```
from tensorflow import keras
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.preprocessing import image
#from tensorflow.keras.applications.inception_v3 import InceptionV3, preprocess_input
from tensorflow.keras.applications.inception_resnet_v2 import InceptionResNetV2, preprocess_input
from tensorflow.keras.layers import Dense, Flatten
from tensorflow.keras.models import Model
from tensorflow.keras.optimizers import Adam
import numpy as np
import random
import matplotlib.pyplot as plt
%matplotlib inline
```

```
import os
import zipfile
```

```
import zipfile
import os

# Define the path to the ZIP file
zip_file_path = "/content/archive (2).zip"

# Define the extraction directory
extraction_directory = "/content"  # Replace with the desired directory

# Create a ZipFile object
with zipfile.ZipFile(zip_file_path, 'r') as zip_ref:
    # Extract all the contents of the ZIP file to the extraction directory
    zip_ref.extractall(extraction_directory)

# Define TRAINING_DIR as the path to the 'train' folder within the extraction directory
TRAINING_DIR = os.path.join(extraction_directory, 'ds_frutas_am/train')
TEST_DIR = os.path.join(extraction_directory,'ds_frutas_am/test')
```

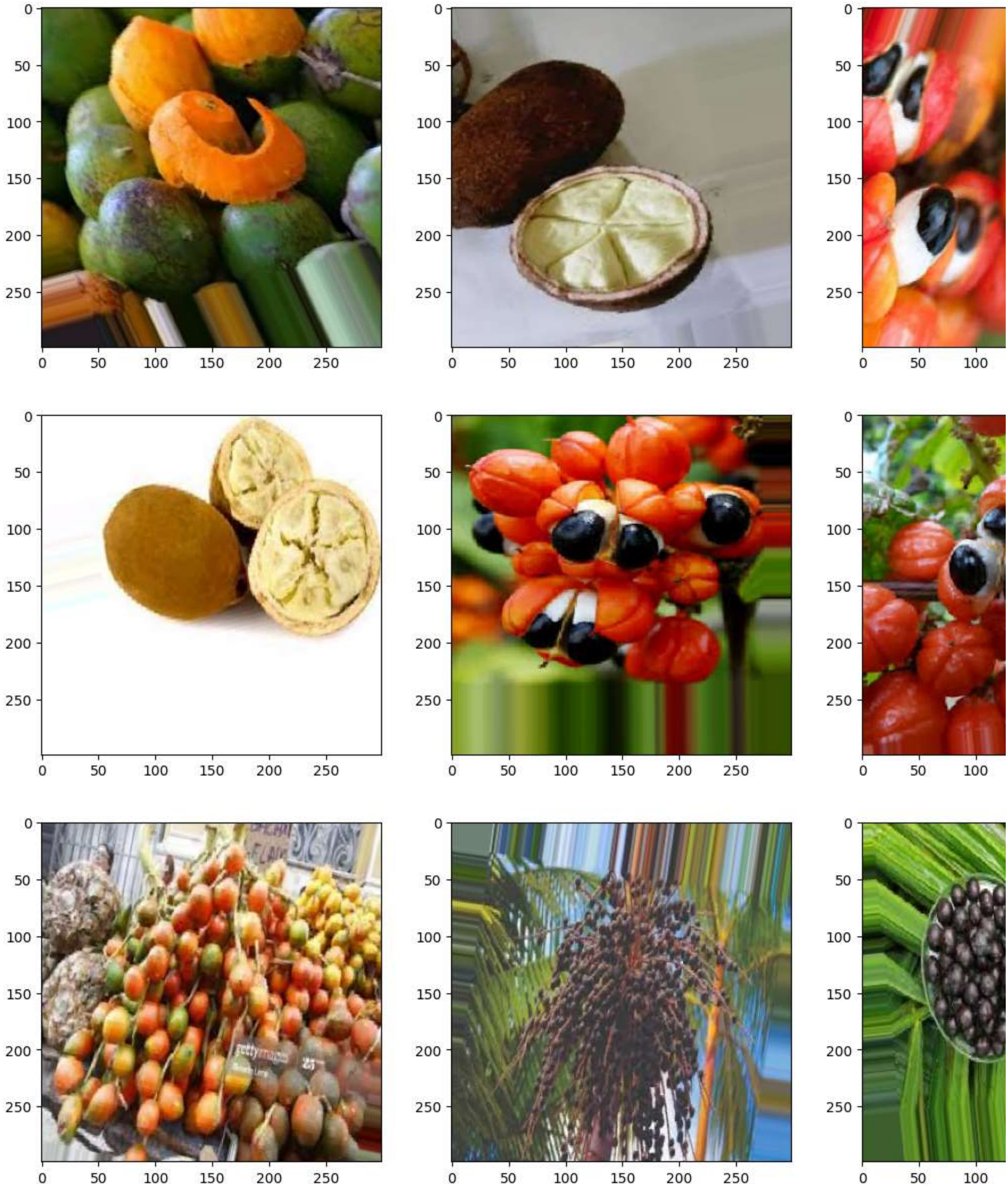
```
# Alguns parâmetros para leitura do dataset
im_shape = (299,299)


seed = 10

BATCH_SIZE = 16
```

```
#Using keras ImageGenerator and flow_from_directoty

# Image dataset without augmentation
#data_generator = ImageDataGenerator(preprocessing_function=preprocess_input, validation_split=0.2)
# With augmentation
data_generator = ImageDataGenerator(
        validation_split=0.2,
        rotation_range=20,
        width_shift_range=0.2,
        height_shift_range=0.2,
        preprocessing_function=preprocess_input,
        shear_range=0.2,
        zoom_range=0.2,
        horizontal_flip=True,
        fill_mode='nearest')
val_data_generator = ImageDataGenerator(preprocessing_function=preprocess_input,validation_split=0.2)


# Generator para parte train
train_generator = data_generator.flow_from_directory(TRAINING_DIR, target_size=im_shape, shuffle=True, seed=seed,class_mode='
# Generator para parte validação
validation_generator = val_data_generator.flow_from_directory(TRAINING_DIR, target_size=im_shape, shuffle=False, seed=seed,cl

# Generator para dataset de teste
test_generator = ImageDataGenerator(preprocessing_function=preprocess_input)
test_generator = test_generator.flow_from_directory(TEST_DIR, target_size=im_shape, shuffle=False, seed=seed,class_mode='cate

nb_train_samples = train_generator.samples
nb_validation_samples = validation_generator.samples
nb_test_samples = test_generator.samples
classes = list(train_generator.class_indices.keys())
print('Classes: '+str(classes))
num_classes  = len(classes)
```

```
Found 72 images belonging to 6 classes.
Found 18 images belonging to 6 classes.
```

```
    Found 30 images belonging to 6 classes.
    Classes: ['acai', 'cupuacu', 'graviola', 'guarana', 'pupunha', 'tucuma']
```

```
# Visualizando alguns exemplos do dataset por meio do Generator criado
plt.figure(figsize=(15,15))
for i in range(9):
    #gera subfigures
    plt.subplot(330 + 1 + i)
    batch = (train_generator.next()[0]+1)/2*255
    image = batch[0].astype('uint8')
    plt.imshow(image)
plt.show()
```



```
base_model = InceptionResNetV2(weights='imagenet',include_top=False, input_shape=(im_shape[0], im_shape[1], 3))

x = base_model.output
x = Flatten()(x)
```

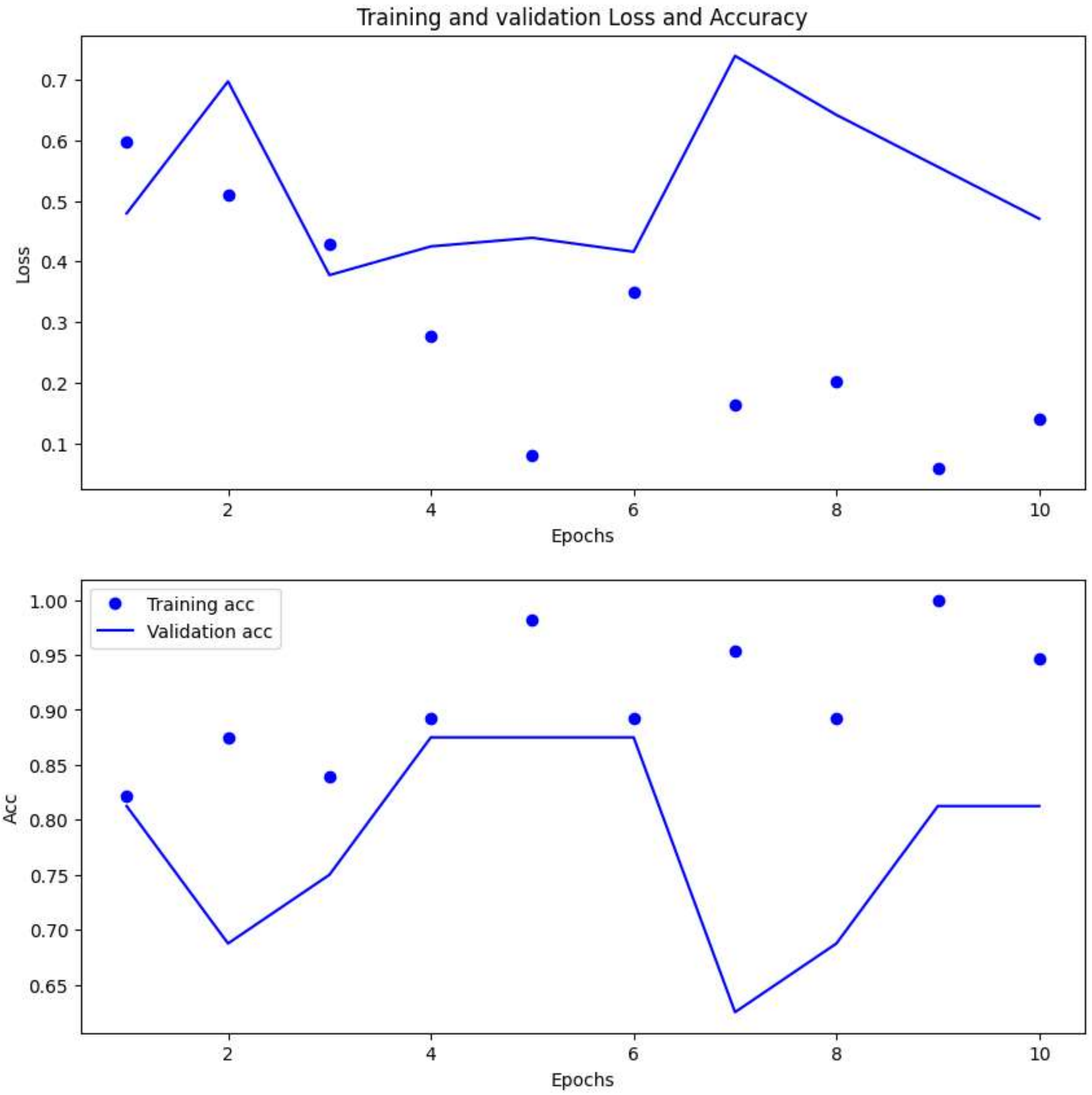
```
x = Dense(100, activation='relu')(x)
predictions = Dense(num_classes, activation='softmax', kernel_initializer='random_uniform')(x)

model = Model(inputs=base_model.input, outputs=predictions)

# Freezing pretrained layers
base_model.trainable = False

optimizer = Adam()
model.compile(optimizer=optimizer,loss='categorical_crossentropy',metrics=['accuracy'])
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/inception_resnet_v2/inception_resnet_
219055592/219055592 [==============================] - 1s 0us/step
```

```
epochs = 10

# Saving the best model
callbacks_list = [
    keras.callbacks.ModelCheckpoint(
        filepath='model.h5',
        monitor='val_loss', save_best_only=True, verbose=1),
    keras.callbacks.EarlyStopping(monitor='val_loss', patience=50,verbose=1)
]

history = model.fit(
        train_generator,
        steps_per_epoch=nb_train_samples // BATCH_SIZE,
        epochs=epochs,
        callbacks = callbacks_list,
        validation_data=validation_generator,
        verbose = 1,
        validation_steps=nb_validation_samples // BATCH_SIZE)
```

```
Epoch 1/10
4/4 [==============================] - ETA: 0s - loss: 0.5988 - accuracy: 0.8214
Epoch 1: val_loss improved from inf to 0.47956, saving model to model.h5
4/4 [==============================] - 39s 12s/step - loss: 0.5988 - accuracy: 0.8214 - val_loss: 0.4796 - val_accuracy:
Epoch 2/10
4/4 [==============================] - ETA: 0s - loss: 0.5096 - accuracy: 0.8750
Epoch 2: val_loss did not improve from 0.47956
4/4 [==============================] - 34s 10s/step - loss: 0.5096 - accuracy: 0.8750 - val_loss: 0.6976 - val_accuracy:
Epoch 3/10
4/4 [==============================] - ETA: 0s - loss: 0.4279 - accuracy: 0.8393
Epoch 3: val_loss improved from 0.47956 to 0.37758, saving model to model.h5
4/4 [==============================] - 41s 11s/step - loss: 0.4279 - accuracy: 0.8393 - val_loss: 0.3776 - val_accuracy:
Epoch 4/10
4/4 [==============================] - ETA: 0s - loss: 0.2768 - accuracy: 0.8929
Epoch 4: val_loss did not improve from 0.37758
4/4 [==============================] - 37s 9s/step - loss: 0.2768 - accuracy: 0.8929 - val_loss: 0.4250 - val_accuracy:
Epoch 5/10
4/4 [==============================] - ETA: 0s - loss: 0.0802 - accuracy: 0.9821
Epoch 5: val_loss did not improve from 0.37758
4/4 [==============================] - 37s 10s/step - loss: 0.0802 - accuracy: 0.9821 - val_loss: 0.4392 - val_accuracy:
Epoch 6/10
4/4 [==============================] - ETA: 0s - loss: 0.3488 - accuracy: 0.8929
Epoch 6: val_loss did not improve from 0.37758
4/4 [==============================] - 33s 9s/step - loss: 0.3488 - accuracy: 0.8929 - val_loss: 0.4161 - val_accuracy:
Epoch 7/10
4/4 [==============================] - ETA: 0s - loss: 0.1632 - accuracy: 0.9531
Epoch 7: val_loss did not improve from 0.37758
4/4 [==============================] - 37s 10s/step - loss: 0.1632 - accuracy: 0.9531 - val_loss: 0.7398 - val_accuracy:
Epoch 8/10
4/4 [==============================] - ETA: 0s - loss: 0.2023 - accuracy: 0.8929
Epoch 8: val_loss did not improve from 0.37758
4/4 [==============================] - 37s 9s/step - loss: 0.2023 - accuracy: 0.8929 - val_loss: 0.6422 - val_accuracy:
Epoch 9/10
4/4 [==============================] - ETA: 0s - loss: 0.0588 - accuracy: 1.0000
Epoch 9: val_loss did not improve from 0.37758
4/4 [==============================] - 34s 8s/step - loss: 0.0588 - accuracy: 1.0000 - val_loss: 0.5565 - val_accuracy:
Epoch 10/10
4/4 [==============================] - ETA: 0s - loss: 0.1406 - accuracy: 0.9464
Epoch 10: val_loss did not improve from 0.37758
4/4 [==============================] - 37s 9s/step - loss: 0.1406 - accuracy: 0.9464 - val_loss: 0.4707 - val_accuracy:
```

```
#Vamos ver como foi o treino?
import matplotlib.pyplot as plt

history_dict = history.history
loss_values = history_dict['loss']
val_loss_values = history_dict['val_loss']

epochs_x = range(1, len(loss_values) + 1)
plt.figure(figsize=(10,10))
plt.subplot(2,1,1)
plt.plot(epochs_x, loss_values, 'bo', label='Training loss')
plt.plot(epochs_x, val_loss_values, 'b', label='Validation loss')
plt.title('Training and validation Loss and Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Loss')
#plt.legend()
plt.subplot(2,1,2)
acc_values = history_dict['accuracy']
val_acc_values = history_dict['val_accuracy']
plt.plot(epochs_x, acc_values, 'bo', label='Training acc')
plt.plot(epochs_x, val_acc_values, 'b', label='Validation acc')
#plt.title('Training and validation accuracy')
plt.xlabel('Epochs')
plt.ylabel('Acc')
plt.legend()
plt.show()
```



```
from tensorflow.keras.models import load_model
# Load the best saved model
model = load_model('model.h5')


# Using the validation dataset
score = model.evaluate_generator(validation_generator)
print('Val loss:', score[0])
print('Val accuracy:', score[1])
```

```
<ipython-input-33-0b1386c018fa>:2: UserWarning: `Model.evaluate_generator` is deprecated and will be removed in a futur
  score = model.evaluate_generator(validation_generator)
Val loss: 0.4711366891860962
Val accuracy: 0.6666666865348816
```

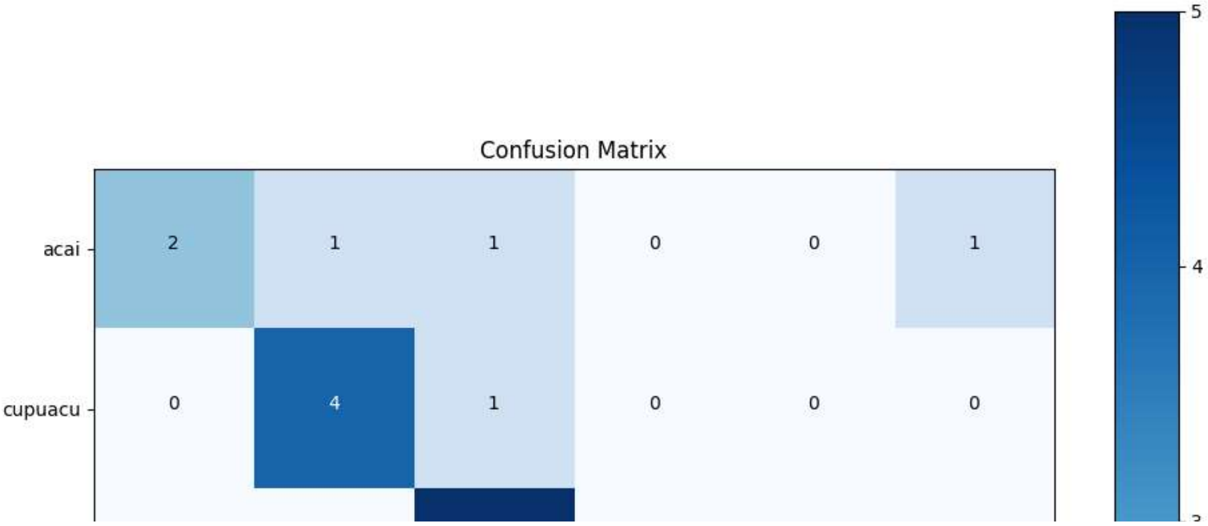
```
# Using the test dataset
score = model.evaluate_generator(test_generator)
print('Test loss:', score[0])
print('Test accuracy:', score[1])
```

```
<ipython-input-34-8baddd65724c>:2: UserWarning: `Model.evaluate_generator` is deprecated and will be removed in a future
  score = model.evaluate_generator(test_generator)
Test loss: 0.5224454402923584
Test accuracy: 0.800000011920929
```

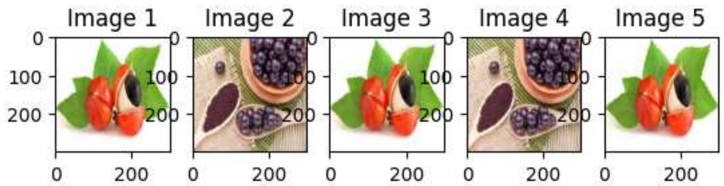
```
import itertools

#Plot the confusion matrix. Set Normalize = True/False
def plot_confusion_matrix(cm, classes, normalize=True, title='Confusion matrix', cmap=plt.cm.Blues):
    """
    This function prints and plots the confusion matrix.
    Normalization can be applied by setting `normalize=True`.
    """
    plt.figure(figsize=(10,10))
    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title)
    plt.colorbar()
    tick_marks = np.arange(len(classes))
    plt.xticks(tick_marks, classes, rotation=45)
    plt.yticks(tick_marks, classes)
    if normalize:
        cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
        cm = np.around(cm, decimals=2)
        cm[np.isnan(cm)] = 0.0
    thresh = cm.max() / 2.
    for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
        plt.text(j, i, cm[i, j],
                 horizontalalignment="center",
                 color="white" if cm[i, j] > thresh else "black")
    plt.tight_layout()
    plt.ylabel('True label')
    plt.xlabel('Predicted label')
```

```
# Some reports
from sklearn.metrics import classification_report, confusion_matrix
import numpy as np

#Confution Matrix and Classification Report
Y_pred = model.predict_generator(test_generator)#, nb_test_samples // BATCH_SIZE, workers=1)
y_pred = np.argmax(Y_pred, axis=1)
target_names = classes

#Confution Matrix
cm = confusion_matrix(test_generator.classes, y_pred)
plot_confusion_matrix(cm, target_names, normalize=False, title='Confusion Matrix')
print('Classification Report')
print(classification_report(test_generator.classes, y_pred, target_names=target_names))
```

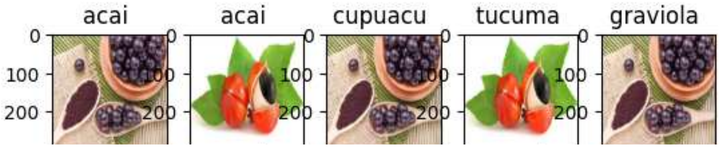
```
<ipython-input-36-c1926f3dc117>:6: UserWarning: `Model.predict_generator` is deprecated and will be removed in a future
  Y_pred = model.predict_generator(test_generator)#, nb_test_samples // BATCH_SIZE, workers=1)
Classification Report
              precision    recall  f1-score   support

        acai       1.00      0.40      0.57         5
     cupuacu       0.80      0.80      0.80         5
    graviola       0.71      1.00      0.83         5
     guarana       1.00      0.80      0.89         5
     pupunha       0.80      0.80      0.80         5
      tucuma       0.71      1.00      0.83         5

    accuracy                           0.80        30
   macro avg       0.84      0.80      0.79        30
weighted avg       0.84      0.80      0.79        30
```



```
import matplotlib.pyplot as plt

# Assuming you have a test generator named test_generator

# Display the first batch of images from the test generator
batch = test_generator.next()[0]
# Assuming the images need to be rescaled to be in the [0, 255] range
batch = (batch + 1) / 2 * 255
image = batch[0].astype('uint8')  # Displaying the first image in the batch

# Display the image
plt.imshow(image, cmap="Greys")
plt.title('{} '.format(target_names[y_pred[0]]))
plt.show()
```



```
import matplotlib.pyplot as plt

# Assuming you have a test generator named test_generator
num_images_to_display = 5

# Loop through the first 5 batches and display the first image in each batch
for i in range(num_images_to_display):
    batch = test_generator.next()[0]
    batch = (batch + 1) / 2 * 255
    image = batch[0].astype('uint8')

    # Display the image
    plt.subplot(1, num_images_to_display, i+1)
    plt.imshow(image, cmap="Greys")
    plt.title('{} '.format(target_names[y_pred[i]]))

plt.show()
```



```
target_names
```

```
['acai', 'cupuacu', 'graviola', 'guarana', 'pupunha', 'tucuma']
```