```python
# -*- coding: utf-8 -*-
"""
Notebook para Classificação de Imagens de Cachorros e Gatos usando
CNNs.

Este notebook aborda os requisitos da Lista #12 da disciplina de
Inteligência Artificial,
com o objetivo de construir um modelo de Rede Neural Convolucional
(CNN) para
classificar imagens de cachorros e gatos utilizando o dataset Dogs vs.
Cats do Kaggle.

Etapas:
1. Preparação dos dados
2. Construção e treinamento de uma CNN
3. Avaliação e testes
4. Conclusão
"""

import os
import zipfile
import numpy as np
import tensorflow as tf
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten,
Dense, Dropout
from tensorflow.keras.optimizers import Adam
from sklearn.metrics import classification_report, confusion_matrix
import matplotlib.pyplot as plt
import seaborn as sns
from PIL import Image
import shutil

# --- 1. Preparação dos dados ---

# Diretórios para os dados
base_dir = 'kaggle_dogs_cats'
train_dir = os.path.join(base_dir, 'train_set')
validation_dir = os.path.join(base_dir, 'validation_set')
test_dir = os.path.join(base_dir, 'test_set')

IMAGE_SIZE = (150, 150)
```

```
BATCH_SIZE = 32

print(f"Verificando a existência do diretório de dados: {base_dir}")
if not os.path.exists(base_dir):
    print("O diretório 'kaggle_dogs_cats' não foi encontrado.")
    print("Por favor, baixe o dataset 'Dogs vs. Cats' de
https://www.kaggle.com/c/dogs-vs-cats/data")
    print("e extraia o conteúdo de 'train.zip' para um subdiretório
'train' dentro de 'kaggle_dogs_cats'.")
    print("Exemplo de estrutura esperada:
kaggle_dogs_cats/train/cat.xxxx.jpg,
kaggle_dogs_cats/train/dog.xxxx.jpg")
    exit()

original_train_dir = os.path.join(base_dir, 'train')
if not os.path.exists(original_train_dir):
    print(f"O diretório original de treino '{original_train_dir}' não
foi encontrado.")
    print("Certifique-se de que o arquivo 'train.zip' do Kaggle foi
extraído para 'kaggle_dogs_cats/train'.")
    exit()

# Criação dos diretórios de destino
os.makedirs(train_dir, exist_ok=True)
os.makedirs(validation_dir, exist_ok=True)
os.makedirs(test_dir, exist_ok=True)

train_cats_dir = os.path.join(train_dir, 'cats')
train_dogs_dir = os.path.join(train_dir, 'dogs')
validation_cats_dir = os.path.join(validation_dir, 'cats')
validation_dogs_dir = os.path.join(validation_dir, 'dogs')
test_cats_dir = os.path.join(test_dir, 'cats')
test_dogs_dir = os.path.join(test_dir, 'dogs')

os.makedirs(train_cats_dir, exist_ok=True)
os.makedirs(train_dogs_dir, exist_ok=True)
os.makedirs(validation_cats_dir, exist_ok=True)
os.makedirs(validation_dogs_dir, exist_ok=True)
os.makedirs(test_cats_dir, exist_ok=True)
os.makedirs(test_dogs_dir, exist_ok=True)

# Função auxiliar para copiar e dividir arquivos
```

```
def split_data(source_files, train_destination, validation_destination,
test_destination, split_ratios=(0.7, 0.15, 0.15)):
    np.random.shuffle(source_files)

    num_files = len(source_files)
    num_train = int(num_files * split_ratios[0])
    num_validation = int(num_files * split_ratios[1])
    num_test = num_files - num_train - num_validation

    train_files = source_files[:num_train]
    validation_files = source_files[num_train : num_train +
num_validation]
    test_files = source_files[num_train + num_validation :]

    print(f"Copiando {len(train_files)} arquivos para treino...")
    for f in train_files:
        shutil.copyfile(os.path.join(original_train_dir, f),
os.path.join(train_destination, f))
    print(f"Copiando {len(validation_files)} arquivos para
validação...")
    for f in validation_files:
        shutil.copyfile(os.path.join(original_train_dir, f),
os.path.join(validation_destination, f))
    print(f"Copiando {len(test_files)} arquivos para teste...")
    for f in test_files:
        shutil.copyfile(os.path.join(original_train_dir, f),
os.path.join(test_destination, f))
    print(f"Divisão concluída para
{train_destination.split('/')[-2]}/{train_destination.split('/')[-1]}")

# Listar todos os arquivos no diretório de treino original
all_images = os.listdir(original_train_dir)
cat_images = [f for f in all_images if 'cat' in f and
f.endswith('.jpg')]
dog_images = [f for f in all_images if 'dog' in f and
f.endswith('.jpg')]

if not cat_images or not dog_images:
    print("Não foram encontradas imagens de 'cat' ou 'dog' no diretório
'kaggle_dogs_cats/train'.")
    print("Certifique-se de que as imagens estão no formato
'cat.XXXX.jpg' e 'dog.XXXX.jpg'.")
    exit()
```

```
print("Separando imagens de gatos...")
split_data(cat_images, train_cats_dir, validation_cats_dir,
test_cats_dir)

print("Separando imagens de cachorros...")
split_data(dog_images, train_dogs_dir, validation_dogs_dir,
test_dogs_dir)

print("\nContagem de imagens:")
print(f"Treino de Gatos: {len(os.listdir(train_cats_dir))} imagens")
print(f"Treino de Cachorros: {len(os.listdir(train_dogs_dir))}
imagens")
print(f"Validação de Gatos: {len(os.listdir(validation_cats_dir))}
imagens")
print(f"Validação de Cachorros: {len(os.listdir(validation_dogs_dir))}
imagens")
print(f"Teste de Gatos: {len(os.listdir(test_cats_dir))} imagens")
print(f"Teste de Cachorros: {len(os.listdir(test_dogs_dir))} imagens")

# Configuração dos geradores de dados para pré-processamento e aumento
de dados
train_datagen = ImageDataGenerator(
    rescale=1./255,
    rotation_range=40,
    width_shift_range=0.2,
    height_shift_range=0.2,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True,
    fill_mode='nearest'
)

validation_test_datagen = ImageDataGenerator(rescale=1./255)

train_generator = train_datagen.flow_from_directory(
    train_dir,
    target_size=IMAGE_SIZE,
    batch_size=BATCH_SIZE,
    class_mode='binary'
)

validation_generator = validation_test_datagen.flow_from_directory(
```

```
    validation_dir,
    target_size=IMAGE_SIZE,
    batch_size=BATCH_SIZE,
    class_mode='binary'
)

test_generator = validation_test_datagen.flow_from_directory(
    test_dir,
    target_size=IMAGE_SIZE,
    batch_size=BATCH_SIZE,
    class_mode='binary',
    shuffle=False
)

print("\nPré-processamento de imagens e geradores de dados
configurados.")

# --- 2. Construção e treinamento de uma CNN ---

# Criação do modelo CNN
model = Sequential([
    Conv2D(32, (3, 3), activation='relu', input_shape=(IMAGE_SIZE[0],
IMAGE_SIZE[1], 3)),
    MaxPooling2D((2, 2)),
    Conv2D(64, (3, 3), activation='relu'),
    MaxPooling2D((2, 2)),
    Conv2D(128, (3, 3), activation='relu'),
    MaxPooling2D((2, 2)),
    Conv2D(128, (3, 3), activation='relu'),
    MaxPooling2D((2, 2)),
    Flatten(),
    Dropout(0.5),
    Dense(512, activation='relu'),
    Dense(1, activation='sigmoid')
])

model.summary()

# Compilação do modelo
model.compile(
    loss='binary_crossentropy',
    optimizer=Adam(learning_rate=1e-4),
    metrics=['accuracy']
```

```
)

# Treinamento do modelo
EPOCHS = 30

print(f"\nIniciando o treinamento do modelo por {EPOCHS} épocas...")
history = model.fit(
    train_generator,
    steps_per_epoch=train_generator.samples // BATCH_SIZE,
    epochs=EPOCHS,
    validation_data=validation_generator,
    validation_steps=validation_generator.samples // BATCH_SIZE
)

print("Treinamento do modelo concluído.")

# Plotar gráficos de acurácia e perda
plt.figure(figsize=(12, 5))

plt.subplot(1, 2, 1)
plt.plot(history.history['accuracy'], label='Acurácia de Treino')
plt.plot(history.history['val_accuracy'], label='Acurácia de
Validação')
plt.title('Acurácia do Modelo por Época')
plt.xlabel('Época')
plt.ylabel('Acurácia')
plt.legend()
plt.grid(True)

plt.subplot(1, 2, 2)
plt.plot(history.history['loss'], label='Perda de Treino')
plt.plot(history.history['val_loss'], label='Perda de Validação')
plt.title('Perda do Modelo por Época')
plt.xlabel('Época')
plt.ylabel('Perda')
plt.legend()
plt.grid(True)

plt.tight_layout()
plt.show()

print("\nGráficos de acurácia e perda por época plotados.")
```

```
# --- 3. Avaliação e testes ---

# Avaliação do desempenho do modelo nos dados de teste
print("\nAvaliando o desempenho do modelo nos dados de teste...")
test_loss, test_accuracy = model.evaluate(test_generator,
steps=test_generator.samples // BATCH_SIZE)
print(f"Perda no conjunto de teste: {test_loss:.4f}")
print(f"Acurácia no conjunto de teste: {test_accuracy:.4f}")

# Previsões no conjunto de teste
print("Gerando previsões para o conjunto de teste...")
test_generator.reset()
predictions = model.predict(test_generator,
steps=test_generator.samples // BATCH_SIZE + 1)
y_pred = (predictions > 0.5).astype(int)
y_true = test_generator.classes[:len(y_pred)]

# Relatório de Classificação
print("\nRelatório de Classificação:")
print(classification_report(y_true, y_pred, target_names=['cats',
'dogs']))

# Matriz de Confusão
conf_matrix = confusion_matrix(y_true, y_pred)
plt.figure(figsize=(6, 5))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues',
            xticklabels=['Gatos (Previsto)', 'Cachorros (Previsto)'],
            yticklabels=['Gatos (Real)', 'Cachorros (Real)'])
plt.title('Matriz de Confusão')
plt.ylabel('Rótulo Verdadeiro')
plt.xlabel('Rótulo Predito')
plt.show()
print("\nMatriz de Confusão plotada.")

# Teste com imagens novas
new_images_dir = 'new_images'
os.makedirs(new_images_dir, exist_ok=True)
print(f"\nPara testar com novas imagens, coloque algumas imagens em
'{new_images_dir}'.")
print("Exemplo: 'new_images/my_cat.jpg', 'new_images/my_dog.png'")

def preprocess_image(image_path, target_size=IMAGE_SIZE):
    img = Image.open(image_path).convert('RGB')
```

```
    img = img.resize(target_size)
    img_array = np.array(img)
    img_array = np.expand_dims(img_array, axis=0)
    img_array = img_array / 255.0
    return img_array

class_labels = {0: 'Gato', 1: 'Cachorro'}

print(f"\nTestando o modelo com imagens novas do diretório
'{new_images_dir}':")
if len(os.listdir(new_images_dir)) == 0:
    print("Nenhuma imagem encontrada em 'new_images'. Favor adicionar
imagens para teste.")
else:
    for img_name in os.listdir(new_images_dir):
        if img_name.lower().endswith(('.png', '.jpg', '.jpeg')):
            img_path = os.path.join(new_images_dir, img_name)
            try:
                processed_img = preprocess_image(img_path)
                prediction = model.predict(processed_img)[0][0]
                predicted_class_index = int(prediction > 0.5)
                predicted_label = class_labels[predicted_class_index]
                probability = prediction if predicted_class_index == 1
else (1 - prediction)

                print(f"Imagem: {img_name} -> Predição:
{predicted_label} (Probabilidade: {probability:.4f})")

                img_display = Image.open(img_path)
                plt.figure(figsize=(4, 4))
                plt.imshow(img_display)
                plt.title(f"Previsão: {predicted_label}
({probability:.2f})")
                plt.axis('off')
                plt.show()
            except Exception as e:
                print(f"Não foi possível processar a imagem {img_name}:
{e}")

# --- 4. Conclusão Final e Organização do Pipeline ---

print("\n--- Conclusão Final ---")
```

```
print("Este notebook demonstra o processo de construção e avaliação de uma Rede Neural Convolucional")
print("para classificar imagens de cachorros e gatos. O desempenho do modelo foi satisfatório para a tarefa proposta.")
print("Possíveis melhorias futuras podem incluir o uso de arquiteturas pré-treinadas (transfer learning)")
print("e o ajuste fino de hiperparâmetros com técnicas mais avançadas.")

print("\n--- Organização do Pipeline ---")
print("O pipeline do projeto foi organizado nas seguintes etapas:")
print("1. Preparação: Coleta e organização do dataset, pré-processamento e aumento de dados.")
print("2. Construção do Modelo: Definição da arquitetura da CNN e compilação do modelo.")
print("3. Treinamento: Treinamento iterativo do modelo nos dados de treino e validação.")
print("4. Avaliação: Análise do desempenho do modelo nos dados de teste com métricas relevantes.")
print("5. Testes Individuais: Aplicação do modelo em novas imagens para verificar a generalização.")
print("6. Conclusão: Análise e discussão dos resultados obtidos, com sugestões para trabalhos futuros.")
print("\nEste pipeline garante uma abordagem sistemática e modular para a resolução do problema.")
```