DETECTION OF PLANT LEAF DISEASE

1932055 - Vandana A

ABSTRACT:

Crop diseases are a huge danger to food security, but due to a lack of infrastructure in many regions of the world, timely detection is challenging. Agriculture has a significant part in a developing country like India. Agricultural intervention in rural India's livelihood accounts for around 58 per cent of the total. Thus, minimizing severe losses in tomato quantity and output is largely contingent on recognising and classifying diseases that a plant may have. Image processing, a cutting-edge technology, is utilized to address such challenges utilizing a variety of approaches and algorithms.

When a plant becomes infected with a certain disease, the leaves of the plant are first impacted. The type of sickness is discovered in this project through five stages. Pre-processing, leaf segmentation, feature extraction, classification and predictions are the five processes. Pre-processing is utilized to eliminate noise, and image segmentation is employed to separate the affected or damaged areas of the leaf. The k-nearest neighbors (KNN) approach is used to solve problems relating to classification and regression. It is a directed, supervised, and advanced machine learning algorithm. The treatment is recommended to the user during the terminal stage. Diseases wreak havoc on living plants in particular. This project will present a representation of leaf disease identification using image processing to identify flaws in plants from photos using color, binding, and texture to provide farmers with quick and consistent findings.

WORKING:

The dataset is taken from kaggle . The link is given below
https://www.kaggle.com/emmarex/plantdisease

The main idea of this project is to help the farmers to identify the type of disease in tomato plants by creating a model using CNN that can predict the type of disease in tomato plants. This dataset has 10 labels and a total of 16011 images. I have splitted the dataset into train , test and validation to feed into the model. Training - 80% of the data. Testing - 10% of the data. Validation - 10% of the data. We have rescaled the data and augmented the data and converted to greyscale to get a good accuracy of the model.

CODE:

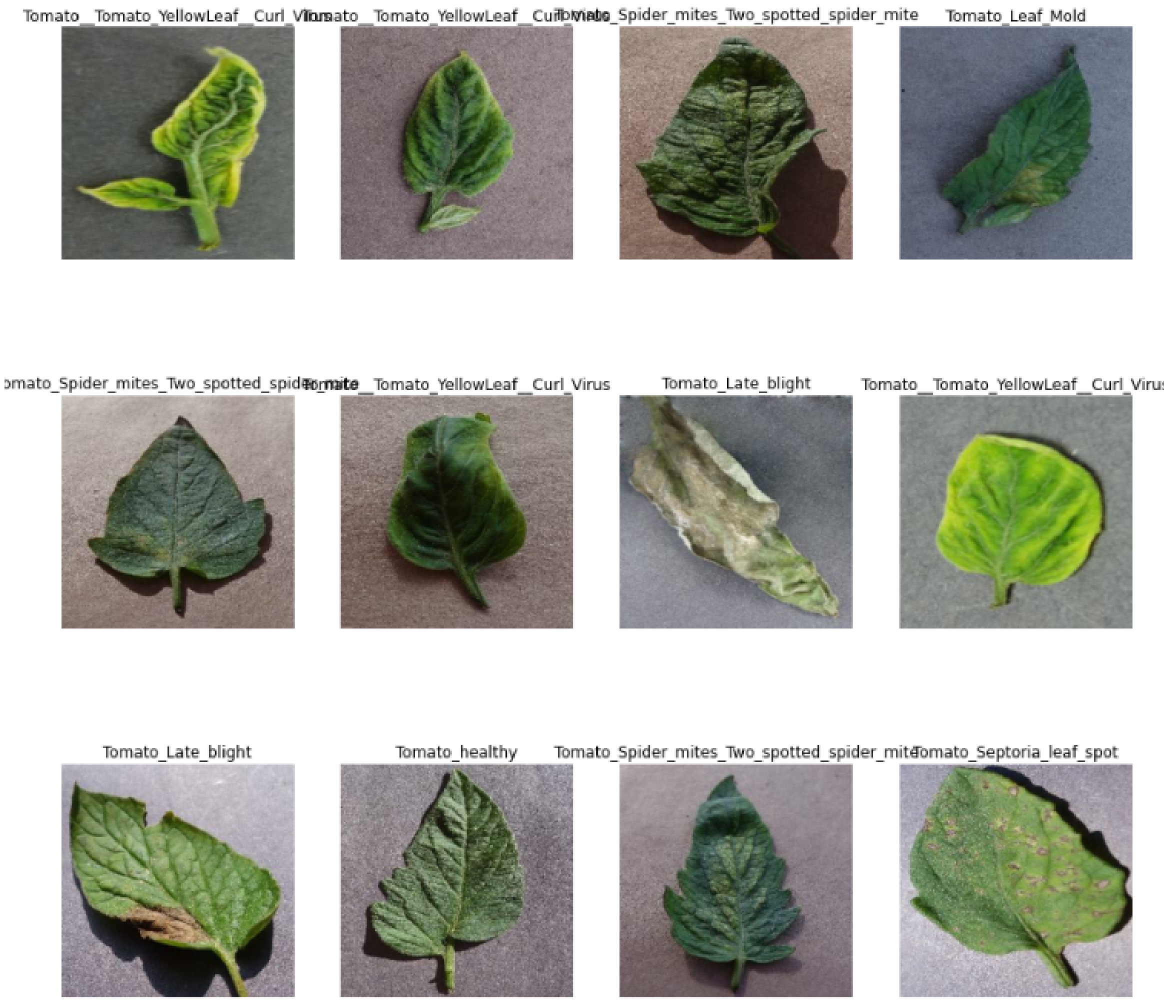
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import tensorflow as tf
from tensorflow.keras import layers,models
%matplotlib inline
IMAGE_SIZE = 256
BATCH_SIZE = 32
EPOCHS = 50
dataset = tf.keras.preprocessing.image_dataset_from_directory(
    "PlantVillage",
    shuffle = True,
    image_size = (IMAGE_SIZE,IMAGE_SIZE),
    batch_size = BATCH_SIZE )
class_names = dataset.class_names
class_names
len(dataset)plt.figure(figsize = (15,15))
for image_batch,label_batch in dataset.take(1):
    for i in range(12):
        a = plt.subplot(3,4,i+1)
        plt.imshow(image_batch[i].numpy().astype('uint8'))
        plt.title(class_names[label_batch[i]])
        plt.axis('off')
def get_dataset_partitions_tf(ds,train_split = 0.8,test_split =
0.1,val_split = 0.1,shuffle = True,shuffle_size = 10000):
    ds_size = len(ds)
    if shuffle:
        ds = ds.shuffle(shuffle_size,seed = 12)
    train_size = int(train_split * ds_size)
    val_size = int(val_split * ds_size)

    train_ds = ds.take(train_size)
    val_ds = ds.skip(train_size).take(val_size)
    test_ds = ds.skip(train_size).skip(val_size)
    return train_ds,test_ds,val_ds
```

```
train_ds , test_ds, val_ds = get_dataset_partitions_tf(dataset)
len(train_ds)
len(train_ds)
len(val_ds)
train_ds = train_ds.cache().shuffle(1000).prefetch(buffer_size =
tf.data.experimental.AUTOTUNE)
test_ds = train_ds.cache().shuffle(1000).prefetch(buffer_size =
tf.data.experimental.AUTOTUNE)
val_ds = train_ds.cache().shuffle(1000).prefetch(buffer_size =
tf.data.experimental.AUTOTUNE)
resize_and_rescale = tf.keras.Sequential([
        layers.experimental.preprocessing.Resizing(256,256),
        layers.experimental.preprocessing.Rescaling(1.0/255)
    ])
data_augmentation = tf.keras.Sequential([

layers.experimental.preprocessing.RandomFlip("horizontal_and_ver
tical"),
    layers.experimental.preprocessing.RandomRotation(0.2)
])
input_shape = (32,256,256,3)
n_classes = len(class_names)
model = models.Sequential([
    resize_and_rescale,
    data_augmentation,
    layers.Conv2D(32,(3,3),activation = 'relu',input_shape =
input_shape),
    layers.MaxPooling2D((2,2)),
    layers.Conv2D(64,(3,3),activation = 'relu'),
    layers.MaxPooling2D((2,2)),
    layers.Conv2D(64,(3,3),activation = 'relu'),
    layers.MaxPooling2D((2,2)),
    layers.Conv2D(128,(3,3),activation = 'relu'),
    layers.MaxPooling2D((2,2)),
    layers.Flatten(),
    layers.Dense(256,activation = 'relu'),
    layers.Dense(n_classes,activation = 'softmax'),
])
model.build(input_shape = input_shape)
model.summary()
```

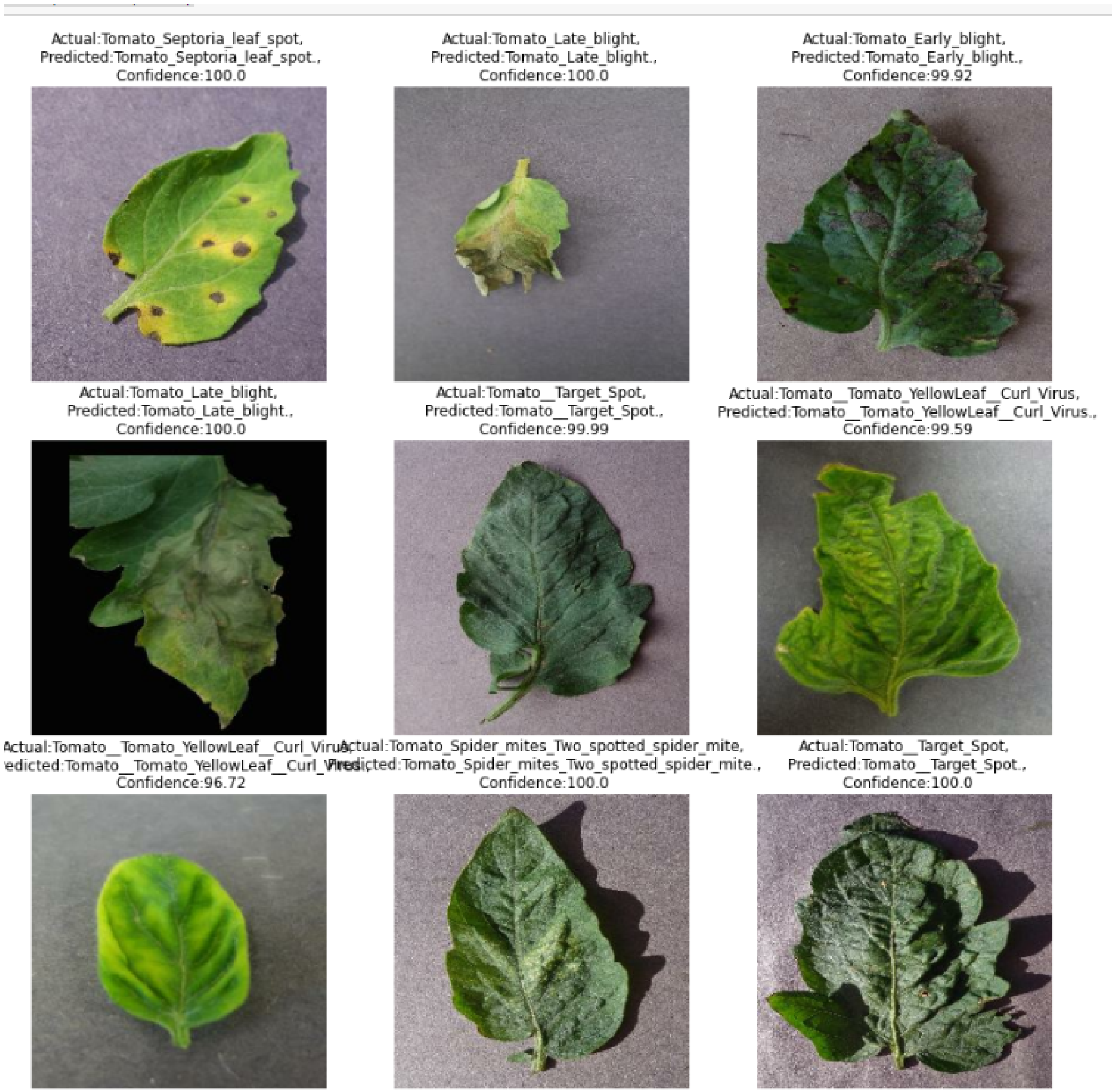
```
model.compile(
    optimizer = 'adam',
    loss =
tf.keras.losses.SparseCategoricalCrossentropy(from_logits =
False),
    metrics = ['accuracy']
)
history = model.fit(
    train_ds,
    epochs = EPOCHS,
    batch_size = BATCH_SIZE,
    verbose = 1,
    validation_data = val_ds)
scores = model.evaluate(test_ds)
Scores
History.params
history.history.keys()
acc = history.history['accuracy']
val_acc = history.history['val_accuracy']

loss = history.history['loss']
val_loss = history.history['val_loss']
plt.figure(figsize = (8,8))
plt.subplot(1,2,1)
plt.plot(range(EPOCHS),acc,label='Training accuracy')
plt.plot(range(EPOCHS),val_acc,label='Validation accuracy')
plt.legend(loc = 'lower right')
plt.title('Training and Validation Accuracy')


plt.subplot(1,2,2)
plt.plot(range(EPOCHS),loss,label='Training loss')
plt.plot(range(EPOCHS),val_loss,label='Validation loss')
plt.legend(loc = 'upper right')
plt.title('Training and Validation Loss')

import numpy as np
for image_batch,label_batch in test_ds.take(1):
    first_image = image_batch[0].numpy().astype('uint8')
    first_label = label_batch[0].numpy()
```

```
    print("First Image to Predict:")
    plt.imshow(first_image)
    plt.axis("off")
    print("Actual Label:",class_names[first_label])

    batch_prediction = model.predict(image_batch)
    print("Predicted
Label:",class_names[np.argmax(batch_prediction[0])])
def predict(model,img):
    img_array =
tf.keras.preprocessing.image.img_to_array(images[i].numpy())
    img_array = tf.expand_dims(img_array,0)

    predictions = model.predict(img_array)

    predicted_class = class_names[np.argmax(predictions[0])]
    confidence = round(100*(np.max(predictions[0])),2)
    return predicted_class,confidence
plt.figure(figsize = (15,15))
for images,labels in test_ds.take(1):
    for i in range(9):
        ax = plt.subplot(3,3,i+1)
        plt.imshow(images[i].numpy().astype('uint8'))

        predicted_class,confidence =
predict(model,images[i].numpy())
        actual_class = class_names[labels[i]]

        plt.title(f"Actual:{actual_class},\n
Predicted:{predicted_class}.,\n Confidence:{confidence}")

        plt.axis("off")
```

OUTPUT:




```
First Image to Predict:
Actual Label: Tomato_healthy
Predicted Label: Tomato_healthy
```


Actual:Tomato_Septoria_leaf_spot,
Predicted:Tomato_Septoria_leaf_spot.,
Confidence:100.0
Actual:Tomato_Late_blight,
Predicted:Tomato_Late_blight.,
Confidence:100.0
Actual:Tomato_Early_blight,
Predicted:Tomato_Early_blight.,
Confidence:99.92
Actual:Tomato_Late_blight,
Predicted:Tomato_Late_blight.,
Confidence:100.0
Actual:Tomato__Target_Spot,
Predicted:Tomato__Target_Spot.,
Confidence:99.99
Actual:Tomato__Tomato_YellowLeaf__Curl_Virus,
Predicted:Tomato__Tomato_YellowLeaf__Curl_Virus.,
Confidence:99.59
ctual:Tomato__Tomato_YellowLeaf__Curl_Virus,
edicted:Tomato__Tomato_YellowLeaf__Curl_Vi
Confidence:96.72
Actual:Tomato_Spider_mites_Two_spotted_spider_mite,
dicted:Tomato_Spider_mites_Two_spotted_spider_mite.,
Confidence:100.0
Actual:Tomato__Target_Spot,
Predicted:Tomato__Target_Spot.,
Confidence:100.0