CLASIFICACIÓN DE HOJAS DE PLANTAS DE DIFERENTES ESPECIES A PARTIR DE IMÁGENES DE ÁREA FOLIAR CON AFECTACIONES DE DIVERSAS ENFERMEDADES

Presentado por:

Tatiana Camila Puentes Escobar y Diego Alejandro Grajales González

Materia:

Fundamentos de Deep Learning

Profesor:

Raul Ramos Pollan

Universidad de Antioquia

Facultad de Ingeniería

Medellín

2023

1. Contexto de aplicación

La producción agrícola, eje central de la seguridad alimentaria, depende de las condiciones del medio ambiente, por lo que, se ve afectada por el cambio climático, la contaminación del recurso hídrico y la calidad del suelo (Lira-Saldivar & Lara-Viveros, 2023). Por otro lado, las plagas y enfermedades tienen un impacto negativo en el rendimiento y la calidad de las cosechas (Lira-Saldivar & Lara Viveros, 2023), agravado por el escaso conocimiento técnico de los productores sobre el manejo de estas, y la inapropiada asistencia técnica que hacen que las aplicaciones de plaguicidas in situ sean inadecuadas (Parra et al., 2012).

En contrapartida, que el rendimiento de los cultivos dependa de las condiciones del ambiente, ha generado soluciones basadas en tecnologías de inteligencia artificial (IA) en las áreas de monitoreo y establecimiento de los cultivos, la detección de variables físicas y nutricionales del suelo, presencia de malezas y plagas, entre otras (Saxena et al., 2023; Lira-Saldivar & Lara-Viveros, 2023). Sudar et al. (2022) aplicó sistemas de IA para el análisis y categorización de plagas y enfermedades del cultivo de tomate en etapa temprana, con base en conocimientos agrónomos, determinando que pueden ayudar a limitar la transmisión de patógenos en el área sembrada agrícola y al mismo tiempo, ayudar a aumentar el rendimiento de los cultivos (Lira-Saldivar & Lara-Viveros, 2023). Por tal motivo, es factible la aplicación de modelos de redes neuronales para brindar soluciones y estrategias para que los cultivos mejoren los rendimientos y reduzcan los costos de la implementación del paquete tecnológico de producción agronómica.

En este sentido, a partir de una *dataset* de imágenes disponible se pretende “Emplear una arquitectura de transfer learning basado en redes neuronales convolucionales (CNN) para la clasificación de hojas de plantas de diferentes especies a partir de imágenes de área foliar con afectaciones de diversas enfermedades”

2. Exploración de datos

2.1. Preparación del entorno de Kaggle

En primer lugar, se creó un entorno de Kaggle en Colab, donde es necesario instalar la biblioteca de Kaggle, cargar nuestras credenciales de Kaggle, y descargar y descomprimir el *Dataset* objetivo, para poder acceder y trabajar con el mismo.

2.2. Creación de un conjunto de datos de metadatos

Una vez se obtiene el *Dataset*, se emplean las imágenes de color, las cuales equivalen a 54.305 archivos, para crear un *Data Frame* que sirve como conjunto de datos de la información de las imágenes del directorio proporcionado, donde cada fila contiene la categoría (equivalente a 38 clases) y la ruta de la imagen correspondiente.

	CATEGORY	IMAGE_PATH
0	Tomato___Bacterial_spot	/content/plantvillage dataset/color/Tomato___B...
1	Tomato___Bacterial_spot	/content/plantvillage dataset/color/Tomato___B...
2	Tomato___Bacterial_spot	/content/plantvillage dataset/color/Tomato___B...
3	Tomato___Bacterial_spot	/content/plantvillage dataset/color/Tomato___B...
4	Tomato___Bacterial_spot	/content/plantvillage dataset/color/Tomato___B...
...	...	...
54300	Tomato___Spider_mites Two-spotted_spider_mite	/content/plantvillage dataset/color/Tomato___S...
54301	Tomato___Spider_mites Two-spotted_spider_mite	/content/plantvillage dataset/color/Tomato___S...
54302	Tomato___Spider_mites Two-spotted_spider_mite	/content/plantvillage dataset/color/Tomato___S...
54303	Tomato___Spider_mites Two-spotted_spider_mite	/content/plantvillage dataset/color/Tomato___S...
54304	Tomato___Spider_mites Two-spotted_spider_mite	/content/plantvillage dataset/color/Tomato___S...

Figura 1. *Data Frame* de la información sobre las imágenes ubicadas en el directorio proporcionado.

2.3. Análisis de los valores atípicos según el criterio de la mediana y de las categorías heredadas

Posteriormente, se procesaron las imágenes del *Dataset*, calculando la mediana de cada imagen después de haber sido convertida a escala de grises y redimensionada, y se almacena esta información en un nuevo *Data Frame* llamado df_medians.

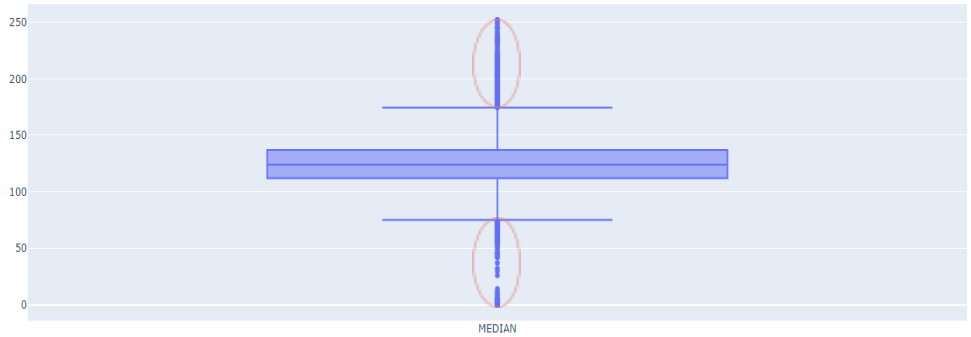


Figura 2. Gráfico boxplot de las medianas de las imágenes identificando *outliers*.

Se puede observar en la parte superior del gráfico que hay imágenes con una intensidad alta, es decir, están saturadas, y en la parte inferior, imágenes oscuras. En consecuencia, se empleó el criterio del rango intercuartílico (IQR) y un criterio adicional definido como el parámetro `CRITERIA` (igual a 2) en las medianas de las imágenes para filtrar aquellos valores atípicos con un rango más amplio.

Figura 3. Muestra aleatoria de imágenes que se consideran outliers (*n=767*) según el criterio definido por el IQR y el parámetro adicional `CRITERIA`.

Se logran identificar en la figura 3 aquellas imágenes que son atípicas, es decir, las imágenes cuya fotografía no fue tomada al área foliar completa de la hoja, o imágenes cuyo brillo, contraste y/o sombra no permitirían una identificación y clasificación correcta. Al aplicar el filtro, el *Dataset* se recude a 53.538.

A continuación, se realizó un análisis exploratorio de los datos, donde se evidenció que las categorías con el mayor número de muestras son: *Orange Haunglongbing (Citrus Greening)*, *Tomato Tomato Yellow Leaf Curl Virus* y *Soybean healthy*. Lo que indica que hay un desequilibrio general entre las diferentes categorías.

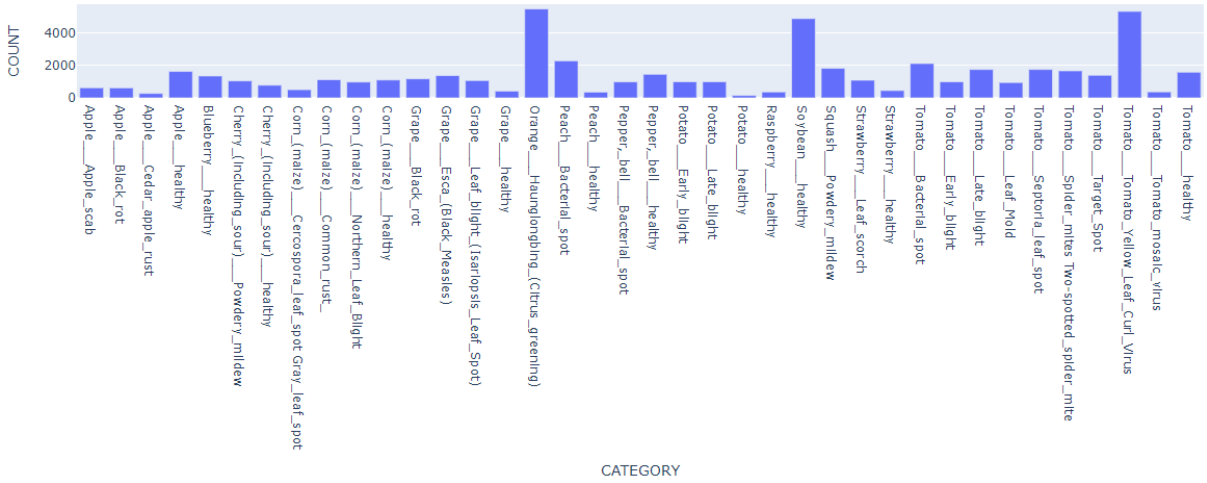


Figura 4. Gráfico de barras con las frecuencias absolutas de las categorías (*i=38*) heredadas del conjunto de datos.

2.4. Transformación del conjunto de datos según el objetivo de machine learning

De acuerdo con lo establecido, nuestro objetivo principal es clasificar hojas de diferentes especies entre saludable y no saludable. Por lo que, se formatea el *Dataset* definiendo las categorías “`healthy`" y "`no healthy`".

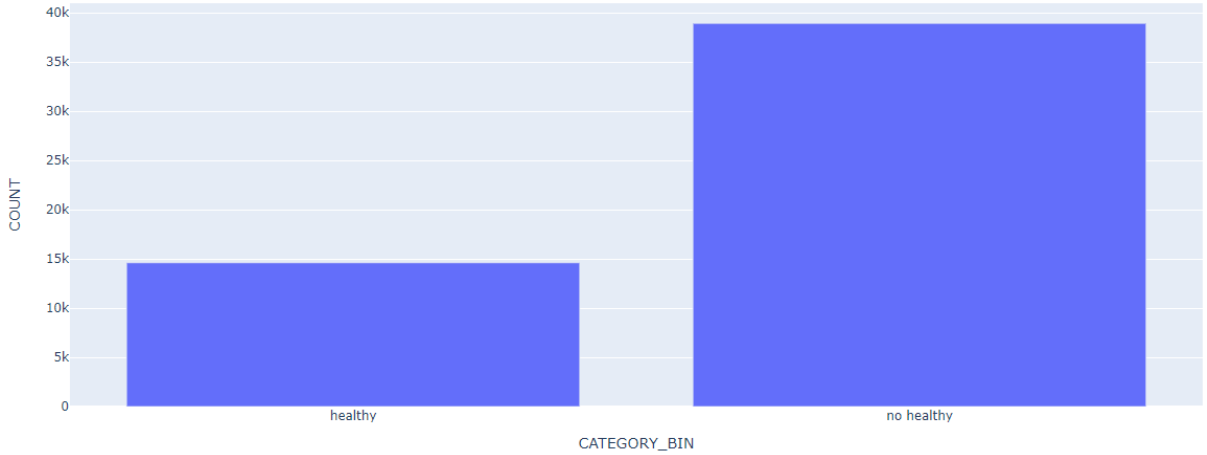


Figura 5. Gráfico de barras con las frecuencias absolutas de las categorías (*i=2*) transformadas

Finalmente, se obtuvieron un total de 14.606 imágenes para la categoría `healthy` y 38.932 imágenes para `no healthy`. Esta información indica que el Dataset conserva el

desequilibrio inicial, donde la categoría `no healthy` tiene 2.6 veces más datos que la categoría healthy, lo que debe tenerse en cuenta durante el entrenamiento del modelo.

3. Preprocesamiento de imágenes

3.1. Conversión de etiquetas de categoría a numéricas

En primer lugar, se emplea la biblioteca `scikit-learn` y el módulo `preprocessing` para convertir las etiquetas de categoría a representaciones numéricas, empleando un `LabelEncoder` para asignar un valor numérico a cada categoría de la columna del *Data Frame* resultado de la transformación del Dataset en las categorías "`healthy`" y "`no healthy`". Posteriormente, se definen las variables `x` y `y` como la ruta de las imágenes y las categorías numéricas, respectivamente.

3.2. *Denoising* básico, eliminación de la mediana y escalado

De acuerdo con el tipo de imágenes descritas en la exploración de datos, se procedió a la definición de funciones para el preprocesamiento de las mismas, y mediante la aplicación de la clase `CustomDataGenerator`, se generaron lotes de datos de imágenes, los cuales corresponden a una función de procesamiento específica según los objetivos de clasificación.

Las funciones empleadas se presentan a continuación,

Filtro para remoción de ruido

```
def basic_preprocessing(path):
  target_resolution = (224, 224)
  img_bgr = cv2.imread(path)
  img_rgb = cv2.cvtColor(img_bgr,
cv2.COLOR_BGR2RGB)
  resize_image = cv2.resize(img_rgb,
target_resolution)
  denoised_image =
cv2.medianBlur(resize_image, 3)
  return denoised_image
```

Filtro para remoción de la mediana

```
def remove_median(path):
    # Retrieve image
    image =
basic_preprocessing(path)

    return abs(image -
np.median(image))
```

Filtro para el escalamiento

```
def scaling(path):
    # Retrieve image
    image =
basic_preprocessing(path)
    image_center =
image.astype(np.float) -
np.median(image)
    max = image_center.max()
    min = image_center.min()
    factor = 1/(max - min)
    processed = factor *
image_center
    return processed
```

Una vez aplicado el filtro a las imágenes, se obtiene un objeto que puede ser usado por `keras` para el entrenamiento.

3.2.1. Filtro para remoción de ruido

Esta función realiza la lectura de imagen, una conversión de formato de color, un redimensionamiento y la reducción de ruido, por medio de un filtro de mediana (`cv2.medianBlur`) con un tamaño del kernel de 3. El resultado se puede observar en la figura 6.

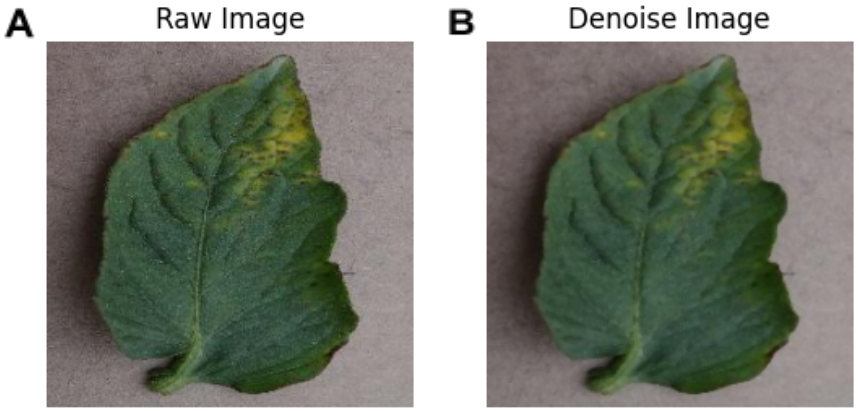


Figura 6. A. Imagen cruda aleatoria del Dataset donde se puede observar el ruido de “pimienta”, es decir, puntos imprecisos en el área foliar de la hoja. **B.** Imagen con la aplicación del filtro de mediana para aumentar la precisión de la afectación del área foliar de la hoja.

3.2.2. Filtro para remoción de la mediana

Luego, a la imagen con el filtro de mediana se le sustrae la mediana, obteniendo una imagen donde se destaca la afectación en el área foliar de la hoja.

Figura 7. A. Imagen cruda aleatoria del Dataset. **B.** Imagen con la aplicación de la resta de la mediana para aumentar la precisión de la afectación del área foliar de la hoja, observándose que el amarillo de la afectación se mantiene.

3.2.3. Filtro para el escalamiento

Finalmente, a la imagen con el preprocesamiento básico, se le realiza un escalado después de centrar en la mediana, normalizando los valores de píxeles para que estén en un rango específico.

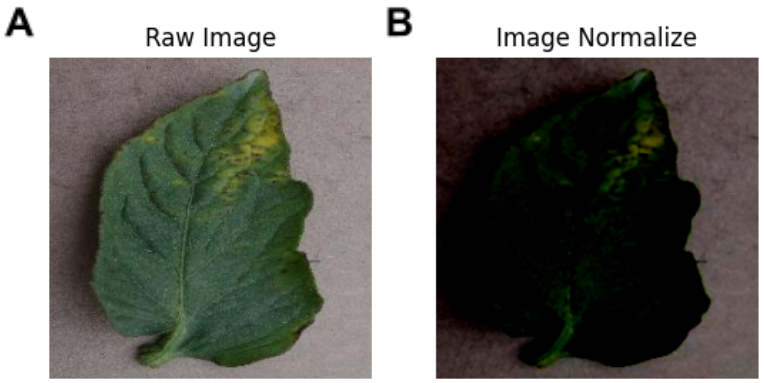


Figura 8. A. Imagen cruda aleatoria del Dataset. **B.** Imagen con la aplicación de la resta de la mediana para aumentar la precisión de la afectación del área foliar de la hoja, observándose que el amarillo de la afectación se mantiene.

4. Modelo

Finalizado el preprocesado de los datos se procedió con la estructuración y entrenamiento del modelo. El proceso inicia teniendo en cuenta el desbalance en las clases del dataset, por ello son clasificadas como clase **1:** `healthy` y **0:** `no healthy`

```
def get_weights(y):
    total = len(y)
    total_1 = len(dataset_df[dataset_df["CATEGORY_BIN_ENCODED"] == 1])
    total_0 = total - total_1
    weight_1 = total_1 / total
    weight_0 = total_0 / total
    class_weight = {0: weight_0, 1: weight_1}
    return class_weight
```

La función `ExperimentSetUp` se utilizó como una clase para el entrenamiento. Se suministraron imágenes RGB de 224x224 píxeles para el entrenamiento de la red. Se entrenó la red con lotes (batches) de 32 imágenes, teniendo en cuenta la capacidad de la GPU con la que se trabajó (Google Colab). Se utilizó la función `class_count` para definir una neurona en la capa de salida de la red, ya que se empleó una función `Sigmoide` que clasifica basada en el resultado mayor o menor a 0.5. Es decir, si el resultado es mayor a 0.5, se clasifica como clase 1; de lo contrario, se clasifica como clase 0.

En el código, x representa las imágenes y y representa las etiquetas. Finalmente, `function` es la función que se aplica según los parámetros definidos en el preprocesamiento, y `scaling` es el escalado que se desea realizar, es decir, la salida se normaliza para que los valores estén entre 0 y 1.

```
class ExperimentSetUp():
    def __init__(self, x, y, function, scaling):
        self.img_size = (224, 224)
        self.channels = 3
        self.batch_size=32
        self.img_shape = (self.img_size[0], self.img_size[1], self.channels)
        self.class_count = 1
```

```
        self.x = x
        self.y = y
        self.function = function
        self.scaling = scaling
        self.epochs = 1
```

Consecuentemente, `prepare_model` como su nombre indica arma el modelo que utiliza una `ResNet50` que a su vez se le agregaron dos (2) capas densas a la salida, un (1) `Dropout` de 0.45 y finalmente la última capa de la neurona que decide si la imagen es de clase 0 o de clase 1.

```
def prepare_model(self):
        base_model = tf.keras.applications.resnet50.ResNet50(
        include_top=True, weights='imagenet', input_tensor=None,
        input_shape=None, pooling="max", classes=1000)
        self.model = Sequential([
            base_model,
            BatchNormalization(axis= -1, momentum= 0.99, epsilon= 0.001),
                    Dense(256, kernel_regularizer= regularizers.l2(l= 0.016), activity_regularizer=
regularizers.l1(0.006),
                        bias_regularizer= regularizers.l1(0.006), activation= 'relu'),
                    Dense(128, kernel_regularizer= regularizers.l2(l= 0.016), activity_regularizer=
regularizers.l1(0.006),
                        bias_regularizer= regularizers.l1(0.006), activation= 'relu'),
            Dropout(rate= 0.45, seed= 123),
            Dense(self.class_count, activation= 'sigmoid')
        ])
             self.model.compile(Adamax(learning_rate= 0.001), loss= 'binary_crossentropy', metrics=
['accuracy'])
        self.model.summary()
```

Para la división del conjunto de datos entre entrenamiento y prueba, se empleó la función *split_test_train* utilizando un 15% para prueba y un 85% para entrenamiento. La elección de estos porcentajes se basó en que el 15% es un valor representativo en función de la cantidad de imágenes presentes en el *Dataset*.

```
def split_test_train(self):
        X_train, X_test, y_train, y_test = train_test_split(
            self.x,
            self.y,
            stratify=self.y,
            test_size=0.15
        )
        self.X_train = X_train
        self.X_test = X_test
        self.y_train = y_train
        self.y_test = y_test
```

Finalmente, en el modelo se aplicó las funciones al conjunto de entrenamiento y testeo y se entrenó. Cada clase del modelo corresponde a un experimento definido en el preprocesamiento del *Dataset*

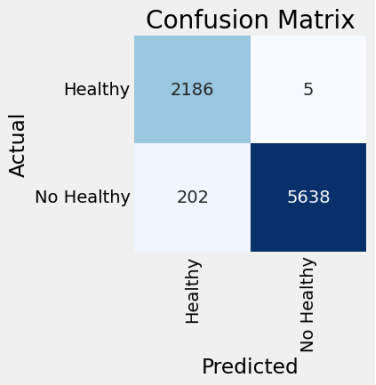
```
def model_training(self):
        self.train_data = CustomDataGenerator(
                self.X_train,
                self.y_train,
                self.batch_size,
```

```
                self.function,
                self.scaling
            )
        self.val_data = CustomDataGenerator(
                self.X_test,
                self.y_test,
                self.batch_size,
                self.function,
                self.scaling
            )
        self.history = self.model.fit(
            self.train_data,
            epochs= self.epochs,
            verbose=1,
            validation_data= self.val_data,
            validation_steps= None,
            shuffle= False,
            class_weight=get_weights(self.y)
        )
```

5. Resultados

Después de ejecutado el modelo para cada uno de los experimentos propuestos se obtuvo la matriz de confusión que muestra en el eje x (columnas) las predicciones del modelo y en el eje y (filas), las clases reales.

Experimento remoción de ruido



Para la clase "Healthy"

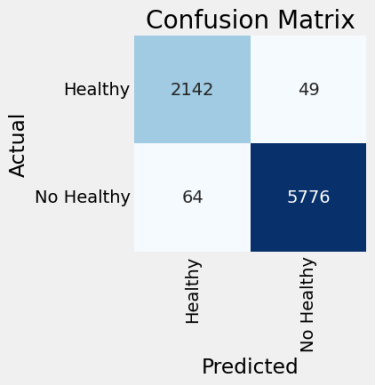
Precisión: 0.92
Sensibilidad: 1.00

Para la clase "No Healthy"

Precisión: 1.00

Sensibilidad: 0.97

Experimento remoción de la mediana



Para la clase "Healthy"

Precisión: 0.97

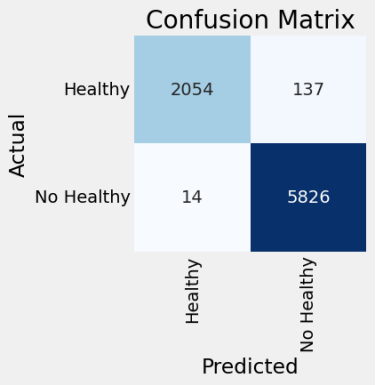
Sensibilidad: 0.98

Para la clase "No Healthy"

Precisión: 0.99

Sensibilidad: 0.99

Experimento de escalamiento



Para la clase "Healthy"

Precisión: 0.99

Sensibilidad 0.94

Para la clase "No Healthy"

Precisión: 0.98

Sensibilidad: 1.00

Comparando los resultados de los tres experimentos en términos de precisión y sensibilidad para las clases "healthy" y "no healthy", se lograron observar variaciones en el rendimiento. En el primer experimento, se logró una precisión del 0.92 para la clase "healthy" con una sensibilidad del 1.00, lo que indica que el 92% de las instancias clasificadas como "healthy" fueron correctas y que el modelo identificó todas las instancias reales de "healthy". Para la clase "no healthy", se obtuvo una precisión y sensibilidad del 1.00 y 0.97, respectivamente.

En el segundo modelo, las métricas mejoraron, con una precisión del 0.97 para la clase "healthy" y una sensibilidad del 0.98. Para la clase "no healthy", la precisión y la sensibilidad alcanzaron el 0.99. Finalmente, el tercer modelo presentó una precisión destacada del 0.99 para la clase "healthy" y una sensibilidad del 0.94. Para la clase "no healthy", la precisión fue de 0.98 y la sensibilidad de 1.00.

En general, el segundo y tercer modelos mostraron mejoras en la precisión para la clase "healthy", alcanzando un máximo de 0.99 en el tercer modelo. La sensibilidad también se mantuvo alta en todos los modelos, indicando que la mayoría de las instancias reales fueron correctamente identificadas. En conclusión, los modelos parecen haber mejorado en la capacidad de identificar tanto instancias positivas como negativas, mostrando un rendimiento sólido en la clasificación global, no obstante, se selecciona el modelo cuya entrada son las imágenes resultado del preprocesamiento de escalamiento.

Bibliografía

Falaschetti, L., Manoni, L., Leo, D. Di, Pau, D., Tomaselli, V., & Turchetti, C. (2022). A CNN-based image detector for plant leaf diseases classification. HardwareX, 12. https://doi.org/10.17605/OSF.IO/UCM8D

Geetharamani, G., & Arun Pandian, J. (2019). Identification of plant leaf diseases using a nine-layer deep convolutional neural network. Computers and Electrical Engineering, 76, 323–338. https://doi.org/10.1016/j.compeleceng.2019.04.011

Lira-Saldivar, R. H., & Lara-Viveros, F. M. (2023). Inteligencia artificial en agricultura - Uso de algoritmos, drones y biosensores. Atena.

Lu, J., Tan, L., & Jiang, H. (2021). Review on convolutional neural network (CNN) applied to plant leaf disease classification. In Agriculture (Switzerland) (Vol. 11, Issue 8). MDPI AG. https://doi.org/10.3390/agriculture11080707

Parra, M., Rivera, P. A., Rodríguez, A., & Aguilar, O. E. (2012). Acuerdo de competitividad para la cadena productiva de Pasifloras en Colombia.