```
#MONTAGEM DO DRIVE
from google.colab import drive
drive.mount('/content/drive')
```

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

```
#IMPORTANDO AS LIBS
import os
import numpy as np
import seaborn as sns
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
```

```
#CAMINHOS ONDE ENCONTRAM-SE AS IMAGENS
train_dir = '/content/drive/MyDrive/rice/grao_quebrado'
test_dir = '/content/drive/MyDrive/rice/graos_inteiros'
```

```
#GERADORES COM DATA AUGMENTATION NO TREINO
train_datagen = ImageDataGenerator(
    rescale=1./255,
    rotation_range=9,
#    width_shift_range=0.2,
#    height_shift_range=0.2,
#    zoom_range=0.2,
    horizontal_flip=True
)

test_datagen = ImageDataGenerator(rescale=1./255)
```

```
#GERADORES
train_generator = train_datagen.flow_from_directory(
    train_dir,
    target_size=(32, 32),
    batch_size=32,
    class_mode='binary',
    shuffle=True
)

val_generator = train_datagen.flow_from_directory(
    test_dir,
    target_size=(32, 32),
    batch_size=32,
    class_mode='binary',
    subset='validation',
    shuffle=False
)

test_generator = test_datagen.flow_from_directory(
    test_dir,
    target_size=(32, 32),
    batch_size=32,
    class_mode='binary',
    shuffle=False
)
```

```
Found 74 images belonging to 2 classes.
Found 0 images belonging to 2 classes.
Found 118 images belonging to 2 classes.
```

```
#PESOS DAS CLASSES (INVERSO DA FREQUÊNCIA RELATIVA)
from sklearn.utils.class_weight import compute_class_weight

classes = np.array([0, 1]) #ONDE 0.:GRÃO_QUEBRADO, 1.:GRÃO_INTEIRO
weight = compute_class_weight(
    class_weight='balanced',
    classes=classes,
    y=train_generator.classes
)
class_weights = dict(zip(classes, weight))
print("Class weights.:", class_weights[0], '&', class_weights[1])
```

```
Class weights.: 2.3125 & 0.6379310344827587
```

```
#MODELOS
model = Sequential([
    Input(shape=(32, 32, 3)),
    Conv2D(64, (3, 3), activation='relu'),
    # MaxPooling2D(2, 2),
    # Dropout(0.3),

    Conv2D(128, (3, 3), activation='relu'),
    # MaxPooling2D(2, 2),
    # Dropout(0.3),

    Conv2D(256, (3, 3), activation='relu'),
    # MaxPooling2D(2, 2),
    # Dropout(0.3),


    Flatten(),
    Dense(128, activation='relu'),
    Dropout(0.2),
    Dense(64, activation='relu'),
    Dropout(0.1),
    Dense(1, activation='sigmoid')
])

model.compile(optimizer=Adam(learning_rate=5e-4),
              loss='binary_crossentropy',
              metrics=['accuracy'])

early_stop = EarlyStopping(monitor='val_loss', patience=20, restore_best_weights=True)
```

```
#TREINAMENTO
history = model.fit(
    train_generator,
    epochs=50,
    validation_data=test_generator,
    class_weight=class_weights,
    callbacks=[early_stop]
)
```

```
/usr/local/lib/python3.11/dist-packages/keras/src/trainers/data_adapters/py_dataset_adapter.py:121: UserWarning: Your `PyDataset`
  self._warn_if_super_not_called()
Epoch 1/50
3/3 ━━━━━━━━━━━━━━━━━━━━ 7s 2s/step - accuracy: 0.3841 - loss: 0.8337 - val_accuracy: 0.1864 - val_loss: 0.8747
Epoch 2/50
3/3 ━━━━━━━━━━━━━━━━━━━━ 1s 446ms/step - accuracy: 0.3735 - loss: 0.6894 - val_accuracy: 0.8136 - val_loss: 0.6445
Epoch 3/50
3/3 ━━━━━━━━━━━━━━━━━━━━ 1s 303ms/step - accuracy: 0.8025 - loss: 0.6555 - val_accuracy: 0.8136 - val_loss: 0.6207
Epoch 4/50
3/3 ━━━━━━━━━━━━━━━━━━━━ 1s 249ms/step - accuracy: 0.7797 - loss: 0.6953 - val_accuracy: 0.8136 - val_loss: 0.6657
Epoch 5/50
3/3 ━━━━━━━━━━━━━━━━━━━━ 1s 256ms/step - accuracy: 0.8463 - loss: 0.6421 - val_accuracy: 0.8220 - val_loss: 0.6665
Epoch 6/50
3/3 ━━━━━━━━━━━━━━━━━━━━ 1s 278ms/step - accuracy: 0.8031 - loss: 0.6754 - val_accuracy: 0.8136 - val_loss: 0.6261
Epoch 7/50
3/3 ━━━━━━━━━━━━━━━━━━━━ 1s 454ms/step - accuracy: 0.7950 - loss: 0.6918 - val_accuracy: 0.8136 - val_loss: 0.5380
Epoch 8/50
3/3 ━━━━━━━━━━━━━━━━━━━━ 1s 256ms/step - accuracy: 0.7719 - loss: 0.6474 - val_accuracy: 0.7203 - val_loss: 0.5952
Epoch 9/50
3/3 ━━━━━━━━━━━━━━━━━━━━ 1s 262ms/step - accuracy: 0.6283 - loss: 0.5852 - val_accuracy: 0.7119 - val_loss: 0.5523
Epoch 10/50
3/3 ━━━━━━━━━━━━━━━━━━━━ 1s 277ms/step - accuracy: 0.6514 - loss: 0.5871 - val_accuracy: 0.8136 - val_loss: 0.7402
Epoch 11/50
3/3 ━━━━━━━━━━━━━━━━━━━━ 1s 314ms/step - accuracy: 0.7935 - loss: 0.8715 - val_accuracy: 0.7542 - val_loss: 0.5395
Epoch 12/50
3/3 ━━━━━━━━━━━━━━━━━━━━ 1s 277ms/step - accuracy: 0.5398 - loss: 0.5710 - val_accuracy: 0.5508 - val_loss: 0.6763
Epoch 13/50
3/3 ━━━━━━━━━━━━━━━━━━━━ 1s 252ms/step - accuracy: 0.5068 - loss: 0.6112 - val_accuracy: 0.7034 - val_loss: 0.6229
Epoch 14/50
3/3 ━━━━━━━━━━━━━━━━━━━━ 1s 464ms/step - accuracy: 0.6990 - loss: 0.6068 - val_accuracy: 0.7797 - val_loss: 0.5324
Epoch 15/50
3/3 ━━━━━━━━━━━━━━━━━━━━ 1s 473ms/step - accuracy: 0.6958 - loss: 0.5588 - val_accuracy: 0.8220 - val_loss: 0.4614
Epoch 16/50
3/3 ━━━━━━━━━━━━━━━━━━━━ 1s 508ms/step - accuracy: 0.7498 - loss: 0.5567 - val_accuracy: 0.8220 - val_loss: 0.4402
Epoch 17/50
3/3 ━━━━━━━━━━━━━━━━━━━━ 1s 442ms/step - accuracy: 0.8078 - loss: 0.5006 - val_accuracy: 0.7966 - val_loss: 0.4456
Epoch 18/50
3/3 ━━━━━━━━━━━━━━━━━━━━ 1s 256ms/step - accuracy: 0.6283 - loss: 0.5186 - val_accuracy: 0.7119 - val_loss: 0.5214
Epoch 19/50
3/3 ━━━━━━━━━━━━━━━━━━━━ 1s 254ms/step - accuracy: 0.6554 - loss: 0.4560 - val_accuracy: 0.8136 - val_loss: 0.5114
Epoch 20/50
3/3 ━━━━━━━━━━━━━━━━━━━━ 1s 261ms/step - accuracy: 0.8358 - loss: 0.5899 - val_accuracy: 0.7881 - val_loss: 0.4474
Epoch 21/50
```

```
3/3 ━━━━━━━━━━━━━━━━━━━━ 1s 300ms/step - accuracy: 0.7756 - loss: 0.4246 - val_accuracy: 0.7712 - val_loss: 0.4618
Epoch 22/50
3/3 ━━━━━━━━━━━━━━━━━━━━ 1s 408ms/step - accuracy: 0.7699 - loss: 0.3567 - val_accuracy: 0.8220 - val_loss: 0.4986
Epoch 23/50
3/3 ━━━━━━━━━━━━━━━━━━━━ 1s 256ms/step - accuracy: 0.8390 - loss: 0.5308 - val_accuracy: 0.7881 - val_loss: 0.4767
Epoch 24/50
3/3 ━━━━━━━━━━━━━━━━━━━━ 1s 268ms/step - accuracy: 0.7032 - loss: 0.4426 - val_accuracy: 0.7034 - val_loss: 0.5495
Epoch 25/50
3/3 ━━━━━━━━━━━━━━━━━━━━ 1s 270ms/step - accuracy: 0.5544 - loss: 0.5131 - val_accuracy: 0.7797 - val_loss: 0.4951
Epoch 26/50
3/3 ━━━━━━━━━━━━━━━━━━━━ 1s 266ms/step - accuracy: 0.7686 - loss: 0.4443 - val_accuracy: 0.8220 - val_loss: 0.4731
Epoch 27/50
3/3 ━━━━━━━━━━━━━━━━━━━━ 1s 442ms/step - accuracy: 0.9281 - loss: 0.4198 - val_accuracy: 0.8220 - val_loss: 0.4982
Epoch 28/50
```

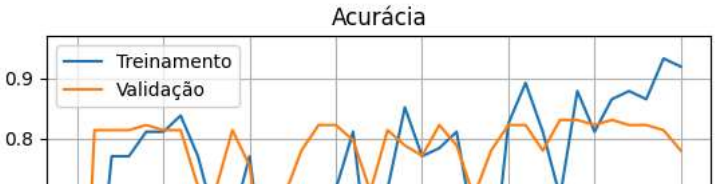
```
#AVALIAÇÃO
loss, accuracy = model.evaluate(test_generator)
print(f"\n Acurácia no teste.:{accuracy:.4f}")
```

```
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 87ms/step - accuracy: 0.7007 - loss: 0.6026

 Acurácia no teste.:0.8220
```

```
#GRÁFICOS
plt.plot(history.history['accuracy'], label='Treinamento')
plt.plot(history.history['val_accuracy'], label = 'Validação')
plt.title('Acurácia')
plt.xlabel('ÉPOCAS')
plt.ylabel('ACURÁCIA')
plt.legend()
plt.grid(True)
plt.show()

plt.plot(history.history['loss'], label='Treinamento')
plt.plot(history.history['val_loss'], label = 'Validação')
plt.title('ERRO')
plt.xlabel('ÉPOCAS')
plt.ylabel('ERRO')
plt.legend()
plt.grid(True)
plt.show()
```

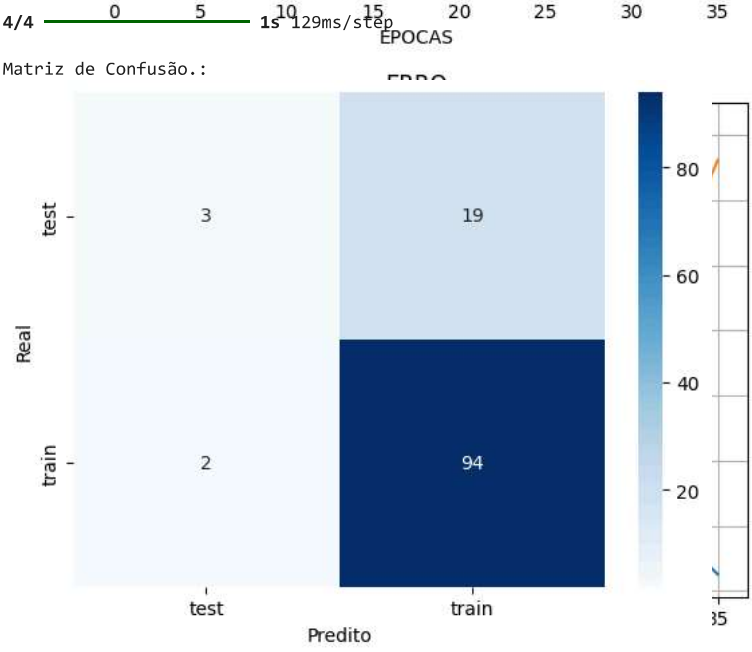



```
#AVALIAÇÃO DETALHADA
y_true = test_generator.classes
y_pred = (model.predict(test_generator) > 0.5).astype("int32").flatten()

print("\nMatriz de Confusão.:")
cm = confusion_matrix(y_true, y_pred)
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=train_generator.class_indices, yticklabels=train_generator.class_ind
plt.xlabel('Predito')
plt.ylabel('Real')
plt.show()

print("\nRelatório de Classificação.:")
print(classification_report(y_true, y_pred, target_names=list(train_generator.class_indices.keys())))
```

```
4/4 ━━━━━━━━━━━━━━━━━━━━ 1s 129ms/step
```




```
Relatório de Classificação.:
              precision    recall  f1-score   support

        test       0.60      0.14      0.22        22
       train       0.83      0.98      0.90        96

    accuracy                           0.82       118
   macro avg       0.72      0.56      0.56       118
weighted avg       0.79      0.82      0.77       118
```