SMART SORTING : TRANSFER LEARNING FOR IDENTIFYING ROTTEN FRUITS AND VEGETABLES

DATE	28-02-2026
TEAM ID	LTVIP2026TMIDS90622
PROJECT NAME	SMART SORTING : TRANSFER LEARNING FOR IDENTIFYING ROTTEN FRUITS AND VEGETABLES
MAXIMUM MARKS	4 MARKS

6.4 MODEL BUILDING

Split Data and Model Building

Train-Test-Split:

In this project, we have already separated data for training and testing.

```
trainpath = "/content/output_dataset/train"
testpath= "/content/output_dataset/test"

train_datagen = ImageDataGenerator(rescale = 1./255,zoom_range= 0.2,shear_range= 0.2)
test_datagen = ImageDataGenerator(rescale = 1./255)

train = train_datagen.flow_from_directory(trainpath,target_size =(224,224),batch_size = 20)
test = test_datagen.flow_from_directory(testpath,target_size =(224,224),batch_size = 20) ,#5 ,15 , 32, 50

Found 3358 images belonging to 28 classes.
Found 1120 images belonging to 28 classes.
```

Model Building:

Vgg16 Transfer-Learning Model:

The VGG16-based neural network is created using a pre-trained VGG16 architecture with frozen weights. The model is built sequentially, incorporating the VGG16 base, a flattening layer, dropout for regularization, and a dense layer with SoftMax activation for classification into five categories. The model is compiled using the Adam optimizer and sparse categorical cross-entropy loss. During training, which spans 10 epochs, a generator is employed for the training data, and validation is conducted, incorporating call-backs such as Model Checkpoint and Early Stopping. The best-performing model is saved as "healthy_vs_rotten.h5 " for potential future use. The model summary provides an overview of the architecture, showcasing the layers and parameters involved.

```
from tensorflow.keras.applications.vgg16 import VGG16
from tensorflow.keras.layers import Dense, Flatten
from tensorflow.keras.models import Model
```

```
vgg = VGG16(include_top = False,input_shape = (224,224,3))
```

```
for layer in vgg.layers:
  print(layer)
```

```
<keras.src.engine.input_layer.InputLayer object at 0x79c096fde230>
<keras.src.layers.convolutional.conv2d.Conv2D object at 0x79c096fde4d0>
<keras.src.layers.convolutional.conv2d.Conv2D object at 0x79c0081b7a90>
<keras.src.layers.pooling.max_pooling2d.MaxPooling2D object at 0x79bff7ef2f80>
<keras.src.layers.convolutional.conv2d.Conv2D object at 0x79c094405810>
<keras.src.layers.convolutional.conv2d.Conv2D object at 0x79c00834ba30>
<keras.src.layers.pooling.max_pooling2d.MaxPooling2D object at 0x79bff6dad540>
<keras.src.layers.convolutional.conv2d.Conv2D object at 0x79c096fd2c20>
<keras.src.layers.convolutional.conv2d.Conv2D object at 0x79c094405360>
<keras.src.layers.convolutional.conv2d.Conv2D object at 0x79c094405db0>
<keras.src.layers.pooling.max_pooling2d.MaxPooling2D object at 0x79bfcc0fc490>
<keras.src.layers.convolutional.conv2d.Conv2D object at 0x79bff6dae7d0>
<keras.src.layers.convolutional.conv2d.Conv2D object at 0x79bff6dad4b0>
<keras.src.layers.convolutional.conv2d.Conv2D object at 0x79bff6dae020>
<keras.src.layers.pooling.max_pooling2d.MaxPooling2D object at 0x79bfcc0fff10>
<keras.src.layers.convolutional.conv2d.Conv2D object at 0x79bfcc0fe0b0>
<keras.src.layers.convolutional.conv2d.Conv2D object at 0x79bfcc0fe770>
<keras.src.layers.convolutional.conv2d.Conv2D object at 0x79bfcc0fd300>
<keras.src.layers.pooling.max_pooling2d.MaxPooling2D object at 0x79bfcc0fe650>
```

```
len(vgg.layers)
```

```
19
```

```
for layer in vgg.layers:
  layer.trainable = False
```

```
x= Flatten()(vgg.output)
```

```
output = Dense(28, activation ='softmax')(x)
```

```
vgg16 = Model(vgg.input,output)
```

```
vgg16.summary()
```

```
Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_1 (InputLayer)        [(None, 224, 224, 3)]     0

 block1_conv1 (Conv2D)       (None, 224, 224, 64)      1792

 block1_conv2 (Conv2D)       (None, 224, 224, 64)      36928

 block1_pool (MaxPooling2D)  (None, 112, 112, 64)      0

 block2_conv1 (Conv2D)       (None, 112, 112, 128)     73856

 block2_conv2 (Conv2D)       (None, 112, 112, 128)     147584

 block2_pool (MaxPooling2D)  (None, 56, 56, 128)       0

 block3_conv1 (Conv2D)       (None, 56, 56, 256)       295168

 block3_conv2 (Conv2D)       (None, 56, 56, 256)       590080

 block3_conv3 (Conv2D)       (None, 56, 56, 256)       590080
```

```
from keras.callbacks import EarlyStopping
from keras.optimizers import Adam
opt = Adam(lr=0.0001)

# Assuming you have defined your VGG16 model as vgg16

# Define Early Stopping callback
early_stopping = EarlyStopping(monitor='val_accuracy', patience=3, restore_best_weights=True)

# Compile the model (you may have already done this)
vgg16.compile(optimizer = "Adam" , loss='categorical_crossentropy', metrics=['accuracy'])


# # Train the model with early stopping callback
history = vgg16.fit(train, validation_data=test,
                    epochs=15,
                   steps_per_epoch=20,
                  callbacks=[early_stopping])
```

```
WARNING:absl:`lr` is deprecated in Keras optimizer, please use `learning_rate` or use the legacy optimizer, e.g.,tf.keras.optimizers.legacy.Adam.
Epoch 1/15
20/20 [==============================] - 28s 1s/step - loss: 1.4851 - accuracy: 0.6100 - val_loss: 1.3616 - val_accuracy: 0.6390
Epoch 2/15
20/20 [==============================] - 25s 1s/step - loss: 1.2048 - accuracy: 0.6750 - val_loss: 1.4731 - val_accuracy: 0.6336
Epoch 3/15
20/20 [==============================] - 31s 2s/step - loss: 1.1071 - accuracy: 0.7075 - val_loss: 1.6028 - val_accuracy: 0.6023
Epoch 4/15
20/20 [==============================] - 25s 1s/step - loss: 0.8216 - accuracy: 0.7675 - val_loss: 1.1175 - val_accuracy: 0.6979
Epoch 5/15
20/20 [==============================] - 26s 1s/step - loss: 0.7029 - accuracy: 0.7875 - val_loss: 1.2511 - val_accuracy: 0.6836
Epoch 6/15
20/20 [==============================] - 26s 1s/step - loss: 0.8559 - accuracy: 0.7575 - val_loss: 1.2702 - val_accuracy: 0.6685
Epoch 7/15
20/20 [==============================] - 26s 1s/step - loss: 0.6845 - accuracy: 0.8100 - val_loss: 1.0867 - val_accuracy: 0.7265
Epoch 8/15
20/20 [==============================] - 25s 1s/step - loss: 0.5509 - accuracy: 0.8325 - val_loss: 1.2089 - val_accuracy: 0.7069
Epoch 9/15
20/20 [==============================] - 27s 1s/step - loss: 0.5796 - accuracy: 0.8400 - val_loss: 0.8013 - val_accuracy: 0.7802
Epoch 10/15
20/20 [==============================] - 25s 1s/step - loss: 0.4136 - accuracy: 0.8850 - val_loss: 0.9262 - val_accuracy: 0.7560
Epoch 11/15
20/20 [==============================] - 26s 1s/step - loss: 0.4959 - accuracy: 0.8575 - val_loss: 1.0332 - val_accuracy: 0.7292
Epoch 12/15
20/20 [==============================] - 27s 1s/step - loss: 0.5788 - accuracy: 0.8300 - val_loss: 0.8914 - val_accuracy: 0.7587
```

```
vgg16.save('healthy_vs_rotten')
```

Now, Our transfer learning model “healthy_vs_rotten.h5” is saved .

APP.PY CODE:

```
from sklearn.model_selection import train_test_split  # Fixed import
statement
import os
import shutil
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from IPython.display import Image, display
import random
from sklearn.model_selection import train_test_split
from tensorflow.keras.applications.vgg16 import VGG16
from tensorflow.keras.layers import Dense, Flatten
from tensorflow.keras.models import Model
```

```
from keras.optimizers import Adam
from tensorflow.keras.applications.resnet50 import ResNet50
from tensorflow.keras.layers import Dense, Flatten
from tensorflow.keras.models import Model
from keras.preprocessing import image
from keras.applications.vgg16 import preprocess_input
from tensorflow.keras.preprocessing.image import load_img, img_to_array
import numpy as np
from sklearn.model_selection import train_test_split
from keras.callbacks import EarlyStopping
from keras.optimizers import Adam


# Set the path to the dataset - Update this path to the correct location of your dataset
dataset_dir = 'C:/Users/tejas/PycharmProjects/PythonProject/content/Fruit And Vegetable Diseases Dataset'  # CHANGE THIS to your actual dataset path with full absolute path

# Check if the directory exists before proceeding
if not os.path.exists(dataset_dir):
    raise FileNotFoundError(f"The dataset directory '{dataset_dir}' does not exist. Please check the path.")

# Create directories for train, val, and test sets
output_dir = 'output_dataset'
os.makedirs(output_dir, exist_ok=True)
os.makedirs(os.path.join(output_dir, 'train'), exist_ok=True)
os.makedirs(os.path.join(output_dir, 'val'), exist_ok=True)
os.makedirs(os.path.join(output_dir, 'test'), exist_ok=True)

classes = os.listdir(dataset_dir)

for cls in classes:
    os.makedirs(os.path.join(output_dir, 'train', cls), exist_ok=True)
    os.makedirs(os.path.join(output_dir, 'val', cls), exist_ok=True)
    os.makedirs(os.path.join(output_dir, 'test', cls), exist_ok=True)
```

```
    class_dir = os.path.join(dataset_dir, cls)
    images = os.listdir(class_dir)[:200]

    print(cls, len(images))

    train_and_val_images, test_images = train_test_split(
        images, test_size=0.2, random_state=42
    )
    train_images, val_images = train_test_split(
        train_and_val_images, test_size=0.25, random_state=42
    )  # 0.25 x 0.8 = 0.2

    # Copy images to respective directories
    for img in train_images:
        shutil.copy(
            os.path.join(class_dir, img),
            os.path.join(output_dir, 'train', cls, img)
        )
    for img in val_images:
        shutil.copy(
            os.path.join(class_dir, img),
            os.path.join(output_dir, 'val', cls, img)
        )
    for img in test_images:
        shutil.copy(
            os.path.join(class_dir, img),
            os.path.join(output_dir, 'test', cls, img)
        )

print("Dataset split into training, validation, and test sets.")
# Define directories
dataset_dir = 'output_dataset'
train_dir = os.path.join(dataset_dir, 'train')
val_dir = os.path.join(dataset_dir, 'val')
test_dir = os.path.join(dataset_dir, 'test')

# Define image size expected by the pre-trained model
IMG_SIZE = (224, 224)  # Common size for many models like ResNet, VGG,
```

MobileNet

```
# Create ImageDataGenerators for resizing and augmenting the images
train_datagen = ImageDataGenerator(
    rescale=1./255,
    rotation_range=20,
    width_shift_range=0.2,
    height_shift_range=0.2,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True,
    fill_mode='nearest'
)

val_test_datagen = ImageDataGenerator(rescale=1./255)

# Load and resize the images from directories
train_generator = train_datagen.flow_from_directory(
    train_dir,
    target_size=IMG_SIZE,
    batch_size=32,
    class_mode='binary'  # Assuming binary classification for healthy
vs rotten
)

val_generator = val_test_datagen.flow_from_directory(
    val_dir,
    target_size=IMG_SIZE,
    batch_size=32,
    class_mode='binary'
)

test_generator = val_test_datagen.flow_from_directory(
    test_dir,
    target_size=IMG_SIZE,
    batch_size=32,
    class_mode='binary',
    shuffle=False  # Do not shuffle test data
)
```

```
# Print class indices for reference
print(train_generator.class_indices)
print(val_generator.class_indices)
print(test_generator.class_indices)
# Specify the path to your image folder
folder_path = 'output_dataset/train/Apple__Healthy'   # Replace with
the actual path to your image folder

# List all files in the folder
image_files   =   [f   for   f   in   os.listdir(folder_path)   if
f.endswith(('.jpg', '.png', '.jpeg'))]

# Select a random image from the list
selected_image = random.choice(image_files)

# Display the randomly selected image
image_path = os.path.join(folder_path, selected_image)
display(Image(filename=image_path))
# Specify the path to your image folder
folder_path = 'output_dataset/test/Strawberry__Healthy'   # Replace
with the actual path to your image folder

# List all files in the folder
image_files   =   [f   for   f   in   os.listdir(folder_path)   if
f.endswith(('.jpg', '.png', '.jpeg'))]

# Select a random image from the list
selected_image = random.choice(image_files)

# Display the randomly selected image
image_path = os.path.join(folder_path, selected_image)
display(Image(filename=image_path))
# Specify the path to your image folder
folder_path = 'output_dataset/test/Cucumber__Rotten'   # Replace with
the actual path to your image folder

# List all files in the folder
image_files   =   [f   for   f   in   os.listdir(folder_path)   if
```

```
f.endswith(('.jpg', '.png', '.jpeg'))]

# Select a random image from the list
selected_image = random.choice(image_files)

# Display the randomly selected image
image_path = os.path.join(folder_path, selected_image)
display(Image(filename=image_path))
# Specify the path to your image folder
folder_path = 'output_dataset/test/Strawberry__Rotten'  # Replace with
the actual path to your image folder
# List all files in the folder
image_files    =    [f    for    f    in    os.listdir(folder_path)    if
f.endswith(('.jpg', '.png', '.jpeg'))]
# Select a random image from the list
selected_image = random.choice(image_files)

# Display the randomly selected image
image_path = os.path.join(folder_path, selected_image)
display(Image(filename=image_path))
trainpath = "output_dataset/train"
testpath = "output_dataset/test"

train_datagen = ImageDataGenerator(
    rescale=1./255,
    zoom_range=0.2,
    shear_range=0.2
)
test_datagen = ImageDataGenerator(rescale=1./255)

train = train_datagen.flow_from_directory(
    trainpath,
    target_size=(224, 224),
    batch_size=20
)
test = test_datagen.flow_from_directory(
    testpath,
    target_size=(224, 224),
    batch_size=20
```

```
)  # 5, 15, 32, 50
vgg = VGG16(include_top=False, input_shape=(224, 224, 3))

for layer in vgg.layers:
    print(layer)
len(vgg.layers)
for layer in vgg.layers:
    layer.trainable = False
x = Flatten()(vgg.output)
output = Dense(30 , activation='softmax')(x)
vgg16 = Model(vgg.input, output)
vgg16.summary()
opt = Adam(learning_rate=0.0001)
# Assuming you have defined your VGG16 model as vgg16
# Define Early Stopping callback
early_stopping = EarlyStopping(
    monitor='val_accuracy',
    patience=3,
    restore_best_weights=True
)
# Compile the model (you may have already done this)
vgg16.compile(
    optimizer="Adam",
    loss='categorical_crossentropy',
    metrics=['accuracy']
)

# Train the model with early stopping callback
history = vgg16.fit(
    train,
    validation_data=test,
    epochs=15,
    steps_per_epoch=20,
    callbacks=[early_stopping]
)
vgg16.save('healthy_vs_rotten.h5')
```