Experiment no :- 6 Object detection using Transfer Learning of CNN architectures.

Load in a pre-trained CNN model trained on a large dataset

In [1]:

```
import tensorflow as tf
from tensorflow.keras.applications import VGG16
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Dense, Flatten
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.preprocessing.image import ImageDataGenerator, load_img
, img_to_array
import numpy as np
import matplotlib.pyplot as plt
from tensorflow.keras.callbacks import EarlyStopping
```

In [2]:

```
dataset_dir = r"C:\Users\ADMIN\Downloads\archive (6)"
```

In [3]:

```
dataset_datagen = ImageDataGenerator(rescale=1.0 / 255)
```

In [4]:

```
batch_size = 64 # Reduced batch size
```

In [6]:

```
dataset_generator = dataset_datagen.flow_from_directory(
 dataset_dir,
 target_size=(64, 64),
 batch_size=batch_size,
 class_mode='categorical'
)
```

```
Found 9144 images belonging to 1 classes.
```

In [7]:

```
x_train, y_train = dataset_generator[0]
x_test, y_test = dataset_generator[1]
```

In [8]:

```
base_model = VGG16(weights='imagenet', include_top=False, input_shape=(64, 64
, 3))
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applic
ations/vgg16/vgg16_weights_tf_dim_ordering_tf_kernels_notop.h5 (https://stor
age.googleapis.com/tensorflow/keras-applications/vgg16/vgg16_weights_tf_dim_
ordering_tf_kernels_notop.h5)
58889256/58889256 ━━━━━━━━━━━━━━━━━━━━ 9s 0us/step
```

Freeze parameters (weights) in the model's lower convolutional layers

In [9]:

```
# Step b: Freeze lower layers of the model
for layer in base_model.layers:
 layer.trainable = False
```

Add a custom classifier with several layers of trainable parameters to model

In [12]:

```
# Step c: Add custom classifier on top
x = Flatten()(base_model.output)
x = Dense(64, activation='relu')(x)
predictions = Dense(len(dataset_generator.class_indices), activation='softmax')(x)  # Use t
# Create and compile the model
model = Model(inputs=base_model.input, outputs=predictions)
model.compile(optimizer="adam", loss='categorical_crossentropy', metrics=['accuracy'])
# Step d: Implement early stopping
early_stopping = EarlyStopping(monitor='val_loss', patience=3, restore_best_weights=True)
```

Train classifier layers on training data available for the task

In [14]:

```python
# Train the model and store history
history = model.fit(x_train, y_train, batch_size=batch_size, epochs=5, validation_data=(x_t
```

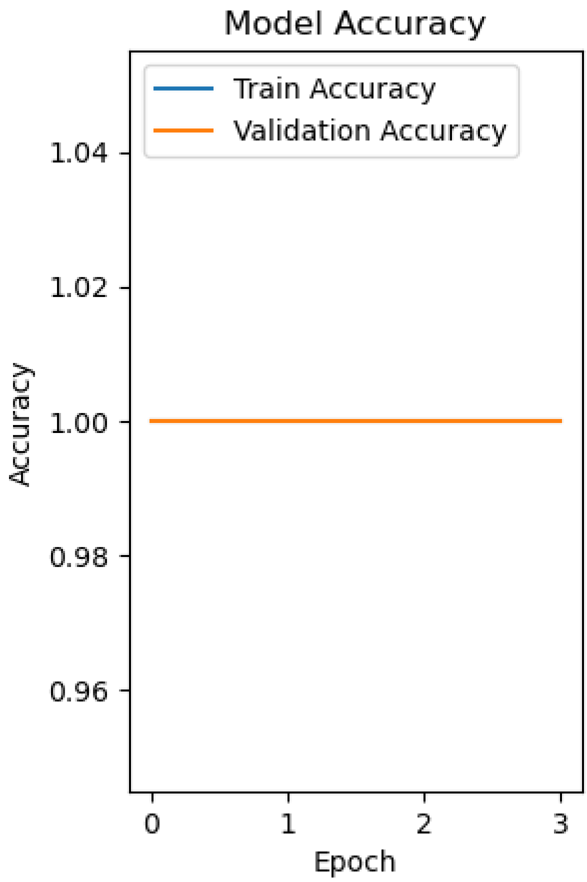
```
Epoch 1/5

C:\Users\ADMIN\anaconda3\Lib\site-packages\keras\src\ops\nn.py:545: UserWarn
ing: You are using a softmax over axis -1 of a tensor of shape (64, 1). This
axis has size 1. The softmax operation will always return the value 1, which
is likely not what you intended. Did you mean to use a sigmoid instead?
  warnings.warn(
C:\Users\ADMIN\anaconda3\Lib\site-packages\keras\src\losses\losses.py:27: Sy
ntaxWarning: In loss categorical_crossentropy, expected y_pred.shape to be
(batch_size, num_classes) with num_classes > 1. Received: y_pred.shape=(64,
1). Consider using 'binary_crossentropy' if you only have 2 classes.
  return self.fn(y_true, y_pred, **self._fn_kwargs)

1/1 ━━━━━━━━━━━━━━━━━━━━ 2s 2s/step - accuracy: 1.0000 - loss: 0.0000e+00 -
 val_accuracy: 1.0000 - val_loss: 0.0000e+00
Epoch 2/5
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 580ms/step - accuracy: 1.0000 - loss: 0.0000e+00
- val_accuracy: 1.0000 - val_loss: 0.0000e+00
Epoch 3/5
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 599ms/step - accuracy: 1.0000 - loss: 0.0000e+00
- val_accuracy: 1.0000 - val_loss: 0.0000e+00
Epoch 4/5
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 563ms/step - accuracy: 1.0000 - loss: 0.0000e+00
- val_accuracy: 1.0000 - val_loss: 0.0000e+00
```

In [19]:

```
# Plot training & validation accuracy
plt.subplot(1, 2, 1)
plt.plot(history.history['accuracy'], label='Train Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.title('Model Accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(loc='upper left')
```

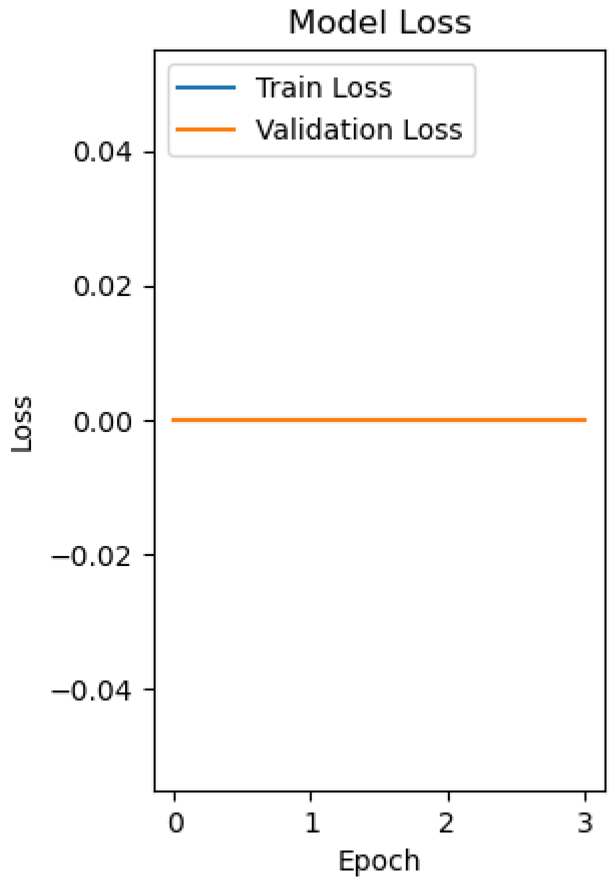
Out[19]:

```
<matplotlib.legend.Legend at 0x241eebebb50>
```



In [20]:

```
# Plot training & validation Loss
plt.subplot(1, 2, 2)
plt.plot(history.history['loss'], label='Train Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.title('Model Loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(loc='upper left')
plt.show()
```



Fine-tune hyperparameters and unfreeze more layers as needed

In [22]:

```
# Step e: Fine-tune hyperparameters and unfreeze more layers if necessary
for layer in base_model.layers[-4:]:
 layer.trainable = True
```

In [23]:

```
# Update the classifier
x = Flatten()(base_model.output)
x = Dense(512, activation='relu')(x)
x = tf.keras.layers.Dropout(0.3)(x)
predictions = Dense(len(dataset_generator.class_indices), activation='softmax')(x)
```

In [25]:

```
# Create and compile the fine-tuned model
model = Model(inputs=base_model.input, outputs=predictions)
model.compile(optimizer=Adam(learning_rate=0.001), loss='categorical_crossentropy', metrics
```

In [34]:

```
# Load and preprocess an external image for prediction
image_path = r'C:\Users\ADMIN\Downloads\archive (6)\caltech-101\strawberry\image_0003.jpg'
img = load_img(image_path, target_size=(64, 64))
```

In [35]:

```
# Train the fine-tuned model with early stopping
history_fine_tune = model.fit(x_train, y_train, batch_size=batch_size, epochs
=5, validation_data=(x_test, y_test), callbacks=[early_stopping])
```

```
Epoch 1/5
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 688ms/step - accuracy: 1.0000 - loss: 0.0000e+00
- val_accuracy: 1.0000 - val_loss: 0.0000e+00
Epoch 2/5
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 685ms/step - accuracy: 1.0000 - loss: 0.0000e+00
- val_accuracy: 1.0000 - val_loss: 0.0000e+00
Epoch 3/5
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 626ms/step - accuracy: 1.0000 - loss: 0.0000e+00
- val_accuracy: 1.0000 - val_loss: 0.0000e+00
```

In [36]:

```
# Visualize the image and the prediction
plt.imshow(img)
#plt.title(f"Predicted: {predicted_class}")
plt.axis('off')
plt.show()
```

In [ ]:

```

```