain = np.array(dh_manager.return_classvector())

#Creamos el set de prueba
dh_manager.reset_vectors()
dh_manager.makeVectors2(strlist_directoriosFinales)
dh_manager.shuffle(len(dh_manager.return_classvector()))
dh_manager.image_vector_state_declaration()
xtest  = np.array(dh_manager.return_vectorrgb())
ytest =  np.array(dh_manager.return_classvector())
```

Se crean los dos datasets, el de entrenamiento y el de prueba

3.2.3 Se detecta el tipo de canal que se usa, se crea el modelo y se entrena:

```
if backend.image_data_format() == 'channels_first':
    xtrain = xtrain.reshape(len(xtrain), int_canales, int_xsize, int_ysize)
    xtest  =  xtest.reshape(len(xtest), int_canales, int_xsize, int_ysize)
    input_shape = (int_canales, int_xsize, int_ysize)

else:
    xtrain = xtrain.reshape(xtrain.shape[0], int_xsize, int_ysize,
int_canales)
    xtest = xtest.reshape(xtest.shape[0], int_xsize, int_ysize, int_canales)
    input_shape = (int_xsize, int_ysize, int_canales)

ytrain = keras.utils.to_categorical(ytrain, 10)
ytest  = keras.utils.to_categorical(ytest, 10)

model = build_model(input_shape)
model.summary()
model.fit(xtrain, ytrain, batch_size=128, epochs=10, verbose=1,
validation_data=(xtest, ytest))
```

3.2.4 Éxitos en la revisión humana y en la revisión de los archivos:

```
#Contamos Exitos en la revision humana
print("------------------------------------------------------------------------
-------------------------------------------------")
porcentaje = "null"

for int_i in range(0, len(strlist_directoriosOrigen)):
    if strlist_directoriosOrigenContadorGeneral[int_i] == 0:
        porcentaje = " No hubo revision de esta clase."
    else:
        porcentaje = str(100/strlist_directoriosOrigenContadorGeneral[int_i]
* strlist_directoriosOrigenContadorExitos[int_i])

    print("De la clase: ", dh_manager.get_strclassimage(int_i),
          ", se acertaron: ",
strlist_directoriosOrigenContadorExitos[int_i], ", de esta cantidad
revisada: ", strlist_directoriosOrigenContadorGeneral[int_i],
          ", porcentaje de acierto en esta clase: ", porcentaje)

if int_numeroDeRevisionesAlFinal <= 0:
    porcentaje = "No hubo revision humana."
else:
    porcentaje = str(100/int_numeroDeRevisionesAlFinal * contador_exitos)

print("Exitos en general: ", contador_exitos, ", de :",
int_numeroDeRevisionesAlFinal, ", porcentaje: ", porcentaje)



#Contamos Exitos en la revision serializada
print("------------------------------------------------------------------------
-------------------------------------------------")

strlist_directoriosOrigenContadorExitos  = []
strlist_directoriosOrigenContadorGeneral = []
contador_exitos = 0

for i in strlist_directoriosOrigen:
    strlist_directoriosOrigenContadorExitos.append(0)
    strlist_directoriosOrigenContadorGeneral.append(0)

to_not_hiperpaginate = len(dh_manager.return_classvector())

for i in range(0, to_not_hiperpaginate):

    print("Revision :", i, " de : ", to_not_hiperpaginate)
    patron = xtest[i].reshape(1, int_xsize, int_ysize, int_canales)

    int_prediccion = int(np.argmax(model.predict(patron)))
    int_realidad   = dh_manager.return_classvector()[i]
    strlist_directoriosOrigenContadorGeneral[int_realidad] =
strlist_directoriosOrigenContadorGeneral[int_realidad] + 1

    if int_prediccion == int_realidad:
        contador_exitos = contador_exitos + 1
```

```
        strlist_directoriosOrigenContadorExitos[int_realidad] =
strlist_directoriosOrigenContadorExitos[int_realidad] + 1

for int_i in range(0, len(strlist_directoriosOrigen)):
    if strlist_directoriosOrigenContadorGeneral[int_i] == 0:
        porcentaje = " No hubo revision de esta clase."
    else:
        porcentaje = str(100/strlist_directoriosOrigenContadorGeneral[int_i]
* strlist_directoriosOrigenContadorExitos[int_i])

    print("De la clase: ", dh_manager.get_strclassimage(int_i),
          ", se acertaron: ",
strlist_directoriosOrigenContadorExitos[int_i], ", de esta cantidad
revisada: ", strlist_directoriosOrigenContadorGeneral[int_i],
          ", porcentaje de acierto en esta clase: ", porcentaje)

print("Exitos en general: ", contador_exitos, ", de :",
len(dh_manager.return_classvector()), ", porcentaje: ",
str(100/len(dh_manager.return_classvector()) * contador_exitos))
```

4.0.0 Aprendizaje y generalización del modelo:

1. Intento:

Entrenamiento: 160 Imágenes.

Testeo: 337 Imágenes.

Pixelaje: 144x144.

Épocas: 10.

Porcentaje de acierto en pruebas: 86%.

```
Epoch 10/10
2/2 [==============================] - 6s 3s/step - loss: 0.9729 - accuracy: 0.8687 - val_loss: 1.7612 - val_accuracy: 0.8257
```

Porcentaje de acierto en el testeo: 82 %.

```
De la clase:  Acuatico , se acertaron:  61 , de esta cantidad revisada:  82 , porcentaje de acierto en esta clase:  74.39024390243902
De la clase:  Colina_Montana , se acertaron:  49 , de esta cantidad revisada:  55 , porcentaje de acierto en esta clase:  89.0909090909091
De la clase:  Desertico , se acertaron:  36 , de esta cantidad revisada:  43 , porcentaje de acierto en esta clase:  83.72093023255815
De la clase:  Verde , se acertaron:  124 , de esta cantidad revisada:  147 , porcentaje de acierto en esta clase:  84.35374149659863
Exitos en general:  270 , de : 327 , porcentaje:  82.56880733944953
```

2. Intento:

Entrenamiento: 160 Imágenes.

Testeo: 337 Imágenes.

Pixelaje: 144x144.

Épocas: 15.

Porcentaje de acierto en pruebas: 88%.

```
Epoch 15/15
2/2 [==============================] - 6s 3s/step - loss: 0.3090 - accuracy: 0.8875 - val_loss: 2.1986 - val_accuracy: 0.7554
```

Porcentaje de acierto en el testeo: 75 %.

```
De la clase:  Acuatico , se acertaron:  51 , de esta cantidad revisada:  82 , porcentaje de acierto en esta clase:  62.19512195121951
De la clase:  Colina_Montana , se acertaron:  48 , de esta cantidad revisada:  55 , porcentaje de acierto en esta clase:  87.27272727272727
De la clase:  Desertico , se acertaron:  43 , de esta cantidad revisada:  43 , porcentaje de acierto en esta clase:  100.0
De la clase:  Verde , se acertaron:  105 , de esta cantidad revisada:  147 , porcentaje de acierto en esta clase:  71.42857142857143
Exitos en general:  247 , de : 327 , porcentaje:  75.53516819571865
```

3. Intento:

Entrenamiento: 160 Imágenes.

Testeo: 337 Imágenes.

Pixelaje: 144x144.

Épocas: 15.

Porcentaje de acierto en pruebas: 89%.

```
Epoch 13/13
2/2 [==============================] - 6s 3s/step - loss: 0.4663 - accuracy: 0.8938 - val_loss: 1.6304 - val_accuracy: 0.7737
```

Porcentaje de acierto en el testeo: 75 %.

```
De la clase:  Acuatico , se acertaron:  55 , de esta cantidad revisada:  82 , porcentaje de acierto en esta clase:  67.07317073170732
De la clase:  Colina_Montana , se acertaron:  54 , de esta cantidad revisada:  55 , porcentaje de acierto en esta clase:  98.18181818181817
De la clase:  Desertico , se acertaron:  40 , de esta cantidad revisada:  43 , porcentaje de acierto en esta clase:  93.0232558139535
De la clase:  Verde , se acertaron:  104 , de esta cantidad revisada:  147 , porcentaje de acierto en esta clase:  70.74829931972789
Exitos en general:  253 , de : 327 , porcentaje:  77.37003058103976
```

4. Intento:

Entrenamiento: 160 Imágenes.

Testeo: 272 Imágenes.

Pixelaje: 144x144.

Épocas: 12.

Porcentaje de acierto en pruebas: 79%.

```
1/1 [==============================] - 5s 5s/step - loss: 10.1782 - accuracy: 0.7917 - val_loss: 6.2702 - val_accuracy: 0.8824
```

Porcentaje de acierto en el testeo: 89 %.

```
De la clase:  Acuatico , se acertaron:  72 , de esta cantidad revisada:  82 , porcentaje de acierto en esta clase:  87.80487804878048
De la clase:  Desertico , se acertaron:  31 , de esta cantidad revisada:  43 , porcentaje de acierto en esta clase:  72.09302325581396
De la clase:  Verde , se acertaron:  137 , de esta cantidad revisada:  147 , porcentaje de acierto en esta clase:  93.19727891156462
Exitos en general:  240 , de : 272 , porcentaje:  88.23529411764707
```

Nota: Se simplifico el modelo al eliminar la clase montaña.

4.1.0 Inferencias y pensamientos de mejora a futuro:

El modelo aquí presentado muestra una tendencia a mejorar alrededor de las 10 a 12 épocas, también muestra una tendencia a mejorar conforme más imágenes se analizan, sin embargo, se juzga que convendrían más imágenes de análisis aun sí puesto que a lo largo de varias ejecuciones ha llegado a tener rangos desde 85 a 92s de maneras muy dispersas.

Esto deduzco que se debe a la naturaleza del azar a la hora de elegir los datos de prueba y que algunos de estos resultan poco ejemplificadores del paisaje medio.

También tras varias pruebas al reducir la cantidad de clases a clasificar, tiende a mostrar una tendencia a la mejoría. Principalmente al eliminar la clase “montaña” o la clase “desierto”. Ya que la primera tiende a confundirse con la clase “verde”y la segunda con la clase “acuática” cuando se trata de tomas de playa.

Una aproximación a futuro que pudiera darle a este problema, seria utilizar algún modelo que pudiera clasificar una sola entrada en varias clasificaciones, puesto que hay paisajes que NO son mutuamente excluyente uno del otro. Como una montaña boreal que entraría en “verde” y “montaña”. O una selva con rio la cual entraría en “verde” y “acuático.”

5.0.0 Códigos y repositorios:

Repositorio: https://github.com/GabrielZarate/IA_DetectorDePaisajes

Main.py

```
from directory_handler import *

import numpy as np
import matplotlib.pyplot as plt
from tensorflow import keras
from keras.models import Sequential
from keras.layers import Dense, Dropout, Flatten
from keras.layers import Conv2D, MaxPooling2D
from keras import backend

def fun_origin_end_directoriesAndClasses(str_list):
    str_list_end = []
    for string in str_list:
        str_list_end.append(string + "_resize")
    return str_list_end

def build_model(input_shape):
    model = Sequential()

    #Modificamos el kernel para mas precision ya que en este ejemplos los colores son importantes.
    """
    model.add(Conv2D(32, kernel_size=(3, 3), activation='relu', input_shape=input_shape))
    model.add(Conv2D(64, kernel_size=(3, 3), activation='relu'))
    """
    model.add(Conv2D(32, kernel_size=(3, 3), activation='relu', input_shape=input_shape))
    model.add(Conv2D(64, kernel_size=(3, 3), activation='relu'))
    model.add(MaxPooling2D(pool_size=(2, 2)))
    model.add(Dropout(0.25))
    model.add(Flatten())
    model.add(Dense(128, activation='relu'))
    model.add(Dropout(0.5))
    model.add(Dense(10, activation='softmax'))

    model.compile(loss=keras.losses.categorical_crossentropy, optimizer='adam', metrics=['accuracy'])

    return model

def main():

    int_workMode = -1
    int_numeroDeRevisionesAlFinal = 10

    strlist_directoriosOrigen   =   ["Acuatico", "Desertico", "Verde"]
```

```
    #strlist_directoriosOrigen   =   ["Acuatico", "Colina_Montana",
"Desertico", "Verde"]
    strlist_directoriosFinales  =
fun_origin_end_directoriesAndClasses(strlist_directoriosOrigen)

    int_xsize   = 144
    int_ysize   = 144
    int_canales = 3

    dh_manager                  =
obj_directoryManager(strlist_directoriosOrigen)
    dh_manager.workMode(int_workMode, strlist_directoriosOrigen,
strlist_directoriosFinales, int_xsize, int_ysize)

    #Creamos el set de entrenamiento.
    dh_manager.makeVectors(strlist_directoriosFinales, 40)
    dh_manager.shuffle(160)
    dh_manager.image_vector_state_declaration()
    xtrain = np.array(dh_manager.return_vectorrgb())
    ytrain = np.array(dh_manager.return_classvector())

    #Creamos el set de prueba
    dh_manager.reset_vectors()
    dh_manager.makeVectors2(strlist_directoriosFinales)
    dh_manager.shuffle(len(dh_manager.return_classvector()))
    dh_manager.image_vector_state_declaration()
    xtest  = np.array(dh_manager.return_vectorrgb())
    ytest =  np.array(dh_manager.return_classvector())

    #Aqui inicia la ia ----------------------------------------------------
----------------------------------------------------------


    if backend.image_data_format() == 'channels_first':
        xtrain = xtrain.reshape(len(xtrain), int_canales, int_xsize,
int_ysize)
        xtest  =  xtest.reshape(len(xtest), int_canales, int_xsize,
int_ysize)
        input_shape = (int_canales, int_xsize, int_ysize)

    else:
        xtrain = xtrain.reshape(xtrain.shape[0], int_xsize, int_ysize,
int_canales)
        xtest = xtest.reshape(xtest.shape[0], int_xsize, int_ysize,
int_canales)
        input_shape = (int_xsize, int_ysize, int_canales)

    ytrain = keras.utils.to_categorical(ytrain, 10)
    ytest  = keras.utils.to_categorical(ytest, 10)

    model = build_model(input_shape)
    model.summary()
    model.fit(xtrain, ytrain, batch_size=120, epochs=12, verbose=1,
validation_data=(xtest, ytest))
```

```
    #Aquí inicia la revision humana:

    strlist_directoriosOrigenContadorExitos  = []
    strlist_directoriosOrigenContadorGeneral = []
    contador_exitos = 0

    for i in strlist_directoriosOrigen:
        strlist_directoriosOrigenContadorExitos.append(0)
        strlist_directoriosOrigenContadorGeneral.append(0)

    for i in range(0, int_numeroDeRevisionesAlFinal):

        random_number = np.random.randint(0,
len(dh_manager.return_classvector()))
        patron = xtest[random_number].reshape(1, int_xsize, int_ysize,
int_canales)

        int_prediccion = int(np.argmax(model.predict(patron)))
        int_realidad   = dh_manager.return_classvector()[random_number]
        strlist_directoriosOrigenContadorGeneral[int_realidad] =
strlist_directoriosOrigenContadorGeneral[int_realidad] + 1

        if int_prediccion == int_realidad:
            contador_exitos = contador_exitos + 1
            strlist_directoriosOrigenContadorExitos[int_realidad] =
strlist_directoriosOrigenContadorExitos[int_realidad] + 1

        plt.imshow(xtest[random_number])
        plt.title('Prediction: ' +
dh_manager.get_strclassimage(int_prediccion))
        plt.show()

        plt.imshow(dh_manager.return_vectorrgb()[random_number])
        plt.title('Realidad: ' + dh_manager.get_strclassimage(int_realidad))
        plt.show()


    #Contamos Exitos en la revision humana
    print("-------------------------------------------------------------------
-----------------------------------------------------")
    porcentaje = "null"

    for int_i in range(0, len(strlist_directoriosOrigen)):
        if strlist_directoriosOrigenContadorGeneral[int_i] == 0:
            porcentaje = " No hubo revision de esta clase."
        else:
            porcentaje =
str(100/strlist_directoriosOrigenContadorGeneral[int_i] *
strlist_directoriosOrigenContadorExitos[int_i])

        print("De la clase: ", dh_manager.get_strclassimage(int_i),
              ", se acertaron: ",
strlist_directoriosOrigenContadorExitos[int_i], ", de esta cantidad
revisada: ", strlist_directoriosOrigenContadorGeneral[int_i],
              ", porcentaje de acierto en esta clase: ", porcentaje)
```

```
    if int_numeroDeRevisionesAlFinal <= 0:
        porcentaje = "No hubo revision humana."
    else:
        porcentaje = str(100/int_numeroDeRevisionesAlFinal *
contador_exitos)

    print("Exitos en general: ", contador_exitos, ", de :",
int_numeroDeRevisionesAlFinal, ", porcentaje: ", porcentaje)



    #Contamos Exitos en la revision serializada
    print("------------------------------------------------------------------
-------------------------------------------------")

    strlist_directoriosOrigenContadorExitos  = []
    strlist_directoriosOrigenContadorGeneral = []
    contador_exitos = 0

    for i in strlist_directoriosOrigen:
        strlist_directoriosOrigenContadorExitos.append(0)
        strlist_directoriosOrigenContadorGeneral.append(0)

    to_not_hiperpaginate = len(dh_manager.return_classvector())

    for i in range(0, to_not_hiperpaginate):

        #print("Revision :", i, " de : ", to_not_hiperpaginate)
        patron = xtest[i].reshape(1, int_xsize, int_ysize, int_canales)

        int_prediccion = int(np.argmax(model.predict(patron)))
        int_realidad   = dh_manager.return_classvector()[i]
        strlist_directoriosOrigenContadorGeneral[int_realidad] =
strlist_directoriosOrigenContadorGeneral[int_realidad] + 1

        if int_prediccion == int_realidad:
            contador_exitos = contador_exitos + 1
            strlist_directoriosOrigenContadorExitos[int_realidad] =
strlist_directoriosOrigenContadorExitos[int_realidad] + 1

    for int_i in range(0, len(strlist_directoriosOrigen)):
        if strlist_directoriosOrigenContadorGeneral[int_i] == 0:
            porcentaje = " No hubo revision de esta clase."
        else:
            porcentaje =
str(100/strlist_directoriosOrigenContadorGeneral[int_i] *
strlist_directoriosOrigenContadorExitos[int_i])

        print("De la clase: ", dh_manager.get_strclassimage(int_i),
              ", se acertaron: ",
strlist_directoriosOrigenContadorExitos[int_i], ", de esta cantidad
revisada: ", strlist_directoriosOrigenContadorGeneral[int_i],
              ", porcentaje de acierto en esta clase: ", porcentaje)

    print("Exitos en general: ", contador_exitos, ", de :",
len(dh_manager.return_classvector()), ", porcentaje: ",
```

```
str(100/len(dh_manager.return_classvector()) * contador_exitos))


if __name__ == '__main__':
    main()
```

directory_handler.py

```
from os import listdir, rename, remove, mkdir
from os.path import isfile, join, exists
from PIL import Image
import matplotlib.image as mpimg
import numpy as np
import random
import matplotlib.pyplot as plt

class obj_directoryManager:

    def __init__(self, strlist_classesnames):
        self._strList_filesNames    =  []
        self._imagevector           =  []
        self._classvector           =  []
        self._classvectorstr        =  strlist_classesnames


    def __setget_list_fileNames(self, str_directoryPath):
        self._strList_filesNames = []
        self._strList_filesNames = [f for f in listdir(str_directoryPath) if
isfile(join(str_directoryPath, f))]

    def extset_changeFileNamesOfDirectory(self, str_directoryPath,
str_baseName):

        self.__setget_list_fileNames(str_directoryPath)

        for int_i in range(0, len(self._strList_filesNames)):
            str_fileDirectoryPlusName = (join(str_directoryPath,
self._strList_filesNames[int_i]))
            #str_fileName = str_fileDirectoryPlusName.split(sep=".")[0]
            str_fileExtn = str_fileDirectoryPlusName.split(sep=".")[1]
            print(str_fileDirectoryPlusName)
            print(join(str_directoryPath, str_baseName + str(int_i) + "." +
str_fileExtn))

            if isfile(join(str_directoryPath, str_baseName + str(int_i) +
"." + str_fileExtn)):
                print("Archivos ya ordenados.")
                break

            rename(str_fileDirectoryPlusName, join(str_directoryPath,
str_baseName + str(int_i) + "." + str_fileExtn))

    def __extset_deleteFilesFromDirectory(self, str_directoryPath):
        for f in listdir(str_directoryPath):
            remove(join(str_directoryPath, f))

    def extset_imageResizerSetToDirectory(self, str_sourceDirectoryPath,
```

```
str_endDirectoryPath, lon_size1, lon_size2):

        if exists(str_endDirectoryPath):
            self.__extset_deleteFilesFromDirectory(str_endDirectoryPath)
        else:
            mkdir(str_endDirectoryPath)

        self.__setget_list_fileNames(str_sourceDirectoryPath)

        for int_i in range(0, len(self._strList_filesNames)):

            str_fileDirectoryPlusName = (join(str_sourceDirectoryPath,
self._strList_filesNames[int_i]))
            str_originalFileName = self._strList_filesNames[int_i]
            str_fileNameWithOutExtension =
str_originalFileName.split(sep=".")[0]
            str_fileExtensionWithOutPointAndName =
str_originalFileName.split(sep=".")[1]

            image = Image.open(str_fileDirectoryPlusName)
            image = image.resize((lon_size1, lon_size2))
            str_final_name = join(str_endDirectoryPath,
str_fileNameWithOutExtension + "_resize" + "." +
str_fileExtensionWithOutPointAndName)
            print(str_final_name)
            image.save(str_final_name)

    def __extget_rgbvector255OfImage(self, str_fileName):
        img = mpimg.imread(str_fileName)
        vector_rgb = np.array(img, dtype = np.int)
        return vector_rgb


    def extset_rgbvectorOfImageCollectionAndClass(self, str_directoryPath,
int_class, int_limit):
        self.__setget_list_fileNames(str_directoryPath)

        for int_i in range(0, len(self._strList_filesNames)):
            if int_i >= int_limit:
                break

self._imagevector.append(self.__extget_rgbvector255OfImage(join(str_director
yPath, self._strList_filesNames[int_i])))
            self._classvector.append(int_class)

        print("Hay tal cantidad de datos en el vector de color: ",
len(self._imagevector))
        print("Hay tal cantidad de datos en el vector de clases: ",
len(self._classvector))

    def extset_rgbvectorOfImageCollectionAndClass2(self, str_directoryPath,
int_class):
        self.__setget_list_fileNames(str_directoryPath)

        for int_i in range(0, len(self._strList_filesNames)):
```

```
self._imagevector.append(self.__extget_rgbvector255OfImage(join(str_directorYPath, self._strList_filesNames[int_i])))
            self._classvector.append(int_class)

        print("Hay tal cantidad de datos en el vector de color: ", len(self._imagevector))
        print("Hay tal cantidad de datos en el vector de clases: ", len(self._classvector))

    def image_vector_state_declaration(self):
        print("Existe este vector: ", self._classvector)
        #print("Existe estos vectores de imagen float rgb: ", self._imagevector)

    def shuffle(self, int_howManyTimes):
        for i in range(0, int_howManyTimes):

            n1 = random.randint(0, len(self._classvector)-1)
            n2 = random.randint(0, len(self._classvector)-1)

            self._imagevector[n1], self._imagevector[n2] = self._imagevector[n2], self._imagevector[n1]
            self._classvector[n1], self._classvector[n2] = self._classvector[n2], self._classvector[n1]

    def fun_ocurrences_of_each_class(self):
        for i in range(0, len(self._classvectorstr)):
            print("De la clase : ", self.get_strclassimage(i), ", hay: ", self._classvector.count(i))

    def return_vectorrgb(self):
        return self._imagevector

    def return_classvector(self):
        return self._classvector

    def print_rgbvector_toImage(self, vector_rgb, str_tittle):
        plt.title(str_tittle)
        plt.imshow(vector_rgb)
        plt.show()

    def get_strclassimage(self, int_class):
        return self._classvectorstr[int_class]

    def workMode(self, int_fileMagmentMode, strlist_directoriosOrigen, strlist_directoriosFinales, int_xsize, int_ysize):

        if int_fileMagmentMode == -1:
            return

        if int_fileMagmentMode == 0:
            for int_i in range(0, len(strlist_directoriosOrigen)):

self.extset_changeFileNamesOfDirectory(strlist_directoriosOrigen[int_i], strlist_directoriosFinales[int_i])
```

```
self.extset_imageResizerSetToDirectory(strlist_directoriosOrigen[int_i],
strlist_directoriosFinales[int_i], int_xsize, int_ysize)

        if int_fileMagmentMode == 1:
            for int_i in range(0, len(strlist_directoriosOrigen)):

self.extset_imageResizerSetToDirectory(strlist_directoriosOrigen[int_i],
strlist_directoriosFinales[int_i], int_xsize, int_ysize)

    def makeVectors(self, strlist_directorios,
int_numeroDeMuestrasPorDirectorio):
        for int_i in range(0, len(strlist_directorios)):

self.extset_rgbvectorOfImageCollectionAndClass(strlist_directorios[int_i],
int_i, int_numeroDeMuestrasPorDirectorio)

    def makeVectors2(self, strlist_directorios):
        for int_i in range(0, len(strlist_directorios)):

self.extset_rgbvectorOfImageCollectionAndClass2(strlist_directorios[int_i],
int_i)

    def reset_vectors(self):
        self._imagevector = []
        self._classvector = []
```