Medellín a 13 de junio de 2023

Dr. Raul Ramos Pollan

AI Professor Researcher Universidad de Antioquia - NASA FDL

El objetivo de este informe es presentar el desarrollo de la implementación de un modelo de Deep Learning para la clasificación de imágenes de plantas en la competencia de Kaggle “Plant Pathology 2021- FGVC8” la cual tiene como objetivo la clasificación de las distintas enfermedades que se pueden presentar en los árboles de manzanas, el dataset de imágenes cuenta con 18637 archivos de imágenes que fueron clasificadas previamente. Se desarrolló un modelo con los conocimientos adquiridos en la clase de Fundamentos de Deep Learning impartida por el Dr. Raul Ramos Pollan.

El impacto del desarrollo de este proyecto fue poder aplicar los conocimientos adquiridos en el curso a un problema actual en una plataforma de competencias, esto ayuda al estudiante a poder mejorar sus habilidades y sus conocimientos teóricos acerca de la materia. Se lograron los objetivos de este proyecto, lamentablemente no se contó con la capacidad de almacenamiento para poder con el dataset entero, pero este puede ser obtenido fácilmente y replicar los modelos que se utilizaron durante el proceso del proyecto.

Para poder desarrollar este proyecto fueron necesarias dos herramientas tanto kaggle como Google Colab, ambas plataformas son capaces de poder ejecutar el código, la ventaja de kaggle es que nos proporciona el dataset de manera automática mientras que para Google Colab es necesario que nosotros carguemos nuestros datos.

Los resultados de las distintas iteraciones fueron notorios ya que entre cada iteración la predicción y la exactitud de los modelos fueron mejorando tanto como para Google Colab como para kaggle, otra ventaja de kaggle es que nos puede proporcionar un puntaje de acuerdo con nuestro código, esto es importante ya que kaggle utiliza sus propias métricas para probar nuestros modelos, además de que cuenta con otras imágenes para poder validarlo.

Desarrollo

Obtención de los datos

Se tienen dos maneras de poder acceder a los datos de la competencia de clasificación de enfermedades de plantas “Plant Pathology 2021”, una es mediante el siguiente link: https://www.kaggle.com/competitions/plant-pathology-2021-fgvc8/data , el cual en la parte inferior nos permitirá descargar tres archivos; el archivo de entrenamiento (imágenes sin etiquetas), el archivo de testeo (3 imágenes sin etiquetas), y por ultimo un archivo “csv” en el cual se encontrarán todas las etiquetas de las imágenes del archivo de entrenamiento. La otra manera de conseguir el dataset es la siguiente carpeta ; https://drive.google.com/drive/folders/1R2fWq5aoamOBk8IEbusIeRXD_8Fd3qhC?usp=sharing , el cual contiene la carpeta “train_images2” el cual es el archivo de imágenes de entrenamiento con la cual podrán ejecutarse los notebooks de Google Colab, el link también contiene el archivo “csv” donde vienen las etiquetas de las imágenes de entrenamiento

Nota: Para ejecutar en Google Colab es necesario añadir las carpetas a su propio Drive para poder así extraer las imágenes.

La estructura de los notebooks es la siguiente:

- 01_Exploración_de_datos.ipynb
- 02Modelo_1.ipynb
- 03Modelo_2.ipynb
- 04Modelo_3.ipynb
- 05Modelo_1_kaggle.ipynb
- 06Modelo_2_kaggle.ipynb
- 07Modelo_3_kaggle.ipynb

Los notebooks 01,02,03 y 04 son notebooks que están pensados para poder ser ejecutados en Google Colab, mientras que los notebooks 05,06 y07 son notebooks que son pensados para poder ser ejecutados en kaggle esto porque tienen la finalidad de ser subidos para poder obtener una puntuación en kaggle. Los notebooks 02,03 y 04 tienen la misma estructura de red neuronal que los notebooks 05,06 y 07, solo se desea comparar la puntuación de kaggle con lo que se obtiene en los notebooks 02,03 y 04 . Esta diferencia es clara ya que en los notebooks ejecutados en kaggle tenemos disponibles todos los archivos de entramiento mientras que los otros notebooks son solo una partición del dataset original.

Nota: Para poder ejecutar los notebooks 02,03 y 04, es necesario ejecutar el notebook 01 debido a que primero se exploran los datos y se extraen algunas variables para poder ser utilizados en los siguientes notebooks.

Solución

Primero se debe ejecutar el notebook 01, el cual es dedicado totalmente a trabajar con los datos para poder así extraer algunas variables.

En este notebook se tiene la siguiente estructura; la cual se describirá a continuación

1. Módulos para utilizar: Aquí se cargan las librerías que se van a utilizar para el desarrollo del proyecto
2. Verificar si se utilizar GPU; esto para poder acelerar el tiempo de computo y recordar al usuario utilizar un entorno con GPU
3. Carga de direcciones.

```
train_dir ='/content/drive/MyDrive/archivos_plant/train_images2/'
test_dir = '/content/drive/MyDrive/archivos_plant/test_images/'
df =pd.read_csv('/content/drive/MyDrive/archivos_plant/train.csv')
```

Estas son las direcciones donde se deberá ubicar las carpetas que contienen los archivos antes mencionados, es necesario vincular su propio Drive para poder utilizar las imágenes.

4. Se verifica el número de archivos a entrenar; si se utiliza la segunda opción los archivos deberán ser 4125 imágenes para poder entrenar, en el caso del primero se utilizan 18637 archivos de imágenes.
5. Se hace una clasificación de imágenes, la cual nos dice cuantos tipos de imágenes tenemos clasificadas de acuerdo con el archivo “csv”

```
scab                                4826
healthy                             4624
frog_eye_leaf_spot                  3181
rust                                1860
complex                             1602
powdery_mildew                      1184
scab frog_eye_leaf_spot              686
scab frog_eye_leaf_spot complex      200
frog_eye_leaf_spot complex           165
rust frog_eye_leaf_spot              120
rust complex                          97
powdery_mildew complex                87
Name: labels, dtype: int64
```

6. Debido a que se tienen varias etiquetas, y algunas etiquetas son la combinación de algunas otras, empleamos una estrategia de Binarización para así marcar con 0 y 1 las etiquetas, es decir por cada etiqueta que se nombre en la planta se pondrá un 1, y si no se menciona se pondrá un 0, esto se hace en una copia para no modificar los archivos originales.

```
# converting labels to binary attributes
#Se crea una copia de train
train = df.copy()
#Se crea una lista de etiquetas para la copia de train
train['labels'] = df['labels'].apply(lambda string: string.split(' '))

#Transforma las etiquetas para en un formato binario
s = list(train['labels'])
mlb = MultiLabelBinarizer()

#crea el data frame trainx aplicando las transformaciones de s, los nombres de las columnas se
# se obtienen con mlb.classes, los indices se mantienen de la misma manera que los índices de train

trainx = pd.DataFrame(mlb.fit_transform(s), columns=mlb.classes_, index=train.index)
#agrega la columna de imagenes del dataframe de train a trainx
trainx['image'] = train['image']


#Se verifica coom estan los datos

print(trainx.head(10), trainx.columns)
```

```
   complex  frog_eye_leaf_spot  healthy  powdery_mildew  rust  scab  \
0        0                   0        1               0     0     0
1        1                   1        0               0     0     1
2        0                   0        0               0     0     1
3        0                   0        0               0     0     1
4        1                   0        0               0     0     0
5        0                   0        1               0     0     0
6        0                   0        0               0     1     0
7        0                   0        1               0     0     0
8        1                   0        0               0     0     0
9        0                   0        1               0     0     0
```

7. Se establecen los parámetros para las imágenes

```
#Parametros

TARGET_SIZE = 128
BATCH_SIZE = 64
EPOCHS = 1
DATA_LIMIT = 12000
trainx = trainx[:DATA_LIMIT]
```

8. Se crea un dataset que solo contenga las características de las imágenes que extrajimos del archivo original.

```
# Hacemos un data frame de las imágenes que extrajimos del archivo original

from tensorflow.keras.preprocessing import image
import os

train_image = []
trainxN = []
Y = []
direccion=[]

for i in tqdm(range(trainx.shape[0])):
    try:
        img_path = os.path.join(train_dir, trainx['image'][i])

        #Check if the image file exists
        if not os.path.isfile(img_path):
            continue
        trainxN.append(trainx.iloc[i])

        img = image.load_img(img_path, target_size=(TARGET_SIZE, TARGET_SIZE, 3))
        img = image.img_to_array(img)
        img = img / 255
        direccion.append(img_path)
        train_image.append(img)

        y = trainx.iloc[i].drop(['image'], axis=0)
        Y.append(y)

    except Exception as e:
        print(f"Error al procesar imagen {img_path}: {str(e)}")
        continue

X = np.array(train_image)  #Formato de imagen
Y = np.array(Y)
Y64 = Y.astype(np.int64)
trainxNdf = pd.DataFrame(trainxN)  #Formato de tabla
```

```
100%|██████████| 12000/12000 [09:51<00:00, 20.28it/s]
```

```
len(trainxNdf) #misma cantidad de los archivos que extrajimos
```

```
4125
```

Como podemos observar, nuestro dataset cumple con la cantidad de imágenes que extrajimos, teniendo solo las características de estas imágenes, permitiendo así que podamos extraer cualquier cantidad de imágenes al azar, ya que con este código podemos averiguar cuales fueron y agregarlas a otro dataset sin afectar al original.

Al final podemos visualizar imágenes al azar y ver cómo es que fueron clasificadas

```
]: complex                                0
   frog_eye_leaf_spot                     0
   healthy                                0
   powdery_mildew                         0
   rust                                   0
   scab                                   1
   image                 a1a1d5a4cada586f.jpg
   Name: 4155, dtype: object
```

Extraemos algunas variables y guardamos un archivo con ellas para poder después entrenar nuestros modelos.

```
#Los notebook tienen que estar en la misma dirección para poder cargar las variables

import pickle

# Guardar las variables en un archivo
with open('variables.pkl', 'wb') as f:
    pickle.dump((X,Y,Y64,trainxNdf,trainx,train_dir,test_dir,df,train_files,train_image,trainxN,direccion,TARGET_SIZE,BA

print(os.getcwd())
```

```
/content
```

```
from google.colab import drive
import shutil

# Montar Google Drive
drive.mount('/content/drive')

# Mover el archivo a Google Drive
shutil.move('variables.pkl', '/content/drive/MyDrive/variables.pkl')
```

Ejecución de los modelos

1. De igual manera que en la exploración de datos se cargan los módulos a utilizar y se comprueba si se esta utilizando una GPU en el entorno de ejecución

2. Se cargan las variables que fueron guardadas previamente con el archivo de exploración de datos con el siguiente código, es importante dar los permisos a Google para que este puede almacenar el archivo en nuestro drive

```
]: from google.colab import drive
   import shutil
   drive.mount("/content/drive", force_remount=True)
```

```
Mounted at /content/drive
```

```
]: #Cargar variables

   import pickle
   # Cargar las variables desde el archivo
   with open('/content/drive/MyDrive/variables.pkl', 'rb') as f:
       X,Y,Y64,trainxNdf,trainx,train_dir,test_dir,df,train_files,train_image,trainxN,direccion,TARGET_SIZE,BATCH_SIZE,EPOC
```

3. Se dividen nuestros datos, se utiliza el 10% para poder testear el modelo

Dividimos nuestros datos

```
X_train, X_test, y_train, y_test = train_test_split(X, Y64, test_size=0.1)
```

4. Se define el modelo en nuestro caso, el modelo 1 es el modelo base el cual iremos cambiando sus características para poder ir mejorando el entrenamiento

Definimos la estructura de nuestro Modelo

```
#Modelo_01

from keras.models import Sequential
from keras.layers import Dense, Flatten
from keras.layers import Conv2D

model = Sequential()
model.add(Conv2D(filters=16, kernel_size=(5, 5), activation="relu", input_shape=(TARGET_SIZE,TARGET_SIZE,3)))
model.add(Conv2D(filters=32, kernel_size=(5, 5), activation='relu'))
model.add(Conv2D(filters=64, kernel_size=(5, 5), activation="relu"))
model.add(Conv2D(filters=64, kernel_size=(5, 5), activation='relu'))
model.add(Flatten())
model.add(Dense(128, activation='relu'))
model.add(Dense(64, activation='relu'))
model.add(Dense(6, activation='sigmoid'))
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])

model.summary()
```

La estructura del modelo esta compuesta por capas convolucionales con filtros de 16,32 y 64, una capa de aplanado y después capas densas de 128,64 y 6, estás últimas debido a que tenemos 6 clases que son combinadas para poder establecer a que tipo de hoja de árbol pertenecen, todas las capas tienen una función de activación tipo “relu” mientras que la última tiene una sigmoide, con una función de pérdida “binary_crossentropy”.

La función de pérdida “binary_crossentropy”

$$-\frac{1}{n}\sum_{j=1}^{n}\sum_{i=1}^{c}[y_i \log(p_i) + (1-y_i)\log(1-p_i)]$$

Tomada de: https://rubialesalberto.medium.com/funciones-de-error-con-entropia-cross-entropy-y-binary-cross-entropy-8df8442cdf35, Es usada para problemas binarios de clasificación.

5. Después se entrena en el modelo con los parámetros previamente establecidos, en los modelos que no fueron ejecutados en kaggle se utilizaron 50 épocas para entrenar, mientras que en los notebooks que fueron ejecutados en kaggle vario de 10 o 50, esto por el tiempo que tardaba en ejecutar cada notebook.
6. Debido al manejo de las imágenes se realizó código para averiguar las etiquetas reales de la parte de imágenes que se utilizaron para testear y comprarlas con las que predice el modelo.

```
#Obtenemos un dataframe pero solo de la parte de test
#Hay que ordenarlo porque queda desordenado

from tensorflow.keras.preprocessing import image
import os

direccion_test=[]
dataframe_test = []
img_comprobacion=[]

for i in tqdm(range(len(trainx))):

        img_path = os.path.join(train_dir, trainx['image'][i])

        #Check if the image file exists
        if not os.path.isfile(img_path):
            continue

        img = image.load_img(img_path, target_size=(TARGET_SIZE, TARGET_SIZE, 3))
        img = image.img_to_array(img)
        img = img / 255

        for j in range(len(X_test)):
         if np.array_equal(img, X_test[j]):
            direccion_test.append(img_path)
            dataframe_test.append(trainx.iloc[i])
            img_comprobacion.append(img)  #img_comprobacion va igual que dataframe_test
            break #Esto es para que no se repita

dataframeN_test = pd.DataFrame(dataframe_test)
img_comprobacion=np.array(img_comprobacion)

#Hay que encontrar donde image_path es igual, ahí encontraremos
```

Asignamos las etiquetas reales a nuestros datos de prueba, esto comparándolo con las imágenes que extrajimos del archivo de entrenamiento y asignándoles la etiqueta con el archivo “csv”.

```
##Con este código nos damos cuenta cómo estan ordenados X_test con respecto a nuestro data frame

cuenta=0
indice_X_test=[]
indice_img_comprobacion=[]

for i in range(len(X_test)):
    for j in range(len(X_test)):
        if np.array_equal(img_comprobacion[j],X_test[i]): #Las ordenamos con respecto a X_test, con eso ordenamos el dat
            indice_X_test.append(i)
            indice_img_comprobacion.append(j)
            cuenta=cuenta+1 ##lo lograremos cuando corramos de nuevo el código y coindican las i y j
print(cuenta)
indice_X_test=np.array(indice_X_test)
indice_img_comprobacion=np.array(indice_img_comprobacion)
```

```
##Ordenando las imagenes con respecto a X_test, dataframeN_test tiene que quedar ordenado
##Nuestros array son img_comprobacion y X_test
## Si se logra ordenar, tenemos que ordenar el dataframeN_test

df_img_comprobacion_test=[]
dataframeNN_test=[]

for i in range(len(indice_X_test)):
    dato_ordenado=indice_img_comprobacion[i]
    df_img_comprobacion_test.append(img_comprobacion[dato_ordenado])
    dataframeNN_test.append(dataframeN_test.iloc[dato_ordenado])

df_img_comprobacion_test=np.array(df_img_comprobacion_test)
dataframeNN_test = pd.DataFrame(dataframeNN_test)
```

Con este código se ordenan los archivos de prueba, con los archivos de prueba ya con etiquetas, esto es para poder comparar.

```
## Se comprueba que si estan ordenados :)
cuenta=0
indice_X_test=[]
indice_img_comprobacion=[]
for i in range(len(X_test)):
    for j in range(len(X_test)):
        if np.array_equal(df_img_comprobacion_test[j],X_test[i]): #Las ordenamos con respecto a X_test, con eso ordenamo
            #print(i,j)
            cuenta=cuenta+1 ##lo lograremos cuando corramos de nuevo el código y coindican las i y j
print(cuenta)
```

Con este código comprobamos cómo están ordenados, la prueba es que los índices que se imprimen sean iguales.

7. Después se le asigna el nombre del tipo de enfermedad a las etiquetas que fueron predichas con el modelo, asignándoles un rango de probabilidad, si se sobrepasa ese valor se asignará la etiqueta o las etiquetas.

```
name = {0: 'complex',
        1: 'frog_eye_leaf_spot',
        2: 'healthy',
        3: 'powdery_mildew',
        4: 'rust',
        5: 'scab'}

threshold = {0: 0.3,
             1: 0.3,
             2: 0.4,
             3: 0.3,
             4: 0.3,
            5:0.3}

y_predictN=[]

for j in  range(len(dataframeNN_test)) :  #train_files:  #test files son los de prueba que nos proporcionó kaggle

    #Se carga la imagen del test_file
    columnas = y_predict[j]
    d = []
    for i in range(len(columnas)):
        if columnas[i] > threshold[i]:
            d.append(name[i])

    y_predictN.append({'image': dataframeNN_test.iloc[j]['image'], 'labels': ' '.join(d)})
```

8. Teniendo esto ya se tienen las etiquetas verdaderas y las etiquetas predichas por lo cual se procede a hacer la matriz de confusión

```
import numpy as np
from sklearn.metrics import confusion_matrix

# Obtén las etiquetas verdaderas y predichas en formato de lista
y_truecm = [data['labels'] for data in y_true]
y_predcm = [data['labels'] for data in y_predictN]

# Obtén todas las clases únicas presentes en las etiquetas verdaderas y predichas
#classes = np.unique(np.concatenate((y_truecm, y_predcm)))
classes=np.unique(y_truecm)
# Crea la matriz de confusión
cm = confusion_matrix(y_truecm, y_predcm, labels=classes)
```

```
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

# Configura la figura y los ejes
plt.figure(figsize=(10, 10))
sns.set(font_scale=1)  # Ajusta el tamaño de la fuente para las etiquetas de los ejes
sns.heatmap(cm, annot=True, cmap='Greens', fmt='.2f', xticklabels=classes, yticklabels=classes)

# Configura los títulos y las etiquetas de los ejes
plt.title('Matriz de Confusión')
plt.xlabel('Etiqueta Predicha')
plt.ylabel('Etiqueta Verdadera')

# Muestra la gráfica
plt.show()
```

Este es el código para poder graficar la matriz de confusión.

Resultados

Los resultados se toman de la última época de entrenamiento que se hizo, estos resultados pueden ser verificados en el repositorio GitHub

Modelo	Función de pérdida en los datos de entrenamiento	Exactitud en los datos de entrenamiento	Función de pérdida en los datos de validación	Exactitud en los datos de validación	Épocas de entrenamiento	No. De imágenes
02Modelo_1	6.4427e-4	0.9607	2.1184	0.3777	50	4125
03Modelo_2	0.1238	0.8486	0.7300	0.3874	50	4125
04Modelo_3	0.1815	0.7834	0.3519	0.5835	50	4125
05Modelo_1_kaggle	0.0262	0.9478	0.8121	0.5370	10	10000
06Modelo_1_kaggle	0.4209	0.3526	0.4023	0.4090	10	10000
07Modelo_1_kaggle	0.3082	0.594	0.3115	0.5610	10	12000

La parte de resultados la podemos dividir en dos partes, la parte donde de los modelos de 02 al 04 que son modelos que fueron ejecutados con 4125 imágenes durante 50 épocas mientras que la segunda parte es del modelo 05 al modelo 07 que fueron ejecutados durante 10 épocas con 10000 imágenes, la segunda parte no fue entrenada con todas las imágenes disponibles (18637) debido a que los entrenamientos tardaron entre 5 y 7 horas en ejecución con lo cual hacerlo con el total de imágenes tardaría aún más. El comportamiento que se puede observar con la primera parte de los notebooks es que existe “overfitting” debido a que la exactitud de los datos de entrenamiento ya que se tienen valores altos de exactitud mientras que en la predicción se tienen valores bajos de exactitud, esto va mejorando como van avanzando los notebooks esto se debe a que utilizamos técnicas de “dropout” y de “pooling” las cuales nos ayudan a mejorar el rendimiento de nuestros modelos. Por otra parte en la segunda parte de los resultados en los notebooks que fueron ejecutados en kaggle tenemos un comportamiento similar a los primeros en término de “overfitting”, mientras que el notebook 06, ya no tuvo el mismo comportamiento pero no mejoró la exactitud del modelo tanto en validación como en datos de entrenamiento, por último tenemos el modelo 07 que fue ejecutado en kaggle siendo la misma estructura del modelo 04, este tuvo un mejor rendimiento que los notebooks previos de kaggle teniendo menos cantidad de “overfitting” ya que la exactitud tanto de la validación como de los datos de entrenamiento fue muy similar, esto lo podríamos mejorar aumentando más épocas de entrenamiento ya que fue el modelo que menos tardo en ejecutarse.

Un beneficio extra de kaggle es que nos permite subir nuestros códigos y ellos evalúan estos con otros datos que no se hacen públicos, pero también es una forma de medir el rendimiento de nuestros modelos.