```
import os
import shutil
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import tensorflow as tf
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.model_selection import train_test_split

# --- 1. PREPARAÇÃO E DIVISÃO DOS DADOS ---
# Esta seção cria a estrutura de pastas limpa para o treinamento.

print("--- Iniciando Preparação dos Dados ---")

base_dir = '/content/drive/MyDrive/Arroz/Conjunto de dados de imagens de arroz/dataset'
train_dir_original = os.path.join(base_dir, 'train')
test_dir_original = os.path.join(base_dir, 'test')

# Novos caminhos para uma estrutura limpa em /content/
new_base_dir = '/content/arroz_dataset'
train_dir = os.path.join(new_base_dir, 'train')
validation_dir = os.path.join(new_base_dir, 'validation')
test_dir = os.path.join(new_base_dir, 'test')

# Limpa diretórios antigos se existirem
if os.path.exists(new_base_dir):
    shutil.rmtree(new_base_dir)

# Cria as novas pastas
os.makedirs(train_dir)
os.makedirs(validation_dir)
os.makedirs(test_dir)

# Tratamento para caso os diretórios originais não existam no ambiente
if not os.path.exists(base_dir):
    print(f"AVISO: O caminho '{base_dir}' não foi encontrado. Certifique-se de que o Google Drive está montado e o caminho está correto.
else:
    # Divide o diretório de treino original em um novo conjunto de treino e um de validação (80/20)
    for class_name in os.listdir(train_dir_original):
        os.makedirs(os.path.join(train_dir, class_name), exist_ok=True)
        os.makedirs(os.path.join(validation_dir, class_name), exist_ok=True)

        source_dir = os.path.join(train_dir_original, class_name)
        if os.path.isdir(source_dir):
            files = os.listdir(source_dir)
            train_files, val_files = train_test_split(files, test_size=0.2, random_state=42)

            for f in train_files:
                shutil.copy(os.path.join(source_dir, f), os.path.join(train_dir, class_name, f))
            for f in val_files:
                shutil.copy(os.path.join(source_dir, f), os.path.join(validation_dir, class_name, f))

    # Copia o diretório de teste original para a nova estrutura
    if os.path.exists(test_dir_original):
        shutil.copytree(test_dir_original, test_dir, dirs_exist_ok=True)

print("--- Divisão de Dados Concluída ---")

print("--- Construindo pipeline tf.data com Mixup ---")

IMG_SIZE = (150, 150)
BATCH_SIZE = 32
AUTO = tf.data.AUTOTUNE

train_ds = tf.keras.utils.image_dataset_from_directory(train_dir, labels='inferred', label_mode='binary', image_size=IMG_SIZE, interpola
val_ds = tf.keras.utils.image_dataset_from_directory(validation_dir, labels='inferred', label_mode='binary', image_size=IMG_SIZE, interp
test_ds = tf.keras.utils.image_dataset_from_directory(test_dir, labels='inferred', label_mode='binary', image_size=IMG_SIZE, interpolati

def augment(image, label):
    image = tf.image.random_flip_left_right(image)
    image = tf.image.random_brightness(image, max_delta=0.2)
    return image, label

def preprocess(image, label):
    return tf.cast(image, tf.float32) / 255.0, label

def mixup(ds_one, ds_two, alpha=0.2):
```

```
def mixup(ds_one, ds_two, alpha=0.2):
    images_one, labels_one = ds_one
    images_two, labels_two = ds_two
    batch_size = tf.shape(images_one)[0]

    # CORREÇÃO: Simula a distribuição Beta usando a distribuição Gamma.
    # Isto é matematicamente equivalente a tf.random.beta e funciona em versões mais antigas.
    gamma_dist_one = tf.random.gamma(shape=(batch_size,), alpha=alpha)
    gamma_dist_two = tf.random.gamma(shape=(batch_size,), alpha=alpha)
    l = gamma_dist_one / (gamma_dist_one + gamma_dist_two)

    # Ajusta o formato para a mistura
    x_l = tf.reshape(l, (batch_size, 1, 1, 1))
    y_l = tf.reshape(l, (batch_size, 1))

    # Mistura as imagens e os rótulos
    images = images_one * x_l + images_two * (1 - x_l)
    labels = labels_one * y_l + labels_two * (1 - y_l)
    return images, labels


# Construindo o pipeline de TREINO com Augmentation e Mixup
train_ds_one = train_ds.map(preprocess, num_parallel_calls=AUTO).map(augment, num_parallel_calls=AUTO)
train_ds_two = train_ds.map(preprocess, num_parallel_calls=AUTO).map(augment, num_parallel_calls=AUTO)
train_ds_mu = tf.data.Dataset.zip((train_ds_one, train_ds_two))
train_ds_mu = train_ds_mu.map(mixup, num_parallel_calls=AUTO).prefetch(AUTO)

# Construindo os pipelines de VALIDAÇÃO e TESTE
val_ds_p = val_ds.map(preprocess, num_parallel_calls=AUTO).prefetch(AUTO)
test_ds_p = test_ds.map(preprocess, num_parallel_calls=AUTO).prefetch(AUTO)


# --- 3. ARQUITETURA DO MODELO E COMPILAÇÃO ---
model = Sequential([
    Input(shape=IMG_SIZE + (3,)),
    Conv2D(32, (3, 3), activation='relu', padding='same'),
    MaxPooling2D((2, 2)),
    Conv2D(64, (3, 3), activation='relu', padding='same'),
    MaxPooling2D((2, 2)),
    Conv2D(128, (3, 3), activation='relu', padding='same'),
    MaxPooling2D((2, 2)),
    Flatten(),
    Dense(128, activation='relu'),
    Dropout(0.5),
    Dense(1, activation='sigmoid')
])

# Agendador de Taxa de Aprendizado (ExponentialDecay)
initial_learning_rate = 0.001
num_train_images = len(list(train_ds.unbatch().as_numpy_iterator()))
steps_per_epoch = int(np.ceil(num_train_images / BATCH_SIZE))
decay_steps = steps_per_epoch * 10

lr_schedule = tf.keras.optimizers.schedules.ExponentialDecay(
    initial_learning_rate,
    decay_steps=decay_steps,
    decay_rate=0.9,
    staircase=True)

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

model.summary()


# --- 4. TREINAMENTO DO MODELO ---
callbacks = [
    EarlyStopping(monitor='val_loss', patience=20, restore_best_weights=True, verbose=1)
]

history = model.fit(
    train_ds_mu,
    epochs=100,
    validation_data=val_ds_p,
    callbacks=callbacks
)


# --- 5. AVALIAÇÃO E VISUALIZAÇÃO DOS RESULTADOS ---
print("\n--- Avaliação Final no Conjunto de Teste ---")
```

```
print("\n--- Avaliação Final no Conjunto de Teste ---")
loss, acc = model.evaluate(test_ds_p, verbose=0)
print(f"\nAcurácia no teste: {acc:.4f}")
print(f"Erro (Loss) no teste: {loss:.4f}")

fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(18, 6))
ax1.plot(history.history['accuracy'], label='Acurácia de Treino')
ax1.plot(history.history['val_accuracy'], label='Acurácia de Validação')
ax1.set_title('Evolução da Acurácia por Época')
ax1.set_xlabel('Épocas')
ax1.set_ylabel('Acurácia')
ax1.legend()
ax1.grid(True)

ax2.plot(history.history['loss'], label='Perda de Treino')
ax2.plot(history.history['val_loss'], label='Perda de Validação')
ax2.set_title('Evolução da Perda (Erro) por Época')
ax2.set_xlabel('Épocas')
ax2.set_ylabel('Erro (Loss)')
ax2.legend()
ax2.grid(True)
plt.show()

y_true = np.concatenate([y for x, y in test_ds_p], axis=0)
y_true = y_true.flatten().astype("int32")
y_pred_prob = model.predict(test_ds_p)
y_pred = (y_pred_prob > 0.5).astype("int32").flatten()
class_names = train_ds.class_names

print("\nRelatório de Classificação:")
print(classification_report(y_true, y_pred, target_names=class_names))
print("\nMatriz de Confusão:")
cm = confusion_matrix(y_true, y_pred)
plt.figure(figsize=(8, 6))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=class_names, yticklabels=class_names)
plt.xlabel('Rótulo Predito')
plt.ylabel('Rótulo Real')
plt.title('Matriz de Confusão')
plt.show()
```

```
--- Iniciando Preparação dos Dados ---
--- Divisão de Dados Concluída ---
--- Construindo pipeline tf.data com Mixup ---
Found 122 files belonging to 2 classes.
Found 32 files belonging to 2 classes.
Found 38 files belonging to 2 classes.
Model: "sequential_4"
```

| Layer (type) | Output Shape | Param # |
|---|---|---|
| conv2d_13 (Conv2D) | (None, 150, 150, 32) | 896 |
| max_pooling2d_13 (MaxPooling2D) | (None, 75, 75, 32) | 0 |
| conv2d_14 (Conv2D) | (None, 75, 75, 64) | 18,496 |
| max_pooling2d_14 (MaxPooling2D) | (None, 37, 37, 64) | 0 |
| conv2d_15 (Conv2D) | (None, 37, 37, 128) | 73,856 |
| max_pooling2d_15 (MaxPooling2D) | (None, 18, 18, 128) | 0 |
| flatten_4 (Flatten) | (None, 41472) | 0 |
| dense_8 (Dense) | (None, 128) | 5,308,544 |
| dropout_13 (Dropout) | (None, 128) | 0 |
| dense_9 (Dense) | (None, 1) | 129 |

```
 Total params: 5,401,921 (20.61 MB)
 Trainable params: 5,401,921 (20.61 MB)
 Non-trainable params: 0 (0.00 B)
Epoch 1/100
4/4 ━━━━━━━━━━━━━━━━━━━━ 12s 2s/step - accuracy: 0.4674 - loss: 1.0423 - val_accuracy: 0.3750 - val_loss: 0.8824
Epoch 2/100
4/4 ━━━━━━━━━━━━━━━━━━━━ 8s 2s/step - accuracy: 0.1869 - loss: 0.7551 - val_accuracy: 0.3750 - val_loss: 0.6937
Epoch 3/100
4/4 ━━━━━━━━━━━━━━━━━━━━ 8s 2s/step - accuracy: 0.2680 - loss: 0.6844 - val_accuracy: 0.5312 - val_loss: 0.6811
Epoch 4/100
4/4 ━━━━━━━━━━━━━━━━━━━━ 9s 2s/step - accuracy: 0.3142 - loss: 0.6827 - val_accuracy: 0.6875 - val_loss: 0.6466
Epoch 5/100
4/4 ━━━━━━━━━━━━━━━━━━━━ 8s 2s/step - accuracy: 0.4632 - loss: 0.6299 - val_accuracy: 0.9688 - val_loss: 0.5705
Epoch 6/100
4/4 ━━━━━━━━━━━━━━━━━━━━ 11s 2s/step - accuracy: 0.2929 - loss: 0.6041 - val_accuracy: 1.0000 - val_loss: 0.4730
Epoch 7/100
4/4 ━━━━━━━━━━━━━━━━━━━━ 7s 2s/step - accuracy: 0.4227 - loss: 0.4905 - val_accuracy: 0.6250 - val_loss: 0.4808
Epoch 8/100
4/4 ━━━━━━━━━━━━━━━━━━━━ 9s 2s/step - accuracy: 0.5388 - loss: 0.5932 - val_accuracy: 0.5000 - val_loss: 0.6972
Epoch 9/100
4/4 ━━━━━━━━━━━━━━━━━━━━ 7s 2s/step - accuracy: 0.4049 - loss: 0.6092 - val_accuracy: 0.8125 - val_loss: 0.4587
Epoch 10/100
4/4 ━━━━━━━━━━━━━━━━━━━━ 9s 2s/step - accuracy: 0.5314 - loss: 0.4455 - val_accuracy: 0.9062 - val_loss: 0.3537
Epoch 11/100
4/4 ━━━━━━━━━━━━━━━━━━━━ 6s 2s/step - accuracy: 0.5097 - loss: 0.4315 - val_accuracy: 0.8750 - val_loss: 0.3036
Epoch 12/100
4/4 ━━━━━━━━━━━━━━━━━━━━ 10s 2s/step - accuracy: 0.4753 - loss: 0.5198 - val_accuracy: 0.9688 - val_loss: 0.2698
Epoch 13/100
4/4 ━━━━━━━━━━━━━━━━━━━━ 12s 2s/step - accuracy: 0.4507 - loss: 0.3549 - val_accuracy: 0.9375 - val_loss: 0.2643
Epoch 14/100
4/4 ━━━━━━━━━━━━━━━━━━━━ 9s 2s/step - accuracy: 0.4744 - loss: 0.3892 - val_accuracy: 1.0000 - val_loss: 0.2370
Epoch 15/100
4/4 ━━━━━━━━━━━━━━━━━━━━ 8s 2s/step - accuracy: 0.4784 - loss: 0.3944 - val_accuracy: 1.0000 - val_loss: 0.1997
Epoch 16/100
4/4 ━━━━━━━━━━━━━━━━━━━━ 8s 2s/step - accuracy: 0.4827 - loss: 0.2533 - val_accuracy: 0.9375 - val_loss: 0.2128
Epoch 17/100
4/4 ━━━━━━━━━━━━━━━━━━━━ 7s 2s/step - accuracy: 0.5338 - loss: 0.2839 - val_accuracy: 1.0000 - val_loss: 0.1067
Epoch 18/100
4/4 ━━━━━━━━━━━━━━━━━━━━ 12s 2s/step - accuracy: 0.5161 - loss: 0.2662 - val_accuracy: 0.9062 - val_loss: 0.2069
Epoch 19/100
4/4 ━━━━━━━━━━━━━━━━━━━━ 7s 2s/step - accuracy: 0.5761 - loss: 0.1993 - val_accuracy: 1.0000 - val_loss: 0.0960
Epoch 20/100
4/4 ━━━━━━━━━━━━━━━━━━━━ 10s 2s/step - accuracy: 0.5676 - loss: 0.2148 - val_accuracy: 1.0000 - val_loss: 0.0859
Epoch 21/100
4/4 ━━━━━━━━━━━━━━━━━━━━ 9s 2s/step - accuracy: 0.6463 - loss: 0.1795 - val_accuracy: 1.0000 - val_loss: 0.0854
Epoch 22/100
4/4 ━━━━━━━━━━━━━━━━━━━━ 8s 2s/step - accuracy: 0.4627 - loss: 0.2582 - val_accuracy: 1.0000 - val_loss: 0.0752
Epoch 23/100
4/4 ━━━━━━━━━━━━━━━━━━━━ 10s 2s/step - accuracy: 0.5484 - loss: 0.2515 - val_accuracy: 0.9062 - val_loss: 0.1743
Epoch 24/100
4/4 ━━━━━━━━━━━━━━━━━━━━ 12s 2s/step - accuracy: 0.5589 - loss: 0.2834 - val_accuracy: 1.0000 - val_loss: 0.0818
Epoch 25/100
4/4 ━━━━━━━━━━━━━━━━━━━━ 8s 2s/step - accuracy: 0.4661 - loss: 0.2319 - val_accuracy: 1.0000 - val_loss: 0.0913
Epoch 26/100
4/4 ━━━━━━━━━━━━━━━━━━━━ 9s 2s/step - accuracy: 0.5807 - loss: 0.2540 - val_accuracy: 0.9062 - val_loss: 0.1623
Epoch 27/100
4/4 ━━━━━━━━━━━━━━━━━━━━ 8s 2s/step - accuracy: 0.5195 - loss: 0.2349 - val_accuracy: 1.0000 - val_loss: 0.0808
Epoch 28/100
4/4 ━━━━━━━━━━━━━━━━━━━━ 8s 2s/step - accuracy: 0.5377 - loss: 0.2078 - val_accuracy: 1.0000 - val_loss: 0.0747
Epoch 29/100
```