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

```
Mounted at /content/drive
```

```
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
```

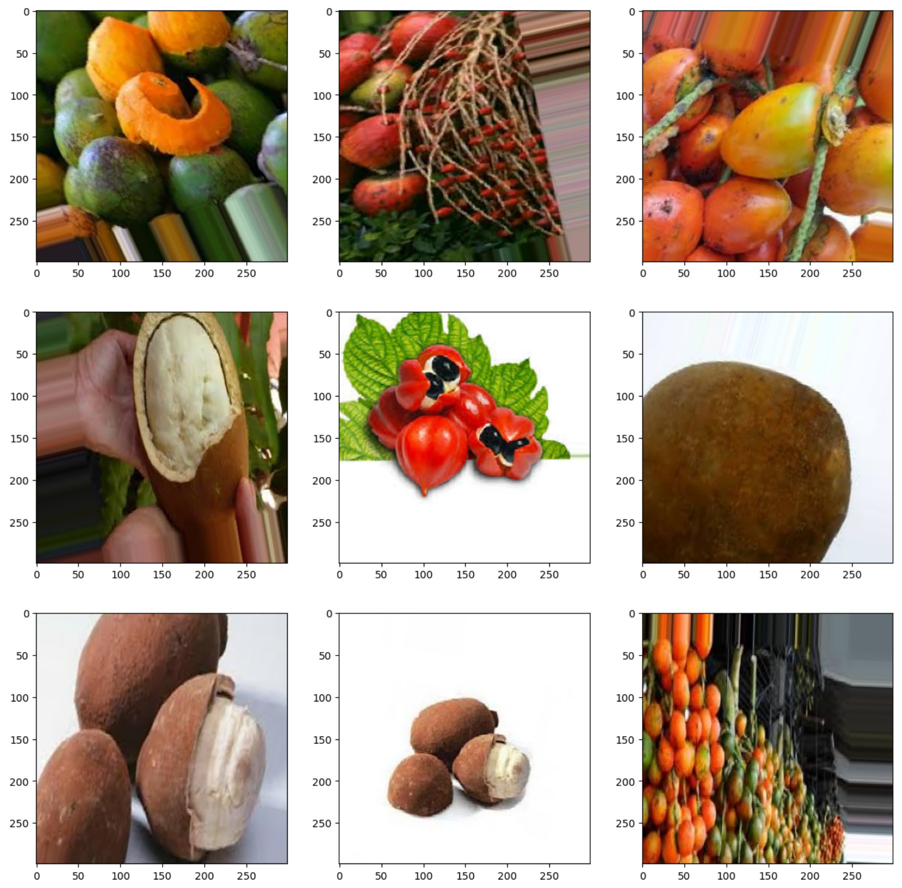
```
im_shape = (299,299)
TRAINING_DIR = '/content/drive/MyDrive/ds_frutas_am/train'
TEST_DIR = '/content/drive/MyDrive/ds_frutas_am/test'
seed = 10
BATCH_SIZE = 10
```

```
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)
```

```
train_generator = data_generator.flow_from_directory(TRAINING_DIR, target_size=im_shape, shuffle=True, seed=seed,
                                                     class_mode='categorical', batch_size=BATCH_SIZE, subset="training")
validation_generator = val_data_generator.flow_from_directory(TRAINING_DIR, target_size=im_shape, shuffle=False, seed=seed,
                                                     class_mode='categorical', batch_size=BATCH_SIZE, subset="validation")
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='categorical', batch_size=BATCH_SIZE)
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']
```

```
plt.figure(figsize=(15,15))
for i in range(9):
    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'])

```

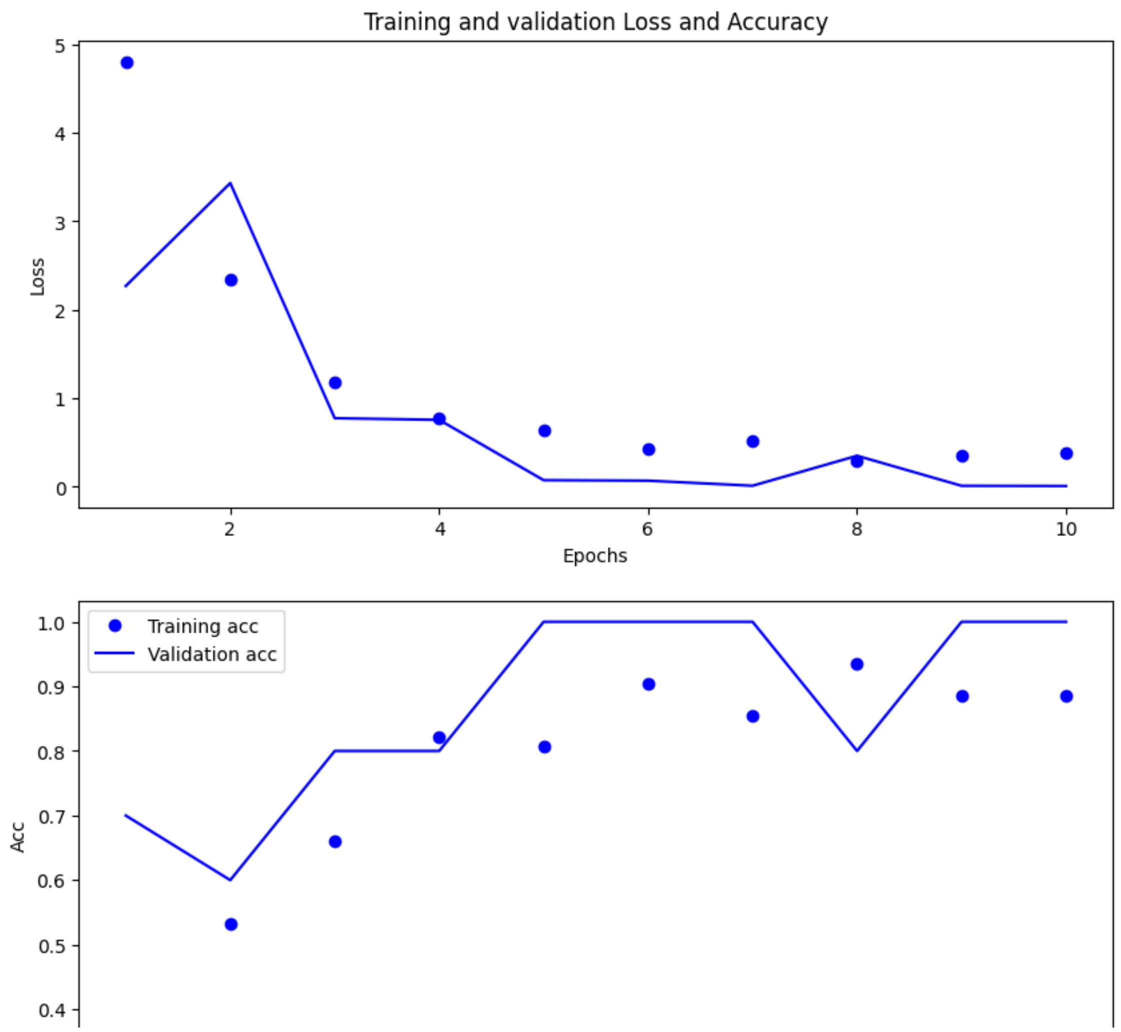
```
epochs = 10

# 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
7/7 [==============================] - ETA: 0s - loss: 4.8056 - accuracy: 0.3429
Epoch 1: val_loss improved from inf to 2.26993, saving model to model.h5
/usr/local/lib/python3.10/dist-packages/keras/src/engine/training.py:3000: UserWarning: You are saving your model as an HDF5 file via `m
  saving_api.save_model(
7/7 [==============================] - 82s 10s/step - loss: 4.8056 - accuracy: 0.3429 - val_loss: 2.2699 - val_accuracy: 0.7000
Epoch 2/10
7/7 [==============================] - ETA: 0s - loss: 2.3466 - accuracy: 0.5323
Epoch 2: val_loss did not improve from 2.26993
7/7 [==============================] - 43s 6s/step - loss: 2.3466 - accuracy: 0.5323 - val_loss: 3.4326 - val_accuracy: 0.6000
Epoch 3/10
7/7 [==============================] - ETA: 0s - loss: 1.1758 - accuracy: 0.6613
Epoch 3: val_loss improved from 2.26993 to 0.77618, saving model to model.h5
7/7 [==============================] - 45s 6s/step - loss: 1.1758 - accuracy: 0.6613 - val_loss: 0.7762 - val_accuracy: 0.8000
Epoch 4/10
7/7 [==============================] - ETA: 0s - loss: 0.7793 - accuracy: 0.8226
Epoch 4: val_loss improved from 0.77618 to 0.75720, saving model to model.h5
7/7 [==============================] - 47s 7s/step - loss: 0.7793 - accuracy: 0.8226 - val_loss: 0.7572 - val_accuracy: 0.8000
Epoch 5/10
7/7 [==============================] - ETA: 0s - loss: 0.6379 - accuracy: 0.8065
Epoch 5: val_loss improved from 0.75720 to 0.07566, saving model to model.h5
7/7 [==============================] - 45s 6s/step - loss: 0.6379 - accuracy: 0.8065 - val_loss: 0.0757 - val_accuracy: 1.0000
Epoch 6/10
7/7 [==============================] - ETA: 0s - loss: 0.4326 - accuracy: 0.9032
Epoch 6: val_loss improved from 0.07566 to 0.07004, saving model to model.h5
7/7 [==============================] - 44s 6s/step - loss: 0.4326 - accuracy: 0.9032 - val_loss: 0.0700 - val_accuracy: 1.0000
Epoch 7/10
7/7 [==============================] - ETA: 0s - loss: 0.5137 - accuracy: 0.8548
Epoch 7: val_loss improved from 0.07004 to 0.01304, saving model to model.h5
7/7 [==============================] - 47s 7s/step - loss: 0.5137 - accuracy: 0.8548 - val_loss: 0.0130 - val_accuracy: 1.0000
Epoch 8/10
7/7 [==============================] - ETA: 0s - loss: 0.2939 - accuracy: 0.9355
Epoch 8: val_loss did not improve from 0.01304
7/7 [==============================] - 42s 6s/step - loss: 0.2939 - accuracy: 0.9355 - val_loss: 0.3517 - val_accuracy: 0.8000
Epoch 9/10
7/7 [==============================] - ETA: 0s - loss: 0.3588 - accuracy: 0.8857
Epoch 9: val_loss improved from 0.01304 to 0.01240, saving model to model.h5
7/7 [==============================] - 48s 7s/step - loss: 0.3588 - accuracy: 0.8857 - val_loss: 0.0124 - val_accuracy: 1.0000
Epoch 10/10
7/7 [==============================] - ETA: 0s - loss: 0.3771 - accuracy: 0.8857
Epoch 10: val_loss improved from 0.01240 to 0.01022, saving model to model.h5
7/7 [==============================] - 51s 8s/step - loss: 0.3771 - accuracy: 0.8857 - val_loss: 0.0102 - val_accuracy: 1.0000
```

```
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')
model
```

```
<keras.src.engine.functional.Functional at 0x7cd6bb5b6b00>
```

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

```
<ipython-input-20-5bb3ca6a84d7>:1: UserWarning: `Model.evaluate_generator` is deprecated and will be removed in a future version. Please
  score = model.evaluate_generator(validation_generator)
Val loss: 0.4767059087753296
Val accuracy: 0.7777777910232544
```

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

```
<ipython-input-21-9f0bbecda02c>:1: UserWarning: `Model.evaluate_generator` is deprecated and will be removed in a future version. Please
  score = model.evaluate_generator(test_generator)
Test loss: 0.7285047173500061
Test accuracy: 0.800000011920929
```

```
import itertools
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')
```

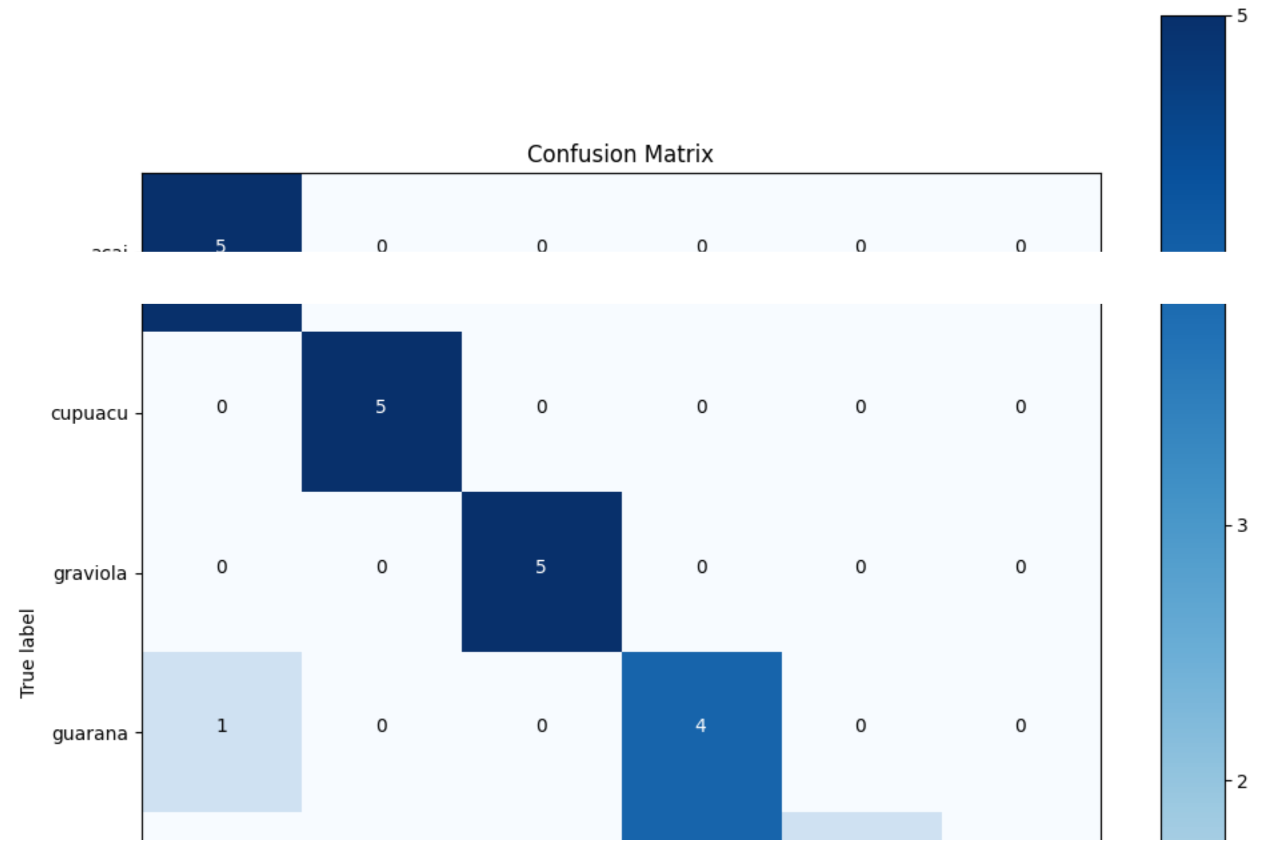
```
from sklearn.metrics import classification_report, confusion_matrix
import numpy as np

#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))
```

```
        acai       0.71      1.00      0.83         5
     cupuacu       1.00      1.00      1.00         5
    graviola       1.00      1.00      1.00         5
     guarana       0.50      0.80      0.62         5
     pupunha       1.00      0.20      0.33         5
      tucuma       1.00      0.80      0.89         5

    accuracy                           0.80        30
   macro avg       0.87      0.80      0.78        30
weighted avg       0.87      0.80      0.78        30
```



acai	5	0	0	0	0	0
cupuacu	0	5	0	0	0	0
graviola	0	0	5	0	0	0
guarana	1	0	0	4	0	0

1