## Imports e Dir base

```
from google.colab import drive
drive.mount('/content/drive', force_remount=True)

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
from sklearn.utils.class_weight import compute_class_weight

# Caminhos
path = '/content/drive/MyDrive/Colab Notebooks/img/dataset'

path_train = path + '/train'
path_test = path + '/test'
```

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

## Generators

```
train_datagen = ImageDataGenerator(
    rescale=1./255,
    rotation_range=20,
    width_shift_range=0.1,
    height_shift_range=0.1,
    zoom_range=0.2,
    horizontal_flip=True,
    validation_split=0.2  # 20% para validação
)

test_datagen = ImageDataGenerator(rescale=1./255)

train_generator = train_datagen.flow_from_directory(
    path_train,
    target_size=(64, 64),
    batch_size=16,
    class_mode='binary',
    subset='training',  # Define como treinamento
    shuffle=True
)

validation_generator = train_datagen.flow_from_directory(
    path_train,
    target_size=(64, 64),
    batch_size=16,
    class_mode='binary',
    subset='validation',  # Define como validação
    shuffle=True
)

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

```
Found 124 images belonging to 2 classes.
Found 30 images belonging to 2 classes.
Found 38 images belonging to 2 classes.
```

## Pesos das classes

```
classes = np.array([0, 1])  # 0 -  grãos quebrados, 1 - grãos inteiros
weights = compute_class_weight(
```

```
    class_weight='balanced',
    classes=classes,
    y=train_generator.classes
)
class_weights = dict(zip(classes, weights))
```

## Modelo

```
model = Sequential([
    Input(shape=(64, 64, 3)),
    Conv2D(128, (3, 3), activation='relu'),
    MaxPooling2D(2, 2),
    Dropout(0.3),

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

    Conv2D(32, (3, 3), activation='relu'),
    MaxPooling2D(2, 2),
    Dropout(0.3),

    Flatten(),

    Dense(256, activation='relu'),
    Dropout(0.2),
    Dense(128, activation='relu'),
    Dropout(0.2),
    Dense(64, activation='relu'),
    Dropout(0.3),
    Dense(32, activation='relu'),
    Dropout(0.3),
    Dense(16, activation='relu'),
    Dropout(0.3),
    Dense(8, activation='relu'),
    Dropout(0.3),
    Dense(1, activation='sigmoid')
])

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

## Treinamento

```
history = model.fit(
    train_generator,
    epochs=350,
    validation_data=validation_generator,
    class_weight=class_weights,
)
```

```
8/8 ━━━━━━━━━━━━━━━━━━━━ 1s 98ms/step - accuracy: 0.3838 - loss: 0.6971 - val_accuracy: 0.3667 - val_loss: 0.6955
Epoch 130/350
8/8 ━━━━━━━━━━━━━━━━━━━━ 1s 100ms/step - accuracy: 0.3947 - loss: 0.6981 - val_accuracy: 0.3667 - val_loss: 0.6955
Epoch 131/350
8/8 ━━━━━━━━━━━━━━━━━━━━ 1s 101ms/step - accuracy: 0.4077 - loss: 0.7029 - val_accuracy: 0.3667 - val_loss: 0.6957
Epoch 132/350
8/8 ━━━━━━━━━━━━━━━━━━━━ 1s 104ms/step - accuracy: 0.3593 - loss: 0.6868 - val_accuracy: 0.3667 - val_loss: 0.6955
Epoch 133/350
8/8 ━━━━━━━━━━━━━━━━━━━━ 1s 100ms/step - accuracy: 0.3804 - loss: 0.6902 - val_accuracy: 0.3667 - val_loss: 0.6955
Epoch 134/350
8/8 ━━━━━━━━━━━━━━━━━━━━ 1s 97ms/step - accuracy: 0.3053 - loss: 0.6689 - val_accuracy: 0.3667 - val_loss: 0.6954
Epoch 135/350
8/8 ━━━━━━━━━━━━━━━━━━━━ 1s 94ms/step - accuracy: 0.4136 - loss: 0.7015 - val_accuracy: 0.3667 - val_loss: 0.6954
Epoch 136/350
8/8 ━━━━━━━━━━━━━━━━━━━━ 1s 94ms/step - accuracy: 0.4068 - loss: 0.6965 - val_accuracy: 0.3667 - val_loss: 0.6953
Epoch 137/350
8/8 ━━━━━━━━━━━━━━━━━━━━ 1s 144ms/step - accuracy: 0.4245 - loss: 0.7061 - val_accuracy: 0.3667 - val_loss: 0.6953
Epoch 138/350
8/8 ━━━━━━━━━━━━━━━━━━━━ 1s 159ms/step - accuracy: 0.3903 - loss: 0.6945 - val_accuracy: 0.3667 - val_loss: 0.6951
Epoch 139/350
8/8 ━━━━━━━━━━━━━━━━━━━━ 3s 163ms/step - accuracy: 0.4329 - loss: 0.6978 - val_accuracy: 0.3667 - val_loss: 0.6950
Epoch 140/350
8/8 ━━━━━━━━━━━━━━━━━━━━ 1s 100ms/step - accuracy: 0.4799 - loss: 0.6703 - val_accuracy: 0.3667 - val_loss: 0.6947
Epoch 141/350
8/8 ━━━━━━━━━━━━━━━━━━━━ 1s 95ms/step - accuracy: 0.4698 - loss: 0.6998 - val_accuracy: 0.3667 - val_loss: 0.6947
Epoch 142/350
8/8 ━━━━━━━━━━━━━━━━━━━━ 1s 95ms/step - accuracy: 0.5047 - loss: 0.6868 - val_accuracy: 0.3667 - val_loss: 0.6940
Epoch 143/350
8/8 ━━━━━━━━━━━━━━━━━━━━ 1s 96ms/step - accuracy: 0.5752 - loss: 0.6748 - val_accuracy: 0.7667 - val_loss: 0.6908
Epoch 144/350
8/8 ━━━━━━━━━━━━━━━━━━━━ 1s 119ms/step - accuracy: 0.5148 - loss: 0.6776 - val_accuracy: 0.6333 - val_loss: 0.6881
```

## Avaliação

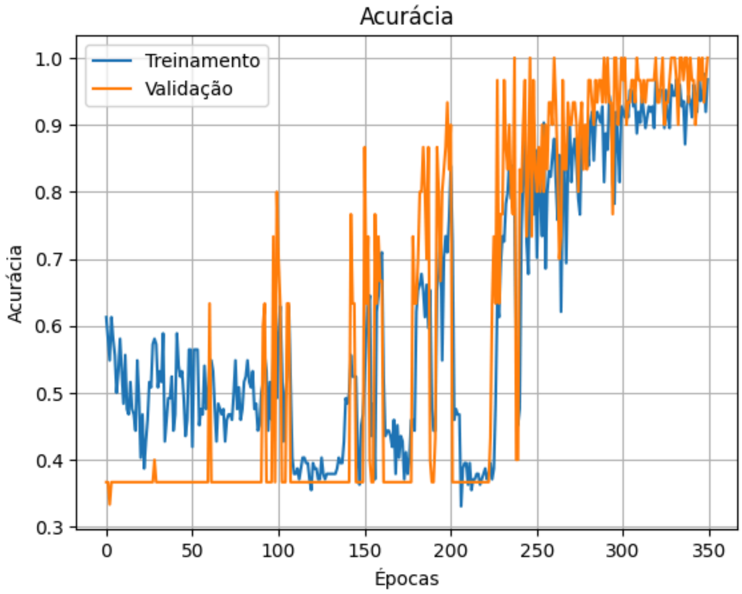
```
loss, acc = model.evaluate(test_generator)
print(f"\n🌟 Acurácia no teste: {acc:.4f}")
```

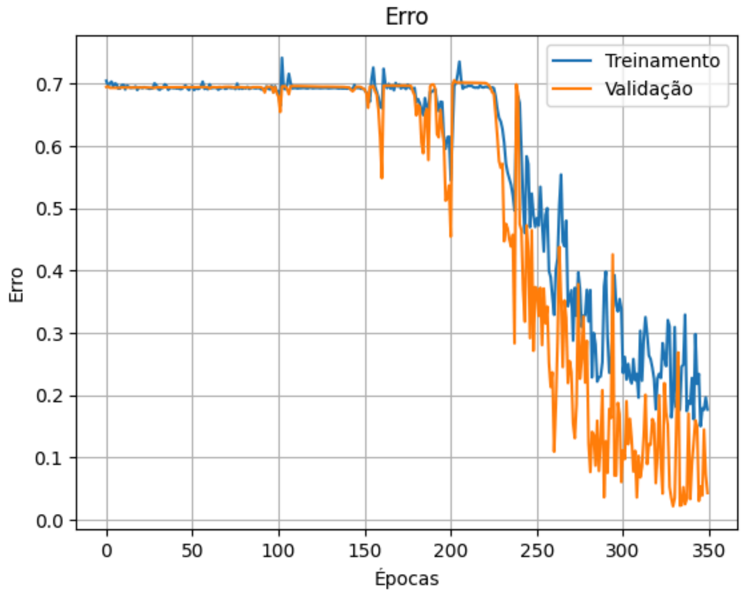
```
3/3 ━━━━━━━━━━━━━━━━━━━━ 1s 269ms/step - accuracy: 0.9371 - loss: 0.2722

🌟 Acurácia no teste: 0.9211
```

## Gráficos

## Acuracia e Erro

```
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()
```

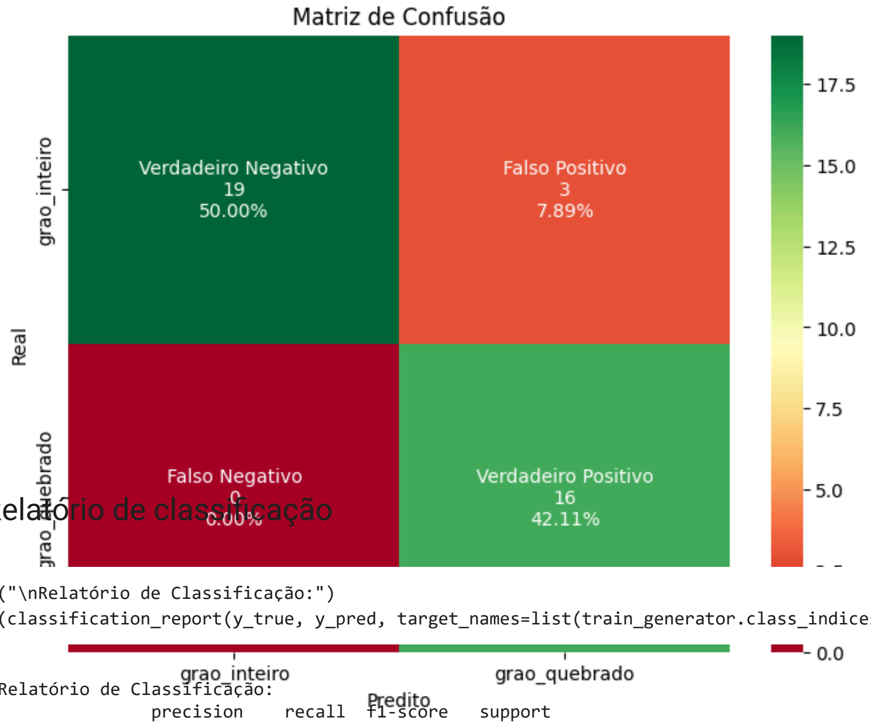



## Matriz de Confusão

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

cm = confusion_matrix(y_true, y_pred)
labels = ['Verdadeiro Negativo', 'Falso Positivo', 'Falso Negativo', 'Verdadeiro Positivo']
counts = [f"{value:0.0f}" for value in cm.flatten()]
percentages = [f"{value:.2%}" for value in cm.flatten()/np.sum(cm)]

annotations = [f"{label}\n{count}\n{percentage}" for label, count, percentage in zip(labels, counts, percentages)]
annotations = np.asarray(annotations).reshape(2,2)

plt.figure(figsize=(8, 6))
sns.heatmap(cm, annot=annotations, fmt='', cmap='RdYlGn', xticklabels=train_generator.class_indices.keys(), yticklabels=train_generator.
plt.title('Matriz de Confusão')
plt.xlabel('Predito')
plt.ylabel('Real')
plt.show()
```

```
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 78ms/step
```




## Relatório de classificação

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

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

 grao_inteiro       1.00      0.86      0.93        22
grao_quebrado       0.84      1.00      0.91        16

     accuracy                           0.92        38
    macro avg       0.92      0.93      0.92        38
 weighted avg       0.93      0.92      0.92        38
```

## Métricas

```
tn, fp, fn, tp = cm.ravel()

precisao = tp / (tp + fp) if (tp + fp) != 0 else 0
sensibilidade = tp / (tp + fn) if (tp + fn) != 0 else 0
especificidade = tn / (tn + fp) if (tn + fp) != 0 else 0
f1_score = 2 * (precisao * sensibilidade) / (precisao + sensibilidade) if (precisao + sensibilidade) != 0 else 0

print(f"Precisão: {precisao:.2f} %")
print(f"Sensibilidade (Recall): {sensibilidade:.2f} %")
print(f"Especificidade: {especificidade:.2f} %")
print(f"F1-Score: {f1_score:.2f} %")
```

```
Precisão: 0.84 %
Sensibilidade (Recall): 1.00 %
Especificidade: 0.86 %
F1-Score: 0.91 %
```