Type of the Paper (Article)

Modelo de clasificación de imágenes de las plagas que atacan los cultivos de papa

Leidy Milena Garzón Garzón[1,*], Andrés Fabián Aya Cifuentes[2] and Jonathan Dixon Pedraza Josa[3]

[1] Universidad Cooperativa de Colombia; leidy.garzong@campusucc.edu.co
[2] Universidad Cooperativa de Colombia; andres.aya@campusucc.edu.co
[3] Universidad Cooperativa de Colombia; jonathan.pedraza@campusucc.edu.co
* Correspondence: leidy.garzong@campusucc.edu.co

Received: 15/03/2021; Accepted: 15/06/2021; Published: 30/06/2021

1. Introducción

En Colombia la papa es uno de los alimentos que se siembran principalmente en 4 departamentos los cuales son: Cundinamarca, Boyacá, Nariño y Antioquia; con al menos 166.000 hectáreas, este alimento se comercializa en Colombia en un 90% y se consume principalmente freso. El 10% restante se exporta. Lo que garantiza una permanente demanda del producto, además de que esta actividad llega a generar en Colombia 75.000 empleos directos y 189.000 empleos indirectos para un total de alrededor de 264.000 empleos (Agricultura de las Américas, 2020).

Sin embargo, existen diversas enfermedades y plagas que atacan estos cultivos generando millones de pérdidas a los campesinos colombianos, algunas de estas plagas son:

- El gusano blanco
- Las polillas guatemaltecas
- La larva
- La pulguilla
- El trips
- La mosca minadora
- El pulgón

Cada una de las plagas anteriormente nombradas tienen una forma diferente de tratar y realizan diferentes daños a los cultivos.

Por ejemplo, en el caso del gusano blanco (Premnotrypes vorax) se reconoce porque su cuerpo es de color blanco cremoso, con la cabeza de color café. Llega a medir 14 mm de largo y su cuerpo tiene forma de "C". El daño que esté puede llegar a ocasionar es comerse los filos de las hojas en forma de medialuna y la base del tallo, además de que se alimentan de las papas y hacen huecos en ellas. (Fabián Montesdeoca (INIAP), 2013).

Por otro lado, están las polillas guatemaltecas las cuales son mariposas de color café que miden cerca de 10 mm de largo, algunas tienen manchas triangulares en las alas, otras una línea negra a lo largo de cada ala o tienen manchas pequeñas en las alas. (ICA, 2016).

Si se habla de la larva esta mide entre 12 y 15 mm y son de color rojizo, estas hacen minas en las hojas, huecos a los tallos además de huecos a las papas. El clima en el que más atacan es en climas cálidos y secos con temperaturas mayores a 20°C, suelen aparecer desde la siembra hasta la cosecha e incluso el almacenamiento.

También se encuentra la pulguilla (Epotrix spp) la cual es un pequeño escarabajo que mide entre 1 a 2 mm de largo y es de color negro con brillo metálico, pueden causar daños como perforaciones circulares en los brotes de la planta, los cuales aumentan de tamaño conforme crece la hoja, además de causar que las plantas emerjan de forma desigual.

En el caso de los trips (Frankliniella tuberosi) sus cuerpos son pequeños y alargados (1mm aproximadamente) y de un color amarillo o verde. Suelen provocar daño en la epidermis del envés de la hoja inferior raspando y chupando el líquido celular provocando así manchas de color plateado.

Para el caso de la mosca minadora (Liriomyza spp) está puede llegar a medir 3 mm de largo con una coloración amarilla en la mitad de la cabeza y en el tórax. Esta plaga hace túneles en el interior de la hoja, sin dañar la parte externa de la misma, haciendo que las hojas se sequen y pueda matar la planta.

Por último, se tiene el pulgón (Myzus persicae, Macrosiphum euphorbiae) este tiene un cuerpo en forma de pera, es de color verde claro a oscuro y mide entre 1,5 a 2,5 mm. Esta plaga se come las hojas de la planta o de los brotes del tubérculo, además de que pueden transmitir virus al alimentarse. (Fabián Montesdeoca (INIAP), 2013).

Como se puede comprender, existe una gran cantidad de plagas que atacan los cultivos de papa, sin embargo, cada día aparecen más y sus formas de hacer daño pueden ser similares unas a otras, por lo que en la actualidad muchas multinacionales de protección de cultivos están estudiando este tipo de plagas para así poder con mayor facilidad detectarlas y dar un tratamiento oportuno y adecuado para evitar el daño total de los cultivos.

Un ejemplo de lo anteriormente dicho está el departamento de Nariño, uno de los departamentos más grandes para la cosecha de papa en Colombia y con una gran cantidad de cultivadores entre ellos algunos que no tienen mayores conocimientos y más porque los impactos de las temperaturas pueden llegar a originar tipos de plagas que anterior mente no habían sido identificadas. Por esto mismo el instituto colombiano de agropecuario (ICA) da la advertencia a los productores para poder identificar las plagas e intervenir a tiempo ya que pueden provocar el retraso y crecimiento de los cultivos, y un alto nivel de reproducción de las plagas.

Además, es muy importante tener una producción de cultivos limpios para poder enviar al consumidor, cabe resaltar que al momento de tener estos pequeños animalitos en los cultivos implica aumentar el uso de las sustancias toxicas que afectan la salud del ser humano y el medio ambiente como lo comenta uno de los expertos el señor Salvador Arias (Director Técnico Programa de Alianzas Comerciales de USAID) se debe tener un mejor control de estas de manera unificada debe ser seleccionada dependiendo del ambiente ya que muchas de las plagas existentes pueden adaptarse y no tener un control sobre ellas, tener una anticipación a cada afectación para evitar sobre costos en procesos curativos. (Manejo y control de plagas y enfermedades en los cultivos de papa - Dayana Forero Mora RCN RADIO).

Por esta razón se debe entender la dinámica de las plagas para lo que se debe conocer sus diferentes formas y estudios. Además, se debe tener en cuenta cómo afectan y en qué medida, el éxito de su control está en reconocerlas, saber cuándo y cómo controlarlas.

Es por ello que el objetivo principal es poder determinar las plagas que se encuentran perjudicando los cultivos de los agricultores mediante un sistema de clasificación de imágenes de las plagas que atacan los cultivos de papa, ya que muchos de ellos tienen desconocimiento de qué tipo de plaga puede ser, para así poder dar un tratamiento mucho más rápido y evitar la pérdida total o parcial de sus cultivos, favoreciendo de esta forma la economía del país al generar un mayor empleo y teniendo más venta de su producto, lo cual genera una mayor ganancia.

Por lo que como se nombró anteriormente se propone la creación de un sistema de clasificación de imágenes para la identificación de las principales plagas que perjudican los cultivos de papa en Colombia. En primera instancia se deben obtener una gran cantidad de imágenes de cada una de las

plagas y organizarlas dentro de un algoritmo programado en Python con el fin poder determinar la plaga que está afectando al cultivo de los paperos sin conocimiento en Colombia.

Para la elaboración del programa de clasificación que clasifique estas plagas en los cultivos de papa, vamos a requerir dos aspectos fundamentales como lo son un modelo de predicción y un algoritmo de procesamiento. El primero definirá la estructura y organización con en que serán interpretados los datos extraídos de las imágenes mientras que el segundo hará los cálculos matemáticos y encontrará los resultados para las coincidencias de la imagen procesada.

Algunos de los principales modelos de clasificación usados en imágenes son:

- ResNet
- Wide ResNet
- ResNeXt
- DenseNet
- NASNet
- SENet
- CapsNet

Por otra parte, tenemos que algunos de algoritmos de clasificación que se usan principalmente en imágenes son:

- k-Nearest Neighbors
- Linear Regression
- Logistic Regression
- Support Vector Machines
- Bayesian Classifiers
- Decision Tress and Random Forest
- Neural Networks
- Deep Learning

Para construir el programa se usará el lenguaje de programación Python, este lenguaje es la elección más apropiada para aquellos que empezaban en la ciencia de datos. Por otro lado, Python ofrece muchos beneficios, lo que significa que poco a poco se está convirtiendo en el lenguaje más utilizado para el aprendizaje profundo. Algunas de las razones por las que se puede elegir este lenguaje son: la facilidad de uso y construcción de herramientas de análisis, su versatilidad, el crecimiento de la comunidad de usuarios y ser de gran utilidad para el Deep Learning gracias a existir numerosos paquetes destinados para ello como TensorFlow, Theano o Keras que hacen que sea realmente sencillo crear redes neuronales profundas.

2. Materiales y Métodos

2.1 Obtención de los datos

Por medio de una investigación realizada se obtienen imágenes desde internet de las plagas más comunes que se presentan en los cultivos de papa, la fuente principal de las imágenes principal son los motores de búsqueda de imágenes Google, Bing y Yandex.

También se extraen imágenes de videos de Youtube con la herramienta Ffmpeg.

2.2 Organización de las imágenes

Una vez descargadas localmente las imágenes se deben redimensionar todas las imágenes de entrada al programa para que tengan el mismo tamaño.

Las imágenes para cada plaga se deben organizar por carpetas donde el nombre de la carpeta será el posterior nombre que se asignara a la etiqueta y dentro de esta carpeta se deben alojar directamente las imágenes correspondientes al tipo de plaga.

2.3 Importación de librerías

Se debe tener claro las librerías a usar ya que existen varias opciones y la librería nos define como debemos escribir el código para el ejercicio.

2.4 Cargar las imágenes en la memoria del ordenador

En este punto se usara la línea *plt.imread(filepath)* la cual cargará a la memoria un arreglo las imágenes a usar, esto puede tomar varios minutos y consumirá algo de memoria RAM del ordenador.

2.5 Creación de etiquetas y clases

El programa debe generar con base a los nombres de las carpetas las etiquetas y las clases que serán usadas posteriormente para el modelo de datos.

2.6 Configuración de entrenamiento

Aquí indicamos la forma en que se va a guardar en memoria la lectura de las imágenes pixel a pixel, esto se guarda en arreglos numpy de 8bits, de la misma forma preparamos las etiquetas y nombres de clases para que funcionen correctamente con la red neuronal que se usara más adelante para el entrenamiento y la predicción de futuras imágenes.

2.7 Creación de red y configuración de algoritmo

Se establecen los diferentes parámetros con los que funcionan los algoritmos en conjunto con la red neuronal.

2.8 Uso del Algoritmo (CNN)

Utilizaremos la librería previa importada Keras para crear la Convolutional Neural Network o red neuronal convolucional.

Una red neuronal convolucional es un tipo de red neuronal artificial donde las neuronas corresponden a campos receptivos de una manera muy similar a las neuronas en la corteza visual primaria (versión 1) de un cerebro biológico. Este tipo de red es una variación de un perceptron multicapa, sin embargo, debido a que su aplicación es realizada en matrices bidimensionales, son muy efectivas para tareas de visión artificial, como en la clasificación y segmentación de imágenes, entre otras aplicaciones.

Neuronas convolucionales: En la fase de extracción de características, las neuronas sencillas de un perceptron son reemplazadas por procesadores en matriz que realizan una operación sobre los datos de imagen 2D que pasan por ellas, en lugar de un único valor numérico. La salida de cada neurona convolucional se calcula como:

$$Y_j = g\left(b_j + \sum_i K_{ij} \otimes Y_i\right)$$

Donde la salida Yj de una neurona j es una matriz que se calcula por medio de la combinación lineal de las salidas Yi de las neuronas en la capa anterior cada una de ellas operadas con el núcleo de convolucional Kij correspondiente a esa conexión. Esta cantidad es sumada a una influencia bj y luego se pasa por una función de activación g. no-lineal.

El operador de convolución tiene el efecto de filtrar la imagen de entrada con un núcleo previamente entrenado. Esto transforma los datos de tal manera que ciertas características (determinadas por la forma del núcleo) se vuelven más dominantes en la imagen de salida al tener estas un valor numérico más alto asignados a los pixeles que las representan. Estos núcleos tienen habilidades de procesamiento de imágenes específicas, como por ejemplo la detección de bordes que se puede realizar con núcleos que resaltan la gradiente en una dirección en particular. Sin embargo, los núcleos que son entrenados por una red neuronal convolucional generalmente son más complejos para poder extraer otras características más abstractas y no triviales.

Neuronas de Reducción de Muestreo: Las redes neuronales cuentan con cierta tolerancia a pequeñas perturbaciones en los datos de entrada. Por ejemplo, si dos imágenes casi idénticas (diferenciadas únicamente por un traslado de algunos pixeles lateralmente) se analizan con una red neuronal, el resultado debería de ser esencialmente el mismo. Esto se obtiene, en parte, dado a la reducción de muestreo que ocurre dentro de una red neuronal convolucional. Al reducir la resolución, las mismas características corresponderán a un mayor campo de activación en la imagen de entrada.

Neuronas de Clasificación: Después de una o más fases de extracción de características, los datos finalmente llegan a la fase de clasificación. Para entonces, los datos han sido depurados hasta una serie de características únicas para la imagen de entrada, y es ahora la labor de esta última fase el poder clasificar estas características hacia una etiqueta u otra, según los objetivos de entrenamiento.

Las neuronas en esta fase funcionan de manera idéntica a las de un perceptron multicapas, donde la salida de cada una se calcula de esta forma:

$$y_j = g\left(b_j + \sum_i w_{ij} \cdot y_i\right)$$

2.8 Entrenamiento

Una vez aplicada toda la configuración anterior ya está listo nuestro programa para entrenar con base a la casuística que hemos venido realizando a través de las configuraciones. Cuando se realiza el entrenamiento se generan un archivo de post procesamiento el cual nos sirve para que la siguiente vez que se ejecute nuestro programa no se tenga que realizar el entrenamiento desde cero, sino que se pueda usar estos archivos a manera de cache y acelerar el proceso.

2.9 Predicción y clasificación de imágenes

Nuestra red neuronal ya está lista para procesar imágenes a petición e identificar con base a las categorías e imanes de entrada suministradas. Para la detección se compara pixel por pixel en conjuntos de árboles de datos para encontrar similitudes con base al algoritmo configurado.

3. Resultados

```
from skimage.transform import resize
import numpy as np
import os
import re
import warnings
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report

import keras
from keras.utils.np_utils import to_categorical
from keras.models import Sequential
from keras.layers import Dense, Dropout, Flatten
from keras.layers import Conv2D, MaxPooling2D
from keras.layers.advanced_activations import LeakyReLU
```

Se realizaron todas las importaciones necesarias para el funcionamiento del programa, estas importaciones están compuestas principalmente por numpy (para la gestión de datos), matplotlib (generador de graficas), sklearn (algoritmos de aprendizaje) y keras (para el procesamiento de imágenes).

```
dirname = os.path.join(os.getcwd(), 'plagas')
imgpath = dirname + os.sep
```

Se extrae la ruta absoluta de la carpeta donde se encuentra las imágenes para el aprendizaje, por defecto estarán localizadas en una carpeta “plagas” en la misma carpeta donde se encuentre el script Python del programa.

```
images = []
directories = []
dircount = []
prevRoot=''
cant=0
```

Se inicializan las variables que se utilizaran en el transcurso del programa en estas variables se almacenara los contadores y la información básica del programa y posteriormente se ingresaran en los métodos que se encargaran del entrenamiento del programa.

```
for root, dirnames, filenames in os.walk(imgpath):
    for filename in filenames:
        if re.search("\.(jpg|jpeg|png|bmp|tiff)$", filename):
            cant=cant+1
            filepath = os.path.join(root, filename)
            image = plt.imread(filepath)
            images.append(image)
            b = "Leyendo..." + str(cant)
            print (b, end="\r")
            if prevRoot !=root:
                print(root, cant)
                prevRoot=root
                directories.append(root)
                dircount.append(cant)
                cant=0
                dircount.append(cant)
```

Con un ciclo se recorre cada directorio y a su vez cada imagen en cada directorio donde almaceno la información de cada imagen. El ciclo solo se leerán los archivos con extensiones de imagen especificadas y el resto de archivos serán ignorados. La información de las imágenes es guardada de forma binaria en un arreglo que se va construyendo conforme se va avanzando en el ciclo.

```
dircount = dircount[1:]
dircount[0]=dircount[0]+1
print('Directorios leidos:',len(directories))
print("Imagenes en cada directorio", dircount)
print('suma Total de imagenes en subdirs:',sum(dircount))
```

Se imprime en pantalla a nivel informativo la cantidad de imágenes y directorios leídos.

```
labels=[]
indice=0
for cantidad in dircount:
    for i in range(cantidad):
        labels.append(indice)
    indice=indice+1
print("Cantidad etiquetas creadas: ",len(labels))
```

Se extrae e imprime en pantalla las etiquetas que serán usadas para clasificar las imágenes basado en los nombres de cada subcarpeta dentro de nuestra carpeta “Plagas”

```
plagas = []
indice = 0
for directorio in directories:
    name = directorio.split(os.sep)
    print(indice, name[len(name)-1])
    plagas.append(name[len(name)-1])
    indice = indice+1
```

Para las etiquetas anteriormente creadas prepago un arreglo categórico el cual va a servir para los pasos de entrenamiento de la red neuronal.

```
y = np.array(labels)
X = np.array(images, dtype=np.uint8)
```

Convierto la lista de etiquetas a datos Numpy, esto para mayor compatibilidad en pasos posteriores.

```
classes = np.unique(y)
nClasses = len(classes)
print('Total number of outputs : ', nClasses)
print('Output classes : ', classes)
```

Se filtra el arreglo de etiquetas con el método numpy que permite remover duplicados. Se deben remover los duplicados del arreglo de etiquetas ya que serán las clases que usara el algoritmo de entrenamiento a usar. Luego se imprime el resultado final en pantalla.

```
train_X, test_X, train_Y, test_Y = train_test_split(X, y, test_size=0.2)
print('Training data shape : ', train_X.shape, train_Y.shape)
print('Testing data shape : ', test_X.shape, test_Y.shape)
```

Se crean los respectivos sets, dos para entrenamiento y 2 para pruebas. Se imprimen los resultantes en pantalla.

```
plt.figure(figsize=[5, 5])

plt.subplot(121)
plt.imshow(train_X[0, :, :], cmap='gray')
plt.title("Ground Truth : {}".format(train_Y[0]))

plt.subplot(122)
plt.imshow(test_X[0, :, :], cmap='gray')
plt.title("Ground Truth : {}".format(test_Y[0]))
```

Se inicializa un objeto de gráfico y se imprimen los objetos de entrenamiento previamente creados. Se generan grafico para el set de entrenamiento como para el de pruebas.

```
train_X = train_X.astype('float32')
test_X = test_X.astype('float32')
train_X = train_X / 255.
test_X = test_X / 255.
```

Se pre-procesan las imágenes, Tanto las del set de entrenamiento como para el de pruebas. Indicamos también que el tipo de datos de nuestro set es de números flotantes de 32 bits.

```
train_Y_one_hot = to_categorical(train_Y)
test_Y_one_hot = to_categorical(test_Y)

print('Original label:', train_Y[0])
print('After conversion to one-hot:', train_Y_one_hot[0])
```

Se convierten las matrices categóricas de entrenamiento y de pruebas en matrices de binarios, esto retornara columnas iguales al número de categorías en los datos. Se imprimen los resultados en pantalla.

```
train_X, valid_X, train_label, valid_label = train_test_split(
    train_X, train_Y_one_hot, test_size=0.2, random_state=13)

print(train_X.shape, valid_X.shape, train_label.shape, valid_label.shape)
```

En estas líneas se crea un set de entrenamiento y validación el cual nos servirá para almacenar los datos de aprendizaje generados por el el algorimo de Convolutional Neural Network (CNN). Luego se imprime la estructura de estas variables en pantalla.

```
INIT_LR = 1e-3
epochs = 6
batch_size = 64
```

Se declaran los parámetros iniciales del modelo CNN. Estas constantes definen la configuración general inicial con la que el algoritmo funcionara en su entrenamiento y posterior clasificación de nuevas imágenes. Cambiar estas constantes puede variar drásticamente el comportamiento del algoritmo.

```
plagas_model = Sequential()
plagas_model.add(Conv2D(32, kernel_size=(3, 3), activation='linear',
                        padding='same', input_shape=(150, 200, 3)))
plagas_model.add(LeakyReLU(alpha=0.1))
plagas_model.add(MaxPooling2D((2, 2), padding='same'))
plagas_model.add(Dropout(0.5))

plagas_model.add(Flatten())
plagas_model.add(Dense(32, activation='linear'))
plagas_model.add(LeakyReLU(alpha=0.1))
plagas_model.add(Dropout(0.5))
plagas_model.add(Dense(nClasses, activation='softmax'))

plagas_model.summary()

plagas_model.compile(
    loss=keras.losses.categorical_crossentropy, metrics=['accuracy'])
```

Se declara el modelo principal de entrenamiento. Este modelo es almacenado en el objeto *plagas_model* y añadimos las características y parámetros de configuración de la red neuronal.

Los principales valores a destacar son:

learning rate, la cantidad de iteraciones completas al conjunto de imágenes de entrenamiento y la cantidad de imágenes que se toman a la vez en memoria.

Luego de configurar la red se imprime en pantalla el resumen de toda la configuración aplicada y se procede a compilar el modelo. La instrucción de compilar es equivalente a "*entrenar*" la red neuronal para el reconocimiento de imágenes.

```
plagas_model.save("plagas_mnist.h5py")
```

Se guarda la red resultante del entrenamiento, para reutilizarla en el futuro, sin tener que volver a entrenarla (compilar). Ya que compilar es el proceso que más consume tiempo y recursos del ordenador.

```
test_eval = plagas_model.evaluate(test_X, test_Y_one_hot, verbose=1)

print('Test loss:', test_eval[0])
print('Test accuracy:', test_eval[1])
```

En esta línea se realiza una evaluación de la red anteriormente entrenada. Esto no es más que verificar la integridad de los datos resultantes luego del entrenamiento. Posterior a esto imprimimos el resultado de la evaluación en pantalla.

```
accuracy = plagas_train.history['accuracy']
val_accuracy = plagas_train.history['val_accuracy']
loss = plagas_train.history['loss']
val_loss = plagas_train.history['val_loss']
epochs = range(len(accuracy))
plt.plot(epochs, accuracy, 'bo', label='Training accuracy')
plt.plot(epochs, val_accuracy, 'b', label='Validation accuracy')
plt.title('Training and validation accuracy')
plt.legend()
plt.figure()
plt.plot(epochs, loss, 'bo', label='Training loss')
plt.plot(epochs, val_loss, 'b', label='Validation loss')
plt.title('Training and validation loss')
plt.legend()
plt.show()
```

Se prepara una gráfica para ilustrar los resultados del entrenamiento realizado. Estas líneas no repercuten en el funcionamiento del código y son netamente ilustrativas. Pueden ser removidas si se desea.

```
predicted_classes2 = plagas_model.predict(test_X)

predicted_classes = []
for predicted_plaga in predicted_classes2:
    predicted_classes.append(
        predicted_plaga.tolist().index(max(predicted_plaga)))
predicted_classes = np.array(predicted_classes)

predicted_classes.shape, test_Y.shape
```

Se configura un set de clases de entrenamiento de prueba. Al cual suministra los objetos creados en líneas más arriba, esto servirá para en pasos posteriores la red aprenda de sus propios errores, es decir, pulir la clasificación de imágenes induciendo a clasificaciones errores a propósito e indicar el error.

```
correct = np.where(predicted_classes == test_Y)[0]
print("Found %d correct labels" % len(correct))
for i, correct in enumerate(correct[0:9]):
    plt.subplot(3, 3, i+1)
    plt.imshow(test_X[correct].reshape(150, 200, 3),
               cmap='gray', interpolation='none')
    plt.title("{}, {}".format(
        plagas[predicted_classes[correct]], plagas[test_Y[correct]]))

    plt.tight_layout()
```

s sentencias de prueba y error para que la red aprenda de errores causados adrede. En este caso generamos un escenario en el cual una clasificación es correcta.

```
incorrect = np.where(predicted_classes != test_Y)[0]
print("Found %d incorrect labels" % len(incorrect))
for i, incorrect in enumerate(incorrect[0:9]):
    plt.subplot(3, 3, i+1)
    plt.imshow(test_X[incorrect].reshape(150, 200, 3),
               cmap='gray', interpolation='none')
    plt.title("{}, {}".format(
        plagas[predicted_classes[incorrect]], plagas[test_Y[incorrect]]))
    plt.tight_layout()
```

Para ese caso se repite el código anterior pero esta vez simulamos una predicción incorrecta.

```
target_names = ["Class {}".format(i) for i in range(nClasses)]
print(classification_report(test_Y, predicted_classes, target_names=target_names))
```

Ya con los datos recogidos a través de la ejecución del programa, la red entrenada e incluso almacenada localmente para resumir tiempos posteriores de ejecución. Se muestran los resultados de clasificación finales en pantalla y los nombres de las etiquetas que se generaron para este programa.

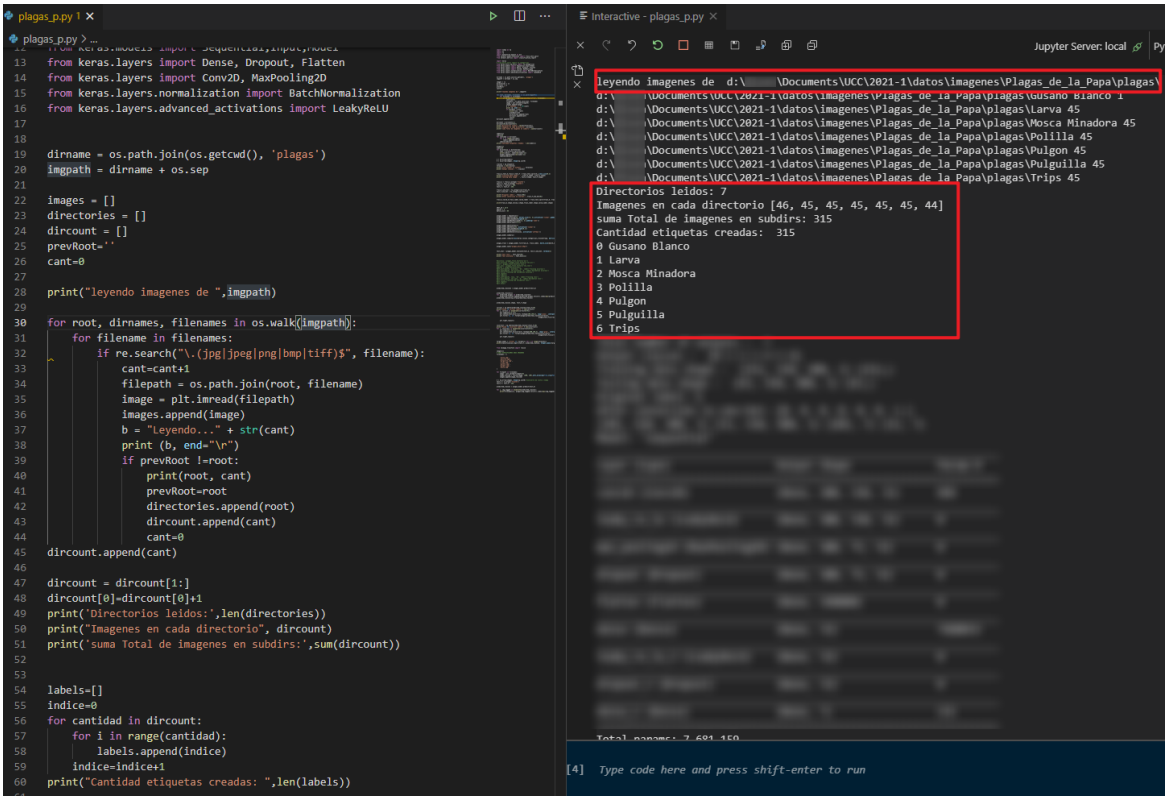
En este punto nuestro programa ya estaría listo para recibir nuevas imágenes e intentar clasificarlas según lo aprendido.

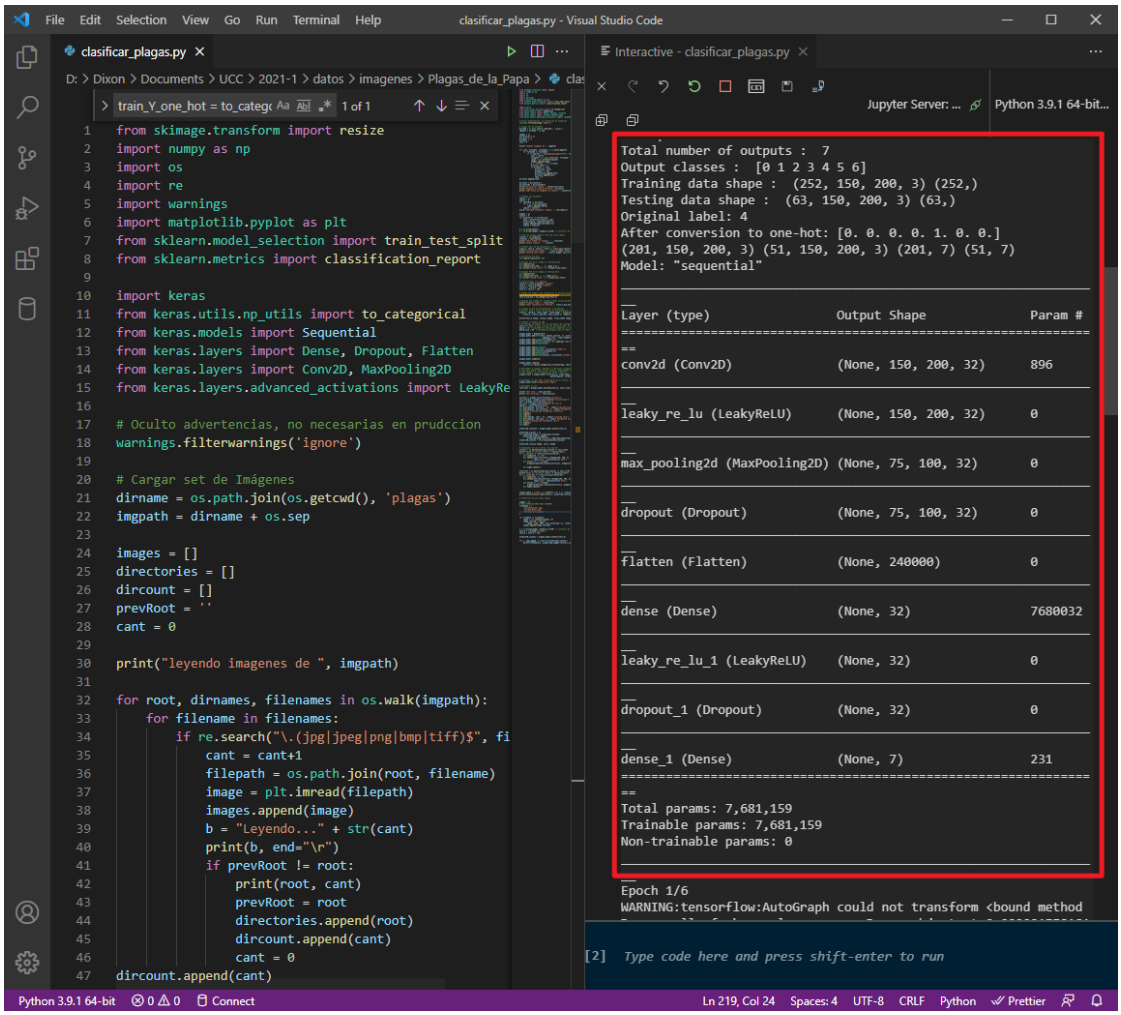
```
images = []
filenames = [
    'entrada/gusano.jpg',
    'entrada/mosca.jpg',
    'entrada/larva.jpg'
]

for filepath in filenames:
    image = plt.imread(filepath, 0)
    image_resized = resize(
        image, (150, 200), anti_aliasing=True, clip=False, preserve_range=True)
    images.append(image_resized)

X = np.array(images, dtype=np.uint8)  # convierto de lista a numpy
test_X = X.astype('float32')
test_X = test_X / 255.

predicted_classes = plagas_model.predict(test_X)

for i, img_tagged in enumerate(predicted_classes):
    print(filenames[i], plagas[img_tagged.tolist().index(max(img_tagged))])
```

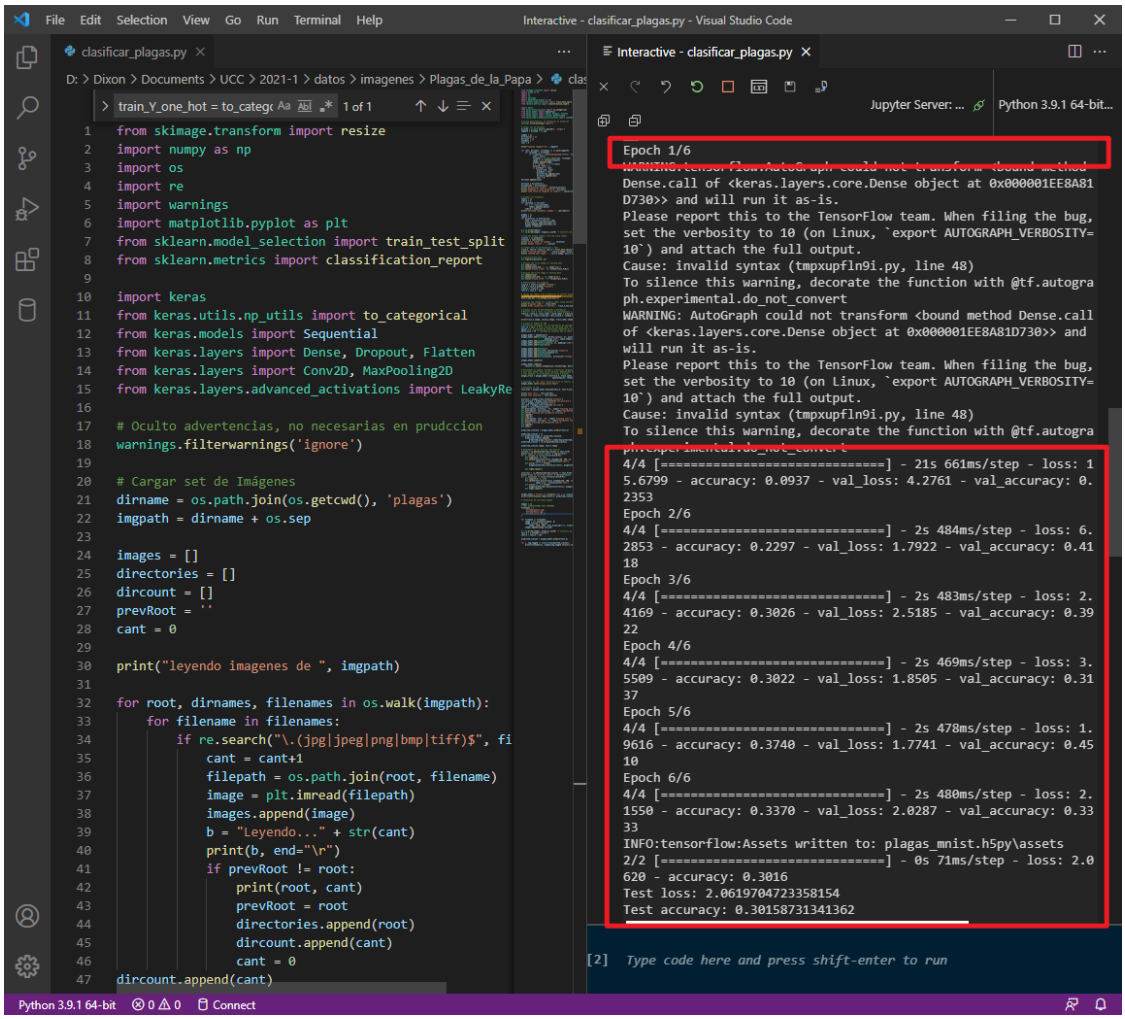
Este código que es indiferente al programa, pero el cual realiza su uso de la red generada, carga 3 imágenes e indica que deben ser procesadas para clasificación. Luego muestra los resultados en pantalla.

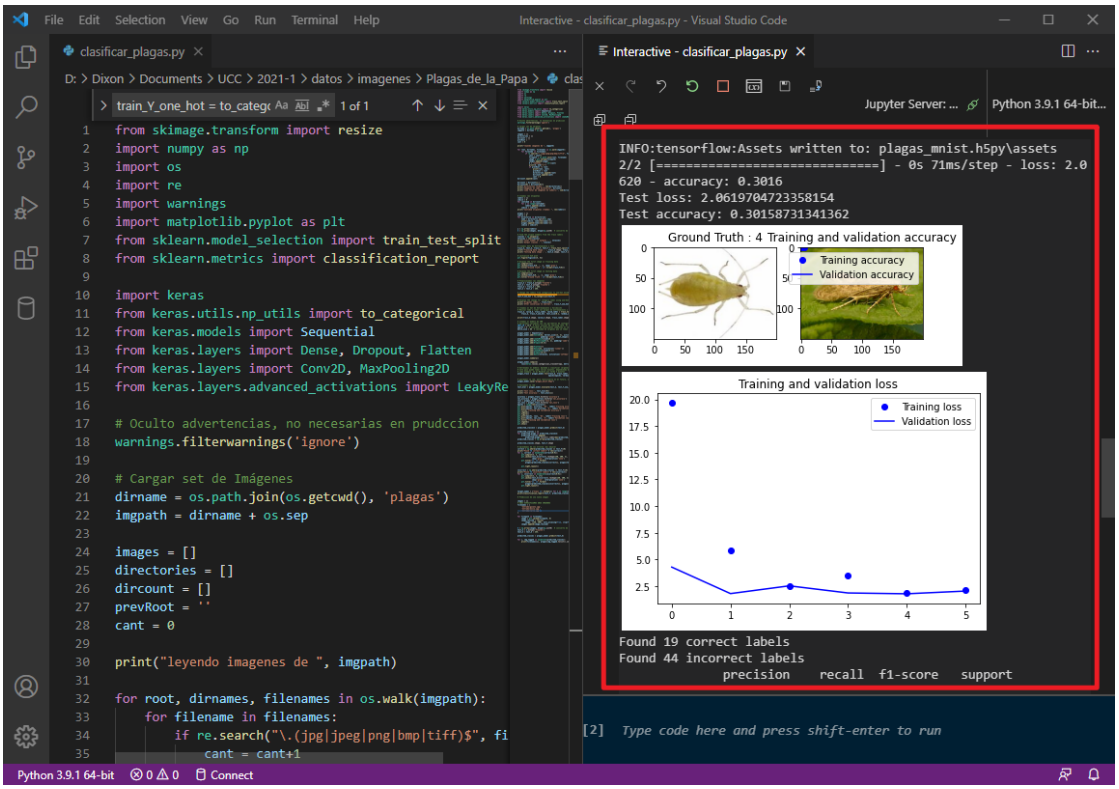
3. Resultados

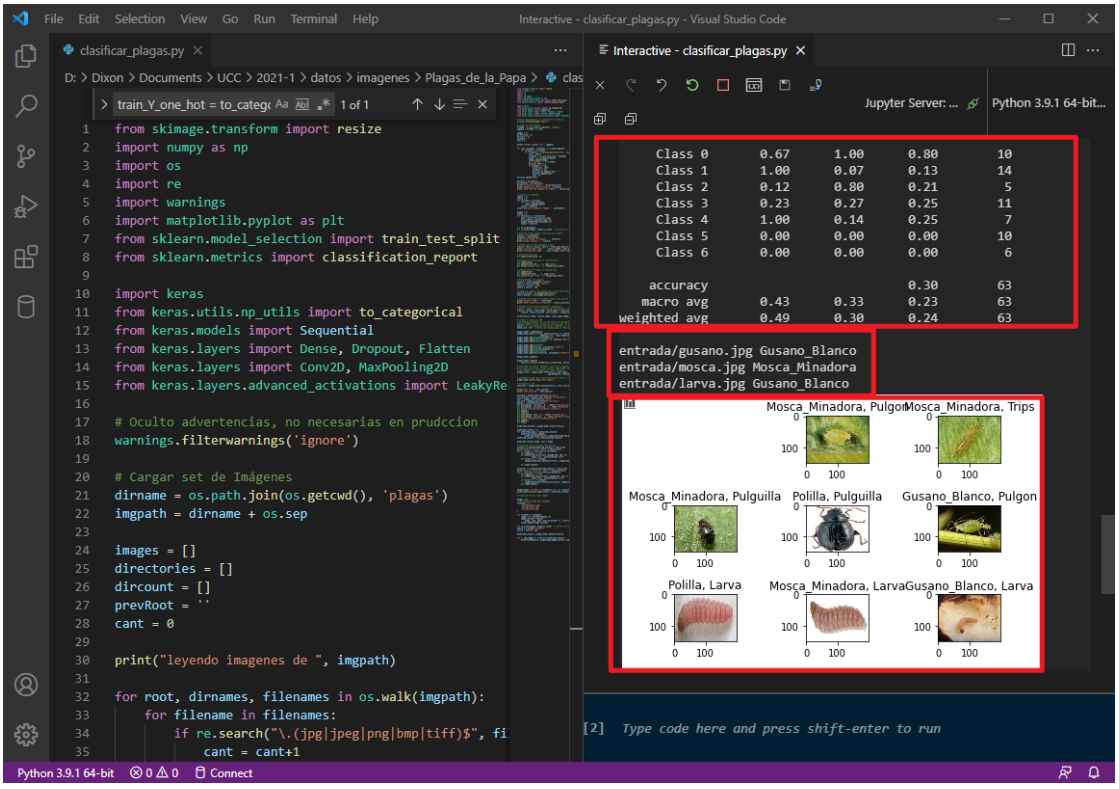
Este al ser un programa por consola, no se obtiene una interfaz en la cual se puedan apreciar los resultados de forma vistosa. No obstante, la ejecución de programa hasta la parte en donde lee los directorios y extrae la información de las imágenes y etiquetas podemos observar un resumen en la consola.




File Edit Selection View Go Run Terminal Help
clasificar_plagas.py - Visual Studio Code
clasificar_plagas.py
Interactive - clasificar_plagas.py
D: > Dixon > Documents > UCC > 2021-1 > datos > imagenes > Plagas_de_la_Papa
train_Y_one_hot = to_categ 1 of 1
Jupyter Server: ...
Python 3.9.1 64-bit...
from skimage.transform import resize
import numpy as np
import os
import re
import warnings
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report
import keras
from keras.utils.np_utils import to_categorical
from keras.models import Sequential
from keras.layers import Dense, Dropout, Flatten
from keras.layers import Conv2D, MaxPooling2D
from keras.layers.advanced_activations import LeakyRe
# Oculto advertencias, no necesarias en prudccion
warnings.filterwarnings('ignore')
# Cargar set de Imágenes
dirname = os.path.join(os.getcwd(), 'plagas')
imgpath = dirname + os.sep
images = []
directories = []
dircount = []
prevRoot = ''
cant = 0
print("leyendo imagenes de ", imgpath)
for root, dirnames, filenames in os.walk(imgpath):
for filename in filenames:
if re.search("\.(jpg|jpeg|png|bmp|tiff)$", fi
cant = cant+1
filepath = os.path.join(root, filename)
image = plt.imread(filepath)
images.append(image)
b = "Leyendo..." + str(cant)
print(b, end="\r")
if prevRoot != root:
print(root, cant)
prevRoot = root
directories.append(root)
dircount.append(cant)
cant = 0
dircount.append(cant)
Total number of outputs : 7
Output classes : [0 1 2 3 4 5 6]
Training data shape : (252, 150, 200, 3) (252,)
Testing data shape : (63, 150, 200, 3) (63,)
Original label: 4
After conversion to one-hot: [0. 0. 0. 0. 1. 0. 0.]
(201, 150, 200, 3) (51, 150, 200, 3) (201, 7) (51, 7)
Model: "sequential"
Layer (type) Output Shape Param #
conv2d (Conv2D) (None, 150, 200, 32) 896
leaky_re_lu (LeakyReLU) (None, 150, 200, 32) 0
max_pooling2d (MaxPooling2D) (None, 75, 100, 32) 0
dropout (Dropout) (None, 75, 100, 32) 0
flatten (Flatten) (None, 240000) 0
dense (Dense) (None, 32) 7680032
leaky_re_lu_1 (LeakyReLU) (None, 32) 0
dropout_1 (Dropout) (None, 32) 0
dense_1 (Dense) (None, 7) 231
Total params: 7,681,159
Trainable params: 7,681,159
Non-trainable params: 0
Epoch 1/6
WARNING:tensorflow:AutoGraph could not transform <bound method
[2] Type code here and press shift-enter to run
Python 3.9.1 64-bit
0 0
Connect
Ln 219, Col 24
Spaces: 4
UTF-8
CRLF
Python
Prettier


File Edit Selection View Go Run Terminal Help
Interactive - clasificar_plagas.py - Visual Studio Code
clasificar_plagas.py
Interactive - clasificar_plagas.py
D: > Dixon > Documents > UCC > 2021-1 > datos > imagenes > Plagas_de_la_Papa
train_Y_one_hot = to_categ 1 of 1
Jupyter Server: ...
Python 3.9.1 64-bit...
from skimage.transform import resize
import numpy as np
import os
import re
import warnings
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report
import keras
from keras.utils.np_utils import to_categorical
from keras.models import Sequential
from keras.layers import Dense, Dropout, Flatten
from keras.layers import Conv2D, MaxPooling2D
from keras.layers.advanced_activations import LeakyRe
# Oculto advertencias, no necesarias en prudccion
warnings.filterwarnings('ignore')
# Cargar set de Imágenes
dirname = os.path.join(os.getcwd(), 'plagas')
imgpath = dirname + os.sep
images = []
directories = []
dircount = []
prevRoot = ''
cant = 0
print("leyendo imagenes de ", imgpath)
for root, dirnames, filenames in os.walk(imgpath):
for filename in filenames:
if re.search("\.(jpg|jpeg|png|bmp|tiff)$", fi
cant = cant+1
filepath = os.path.join(root, filename)
image = plt.imread(filepath)
images.append(image)
b = "Leyendo..." + str(cant)
print(b, end="\r")
if prevRoot != root:
print(root, cant)
prevRoot = root
directories.append(root)
dircount.append(cant)
cant = 0
dircount.append(cant)
Epoch 1/6
Dense.call of <keras.layers.core.Dense object at 0x000001EE8A81
D730>> and will run it as-is.
Please report this to the TensorFlow team. When filing the bug,
set the verbosity to 10 (on Linux, `export AUTOGRAPH_VERBOSITY=
10`) and attach the full output.
Cause: invalid syntax (tmpxupfln9i.py, line 48)
To silence this warning, decorate the function with @tf.autogra
ph.experimental.do_not_convert
WARNING: AutoGraph could not transform <bound method Dense.call
of <keras.layers.core.Dense object at 0x000001EE8A81D730>> and
will run it as-is.
Please report this to the TensorFlow team. When filing the bug,
set the verbosity to 10 (on Linux, `export AUTOGRAPH_VERBOSITY=
10`) and attach the full output.
Cause: invalid syntax (tmpxupfln9i.py, line 48)
To silence this warning, decorate the function with @tf.autogra
4/4 [==========] - 21s 661ms/step - loss: 1
5.6799 - accuracy: 0.0937 - val_loss: 4.2761 - val_accuracy: 0.
2353
Epoch 2/6
4/4 [==========] - 2s 484ms/step - loss: 6.
2853 - accuracy: 0.2297 - val_loss: 1.7922 - val_accuracy: 0.41
18
Epoch 3/6
4/4 [==========] - 2s 483ms/step - loss: 2.
4169 - accuracy: 0.3026 - val_loss: 2.5185 - val_accuracy: 0.39
22
Epoch 4/6
4/4 [==========] - 2s 469ms/step - loss: 3.
5509 - accuracy: 0.3022 - val_loss: 1.8505 - val_accuracy: 0.31
37
Epoch 5/6
4/4 [==========] - 2s 478ms/step - loss: 1.
9616 - accuracy: 0.3740 - val_loss: 1.7741 - val_accuracy: 0.45
10
Epoch 6/6
4/4 [==========] - 2s 480ms/step - loss: 2.
1550 - accuracy: 0.3370 - val_loss: 2.0287 - val_accuracy: 0.33
33
INFO:tensorflow:Assets written to: plagas_mnist.h5py\assets
2/2 [==========] - 0s 71ms/step - loss: 2.0
620 - accuracy: 0.3016
Test loss: 2.0619704723358154
Test accuracy: 0.30158731341362
[2] Type code here and press shift-enter to run
Python 3.9.1 64-bit
0 0
Connect





En la anterior imagen se resalta el recuadro rojo central la forma en que el algoritmo clasifica las imágenes introducidas. Se puede observar que ha clasificado correctamente 2 de las 3 imágenes

suministradas para clasificación. Esto es un buen indicio ya que debido a la limitada cantidad de imágenes que por ahora fueron suministradas al programa para el entrenamiento se está cerca de un 66% de clasificación correcta.

References

Modelos CNN en la clasificación de imágenes clásicas y modernas https://medium.com/datos-y-ciencia/modelos-cnn-en-la-clasificaci%C3%B3n-de-im%C3%A1genes-cl%C3%A1sicas-y-modernas-d072a6718689

Extraña enfermedad ataca los cultivos de papa http://agriculturadelasamericas.com/agricultura/extrana-enfermedad-ataca-los-cultivos-de-papa/

ICA advierte la presencia de insectos en cultivos de papa en nariño https://www.agronegocios.co/agricultura/ica-advierte-presencia-de-insectos-asociados-a-la-punta-morada-de-la-papa-en-narino-3131144

Manejo y control de plagas en cultivos de Papa https://www.rcnradio.com/colombia/manejo-y-control-de-plagas-y-enfermedades-en-los-cultivos-de-papa

Guía Fotográfica de las Principales Plagas del Cultivo de Papa http://cipotato.org/wp-content/uploads/2013/04/0060841-1.pdf

El algoritmo k-means aplicado a clasificación y procesamiento de imágenes https://www.unioviedo.es/compnum/laboratorios_py/kmeans/kmeans.html

Informe especial: Polilla Guatemalteca o Polilla de la Papa https://www.ica.gov.co/noticias/todas/2016/informe-especial-polilla-guatemalteca-o-polilla-de#:~:text=Da%C3%B1os%20que%20causa%3A&text=La%20Polilla%20Guatemalteca%20ataca%20%C3%BAnicamente,la%20apariencia%20de%20los%20mismos..