1. Connecting Drive

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mour
```

2. Dataset Paths:

```
train_dir = "/content/drive/MyDrive/AI&ML-Level6/Week - 5/Week - 5 - Image Classification wi
test_dir = "/content/drive/MyDrive/AI&ML-Level6/Week - 5/Week - 5 - Image Classification wit
```

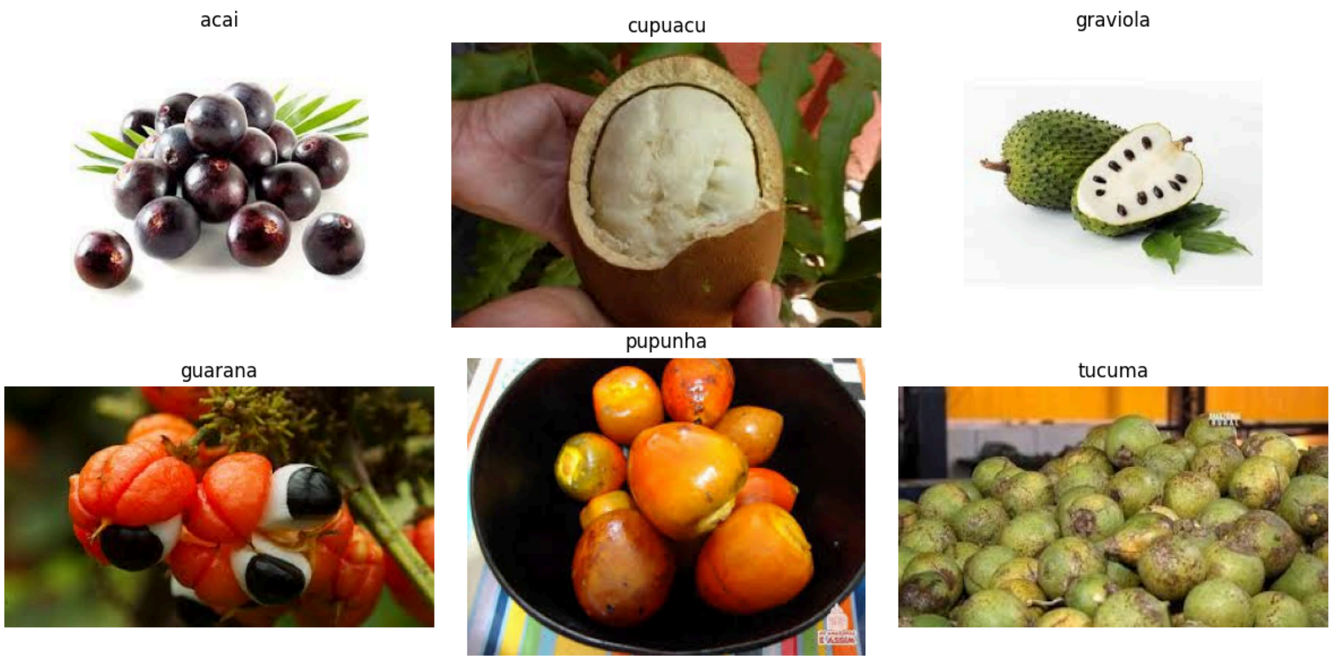
Task 1: Improve the Model from Worksheet 5

Step 1: Data Understanding and Preprocessing

```
import os
from PIL import Image, UnidentifiedImageError
import matplotlib.pyplot as plt
import random

# Define dataset paths
train_dir = "/content/drive/MyDrive/AI&ML-Level6/Week - 5/Week - 5 - Image Classification wi
test_dir = "/content/drive/MyDrive/AI&ML-Level6/Week - 5/Week - 5 - Image Classification wit

# Get class names from train directory
class_names = sorted(os.listdir(train_dir))
if not class_names:
    print("No class directories found in the train folder!")
else:
    print(f"Found {len(class_names)} classes: {class_names}")

# Check for corrupted images in train directory
corrupted_images = []
for class_name in class_names:
    class_path = os.path.join(train_dir, class_name)
    if os.path.isdir(class_path):
        images = os.listdir(class_path)
        for img_name in images:
            img_path = os.path.join(class_path, img_name)
            try:
                with Image.open(img_path) as img:
                    img.verify()
            except (IOError, UnidentifiedImageError):
```

```
                corrupted_images.append(img_path)

    if corrupted_images:
        print("\nCorrupted Images Found: ")
        for img in corrupted_images:
            print(img)
    else:
        print("\nNo corrupted images found.")

    # Check class balance
    class_counts = {}
    for class_name in class_names:
        class_path = os.path.join(train_dir, class_name)
        if os.path.isdir(class_path):
            images = [img for img in os.listdir(class_path) if img.lower().endswith(('.png', '.j
            class_counts[class_name] = len(images)
    print("\nClass Distribution: ")
    print("=" * 45)
    for class_name, count in class_counts.items():
        print(f"{class_name}: {count}")
    print("=" * 45)

    # Visualize random images
    selected_images, selected_labels = [], []
    for class_name in class_names:
        class_path = os.path.join(train_dir, class_name)
        images = [img for img in os.listdir(class_path) if img.lower().endswith(('.png', '.jpg',
        if images:
            selected_img = os.path.join(class_path, random.choice(images))
            selected_images.append(selected_img)
            selected_labels.append(class_name)

    # Plot random images
    cols = (len(selected_images) + 1) // 2
    fig, axes = plt.subplots(2, cols, figsize=(12, 6))
    for i, ax in enumerate(axes.flat):
        if i < len(selected_images):
            img = plt.imread(selected_images[i])
            ax.imshow(img)
            ax.set_title(selected_labels[i])
            ax.axis("off")
        else:
            ax.axis("off")
    plt.tight_layout()
    plt.show()
```

```
Found 6 classes: ['acai', 'cupuacu', 'graviola', 'guarana', 'pupunha', 'tucuma']

No corrupted images found.

Class Distribution:
=============================================
acai: 15
cupuacu: 15
graviola: 15
guarana: 15
pupunha: 15
tucuma: 15
=============================================
```

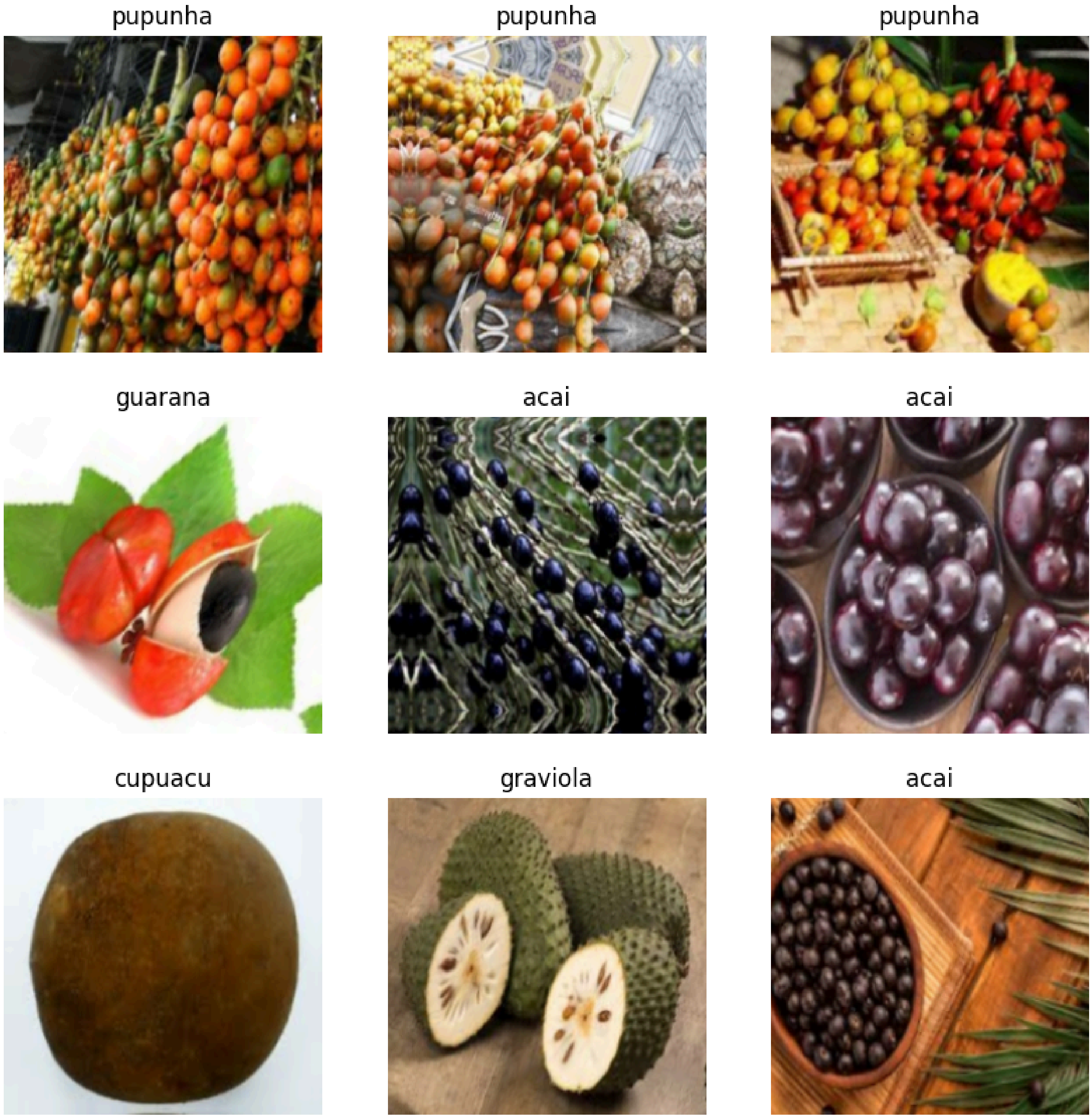


Step 2: Data Generation and Augmentation

```
import tensorflow as tf
from tensorflow.keras import layers
```

```
# Define image size and batch size
image_size = (224, 224)  # Matches VGG16 input size
batch_size = 32

# Load train dataset and split into train/validation
train_ds, val_ds = tf.keras.utils.image_dataset_from_directory(
    train_dir,
    validation_split=0.2,
    subset="both",
    seed=1337,
    image_size=image_size,
    batch_size=batch_size,
    label_mode='int'
)

# Define data augmentation layers
data_augmentation_layers = [
    layers.RandomFlip("horizontal"),
    layers.RandomRotation(0.1),
    layers.RandomZoom(0.2),
]

def data_augmentation(images):
    for layer in data_augmentation_layers:
        images = layer(images)
    return images

# Apply augmentation and rescaling
augmented_train_ds = train_ds.map(lambda x, y: (data_augmentation(x), y))
train_ds = augmented_train_ds.map(lambda x, y: (layers.Rescaling(1./255)(x), y))
val_ds = val_ds.map(lambda x, y: (layers.Rescaling(1./255)(x), y))

# Visualize augmented images
plt.figure(figsize=(10, 10))
for images, labels in train_ds.take(1):
    for i in range(9):
        ax = plt.subplot(3, 3, i + 1)
        plt.imshow(images[i].numpy())
        plt.title(class_names[labels[i]])
        plt.axis("off")
plt.show()
```

```
Found 90 files belonging to 6 classes.
Using 72 files for training.
Using 18 files for validation.
```



Step 3: Build and Train the Enhanced Model

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, BatchNormalizatior

# Define the model
model = Sequential([
    layers.Input(shape=(224, 224, 3)),
    layers.Rescaling(1./255),  # Already applied, but included for consistency
    # Convolutional Block 1
    Conv2D(32, (3, 3), padding='same'),
    BatchNormalization(),
    Activation('relu'),
    MaxPooling2D((2, 2)),
    Dropout(0.25),
    # Convolutional Block 2
    Conv2D(64, (3, 3), padding='same'),
    BatchNormalization(),
    Activation('relu'),
    MaxPooling2D((2, 2)),
    Dropout(0.25),
    # Convolutional Block 3
    Conv2D(128, (3, 3), padding='same'),
    BatchNormalization(),
    Activation('relu'),
    MaxPooling2D((2, 2)),
    Dropout(0.25),
    # Flatten and Dense Layers
    Flatten(),
    Dense(512),
    BatchNormalization(),
    Activation('relu'),
    Dropout(0.5),
    Dense(len(class_names), activation='softmax')  # Number of classes dynamically set
])

# Compile the model
model.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accuracy']

# Model summary
model.summary()

# Train the model
history = model.fit(train_ds, epochs=10, validation_data=val_ds)

# Evaluate on validation set
test_loss, test_acc = model.evaluate(val_ds)
print(f"Validation accuracy: {test_acc:.4f}")

# Plot training behavior
plt.plot(history.history['accuracy'], label='Training Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.title('Model Accuracy')
```

```
    plt.xlabel('Epoch')
    plt.ylabel('Accuracy')
    plt.legend()
    plt.show()
```

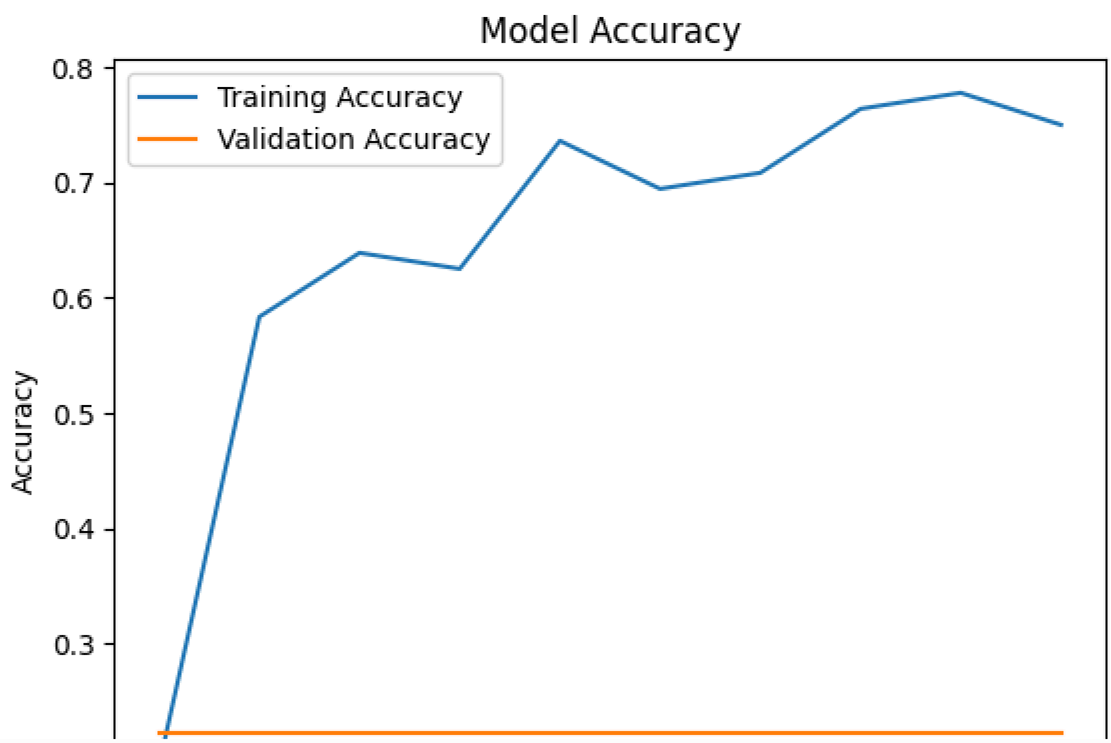
Model: "sequential_1"

Layer (type)	Output Shape	Param
rescaling_3 (Rescaling)	(None, 224, 224, 3)	
conv2d_2 (Conv2D)	(None, 224, 224, 32)	89
batch_normalization (BatchNormalization)	(None, 224, 224, 32)	12
activation (Activation)	(None, 224, 224, 32)	
max_pooling2d_2 (MaxPooling2D)	(None, 112, 112, 32)	
dropout (Dropout)	(None, 112, 112, 32)	
conv2d_3 (Conv2D)	(None, 112, 112, 64)	18,49
batch_normalization_1 (BatchNormalization)	(None, 112, 112, 64)	25
activation_1 (Activation)	(None, 112, 112, 64)	
max_pooling2d_3 (MaxPooling2D)	(None, 56, 56, 64)	
dropout_1 (Dropout)	(None, 56, 56, 64)	
conv2d_4 (Conv2D)	(None, 56, 56, 128)	73,85
batch_normalization_2 (BatchNormalization)	(None, 56, 56, 128)	51
activation_2 (Activation)	(None, 56, 56, 128)	
max_pooling2d_4 (MaxPooling2D)	(None, 28, 28, 128)	
dropout_2 (Dropout)	(None, 28, 28, 128)	
flatten_1 (Flatten)	(None, 100352)	
dense_3 (Dense)	(None, 512)	51,380,73
batch_normalization_3 (BatchNormalization)	(None, 512)	2,04
activation_3 (Activation)	(None, 512)	
dropout_3 (Dropout)	(None, 512)	
dense_4 (Dense)	(None, 6)	3,07

Total params: 51,480,006 (196.38 MB)
Trainable params: 51,478,534 (196.38 MB)
Non-trainable params: 1,472 (5.75 KB)

Epoch 1/10

```
3/3 ━━━━━━━━━━━━━━━━━━━━ 24s 5s/step - accuracy: 0.1832 - loss: 3.0144 - val_accuracy
Epoch 2/10
3/3 ━━━━━━━━━━━━━━━━━━━━ 19s 6s/step - accuracy: 0.5768 - loss: 1.6180 - val_accuracy
Epoch 3/10
3/3 ━━━━━━━━━━━━━━━━━━━━ 19s 5s/step - accuracy: 0.6515 - loss: 1.0613 - val_accuracy
Epoch 4/10
3/3 ━━━━━━━━━━━━━━━━━━━━ 19s 5s/step - accuracy: 0.6523 - loss: 0.9527 - val_accuracy
Epoch 5/10
3/3 ━━━━━━━━━━━━━━━━━━━━ 19s 5s/step - accuracy: 0.7313 - loss: 0.7199 - val_accuracy
Epoch 6/10
3/3 ━━━━━━━━━━━━━━━━━━━━ 20s 5s/step - accuracy: 0.6988 - loss: 0.6796 - val_accuracy
Epoch 7/10
3/3 ━━━━━━━━━━━━━━━━━━━━ 20s 5s/step - accuracy: 0.6940 - loss: 0.8836 - val_accuracy
Epoch 8/10
3/3 ━━━━━━━━━━━━━━━━━━━━ 18s 6s/step - accuracy: 0.7530 - loss: 0.5451 - val_accuracy
Epoch 9/10
3/3 ━━━━━━━━━━━━━━━━━━━━ 19s 6s/step - accuracy: 0.7405 - loss: 0.6835 - val_accuracy
Epoch 10/10
3/3 ━━━━━━━━━━━━━━━━━━━━ 21s 5s/step - accuracy: 0.7578 - loss: 0.6711 - val_accuracy
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 739ms/step - accuracy: 0.2222 - loss: 3.1384
Validation accuracy: 0.2222
```



Step 4: Analyze Results

```
model.summary()
```

Model: "sequential_1"

Layer (type)	Output Shape	Param #
rescaling_3 (Rescaling)	(None, 224, 224, 3)	0
conv2d_2 (Conv2D)	(None, 224, 224, 32)	896
batch_normalization (BatchNormalization)	(None, 224, 224, 32)	128
activation (Activation)	(None, 224, 224, 32)	0
max_pooling2d_2 (MaxPooling2D)	(None, 112, 112, 32)	0
dropout (Dropout)	(None, 112, 112, 32)	0
conv2d_3 (Conv2D)	(None, 112, 112, 64)	18,496
batch_normalization_1 (BatchNormalization)	(None, 112, 112, 64)	256
activation_1 (Activation)	(None, 112, 112, 64)	0
max_pooling2d_3 (MaxPooling2D)	(None, 56, 56, 64)	0
dropout_1 (Dropout)	(None, 56, 56, 64)	0
conv2d_4 (Conv2D)	(None, 56, 56, 128)	73,856
batch_normalization_2 (BatchNormalization)	(None, 56, 56, 128)	512
activation_2 (Activation)	(None, 56, 56, 128)	0
max_pooling2d_4 (MaxPooling2D)	(None, 28, 28, 128)	0
dropout_2 (Dropout)	(None, 28, 28, 128)	0
flatten_1 (Flatten)	(None, 100352)	0
dense_3 (Dense)	(None, 512)	51,380,736
batch_normalization_3 (BatchNormalization)	(None, 512)	2,048
activation_3 (Activation)	(None, 512)	0
dropout_3 (Dropout)	(None, 512)	0
dense_4 (Dense)	(None, 6)	3,078

Task 2: Image Classification via Fine-Tuning with VGG16

Step 1: Reuse Dataset from Task 1

```
# Load test dataset
test_ds = tf.keras.utils.image_dataset_from_directory(
    test_dir,
    image_size=image_size,
    batch_size=batch_size,
    label_mode='int',
    shuffle=False  # Keep order for evaluation
)
test_ds = test_ds.map(lambda x, y: (layers.Rescaling(1./255)(x), y))
```

```
Found 30 files belonging to 6 classes.
```

Step 2: Load the Pre-trained VGG16 Model

```
from tensorflow.keras.applications import VGG16
from tensorflow.keras.layers import GlobalAveragePooling2D, Dense
from tensorflow.keras.models import Model

# Load VGG16 pre-trained on ImageNet
base_model = VGG16(weights='imagenet', include_top=False, input_shape=(224, 224, 3))
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/vgg16
58889256/58889256 ━━━━━━━━━━━━━━━━━━━━ 0s 0us/step
```

Step 3: Freeze the Base Model Layers

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

Step 4: Add Custom Layers

```
# Add custom layers
x = base_model.output
x = GlobalAveragePooling2D()(x)
x = Dense(1024, activation='relu')(x)
x = Dense(len(class_names), activation='softmax')(x)  # Dynamic number of classes

# Create the final model
model = Model(inputs=base_model.input, outputs=x)
```

Step 5: Compile and Train the Model

```
from tensorflow.keras.optimizers import Adam

# Compile the model
model.compile(optimizer=Adam(), loss='sparse_categorical_crossentropy', metrics=['accuracy']

# Model summary
model.summary()

# Train the model
history = model.fit(train_ds, epochs=10, validation_data=val_ds)

# Evaluate on validation set
val_loss, val_acc = model.evaluate(val_ds)
print(f"Validation accuracy: {val_acc:.4f}")
```