Detección de enfermedades en cultivos de tomate

Integrantes:

Joseph Fabián Basto Cuadros

Juan José Bayona Sepúlveda

Problema

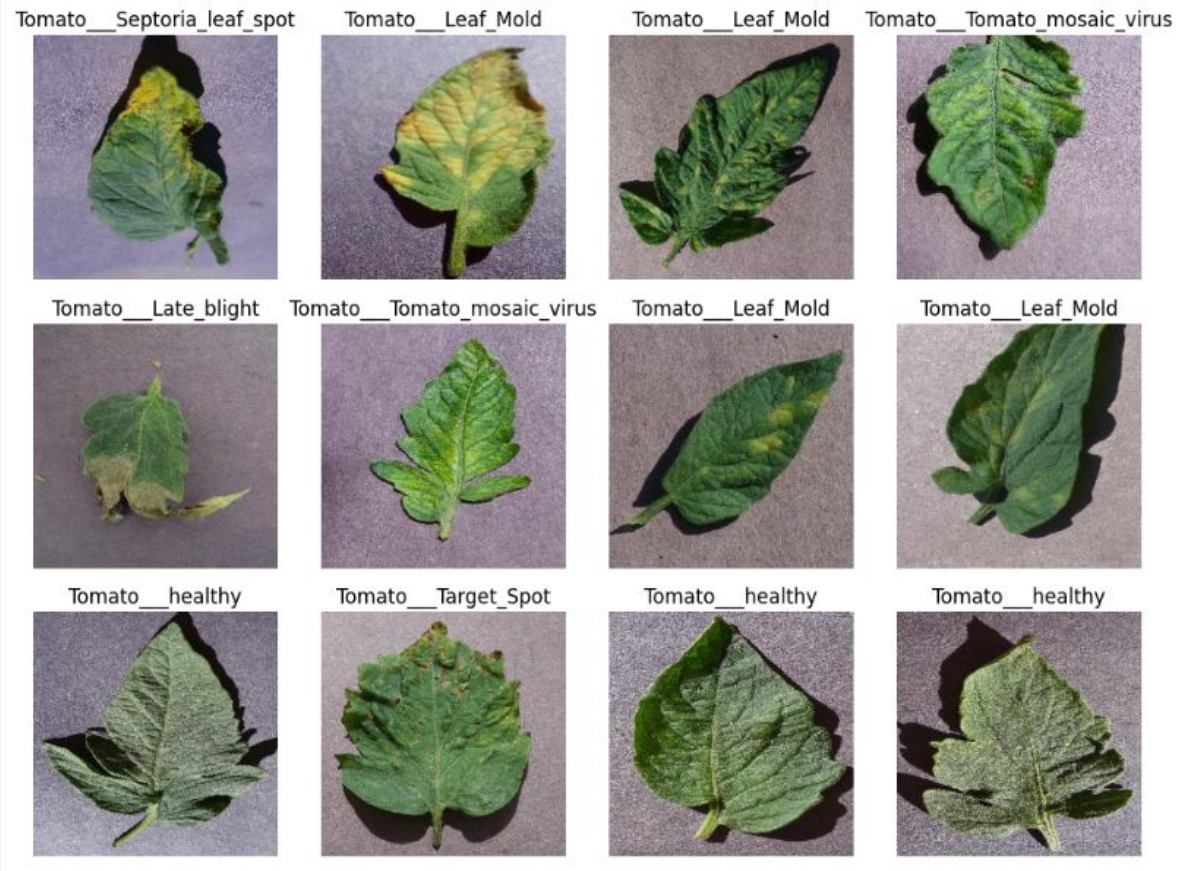
La producción de frutas, verduras, granos y otros son esenciales para el sostenimiento de la humanidad, pero la calidad de los alimentos que consumimos puede verse afectada por un mal manejo de los productos o por enfermedades en las plantas que los producen, por lo tanto, la detección de estas enfermedades es importante ya que nos permite tratar de manera adecuada a las plantas para asegurar la continua producción de alimentos y la salubridad en general. El objetivo es apoyar a los agricultores en garantizar la salud de los cultivos y detectar anomalías en estos.

Dataset

El dataset de PlantVillage es un conjunto de imágenes de hojas de plantas enfermas, recopiladas por David Hughes y su equipo de la Universidad Estatal de Pensilvania. El dataset contiene más de 54,000 imágenes de hojas de plantas enfermas, que representan 38 categorías de enfermedades de plantas diferentes, incluyendo manchas foliares, moho, tizón, marchitez y otras enfermedades. Las imágenes fueron tomadas en condiciones de iluminación y fondo variables, lo que las hace más desafiantes para la clasificación automática.

Del dataset se tomaron 10 clases para las hojas de tomate (1 de hojas sanas y 9 con enfermedades distintas) cada una con 1000 imágenes.

Dataset



Fuente: https://www.tensorflow.org/datasets/catalog/plant_village?hl=es-419

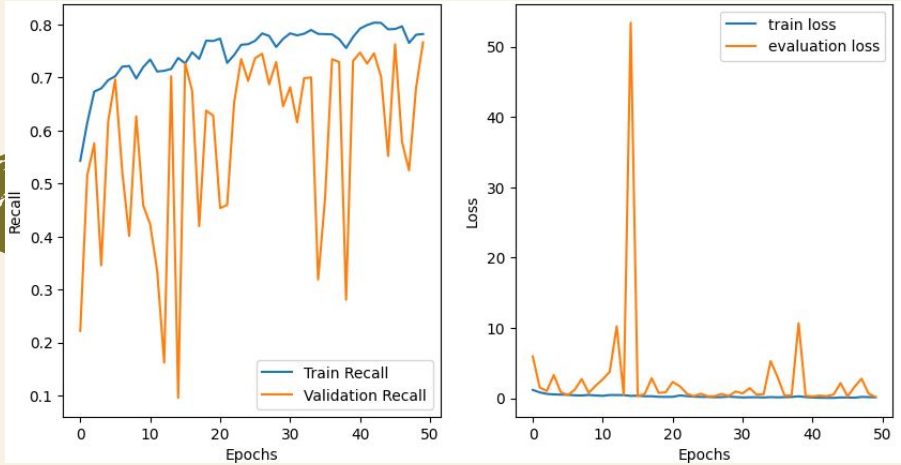
Tratamiento de datos y características generales

- Las imágenes contaban con una resolución de 256x256 pero fueron bajadas a 128x128.
- Se realizó una partición de datos de 70% para entrenamiento y 30% para el testeo.
- Se experimentó con tres redes distintas: una red convolucional proveniente del estado del arte, una red convolucional propia y un transfer learning a partir de una ResNet.
- Se optó por usar la métrica “recall” para cada una de las redes propuestas.
- Por último, se experimentó con la métrica “precision”.

Estado del arte

```
model1 = tf.keras.Sequential([
    tf.keras.layers.Conv2D(32, (3, 3), activation='relu',padding="same", input_shape=(Tam_image, Tam_image, 3)),
    tf.keras.layers.BatchNormalization(),
    tf.keras.layers.MaxPooling2D(3, 3),
    tf.keras.layers.Dropout(0.25),

    tf.keras.layers.Conv2D(64, (3, 3), activation='relu',padding="same"),
    tf.keras.layers.BatchNormalization(),

    tf.keras.layers.Conv2D(64, (3, 3), activation='relu',padding="same"),
    tf.keras.layers.BatchNormalization(),
    tf.keras.layers.MaxPooling2D(2, 2),
    tf.keras.layers.Dropout(0.25),

    tf.keras.layers.Conv2D(128, (3, 3), activation='relu',padding="same"),
    tf.keras.layers.BatchNormalization(),

    tf.keras.layers.Conv2D(128, (3, 3), activation='relu',padding="same"),
    tf.keras.layers.BatchNormalization(),
    tf.keras.layers.MaxPooling2D(2, 2),
    tf.keras.layers.Dropout(0.25),

    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(1024, activation='relu'),
    tf.keras.layers.BatchNormalization(),
    tf.keras.layers.Dropout(0.5),
    tf.keras.layers.Dense(len(categorias), activation='softmax')
])

model1.summary()
```

Resultados

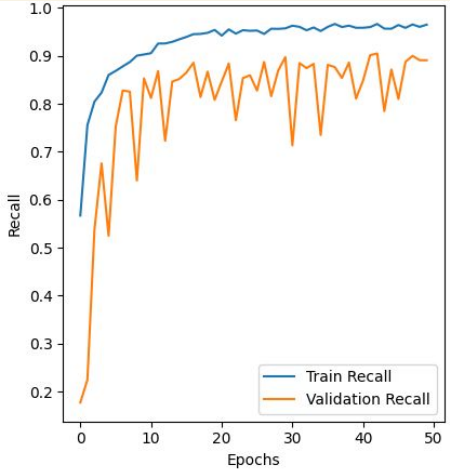


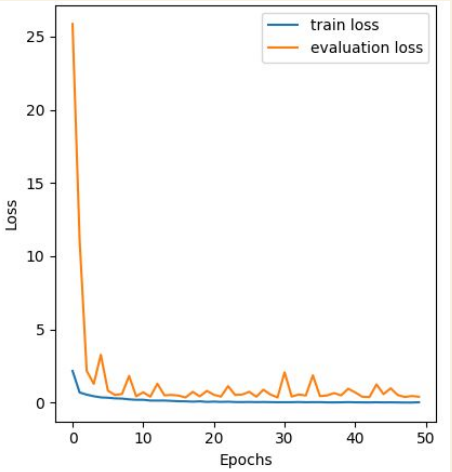

```
Matriz de Confusión:
[[281  10   1   0   6   0   3   2   0   0]
 [  1 281  11   2   2   0   8   3   1   0]
 [  0  22 263   6   6   0   2   0   0   0]
 [  0   2   0 297   2   2   0   0   2   1]
 [  0   3   2   9 303   0   2   0   1   0]
 [  0   0   0   1   0 267  15   0   0   0]
 [  0   0   1   1   2  16 272   0   3   1]
 [  3   1   0   0   1   3   0 297   0   0]
 [  0   1   0   0   2   0   0   0 280   0]
 [  0   0   0   0   0   0   1   0   0 295]]
Precision: [0.98596491 0.878125   0.94604317 0.93987342 0.93518519 0.92708333
 0.89768977 0.98344371 0.97560976 0.99326599]
Recall: [0.92739274 0.90938511 0.87959866 0.97058824 0.946875   0.9434629
 0.91891892 0.97377049 0.98939929 0.99662162]
MeanRecall: 0.945601297320055
```

Modelo Planteado

```
model = tf.keras.Sequential([
    tf.keras.layers.Conv2D(filters=32, kernel_size=(3,3), activation='elu', kernel_initializer='glorot_uniform', bias_initializer='he_uniform', padding='same', input_shape=(Tam_image, Tam_image, 3)),
    tf.keras.layers.BatchNormalization(),
    tf.keras.layers.Conv2D(filters=32, kernel_size=(3,3), activation='elu', kernel_initializer='glorot_uniform', bias_initializer='he_uniform', padding='same'),
    tf.keras.layers.BatchNormalization(),
    tf.keras.layers.MaxPooling2D(2),
    tf.keras.layers.Dropout(0.2),
    tf.keras.layers.Conv2D(filters=64, kernel_size=(3,3), activation='elu', kernel_initializer='glorot_uniform', bias_initializer='he_uniform', padding='same'),
    tf.keras.layers.BatchNormalization(),
    tf.keras.layers.Conv2D(filters=64, kernel_size=(3,3), activation='elu', kernel_initializer='glorot_uniform', bias_initializer='he_uniform', padding='same'),
    tf.keras.layers.BatchNormalization(),
    tf.keras.layers.MaxPooling2D(2),
    tf.keras.layers.Dropout(0.2),
    tf.keras.layers.Conv2D(filters=128, kernel_size=(3,3), activation='elu', kernel_initializer='glorot_uniform', bias_initializer='he_uniform', padding='same'),
    tf.keras.layers.BatchNormalization(),
    tf.keras.layers.Dropout(0.2),
    tf.keras.layers.Conv2D(filters=128, kernel_size=(3,3), activation='elu', kernel_initializer='glorot_uniform', bias_initializer='he_uniform', padding='same'),
    tf.keras.layers.BatchNormalization(),
    tf.keras.layers.MaxPooling2D(2),
    tf.keras.layers.Dropout(0.2),
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(128, activation='relu'),
    tf.keras.layers.Dense(len(categorias), activation='softmax')
])

model.summary()
```

Resultados

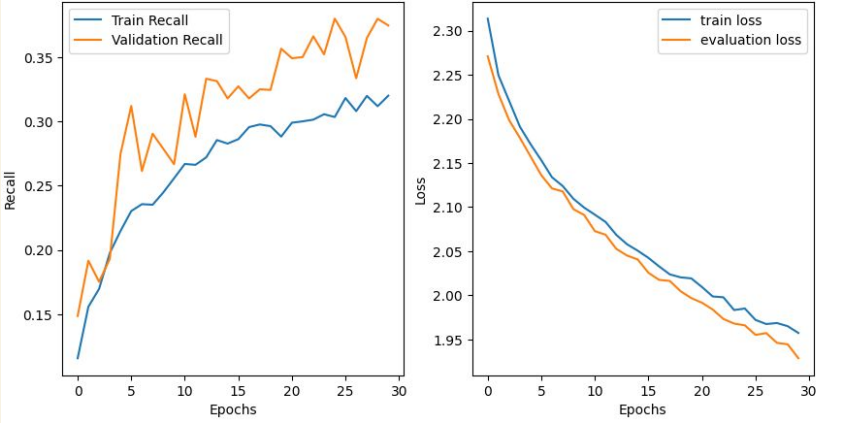




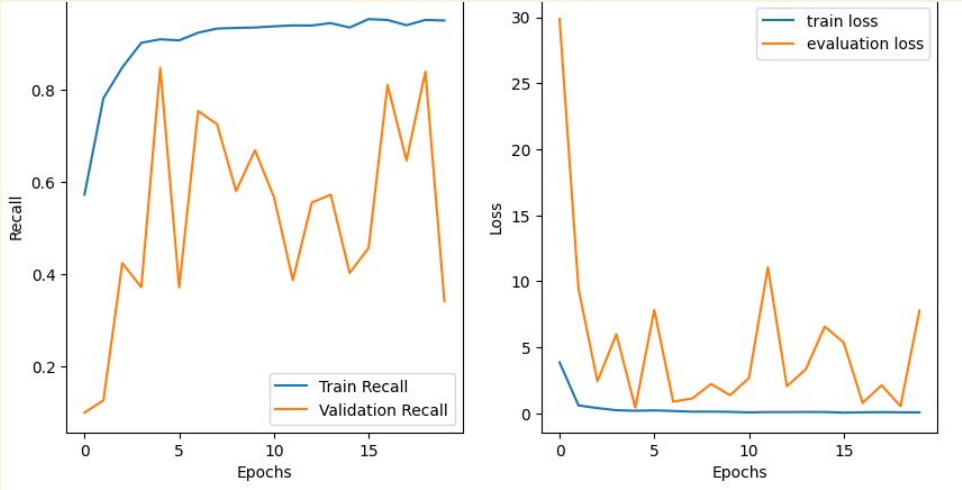

```
Matriz de Confusión:
[[286   4   1   1   0   1   1   9   0   0]
 [  8 260  12   4   9   1  12   3   0   0]
 [  0  17 249   8  17   4   1   3   0   0]
 [  0   3   3 270  19   5   2   1   1   2]
 [  1   2   1   2 313   0   0   0   1   0]
 [  0   3   1   0   2 260  17   0   0   0]
 [  0   5   1   1   4  19 262   1   2   1]
 [  7   2   0   0   1   6   0 289   0   0]
 [  0   1   0   1   5   1   4   0 271   0]
 [  0   0   1   0   0   0   1   0   0 294]]
Precision: [0.94701987 0.87542088 0.92565056 0.94076655 0.84594595 0.87542088
 0.87333333 0.94444444 0.98545455 0.98989899]
Recall: [0.94389439 0.84142395 0.83277592 0.88235294 0.978125   0.91872792
 0.88513514 0.94754098 0.95759717 0.99324324]
MeanRecall: 0.918081664889208
```

Transfer Learning

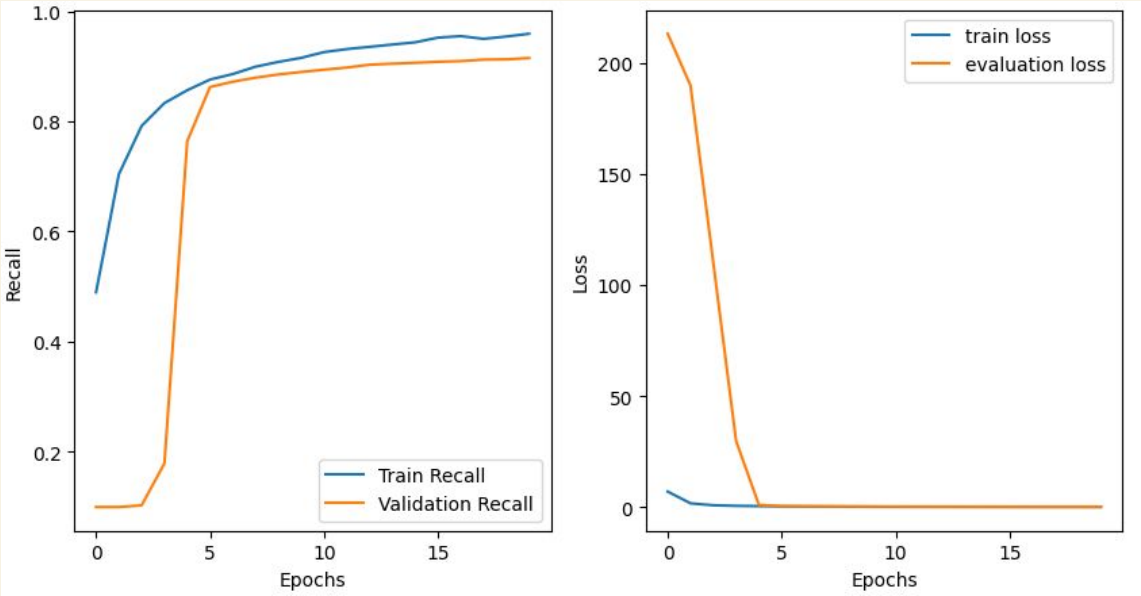
```
model_A =  tf.keras.applications.ResNet50(input_shape=x_train[0].shape, weights='imagenet', include_top=False)
model_A.trainable = False
model_A.summary()

prediction_layer = tf.keras.layers.Dense(len(categorias), activation='softmax')
flatten_layer = tf.keras.layers.Flatten()
dropout_layer = tf.keras.layers.Dropout(0.3)
global_average_layer = tf.keras.layers.GlobalAveragePooling2D()

model_B_on_A = tf.keras.Sequential([
  model_A,
  global_average_layer,
  dropout_layer,
  prediction_layer
])

learning_rate= 0.001
opt = tf.keras.optimizers.Adam(learning_rate=learning_rate)
model_B_on_A.compile(optimizer=opt, loss='categorical_crossentropy',metrics=[mrecall_metric.mean_recall])
history = model_B_on_A.fit(x_train, y_train, epochs=30,verbose=1, batch_size=32,validation_data=(x_test, y_test))
```



```
model_A.trainable = True
print("Total layers of ResNet50:", len(model_A.layers))
for layer in model_A.layers[:int(len(model_A.layers) * 0.25)]:
    layer.trainable = False

print("Total trainable layers:", sum([1 for layer in model_A.layers if layer.trainable]))

model_B_on_A.summary()
len(model_B_on_A.trainable_variables)

opt = tf.keras.optimizers.Adam(learning_rate=learning_rate / 10)
model_B_on_A.compile(optimizer=opt, loss='categorical_crossentropy', metrics=[mrecall_metric.mean_recall])

# Entrenamiento descongelando el 75% restante
history = model_B_on_A.fit(x_train, y_train, epochs=20, verbose=1, batch_size=32, validation_data=(x_test, y_test))
```



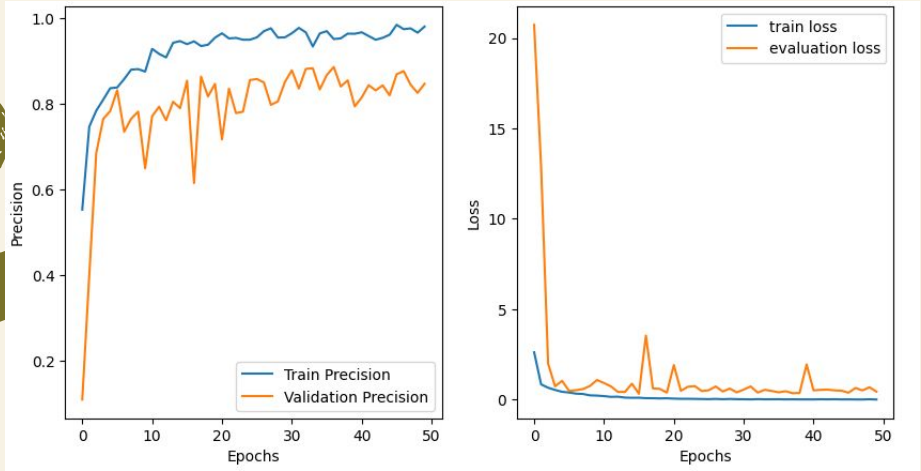
```
model_A.trainable = True
print("Total layers of ResNet50:", len(model_A.layers))
model_B_on_A.summary()
len(model_B_on_A.trainable_variables)

opt = tf.keras.optimizers.Adam(learning_rate=learning_rate / 500)
model_B_on_A.compile(optimizer=opt, loss='categorical_crossentropy', metrics=[mrecall_metric.mean_recall])

# Entrenamiento con todos los parámetros descongelados
history = model_B_on_A.fit(x_train, y_train, epochs=20, verbose=1, batch_size=32, validation_data=(x_test, y_test))
```



Resultados

```
Matriz de Confusión:
[[278  10   3   1   5   0   3   3   0   0]
 [  4 272  14   3   4   5   7   0   0   0]
 [  1  15 277   3   1   0   2   0   0   0]
 [  1   2   2 295   3   0   1   2   0   0]
 [  5   3   2   3 300   1   5   0   1   0]
 [  0   3   0   0   0 272   8   0   0   0]
 [  1   9   0   3   2  17 260   0   0   4]
 [  5   0   0   1   0   1   0 297   1   0]
 [  0   0   0   0   2   0   0   0 281   0]
 [  0   0   0   0   0   1   3   0   0 292]]
Precision: [0.94237288 0.86624204 0.9295302  0.95469256 0.94637224 0.91582492
 0.89965398 0.98344371 0.99293286 0.98648649]
Recall: [0.91749175 0.8802589  0.9264214  0.96405229 0.9375     0.96113074
 0.87837838 0.97377049 0.99293286 0.98648649]
MeanRecall: 0.9418423302023692
```

Resultados usando la métrica “precision”




```
94/94 [==============================] - 1s 7ms/step
Matriz de Confusión:
[[262   1   1   0   4   0   0   3   0   0]
 [  9 238  15   1   9   6  20   4   0   0]
 [  3  11 258   1  12   1   5   3   1   0]
 [  1   3   6 266  16   3   4   1   3   1]
 [  1   6   2   2 272   0   6   0   4   0]
 [  0   2   0   1   3 268  22   0   1   0]
 [  0   6   1   0   6   4 288   0   3   0]
 [  0   0   0   0   1   4   0 311   0   0]
 [  0   0   0   0   1   0   0   0 288   0]
 [  0   0   0   0   1   2   5   0   0 317]]
Precision: [0.94927536 0.89138577 0.91166078 0.98154982 0.83692308 0.93055556
 0.82285714 0.96583851 0.96       0.99685535]
Recall: [0.96678967 0.78807947 0.87457627 0.875      0.92832765 0.9023569
 0.93506494 0.98417722 0.99653979 0.97538462]
Precision: 0.911660777385159
```