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

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

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).
```

```
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
from sklearn.utils.class_weight import compute_class_weight

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, #estou normalizando com o data aumentation
    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), #redimensionando todas as imagens para 64x64
    batch_size=16,
    class_mode='binary',
    shuffle=True #embaraljando a cada ciclo(epochs)
)

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'), #o relu vai melhorar o aprendizado
    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']
)
```

```
history = model.fit(
    train_generator,
    epochs=138,
    validation_data=test_generator,
    class_weight=class_weights
)


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

```
Epoch 1/138
10/10 ━━━━━━━━━━━━━━━━━━━━ 1s 130ms/step - accuracy: 0.9849 - loss: 0.0511 - val_accuracy: 0.9211 - val_loss: 0.1969
Epoch 2/138
10/10 ━━━━━━━━━━━━━━━━━━━━ 2s 120ms/step - accuracy: 0.9943 - loss: 0.0284 - val_accuracy: 0.9211 - val_loss: 0.1654
Epoch 3/138
10/10 ━━━━━━━━━━━━━━━━━━━━ 1s 91ms/step - accuracy: 0.9955 - loss: 0.0301 - val_accuracy: 0.8947 - val_loss: 0.2441
Epoch 4/138
10/10 ━━━━━━━━━━━━━━━━━━━━ 1s 91ms/step - accuracy: 0.9970 - loss: 0.0302 - val_accuracy: 0.8684 - val_loss: 0.3184
Epoch 5/138
10/10 ━━━━━━━━━━━━━━━━━━━━ 1s 95ms/step - accuracy: 0.9877 - loss: 0.0304 - val_accuracy: 0.8421 - val_loss: 0.4638
Epoch 6/138
10/10 ━━━━━━━━━━━━━━━━━━━━ 1s 97ms/step - accuracy: 0.9719 - loss: 0.0401 - val_accuracy: 0.9737 - val_loss: 0.0856
Epoch 7/138
10/10 ━━━━━━━━━━━━━━━━━━━━ 1s 94ms/step - accuracy: 0.9910 - loss: 0.0327 - val_accuracy: 0.8421 - val_loss: 0.4060
Epoch 8/138
10/10 ━━━━━━━━━━━━━━━━━━━━ 1s 90ms/step - accuracy: 0.9759 - loss: 0.0431 - val_accuracy: 0.8158 - val_loss: 0.5220
Epoch 9/138
10/10 ━━━━━━━━━━━━━━━━━━━━ 1s 92ms/step - accuracy: 0.9659 - loss: 0.0767 - val_accuracy: 0.8421 - val_loss: 0.5375
Epoch 10/138
10/10 ━━━━━━━━━━━━━━━━━━━━ 1s 95ms/step - accuracy: 0.9505 - loss: 0.0707 - val_accuracy: 0.9211 - val_loss: 0.1813
Epoch 11/138
10/10 ━━━━━━━━━━━━━━━━━━━━ 1s 95ms/step - accuracy: 0.9637 - loss: 0.0866 - val_accuracy: 0.8421 - val_loss: 0.3343
Epoch 12/138
10/10 ━━━━━━━━━━━━━━━━━━━━ 1s 91ms/step - accuracy: 0.9899 - loss: 0.0377 - val_accuracy: 0.8947 - val_loss: 0.2274
Epoch 13/138
10/10 ━━━━━━━━━━━━━━━━━━━━ 1s 116ms/step - accuracy: 0.9931 - loss: 0.0439 - val_accuracy: 0.9211 - val_loss: 0.1752
Epoch 14/138
10/10 ━━━━━━━━━━━━━━━━━━━━ 1s 115ms/step - accuracy: 0.9704 - loss: 0.0591 - val_accuracy: 0.8684 - val_loss: 0.4050
Epoch 15/138
10/10 ━━━━━━━━━━━━━━━━━━━━ 1s 135ms/step - accuracy: 0.9937 - loss: 0.0420 - val_accuracy: 0.9211 - val_loss: 0.1754
Epoch 16/138
10/10 ━━━━━━━━━━━━━━━━━━━━ 2s 95ms/step - accuracy: 0.9883 - loss: 0.0626 - val_accuracy: 0.8421 - val_loss: 0.4196
Epoch 17/138
10/10 ━━━━━━━━━━━━━━━━━━━━ 1s 104ms/step - accuracy: 0.9796 - loss: 0.0405 - val_accuracy: 0.9211 - val_loss: 0.1687
Epoch 18/138
10/10 ━━━━━━━━━━━━━━━━━━━━ 1s 102ms/step - accuracy: 0.9611 - loss: 0.0796 - val_accuracy: 0.8421 - val_loss: 0.4380
Epoch 19/138
```

```
10/10 ━━━━━━━━━━━━━━━━━━━━ 1s 94ms/step - accuracy: 0.9679 - loss: 0.0665 - val_accuracy: 0.9737 - val_loss: 0.0501
Epoch 20/138
10/10 ━━━━━━━━━━━━━━━━━━━━ 1s 97ms/step - accuracy: 0.9698 - loss: 0.0774 - val_accuracy: 0.8421 - val_loss: 0.4809
Epoch 21/138
10/10 ━━━━━━━━━━━━━━━━━━━━ 1s 93ms/step - accuracy: 0.9853 - loss: 0.0336 - val_accuracy: 0.9211 - val_loss: 0.2130
Epoch 22/138
10/10 ━━━━━━━━━━━━━━━━━━━━ 1s 92ms/step - accuracy: 0.9966 - loss: 0.0168 - val_accuracy: 0.8947 - val_loss: 0.2900
Epoch 23/138
10/10 ━━━━━━━━━━━━━━━━━━━━ 1s 94ms/step - accuracy: 0.9956 - loss: 0.0195 - val_accuracy: 0.8421 - val_loss: 0.5960
Epoch 24/138
10/10 ━━━━━━━━━━━━━━━━━━━━ 1s 117ms/step - accuracy: 0.9909 - loss: 0.0482 - val_accuracy: 0.9211 - val_loss: 0.2490
Epoch 25/138
10/10 ━━━━━━━━━━━━━━━━━━━━ 1s 121ms/step - accuracy: 0.9660 - loss: 0.0432 - val_accuracy: 0.8421 - val_loss: 0.5043
Epoch 26/138
10/10 ━━━━━━━━━━━━━━━━━━━━ 1s 115ms/step - accuracy: 0.9759 - loss: 0.0320 - val_accuracy: 0.9211 - val_loss: 0.2028
Epoch 27/138
10/10 ━━━━━━━━━━━━━━━━━━━━ 1s 108ms/step - accuracy: 0.9982 - loss: 0.0291 - val_accuracy: 0.9211 - val_loss: 0.2423
Epoch 28/138
10/10 ━━━━━━━━━━━━━━━━━━━━ 1s 99ms/step - accuracy: 1.0000 - loss: 0.0271 - val_accuracy: 0.9211 - val_loss: 0.2507
Epoch 29/138
```

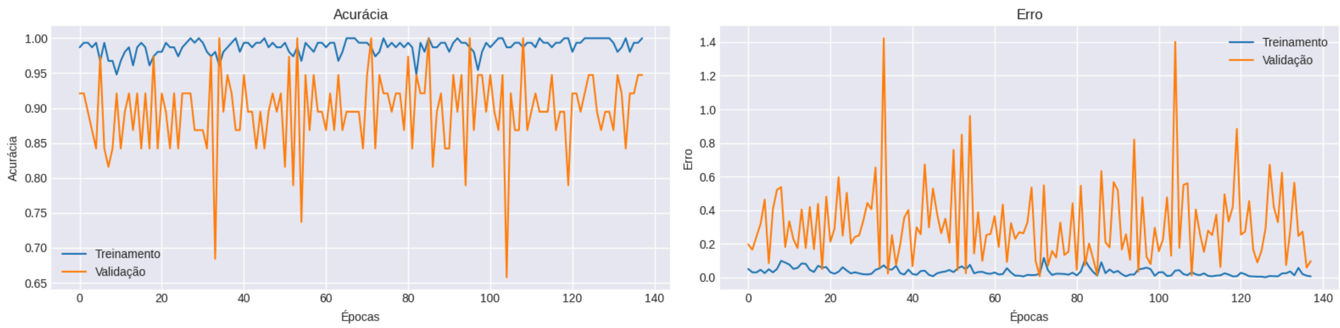
```python
import matplotlib.pyplot as plt

fig, axs = plt.subplots(1, 2, figsize=(16, 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()
```




```python
y_pred_prob = model.predict(test_generator)
y_pred = (y_pred_prob > 0.5).astype("int32")

# Matriz de Confusão
cm = confusion_matrix(test_generator.classes, y_pred)
print("\nMatriz de Confusão:")
print(cm)

target_names = ['grao_quebrado', 'graos_inteiros']
print("\nRelatório de Classificação:")
print(classification_report(test_generator.classes, y_pred, target_names=target_names))

plt.figure(figsize=(8, 6))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=target_names, yticklabels=target_names)
plt.title('Matriz de Confusão')
plt.xlabel('Previsto')
```

```
plt.ylabel('Real')
plt.show()
```

```
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 42ms/step

Matriz de Confusão:
[[20  2]
 [ 0 16]]

Relatório de Classificação:
                precision    recall  f1-score   support

 grao_quebrado       1.00      0.91      0.95        22
graos_inteiros       0.89      1.00      0.94        16

      accuracy                           0.95        38
     macro avg       0.94      0.95      0.95        38
  weighted avg       0.95      0.95      0.95        38
```