NANDHAENGINEERINGCOLLEGE
ERODE - 638052

(An Autonomous Institution, Affiliated to Anna University, Chennai)

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

22CSX01–DeepLearning

Assignment - II

Submitted by

TEAM-10

RUMESHKUMARAN K (22CS077)
SAARUMATHI T (22CS078)

1. You are building a CNN to detect diseases in plant leaves, but the dataset is highly imbalanced (few images of diseased plants). How would you address this issue?

Signature of the students **Signature of the faculty**

22CS077:

22CS078:

Question:

You are building a CNN to detect diseases in plant leaves, but the dataset is highly imbalanced (few images of diseased plants). How would you address this issue?

SOLUTION:

Model Overview

- Model Type: The model used for the plant disease detection task is a **Convolutional Neural Network (CNN)**, which is particularly suited for image classification tasks. A pretrained VGG16 model was used as the base to leverage transfer learning.
- Transfer Learning: VGG16, a deep CNN pre-trained on ImageNet, was used without its top layers (include_top=False). The top layers were replaced with custom layers (flattening, dense layers, and a final output layer with a sigmoid activation) to classify the input images as either "healthy" or "diseased".

Model Architecture and Training:

We employed a **VGG16-based CNN** for plant disease detection. Using **transfer learning**, we leveraged the power of VGG16 pre-trained on ImageNet and fine-tuned it for the binary classification task (healthy vs. diseased). The model architecture was modified by replacing the top layers of VGG16 with custom layers suitable for our dataset.

Image Prediction:

To test the model's performance, new images can be passed to the model, and it will output whether the image contains a "healthy" or "diseased" plant leaf. The model uses a sigmoid activation function, producing values between 0 (healthy) and 1 (diseased). The output determines whether the plant is diseased or healthy based on a threshold of 0.5.

CODING:

Unzip the dataset:

```
import zipfile

import os

# Path to your uploaded zip file

zip_path = '/content/Leaf disease.v2-2024-05-14-8-41am-change-labels.tensorflow.zip'  # Replace with your zip file path

extract_path = '/content/dataset'  # Destination folder
```

```
# Unzip the dataset
with zipfile.ZipFile(zip_path, 'r') as zip_ref:
    zip_ref.extractall(extract_path)
# Check extracted folders
print(os.listdir(extract_path))
```

arrangement of dataset:

```
import shutil
import os
# Define the directories where your images are located
source_train_dir = '/content/dataset/train'
source_valid_dir = '/content/dataset/valid'
source_test_dir = '/content/dataset/test'
# Define the class name (since your dataset only contains diseased images)
class_name = 'diseased'
# Create subfolders for each class inside train, valid, and test directories
for directory in [source_train_dir, source_valid_dir, source_test_dir]:
    class_folder = os.path.join(directory, class_name)
    os.makedirs(class_folder, exist_ok=True)
    # Move all images from the original directory to the class folder
    for image_file in os.listdir(directory):
        src_path = os.path.join(directory, image_file)
        if os.path.isfile(src_path)
```

```
        shutil.move(src_path, os.path.join(class_folder, image_file))

print("Images have been moved to respective class folders.")

# Step 2: Load the images using ImageDataGenerator

from tensorflow.keras.preprocessing.image import ImageDataGenerator

# Initialize the ImageDataGenerator with rescaling and augmentation if desired

train_datagen = ImageDataGenerator(rescale=1./255)

valid_datagen = ImageDataGenerator(rescale=1./255)

test_datagen = ImageDataGenerator(rescale=1./255)

# Set up the generators for train, valid, and test data

train_generator = train_datagen.flow_from_directory(

    directory=source_train_dir,

    target_size=(224, 224),  # Resize images to 224x224 (VGG16 input size)

    batch_size=32,

    class_mode='binary'  # Since it's a binary classification problem

)

valid_generator = valid_datagen.flow_from_directory(

    directory=source_valid_dir,

    target_size=(224, 224),

    batch_size=32,

    class_mode='binary'

)

test_generator = test_datagen.flow_from_directory(

    directory=source_test_dir,
```

```
    target_size=(224, 224),
    batch_size=32,
    class_mode='binary'
)

print("Data generators have been set up.")
```

Model Buliding:

```
import shutil
import os
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.applications import VGG16
from tensorflow.keras import layers, models
from sklearn.utils import class_weight
import numpy as np
# Define the paths to the directories
train_dir = '/content/dataset/train'
valid_dir = '/content/dataset/valid'
test_dir = '/content/dataset/test'
# Create image generators for training, validation, and testing
train_datagen = ImageDataGenerator(
    rescale=1./255,  # Rescale images to [0, 1] range
    rotation_range=30,
```

```
    width_shift_range=0.2,
    height_shift_range=0.2,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True,
    fill_mode='nearest'
)
valid_datagen = ImageDataGenerator(rescale=1./255)
test_datagen = ImageDataGenerator(rescale=1./255)
# Flow images in batches from the directories
train_generator = train_datagen.flow_from_directory(
    train_dir,
    target_size=(224, 224),  # VGG16 expects 224x224 images
    batch_size=32,
    class_mode='binary'  # Binary classification (healthy vs diseased)
)
valid_generator = valid_datagen.flow_from_directory(
    valid_dir,
    target_size=(224, 224),
    batch_size=32,
    class_mode='binary'
)
test_generator = test_datagen.flow_from_directory(test_dir,target_size=(224,
224),batch_size=32, class_mode='binary') # Check class indices
```

```
print(train_generator.class_indices)

# Calculate class weights for imbalanced dataset

class_weights = class_weight.compute_class_weight('balanced',
classes=np.unique(train_generator.classes),y=train_generator.classes)

class_weight_dict = dict(enumerate(class_weights))

# Load the pretrained VGG16 model (without the top layers)

base_model = VGG16(weights='imagenet', include_top=False, input_shape=(224, 224, 3))

# Freeze the base model layers

for layer in base_model.layers:

    layer.trainable = False

# Build the model by adding custom layers

model = models.Sequential([

    base_model,

    layers.Flatten(),

    layers.Dense(256, activation='relu'),

    layers.Dropout(0.5),

    layers.Dense(1, activation='sigmoid')  # For binary classification (healthy vs diseased)

])

# Compile the model

model.compile(

    optimizer='adam',

    loss='binary_crossentropy',  # Binary classification loss function

    metrics=['accuracy']

)
```

```
# Train the model

history = model.fit(

    train_generator,

    steps_per_epoch=train_generator.samples // train_generator.batch_size,

    validation_data=valid_generator,

    validation_steps=valid_generator.samples // valid_generator.batch_size,

    epochs=10,

    class_weight=class_weight_dict  # Pass class weights to address class imbalance

)

# Evaluate the model on the test set

test_loss, test_acc = model.evaluate(test_generator)

print(f"Test accuracy: {test_acc}")

# Evaluate the model on the test dataset

test_loss, test_accuracy = model.evaluate(test_generator, steps=test_generator.samples //
test_generator.batch_size)

print(f"Test Loss: {test_loss}")

print(f"Test Accuracy: {test_accuracy}")

import matplotlib.pyplot as plt

# Plot training and validation accuracy

plt.plot(history.history['accuracy'], label='Training Accuracy')

plt.plot(history.history['val_accuracy'], label='Validation Accuracy')

plt.title('Accuracy over Epochs')

plt.xlabel('Epochs')

plt.ylabel('Accuracy')

plt.legend()
```

```
plt.show()

# Plot training and validation loss

plt.plot(history.history['loss'], label='Training Loss')

plt.plot(history.history['val_loss'], label='Validation Loss')

plt.title('Loss over Epochs')

plt.xlabel('Epochs')

plt.ylabel('Loss')

plt.legend()

plt.show()
```

Test the Code:

```
import numpy as np

from tensorflow.keras.preprocessing import image

import matplotlib.pyplot as plt

# Load the trained model (replace 'model' with your trained model variable if needed)

# model = ... (if you saved it earlier, load it like: model =
tf.keras.models.load_model('path_to_model'))

# Function to load and preprocess the image for prediction

def load_and_preprocess_image(img_path):

    # Load image with the same target size as used in the training (224x224 for VGG16)

    img = image.load_img(img_path, target_size=(224, 224))

    # Convert the image to a numpy array and rescale the pixel values

    img_array = image.img_to_array(img) / 255.0

    # Expand dimensions to match the batch size dimension

    img_array = np.expand_dims(img_array, axis=0)
```

```
    return img_array

# Predict the image class
def predict_image(img_path):
    # Preprocess the image
    img_array = load_and_preprocess_image(img_path)
    # Make prediction
    prediction = model.predict(img_array)  # Predicts in the range [0, 1] (sigmoid output)
    # Check the prediction
    if prediction < 0.5:
        print("The image is Diseased (Class 0).")
        # You can insert any output for Healthy
    else:
        print("The image is Healthy (Class 1).")
        # You can insert any output for Diseased
    # Show the image
    img = image.load_img(img_path)
    plt.imshow(img)
    plt.axis('off')  # Turn off axis numbers
    plt.show()
# Example: Test an image (replace with your image path)
img_path = '/content/l2.jpeg'  # Replace with your image path
predict_image(img_path)
```

Output:

Image of diseased Leaf

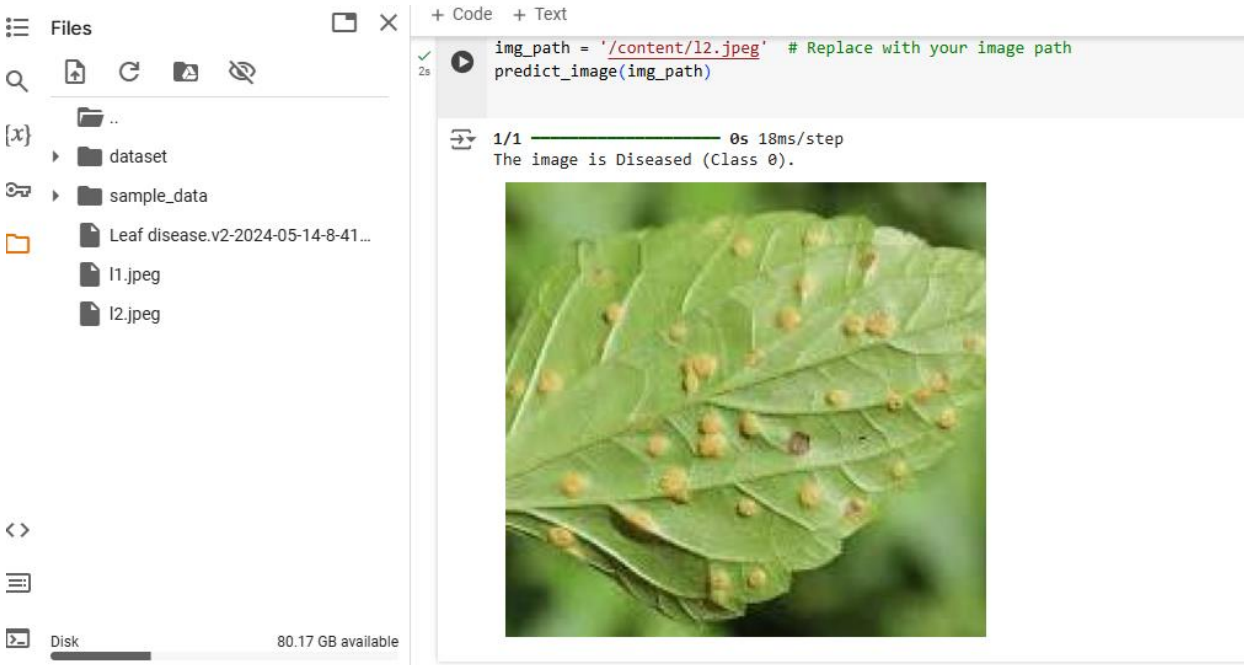

Image of healthy leaf