MANGO DISEASE DETECTION USING GAN

ABSTRACT

Mango disease prediction is crucial for ensuring healthy yields and maintaining the quality of mango produce. Traditional methods of disease detection in mangoes often rely on manual inspection, which is time-consuming and prone to human error. To address these challenges, this project proposes a novel approach using Generative Adversarial Networks (GANs) to enhance detection accuracy. GANs generate high-quality synthetic images of diseased mangoes, improving robustness and accuracy. The system includes a generator that creates synthetic images and a discriminator that distinguishes between real and synthetic images, augmenting the dataset with diverse disease manifestations. A Convolutional Neural Network (CNN)-based classifier is trained on this enhanced dataset to identify diseases such as anthracnose, powdery mildew, and bacterial black spot with high precision. Experimental results demonstrate a significant reduction in false positives and negatives, showcasing the potential of GANs in improving agricultural disease detection. This approach not only enhances the quality and yield of mango production but also offers a scalable model for various crops and diseases, highlighting the revolutionary potential of GANs in agricultural applications.

Data Set : https://www.kaggle.com/datasets/warcoder/mangofruitdds?resource=download

MANGO DISEASE DETECTION USING CNN

MODULE 1 : IMPORTING LIBRARIES AND EXTRACTING LIBRARIES

```
import os
import sys
from tempfile import NamedTemporaryFile
from urllib.request import urlopen
from urllib.parse import unquote, urlparse
from urllib.error import HTTPError
from zipfile import ZipFile
import tarfile
import shutil

CHUNK_SIZE = 40960
DATA_SOURCE_MAPPING = 'mangofruitdds:https%3A%2F%2Fstorage.googleapis.com%2Fkaggle-
data-sets%2F3723789%2F6450350%2Fbundle%2Farchive.zip%3FX-Goog-Algorithm%3DGOOG4-
RSA-SHA256%26X-Goog-Credential%3Dgcp-kaggle-com%2540kaggle-
161607.iam.gserviceaccount.com%252F20240728%252Fauto%252Fstorage%252Fgoog4_request%
26X-Goog-Date%3D20240728T041500Z%26X-Goog-Expires%3D259200%26X-Goog-
SignedHeaders%3Dhost%26X-Goog-
Signature%3Db6d2b3a87ad4665038f0fb681fa4dc41dd46019e17f6920e63829dc3c290ea6d712ddf3
5513cc9b8c90bb82a2b64c8ff140a1ece6d937877fab1be163be15bd9d80c62c9faa7b2c1eec67a461b
6d9c281d720212e2052658baaf3fcc53e21c0255c0513725e3388a17c145b61c8afe9c397c15e6617e2
9d6a0ade1f81a50de272d74b338f4d412a3951ecd9e692e36bf874a7f98ff3f10bcfc4216fd374f15f8
2cdf344dd30555c1c147f419ef55092536a429708b3d500b78cee733078ae1afcf2769518ebead2815b
ca2316e22fb4a683a51d733095daea61bbea4726ddeaa7dfb34925642453313a32e9f2078f9c37b8d05
c7ccafab3919d7d20ff338f730'
```

```
KAGGLE_INPUT_PATH='/kaggle/input'
KAGGLE_WORKING_PATH='/kaggle/working'
KAGGLE_SYMLINK='kaggle'

!umount /kaggle/input/ 2> /dev/null
shutil.rmtree('/kaggle/input', ignore_errors=True)
os.makedirs(KAGGLE_INPUT_PATH, 0o777, exist_ok=True)
os.makedirs(KAGGLE_WORKING_PATH, 0o777, exist_ok=True)

try:
 os.symlink(KAGGLE_INPUT_PATH,os.path.join("..",'input'),target_is_directory=True)
except FileExistsError:
  pass
try:
 os.symlink(KAGGLE_WORKING_PATH,os.path.join("..",'work'),target_is_directory=True)
except FileExistsError:
  pass


for data_source_mapping in DATA_SOURCE_MAPPING.split(','):
    directory, download_url_encoded = data_source_mapping.split(':')
    download_url = unquote(download_url_encoded)
    filename = urlparse(download_url).path
    destination_path = os.path.join(KAGGLE_INPUT_PATH, directory)
    try:
        with urlopen(download_url) as fileres, NamedTemporaryFile() as tfile:
            total_length = fileres.headers['content-length']
            print(f'Downloading {directory}, {total_length} bytes compressed')
            dl = 0
            data = fileres.read(CHUNK_SIZE)
            while len(data) > 0:
             dl += len(data)
             tfile.write(data)
             done = int(50 * dl / int(total_length))
             sys.stdout.write(f"\r[{'='*done}{''*(50-done)}]{dl}bytes downloaded")
             sys.stdout.flush()
             data = fileres.read(CHUNK_SIZE)
            if filename.endswith('.zip'):
              with ZipFile(tfile) as zfile:
                zfile.extractall(destination_path)
            else:
              with tarfile.open(tfile.name) as tarfile:
                tarfile.extractall(destination_path)
            print(f'\nDownloaded and uncompressed: {directory}')
    except HTTPError as e:
        print(f'Failed to load{download_url} to path {destination_path}')
        continue
    except OSError as e:
        print(f'Failed to load {download_url} to path {destination_path}')
        continue

print('Data source import complete.')
```

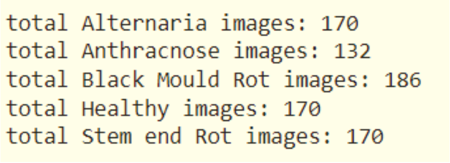
MODULE 2 SPECIFYING AND SPLITTING INPUT PATH

```
import numpy as np
import pandas as pd

import os
for dirname, _, filenames in os.walk('/kaggle/input'):
    for filename in filenames:
        print(os.path.join(dirname, filename))
```

```
a_dir = os.path.join('/kaggle/input/mangofruitdds/MangoFruitDDS/
                      SenMangoFruitDDS_original  /Alternaria')
b_dir = os.path.join('/kaggle/input/mangofruitdds/MangoFruitDDS/
                      SenMangoFruitDDS_original/Anthracnose')
c_dir = os.path.join('/kaggle/input/mangofruitdds/MangoFruitDDS/
                      SenMangoFruitDDS_original/Black Mould Rot')
d_dir = os.path.join('/kaggle/input/mangofruitdds/MangoFruitDDS/
                      SenMangoFruitDDS_original/Healthy')
e_dir = os.path.join('/kaggle/input/mangofruitdds/MangoFruitDDS/
                      SenMangoFruitDDS_original/Stem end Rot')
```

```
a_names = os.listdir(a_dir)
print(a_names[:10])
b_names = os.listdir(b_dir)
print(b_names[:10])
c_names = os.listdir(c_dir)
print(c_names[:10])
d_names = os.listdir(d_dir)
print(d_names[:10])
e_names = os.listdir(e_dir)
print(e_names[:10])
```

```

print('total Alternaria images:', len(os.listdir(a_dir)))
print('total Anthracnose images:', len(os.listdir(b_dir)))
print('total Black Mould Rot images:', len(os.listdir(c_dir)))
print('total Healthy images:', len(os.listdir(a_dir)))
print('total Stem end Rot images:', len(os.listdir(a_dir)))
```



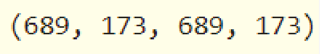
```
total Alternaria images: 170
total Anthracnose images: 132
total Black Mould Rot images: 186
total Healthy images: 170
total Stem end Rot images: 170
```

```
%matplotlib inline

import matplotlib.pyplot as plt
import matplotlib.image as mpimg

# Parameters for our graph; we'll output images in a 4x4 configuration
nrows = 10
ncols = 4

# Index for iterating over images
pic_index = 0
```

```python
# Set up matplotlib fig, and size it to fit 4x4 pics
fig = plt.gcf()
fig.set_size_inches(ncols * 4, nrows * 4)

pic_index += 8
a_pix = [os.path.join(a_dir, fname)
                for fname in a_names[pic_index-8:pic_index]]
b_pix = [os.path.join(b_dir, fname)
                for fname in b_names[pic_index-8:pic_index]]
c_pix = [os.path.join(c_dir, fname)
                for fname in c_names[pic_index-8:pic_index]]
d_pix = [os.path.join(d_dir, fname)
                for fname in d_names[pic_index-8:pic_index]]
e_pix = [os.path.join(e_dir, fname)
                for fname in e_names[pic_index-8:pic_index]]

for i, img_path in enumerate(a_pix + b_pix + c_pix + d_pix + e_pix):
  sp = plt.subplot(nrows, ncols, (i % (nrows * ncols)) + 1)
  sp.axis('Off')
  img = mpimg.imread(img_path)
  plt.imshow(img)
```

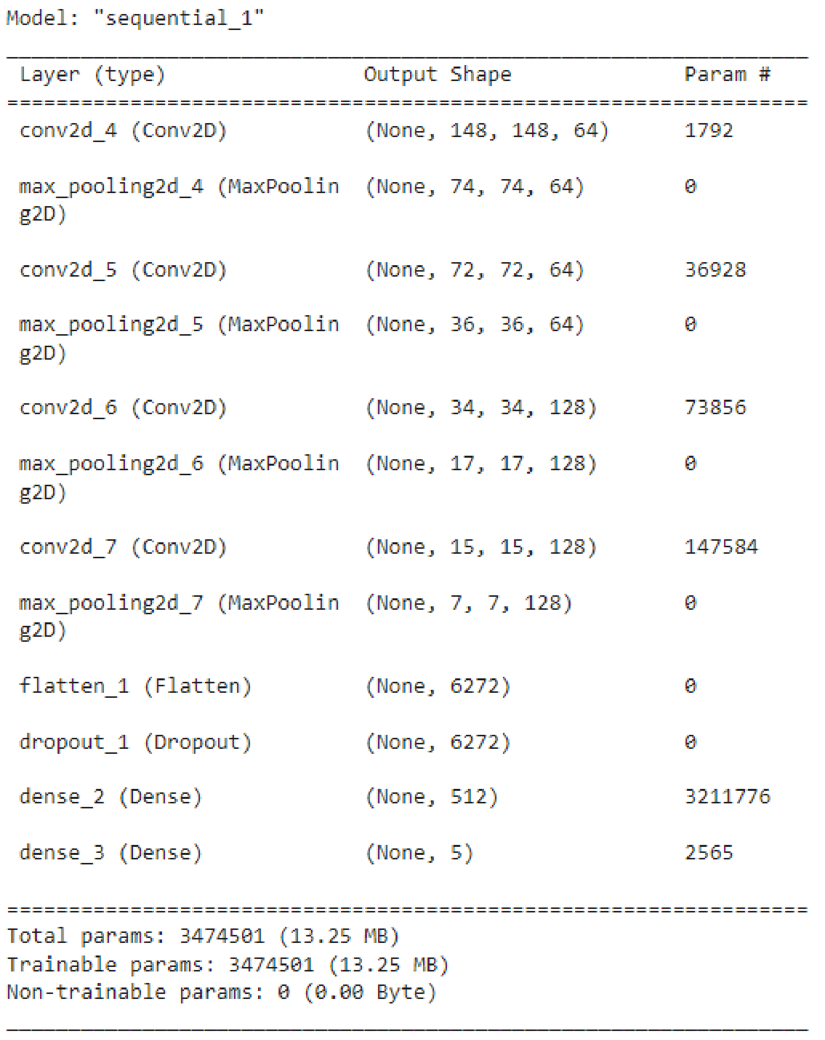
```python
from sklearn.model_selection import train_test_split
data_dir = '/kaggle/input/mangofruitdds/MangoFruitDDS/SenMangoFruitDDS_original'
batch_size = 64
epochs = 30
input_shape = (300, 300, 3)
image_paths = []
labels = []
for category in os.listdir(data_dir):
    category_dir = os.path.join(data_dir, category)
    if os.path.isdir(category_dir):
        for image_filename in os.listdir(category_dir):
            if image_filename.endswith('.jpg'):
                image_path = os.path.join(category_dir, image_filename)
                image_paths.append(image_path)
                labels.append(category)
```

```
train_image_paths, test_image_paths, train_labels, test_labels = train_test_split
                               (image_paths, labels, test_size=0.2, random_state=42)
len(train_image_paths),len(test_image_paths), len(train_labels), len(test_labels)
```



```
(689, 173, 689, 173)
```

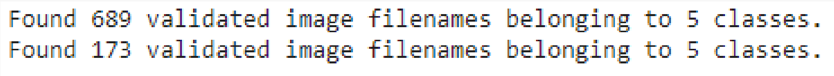
```
import tensorflow as tf

model = tf.keras.models.Sequential([
    tf.keras.layers.Conv2D(64, (3,3), activation='relu', input_shape=(150, 150, 
3)),
    tf.keras.layers.MaxPooling2D(2, 2),
    tf.keras.layers.Conv2D(64, (3,3), activation='relu'),
    tf.keras.layers.MaxPooling2D(2,2),
    tf.keras.layers.Conv2D(128, (3,3), activation='relu'),
    tf.keras.layers.MaxPooling2D(2,2),
    tf.keras.layers.Conv2D(128, (3,3), activation='relu'),
    tf.keras.layers.MaxPooling2D(2,2),
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dropout(0.5),
    tf.keras.layers.Dense(512, activation='relu'),
    tf.keras.layers.Dense(5, activation='softmax')
])
model.summary()
```



```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 conv2d_4 (Conv2D)           (None, 148, 148, 64)      1792      
                                                                 
 max_pooling2d_4 (MaxPoolin  (None, 74, 74, 64)        0         
 g2D)                                                            
                                                                 
 conv2d_5 (Conv2D)           (None, 72, 72, 64)        36928     
                                                                 
 max_pooling2d_5 (MaxPoolin  (None, 36, 36, 64)        0         
 g2D)                                                            
                                                                 
 conv2d_6 (Conv2D)           (None, 34, 34, 128)       73856     
                                                                 
 max_pooling2d_6 (MaxPoolin  (None, 17, 17, 128)       0         
 g2D)                                                            
                                                                 
 conv2d_7 (Conv2D)           (None, 15, 15, 128)       147584    
                                                                 
 max_pooling2d_7 (MaxPoolin  (None, 7, 7, 128)         0         
 g2D)                                                            
                                                                 
 flatten_1 (Flatten)         (None, 6272)              0         
                                                                 
 dropout_1 (Dropout)         (None, 6272)              0         
                                                                 
 dense_2 (Dense)             (None, 512)               3211776   
                                                                 
 dense_3 (Dense)             (None, 5)                 2565      
                                                                 
=================================================================
Total params: 3474501 (13.25 MB)
Trainable params: 3474501 (13.25 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

```
from tensorflow.keras.optimizers import RMSprop

model.compile(loss='categorical_crossentropy',
              optimizer=RMSprop(learning_rate=0.001),
              metrics=['accuracy'])
```

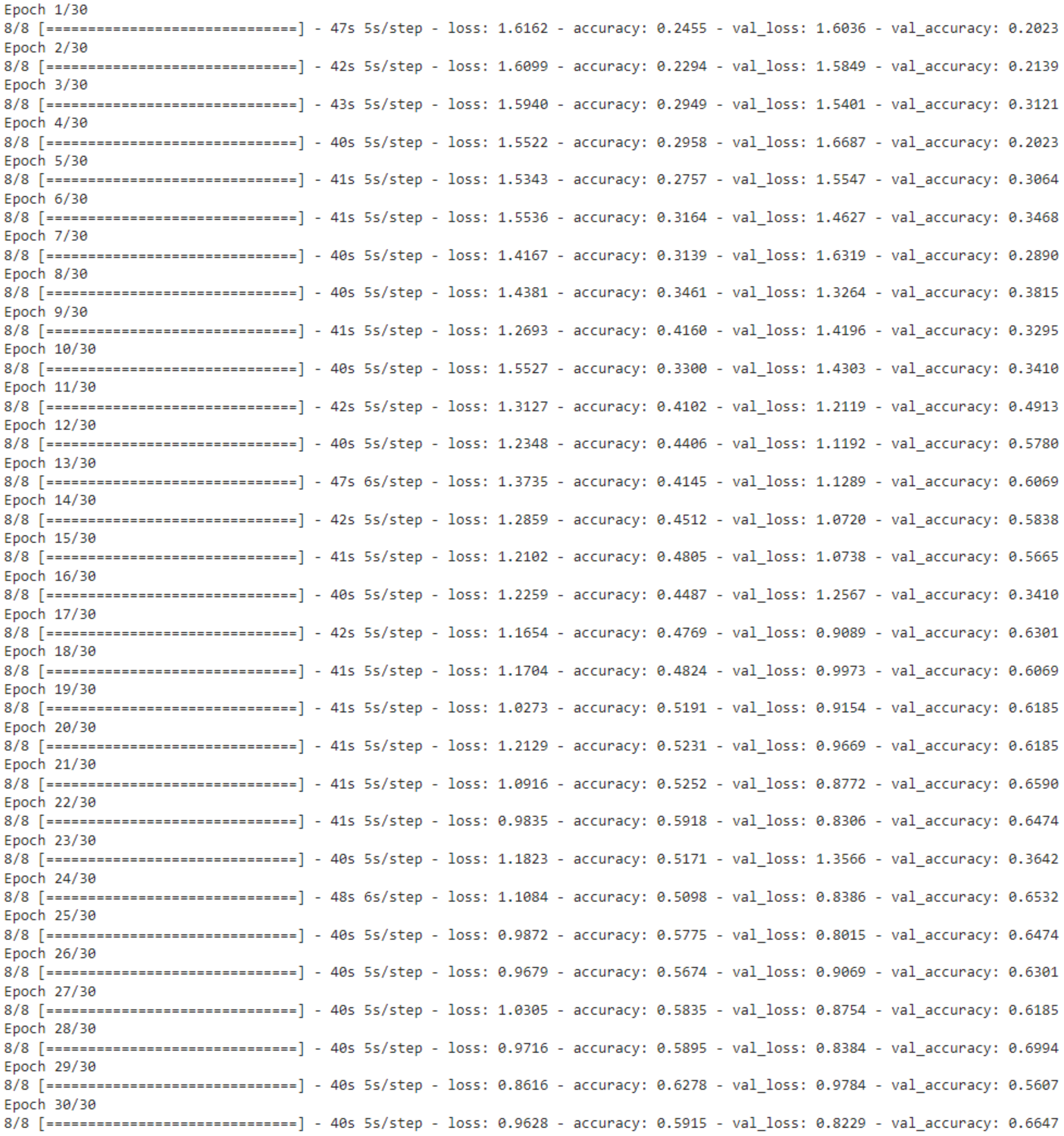
```
from tensorflow.keras.preprocessing.image import ImageDataGenerator

training_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')

validation_datagen = ImageDataGenerator(rescale = 1./255)

train_generator = training_datagen.flow_from_dataframe(
    pd.DataFrame({'image_path': train_image_paths, 'label': train_labels}),
    x_col='image_path',
    y_col='label',
    target_size=(150,150),
    batch_size=64,
    class_mode='categorical'
)

validation_generator = validation_datagen.flow_from_dataframe(
    pd.DataFrame({'image_path': test_image_paths, 'label': test_labels}),
    x_col='image_path',
    y_col='label',
    target_size=(150,150),
    batch_size=64,
    class_mode='categorical'
)
```



```
Found 689 validated image filenames belonging to 5 classes.
Found 173 validated image filenames belonging to 5 classes.
```

CALCULATING LOSS AND ACCURACY

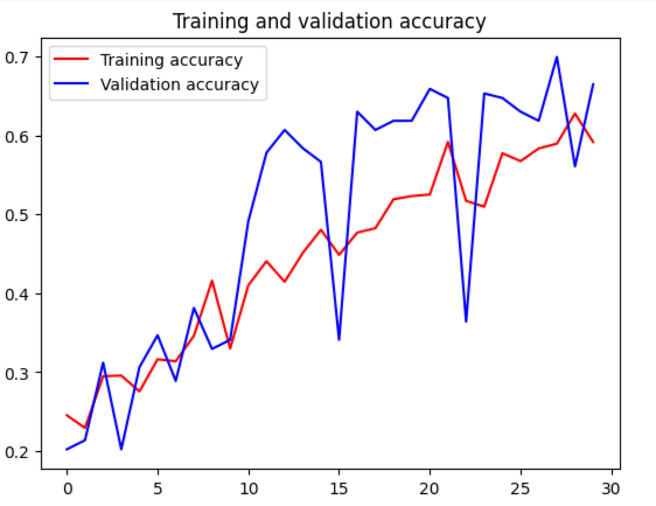
```
history = model.fit(train_generator, epochs=30, steps_per_epoch=8, validation_data
= validation_generator, verbose = 1, validation_steps=3)
```

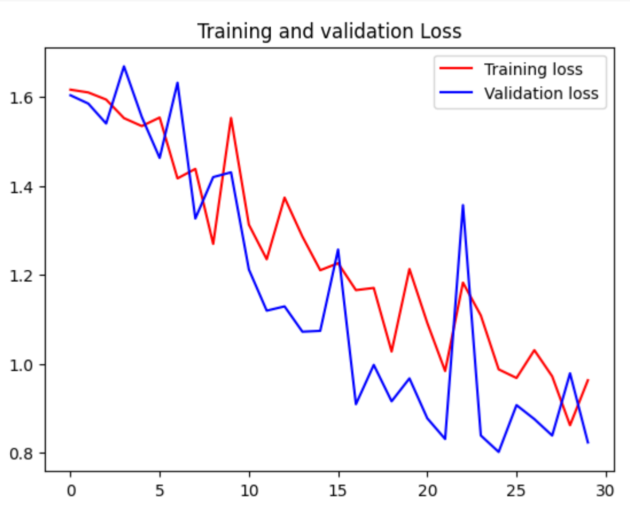


```
Epoch 1/30
8/8 [==============================] - 47s 5s/step - loss: 1.6162 - accuracy: 0.2455 - val_loss: 1.6036 - val_accuracy: 0.2023
Epoch 2/30
8/8 [==============================] - 42s 5s/step - loss: 1.6099 - accuracy: 0.2294 - val_loss: 1.5849 - val_accuracy: 0.2139
Epoch 3/30
8/8 [==============================] - 43s 5s/step - loss: 1.5940 - accuracy: 0.2949 - val_loss: 1.5401 - val_accuracy: 0.3121
Epoch 4/30
8/8 [==============================] - 40s 5s/step - loss: 1.5522 - accuracy: 0.2958 - val_loss: 1.6687 - val_accuracy: 0.2023
Epoch 5/30
8/8 [==============================] - 41s 5s/step - loss: 1.5343 - accuracy: 0.2757 - val_loss: 1.5547 - val_accuracy: 0.3064
Epoch 6/30
8/8 [==============================] - 41s 5s/step - loss: 1.5536 - accuracy: 0.3164 - val_loss: 1.4627 - val_accuracy: 0.3468
Epoch 7/30
8/8 [==============================] - 40s 5s/step - loss: 1.4167 - accuracy: 0.3139 - val_loss: 1.6319 - val_accuracy: 0.2890
Epoch 8/30
8/8 [==============================] - 40s 5s/step - loss: 1.4381 - accuracy: 0.3461 - val_loss: 1.3264 - val_accuracy: 0.3815
Epoch 9/30
8/8 [==============================] - 41s 5s/step - loss: 1.2693 - accuracy: 0.4160 - val_loss: 1.4196 - val_accuracy: 0.3295
Epoch 10/30
8/8 [==============================] - 40s 5s/step - loss: 1.5527 - accuracy: 0.3300 - val_loss: 1.4303 - val_accuracy: 0.3410
Epoch 11/30
8/8 [==============================] - 42s 5s/step - loss: 1.3127 - accuracy: 0.4102 - val_loss: 1.2119 - val_accuracy: 0.4913
Epoch 12/30
8/8 [==============================] - 40s 5s/step - loss: 1.2348 - accuracy: 0.4406 - val_loss: 1.1192 - val_accuracy: 0.5780
Epoch 13/30
8/8 [==============================] - 47s 6s/step - loss: 1.3735 - accuracy: 0.4145 - val_loss: 1.1289 - val_accuracy: 0.6069
Epoch 14/30
8/8 [==============================] - 42s 5s/step - loss: 1.2859 - accuracy: 0.4512 - val_loss: 1.0720 - val_accuracy: 0.5838
Epoch 15/30
8/8 [==============================] - 41s 5s/step - loss: 1.2102 - accuracy: 0.4805 - val_loss: 1.0738 - val_accuracy: 0.5665
Epoch 16/30
8/8 [==============================] - 40s 5s/step - loss: 1.2259 - accuracy: 0.4487 - val_loss: 1.2567 - val_accuracy: 0.3410
Epoch 17/30
8/8 [==============================] - 42s 5s/step - loss: 1.1654 - accuracy: 0.4769 - val_loss: 0.9089 - val_accuracy: 0.6301
Epoch 18/30
8/8 [==============================] - 41s 5s/step - loss: 1.1704 - accuracy: 0.4824 - val_loss: 0.9973 - val_accuracy: 0.6069
Epoch 19/30
8/8 [==============================] - 41s 5s/step - loss: 1.0273 - accuracy: 0.5191 - val_loss: 0.9154 - val_accuracy: 0.6185
Epoch 20/30
8/8 [==============================] - 41s 5s/step - loss: 1.2129 - accuracy: 0.5231 - val_loss: 0.9669 - val_accuracy: 0.6185
Epoch 21/30
8/8 [==============================] - 41s 5s/step - loss: 1.0916 - accuracy: 0.5252 - val_loss: 0.8772 - val_accuracy: 0.6590
Epoch 22/30
8/8 [==============================] - 41s 5s/step - loss: 0.9835 - accuracy: 0.5918 - val_loss: 0.8306 - val_accuracy: 0.6474
Epoch 23/30
8/8 [==============================] - 40s 5s/step - loss: 1.1823 - accuracy: 0.5171 - val_loss: 1.3566 - val_accuracy: 0.3642
Epoch 24/30
8/8 [==============================] - 48s 6s/step - loss: 1.1084 - accuracy: 0.5098 - val_loss: 0.8386 - val_accuracy: 0.6532
Epoch 25/30
8/8 [==============================] - 40s 5s/step - loss: 0.9872 - accuracy: 0.5775 - val_loss: 0.8015 - val_accuracy: 0.6474
Epoch 26/30
8/8 [==============================] - 40s 5s/step - loss: 0.9679 - accuracy: 0.5674 - val_loss: 0.9069 - val_accuracy: 0.6301
Epoch 27/30
8/8 [==============================] - 40s 5s/step - loss: 1.0305 - accuracy: 0.5835 - val_loss: 0.8754 - val_accuracy: 0.6185
Epoch 28/30
8/8 [==============================] - 40s 5s/step - loss: 0.9716 - accuracy: 0.5895 - val_loss: 0.8384 - val_accuracy: 0.6994
Epoch 29/30
8/8 [==============================] - 40s 5s/step - loss: 0.8616 - accuracy: 0.6278 - val_loss: 0.9784 - val_accuracy: 0.5607
Epoch 30/30
8/8 [==============================] - 40s 5s/step - loss: 0.9628 - accuracy: 0.5915 - val_loss: 0.8229 - val_accuracy: 0.6647
```

PLOTTING LOSS AND ACCURACY

```
import matplotlib.pyplot as plt

# Plot the results
acc = history.history['accuracy']
val_acc = history.history['val_accuracy']
loss = history.history['loss']
val_loss = history.history['val_loss']

epochs = range(len(acc))

plt.plot(epochs, acc, 'r', label='Training accuracy')
plt.plot(epochs, val_acc, 'b', label='Validation accuracy')
plt.title('Training and validation accuracy')
plt.legend(loc=0)
plt.figure()

plt.plot(epochs, loss, 'r', label='Training loss')
plt.plot(epochs, val_loss, 'b', label='Validation loss')
plt.title('Training and validation Loss')
plt.legend(loc=0)
plt.figure()

plt.show()
```

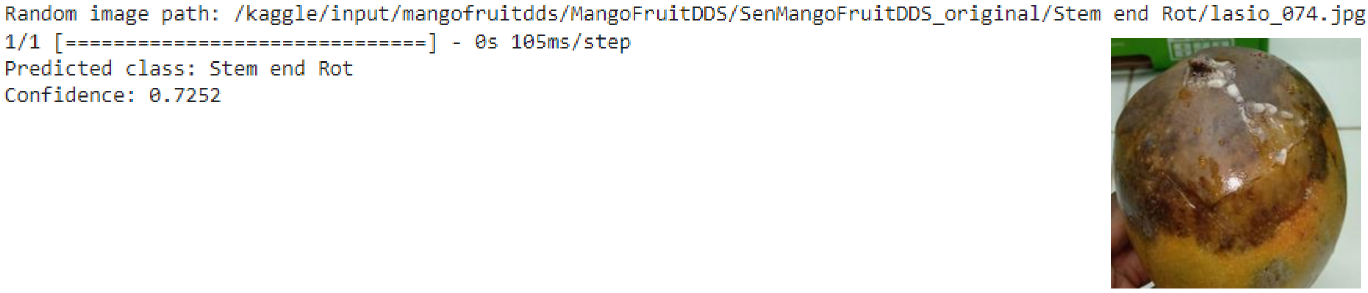




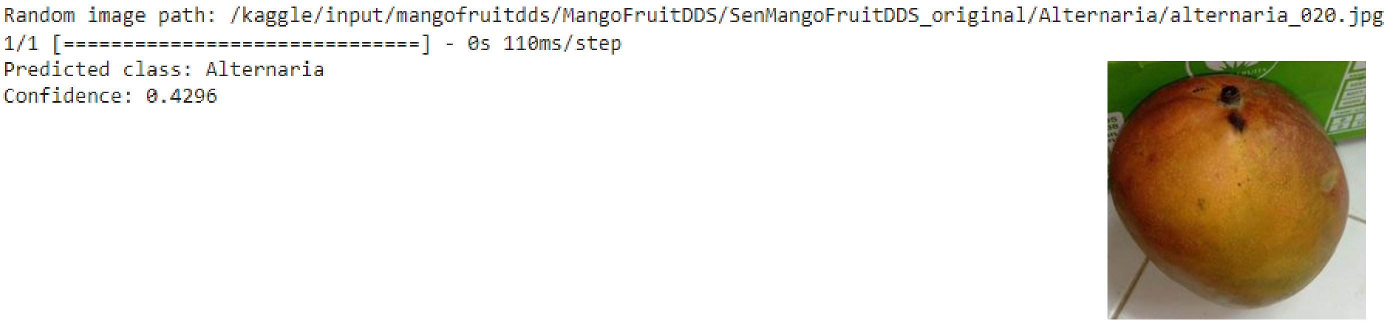
```
import os
import random
import numpy as np
import pandas as pd
import tensorflow as tf
from tensorflow.keras.preprocessing import image
from tensorflow.keras.preprocessing.image import ImageDataGenerator
import matplotlib.pyplot as plt

# Save the model
model.save('/kaggle/working/mango_disease_model.h5')
```

```python
# Function to classify a single image
def classify_image(image_path):
    img = image.load_img(image_path, target_size=(150, 150))
    img_array = image.img_to_array(img)
    img_array = np.expand_dims(img_array, axis=0)
    img_array /= 255.

    # Load the saved model
    model = tf.keras.models.load_model('/kaggle/working/mango_disease_model.h5')

    # Make prediction
    predictions = model.predict(img_array)
    predicted_class = np.argmax(predictions[0])

    # Get class labels from train_generator
    class_labels = list(train_generator.class_indices.keys())

    # Print the prediction
    print(f"Predicted class: {class_labels[predicted_class]}")
    print(f"Confidence: {predictions[0][predicted_class]:.4f}")

# List files in the dataset directories
data_dir = '/kaggle/input/mangofruitdds/MangoFruitDDS/SenMangoFruitDDS_original'
categories = ["Alternaria", "Anthracnose", "Black Mould Rot", "Healthy", "Stem end Rot"]

# Get a random image path for testing
def get_random_image_path():
    category = random.choice(categories)
    category_dir = os.path.join(data_dir, category)
    if os.path.isdir(category_dir):
        image_filename = random.choice(os.listdir(category_dir))
        return os.path.join(category_dir, image_filename)
    return None

random_image_path = get_random_image_path()
print(f"Random image path: {random_image_path}")

if random_image_path:
    classify_image(random_image_path)
else:
    print("No valid image found for classification.")
```

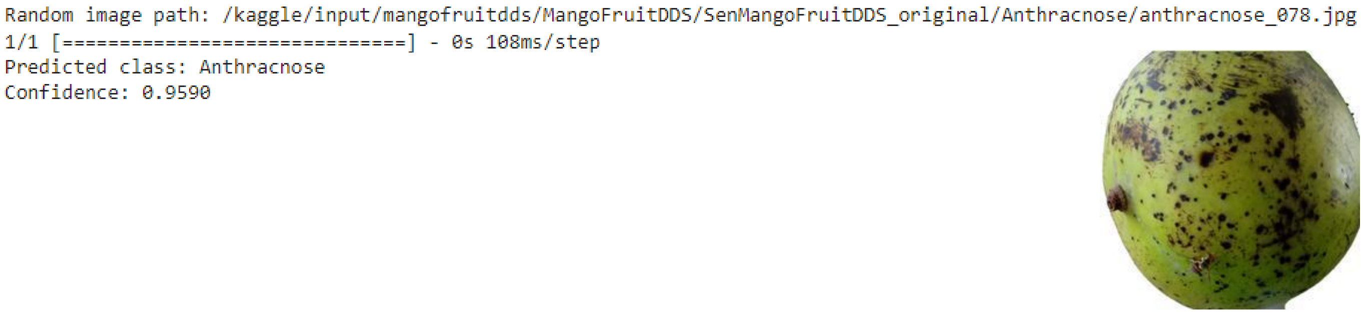
OUPUT :



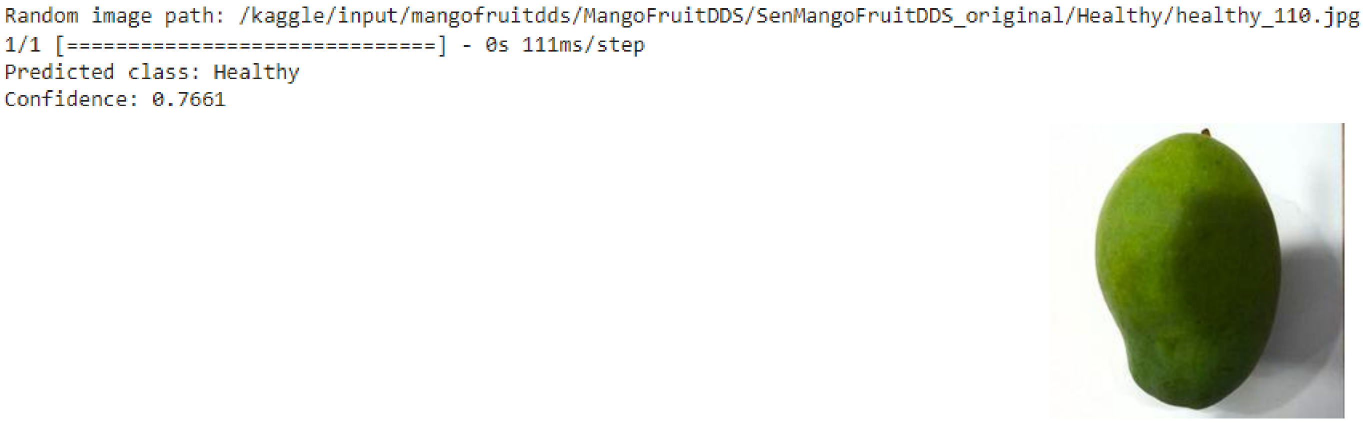
```
Random image path: /kaggle/input/mangofruitdds/MangoFruitDDS/SenMangoFruitDDS_original/Stem end Rot/lasio_074.jpg
1/1 [==============================] - 0s 105ms/step
Predicted class: Stem end Rot
Confidence: 0.7252
```



```
Random image path: /kaggle/input/mangofruitdds/MangoFruitDDS/SenMangoFruitDDS_original/Alternaria/alternaria_020.jpg
1/1 [==============================] - 0s 110ms/step
Predicted class: Alternaria
Confidence: 0.4296
```



```
Random image path: /kaggle/input/mangofruitdds/MangoFruitDDS/SenMangoFruitDDS_original/Anthracnose/anthracnose_078.jpg
1/1 [==============================] - 0s 108ms/step
Predicted class: Anthracnose
Confidence: 0.9590
```



```
Random image path: /kaggle/input/mangofruitdds/MangoFruitDDS/SenMangoFruitDDS_original/Healthy/healthy_110.jpg
1/1 [==============================] - 0s 111ms/step
Predicted class: Healthy
Confidence: 0.7661
```