Plant_Disease_Prediction_CNN_Image_Classifier.ipynb

File Edit View Insert Runtime Tools Help Last saved at 9:02 AM

Comment Share

+ Code + Text

Connect T4 Gemini

Seeding for reproducibility

```
import random
random.seed(0)

import numpy as np
np.random.seed(0)

import tensorflow as tf
tf.random.set_seed(0)
```

Importing the dependencies

```
import os
import json
from zipfile import ZipFile
from PIL import Image

import numpy as np
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras import layers, models
```

Data Curation

Upload the kaggle.json file

```
!pip install kaggle
```

```
Requirement already satisfied: kaggle in /usr/local/lib/python3.10/dist-packages (1.6.17)
Requirement already satisfied: six>=1.10 in /usr/local/lib/python3.10/dist-packages (from kaggle) (1.16.0)
Requirement already satisfied: certifi>=2023.7.22 in /usr/local/lib/python3.10/dist-packages (from kaggle) (2024.8.30)
Requirement already satisfied: python-dateutil in /usr/local/lib/python3.10/dist-packages (from kaggle) (2.8.2)
Requirement already satisfied: requests in /usr/local/lib/python3.10/dist-packages (from kaggle) (2.32.3)
Requirement already satisfied: tqdm in /usr/local/lib/python3.10/dist-packages (from kaggle) (4.66.5)
Requirement already satisfied: python-slugify in /usr/local/lib/python3.10/dist-packages (from kaggle) (8.0.4)
Requirement already satisfied: urllib3 in /usr/local/lib/python3.10/dist-packages (from kaggle) (2.2.3)
Requirement already satisfied: bleach in /usr/local/lib/python3.10/dist-packages (from kaggle) (6.1.0)
Requirement already satisfied: webencodings in /usr/local/lib/python3.10/dist-packages (from bleach->kaggle) (0.5.1)
Requirement already satisfied: text-unidecode>=1.3 in /usr/local/lib/python3.10/dist-packages (from python-slugify->kaggle) (1.3)
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.10/dist-packages (from requests->kaggle) (3.3.2)
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/dist-packages (from requests->kaggle) (3.10)
```

```
kaggle_credentails = json.load(open("kaggle.json"))
```

```
# setup Kaggle API key as environment variables
os.environ['KAGGLE_USERNAME'] = kaggle_credentails["username"]
os.environ['KAGGLE_KEY'] = kaggle_credentails["key"]
```

```
!kaggle datasets download -d abdallahalidev/plantvillage-dataset
```

```
Dataset URL: https://www.kaggle.com/datasets/abdallahalidev/plantvillage-dataset
License(s): CC-BY-NC-SA-4.0
Downloading plantvillage-dataset.zip to /content
100% 2.04G/2.04G [00:51<00:00, 43.1MB/s]
100% 2.04G/2.04G [00:51<00:00, 42.2MB/s]
```

```
!ls
```

```
kaggle.json  plantvillage-dataset.zip  sample_data  test_apple_black_rot.JPG
```

```
with ZipFile("plantvillage-dataset.zip", 'r') as zip_ref:
    zip_ref.extractall()
```

```
print(os.listdir("plantvillage dataset"))


print(len(os.listdir("plantvillage dataset/segmented")))
print(os.listdir("plantvillage dataset/segmented")[:5])

print(len(os.listdir("plantvillage dataset/color")))
print(os.listdir("plantvillage dataset/color")[:5])

print(len(os.listdir("plantvillage dataset/grayscale")))
print(os.listdir("plantvillage dataset/grayscale")[:5])
```

```
['grayscale', 'color', 'segmented']
38
['Cherry_(including_sour)___healthy', 'Grape___Black_rot', 'Strawberry___Leaf_scorch', 'Strawberry___healthy', 'Apple___Apple_scab']
38
['Cherry_(including_sour)___healthy', 'Grape___Black_rot', 'Strawberry___Leaf_scorch', 'Strawberry___healthy', 'Apple___Apple_scab']
38
['Cherry_(including_sour)___healthy', 'Grape___Black_rot', 'Strawberry___Leaf_scorch', 'Strawberry___healthy', 'Apple___Apple_scab']
```

Number of Classes = 38

```
print(len(os.listdir("plantvillage dataset/color/Grape___healthy")))
print(os.listdir("plantvillage dataset/color/Grape___healthy")[:5])
```

```
423
```

```
['bc9f03e8-d545-4ba9-a68d-7032b3c36670___Mt.N.V_HL 6075.JPG', '72cd93bf-63f6-476a-b3ae-43af2006c9f2___Mt.N.V_HL 6132.JPG', 'f229b33f-2257-49a7-a135-17d0ef3ca46b___Mt.N.V_HL 6033.JPG', '
```

Data Preprocessing

```
[ ] # Dataset Path
    base_dir = 'plantvillage dataset/color'
```

```
image_path = '/content/plantvillage dataset/color/Apple___Cedar_apple_rust/025b2b9a-0ec4-4132-96ac-7f2832d0db4a___FREC_C.Rust 3655.JPG'

img = mpimg.imread(image_path)

print(img.shape)
plt.imshow(img)
plt.axis('off')  # Turn off axis numbers
plt.show()
```

```
(256, 256, 3)
```

```
[ ] image_path = '/content/plantvillage dataset/color/Apple___Cedar_apple_rust/025b2b9a-0ec4-4132-96ac-7f2832d0db4a___FREC_C.Rust 3655.JPG'

    # Read the image
    img = mpimg.imread(image_path)

    print(img)
```

```
[[[179 175 176]
  [181 177 178]
  [184 180 181]
  ...
  [115 112 105]
  [108 105  98]
  [101  98  91]]

 [[176 172 173]
  [177 173 174]
  [178 174 175]
  ...
  [113 110 103]
  [111 108 101]
  [109 106  99]]

 [[180 176 177]
  [180 176 177]
  [180 176 177]
  ...
  [108 105  98]
  [111 108 101]
  [114 111 104]]

 ...

 [[137 128 119]
  [131 122 113]
  [125 116 107]
  ...
  [ 74  65  48]
  [ 74  65  48]
  [ 73  64  47]]

 [[136 127 118]
  [132 123 114]
  [128 119 110]
  ...
  [ 77  69  50]
  [ 75  67  48]
  [ 75  67  48]]

 [[133 124 115]
  [133 124 115]
  [132 123 114]
  ...
  [ 81  73  54]
  [ 80  72  53]
  [ 79  71  52]]]
```

```
[ ] # Image Parameters
    img_size = 224
    batch_size = 32
```

Train Test Split

```
[ ] # Image Data Generators
    data_gen = ImageDataGenerator(
        rescale=1./255,
        validation_split 0.2  # Use 20% of data for validation
```

```
    validation_split=0.2  # Use 20% of data for validation
)
```

```
# Train Generator
train_generator = data_gen.flow_from_directory(
    base_dir,
    target_size=(img_size, img_size),
    batch_size=batch_size,
    subset='training',
    class_mode='categorical'
)
```

```
Found 43456 images belonging to 38 classes.
```

```
# Validation Generator
validation_generator = data_gen.flow_from_directory(
    base_dir,
    target_size=(img_size, img_size),
    batch_size=batch_size,
    subset='validation',
    class_mode='categorical'
)
```

```
Found 10849 images belonging to 38 classes.
```

Convolutional Neural Network

```
# Model Definition
model = models.Sequential()

model.add(layers.Conv2D(32, (3, 3), activation='relu', input_shape=(img_size, img_size, 3)))
model.add(layers.MaxPooling2D(2, 2))

model.add(layers.Conv2D(64, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D(2, 2))


model.add(layers.Flatten())
model.add(layers.Dense(256, activation='relu')) Fully connected layer
model.add(layers.Dense(train_generator.num_classes, activation='softmax'))
```

```
/usr/local/lib/python3.10/dist-packages/keras/src/layers/convolutional/base_conv.py:107: UserWarning: Do not pass an `input_shape`/`input_dim` argument to a layer. When using Sequential
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
```

```
# model summary
model.summary()
```

Model: "sequential"

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 222, 222, 32)	896
max_pooling2d (MaxPooling2D)	(None, 111, 111, 32)	0
conv2d_1 (Conv2D)	(None, 109, 109, 64)	18,496
max_pooling2d_1 (MaxPooling2D)	(None, 54, 54, 64)	0
flatten (Flatten)	(None, 186624)	0
dense (Dense)	(None, 256)	47,776,000
dense_1 (Dense)	(None, 38)	9,766

Total params: 47,805,158 (182.36 MB)
Trainable params: 47,805,158 (182.36 MB)
Non-trainable params: 0 (0.00 B)

```
# Compile the Model
model.compile(optimizer='adam',
              loss='categorical_crossentropy',
              metrics=['accuracy'])
```

Model training

```
# Training the Model
history = model.fit(
    train_generator,
    steps_per_epoch=train_generator.samples // batch_size,  # Number of steps per epoch
    epochs=5,  # Number of epochs
    validation_data=validation_generator,
    validation_steps=validation_generator.samples // batch_size  # Validation steps
)
```

```
Epoch 1/5
/usr/local/lib/python3.10/dist-packages/keras/src/trainers/data_adapters/py_dataset_adapter.py:121: UserWarning: Your `PyDataset` class should call `super().__init__(**kwargs)` in its c
  self._warn_if_super_not_called()
1358/1358 ━━━━━━━━━━━━━━━━━━━━ 131s 91ms/step - accuracy: 0.6040 - loss: 1.8662 - val_accuracy: 0.8573 - val_loss: 0.4427
Epoch 2/5
/usr/lib/python3.10/contextlib.py:153: UserWarning: Your input ran out of data; interrupting training. Make sure that your dataset or generator can generate at least `steps_per_epoch *
  self.gen.throw(typ, value, traceback)
1358/1358 ━━━━━━━━━━━━━━━━━━━━ 1s 907us/step - accuracy: 0.0000e+00 - loss: 0.0000e+00 - val_accuracy: 1.0000 - val_loss: 0.6651
Epoch 3/5
1358/1358 ━━━━━━━━━━━━━━━━━━━━ 119s 87ms/step - accuracy: 0.9184 - loss: 0.2566 - val_accuracy: 0.8693 - val_loss: 0.4478
Epoch 4/5
1358/1358 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step - accuracy: 0.0000e+00 - loss: 0.0000e+00 - val_accuracy: 1.0000 - val_loss: 1.4305e-06
Epoch 5/5
1358/1358 ━━━━━━━━━━━━━━━━━━━━ 108s 79ms/step - accuracy: 0.9673 - loss: 0.1027 - val_accuracy: 0.8663 - val_loss: 0.5127
```

Model Evaluation

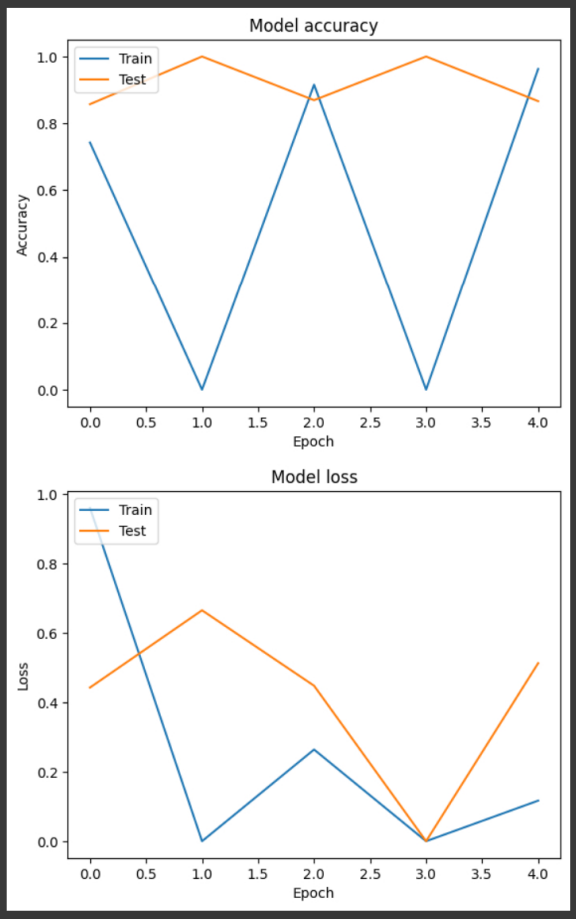
```
# Model Evaluation
```

```
print("Evaluating model...")
val_loss, val_accuracy = model.evaluate(validation_generator, steps=validation_generator.samples // batch_size)
print(f"Validation Accuracy: {val_accuracy * 100:.2f}%")
```

```
Evaluating model...
339/339 ━━━━━━━━━━━━━━━━━━━━ 19s 57ms/step - accuracy: 0.8650 - loss: 0.5054
Validation Accuracy: 86.63%
```

```
# Plot training & validation accuracy values
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('Model accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Train', 'Test'], loc='upper left')
plt.show()

# Plot training & validation loss values
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('Model loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['Train', 'Test'], loc='upper left')
plt.show()
```



Building a Predictive System

```
# Function to Load and Preprocess the Image using Pillow
def load_and_preprocess_image(image_path, target_size=(224, 224)):
    img = Image.open(image_path)
    img = img.resize(target_size)
    img_array = np.array(img)
    img_array = np.expand_dims(img_array, axis=0)
    img_array = img_array.astype('float32') / 255.
    return img_array

# Function to Predict the Class of an Image
def predict_image_class(model, image_path, class_indices):
    preprocessed_img = load_and_preprocess_image(image_path)
    predictions = model.predict(preprocessed_img)
    predicted_class_index = np.argmax(predictions, axis=1)[0]
    predicted_class_name = class_indices[predicted_class_index]
    return predicted_class_name
```

```
# Create a mapping from class indices to class names
class_indices = {v: k for k, v in train_generator.class_indices.items()}
```

```
class_indices
```

```
{0: 'Apple___Apple_scab',
 1: 'Apple___Black_rot',
 2: 'Apple___Cedar_apple_rust',
 3: 'Apple___healthy',
```

```
 4: 'Blueberry___healthy',
 5: 'Cherry_(including_sour)___Powdery_mildew',
 6: 'Cherry_(including_sour)___healthy',
 7: 'Corn_(maize)___Cercospora_leaf_spot Gray_leaf_spot',
 8: 'Corn_(maize)___Common_rust_',
 9: 'Corn_(maize)___Northern_Leaf_Blight',
 10: 'Corn_(maize)___healthy',
 11: 'Grape___Black_rot',
 12: 'Grape___Esca_(Black_Measles)',
 13: 'Grape___Leaf_blight_(Isariopsis_Leaf_Spot)',
 14: 'Grape___healthy',
 15: 'Orange___Haunglongbing_(Citrus_greening)',
 16: 'Peach___Bacterial_spot',
 17: 'Peach___healthy',
 18: 'Pepper,_bell___Bacterial_spot',
 19: 'Pepper,_bell___healthy',
 20: 'Potato___Early_blight',
 21: 'Potato___Late_blight',
 22: 'Potato___healthy',
 23: 'Raspberry___healthy',
 24: 'Soybean___healthy',
 25: 'Squash___Powdery_mildew',
 26: 'Strawberry___Leaf_scorch',
 27: 'Strawberry___healthy',
 28: 'Tomato___Bacterial_spot',
 29: 'Tomato___Early_blight',
 30: 'Tomato___Late_blight',
 31: 'Tomato___Leaf_Mold',
 32: 'Tomato___Septoria_leaf_spot',
 33: 'Tomato___Spider_mites Two-spotted_spider_mite',
 34: 'Tomato___Target_Spot',
 35: 'Tomato___Tomato_Yellow_Leaf_Curl_Virus',
 36: 'Tomato___Tomato_mosaic_virus',
 37: 'Tomato___healthy'}
```

```
[ ] # saving the class names as json file
    json.dump(class_indices, open('class_indices.json', 'w'))
```

```
[ ] # Example Usage
    image_path = '/content/test_apple_black_rot.JPG'
    #image_path = '/content/test_blueberry_healthy.jpg'
    #image_path = '/content/test_potato_early_blight.jpg'
    predicted_class_name = predict_image_class(model, image_path, class_indices)

    # Output the result
    print("Predicted Class Name:", predicted_class_name)
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 489ms/step
Predicted Class Name: Apple___Black_rot
```

Save the model to Google drive or local

```
[ ] # Save the model in HDF5 (.h5) format
    model.save('/content/plant_disease_prediction_model.h5')
```

```
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. This file format is considered legacy. We recommend using instead the nati
```

```
[ ] # Mount Google Drive to save the file there
    from google.colab import drive
    drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

```
[ ] # Move the model to Google Drive
    !cp /content/plant_disease_prediction_model.h5 /content/drive/MyDrive/
```

```
[ ] Start coding or generate with AI.
```

Colab paid products - Cancel contracts here