# CLASIFICACIÓN DE HOJAS DE TOMATE A PARTIR DE IMÁGENES DE ÁREA FOLIAR CON AFECTACIONES DE DIVERSAS ENFERMEDADES

**Presentado por:**

**Oswaldo Rivero Romero**

**Materia:**

**Fundamentos de Deep Learning**

**Profesor:**

**Raúl Ramos Pollan**

**Universidad de Antioquia**

**Facultad de Ingeniería**

**Medellín**

**2024**

## 1. INTRODUCCIÓN

La producción agrícola es un pilar fundamental de la economía global, ya que representa la principal fuente de alimentos para la humanidad. En numerosas regiones, especialmente en los países en desarrollo, la agricultura juega un papel crucial en el desarrollo económico, generando empleo, promoviendo la producción de materias primas y contribuyendo significativamente al Producto Interno Bruto (PIB) [1]. En este contexto, la implementación de estrategias que optimicen los resultados agrícolas mediante el monitoreo continuo de los cultivos puede resultar de gran valor, proporcionando recomendaciones y consejos acertados para mejorar el rendimiento de estos. Por lo tanto, la implementación de tecnologías informáticas y la inteligencia artificial (IA) pueden ser una técnica viable para el rendimiento agrícola [2].

El tomate (*Solanum lycopersicum* L.) es uno de los cultivos más populares a nivel mundial, siendo superado únicamente por la papa [3]. Este fruto se utiliza en una amplia variedad de alimentos, desde ensaladas hasta salsas, e incluso como producto natural en el cuidado de la piel [4]. Sin embargo, uno de los principales desafíos que enfrentan los agricultores es la aparición de enfermedades foliares, las cuales pueden reducir el tamaño de los frutos e incluso provocar la muerte de las plantas. Además, el tiempo necesario para diagnosticar dichas enfermedades y aplicar medidas correctivas afecta negativamente el rendimiento de los cultivos.

Para abordar este problema, diversos investigadores han propuesto el uso de técnicas avanzadas de inteligencia artificial, como el *Machine Learning* (ML), para ayudar a los agricultores a tomar decisiones oportunas que optimicen la producción de tomate. Por ejemplo, Ashok et al. (2020) desarrollaron un sistema para la detección temprana de enfermedades en las hojas de plantas de tomate mediante técnicas de procesamiento de imágenes, logrando un nivel de precisión del 98% utilizando un algoritmo de red neuronal convolucional (CNN) [5]. Asimismo, un estudio reciente aplicó filtros de preprocesamiento (Gaussian Blur y Gaussian Noise) y modelos de color (HSI y CMYK) para mejorar la precisión en la clasificación de imágenes de hojas. En este trabajo, se evaluaron modelos DCNN como Vgg-19, MobileNet-V2 y ResNet-50, alcanzando una precisión máxima del 99.53% con ResNet-50 al combinar filtros de ruido gaussiano con la conversión de color de RGB a CMYK, lo que resalta el potencial de estas metodologías para aplicaciones agrícolas [6].

En este trabajo, a partir de un *dataset* de imágenes disponibles de dominio público se pretendió emplear modelo basado en redes neuronales convolucionales (CNN) para la clasificación de hojas de tomate a partir de imágenes de área foliar con afectaciones de diversas enfermedades.

## 2. EXPLORACIÓN DE DATOS

### 2.1. Preparación del entorno de Kaggle

Primero, se configuró un entorno de Kaggle en Colab, lo que implicó instalar la biblioteca de Kaggle, cargar las credenciales correspondientes y descargar y descomprimir el conjunto de datos objetivo para facilitar su acceso y manipulación. Para este proyecto, se cuenta con un total de 10000 imágenes para entrenamiento y un total de 1000 imágenes para la validación.

### 2.2. Distribución de Imágenes por Clase en el Conjunto de Entrenamiento

Una vez obtenido el *Dataset*, se utilizó el conjunto de imágenes de entrenamiento organizado por carpetas para crear un *DataFrame* que permite resumir la distribución de las imágenes por clase. En la Tabla 1, se muestra el *DataFrame* construido. Cada fila del *DataFrame* representa una categoría específica (clase) junto con la cantidad de imágenes disponibles en esa categoría dentro del Dataset.

Tabla 1: *DataFrame* de distribución de las imágenes.

| | Clase | Número de imágenes |
|---|---|---|
| 0 | Tomato___Tomato_Yellow_Leaf_Curl_Virus | 1000 |
| 1 | Tomato___Leaf_Mold | 1000 |
| 2 | Tomato___Spider_mites_Two-spotted_spider_mite | 1000 |
| 3 | Tomato___Late_blight | 1000 |
| 4 | Tomato___healthy | 1000 |
| 5 | Tomato___Tomato_mosaic_virus | 1000 |
| 6 | Tomato___Early_blight | 1000 |
| 7 | Tomato___Bacterial_spot | 1000 |
| 8 | Tomato___Target_Spot | 1000 |
| 9 | Tomato___Septoria_leaf_spot | 1000 |

### 2.3. Visualización de Imágenes del Conjunto de Entrenamiento

En este paso, se busca la visualización de las imágenes del conjunto de entrenamiento con sus etiquetas de clase (Figura 1). Para ello, primero se obtuvo un diccionario que asigna un número a cada clase. Luego, se creó una figura con el tamaño adecuado para mostrar varias imágenes en una cuadrícula. A continuación, se recorrieron las imágenes, mostrando una de cada clase junto con su etiqueta. Esto permitió explorar mediante las imágenes, las enfermedades de las hojas en términos de su afectación en el área forliar.

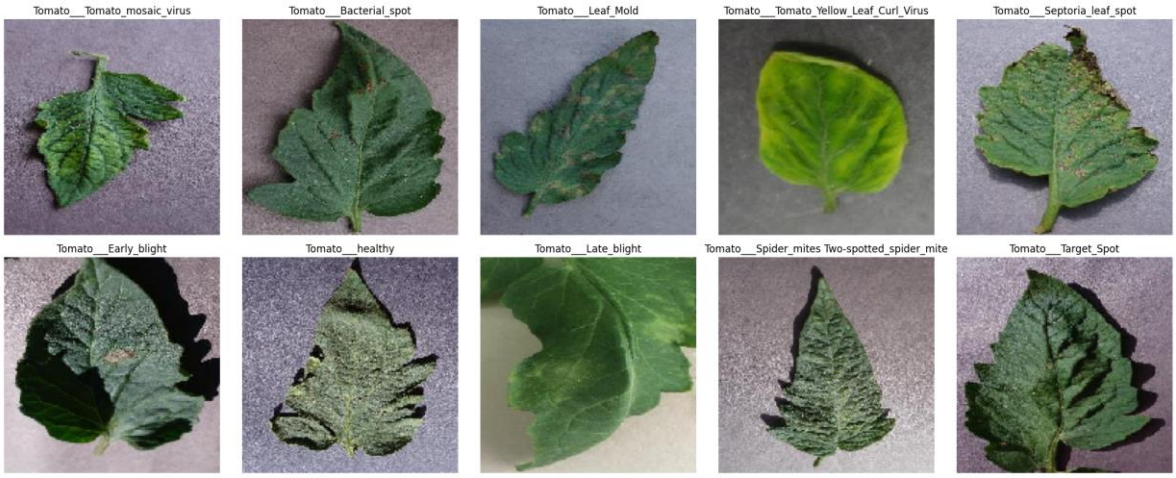


Figura 1: Visualización de Imágenes del Conjunto de Entrenamiento.

### 2.4. Análisis de distribución de las clases

A continuación, se realizó un análisis exploratorio de los datos, para asegurar que no se muestra una distribución desigual que puede dificultar el entrenamiento del modelo. Se observa que el número de imágenes para el entrenamiento están balanceadas para cada clase, lo que indica un equilibrio general entre las clases.

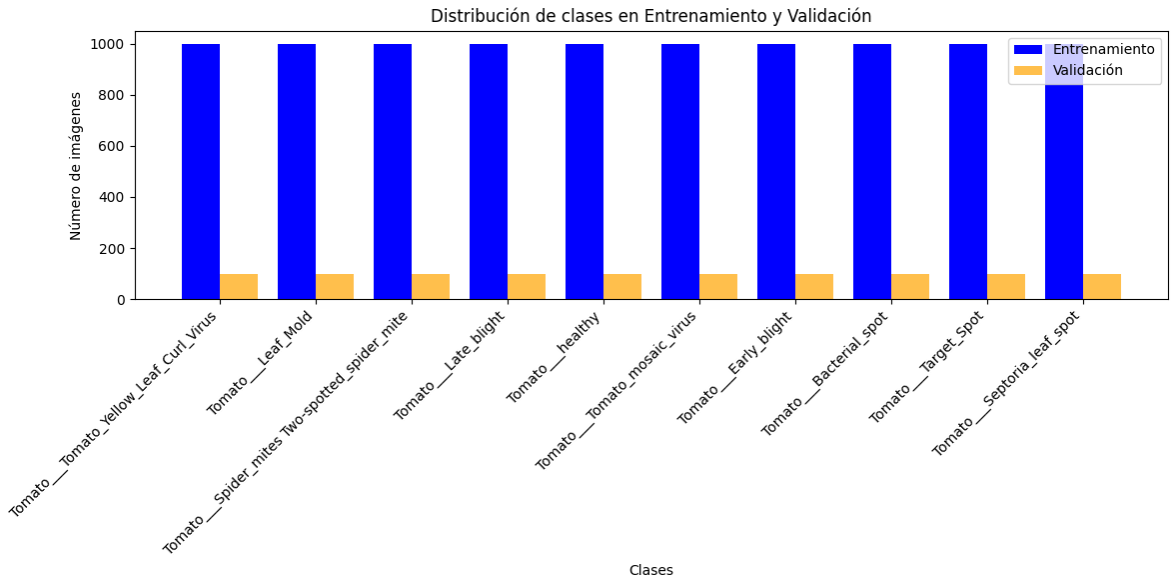


Figura 2: Análisis de distribución de las clases.

### 2.5. Análisis de las dimensiones y formatos de las imágenes

Por último, se verificó que las imágenes presentaran tamaños o formatos inconsistentes. Se obtuvo el siguiente resultado:

```
Tamaños de las imágenes: Counter({(256, 256): 10000})
Formatos de las imágenes: Counter({'JPEG': 10000})
```

### 2.6. Balanceo del modelo

El conjunto de datos implementados en este modelo para el entrenamiento es de 1000 imágenes por cada clase, mientras que para la validación se utiliza una cantidad de 100 imágenes. Esto da una proporción desbalanceada de los datos de 10. Sin embargo, esta configuración es común cuando se dispone de un gran volumen de datos, ya que más datos de entrenamiento pueden ayudar a mejorar la capacidad del modelo para aprender patrones y características complejas [1,2].

```
Número total de imágenes de entrenamiento: 10000
Número total de imágenes de validación: 1000
Proporción entrenamiento-validación: 10.00
```

## 3. PREPROCESAMIENTO DE IMÁGENES

### 3.1. Redimensionamiento y normalización de las imágenes

En primer lugar, se creó un generador de imágenes con *ImageDataGenerator* de *Keras* para preprocesar los datos antes de entrenar el modelo. Las imágenes fueron normalizadas escalando los valores de píxeles a un rango de [0, 1]. Luego, se redimensionaron todas las imágenes a un tamaño uniforme de 150x150 píxeles. Los generadores de entrenamiento y validación se configuraron para cargar imágenes en lotes de 32 y las etiquetas se codificaron en formato *one-hot* para clasificación multiclase. Esto permitió optimizar la carga y el procesamiento de los datos para el modelo implementado.

### 3.2. Generación de Imágenes Aumentadas para el Entrenamiento

Este código aplica aumentación de datos en tiempo real durante el entrenamiento para mejorar la capacidad de generalización del modelo. Se utiliza *ImageDataGenerator* con técnicas de aumento, como rotación, desplazamiento en ancho y alto, corte (shear), zoom y volteo horizontal. Además, se normalizó los valores de los píxeles a un rango de [0, 1]. Las imágenes generadas son redimensionadas a 150x150 píxeles y se cargan en lotes de 32. El generador de entrenamiento realiza estas transformaciones en las imágenes de manera aleatoria, creando variaciones de las imágenes originales para enriquecer el conjunto de datos y evitar sobreajuste. Por otro lado, para el conjunto de validación solo se realizó la normalización sin aumentación.

```
augmented_datagen = ImageDataGenerator(
    rescale=1./255,
    rotation_range=40,
    width_shift_range=0.2,
    height_shift_range=0.2,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True,
    fill_mode='nearest'  # Completar espacios vacíos con bordes o reflejos
)

train_generator = augmented_datagen.flow_from_directory(
    train_dir,
    target_size=(150, 150),
    batch_size=32,
    class_mode='categorical'
)
```

### 3.3. Codificación de Etiquetas y Conversión a One-Hot

En esta etapa, se realizó la codificación de las etiquetas de clase de las imágenes en formato numérico y *one-hot* para su uso en el modelo de entrenamiento. Primero, se extrae las etiquetas generadas por los generadores de datos *train_generator* y *val_generator* que contienen las clases de las imágenes. Luego, se mapea las clases a índices para conocer su orden. Después, se utilizó *to_categorical* para convertir las etiquetas numéricas en formato *one-hot*, el cual es un formato binario comúnmente utilizado en modelos de clasificación múltiple. A continuación, se muestra las primeras etiquetas numéricas y sus correspondientes representaciones en formato *one-hot*, junto con el mapeo de índices a clases.

```
Etiquetas de entrenamiento (numéricas): [0 0 0 0 0]
Etiquetas de entrenamiento (one-hot):
[[1. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [1. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [1. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [1. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [1. 0. 0. 0. 0. 0. 0. 0. 0. 0.]]
Índice a clase: {0: 'Tomato___Bacterial_spot', 1: 'Tomato___Early_blight', 2: 'Tomato___Late_blight', 3: 'Tomato___Leaf_Mold',
```

## 4. MODELO

Finalizado el preprocesado de los datos se procedió con la estructuración y entrenamiento del modelo. El modelo planteado es una red neuronal convolucional (CNN) diseñada para la clasificación de imágenes en 10 categorías. Comienza con una entrada de imágenes de tamaño 150x150 píxeles con tres canales de color (RGB). Luego, emplea tres bloques secuenciales de capas convolucionales (con 32, 64 y 128 filtros, respectivamente, y un tamaño de *kernel* de 3x3) seguidas de capas de *MaxPooling*, que reducen las dimensiones espaciales para extraer características importantes. Después se aplanan las características extraídas con una capa *Flatten* y se incluye en una capa

completamente conectada (512 neuronas) con activación *ReLU* para aprender patrones complejos, seguida de una capa de salida con 10 neuronas y activación *softmax* para generar probabilidades por clase. Además, se utilizó un *Dropout* del 50% en la capa densa para prevenir el sobreajuste. El modelo se compiladó con el optimizador Adam, pérdida de entropía cruzada categórica y métrica de precisión.

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout
# Crear el modelo
model = Sequential()

model.add(Conv2D(32, (3, 3), activation='relu', input_shape=(150, 150, 3)))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Conv2D(64, (3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Conv2D(128, (3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Flatten())
model.add(Dense(512, activation='relu'))  # Más neuronas
model.add(Dropout(0.5))  # Aumentar el Dropout
model.add(Dense(10, activation='softmax'))  # Mantener el número de clases

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

### 4.1. Resumen del modelo

El modelo implementado cuanta con un total de 19,038,794 parámetros, de los cuales 19,038,794 son para el entrenamiento, mientras que 0 parámetro no entrenables.

| Layer (type) | Output Shape | Param # |
|---|---|---|
| conv2d (Conv2D) | (None, 148, 148, 32) | 896 |
| max_pooling2d (MaxPooling2D) | (None, 74, 74, 32) | 0 |
| conv2d_1 (Conv2D) | (None, 72, 72, 64) | 18,496 |
| max_pooling2d_1 (MaxPooling2D) | (None, 36, 36, 64) | 0 |
| conv2d_2 (Conv2D) | (None, 34, 34, 128) | 73,856 |
| max_pooling2d_2 (MaxPooling2D) | (None, 17, 17, 128) | 0 |
| flatten (Flatten) | (None, 36992) | 0 |
| dense (Dense) | (None, 512) | 18,940,416 |
| dropout (Dropout) | (None, 512) | 0 |
| dense_1 (Dense) | (None, 10) | 5,130 |

**Total params:** 19,038,794 (72.63 MB)
**Trainable params:** 19,038,794 (72.63 MB)
**Non-trainable params:** 0 (0.00 B)

## 5. RESULTADOS

Después de ejecutado el modelo se obtuvo mejoras significativas tanto en la precisión de entrenamiento como en la de validación. Comenzando con una precisión del 23 % en la primera época, el modelo alcanzó una precisión de validación del 80 % al final del entrenamiento. Los valores de pérdida disminuyeron en general, lo que indica un aprendizaje efectivo, aunque algunas fluctuaciones en la precisión de validación sugieren un sobreajuste ocasional.

```
Epoch 1/50
/usr/local/lib/python3.10/dist-packages/keras/src/trainers/data_adapters/py_dataset_adapter.py:122: UserWarning: Your `PyD
  self._warn_if_super_not_called()
313/313 ━━━━━━━━━━━━━━━━━━━━ 81s 233ms/step - accuracy: 0.2317 - loss: 2.1180 - val_accuracy: 0.6410 - val_loss: 1.0282
Epoch 2/50
313/313 ━━━━━━━━━━━━━━━━━━━━ 66s 206ms/step - accuracy: 0.6230 - loss: 1.0815 - val_accuracy: 0.4220 - val_loss: 2.4436
Epoch 3/50
313/313 ━━━━━━━━━━━━━━━━━━━━ 66s 207ms/step - accuracy: 0.7145 - loss: 0.8048 - val_accuracy: 0.6390 - val_loss: 1.3219
Epoch 4/50
313/313 ━━━━━━━━━━━━━━━━━━━━ 81s 201ms/step - accuracy: 0.7636 - loss: 0.6718 - val_accuracy: 0.6850 - val_loss: 1.0939
Epoch 5/50
313/313 ━━━━━━━━━━━━━━━━━━━━ 81s 201ms/step - accuracy: 0.8001 - loss: 0.5684 - val_accuracy: 0.7370 - val_loss: 0.9696
Epoch 6/50
313/313 ━━━━━━━━━━━━━━━━━━━━ 64s 202ms/step - accuracy: 0.8156 - loss: 0.5161 - val_accuracy: 0.7520 - val_loss: 0.9631
Epoch 7/50
313/313 ━━━━━━━━━━━━━━━━━━━━ 66s 208ms/step - accuracy: 0.8338 - loss: 0.4823 - val_accuracy: 0.6990 - val_loss: 1.1353
Epoch 8/50
312/313 ━━━━━━━━━━━━━━━━━━━━ 0s 198ms/step - accuracy: 0.8359 - loss: 0.4661
Reached 85.0% val_accuracy so cancelling training!
313/313 ━━━━━━━━━━━━━━━━━━━━ 64s 201ms/step - accuracy: 0.8360 - loss: 0.4660 - val_accuracy: 0.8000 - val_loss: 0.6851
```

El modelo alcanzó una precisión de validación del 80%, lo que indica que es capaz de clasificar correctamente 8 de cada 10 muestras del conjunto de validación, un resultado prometedor para el problema en cuestión. Además, la pérdida de validación, con un valor de 0.685, sugiere que el modelo está minimizando eficazmente el error en sus predicciones.

```
32/32 ━━━━━━━━━━━━━━━━━━━━ 1s 33ms/step - accuracy: 0.8013 - loss: 0.6997
Pérdida en validación: 0.6851435303688049
Precisión en validación: 0.8000000011920929
```

En la Figura 2, se muestran los gráficos de pérdida y presión del entrenamiento y validación. En la Figura 2(a) se aprecia que la pérdida de entrenamiento disminuye de manera constante, lo que indica que el modelo está aprendiendo y ajustando bien los parámetros. Sin embargo, la pérdida de validación presenta un pico notable en la segunda época, lo que podría sugerir un sobreajuste temporal. Después de este pico, la pérdida de validación también disminuye, pero se estabiliza en un valor relativamente alto comparado con la pérdida de entrenamiento. Poe su parte, en la Figura 2(b), La precisión de entrenamiento aumenta de manera consistente, alcanzando valores cercanos al 80%. La precisión de validación, por su parte, también mejora, pero con más fluctuaciones, lo que sugiere que el modelo podría estar experimentando ciertos picos de sobreajuste en algunas épocas (como en la segunda época, donde la precisión de validación baja brevemente). No obstante, la tendencia general es positiva, y la precisión de validación finalmente se estabiliza cerca del 80%, lo que es un buen indicador de la capacidad de generalización del modelo.

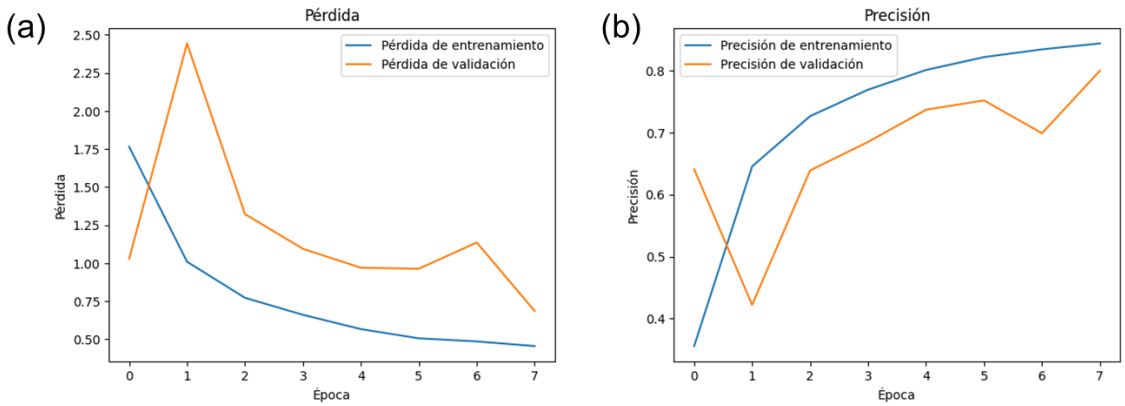


Figura 2: Gráficas de (a) pérdida y (b) precisión de entrenamiento y validación.

En conclusión, el modelo desarrollado para la clasificación de hojas de tomate a partir de imágenes de área foliar con diversas afectaciones de enfermedades muestra un rendimiento prometedor. A lo largo de las épocas de entrenamiento, se observó una mejora continua en la precisión, alcanzando niveles cercanos al 80%, lo que indica que el modelo ha aprendido a identificar patrones relevantes en las imágenes de manera efectiva. Sin embargo, las fluctuaciones en la precisión de validación sugieren que el modelo experimentó un leve sobreajuste en las primeras épocas, lo cual es normal en modelos de clasificación.

## 6. REFERENCIAS


[1] Falaschetti, L., Manoni, L., Di Leo, D., 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

[2] 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

[3] National Plant Data Center, NRCS, USDA. (2023). *The scientific name of Tomato*. The plants database (version 5.1.1). Baton Rouge, LA 70874-4490 USA. http://plants.usda.gov/Reference (Accessed 13 November 2023). For: *Solanum lycopersicum var. lycopersicum*.

[4] National Plant Data Center, NRCS, USDA. (2023). *The scientific name of Potato*. The plants database (version 5.1.1). Baton Rouge, LA 70874-4490 USA. http://plants.usda.gov/Reference (Accessed 13 November 2023). For: *Solanum tuberosum*.

[5] Ashok, S., Kishore, G., Rajesh, V., Suchitra, S., Sophia, S. G. G., & Pavithra, B. (2020). Tomato leaf disease detection using deep learning techniques. *2020 5th International Conference on Communication and Electronics Systems (ICCES)*, 979-983. https://doi.org/10.1109/ICCES48766.2020.9137986.

[6] Hossain, M. I., Jahan, S., Al Asif, M. R., Samsuddoha, M., & Ahmed, K. (2023). Detecting tomato leaf diseases by image processing through deep convolutional neural networks. *Smart Agricultural Technology, 5*, 100301. https://doi.org/10.1016/j.atech.2023.100301.