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

path = '/content/drive/MyDrive/trabalhofinalES2/dataset.zip'
```

```
import zipfile

zip_path = '/content/drive/MyDrive/trabalhofinalES2/dataset.zip'
extract_path = '/content/drive/MyDrive/trabalhofinalES2/'

# Descompactar
with zipfile.ZipFile(zip_path, 'r') as zip_ref:
    zip_ref.extractall(extract_path)

print("Extração realizada!")
```

```
Extração realizada!
```

```
import os
import numpy as np
import matplotlib.pyplot as plt
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout, Input
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.callbacks import EarlyStopping
from sklearn.metrics import classification_report, confusion_matrix
import seaborn as sns
```

```
#teste de grãos quebrados e inteiros
x_teste = '/content/drive/MyDrive/trabalhofinalES2/dataset/testes_graos_arroz'

#treino de grãos inteiros e quebrados
x_treino = '/content/drive/MyDrive/trabalhofinalES2/dataset/treino_graos_arroz'
```

```
#fazer alterções na imagem para ser melhor resultado

train_datagen = ImageDataGenerator(
    rescale=1./255,
    rotation_range=25,
    width_shift_range=0.1,
    height_shift_range=0.1,
    zoom_range=0.2,
    horizontal_flip=True
)
```

```
# Gerador para o conjunto de dados de treinamento
train_generator = train_datagen.flow_from_directory(
    x_treino,
    target_size=(64, 64),
    batch_size=16,
    class_mode='binary',
    shuffle=True
)

test_datagen = ImageDataGenerator(rescale=1./255)

test_generator = test_datagen.flow_from_directory(
    x_teste,
    target_size=(64, 64),
    batch_size=16,
    class_mode='binary',
    shuffle=False
)
```

```
Found 154 images belonging to 2 classes.
Found 38 images belonging to 2 classes.
```

```
# Cálculo dos pesos para lidar com desequilíbrio entre as classes

classes = np.array([0, 1])  # 0: grao_quebrado, 1: graos_inteiros
weights = compute_class_weight(
    class_weight='balanced',
    classes=classes,
    y=train_generator.classes
```

```
)
class_weights = dict(zip(classes, weights))
print("Class weights:", class_weights)
```

```
Class weights: {np.int64(0): np.float64(0.8020833333333334), np.int64(1): np.float64(1.3275862068965518)}
```

```
model = Sequential([
    Input(shape=(64, 64, 3)),

    Conv2D(32, (3, 3), activation='relu'),
    MaxPooling2D(2, 2),
    Dropout(0.2),  # mais leve na primeira camada

    Conv2D(64, (3, 3), activation='relu'),
    MaxPooling2D(2, 2),
    Dropout(0.3),

    Flatten(),
    Dense(128, activation='relu', kernel_initializer='he_normal'),
    Dropout(0.5),  # aumenta para regular melhor a camada densa
    Dense(1, activation='sigmoid')  # saída binária
])

model.compile(
    optimizer=Adam(learning_rate=0.0005),
    loss='binary_crossentropy',
    metrics=['accuracy']
)

early_stop = EarlyStopping(
    monitor='val_loss',
    patience=5,
    restore_best_weights=True
)
```

```
history = model.fit(
    train_generator,
    epochs=101,
    validation_data=test_generator,
    class_weight=class_weights,
    callbacks=[early_stop]
)

# Avaliação
loss, acc = model.evaluate(test_generator)
print(f"\n🤖📋 Resultado Acurácia no teste: {acc:.4f}")
```

```
Epoch 1/101
10/10 ━━━━━━━━━━━━━━━━━━━━ 1s 107ms/step - accuracy: 0.9700 - loss: 0.0796 - val_accuracy: 0.6842 - val_loss: 0.7983
Epoch 2/101
10/10 ━━━━━━━━━━━━━━━━━━━━ 1s 95ms/step - accuracy: 1.0000 - loss: 0.0318 - val_accuracy: 0.9737 - val_loss: 0.1450
Epoch 3/101
10/10 ━━━━━━━━━━━━━━━━━━━━ 1s 87ms/step - accuracy: 0.9826 - loss: 0.0606 - val_accuracy: 0.7895 - val_loss: 0.5785
Epoch 4/101
10/10 ━━━━━━━━━━━━━━━━━━━━ 1s 91ms/step - accuracy: 0.9711 - loss: 0.0443 - val_accuracy: 0.7895 - val_loss: 0.5738
Epoch 5/101
10/10 ━━━━━━━━━━━━━━━━━━━━ 1s 130ms/step - accuracy: 0.9837 - loss: 0.0384 - val_accuracy: 0.9737 - val_loss: 0.1287
Epoch 6/101
10/10 ━━━━━━━━━━━━━━━━━━━━ 1s 135ms/step - accuracy: 0.9894 - loss: 0.0761 - val_accuracy: 0.6842 - val_loss: 1.0756
Epoch 7/101
10/10 ━━━━━━━━━━━━━━━━━━━━ 1s 104ms/step - accuracy: 0.9814 - loss: 0.0667 - val_accuracy: 0.9737 - val_loss: 0.0877
Epoch 8/101
10/10 ━━━━━━━━━━━━━━━━━━━━ 1s 92ms/step - accuracy: 0.9690 - loss: 0.1075 - val_accuracy: 0.6579 - val_loss: 1.0436
Epoch 9/101
10/10 ━━━━━━━━━━━━━━━━━━━━ 1s 95ms/step - accuracy: 0.9393 - loss: 0.0887 - val_accuracy: 0.8947 - val_loss: 0.2256
Epoch 10/101
10/10 ━━━━━━━━━━━━━━━━━━━━ 1s 89ms/step - accuracy: 0.9925 - loss: 0.0376 - val_accuracy: 0.8421 - val_loss: 0.5206
Epoch 11/101
10/10 ━━━━━━━━━━━━━━━━━━━━ 1s 89ms/step - accuracy: 0.9820 - loss: 0.0516 - val_accuracy: 0.9211 - val_loss: 0.1824
Epoch 12/101
10/10 ━━━━━━━━━━━━━━━━━━━━ 1s 96ms/step - accuracy: 0.9859 - loss: 0.0478 - val_accuracy: 0.8421 - val_loss: 0.5696
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 40ms/step - accuracy: 0.9790 - loss: 0.0766

🤖📋 Resultado Acurácia no teste: 0.9737
```

```
import matplotlib.pyplot as plt

fig, axs = plt.subplots(1, 2, figsize=(10, 4))

# Gráfico de Acurácia
axs[0].plot(history.history['accuracy'], label='Treinamento')
axs[0].plot(history.history['val_accuracy'], label='Validação')
```

```
axs[0].set_title('Acurácia')
axs[0].set_xlabel('Épocas')
axs[0].set_ylabel('Acurácia')
axs[0].legend()
axs[0].grid(True)

# Gráfico de Perda
axs[1].plot(history.history['loss'], label='Treinamento')
axs[1].plot(history.history['val_loss'], label='Validação')
axs[1].set_title('Erro')
axs[1].set_xlabel('Épocas')
axs[1].set_ylabel('Erro')
axs[1].legend()
axs[1].grid(True)

plt.tight_layout()

plt.savefig('/content/drive/MyDrive/trabalhofinalES2/graficos.png')

plt.show()
```