In [ ]:

```python
from google.colab import drive
drive.mount('/content/drive')
import numpy as np
import os
from sklearn.metrics import accuracy_score, precision_score, f1_score, roc_curve, roc_auc_score
from skimage.io import imread
from skimage.transform import resize
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from tensorflow.keras.applications import ResNet152V2
from tensorflow.keras.layers import Dense, GlobalAveragePooling2D
from tensorflow.keras.models import Model
from tensorflow.keras.optimizers import Adam

# Function to load images and labels from folder
def load_images_and_labels(folder, target_size=(224, 224)):
    images = []
    labels = []
    for subfolder in os.listdir(folder):
        subfolder_path = os.path.join(folder, subfolder)
        label = subfolder
        for filename in os.listdir(subfolder_path):
            image_path = os.path.join(subfolder_path, filename)
            try:
                img = imread(image_path)
                if img is not None:
                    img_resized = resize(img, target_size, anti_aliasing=True)  # Resize image to target size
                    images.append(img_resized)
                    labels.append(label)
                else:
                    print(f"Error reading image: {image_path}")
            except Exception as e:
                print(f"Error loading image: {image_path} - {str(e)}")
    return np.array(images), np.array(labels)

# Folder paths
train_folder = '/content/drive/MyDrive/Cashew/train'
test_folder = '/content/drive/MyDrive/Cashew/test'
val_folder = '/content/drive/MyDrive/Cashew/val'
```

```
# Load images and labels for training, testing, and validation
X_train, y_train = load_images_and_labels(train_folder)
X_test, y_test = load_images_and_labels(test_folder)
X_val, y_val = load_images_and_labels(val_folder)

# Load pre-trained ResNet152V2 model
base_model = ResNet152V2(weights='imagenet', include_top=False)

# Add custom classifier head
x = base_model.output
x = GlobalAveragePooling2D()(x)
x = Dense(1024, activation='relu')(x)
predictions = Dense(len(os.listdir(train_folder)), activation='softmax')(x)

# Combine base model and custom head
model = Model(inputs=base_model.input, outputs=predictions)

# Freeze layers in base model
for layer in base_model.layers:
    layer.trainable = False

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

# Train the model
model.fit(X_train, y_train, epochs=5, batch_size=32, validation_data=(X_val, y_val))

# Evaluate the model
train_pred = np.argmax(model.predict(X_train), axis=1)
train_accuracy = accuracy_score(y_train, train_pred)
train_precision = precision_score(y_train, train_pred, average='weighted')
train_f1 = f1_score(y_train, train_pred, average='weighted')
train_roc_auc = roc_auc_score(y_train, model.predict_proba(X_train), average='macro', multi_class='ovr')

val_pred = np.argmax(model.predict(X_val), axis=1)
val_accuracy = accuracy_score(y_val, val_pred)
val_precision = precision_score(y_val, val_pred, average='weighted')
val_f1 = f1_score(y_val, val_pred, average='weighted')
val_roc_auc = roc_auc_score(y_val, model.predict_proba(X_val), average='macro', multi_class='ovr')
```

```
test_pred = np.argmax(model.predict(X_test), axis=1)
test_accuracy = accuracy_score(y_test, test_pred)
test_precision = precision_score(y_test, test_pred, average='weighted')
test_f1 = f1_score(y_test, test_pred, average='weighted')
test_roc_auc = roc_auc_score(y_test, model.predict_proba(X_test), average='macro', multi_class='ovr')

# Print evaluation metrics
print("Train data Accuracy:", train_accuracy)
print("Train data Precision:", train_precision)
print("Train data F1 Score:", train_f1)
print("Train data ROC AUC Score:", train_roc_auc)

print("\nValidation data Accuracy:", val_accuracy)
print("Validation data Precision:", val_precision)
print("Validation data F1 Score:", val_f1)
print("Validation data ROC AUC Score:", val_roc_auc)

print("\nTest data Accuracy:", test_accuracy)
print("Test data Precision:", test_precision)
print("Test data F1 Score:", test_f1)
print("Test data ROC AUC Score:", test_roc_auc)

# Plot ROC curve for binary classification
fpr, tpr, _ = roc_curve(y_test, model.predict_proba(X_test)[:, 1])
roc_auc = roc_auc_score(y_test, model.predict_proba(X_test)[:, 1])
plt.figure()
plt.plot(fpr, tpr, color='darkorange', lw=2, label='ROC curve (area = %0.2f)' % roc_auc)
plt.plot([0, 1], [0, 1], color='navy', lw=1, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC) Curve')
plt.legend(loc="lower right")
plt.show()
```

In [33]:

```
import numpy as np
import os
from sklearn.metrics import accuracy_score, precision_score, f1_score, roc_curve, roc_auc_score
from skimage.io import imread
from skimage.transform import resize
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from tensorflow.keras.applications import ResNet152V2
from tensorflow.keras.layers import Dense, GlobalAveragePooling2D
from tensorflow.keras.models import Model
from tensorflow.keras.optimizers import Adam
from sklearn.preprocessing import LabelEncoder
```

In [34]:

```
# Function to load images and labels from folder
def load_images_and_labels(folder, target_size=(224, 224)):
    images = []
    labels = []
    for subfolder in os.listdir(folder):
        subfolder_path = os.path.join(folder, subfolder)
        label = subfolder
        for filename in os.listdir(subfolder_path):
            image_path = os.path.join(subfolder_path, filename)
            try:
                img = imread(image_path)
                if img is not None:
                    img_resized = resize(img, target_size, anti_aliasing=True)  # Resize image to target size
                    images.append(img_resized)
                    labels.append(label)
                else:
                    print(f"Error reading image: {image_path}")
            except Exception as e:
                print(f"Error loading image: {image_path} - {str(e)}")
    return np.array(images), np.array(labels)
```

In [35]:

```
# Folder paths
train_folder = 'C:/Users/KIIT/Downloads/Cashew/Cashew/train'
test_folder ='C:/Users/KIIT/Downloads/Cashew/Cashew/test'
val_folder ='C:/Users/KIIT/Downloads/Cashew/Cashew/val'
```

In [36]:

```
# Load images and labels for training, testing, and validation
X_train, y_train = load_images_and_labels(train_folder)
X_test, y_test = load_images_and_labels(test_folder)
X_val, y_val = load_images_and_labels(val_folder)
```

In [37]:

```
# Convert string labels to integer labels
label_encoder = LabelEncoder()
y_train = label_encoder.fit_transform(y_train)
y_test = label_encoder.transform(y_test)
y_val = label_encoder.transform(y_val)
```

In [38]:

```
# Load pre-trained ResNet152V2 model
base_model = ResNet152V2(weights='imagenet', include_top=False)
```

In [39]:

```
# Add custom classifier head
x = base_model.output
x = GlobalAveragePooling2D()(x)
x = Dense(1024, activation='relu')(x)
predictions = Dense(len(os.listdir(train_folder)), activation='softmax')(x)
```

In [40]:

```
# Combine base model and custom head
model = Model(inputs=base_model.input, outputs=predictions)
```

In [41]:

```
# Freeze layers in base model
for layer in base_model.layers:
    layer.trainable = False
```

In [42]:

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

In [43]:

```
# Train the model
model.fit(X_train, y_train, epochs=10, batch_size=32, validation_data=(X_val, y_val))
```

```
Epoch 1/10
201/201 ━━━━━━━━━━━━━━━━━━━━ 927s 4s/step - accuracy: 0.7961 - loss: 0.6290 - val_accuracy: 0.9041 - val_loss: 0.2415
Epoch 2/10
201/201 ━━━━━━━━━━━━━━━━━━━━ 845s 4s/step - accuracy: 0.9173 - loss: 0.2433 - val_accuracy: 0.9290 - val_loss: 0.2130
Epoch 3/10
201/201 ━━━━━━━━━━━━━━━━━━━━ 818s 4s/step - accuracy: 0.9539 - loss: 0.1306 - val_accuracy: 0.9004 - val_loss: 0.2694
Epoch 4/10
201/201 ━━━━━━━━━━━━━━━━━━━━ 864s 4s/step - accuracy: 0.9514 - loss: 0.1462 - val_accuracy: 0.9215 - val_loss: 0.2151
Epoch 5/10
201/201 ━━━━━━━━━━━━━━━━━━━━ 810s 4s/step - accuracy: 0.9699 - loss: 0.0883 - val_accuracy: 0.9352 - val_loss: 0.2098
Epoch 6/10
201/201 ━━━━━━━━━━━━━━━━━━━━ 815s 4s/step - accuracy: 0.9865 - loss: 0.0457 - val_accuracy: 0.9377 - val_loss: 0.1981
Epoch 7/10
201/201 ━━━━━━━━━━━━━━━━━━━━ 798s 4s/step - accuracy: 0.9923 - loss: 0.0309 - val_accuracy: 0.9340 - val_loss: 0.2704
Epoch 8/10
201/201 ━━━━━━━━━━━━━━━━━━━━ 788s 4s/step - accuracy: 0.9926 - loss: 0.0279 - val_accuracy: 0.9340 - val_loss: 0.2247
Epoch 9/10
201/201 ━━━━━━━━━━━━━━━━━━━━ 796s 4s/step - accuracy: 0.9930 - loss: 0.0231 - val_accuracy: 0.9365 - val_loss: 0.2582
Epoch 10/10
201/201 ━━━━━━━━━━━━━━━━━━━━ 788s 4s/step - accuracy: 0.9977 - loss: 0.0107 - val_accuracy: 0.9340 - val_loss: 0.2715
```

```
Out[43]: <keras.src.callbacks.history.History at 0x20a076389d0>
```

In [44]:

```python
from sklearn.metrics import accuracy_score, precision_score, f1_score, roc_auc_score
import numpy as np

# Predict probability estimates
train_pred_proba = model.predict(X_train)
val_pred_proba = model.predict(X_val)
test_pred_proba = model.predict(X_test)

# Get predicted classes
train_pred = np.argmax(train_pred_proba, axis=1)
val_pred = np.argmax(val_pred_proba, axis=1)
test_pred = np.argmax(test_pred_proba, axis=1)

# Calculate evaluation metrics
train_accuracy = accuracy_score(y_train, train_pred)
train_precision = precision_score(y_train, train_pred, average='weighted')
train_f1 = f1_score(y_train, train_pred, average='weighted')
train_roc_auc = roc_auc_score(y_train, train_pred_proba, average='macro', multi_class='ovr')
```

```python
val_accuracy = accuracy_score(y_val, val_pred)
val_precision = precision_score(y_val, val_pred, average='weighted')
val_f1 = f1_score(y_val, val_pred, average='weighted')
val_roc_auc = roc_auc_score(y_val, val_pred_proba, average='macro', multi_class='ovr')

test_accuracy = accuracy_score(y_test, test_pred)
test_precision = precision_score(y_test, test_pred, average='weighted')
test_f1 = f1_score(y_test, test_pred, average='weighted')
test_roc_auc = roc_auc_score(y_test, test_pred_proba, average='macro', multi_class='ovr')
```

```
201/201 ━━━━━━━━━━━━━━━━━━━━ 702s 3s/step
26/26 ━━━━━━━━━━━━━━━━━━━━ 89s 3s/step
25/25 ━━━━━━━━━━━━━━━━━━━━ 88s 3s/step
```

In [45]:

```python
from sklearn.metrics import precision_score, f1_score, accuracy_score, roc_auc_score
import numpy as np

# Assuming you have already defined and trained your functional API model 'model'

# Make predictions on training data
train_pred_prob = model.predict(X_train)
train_pred = np.argmax(train_pred_prob, axis=1)

# Calculate evaluation metrics
train_precision = precision_score(y_train, train_pred, average='weighted')
train_f1 = f1_score(y_train, train_pred, average='weighted')
train_roc_auc = roc_auc_score(y_train, train_pred_prob, average='macro', multi_class='ovr')

# Make predictions on validation data
val_pred_prob = model.predict(X_val)
val_pred = np.argmax(val_pred_prob, axis=1)
val_accuracy = accuracy_score(y_val, val_pred)
```

```
201/201 ━━━━━━━━━━━━━━━━━━━━ 707s 4s/step
26/26 ━━━━━━━━━━━━━━━━━━━━ 93s 4s/step
```

In [46]:

```python
import tensorflow as tf
model = tf.keras.models.load_model('model_resnet.h5')
```

```
WARNING:absl:Compiled the loaded model, but the compiled metrics have yet to be built. `model.compile_metrics` will be empty un
til you train or evaluate the model.
```

In [47]:

```
from tensorflow.keras.models import load_model
model.save('model_resnet.h5')
```

```
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. This file format
is considered legacy. We recommend using instead the native Keras format, e.g. `model.save('my_model.keras')` or `keras.saving.
save_model(model, 'my_model.keras')`.
```

In [48]:

```
# Print evaluation metrics
print("Train data Accuracy:", train_accuracy)
print("Train data Precision:", train_precision)
print("Train data F1 Score:", train_f1)
print("Train data ROC AUC Score:", train_roc_auc)

print("\nValidation data Accuracy:", val_accuracy)
print("Validation data Precision:", val_precision)
print("Validation data F1 Score:", val_f1)
print("Validation data ROC AUC Score:", val_roc_auc)

print("\nTest data Accuracy:", test_accuracy)
print("Test data Precision:", test_precision)
print("Test data F1 Score:", test_f1)
print("Test data ROC AUC Score:", test_roc_auc)
```

```
Train data Accuracy: 0.99875039050296 79
```

```
Train data Accuracy: 0.9987503905029679
Train data Precision: 0.9987552479297774
Train data F1 Score: 0.998750041536351
Train data ROC AUC Score: 0.9999990890296881

Validation data Accuracy: 0.933997509339975
Validation data Precision: 0.934752466579226
Validation data F1 Score: 0.9335814235781698
Validation data ROC AUC Score: 0.9918673658286641

Test data Accuracy: 0.91875
Test data Precision: 0.9208398299293039
Test data F1 Score: 0.917949499764391
Test data ROC AUC Score: 0.9843552083333333
```

In [ ]: