```
import numpy as np  # Importing NumPy for numerical operations and array manipulations
import matplotlib.pyplot as plt  # Importing Matplotlib for plotting graphs and visualizat
import seaborn as sns  # Importing Seaborn for statistical data visualization, built on to
import tensorflow as tf  # Importing TensorFlow for building and training machine learning
from tensorflow import keras  # Importing Keras, a high-level API for TensorFlow, to simpl
from tensorflow.keras import Layer  # Importing Layer class for creating custom layers in
from tensorflow.keras.models import Sequential  # Importing Sequential model for building
from tensorflow.keras.layers import Rescaling , GlobalAveragePooling2D
from tensorflow.keras import layers, optimizers, callbacks  # Importing various modules fo
from sklearn.utils.class_weight import compute_class_weight  # Importing function to compu
from tensorflow.keras.applications import EfficientNetV2B2  # Importing EfficientNetV2S mo
from sklearn.metrics import confusion_matrix, classification_report  # Importing functions
import gradio as gr  # Importing Gradio for creating interactive web interfaces for machin
```

Generated code may be subject to a license | killuhwhale/amace_validator | HabibaOsama1/Dog-vs-Cat-classification-using-CNN

```
# prompt: i have a zip file of dataset so i want to extract and use that in this notebook

import zipfile
import os

zip_file_path = '/content/garbage.zip' # Replace with the actual path to your zip file
extract_dir = '/content/dataset' # Directory to extract the files to

# Create the extraction directory if it doesn't exist
if not os.path.exists(extract_dir):
    os.makedirs(extract_dir)

# Extract the zip file
with zipfile.ZipFile(zip_file_path, 'r') as zip_ref:
    zip_ref.extractall(extract_dir)

print(f"Dataset extracted to: {extract_dir}")
!ls {extract_dir}
```

```
Dataset extracted to: /content/dataset
garbage
```

```
dataset_dir= r"/content/dataset/garbage/TrashType_Image_Dataset"
image_size = (124, 124)
batch_size = 32
seed = 42
```

```
train_ds = tf.keras.utils.image_dataset_from_directory(
    dataset_dir,
    validation_split=0.2,
    subset="training",
```

```
    seed=seed,
    shuffle = True,
    image_size=image_size,
    batch_size=batch_size
)
```

```
Found 18 files belonging to 6 classes.
Using 15 files for training.
```

```
val_ds = tf.keras.utils.image_dataset_from_directory(
    dataset_dir,
    validation_split=0.2,
    subset="validation",
    seed=seed,
    shuffle = True,
    image_size=image_size,
    batch_size=batch_size
)
val_class= val_ds.class_names
```

```
Found 18 files belonging to 6 classes.
Using 3 files for validation.
```

```
# Get the total number of batches in the validation dataset
val_batches = tf.data.experimental.cardinality(val_ds)

# Split the validation dataset into two equal parts:
# First half becomes the test dataset
test_ds = val_ds.take(val_batches // 2)

# Second half remains as the validation dataset
val_dat = val_ds.skip(val_batches // 2)

# Optimize test dataset by caching and prefetching to improve performance
test_ds_eval = test_ds.cache().prefetch(tf.data.AUTOTUNE)
```

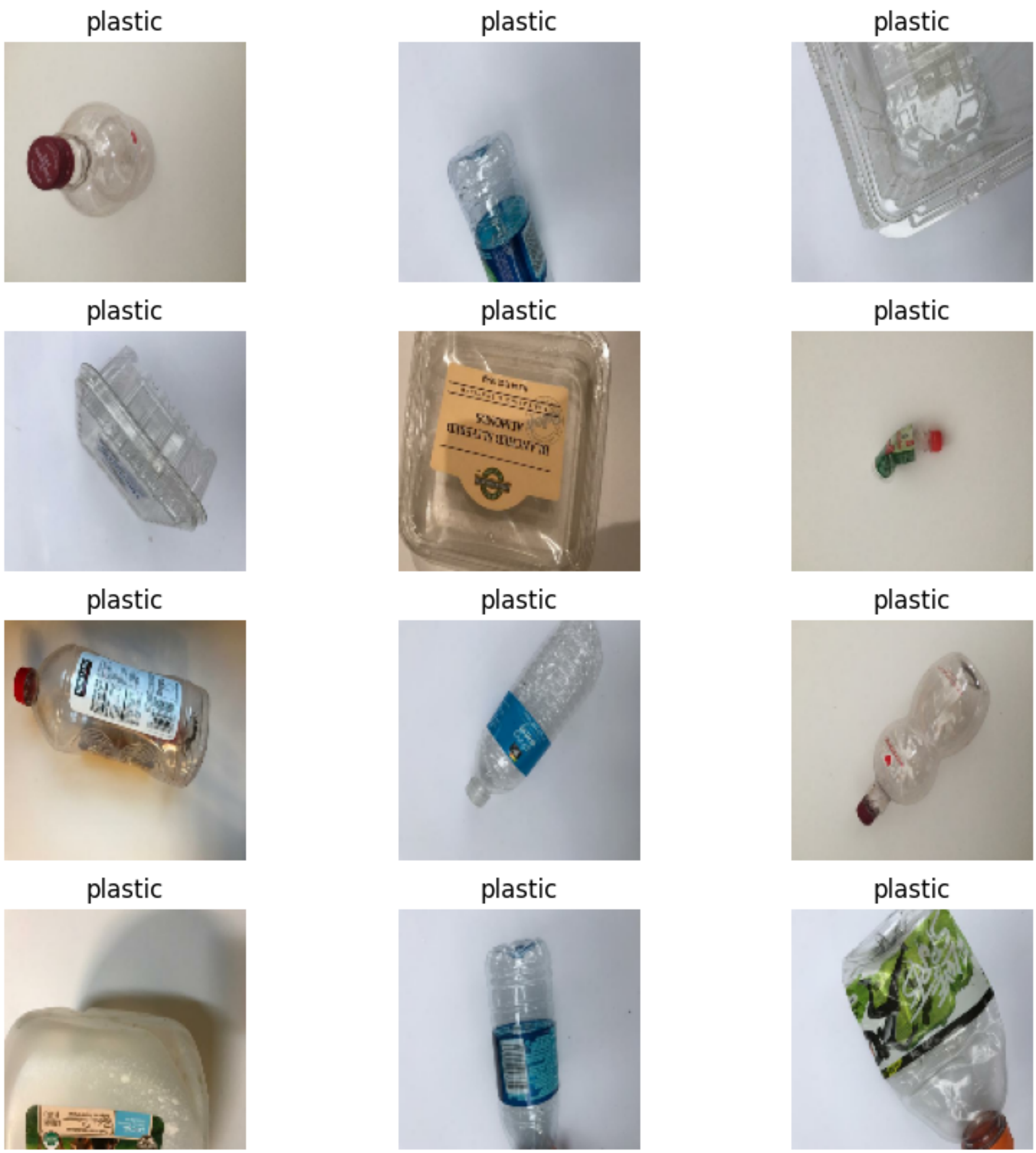
```
print(train_ds.class_names)
print(val_class)
print(len(train_ds.class_names))
```

```
['cardboard', 'glass', 'metal', 'paper', 'plastic', 'trash']
['cardboard', 'glass', 'metal', 'paper', 'plastic', 'trash']
6
```

Visualize sample images from each class.

```
import matplotlib.pyplot as plt

plt.figure(figsize=(10, 10))
for images, labels in train_ds.take(1):
  for i in range(12):
    ax = plt.subplot(4, 3, i + 1)
    plt.imshow(images[i].numpy().astype("uint8"))
    plt.title(train_ds.class_names[labels[i]])
    plt.axis("off")
```



```
def count_distribution(dataset, class_names):
    total = 0
    counts = {name: 0 for name in class_names}

    for _, labels in dataset:
        for label in labels.numpy():
            class_name = class_names[label]
            counts[class_name] += 1
            total += 1

    if total == 0:  # Check if total is zero
        return counts

    for k in counts:
        counts[k] = round((counts[k] / total) * 100, 2)  # Convert to percentage
    return counts


# Function to plot class distribution
def simple_bar_plot(dist, title):
    plt.bar(dist.keys(), dist.values(), color='cornflowerblue')
    plt.title(title)
    plt.ylabel('Percentage (%)')
    plt.xticks(rotation=45)
    plt.ylim(0, 100)
    plt.tight_layout()
    plt.show()



class_names = train_ds.class_names

# Get class distributions
train_dist = count_distribution(train_ds, class_names)
val_dist = count_distribution(val_ds, class_names)
test_dist = count_distribution(test_ds, class_names)
overall_dist = {}
for k in class_names:
    overall_dist[k] = round((train_dist[k] + val_dist[k]) / 2, 2)

print(train_dist)
print(val_dist)
print(test_dist)
print(overall_dist)
```

```
{'cardboard': 0.0, 'glass': 0.0, 'metal': 0.0, 'paper': 0.0, 'plastic': 100.0, 'trash
{'cardboard': 0.0, 'glass': 0.0, 'metal': 0.0, 'paper': 0.0, 'plastic': 100.0, 'trash
```

```
{'cardboard': 0, 'glass': 0, 'metal': 0, 'paper': 0, 'plastic': 0, 'trash': 0}
{'cardboard': 0.0, 'glass': 0.0, 'metal': 0.0, 'paper': 0.0, 'plastic': 100.0, 'trash
```

```
# Show visualizations
simple_bar_plot(train_dist, "Training Set Class Distribution (%)")
simple_bar_plot(val_dist, "Validation Set Class Distribution (%)")
simple_bar_plot(test_dist, "Test Set Class Distribution (%)")
simple_bar_plot(overall_dist, "Overall Class Distribution (%)")
```

```
from sklearn.utils.class_weight import compute_class_weight
import numpy as np

# Count class occurrences and prepare label list
class_counts = {i: 0 for i in range(len(class_names))}
all_labels = []

for images, labels in train_ds:
    for label in labels.numpy():
        class_counts[label] += 1
        all_labels.append(label)

# Compute class weights (index aligned)
class_weights_array = compute_class_weight(
    class_weight='balanced',
    classes=np.unique(all_labels),  # Use unique labels present in all_labels
    y=all_labels
)

# Create dictionary mapping class index to weight
# Map weights back to all class indices, handling classes not in all_labels if necessary
class_weights = {c: w for c, w in zip(np.unique(all_labels), class_weights_array)}

# Ensure all class indices are in class_weights dictionary, setting weight to 0 or 1 for
# A weight of 0 would ignore the class, 1 would treat it normally. Balanced weight for a
# For now, we'll just use the weights for the classes present in all_labels.
# If a class is truly missing from the dataset, its weight is effectively not used in tra



# ✅ Optional: print results
print("Class Counts:", class_counts)
print("Class Weights:", class_weights)
```

```
Class Counts: {0: 0, 1: 0, 2: 0, 3: 0, 4: 15, 5: 0}
Class Weights: {np.int32(4): np.float64(1.0)}
```

```
#  Define data augmentation pipeline
data_augmentation = Sequential([
    layers.RandomFlip("horizontal"),
```

```
    layers.RandomFlip("horizontal"),
    layers.RandomRotation(0.1),
    layers.RandomZoom(0.1),
    layers.RandomContrast(0.1),
])


#  Load the pretrained MobileNetV3Small model (without the top classification layer)
base_model = EfficientNetV2B2(include_top=False, input_shape=(124, 124, 3),include_prepro


#  Freeze early layers (to retain general pretrained features)
base_model.trainable = True
for layer in base_model.layers[:100]:  # You can adjust this number
    layer.trainable = False
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/ef
35839040/35839040 ━━━━━━━━━━━━━━━━━━━━ 0s 0us/step
```

```
#  Build the final model
model = Sequential([
    layers.Input(shape=(124, 124, 3)),
    data_augmentation,
    base_model,
    GlobalAveragePooling2D(),
    layers.Dropout(0.3),
    layers.Dense(6, activation='softmax')  # Change to your number of classes
])



# ⚙️ Compile the model
model.compile(
    optimizer=optimizers.Adam(learning_rate=1e-4),
    loss='sparse_categorical_crossentropy',
    metrics=['accuracy']
)



# Define an EarlyStopping callback to stop training when validation loss stops improving
early = tf.keras.callbacks.EarlyStopping(
    monitor='val_loss',            # Metric to monitor (validation loss here)
    patience=3,                   # Number of epochs to wait after last improvement befor
    restore_best_weights=True     # After stopping, restore the model weights from the ep
)



# 📝 Summary (optional but useful)
```

```
model.summary()
```

```
base_model.summary() # Print the architecture summary of the base model
```

```
# 🏋️ Train the model
epochs = 20 # Define the number of training epochs

history = model.fit(
    train_ds,
    validation_data=val_ds,
    epochs=epochs,
    callbacks=[early],
    class_weight=class_weights # Use the calculated class weights
)
```

```
Epoch 1/20
1/1 ━━━━━━━━━━━━━━━━━━━━ 81s 81s/step - accuracy: 0.0667 - loss: 1.7680 - val_accurac
Epoch 2/20
1/1 ━━━━━━━━━━━━━━━━━━━━ 4s 4s/step - accuracy: 0.2667 - loss: 1.7897 - val_accuracy:
Epoch 3/20
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 1s/step - accuracy: 0.2667 - loss: 1.7984 - val_accuracy:
Epoch 4/20
1/1 ━━━━━━━━━━━━━━━━━━━━ 2s 2s/step - accuracy: 0.2667 - loss: 1.7147 - val_accuracy:
```

```
acc = history.history['accuracy']          # Extract training accuracy from history
val_acc = history.history['val_accuracy']  # Extract validation accuracy from history
loss = history.history['loss']             # Extract training loss from history
val_loss = history.history['val_loss']     # Extract validation loss from history

epochs_range = range(len(acc))             # Define range for epochs based on accuracy le

plt.figure(figsize=(10,8))                 # Set overall figure size for visualization

plt.subplot(1,2,1)                         # Create first subplot (1 row, 2 columns, posi
plt.plot(epochs_range, acc, label='Training Accuracy')       # Plot training accuracy
plt.plot(epochs_range, val_acc, label='Validation Accuracy') # Plot validation accuracy
plt.legend(loc='lower right')              # Place legend in lower-right corner
plt.title('Training vs Validation Accuracy') # Add title for accuracy plot

plt.subplot(1,2,2)                         # Create second subplot (1 row, 2 columns, pos
plt.plot(epochs_range, loss, label='Training Loss')         # Plot training loss
plt.plot(epochs_range, val_loss, label='Validation Loss')   # Plot validation loss
plt.legend(loc='upper right')              # Place legend in upper-right corner
plt.title('Training vs Validation Loss')   # Add title for loss plot

plt.show()                                 # Display the plots
```

```
# Check if the test dataset is empty before evaluation
if tf.data.experimental.cardinality(test_ds_eval).numpy() == 0:
    print("Test dataset is empty. Cannot evaluate the model.")
else:
    loss, accuracy = model.evaluate(test_ds_eval)
    print(f'Test accuracy is {accuracy:.4f}, Test loss is {loss:.4f}')
```

```
Test dataset is empty. Cannot evaluate the model.
```

```
from sklearn.metrics import confusion_matrix, classification_report

# Check if the test dataset is empty before attempting to compute confusion matrix and cl
if tf.data.experimental.cardinality(test_ds_eval).numpy() == 0:
    print("Test dataset is empty. Cannot compute confusion matrix or classification repor
else:
    # Extract true labels from all batches in the test dataset
    y_true = np.concatenate([y.numpy() for x, y in test_ds_eval], axis=0)  # Convert Tens
```

```
    # Get predictions as probabilities from the model
    y_pred_probs = model.predict(test_ds_eval)  # Predict class probabilities for each sa

    # Convert probabilities to predicted class indices
    y_pred = np.argmax(y_pred_probs, axis=1)  # Select the class with the highest probabi

    # Compute the confusion matrix to evaluate classification performance
    cm = confusion_matrix(y_true, y_pred)  # Generate confusion matrix comparing true lab

    # Print metrics to assess model performance
    print(cm)  # Display confusion matrix
    print(classification_report(y_true, y_pred))  # Print precision, recall, and F1-score
```

```
Test dataset is empty. Cannot compute confusion matrix or classification report.
```

```
plt.figure(figsize=(10,8))  # Set figure size for better visualization

sns.heatmap(cm, annot=True, fmt='d',  # Create heatmap using confusion matrix
            xticklabels=class_names,  # Set class names for x-axis (predicted labels)
            yticklabels=class_names,  # Set class names for y-axis (true labels)
            cmap='Blues')  # Use a blue colormap for better contrast

plt.xlabel('Predicted')  # Label x-axis as Predicted classes
plt.ylabel('True')  # Label y-axis as True classes
plt.title('Confusion Matrix')  # Add title to the heatmap
plt.show()  # Display the plot
```

7. Final Testing and Save the Model

- Evaluate the final model on the unseen **test dataset**.

```
# Extract class names from the training dataset
class_names = train_ds.class_names

# Take one batch of images and labels from the test dataset for evaluation
for images, labels in test_ds_eval.take(1):

    # Generate predictions for the batch of images
    predictions = model.predict(images)

    # Get the predicted class index for each image
    pred_labels = tf.argmax(predictions, axis=1)

    # Loop through the first 8 images in the batch
    for i in range(8):
        plt.imshow(images[i].numpy().astype("uint8"))  # Convert and display image
        plt.title(f"True: {class_names[labels[i]]}, Pred: {class_names[pred_labels[i]]}")
        plt.axis("off")  # Hide axes for better visualization
        plt.show()  # Display the image with title
```

Save the trained model using `model.save()` or `save_model()` for future inference.

```
# Save model in Keras format with architecture, weights, and training configuration
model.save('Effiicientnetv2b2.keras')

# Load your Keras model
model = tf.keras.models.load_model('Effiicientnetv2b2.keras')
```

```
!pip install gradio
```

```
Requirement already satisfied: gradio in /usr/local/lib/python3.11/dist-packages (5.3
Requirement already satisfied: aiofiles<25.0,>=22.0 in /usr/local/lib/python3.11/dist
Requirement already satisfied: anyio<5.0,>=3.0 in /usr/local/lib/python3.11/dist-pack
Requirement already satisfied: fastapi<1.0,>=0.115.2 in /usr/local/lib/python3.11/dis
Requirement already satisfied: ffmpy in /usr/local/lib/python3.11/dist-packages (from
Requirement already satisfied: gradio-client==1.10.1 in /usr/local/lib/python3.11/dis
Requirement already satisfied: groovy~=0.1 in /usr/local/lib/python3.11/dist-packages
Requirement already satisfied: httpx>=0.24.1 in /usr/local/lib/python3.11/dist-packag
Requirement already satisfied: huggingface-hub>=0.28.1 in /usr/local/lib/python3.11/d
Requirement already satisfied: jinja2<4.0 in /usr/local/lib/python3.11/dist-packages
Requirement already satisfied: markupsafe<4.0,>=2.0 in /usr/local/lib/python3.11/dist
Requirement already satisfied: numpy<3.0,>=1.0 in /usr/local/lib/python3.11/dist-pack
Requirement already satisfied: orjson~=3.0 in /usr/local/lib/python3.11/dist-packages
Requirement already satisfied: packaging in /usr/local/lib/python3.11/dist-packages (
Requirement already satisfied: pandas<3.0,>=1.0 in /usr/local/lib/python3.11/dist-pac
Requirement already satisfied: pillow<12.0,>=8.0 in /usr/local/lib/python3.11/dist-pa
Requirement already satisfied: pydantic<2.12,>=2.0 in /usr/local/lib/python3.11/dist-
Requirement already satisfied: pydub in /usr/local/lib/python3.11/dist-packages (from
Requirement already satisfied: python-multipart>=0.0.18 in /usr/local/lib/python3.11/
Requirement already satisfied: pyyaml<7.0,>=5.0 in /usr/local/lib/python3.11/dist-pac
Requirement already satisfied: ruff>=0.9.3 in /usr/local/lib/python3.11/dist-packages
Requirement already satisfied: safehttpx<0.2.0,>=0.1.6 in /usr/local/lib/python3.11/d
Requirement already satisfied: semantic-version~=2.0 in /usr/local/lib/python3.11/dis
Requirement already satisfied: starlette<1.0,>=0.40.0 in /usr/local/lib/python3.11/di
Requirement already satisfied: tomlkit<0.14.0,>=0.12.0 in /usr/local/lib/python3.11/d
Requirement already satisfied: typer<1.0,>=0.12 in /usr/local/lib/python3.11/dist-pac
Requirement already satisfied: typing-extensions~=4.0 in /usr/local/lib/python3.11/di
Requirement already satisfied: uvicorn>=0.14.0 in /usr/local/lib/python3.11/dist-pack
Requirement already satisfied: fsspec in /usr/local/lib/python3.11/dist-packages (fro
Requirement already satisfied: websockets<16.0,>=10.0 in /usr/local/lib/python3.11/di
Requirement already satisfied: idna>=2.8 in /usr/local/lib/python3.11/dist-packages (
Requirement already satisfied: sniffio>=1.1 in /usr/local/lib/python3.11/dist-package
Requirement already satisfied: certifi in /usr/local/lib/python3.11/dist-packages (fr
Requirement already satisfied: httpcore==1.* in /usr/local/lib/python3.11/dist-packag
Requirement already satisfied: h11>=0.16 in /usr/local/lib/python3.11/dist-packages (
Requirement already satisfied: filelock in /usr/local/lib/python3.11/dist-packages (f
Requirement already satisfied: requests in /usr/local/lib/python3.11/dist-packages (f
Requirement already satisfied: tqdm>=4.42.1 in /usr/local/lib/python3.11/dist-package
Requirement already satisfied: hf-xet<2.0.0,>=1.1.2 in /usr/local/lib/python3.11/dist
Requirement already satisfied: python-dateutil>=2.8.2 in /usr/local/lib/python3.11/di
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.11/dist-package
Requirement already satisfied: tzdata>=2022.7 in /usr/local/lib/python3.11/dist-packa
Requirement already satisfied: annotated-types>=0.6.0 in /usr/local/lib/python3.11/di
Requirement already satisfied: pydantic-core==2.33.2 in /usr/local/lib/python3.11/dis
Requirement already satisfied: typing-inspection>=0.4.0 in /usr/local/lib/python3.11/
Requirement already satisfied: click>=8.0.0 in /usr/local/lib/python3.11/dist-package
Requirement already satisfied: shellingham>=1.3.0 in /usr/local/lib/python3.11/dist-p
Requirement already satisfied: rich>=10.11.0 in /usr/local/lib/python3.11/dist-packag
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.11/dist-packages (f
Requirement already satisfied: markdown-it-py>=2.2.0 in /usr/local/lib/python3.11/dis
Requirement already satisfied: pygments<3.0.0,>=2.13.0 in /usr/local/lib/python3.11/d
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.11/
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.11/dist-p
Requirement already satisfied: mdurl~=0.1 in /usr/local/lib/python3.11/dist-packages
```

```
Requirement already satisfied: mdurl~=0.1 in /usr/local/lib/python3.11/dist-packages
```

```
from tensorflow.keras.applications.efficientnet_v2 import preprocess_input


def classify_image(img):
    # Resize image to 124x124 pixels (Note: Comment says 128x128, but code resizes to 124
    img = img.resize((124, 124))

    # Convert image to a NumPy array with float32 dtype
    img_array = np.array(img, dtype=np.float32)
    img_array = preprocess_input(img_array)

    # Expand dimensions to match model input shape (adds a batch dimension)
    img_array = np.expand_dims(img_array, axis=0)

    # Make a prediction using the trained model
    prediction = model.predict(img_array)

    # Get the index of the highest predicted probability
    predicted_class_index = np.argmax(prediction)

    # Map the predicted index to its corresponding class name
    predicted_class_name = class_names[predicted_class_index]

    # Extract confidence score (probability of the predicted class)
    confidence = prediction[0][predicted_class_index]

    # Return formatted prediction result with confidence score
    return f"Predicted: {predicted_class_name} (Confidence: {confidence:.2f})"


iface = gr.Interface(
    fn=classify_image,  # Function to classify image using the trained model
    inputs=gr.Image(type="pil"),  # Accepts input as a PIL image
    outputs="text"  # Outputs prediction as text
)

# Launch the interface
iface.launch()  # Start the Gradio interface for user interaction
```