**PRACTICAL 6**

**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

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

*# Step a: Define directory for the dataset* dataset\_dir = r"C:\Users\Samruddhi\Downloads\archive (6)" *# Use raw string*

*# Load first batch as training and second as testing data* x\_train, y\_train = dataset\_generator[0] x\_test, y\_test = dataset\_generator[1]

*# Step a: Load pre-trained VGG16 model without top layers*

base\_model = VGG16(weights='imagenet', include\_top=False, input\_shape=(64, 64 , 3))

1. **Freeze parameters (weights) in the model’s lower**

**convolutional layers**

*# Step b: Freeze lower layers of the model* **for** layer **in** base\_model.layers: layer.trainable = False

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

*# 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 the number of classes*

*# Create and compile the model*

model = Model(inputs=base\_model.input, outputs=predictions)

model.compile(optimizer="adam", loss='categorical\_crossentropy', metrics=['ac curacy'])

*# Step d: Implement early stopping*

early\_stopping = EarlyStopping(monitor='val\_loss', patience=3, restore\_best\_w eights=True)

**d. Train classifier layers on training data available for the task**

*# Train the model and store history*

history = model.fit(x\_train, y\_train, batch\_size=batch\_size, epochs=5, valida tion\_data=(x\_test, y\_test), callbacks=[early\_stopping])

Epoch 1/5

UserWarning: You are using a softmax over axis -1 of a tensor of shape (64, 1). This axis has size 1. The softmax operation will a

1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step - accuracy: 1.0000 - loss: 0.0000e+00 - v

al\_accuracy: 1.0000 - val\_loss: 0.0000e+00

Epoch 2/5

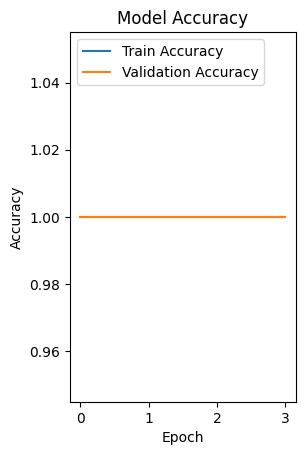
al\_accuracy: 1.0000 - val\_loss: 0.0000e+00

>

<Figure size 1200x400 with 0 Axes>

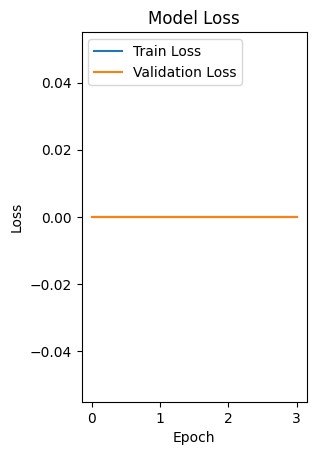
*# Plot training & validation accuracy* plt.subplot(1, 2, 1)

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')



*# 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')



**e. Fine-tune hyperparameters and unfreeze more layers as needed**

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

*# 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)

*# 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 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step - accuracy: 1.0000 - loss: 0.0000e+00 - v

al\_accuracy: 1.0000 - val\_loss: 0.0000e+00

al\_accuracy: 1.0000 - val\_loss: 0.0000e+00

*# Load and preprocess an external image for prediction*

image\_path = r'C:\Users\Samruddhi\Downloads\archive (6)\caltech-101\strawberr y\image\_0021.jpg' *# Replace with the actual image path*

1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 223ms/step

*# Visualize the image and the prediction* plt.imshow(img)

plt.title(f"Predicted: {predicted\_class}") plt.axis('off') plt.show()

